Basic Computer Coding: Java Script

2nd Edition

BIBLIOTEX
Digital Library

BASIC COMPUTER CODING: JAVA SCRIPT

2nd Edition

BIBLIOTEX
Digital Library

www.bibliotex.com

BASIC COMPUTER CODING: JAVA SCRIPT

2ND EDITION

BIBLIOTEX
Digital Library

www.bibliotex.com

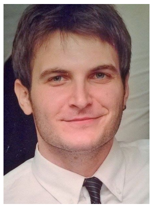
email: info@bibliotex.com

e-book Edition 2022

ISBN: 978-1-98467-603-0 (e-book)

In Collaboration with 3G E-Learning LLC. Originally Published in printed book format by 3G E-Learning LLC with ISBN 978-1-98465-896-8

EDITORIAL BOARD



Aleksandar Mratinković was born on May 5, 1988 in Arandjelovac, Serbia. He has graduated on Economic high school (2007), The College of Tourism in Belgrade (2013), and also has a master degree of Psychology (Faculty of Philosophy, University of Novi Sad). He has been engaged in different fields of psychology (Developmental Psychology, Clinical Psychology, Educational Psychology and Industrial Psychology) and has published several scientific works.

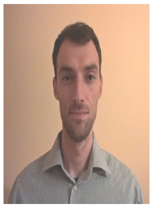
Dan Piestun (PhD) is currently a startup entrepreneur in Israel working on the interface of Agriculture and Biomedical Sciences and was formerly president-CEO of the National Institute of Agricultural Research (INIA) in Uruguay. Dan is a widely published scientist who has received many honours during his career including being a two-time recipient of the Amit Golda Meir Prize from the Hebrew University of Jerusalem, his areas of expertise includes stem cell molecular biology, plant and animal genetics and bioinformatics. Dan's passion for applied science and technological solutions did not stop him from pursuing a deep connection to the farmer, his family and nature. Among some of his interest and practices counts enjoying working as a beekeeper and onboard fishing.

Hazem Shawky Fouda has a PhD. in Agriculture Sciences, obtained his PhD. From the Faculty of Agriculture, Alexandria University in 2008, He is working in Cotton Arbitration & Testing General Organization (CATGO).

Felecia Killings is the Founder and CEO of LiyahAmore Publishing, a publishing company committed to providing technical and educational services and products to Christian Authors. She operates as the Senior Editor and Writer, the Senior Writing Coach, the Content Marketing Specialist, Editor-in-Chief to the company's quarterly magazine, the Executive and Host of an international virtual network, and the Executive Director of the company's online school for Authors. She is a former high-school English instructor and professional development professor. She possesses a Master of Arts degree in Education and a Bachelor's degree in English and African American studies.

Dr. Sandra El Hajj, Ph.D. in Health Sciences from Nova Southeastern University, Florida, USA is a health professional specialized in Preventive and Global Health. With her 12 years of education obtained from one of the most prominent universities in Beirut, in addition to two leading universities in the State of Florida (USA), Dr. Sandra made sure to incorporate interdisciplinary and multicultural approaches in her work. Her long years of studies helped her create her own miniature world of knowledge linking together the healthcare field with Medical Research, Statistics, Food Technology, Environmental & Occupational Health, Preventive Health and most noteworthy her precious last degree of Global Health. Till today, she is the first and only doctor specialized in Global Health in the Middle East area.

Fozia Parveen has a Dphil in Sustainable Water Engineering from the University of Oxford. Prior to this she has received MS in Environmental Sciences from National University of Science and Technology (NUST), Islamabad Pakistan and BS in Environmental Sciences from Fatima Jinnah Women University (FJWU), Rawalpindi.



Igor Krunic 2003-2007 in the School of Economics. After graduating in 2007, he went on to study at The College of Tourism, at the University of Belgrade where he got his bachelor degree in 2010. He was active as a third-year student representative in the student parliament.Then he went on the Faculty of science, at the University of Novi Sad where he successfully defended his master's thesis in 2013. The crown of his study was the work titled Opportunities for development of cultural tourism in Cacak". Later on, he became part of a multinational company where he got promoted to a deputy director of logistic. Nowadays he is a consultant and writer of academic subjects in the field of tourism.

Dr. Jovan Pehcevski obtained his PhD in Computer Science from RMIT University in Melbourne, Australia in 2007. His research interests include big data, business intelligence and predictive analytics, data and information science, information retrieval, XML, web services and service-oriented architectures, and relational and NoSQL database systems. He has published over 30 journal and conference papers and he also serves as a journal and conference reviewer. He is currently working as a Dean and Associate Professor at European University in Skopje, Macedonia.

Dr. Tanjina Nur finished her PhD in Civil and Environmental Engineering in 2014 from University of Technology Sydney (UTS). Now she is working as Post-Doctoral Researcher in the Centre for Technology in Water and Wastewater (CTWW) and published about eight International journal papers with 80 citations. Her research interest is wastewater treatment technology using adsorption process.

Stephen obtained his PhD from the University of North Carolina at Charlotte in 2013 where his graduate research focused on cancer immunology and the tumor microenvironment. He received postdoctoral training in regenerative and translational medicine, specifically gastrointestinal tissue engineering, at the Wake Forest Institute of Regenerative Medicine. Currently, Stephen is an instructor for anatomy and physiology and biology at Forsyth Technical Community College.

Michelle holds a Masters of Business Administration from the University of Phoenix, with a concentration in Human Resources Management. She is a professional author and has had numerous articles published in the Henry County Times and has written and revised several employee handbooks for various YMCA organizations throughout the United States.

HOW TO USE THE BOOK

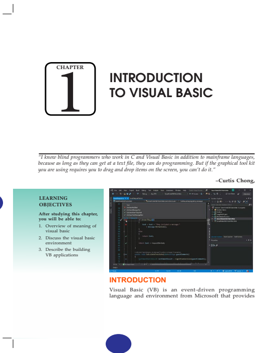
This book has been divided into many chapters. Chapter gives the motivation for this book and the use of templates. The text is presented in the simplest language. Each paragraph has been arranged under a suitable heading for easy retention of concept. Keywords are the words that academics use to reveal the internal structure of an author's reasoning. Review questions at the end of each chapter ask students to review or explain the concepts. References provides the reader an additional source through which he/she can obtain more information regarding the topic.

LEARNING OBJECTIVES

See what you are going to cover and what you should already know at the start of each chapter

ABOUT THIS CHAPTER

An introduction is a beginning of section which states the purpose and goals of the topics which are discussed in the chapter. It also starts the topics in brief.



CHAPTER 1

INTRODUCTION TO VISUAL BASIC

"I know blind programmers who work in C and Visual Basic in addition to mainframe languages, because as long as they can get at a text file, they can do programming. But if the graphical tool kit you are using requires you to drag and drop items on the screen, you can't do it."

–Curtis Chong,

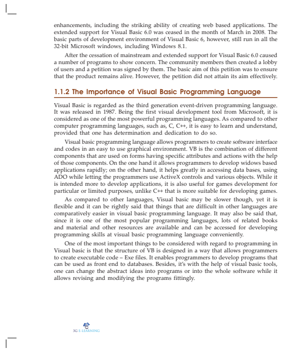
LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Overview of meaning of visual basic
2. Discuss the visual basic environment
3. Describe the building VB applications

INTRODUCTION

Visual Basic (VB) is an event-driven programming language and environment from Microsoft that provides



enhancements, including the striking ability of creating web based applications. The extended support for Visual Basic 6.0 was ceased in the month of March in 2008. The basic parts of development environment of Visual Basic 6, however, still run in all the 32-bit Microsoft windows, including Windows 8.1.

After the cessation of mainstream and extended support for Visual Basic 6.0 caused a number of programs to show concern. The community members then created a lobby of users and a petition was signed by them. The basic aim of this petition was to ensure that the product remains alive. However, the petition did not attain its aim effectively.

1.1.2 The Importance of Visual Basic Programming Language

Visual Basic is regarded as the third generation event-driven programming language. It was released in 1987. Being the first visual development tool from Microsoft, it is considered as one of the most powerful programming languages. As compared to other computer programming languages, such as, C, C++, it is easy to learn and understand, provided that one has determination and dedication to do so.

Visual basic programming language allows programmers to create software interface and codes in an easy to use graphical environment. VB is the combination of different components that are used on forms having specific attributes and actions with the help of those components. On the one hand it allows programmers to develop widows based applications rapidly; on the other hand, it helps greatly in accessing data bases, using ADO while letting the programmers use ActiveX controls and various objects. While it is intended more to develop applications, it is also useful for games development for particular or limited purposes, unlike C++ that is more suitable for developing games.

As compared to other languages, Visual basic may be slower though, yet it is flexible and it can be rightly said that things that are difficult in other languages are comparatively easier in visual basic programming language. It may also be said that, since it is one of the most popular programming languages, lots of related books and material and other resources are available and can be accessed for developing programming skills at visual basic programming language conveniently.

One of the most important things to be considered with regard to programming in Visual basic is that the structure of VB is designed in a way that allows programmers to create executable code – Exe files. It enables programmers to develop programs that can be used as front end to databases. Besides, it's with the help of visual basic tools, one can change the abstract ideas into programs or into the whole software while it allows revising and modifying the programs fittingly.

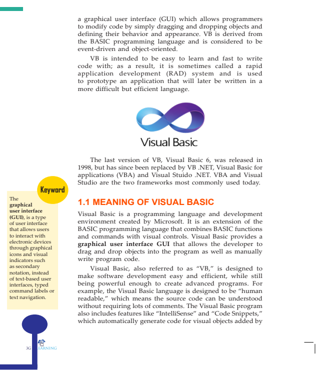
3G E-LEARNING

REMEMBER

This revitalizes a must read information of the topic.

KEYWORDS

This section contains some important definitions that are discussed in the chapter. A keyword is an index entry that identifies a specific record or document. It also gives the extra information to the reader and an easy way to remember the word definition.



a graphical user interface (GUI) which allows programmers to modify code by simply dragging and dropping objects and defining their behavior and appearance. VB is derived from the BASIC programming language and is considered to be event-driven and object-oriented.

VB is intended to be easy to learn and fast to write code with; as a result, it is sometimes called a rapid application development (RAD) system and is used to prototype an application that will later be written in a more difficult but efficient language.

Visual Basic

The last version of VB, Visual Basic 6, was released in 1998, but has since been replaced by VB .NET, Visual Basic for applications (VBA) and Visual Stuido .NET. VBA and Visual Studio are the two frameworks most commonly used today.

Keyword

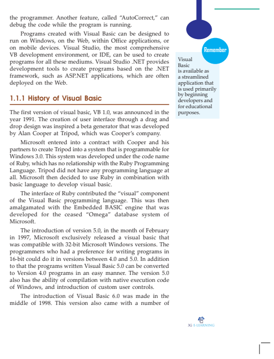
The **graphical user interface (GUI)**, is a type of user interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notation, instead of text-based user interfaces, typed command labels or text navigation.

1.1 MEANING OF VISUAL BASIC

Visual Basic is a programming language and development environment created by Microsoft. It is an extension of the BASIC programming language that combines BASIC functions and commands with visual controls. Visual Basic provides a **graphical user interface GUI** that allows the developer to drag and drop objects into the program as well as manually write program code.

Visual Basic, also referred to as "VB," is designed to make software development easy and efficient, while still being powerful enough to create advanced programs. For example, the Visual Basic language is designed to be "human readable," which means the source code can be understood without requiring lots of comments. The Visual Basic program also includes features like "IntelliSense" and "Code Snippets," which automatically generate code for visual objects added by

3G E-LEARNING



the programmer. Another feature, called "AutoCorrect," can debug the code while the program is running.

Programs created with Visual Basic can be designed to run on Windows, on the Web, within Office applications, or on mobile devices. Visual Studio, the most comprehensive VB development environment, or IDE, can be used to create programs for all these mediums. Visual Studio .NET provides development tools to create programs based on the .NET framework, such as ASP.NET applications, which are often deployed on the Web.

Remember

Visual Basic is available as a streamlined application that is used primarily by beginning developers and for educational purposes.

1.1.1 History of Visual Basic

The first version of visual basic, VB 1.0, was announced in the year 1991. The creation of user interface through a drag and drop design was inspired a beta generator that was developed by Alan Cooper at Tripod, which was Cooper's company.

Microsoft entered into a contract with Cooper and his partners to create Tripod into a system that is programmable for Windows 3.0. This system was developed under the code name of Ruby, which has no relationship with the Ruby Programming Language. Tripod did not have any programming language at all. Microsoft then decided to use Ruby in combination with basic language to develop visual basic.

The interface of Ruby contributed the "visual" component of the Visual Basic programming language. This was then amalgamated with the Embedded BASIC engine that was developed for the ceased "Omega" database system of Microsoft.

The introduction of version 5.0, in the month of February in 1997, Microsoft exclusively released a visual basic that was compatible with 32-bit Microsoft Windows versions. The programmers who had a preference for writing programs in 16-bit could do it in versions between 4.0 and 5.0. In addition to that the programs written Visual Basic 5.0 can be converted to Version 4.0 programs in an easy manner. The version 5.0 also has the ability of compilation with native execution code of Windows, and introduction of custom user controls.

The introduction of Visual Basic 6.0 was made in the middle of 1998. This version also came with a number of

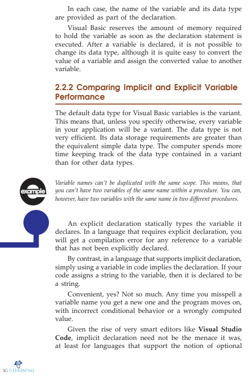
3G E-LEARNING

DID YOU KNOW?

This section equip readers the interesting facts and figures of the topic.

EXAMPLE

The book cabinets' examples to illustrate specific ideas in each chapter.



In each case, the name of the variable and its data type are provided as part of the declaration.

Visual Basic reserves the amount of memory required to hold the variable as soon as the declaration statement is executed. After a variable is declared, it is not possible to change its data type, although it is quite easy to convert the value of a variable and assign the converted value to another variable.

2.2.2 Comparing Implicit and Explicit Variable Performance

The default data type for Visual Basic variables is the variant. This means that, unless you specify otherwise, every variable in your application will be a variant. The data type is not very efficient. Its data storage requirements are greater than the equivalent simple data type. The computer spends more time keeping track of the data type contained in a variant than for other data types.

Variable names can't be duplicated with the same scope. This means, that you can't have two variables of the same name within a procedure. You can, however, have two variables with the same name in two different procedures.

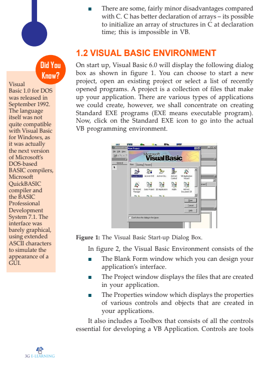
An explicit declaration statically types the variable it declares. In a language that requires explicit declaration, you will get a compilation error for any reference to a variable that has not been explicitly declared.

By contrast, in a language that supports implicit declaration, simply using a variable in code implies the declaration. If your code assigns a string to the variable, then it is declared to be a string.

Convenient, yes? Not so much. Any time you misspell a variable name you get a new one and the program moves on, with incorrect conditional behavior or a wrongly computed value.

Given the rise of very smart editors like **Visual Studio Code**, implicit declaration need not be the menace it was, at least for languages that support the notion of optional

3G E-LEARNING



- There are some, fairly minor disadvantages compared with C. C has better declaration of arrays – its possible to initialize an array of structures in C at declaration time; this is impossible in VB.

1.2 VISUAL BASIC ENVIRONMENT

Did You Know?

Visual Basic 1.0 for DOS was released in September 1992. The language itself was not quite compatible with Visual Basic for Windows, as it was actually the next version of Microsoft's DOS-based BASIC compilers, Microsoft QuickBASIC compiler and the BASIC Professional Development System 7.1. The interface was barely graphical, using extended ASCII characters to simulate the appearance of a GUI.

On start up, Visual Basic 6.0 will display the following dialog box as shown in figure 1. You can choose to start a new project, open an existing project or select a list of recently opened programs. A project is a collection of files that make up your application. There are various types of applications we could create, however, we shall concentrate on creating Standard EXE programs (EXE means executable program). Now, click on the Standard EXE icon to go into the actual VB programming environment.

Figure 1: The Visual Basic Start-up Dialog Box.

In figure 2, the Visual Basic Environment consists of the

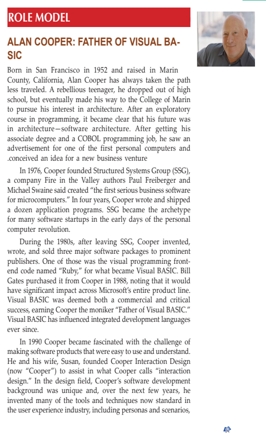
- The Blank Form window which you can design your application's interface.
- The Project window displays the files that are created in your application.
- The Properties window which displays the properties of various controls and objects that are created in your applications.

It also includes a Toolbox that consists of all the controls essential for developing a VB Application. Controls are tools

3G E-LEARNING

ROLE MODEL

A biography of someone who has/had acquired remarkable success in their respective field as Role Models are important because they give us the ability to imagine our future selves.



ROLE MODEL

ALAN COOPER: FATHER OF VISUAL BASIC

Born in San Francisco in 1952 and raised in Marin County, California, Alan Cooper has always taken the path less traveled. A rebellious teenager, he dropped out of high school, but eventually made his way to the College of Marin to pursue his interest in architecture. After an exploratory course in programming, it became clear that his future was in architecture—software architecture. After getting his associate degree and a COBOL programming job, he saw an advertisement for one of the first personal computers and .conceived an idea for a new business venture

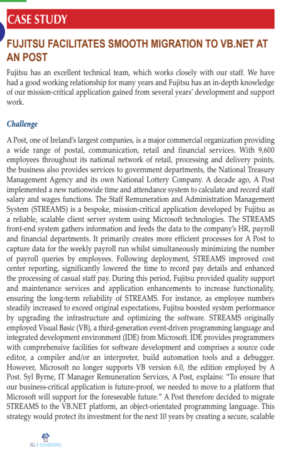
In 1976, Cooper founded Structured Systems Group (SSG), a company Fire in the Valley authors Paul Freiberger and Michael Swaine said created "the first serious business software for microcomputers." In four years, Cooper wrote and shipped a dozen application programs. SSG became the archetype for many software startups in the early days of the personal computer revolution.

During the 1980s, after leaving SSG, Cooper invented, wrote, and sold three major software packages to prominent publishers. One of those was the visual programming front-end code named "Ruby," for what became Visual BASIC. Bill Gates purchased it from Cooper in 1988, noting that it would have significant impact across Microsoft's entire product line. Visual BASIC was deemed both a commercial and critical success, earning Cooper the moniker "Father of Visual BASIC." Visual BASIC has influenced integrated development languages ever since.

In 1990 Cooper became fascinated with the challenge of making software products that were easy to use and understand. He and his wife, Susan, founded Cooper Interaction Design (now "Cooper") to assist in what Cooper calls "interaction design." In the design field, Cooper's software development background was unique and, over the next few years, he invented many of the tools and techniques now standard in the user experience industry, including personas and scenarios,

CASE STUDY

This reveals what students need to create and provide an opportunity for the development of key skills such as communication, group working and problem solving.



CASE STUDY

FUJITSU FACILITATES SMOOTH MIGRATION TO VB.NET AT AN POST

Fujitsu has an excellent technical team, which works closely with our staff. We have had a good working relationship for many years and Fujitsu has an in-depth knowledge of our mission-critical application gained from several years' development and support work.

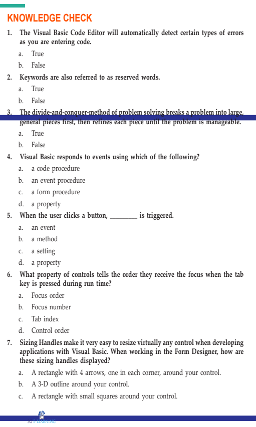
Challenge

A Post, one of Ireland's largest companies, is a major commercial organization providing a wide range of postal, communication, retail and financial services. With 9,600 employees throughout its national network of retail, processing and delivery points, the business also provides services to government departments, the National Treasury Management Agency and its own National Lottery Company. A decade ago, A Post implemented a new nationwide time and attendance system to calculate and record staff salary and wages functions. The Staff Remuneration and Administration Management System (STREAMS) is a bespoke, mission-critical application developed by Fujitsu as a reliable, scalable client server system using Microsoft technologies. The STREAMS front-end system gathers information and feeds the data to the company's HR, payroll and financial departments. It primarily creates more efficient processes for A Post to capture data for the weekly payroll run whilst simultaneously minimizing the number of payroll queries by employees. Following deployment, STREAMS improved cost center reporting, significantly lowered the time to record pay details and enhanced the processing of casual staff pay. During this period, Fujitsu provided quality support and maintenance services and application enhancements to increase functionality, ensuring the long-term reliability of STREAMS. For instance, as employee numbers steadily increased to exceed original expectations, Fujitsu boosted system performance by upgrading the infrastructure and optimizing the software. STREAMS originally employed Visual Basic (VB), a third-generation event-driven programming language and integrated development environment (IDE) from Microsoft. IDE provides programmers with comprehensive facilities for software development and comprises a source code editor, a compiler and/or an interpreter, build automation tools and a debugger. However, Microsoft no longer supports VB version 6.0, the edition employed by A Post. Syl Byrne, IT Manager Remuneration Services, A Post, explains: "To ensure that our business-critical application is future-proof, we needed to move to a platform that Microsoft will support for the foreseeable future." A Post therefore decided to migrate STREAMS to the VB.NET platform, an object-orientated programming language. This strategy would protect its investment for the next 10 years by creating a secure, scalable

3G E-LEARNING

KNOWLEDGE CHECK

This is given to the students for progress check at the end of each chapter.



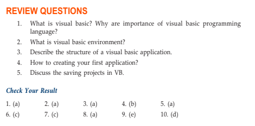
KNOWLEDGE CHECK

1. **The Visual Basic Code Editor will automatically detect certain types of errors as you are entering code.**
   a. True
   b. False
2. **Keywords are also referred to as reserved words.**
   a. True
   b. False
3. **The divide-and-conquer-method of problem solving breaks a problem into large, general pieces first, then refines each piece until the problem is manageable.**
   a. True
   b. False
4. **Visual Basic responds to events using which of the following?**
   a. a code procedure
   b. an event procedure
   c. a form procedure
   d. a property
5. **When the user clicks a button, ________ is triggered.**
   a. an event
   b. a method
   c. a setting
   d. a property
6. **What property of controls tells the order they receive the focus when the tab key is pressed during run time?**
   a. Focus order
   b. Focus number
   c. Tab index
   d. Control order
7. **Sizing Handles make it very easy to resize virtually any control when developing applications with Visual Basic. When working in the Form Designer, how are these sizing handles displayed?**
   a. A rectangle with 4 arrows, one in each corner, around your control.
   b. A 3-D outline around your control.
   c. A rectangle with small squares around your control.

3G E-LEARNING

REVIEW QUESTIONS

This section is to analyze the knowledge and ability of the reader.



REVIEW QUESTIONS

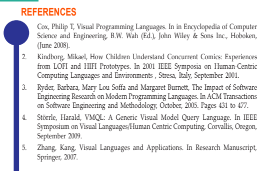
1. What is visual basic? Why are importance of visual basic programming language?
2. What is visual basic environment?
3. Describe the structure of a visual basic application.
4. How to creating your first application?
5. Discuss the saving projects in VB.

Check Your Result

1. (a)	2. (a)	3. (a)	4. (b)	5. (a)
6. (c)	7. (c)	8. (a)	9. (e)	10. (d)

REFERENCES

References refer those books which discuss the topics given in the chapters in almost same manner.



REFERENCES

Cox, Philip T, Visual Programming Languages. In in Encyclopedia of Computer Science and Engineering, B.W. Wah (Ed.), John Wiley & Sons Inc., Hoboken, (June 2008).
2. Kindborg, Mikael, How Children Understand Concurrent Comics: Experiences from LOFI and HIFI Prototypes. In 2001 IEEE Symposia on Human-Centric Computing Languages and Environments , Stresa, Italy, September 2001.
3. Ryder, Barbara, Mary Lou Soffa and Margaret Burnett, The Impact of Software Engineering Research on Modern Programming Languages. In ACM Transactions on Software Engineering and Methodology, October, 2005. Pages 431 to 477.
4. Störrle, Harald, VMQL: A Generic Visual Model Query Language. In IEEE Symposium on Visual Languages/Human Centric Computing, Corvallis, Oregon, September 2009.
5. Zhang, Kang, Visual Languages and Applications. In Research Manuscript, Springer, 2007.

TABLE OF CONTENTS

PREFACE

JavaScript is a programming language that is primarily used by Web browsers to provide users with a dynamic and interactive experience. The majority of the functions and applications that make the Internet indispensable in modern life are written in JavaScript. While JavaScript is not the only client-side scripting language available on the Internet, it was one of the first and remains the most popular. Enterprising programmers have created JavaScript libraries, which are more concise languages built from JavaScript building blocks that are less complex and can be targeted for specific applications. JavaScript has become integral to the Internet experience as developers build increased interaction and complexity into their applications. Search engines, ecommerce, content management systems, responsive design, social media and phone apps would not be possible without it.

Organization of the Book

This edition contains nine chapters. Information is completely revised and new chapters are added. This book provides clear guidance on how to use approaches to writing JavaScript. A guide for beginners offers an overview of JavaScript basics and explains how to create Web pages, identify browsers, and integrate sound, graphics, and animation into Web applications.

Chapter 1 presents an introduction to JavaScript. It also explains the inheritance, including with the cascade. You will also understand the JavaScript tools.

Chapter 2 is intended to focus on JavaScript syntax that specifies the correct combined sequence of symbols to form a correctly structured program using a given programming language.

Chapter 3 focuses on Built in Functions that performs an action or returns a value. You can work with functions as if they were objects. They can also be passed around as arguments to other functions or be returned from those functions.

Chapter 4 gives an exploration on the appearance of cascading style sheets. In this chapter, you will understand forms basics, HTML form elements, and styling HTML forms.

Chapter 5 is aimed to discuss the HTML DOM methods and HTML DOM document. Further, DOM elements and changing HTML are presented. How to change CSS and HTML DOM animation is described as well, including HTML DOM events and HTML DOM EventListener.

Chapter 6 begins with the basics to reading/writing cookies with JavaScript. The settings of different cookie in JavaScript are also given, further.

Chapter 7 discusses about Java Script Classes and Objects. Functions can be used to somewhat simulate classes, but in general JavaScript is a class-less language. Everything is an object. And when it comes to inheritance, objects inherit from objects, not classes from classes as in the "class"-ical languages.

Chapter 8 highlights on JavaScript BOM. BOM refers to Windows objects in JavaScript. Modern browsers have implemented the same methods and properties for JavaScript interactions, often referred to as BOM's methods and properties.

Chapter 9 explores on JavaScript events. The Javascript interacts with the documents HTML code using events, which are triggered when a particular moment of interest happens in the document or the browser window.

CHAPTER 1

INTRODUCTION TO JAVASCRIPT

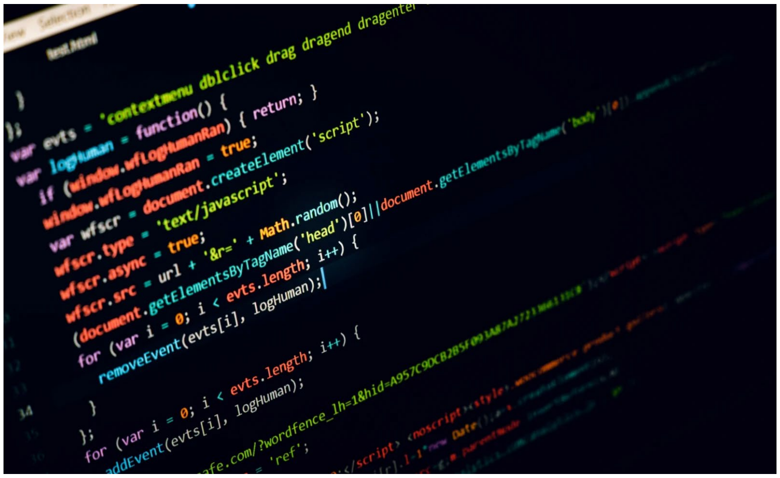
"jQuery is by far the most widely used library for JavaScript. It is used on more than 50% of websites. Many frameworks, such as Backbone and Twitter's Bootstrap, are built on top of jQuery. Being able to extend and write plugins for jQuery can not only save lots of time, but also makes code much cleaner and easier to maintain."

– Robert Duchnik

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Give meaning of JavaScript
2. Define hello world: writing your first JavaScript program
3. Explain the how to add JavaScript to your website using html
4. Understand the JavaScript tools



INTRODUCTION

Java Script often abbreviated as JS, is a programming language that conforms to the ECMA Script specification. JavaScript is high-level, often just-in-time compiled, and multi-paradigm. It has curly-bracket syntax, dynamic typing, prototype-based object-orientation, and first-class functions.

Alongside HTML and CSS, JavaScript is one of the core technologies of the World Wide Web. Over 97% of websites use it client-side for web page behavior, often incorporating third-party libraries. Most web browsers have a dedicated JavaScript engine to execute the code on the user's device.

As a multi-paradigm language, JavaScript supports event-driven, functional, and imperative programming styles. It has application programming interfaces (APIs) for working with text, dates, regular expressions, standard data structures, and the Document Object Model (DOM).

The ECMAScript standard does not include any input/output (I/O), such as networking, storage, or graphics facilities. In practice, the web browser or other runtime system provides JavaScript APIs for I/O.

JavaScript engines were originally used only in web browsers, but they are now core components of other software systems, most notably servers and a variety of applications.

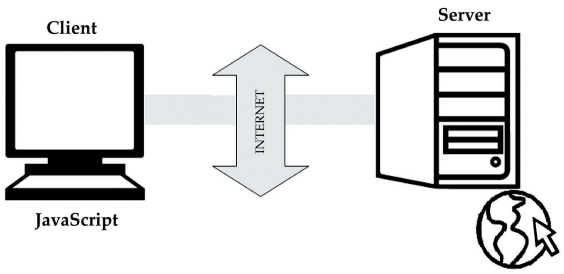
Although there are similarities between JavaScript and Java, including language name, syntax, and respective standard libraries, the two languages are distinct and differ greatly in design.

1.1 MEANING OF JAVASCRIPT

Javascript is a dynamic computer programming language. It is lightweight and most commonly used as a part of **web pages**, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

JavaScript was first known as LiveScript, but Netscape changed its name to JavaScript, possibly because of the excitement being generated by Java. JavaScript made its first appearance in Netscape 2.0 in 1995 with the name LiveScript. The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and other web browsers.

JavaScript is a very powerful client-side scripting language. JavaScript is used mainly for enhancing the interaction of a user with the webpage. In other words, you can make your webpage more lively and interactive, with the help of JavaScript. JavaScript is also being used widely in game development and Mobile application development.



The ECMA-262 Specification defined a standard version of the core JavaScript language.

- JavaScript is a lightweight, interpreted programming language.
- Designed for creating network-centric applications.
- Complementary to and integrated with Java.
- Complementary to and integrated with HTML.
- Open and cross-platform

1.1.1 The History of JavaScript

Beginnings at Netscape

In 1993, the National Center for Supercomputing Applications (NCSA), a unit of the University of Illinois at Urbana-Champaign, released NCSA Mosaic, the first popular graphical Web browser, which played an important part in expanding the growth of the nascent World Wide Web. In 1994, a company called Mosaic Communications was founded in Mountain View, California and employed many of the original NCSA Mosaic authors to create Mosaic Netscape. However, it intentionally shared no code with NCSA Mosaic. The internal codename for the company's browser was Mozilla, which stood for "Mosaic killer", as the company's goal was to displace NCSA Mosaic as the world's number one web browser. The first version of the Web browser, Mosaic Netscape 0.9, was released in late 1994. Within four months it had already taken three-quarters of the

Keyword

A **web page** or webpage is a document commonly written in HyperText Markup Language (HTML) that is accessible through the Internet or other network using an Internet browser.

browser market and became the main browser for the Internet in the 1990s. To avoid trademark ownership problems with the NCSA, the browser was subsequently renamed Netscape Navigator in the same year, and the company took the name Netscape Communications. Netscape Communications realized that the Web needed to become more dynamic. Marc Andreessen, the founder of the company believed that HTML needed a "glue language" that was easy to use by Web designers and part-time programmers to assemble components such as images and plugins, where the code could be written directly in the Web page markup.

In 1995, Netscape Communications recruited Brendan Eich with the goal of embedding the Scheme programming language into its Netscape Navigator. Before he could get started, Netscape Communications collaborated with Sun Microsystems to include in Netscape Navigator Sun's more static programming language Java, in order to compete with Microsoft for user adoption of Web technologies and platforms. To defend the idea of JavaScript against competing proposals, the company needed a prototype. Eich wrote one in 10 days, in May 1995.

Remember

Netscape Communications then decided that the scripting language they wanted to create would complement Java and should have a similar syntax, which excluded adopting other languages such as Perl, Python, TCL, or Scheme.

Although it was developed under the name Mocha, the language was officially called LiveScript when it first shipped in beta releases of Netscape Navigator 2.0 in September 1995, but it was renamed JavaScript when it was deployed in the Netscape Navigator 2.0 beta 3 in December. The final choice of name caused confusion, giving the impression that the language was a spin-off of the Java programming language, and the choice has been characterized as a marketing ploy by Netscape to give JavaScript the cachet of what was then the hot new Web programming language.

There is a common misconception that JavaScript was influenced by an earlier Web page scripting language developed by Nombas named Cmm (not to be confused with the later C-- created in 1997). Brendan Eich, however, had never heard of Cmm before he created LiveScript. Nombas did pitch their embedded Web page scripting to Netscape, though Web page scripting was not a new concept, as shown by the ViolaWWW Web browser. Nombas later switched to offering JavaScript instead of Cmm in their ScriptEase product and was part of the TC39 group that standardized ECMAScript.

Server-side JavaScript

In December 1995, soon after releasing JavaScript for browsers, Netscape introduced an implementation of the language for server-side scripting with Netscape Enterprise Server.

Since 1996, the IIS web-server has supported Microsoft's implementation of server-side Javascript -- JScript—in ASP and .NET pages. Since the mid-2000s, additional server-side JavaScript implementations have been introduced, such as Node.js in 2009.

Adoption by Microsoft

Microsoft script technologies including VBScript and JScript were released in 1996. JScript, a reverse-engineered implementation of Netscape's JavaScript, was part of Internet Explorer 3. JScript was also available for server-side scripting in Internet Information Server. Internet Explorer 3 also included Microsoft's first support for CSS and various extensions to HTML, but in each case the implementation was noticeably different from that found in Netscape Navigator at the time. These differences made it difficult for designers and programmers to make a single website work well in both browsers, leading to the use of "best viewed in Netscape" and "best viewed in Internet Explorer" logos that characterized these early years of the browser wars. JavaScript began to acquire a reputation for being one of the roadblocks to a cross-platform and standards-driven Web. Some developers took on the difficult task of trying to make their sites work in both major browsers, but many could not afford the time. With the release of Internet Explorer 4, Microsoft introduced the concept of Dynamic HTML, but the differences in language implementations and the different and proprietary Document Object Models remained and were obstacles to widespread take-up of JavaScript on the Web.

Standardization

In November 1996, Netscape submitted JavaScript to ECMA International to carve out a standard specification, which other browser vendors could then implement based on the work done at Netscape. This led to the official release of the language specification ECMAScript published in the first edition of the ECMA-262 standard in June 1997, with JavaScript being the most well-known of the implementations. ActionScript and JScript were other well-known implementations of ECMAScript.

The standards process continued in cycles, with the release of ECMAScript 2 in June 1998, which brings some modifications to conform to the ISO/IEC 16262 international standard. The release of ECMAScript 3 followed in December 1999, which is the baseline for modern day JavaScript. The original ECMAScript 4 work led by Waldemar Horwat (then at Netscape, now at Google) started in 2000 and at first, Microsoft seemed to participate and even implemented some of the proposals in their JScript .NET language.

Over time it was clear though that Microsoft had no intention of cooperating or implementing proper JavaScript in Internet Explorer, even though they had no competing proposal and they had a partial (and diverged at this point) implementation on the ..NET server side. So by 2003, the original ECMAScript 4 work was mothballed

The next major event was in 2005, with two major happenings in JavaScript's history. First, Brendan Eich and Mozilla rejoined Ecma International as a not-for-profit member and work started on ECMAScript for XML (E4X), the ECMA-357 standard, which came from ex-Microsoft employees at BEA Systems (originally acquired as Crossgain). This led to working jointly with Macromedia (later acquired by Adobe Systems), who were implementing E4X in ActionScript 3 (ActionScript 3 was a fork of original ECMAScript 4). So, along with Macromedia, work restarted on ECMAScript 4 with the goal of standardizing what was in ActionScript 3. To this end, Adobe Systems released the ActionScript Virtual Machine 2, code named Tamarin, as an open source project. But Tamarin and ActionScript 3 were too different from web JavaScript to converge, as was realized by the parties in 2007 and 2008.

Alas, there was still turmoil between the various players; Douglas Crockford—then at Yahoo!—joined forces with Microsoft in 2007 to oppose ECMAScript 4, which led to the ECMAScript 3.1 effort. The development of ECMAScript 4 was never completed, but that work influenced subsequent versions.

While all of this was happening, the open source and developer communities set to work to revolutionize what could be done with JavaScript. This community effort was sparked in 2005 when Jesse James Garrett released a white paper in which he coined the term Ajax, and described a set of technologies, of which JavaScript was the backbone, used to create web applications where data can be loaded in the background, avoiding the need for full page reloads and leading to more dynamic applications. This resulted in a renaissance period of JavaScript usage spearheaded by open source libraries and the communities that formed around them, with libraries such as Prototype, jQuery, Dojo Toolkit, MooTools, and others being released.

In July 2008, the disparate parties on either side came together in Oslo. This led to the eventual agreement in early 2009 to rename ECMAScript 3.1 to ECMAScript 5 and drive the language forward using an agenda that is known as Harmony. ECMAScript 5 was finally released in December 2009. In June 2011, ECMAScript 5.1 was released to fully align with the third edition of the ISO/IEC 16262 international standard. ECMAScript 2015 was released in June 2015. ECMAScript 2016 was released in June 2016. The current version is ECMAScript 2017, released in June 2017.

Later Developments

JavaScript has become one of the most popular programming languages on the Web. Initially, however, many professional programmers denigrated the language because, among other reasons, its target audience consisted of Web authors and other such "amateurs". The advent of Ajax returned JavaScript to the spotlight and brought more professional programming attention. The result was a proliferation of comprehensive frameworks and libraries, improved JavaScript programming practices, and increased usage of JavaScript outside Web browsers, as seen by the proliferation of Server-side JavaScript platforms. In January 2009, the CommonJS project was founded with the goal of specifying a common standard library mainly for JavaScript development outside the browser. With the rise of single-page applications and JavaScript-heavy sites, it is increasingly being used as a compile target for source-to-source compilers from both dynamic languages and static languages.

Did You Know?

JavaScript was designed by Brendan Eich in 1995 for Netscape to allow developers to enhance web pages with things like animated drop-down menus, and validating form entries.

1.1.2 Client-side JavaScript

Client-side JavaScript is the most common form of the language. The script should be included in or referenced by an HTML document for the code to be interpreted by the browser.

It means that a web page need not be a static HTML, but can include programs that interact with the user, control the browser, and dynamically create HTML content.

The JavaScript client-side mechanism provides many advantages over traditional CGI server-side scripts.

example

you might use JavaScript to check if the user has entered a valid e-mail address in a form field.

The JavaScript code is executed when the user submits the form, and only if all the entries are valid, they would be submitted to the **Web Server**. JavaScript can be used to trap user-initiated events such as button clicks, link navigation, and other actions that the user initiates explicitly or implicitly.

1.1.3 Limitations of JavaScript

- For security reason, JavaScript does not allow the reading or writing of files.
- This doesn't have any multiprocessor/multi threading capabilities.

- As there is no support available, this cannot be used for networking applications.
- Cannot access web pages hosted on a different domain.
- Cannot access databases.
- Depends a lot on the browser.
- Inability to use local devices.
- JavaScript can be disabled.
- Not Search Engine Friendly.
- JavaScript cannot protect your page source or images.

Though the HTML and JavaScript may seem very old, there is nothing inherently problematic about making a complex application with them. The larger problems of web applications must deal with have to do with the nature of the world wide web(WWW) which is inconsistent of network communication and the statelessness of HTTP.

Keyword

A **Web server** is a program that uses HTTP (Hypertext Transfer Protocol) to serve the files that form Web pages to users, in response to their requests, which are forwarded by their computers' HTTP clients.

1.1.4 Advantages of CSS

The biggest advantages to a JavaScript having an ability to produce the same result on all modern browsers.

- **Speed**. Client-side JavaScript is very fast because it can be run immediately within the client-side browser. Unless outside resources are required, JavaScript is unhindered by network calls to a backend server. It also has no need to be compiled on the client side which gives it certain speed advantages (granted, adding some risk dependent on that quality of the code developed).
- **Simplicity**. JavaScript is relatively simple to learn and implement.
- **Popularity**. JavaScript is used everywhere in the web. The resources to learn JavaScript are numerous. StackOverflow and GitHub have many projects that are using Javascript and the language as a whole has gained a lot of traction in the industry in recent years especially.
- **Interoperability**. JavaScript plays nicely with other languages and can be used in a huge variety of

applications. Unlike PHP or SSI scripts, JavaScript can be inserted into any web page regardless of the file extension. JavaScript can also be used inside scripts written in other languages such as Perl and PHP.

- **Server Load**. Being client-side reduces the demand on the website server.
- **Extended Functionality**. Third party add-ons like Greasemonkey enable JavaScript developers to write snippets of JavaScript which can execute on desired web pages to extend its functionality.
- **Versatility**. Nowadays, there are many ways to use JavaScript through Node.js servers. If you were to bootstrap node.js with Express, use a document database like mongodb, and use JavaScript on the front-end for clients, it is possible to develop an entire JavaScript app from front to back using only JavaScript.
- **Updates**. Since the advent of EcmaScript 5 (the scripting specification that Javascript relies on), Ecma International has dedicated to updating JavaScript annually. So far, we have received browser support for ES6 in 2017 and look forward to ES7 being supported in future months.

1.1.5 JavaScript Disadvantages

Biggest disadvantages to a JavaScript, code visible to everyone.

- **Code Always Visible:** The biggest disadvantages is code always visible to everyone .anyone can view JavaScript code
- **Bit of Slow execute:** No matter how much fast JavaScript interpret, JavaScript **DOM (Document Object Model)** is slow and will be a never fast .rendering with HTML
- **Stop Render:** JavaScript single error can stop to render with entire site. However browsers are extremely tolerant of JavaScript errors.

Keyword

The **Document Object Model (DOM)** is a programming API for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated.

1.2 HELLO WORLD: WRITING YOUR FIRST JAVASCRIPT PROGRAM

"Hello World" is a staple of programming courses. The objective of this program is simple: output the text "Hello World" on a computer screen. Because of the simplicity of the message and syntax, it is usually the first program taught to beginners. Writing a "Hello World" program in JavaScript, as we will learn, is exceptionally easy and requires not more than 3 lines of code.

What You'll Need

Since JavaScript is interpreted by the browser itself, we don't need any fancy compilers or additional software to write JS programs. All you need is:

- **A text editor**. Your humble Notepad will do just fine, but we highly recommend Notepad++ (free).
- **A web browser**. You can use anything you want – Google Chrome, Firefox, Internet Explorer or Safari.

1.2.1 Getting Started: Creating the HTML Framework

JavaScript programs are usually embedded within the web page itself. This means they are written along with the HTML, though you can include them externally as well.

To get started, we will first create a simple HTML file where we can include our JavaScript.

Open your text editor and type the following code into it:

```
<!DOCTYPE HTML>
<html>
<head>
<title>JavaScript Hello World</title>
</head>
<body>
<h1>JavaScript Hello World Example</h1>
</body>
</html>
```

Save this file as test.html (make sure to save as 'All Files' if using Notepad).

This is a standard HTML template, nothing special about it. It should be pretty clear to anyone with even a basic grasp of HTML.

1.2.2 Adding the JavaScript Code

We can now go ahead and write the JavaScript program.

Add the following code after the <h1> tag and save the file:

```
<script>
<alert(“Hello World!”)
<script>
```

That’s it! You’ve now successfully crated a JavaScript program.

Now use your web browser to open test.html. This is what you should see:

Easy, right?

All JavaScript code is written between <script></script> tags. We use ‘alert’ to create a function. The text to be displayed is written between quotes in brackets.

But what if we wanted to create a separate “Hello World!” function we can call anytime?

We can do that as well using just a few lines of code.

1.2.3 Creating a “Hello World” JavaScript Function

A function is any block of code that can be ‘called’ any number of times within a program. Functions are extremely useful in programming since you can create them once, use them n number of times.

We created a “Hello World!” alert box in the above example. Now we’ll create a function that will create the same alert box whenever we want.

Type in the following code into your text file:

```
<!DOCTYPE HTML>
<html>
<head>
<script>
function myFunction()
{
alert(“Hello World!”)
}
</script>
</head>
<body>
```

</body>

</html>

Save this as test2.html.

Instead of adding the script in the <body>, we added the script to the <head> and created a function called 'helloWorld'. You can turn any piece of code into a function by wrapping it in { } brackets and adding "function functionName()" before it.

Now that we've created the function, we can call on it any number of times.

Add the following code anywhere between the <body></body> tags:

<p><button onclick="myFunction()">Create a Dialog Box!</button></p>

<p><button onclick="myFunction()">Create Another Dialog Box!</button></p>

Altogether, your code should look like this:

Now open the test2.html file in your web browser. This is what you should see:

Click on either of the two buttons and you'll see the "Hello World!" **dialog box** pop up.

Congratulations! You've now successfully created a function in JavaScript. This is just the beginning, however. There is still a lot more to learn in this wonderful programming language. This course on JavaScript for beginners should help you get started.

Keyword

A **dialog box** is the about box found in many software programs, which usually displays the name of the program, its version number, and may also include copyright information.

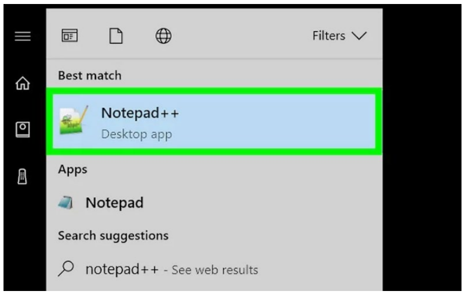
1.3 HOW TO ADD JAVASCRIPT TO YOUR WEBSITE USING HTML

Steps

1. **Open a simple text editor.** Notepad on Windows and TextEdit on Mac are the native text editors that ship with the operating systems.

- On Windows, type Notepad in the Start menu's search field to locate Notepad on your computer, then click Notepad when it appears in the results.

- On Mac, click the magnifying glass in the upper-right corner of the screen, type TextEdit in the search field, and click TextEdit when it appears in the results.



2. **Start an HTML block.** Include the HTML tags, including the <head> and </head> combination pair, as well as the <body> and </body> combination pair. Include all the tags needed to start the page, as shown:

   `<html>

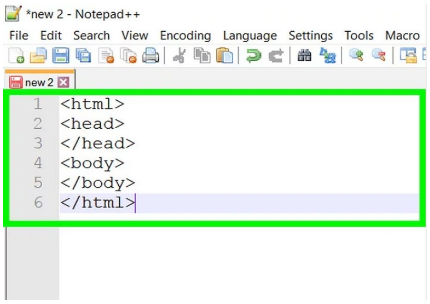
   <head>

   </head>

   <body>

   </body>

   </html>

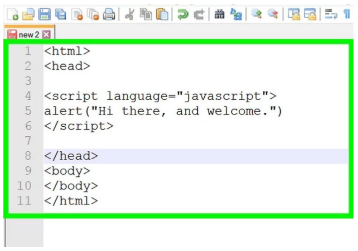

*new 2 - Notepad++
File Edit Search View Encoding Language Settings Tools Macro
new 2

```
<html>
<head>
</head>
<body>
</body>
</html>
```

3. **Add a script tag to the HTML head.** To do so, insert a **<script language="javascript">** tag in the head. This tells the text editor that you›d like to use JavaScript language to write your HTML JavaScript "program." In this example, we will greet the user using an alert.

- Add the script tag the HTML head of your own website to add JavaScript.
- If you want the script to run automatically run when the site loads, don't include a function. If you want to call it, include a function.

```
<html>
<head>
<script language="javascript">
alert("Hi there, and welcome.")
</script>
</head>
<body>
</body>
</html>
```

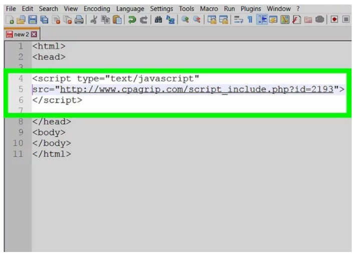


```
<html>
<head>

<script language="javascript">
alert("Hi there, and welcome.")
</script>

</head>
<body>
</body>
</html>
```

4. **Call up other JavaScript scripts using a JavaScript function.** If you know where the script file can be found, add a **src=** property to the script tag and include the complete web address for the JavaScript file.

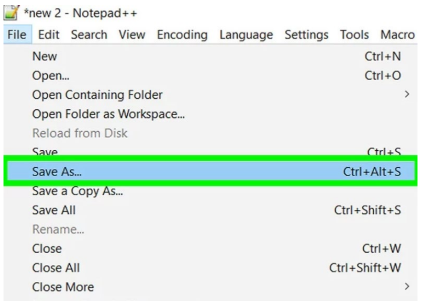
■ When calling up a script on your own site, make sure to link directly to the Javascript file and not to the URL or the other page from where the script is being called.

```
<html>
<head>
<script type="text/javascript" src="http://www.cpagrip.com/script_include.php?id=2193">
</script>
</head>
<body>
</body>
</html>
```

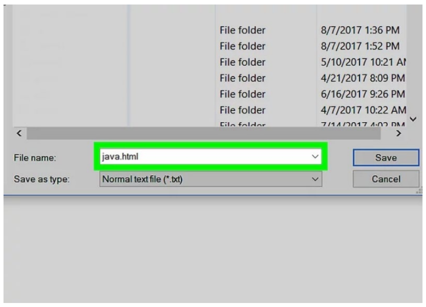

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
new 2

```
<html>
<head>

<script type="text/javascript"
src="http://www.cpagrip.com/script_include.php?id=2193">
</script>

</head>
<body>
</body>
</html>
```

5. Click File in the menu bar, and Save As... (Windows) or Save (Mac).



6. Click .html in the format drop-down.

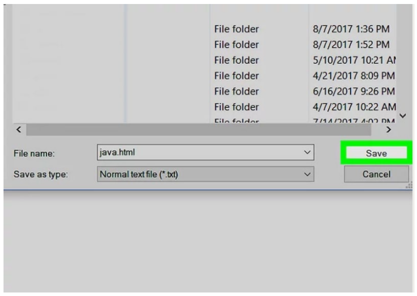


```
File folder    8/7/2017 1:36 PM
File folder    8/7/2017 1:52 PM
File folder    5/10/2017 10:21 AI
File folder    4/21/2017 8:09 PM
File folder    6/16/2017 9:26 PM
File folder    4/7/2017 10:22 AM
```

File name: java.html

Save as type: Normal text file (*.txt)

7. Name your document and click Save.



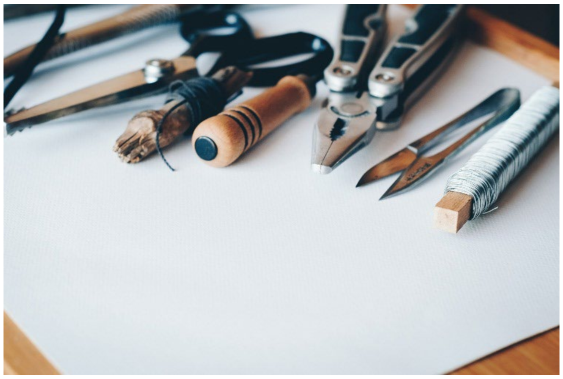
```
File folder    8/7/2017 1:36 PM
File folder    8/7/2017 1:52 PM
File folder    5/10/2017 10:21 AI
File folder    4/21/2017 8:09 PM
File folder    6/16/2017 9:26 PM
File folder    4/7/2017 10:22 AM
```

File name: java.html

Save as type: Normal text file (*.txt)

1.4 JAVASCRIPT TOOLS

Today JavaScript serves as a powerful and stable basis for lots of advanced web applications and websites. In capable hands of experienced JS developer, it can take user experience to the next level and provide with rich features and highly-functional components. Its ecosystem accounts dozens of JavaScript tools. It seems that new libraries or frameworks take the dev community by assault nearly every week. So how to choose necessary tools for JavaScript development?



Selecting front end tools based on their popularity isn't a bad idea. Widely used JavaScript development tools are more stable, supported by a larger community and eventually more reliable. Rating is another key factor that should be taken into account. We've already shared our research on the best JavaScript libraries and frameworks; however, advanced JavaScript programming is a difficult task and requires complex approach, especially talking about cross-browser compatibility and further scaling.

1.4.1 JavaScript Build tools and Automation Systems

Build tools for cross-platform languages like .Net or Java are a usual thing; however, using such capabilities with JavaScript can seem ridiculous. Though, times change. As soon as developers have started to use JS for large-scale projects, they've faced the issues with scalability, maintenance, security and general performance. This is where build tools can prove to be useful.

- **Webpack** is one of the latest front-end dev tools. It is a module bundler that creates a dependency graph with all the modules needed by your JavaScript application. Webpack packages them into one or several small bundles to be loaded by a browser. Beyond that, it is often used as a task runner, because it analyzes dependencies between modules and generates assets.

- Task runners like Grunt are used for one major purpose—automation of repetitive and time-consuming tasks. It comes with a huge ecosystem (over 6010 plugins), though, currently, JavaScript developers tend to apply more advanced tools.
- **Gulp** has been released after Grunt and, despite the fact that it is another task runner, it takes the absolutely different approach, defining tasks as JavaScript functions. It automates painful tasks, offering large ecosystem, and provides better transparency and control over the processes.
- **Browserify** allows software developers to employ node.js style modules in a browser. You define the dependencies and it bundles it all into a neat JS file. As a result, JavaScript files can be included using *"require"* statements and enable modules' import from npm.
- The main idea behind Brunch.io is simplicity and speed. It comes with light and simple configuration and detailed documentation for a quick start. Brunch automatically creates a source map for your JavaScript files, together with CSS stylesheets, simplifying the **debugging** process on the client side.
- **Yeoman** is a multi-operated tool since it can be used with any programming language (JavaScript, Python, C#, Java, etc.). It is a basic scaffolding system for web app development with the rich ecosystem (6213 plugins) and ability to create the new generators. Yeoman allows developers to quickly create new projects and to enhance the maintenance of existing ones.

1.4.2 JavaScript IDE and editors

Keyword

Debugging is the process of finding and resolving defects or problems within a computer program that prevent correct operation of computer software or a system.

JavaScript IDEs and editors can become unparalleled assistants with code completion, debugging and crafting quality apps. They quickly configure the working environment and ensure better productivity. IDE or Integrated Development Environment comes with rich functionality and support for AML systems. While editors include only the essential features, ensuring quick start with smooth responsive performance.

- **WebStorm** is a powerful IDE for advanced JavaScript development. It offers support for various frameworks and stylesheet languages, both web and server, mobile and desktop. WebStorm can be seamlessly integrated with additional tools like test runners, linters, builder, etc. It comes with such functions as code completion, immediate error detection, navigation, embedded terminal, rich plugin ecosystem, and much more.
- **Atom** from GitHub team is the number 1 choice for lots of people. It's an easily customizable text editor that comes with multiple features right out of the box. Atom includes embedded package manager, smart auto-completion, file system browser, cross-

platform support, and some other useful functions.

- **Visual Studio Code** is backed by Microsoft and complete with the ultimate support for TypeScript right out of the box. It offers smart completions and syntax highlighting with IntelliSense, debugging right from the editor, built-in Git commands, version control, and so on. Moreover, the functionality of VS Code can enriched with a wide range of extensions.
- **Brackets** is a lightweight open-source text editor. It is mainly focused on visual tools and preprocessor support, to make it easier for you to design in the browser. Brackets comes with convenient real-time preview and powerful inline editors.

1.4.3 JavaScript Documentation Tools

Documentation turns your application into a glass box, making the inner processes understandable and obvious. It explains how the software operates and how it should be used. Automated documentation tools describe functions and their purposes, thus saving time on analysis and understanding of each in the future.

- **Swagger** is generally a set of rules and tools for describing APIs. And here's the thing, it is a language-agnostic utility for getting everyone (both developers and non-developers) on the same page. Swagger creates clear documentation that is both machine and human readable, allowing for automation of API-dependent processes.
- **JSDoc** Toolkit automatically generates template-formatted, multi-page text-based documentation (HTML, JSON, XML, etc.) from comments in the JavaScript source code. Written in JavaScript, this application can come in handy for managing large-scale projects.
- **jGrouseDoc** (jGD) is a flexible open source tool that allows developers to generate API documentation from the comments in the JS source code. It documents not only variables and functions, but namespaces, interfaces, packages, and some other elements as well.
- **YUIDoc** is a Node.js app that follows the same principles of generating API documentation from comments in source code. It uses syntax similar to Javadoc and Doxygen tools and offers live previews, extensive language support, and advanced markup.
- **Docco** is a free documentation tool written in Literate CoffeeScript. It generates HTML doc to display your comments interlaced with your code. It is not restricted to JavaScript only, since there are versions for Python, Ruby, Clojure, and so others.

1.4.4 JavaScript Testing Tools

JavaScript testing tools or testing frameworks ensure software stability by discovering more errors before software reaches the end users. With the growing complexity of custom applications, automated tests not just enhance the productivity of the development house, but also help companies to keep the budget and avoid excess costs.

Keyword

Behavior Driven Development (BDD) is a methodology for developing software through continuous example-based communication between developers, QAs and BAs.

- **Jasmine** is a **behavior-driven development (BDD)** framework for testing your JavaScript code. There are no external dependencies and it doesn't require DOM to kick-start. To make test writing easier and faster, it has a clean and understandable syntax. It can be also used for testing Node.js, Python and Ruby code.
- **Mocha** is a functional test framework that runs on Node.js and in a browser. Being a "go-to" solution for many developers, it conducts tests in series to provide accurate and flexible reporting, while making asynchronous tests fun and easy. Mocha is often paired with Chai for verifying the test results.
- **PhantomJS** is often used for front-end and unit tests. Due to the fact that it is a headless WebKit, scripts run much faster, compared to common browser-based approach. It also includes native support for different web standards, like JSON, Canvas, DOM handling, SVG, and CSS selector.
- **Protractor** is a Node.js end-to-end test framework for AngularJS and Angular applications. Built on the top of WebDriverJS, it tests your apps like end users would, using browser-specific drivers and native events.

1.4.5 JavaScript Debugging Tools

Debugging code is a time-consuming and laborious task for JS developers. Debuggers can come in handy while debugging thousands of code lines, offering better convenience and ensuring more accurate results.

- **JavaScript Debugger** from Mozilla Developer Network (MDN) can be used as a stand-alone web app for debugging code in other browsers and in Node.js. Firefox offers local and remote functionality, as well as the ability to debug code running on Android device with Firefox for Android.
- **Chrome Dev Tools** kit includes multiple utilities for debugging JavaScript code, editing CSS and testing apps' performance.
- **ng-inspector** is an extension for Firefox, Chrome and Safari browsers to help developers with developing, understanding, and debugging AngularJS applications. This utility comes with real-time updates, DOM highlighting, immediate access to scopes, models, and other apps' elements.
- **Augury** is a Chrome extension for visualizing and debugging Angular 2 applications. It allows Angular 2 developers to get direct insight into app structure, operating characteristics, and change detection.

1.4.6 Security Tools

Open source ready-made components are a gift for most companies since they help to speed up custom software development process at no cost. However, such solutions involve some risks as well. At the average, there are 105 open source components in every application, while 67 percent of apps include security vulnerabilities.

Open source is powerful, but it is essential to track the dependencies and to mitigate the security risks.

- **Snyk** is a commercial tool for discovering, fixing, and preventing known vulnerabilities in JavaScript, Java, and Ruby applications. The service has its own database of vulnerabilities and takes the data from the NSP and the NIST NVD. It allows developers to cure the security risks using patches and upgrades offered by the company.

- **Node Security Project** offers useful tools for scanning dependencies and detecting vulnerabilities. NSP uses its own database, built from npm modules scans, as well as data from public bases like NIST National Vulnerability Database (NVD). On the top of that, NSP provides integration with GitHub Pull Request and CI software, real-time checks, alerts, and recommendations on how to handle vulnerabilities within your Node.js apps.
- **RetireJS** is an open-source dependency checker. It includes various components, like a command-line scanner, Grunt plugin, Firefox and Chrome extensions, Burp and OWASP ZAP plugins. Retirejs collects the vulnerability information from the NIST NVD and other sources, like bug-tracking systems, blogs, and mailing lists.
- **Gemnasium** is a commercial tool with a free trial option. It supports various technologies and packagers, including Ruby, PHP, Bower (JavaScript), Python, and npm (JavaScript). Gemnasium security tool comes with helpful features, like auto-update, real-time alerts, security notifications, and Slack integration.
- **OSSIndex** supports various ecosystems (Java, JavaScript, and .NET/C#) and multiple platforms, like NuGet, npm, Bower, Chocolatey, Maven, Composer, Drupal, and MSI. It gathers the information about vulnerabilities from National Vulnerability Database, various security feeds, and contributions, made by the community.

1.4.7 Code optimization & analysis tools

To verify JavaScript code quality dev houses usually turn to common activities of functional and unit testing. However, there is another approach that allows developers to check the code quality and its compliance with the coding standards, namely static code analysis.

Modern software development houses integrate static code analysis tools in the delivery process to prevent poor code from reaching the production stage.

- **JSLint** is a web-based analytical tool for verifying JavaScript code quality. As soon as it detects a problem in the source, it returns a message with the problem description and its approximate location in the code. JSLint is capable of analyzing some style conventions and disclosing syntax errors and structural problems.
- **JSHint** is a flexible community-driven tool to discover errors and potential issues in your JS code. The main goal of this static code analysis tool is to help JavaScript engineers with complex programs. It is able to detect syntax errors, implicit data type conversion, or leaking variable, though it can’t define whether your software is fast, correct, or includes some memory leaks. JSHint is the fork of JSLint.

- **ESLint** is an open source linting tool for JSX and JavaScript web applications. It helps to discover doubtful patterns or find code that doesn't comply with specific style guidelines. It allows developers to detect errors in the JS code without executing it, thus saving time. Being written in Node.js is offers a prompt runtime environment and smooth installation through npm.
- **Flow** is a static code checker for JavaScript source developed by Facebook. To inspect the source for errors it uses static type annotations. In fact, types are the parameters set by developers and Flow makes sure that your software meets the requirements.

1.4.8 Version Control Tools

JavaScript version control systems are essential for smooth collaboration within a team since they ensure better maintenance of various versions and help to keep track of changes. With versioning tools, developers can work on the same project simultaneously, without conflicts and misunderstandings. Moreover, these utilities archive each version with all changes, deletions, and appendices.

- In recent years **Git** has become a widely-used version control system for both small and large-scale projects. This free utility offers outstanding operating speed and efficiency. Its popularity can be easily explained by the highly-distributed system and different type of controls, as well as a staging area, where commits can be reviewed and formatted right before completing the commit.
- **Subversion** or SVN has gained a huge popularity and it is still widely employed by open source projects and top platforms like Python Apache or Ruby. This CVS comes with lots of functions, enabling versioning of directories as first-class files, atomic commits, versioning or various operations (renaming, copying, deleting, etc.), merge tracking, file locking, and many others.

1.4.9 Package and Dependency Management Tools

Modern software is stored in the form of packages and retained in repositories. Such packages provide the initial components of an operating system, like applications, libraries, services, and docs. Whereas, package management systems take care of various operations, like installation and upgrades, and ensure that the installed software has been approved by package maintainers and developers.

- **Bower** helps to manage assets, frameworks, libraries, and other utilities. Developed by the Twitter team, it offers access to a great number of packages, helping JavaScript developers to streamline the development process and improve the deliverables.

Keyword

Browserify is an open-source JavaScript tool that allows developers to write Node.js-style modules that compile for use in the browser.

- **npm** stands for **node package manager**, though its packages can be used for both front-end and back-end. It is a package management system for JavaScript and the largest software registry in the world, numbering over 475,000 modules.
- **Yarn** is the new kid on the block, though it has already stolen the scene thanks to its promoters: Google, Facebook, Tilde, and Exponent. It's gained a reputation of a smart improvement, as compared with npm. The main focus in brought on security, speed and consistency. This tool enables code sharing via packages or modules, together with a file that describes the package.
- **Duo** takes the best practices from **Browserify**, Component, and Go turning front-end development into a fast and easy process. The main idea behind Duo is to simplify writing of modular components and making large web apps' building painless and fast.

The list of best JavaScript tools for custom web app development can go on and on, though we've just mentioned the major categories that serve as the basis for quality products.

- Some companies also use **JavaScript obfuscator tool** to protect the code. The utility makes the source harder to understand, reuse, or modify without authorization, thus keeping the JS code original.
- **JavaScript code coverage tools** allow you to track how accurately your source code is tested. Such utilities as Istanbul help developers make sure that core components are covered, and they haven't missed the edge cases, zero states, etc.
- There is a huge range of **JavaScript animation tools** to make web projects unique and eye-catching. These utilities create smooth animations and take user experience to the next level.

That being said, each team, project, and skill-set are different. Every tool, system, framework, or library is optional, thus JavaScript development has been reshaped with the last years and it still undergoes sweeping changes. It is very easy to fall into a trap of ever-augmenting complexity or move towards the latest builder every month or two. However, knowledge and experience won't become outdated.

ROLE MODEL

BRENDAN EICH: INVENTOR OF JAVAS-CRIPT PROGRAMMING LANGUAGE

Brendan Eich is an American technologist and creator of the JavaScript programming language. He co-founded the Mozilla project, the Mozilla Foundation and the Mozilla Corporation, and served as the Mozilla Corporation's chief technical officer and briefly its chief executive officer.

Early life

Brendan Eich received his bachelor's degree in mathematics and computer science at Santa Clara University. He received his master's degree in 1985 from the University of Illinois at Urbana-Champaign. Eich started his career at Silicon Graphics, working for seven years on operating system and network code. He then worked for three years at MicroUnity Systems Engineering writing microkernel and DSP code, and doing the very first MIPS R4000 port of GCC.

Netscape and JavaScript

He started work at Netscape Communications Corporation in April 1995. Having originally joined intending to put Scheme "in the browser",Eich was instead commissioned to create a new language that resembled Java, JavaScript for the Netscape Navigator Web browser. The first version was completed in ten days in order to accommodate the Navigator 2.0 Beta release schedule, and was called Mocha, which was later renamed LiveScript in September 1995 and later JavaScript in the same month. Eich continued to oversee the development of Spider Monkey, the specific implementation of JavaScript in Navigator, until 2011.

Mozilla

In early 1998, Eich co-founded the Mozilla project, with a website at mozilla.org, that was meant to manage open-source contributions to the Netscape source code. He served as Mozilla's chief architect. AOL bought Netscape in 1999. After AOL shut down the Netscape browser unit in July 2003, Eich helped spin out the Mozilla Foundation.

In August 2005, after serving as Lead Technologist and as a member of the Board of Directors of the Mozilla Foundation, Eich became CTO of the newly founded Mozilla Corporation, meant to be the Mozilla Foundation's for-profit arm.

CEO appointment and resignation

On March 24, 2014, Eich was promoted to CEO of Mozilla Corporation. His appointment sparked controversy over a $1,000 political donation Eich had made in 2008 to the successful campaign for California Proposition 8, which sought to establish that, "Only marriage between a man and a woman is valid or recognized in California. "This was criticized by gay rights activists on Twitter. In the ensuing public debate, OKCupid and two gay application developers called for a boycott of the company. Others at the Mozilla Corporation spoke out on their blogs in his favor. Board members wanted him to stay in the company with a different role.

On April 3, 2014, Eich stepped down as CEO and resigned from working at Mozilla. In his personal blog, Eich posted that "under the present circumstances, I cannot be an effective leader."

Following Eich's resignation, the National Organization for Marriage called for its own boycott of Mozilla, due to "gay activists who have forced him out of the company he has helped lead for years".

SUMMARY

- JavaScript often abbreviated as JS, is a programming language that conforms to the ECMAScript specification. JavaScript is high-level, often just-in-time compiled, and multi-paradigm. It has curly-bracket syntax, dynamic typing, prototype-based object-orientation, and first-class functions.
- Javascript is a dynamic computer programming language. It is lightweight and most commonly used as a part of **web pages**, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.
- Client-side JavaScript is the most common form of the language. The script should be included in or referenced by an HTML document for the code to be interpreted by the browser.
- "Hello World" is a staple of programming courses. The objective of this program is simple: output the text "Hello World" on a computer screen. Because of the simplicity of the message and syntax, it is usually the first program taught to beginners. Writing a "Hello World" program in JavaScript, as we will learn, is exceptionally easy and requires not more than 3 lines of code.
- Webpack is one of the latest front-end dev tools. It is a module bundler that creates a dependency graph with all the modules needed by your JavaScript application.
- A function is any block of code that can be 'called' any number of times within a program. Functions are extremely useful in programming since you can create them once, use them n number of times.
- JavaScript IDEs and editors can become unparalleled assistants with code completion, debugging and crafting quality apps. They quickly configure the working environment and ensure better productivity. IDE or Integrated Development Environment comes with rich functionality and support for AML systems.
- WebStorm is a powerful IDE for advanced JavaScript development. It offers support for various frameworks and stylesheet languages, both web and server, mobile and desktop. WebStorm can be seamlessly integrated with additional tools like test runners, linters, builder, etc. It comes with such functions as code completion, immediate error detection, navigation, embedded terminal, rich plugin ecosystem, and much more.
- Debugging code is a time-consuming and laborious task for JS developers. Debuggers can come in handy while debugging thousands of code lines, offering better convenience and ensuring more accurate results.
- JavaScript version control systems are essential for smooth collaboration within a team since they ensure better maintenance of various versions and help to keep track of changes.

KNOWLEDGE CHECK

1. Javascript is ………… language.

a. Programming

b. Application

c. None of These

d. Scripting

2. JavaScript is …………… Side Scripting Language.

a. Server

b. ISP

c. None of These

d. Browser

3. JavaScript is designed for following purpose -

a. To Style HTML Pages

b. To add interactivity to HTML Pages.

c. To Perform Server Side Scripting Opertion

d. To Execute Query Related to DB on Server

4. Which of the following JavaScript cannot do?

a. JavaScript can react to events

b. JavaScript can manipulate HTML elements

c. JavaScript can be use to validate data

d. All of the Above

5. JavaScript is an ………….. language.

a. compiled

b. interpreted

6. The “function” and “ var” are known as:

a. Keywords

b. Data types

c. Declaration statements

d. Prototypes

7. Which one of the following is the correct way for calling the JavaScript code?

a. Preprocessor

b. Triggering Event

c. RMI

d. Function/Method

8. In the JavaScript, which one of the following is not considered as an error:

a. Syntax error

b. Missing of semicolons

c. Division by zero

d. Missing of Bracket

REVIEW QUESTIONS

1. What is JavaScript?
2. What is advantage and disadvantage of JavaScript?
3. How to write the first JavaScript program?
4. How to add JavaScript to your website using HTML?
5. Discuss the JavaScript development tools.

Check Your Result

1. (d)	2. (d)	3. (b)	4. (d)	5. (b)
6. (c)	7. (d)	8. (c)		

REFERENCES

1. Flanagan, David. JavaScript: The Definitive Guide. 7th edition. Sebastopol, California: O'Reilly, 2020.
2. Haverbeke, Marijn. Eloquent JavaScript. 3rd edition. No Starch Press, 2018. 472 pages. ISBN 978-1593279509.(download)
3. Zakas, Nicholas. Principles of Object-Oriented JavaScript, 1st edition. No Starch Press, 2014. 120 pages. ISBN 978-1593275402.

CHAPTER 2

LANGUAGE SYNTAX

"A novelist can never be his own reader, except when he is ridding his manuscript of syntax errors, repetitions, or the occasional superfluous paragraph."

– Patrick Modiano

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Learn about the JavaScript syntax
2. Discuss on the appearance of JavaScript

INTRODUCTION

The syntax of a computer language is the set of rules that defines the combinations of symbols that are considered to be correctly structured statements or expressions in that language. This applies both to programming languages, where the document represents source code, and to markup languages, where the document represents data.

The syntax of a language defines its surface form. Text-based computer languages are based on sequences of characters, while visual programming languages are based on the spatial layout and connections between symbols (which may be textual or graphical). Documents that are syntactically invalid are said to have a syntax error. When designing the syntax of a language, a designer might start by writing down examples of both legal and illegal strings, before trying to figure out the general rules from these examples.

```
def add5(x):
   return x+5

def dotwrite(ast):
   nodename = getNodename()
   label=symbol.sym_name.get(int(ast[0]),ast[0])
   print '    %s [label="%s' % (nodename, label),
   if isinstance(ast[1], str):
      if ast[1].strip():
         print '= %s"];' % ast[1]
      else:
         print '"]'
    else:
       print '"];'
       children = []
       for  in n, childenumerate(ast[1:]):
          children.append(dotwrite(child))
       print ,'    %s -> {' % nodename
       for  in :namechildren
          print '%s' % name,
```

Syntax therefore refers to the *form* of the code, and is contrasted with semantics – the *meaning*. In processing computer languages, semantic processing generally comes after syntactic processing; however, in some cases, semantic processing is necessary for complete syntactic analysis, and these are done together or concurrently. In a compiler, the syntactic analysis comprises the frontend, while the semantic analysis comprises the backend (and middle end, if this phase is distinguished).

2.1 JAVASCRIPT SYNTAX

JavaScript can be implemented using JavaScript statements that are placed within the <script>... </script>.

You can place the <script> tags, containing your JavaScript, anywhere within your web page, but it is normally recommended that you should keep it within the <head> tags.

The <script> tag alerts the browser program to start interpreting all the text between these tags as a script. A simple syntax of your JavaScript will appear as follows.

<script ...>

```
  JavaScript code
</script>
```

The script tag takes two important attributes –

- **Language** – This attribute specifies what scripting language you are using. Typically, its value will be javascript. Although recent versions of HTML (and XHTML, its successor) have phased out the use of this attribute.
- **Type** – This attribute is what is now recommended to indicate the scripting language in use and its value should be set to "text/javascript".

So your JavaScript segment will look like –

```
<script language="javascript" type="text/javascript">
  JavaScript code
</script>
```

The First JavaScript Script

Let us take a sample example to print out "Hello World". We added an optional HTML comment that surrounds our JavaScript code. This is to save our code from a browser that does not support JavaScript. The comment ends with a "//-->". Here "//" signifies a comment in JavaScript, so we add that to prevent a browser from reading the end of the HTML comment as a piece of JavaScript code. Next, we call a function document.write which writes a string into our .HTML document

This function can be used to write text, HTML, or both. .Take a look at the following code

```
<html>
<body>
<"script language="javascript" type="text/javascript>
--!>
("!document.write("Hello World
<--//
<script/>
<body/>
```

Remember

Care should be taken while writing variable and function names in JavaScript.

```
<html/>
```

This code will produce the following result –

```
Hello World!
```

Whitespace and Line Breaks

JavaScript ignores spaces, tabs, and newlines that appear in JavaScript programs. You can use spaces, tabs, and newlines freely in your program and you are free to format and indent your programs in a neat and consistent way that makes the code easy to read and understand.

Semicolons are Optional

Simple statements in JavaScript are generally followed by a semicolon character, just as they are in C, C++, and Java. JavaScript, however, allows you to omit this semicolon if each of your statements are placed on a separate line. For example, the following code could be written without semicolons.

```
<script language="javascript" type="text/javascript">
   <!--
      var1 = 10
      var2 = 20
   //-->
</script>
```

But when formatted in a single line as follows, you must use semicolons –

```
<script language="javascript" type="text/javascript">
   <!--
      var1 = 10; var2 = 20;
   //-->
</script>
```

It is a good programming practice to use semicolons.

Case Sensitivity

JavaScript is a case-sensitive language. This means that the language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.

So the identifiers Time and TIME will convey different meanings in JavaScript.

Comments in JavaScript

JavaScript supports both C-style and C++-style comments, Thus −

- Any text between a // and the end of a line is treated as a comment and is ignored by JavaScript.
- Any text between the characters /* and */ is treated as a comment. This may span multiple lines.
- JavaScript also recognizes the HTML comment opening sequence <!--. JavaScript treats this as a single-line comment, just as it does the // comment.
- The HTML comment closing sequence --> is not recognized by JavaScript so it should be written as //-->.

Example

The following example shows how to use comments in JavaScript.

```
<script language="javascript" type="text/javascript">
   <!--

      // This is a comment. It is similar to comments in C++

      /*
      * This is a multiline comment in JavaScript
      * It is very similar to comments in C Programming
      */

   //-->
</script>
```

2.1.1 JavaScript Browser

JavaScript was initially created to "make webpages alive". The programs in this language are called scripts. They can be written right in the HTML and execute automatically as the page loads. Scripts are provided and executed as a plain text. They don't need a special preparation or a compilation to run. In this aspect, JavaScript is very different from another language called Java.

At present, JavaScript can execute not only in the browser, but also on the server, or actually on any device where there exists a special program called the JavaScript engine.

The browser has an embedded engine, sometimes it's also called a "JavaScript virtual machine".

Different engines have different "codenames", for example:

- V8 – in Chrome and Opera.
- SpiderMonkey – in Firefox.
- There are other codenames like "Trident", "Chakra" for different versions of IE, "ChakraCore" for Microsoft Edge, "Nitro" and "SquirrelFish" for Safari etc.

The terms above are good to remember, because they are used in developer articles on the internet. We'll use them too.

If "a feature X is supported by V8", then it probably works in Chrome and Opera.

What can in-browser JavaScript do?

The modern JavaScript is a "safe" programming language. It does not provide low-level access to memory or CPU, because it was initially created for browsers which do not require it.

The capabilities greatly depend on the environment that runs JavaScript. For instance, Node.JS supports functions that allow JavaScript to read/write arbitrary files, perform network requests etc.

In-browser JavaScript can do everything related to webpage manipulation, interaction with the user and the webserver.

For instance, in-browser JavaScript is able to:

- Add new HTML to the page, change the existing content, modify styles.
- React to user actions, run on mouse clicks, pointer movements, key presses.
- Send requests over the network to remote servers, download and upload files (so-called AJAX and COMET technologies).
- Get and set cookies, ask questions to the visitor, show messages.
- Remember the data on the client-side ("local storage").

What CAN'T in-browser JavaScript do?

JavaScript's abilities in the browser are limited for the sake of the user's safety. The aim is to prevent an evil webpage from accessing private information or harming the user's data.

The examples of such restrictions are:

- JavaScript on a webpage may not read/write arbitrary files on the hard disk, copy them or execute programs. It has no direct access to OS system functions.

Modern browsers allow it to work with files, but the access is limited and only provided if the user does certain actions, like "dropping" a file into a browser window or selecting it via an <input> tag.

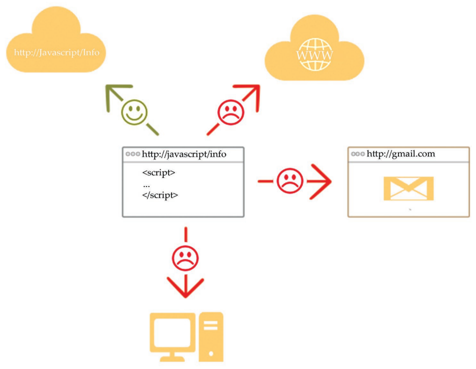
There are ways to interact with camera/microphone and other devices, but they require a user's explicit permission. So a JavaScript-enabled page may not sneakily enable a web-camera, observe the surroundings and send the information to the NSA.

- Different tabs/windows generally do not know about each other. Sometimes they do, for example when one window uses JavaScript to open the other one. But even in this case, JavaScript from one page may not access the other if they come from different sites (from a different domain, protocol or port).

This is called the "Same Origin Policy". To work around that, *both pages* must contain a special JavaScript code that handles data exchange.

The limitation is again for user's safety. A page from http://anysite.com which a user has opened must not be able to access another browser tab with the URL http://gmail.com and steal information from there.

- JavaScript can easily communicate over the net to the server where the current page came from. But its ability to receive data from other sites/domains is crippled. Though possible, it requires explicit agreement (expressed in HTTP headers) from the remote side. Once again, that's safety limitations.



Such limits do not exist if JavaScript is used outside of the browser, for example on a server. Modern browsers also allow installing plugin/extensions which may get extended permissions.

Keyword

Interface in the Java programming language is an abstract type that is used to specify a behavior that classes must implement. They are similar to protocols. Interfaces are declared using the interface keyword, and may only contain method signature and constant declarations.

What makes JavaScript unique?

There are at least *three* great things about JavaScript:

- Full integration with HTML/CSS.
- Simple things done simply.
- Supported by all major browsers and enabled by default.

Combined, these three things exist only in JavaScript and no other browser technology.

That's what makes JavaScript unique. That's why it's the most widespread tool to create browser **interfaces**.

While planning to learn a new technology, it's beneficial to check its perspectives. So let's move on to the modern trends that include new languages and browser abilities.

Languages "over" JavaScript

The syntax of JavaScript does not suit everyone's needs. Different people want different features.

That's to be expected, because projects and requirements are different for everyone. So recently a plethora of new languages appeared, which are transpiled (converted) to JavaScript before they run in the browser.

Modern tools make the transpilation very fast and transparent, actually allowing developers to code in another language and autoconverting it "under the hood".

Examples of such languages:

- CoffeeScript is a "syntactic sugar" for JavaScript, it introduces shorter syntax, allowing to write more precise and clear code. Usually Ruby devs like it.
- TypeScript is concentrated on adding "strict data typing", to simplify development and support of complex systems. It is developed by Microsoft.

- Dart is a standalone language that has its own engine that runs in non-browser environments (like mobile apps). It was initially offered by Google as a replacement for JavaScript, but as of now, browsers require it to be transpiled to JavaScript just like the ones above.

There are more. Of course even if we use one of those languages, we should also know JavaScript, to really understand what we're doing.

2.1.2 Understanding Syntax and Code Structure in JavaScript

Before learning to write in a spoken language, you must first learn the rules of grammar. Here are a few examples of rules you might find in the English language:

- A sentence starts with a capital letter.
- A sentence ends in a period.
- A new paragraph is indented.
- Spoken dialogue is placed inside double quotation marks.

Similarly, all programming languages must adhere to specific rules in order to function. This set of rules that determine the correct structure of programming languages is known as **syntax**. Many programming languages consist largely of similar concepts with variations in syntax.

Functionality and Readability

Functionality and readability are two important reasons to focus on syntax as you begin to work with JavaScript.

There are some syntax rules that are mandatory for JavaScript functionality. If they are not followed, the console will throw an error and the script will cease execution.

Consider a syntax error in the "Hello, World!" program:

broken.js

```
// Example of a broken JavaScript program
console.log("Hello, World!"
```

This code sample is missing the closing parenthesis, and instead of printing the expected "Hello, World!" to the console, the following error will appear:

Output

```
Uncaught SyntaxError: missing ) after argument list
```

The missing) must be added before the script will continue to run. This is an example of how a mistake in JavaScript syntax can break the script, as correct syntax must be followed in order for code to run.

Some aspects of JavaScript syntax and formatting are based on different schools of thought. That is, there are stylistic rules or choices that are not mandatory and will not result in errors when the code is run. However, there are many common conventions that are sensible to follow, as developers between projects and codebases will be more familiar with the style. Adhering to common conventions leads to improved readability.

Consider the following three examples of variable assignment.

```
const greeting=”Hello”;          // no whitespace between variable & string
const greeting =         “Hello”; // excessive whitespace after assignment
const greeting = “Hello”;        // single whitespace between variable & string
```

Although all three of the examples above will function exactly the same in the output, the third option of greeting = “Hello” is by far the most commonly used, and the most readable way of writing the code, especially when considering it within the context of a larger program. It is important to keep your entire coding project’s style consistent. From one organization to another, you will encounter different guidelines to follow, so you must also be flexible. We’ll go over some code examples below for you to familiarize yourself with the syntax and structure of JavaScript code.

Whitespace

Whitespace in JavaScript consists of spaces, tabs, and newlines (pressing ENTER on the keyboard). As demonstrated earlier, excessive whitespace outside of a string and the spaces between operators and other symbols are ignored by JavaScript. This means the following three examples of variable assignment will have the exact same computed output:

```
const userLocation      =     “New York City, “      +  “NY”;
const userLocation=”New York City, “+”NY”;
const userLocation = “New York City, “ + “NY”;
```

userLocation will represent «New York City, NY» no matter which of these styles are written in the script, nor will it make a difference to JavaScript whether the whitespace is written with tabs or spaces.

A good rule of thumb to be able to follow the most common whitespace conventions is to follow the same rules as you are used to in math and language grammar.

For example, let x = 5 * y is more readable than let x=5*y.

One notable exception to this style you may see is during assignment of multiple variables. Note the position of = in the following example:

```
const companyName         = “DigitalOcean”;
const companyHeadquarters = “New York City”;
const companyHandle       = “digitalocean”;
```

All the assignment operators (=) are lined up, with the whitespace after the variable. This type of organization structure is not used by every codebase, but can be used to improve readability.

Excess newlines are also ignored by JavaScript. Generally, an extra newline will be inserted above a comment and after a code block.

Parentheses

For keywords such as if, switch, and for, spaces are usually added before and after the parentheses. Observe the following examples of comparison and loops.

```
// An example of if statement syntax
if () { }

// Check math equation and print a string to the console
if (4 < 5) {
    console.log("4 is less than 5.");
}

// An example of for loop syntax
for () { }

// Iterate 10 times, printing out each iteration number to the console
for (let i = 0; i <= 10; i++) {
    console.log(i);
}
```

As demonstrated, the if statement and for loop have whitespace on each side of the parentheses (but not inside the parentheses).

When the code pertains to a function, method or class, the parentheses will be touching the respective name.

```
// An example function
function functionName() {}

// Initialize a function to calculate the volume of a cube
function cube(number) {
    return Math.pow(number, 3);
}

// Invoke the function
cube(5);
```

In the above example, cube() is a function, and the pair of parentheses () will contain the parameters or arguments. In this case, the **parameters** are number or 5, respectively. Although cube () with an extra space is valid in that the code will execute as expected, it is almost never seen. Keeping them together helps easily associate the function name to the parentheses pair and any associated passed arguments.

Keyword

Parameter is any characteristic that can help in defining or classifying a particular system. That is, a parameter is an element of a system that is useful, or critical, when identifying the system, or when evaluating its performance, status, condition, etc.

Semicolons

JavaScript programs consist of a series of instructions known as statements, just as written paragraphs consist of a series of sentences. While a sentence will end with a period, a JavaScript statement often ends in a semicolon (;).

```
// A single JavaScript statement
const now = new Date();
```

If two or more statements are next to each other, it is obligatory to separate them with a semicolon.

```
// Get the current timestamp and print it to the console
const now = new Date(); console.log(now);
```

If statements are separated by a newline, the semicolon is optional.

```
// Two statements separated by newlines
const now = new Date()
console.log(now)
```

A safe and common convention is to separate statements with a semicolon regardless of newlines. Generally, it is considered good practice to include them to reduce the probability of errors.

```
// Two statements separated by newlines and semicolons
const now = new Date();
console.log(now);
```

Semicolons are also required between the initialization, condition, and increment or decrement of a forloop.

```
for (initialization; condition; increment) {
    // run the loop
}
```

Semicolons are *not* included after any sort of block statement, such as if, for, do, while, class, switch, and function. These block statements are contained in curly brackets {}. Note the examples below.

```
// Initialize a function to calculate the area of a square
function square(number) {
    return Math.pow(number, 2);
}

// Calculate the area of a number greater than 0
if (number > 0) {
    square(number);
}
```

Be careful, as not all code encased in curly brackets will end without a semicolon. Objects are encased in curly brackets, and should end in a semicolon.

```
// An example object
const objectName = {};

// Initialize triangle object
const triangle = {
    type: “right”,
    angle: 90,
    sides: 3,
};
```

It is widely accepted practice to include semicolons after every JavaScript statement except block statements, which end in curly brackets.

Indentation

A complete JavaScript program can technically be written on a single line. However, this would quickly become very difficult to read and maintain. Instead, we use newlines .and indentation

Here's an example of a conditional if/else statement, written on either one line or .with newlines and indentation

```
// Conditional statement written on one line

if (x === 1) { /* execute code if true */ } else { /* execute code if false */ }



// Conditional statement with indentation

if (x === 1) {

// execute code if true

} else {

// execute code if false

}
```

Notice that any code included within a block is indented. The indentation can be done with two spaces, four spaces, or by pressing the tab character. Whether tabs or spaces are used is dependent on either your personal preference (for a solo project) .(or your organization's guidelines (for a collaborative project

Including the opening brace at the end of the first line, as in the above example, is the conventional way to structure JavaScript block statements and objects. Another .way you may see block statements written is with the braces on their own lines

```
// Conditional statement with braces on newlines

if (x === 1)

{

// execute code if true

}

else

{

// execute code if false

}
```

This style is much less common in JavaScript as it is in other languages, but not .unheard of

.Any nested block statement will be indented further

```
// Initialize a function

function isEqualToOne(x) {

// Check if x is equal to one

if (x === 1) {

// on success, return true

return true;

} else {

return false;

}

}
```

Proper indentation of your code is imperative to maintain readability and to mitigate confusion. One exception to this rule to keep in mind is that compressed libraries will have unnecessary characters removed, therefore rendering file sizes smaller to enable .(faster page load times (as in jquery.min.js and d3.min.js

Identifiers

The name of a variable, function, or property is known as an identifier in JavaScript. Identifiers consist of letters and numbers, but they cannot include any symbol outside .of $ and _, and cannot begin with a number

Case Sensitive

These names are case sensitive. The following two examples, myVariable and myvariable .would refer to two distinct variables

```
var myVariable = 1;

var myvariable = 2;
```

The convention of JavaScript names is that they are written in camelCase, meaning the first word is lowercase but every following word starts with an uppercase letter.

You may also see global variables or constants written in all uppercase, separated by .underscores

const INSURANCE_RATE = 0.4;

The exception to this rule is class names, which are often written with every word .(starting in an uppercase letter (PascalCase

```
// Initialize a class
class ExampleClass {
    constructor() { }
}
```

In order to ensure that code is readable, it is best to use clearly different identifiers throughout your program files.

Reserved Keywords

Identifiers also must not consist of any reserved keywords. Keywords are words in the JavaScript language that have a built-in functionality, such as var, if, for, and this.

You would not, for example, be able to assign a value to a variable named var.

var var = "Some value";

Since JavaScript understands var to be a keyword, this will result in a syntax error:

Output

```
SyntaxError: Unexpected token (1:4)
```

2.2 APPEARANCE OF JAVASCRIPT BASICS

JavaScript is a rich and expressive language in its own right. While it will be of particular value to people with no programming experience, even people who have used other programming languages may benefit from learning about some of the peculiarities of JavaScript.

2.2.1 Syntax Basics

Understanding statements, variable naming, whitespace, and other basic JavaScript syntax.

A simple variable declaration

```
var foo = 'hello world';
```

Whitespace has no meaning outside of quotation marks

```
var foo =          'hello world';
```

Parentheses indicate precedence

```
2 * 3 + 5;    // returns 11; multiplication happens first
2 * (3 + 5);  // returns 16; addition happens first
```

Tabs enhance readability, but have no special meaning

```
var foo = function() {
    console.log('hello');
};
```

2.2.2 Operators

JavaScript operators are used to assign values, compare values, perform arithmetic operations, and more. Basic operators allow you to manipulate values.

Concatenation

```
var foo = 'hello';

var bar = 'world';

console.log(foo + ' ' + bar); // 'hello world'
```

Multiplication and division

```
2 * 3;
2 / 3;
```

Incrementing and decrementing

var i = 1;

var j = ++i; // pre - increment : j equals 2; i equals 2

var k = i ++; // post - increment : k equals 2; i equals 3

Operations on Numbers and Strings: In JavaScript, numbers and strings will occasionally behave in ways you might not expect.

Addition vs. concatenation

```
var foo = 1;
var bar = '2';

console.log(foo + bar);  // 12. uh oh
```

Forcing a string to act as a number

```
var foo = 1;
var bar = '2';

// coerce the string to a number
console.log(foo + Number(bar));
```

The Number constructor, when called as a function (like above) will have the effect of casting its argument into a number. You could also use the unary plus operator, which does the same thing:

Forcing a string to act as a number (using the unary-plus operator)

```
console.log(foo + +bar);
```

Logical Operators

Logical operators allow you to evaluate a series of operands using AND and OR operations.

Logical AND and OR operators

```
var foo = 1;
var bar = 0;
var baz = 2;

foo || bar;   // returns 1, which is true
bar || foo;   // returns 1, which is true

foo && bar;   // returns 0, which is false
foo && baz;   // returns 2, which is true
baz && foo;   // returns 1, which is true
```

Though it may not be clear from the example, the || operator returns the value of the first truthy operand, or, in cases where neither operand is truthy, it'll return the last of both operands. The && operator returns the value of the first false operand, or the value of the last operand if both operands are truthy. Be sure to consult the section called "Truthy and Falsy Things" for more details on which values evaluate to true and which evaluate to false.

You'll sometimes see developers use these logical operators for flow control instead of using if statements. For example:

// do something with foo if foo is truthy

foo && doSomething (foo);

// set bar to baz if baz is truthy ;

// otherwise , set it to the return

// value of createBar ()

```
var bar = baz || createBar ();
```

This style is quite elegant and pleasantly terse; that said, it can be really hard to read, especially for beginners. I bring it up here so you'll recognize it in code you read, but I don't recommend using it until you're extremely comfortable with what it means and how you can expect it to behave.

Comparison Operators

Comparison operators allow you to test whether values are equivalent or whether values are identical.

```
var foo = 1;
var bar = 0;
var baz = '1 ';
var bim = 2;

foo == bar ; // returns false
foo != bar ; // returns true
foo == baz ; // returns true ; careful !

foo === baz ; // returns false
foo !== baz ; // returns true
foo === parseInt ( baz ); // returns true

foo > bim ; // returns false
bim > baz ; // returns true
foo <= baz ; // returns true
```

2.2.3 Conditional Code

Sometimes you only want to run a block of code under certain conditions. Flow control — via if and else blocks — lets you run code only under certain conditions.

Flow control

```
var foo = true ;
var bar = false ;
if ( bar ) {
// this code will never run
console . log ( ' hello ! ');
    }

    if ( bar ) {

    // this code won 't run

    } else {

    if ( foo ) {

    // this code will run

    } else {

    // this code would run if foo and bar were both false

    }

    }
```

Remember

While curly braces aren't strictly required around single-line if statements, using them consistently, even when they aren't strictly required, makes for vastly more readable code. Be mindful not to define functions with the same name multiple times within separate if/else blocks, as doing so may not have the expected result.

Truthy and Falsy

Things In order to use flow control successfully, it's important to understand which kinds of values are "truthy" and which kinds of values are "falsy." Sometimes, values that seem like they should evaluate one way actually evaluate another.

Values that evaluate to true

```
'0 ';
'any string ';
[]; // an empty array
{}; // an empty object
1; // any non - zero number
```

Values that evaluate to false

```
0;
' '; // an empty string
NaN ; // JavaScript 's "not -a- number " variable
null ;
undefined ; // be careful -- undefined can be redefined !
```

Conditional Variable Assignment with the Ternary Operator

Sometimes you want to set a variable to a value depending on some condition. You could use an if/else statement, but in many cases the ternary operator is more convenient. [Definition: The **ternary operator** tests a condition; if the condition is true, it returns a certain value, otherwise it returns a different value.]

Keyword

Ternary operator is an operator that takes three arguments. The first argument is a comparison argument, the second is the result upon a true comparison, and the third is the result upon a false comparison. If it helps you can think of the operator as shortened way of writing an if-else statement.

The ternary operator

```
// set foo to 1 if bar is true ;
// otherwise , set foo to 0
var foo = bar ? 1 : 0;
```

While the ternary operator can be used without assigning the return value to a variable, this is generally discouraged.

Switch Statements

Rather than using a series of if/else if/else blocks, sometimes it can be useful to use a switch statement instead. [Definition: Switch statements look at the value of a variable or expression, and run different blocks of code depending on the value.]

```
switch ( foo ) {
case 'bar ':
alert ( ' the value was bar -- yay ! ');
break ;
case 'baz ':
alert ( ' boo baz :( ');
break ;
default :
alert ( ’ everything else is just ok ’);
break ;
}
```

Switch statements have somewhat fallen out of favor in JavaScript, because often the same behavior can be accomplished by creating an object that has more potential for reuse, testing, etc. For example:

```
var stuffToDo = {
’bar ’ : function () {
alert ( ’ the value was bar -- yay ! ’);
} ,
’baz ’ : function () {
alert ( ’ boo baz :( ’);
} ,
’default ’ : function () {
alert ( ’ everything else is just ok ’);
}
};
if ( stuffToDo [ foo ]) {
stuffToDo [ foo ]();
} else {
stuffToDo [ ’ default ’]();
}
```

2.2.4 Loops

A loop is a sequence of instruction s that is continually repeated until a certain condition is reached. Typically, a certain process is done, such as getting an item of data and changing it, and then some condition is checked such as whether a counter has reached a prescribed number. If it hasn’t, the next instruction in the sequence is an instruction to return to the first instruction in the sequence and repeat the sequence. If the condition has been reached, the next instruction “falls through” to the next sequential instruction or branches outside the loop. A loop is a fundamental programming idea that is commonly used in writing programs.

Loops let you run a block of code a certain number of times.

```
// logs ’try 0 ’ , ’try 1 ’ , ... , ’try 4 ’
```

```
for ( var i =0; i <5; i ++) {
console . log ( ' try ' + i);
}
```

The for loop

A for loop is made up of four statements and has the following structure:

```
for ([ initialisation ]; [ conditional ]; [ iteration ])
[ loopBody ]
```

The initialization statement is executed only once, before the loop starts. It gives you an opportunity to prepare or declare any variables.

The conditional statement is executed before each iteration, and its return value decides whether or not the loop is to continue. If the conditional statement evaluates to a falsey value then the loop stops.

The **iteration statement** is executed at the end of each iteration and gives you an opportunity to change the state of important variables. Typically, this will involve incrementing or decrementing a counter and thus bringing the loop ever closer to its end.

The loopBody statement is what runs on every iteration. It can contain anything you want. You'll typically have multiple statements that need to be executed and so will wrap them in a block ({...}).

Here's a typical for loop:

Keyword

Iteration statements cause embedded statements to be executed a number of times, subject to the loop-termination criteria. These statements are executed in order, except when a jump statement is encountered.

A typical for loop

```
for ( var i = 0 , limit = 100; i < limit ; i ++) {
// This block will be executed 100 times
console . log ( ' Currently at ' + i);
// Note : the last log will be " Currently at 99"
}
```

The while loop

A while loop is similar to an if statement, except that its body will keep executing until the condition evaluates to false.

```
while ([ conditional ]) [ loopBody ]
```

Here's a typical while loop:

A typical while loop

```
var i = 0;
while (i < 100) {
// This block will be executed 100 times
console . log ( ' Currently at ' + i);
i ++; // increment i
}
```

You'll notice that we're having to increment the counter within the loop's body. It is possible to combine the conditional and incrementer, like so:

A while loop with a combined conditional and incrementer

```
var i = -1;
while (++ i < 100) {
// This block will be executed 100 times
console . log ( ' Currently at ' + i);
}
```

Notice that we're starting at -1 and using the prefix incrementer (++i).

The do-while loop

This is almost exactly the same as the while loop, except for the fact that the loop's body is executed at least once before the condition is tested.

```
do [ loopBody ] while ([ conditional ])
```

Here's a do-while loop:

A do-while loop

```
do {
// Even though the condition evaluates to false
// this loop 's body will still execute once .
alert ( 'Hi there ! ');
```

} while (false);

These types of loops are quite rare since only few situations require a loop that blindly executes at least once. Regardless, it’s good to be aware of it.

Breaking and continuing

Usually, a loop’s termination will result from the conditional statement not evaluating to true, but it is possible to stop a loop in its tracks from within the loop’s body with the break statement.

Stopping a loop

```
for ( var i = 0; i < 10; i ++) {
          if ( something ) {
 break ;
           }
}
```

You may also want to continue the loop without executing more of the loop’s body. This is done using the continue statement.

Skipping to the next iteration of a loop

```
for (var i = 0; i < 10; i++) {

    if (something) {
        continue;
    }

    // The following statement will only be executed
    // if the conditional ’something’ has not been met
    console.log(’I have been reached’);

}
```

2.2.5 Reserved Words

Reserved words are terms or phrases appropriated for special use that may not be utilized in the creation of variable names. For example, “print” is a reserved word because it is a function in many languages to show text on the screen.

Reserved words are used in **operating systems** as a method of identifying a device file or other service. Below is a listing of Microsoft reserved words in MS-DOS and Windows operating systems. When attempting to use any of the below reserved words as a name of a file, or in a command you may encounter and unusual response. For example, attempting to save a file as CON or CON.txt may generate a reserved file name or access denied error or say the file already exists.

JavaScript has a number of “reserved words,” or words that have special meaning in the language. You should avoid using these words in your code except when using them with their intended meaning.

abstract	boolean	break	byte
case	catch	char	class
const	continue	debugger	default
delete	do	double	else
enum	export	extends	final
finally	float	for	function
goto	if	implements	import
in	instanceof	int	interface
long	native	new	package
private	protected	public	return
short	static	super	switch
synchronized	this	throw	throws
transient	try	typeof	var
void	volatile	while	with

Keyword

Operating system (OS) is system software that manages computer hardware and software resources and provides common services for computer programs.

2.2.6 Arrays

Arrays are zero-indexed lists of values. They are a handy way to store a set of related items of the same type (such as strings), though in reality, an array can include multiple types of items, including other arrays. Objects allow to store keyed collections of values. That's fine. But quite often we find that we need an ordered collection, where we have a 1st, a 2nd, a 3rd element and so on. For example, we need that to store a list of something: users, goods, HTML elements etc. It is not convenient to use an object here, because it provides no methods to manage the order of elements. We can't insert a new property "between" the existing ones. Objects are just not meant for such use.

A simple array

```
var myArray = [ 'hello ' , 'world ' ];
```

Accessing array items by index

```
var myArray = [ 'hello ' , 'world ' , 'foo ' , 'bar ' ];
console . log ( myArray [3]); // logs 'bar '
```

Testing the size of an array

```
var myArray = [ 'hello ' , 'world ' ];
console . log ( myArray . length ); // logs 2
```

Changing the value of an array item

```
var myArray = [ 'hello ' , 'world ' ];
myArray [1] = 'changed ';
```

While it’s possible to change the value of an array item as shown in “Changing the value of an array item”, it’s generally not advised.

Adding elements to an array

```
var myArray = [ ’hello ’ , ’world ’ ];
myArray . push ( ’new ’);
```

Working with arrays

```
var myArray = [ ’h’ , ’e’ , ’l’ , ’l’ , ’o’ ];
var myString = myArray . join ( ’ ’); // ’hello ’
var mySplit = myString . split ( ’ ’); // [ ’h’ , ’e’ , ’l’ , ’l’ , ’o’ ]
```

2.2.7 Objects

Objects contain one or more key-value pairs. The key portion can be any string. The value portion can be any type of value: a number, a string, an array, a function, or even another object. [Definition: When one of these values is a function, it’s called a method of the object.] Otherwise, they are called properties. As it turns out, nearly everything in JavaScript is an object — arrays, functions, numbers, even strings — and they all have properties and methods.

Creating an “object literal”

```
var myObject = {
sayHello : function () {
console . log ( ’hello ’);
} ,
myName : ’Rebecca ’
};
myObject . sayHello (); // logs ’hello ’
console . log ( myObject . myName ); // logs ’Rebecca ’
```

When creating object literals, you should note that the key portion of each key-value pair can be written as any valid JavaScript identifier, a string (wrapped in quotes) or a number:

```
var myObject = {
validIdentifier : 123 ,
’some string ’: 456 ,
99999: 789
};
```

2.2.8 Testing Type

JavaScript offers a way to test the "type" of a variable. However, the result can be confusing — for example, the type of an Array is "object". It's common practice to use the typeof operator when trying to determining the type of a specific value.

```
Testing the type of various variables
var myFunction = function () {
console . log ( 'hello ');
};
var myObject = {
foo : 'bar '
};
var myArray = [ 'a' , 'b' , 'c' ];
var myString = 'hello ';
var myNumber = 3;
typeof myFunction ; // returns 'function '
typeof myObject ; // returns 'object '
typeof myArray ; // returns 'object ' -- careful !
typeof myString ; // returns 'string ';
typeof myNumber ; // returns 'number '
typeof null ; // returns 'object ' -- careful !
if ( myArray . push && myArray . slice && myArray . join ) {
// probably an array
// ( this is called " duck typing ")
}
if ( Object . prototype . toString . call ( myArray ) === '[ object Array ] ') {
// Definitely an array !
// This is widely considered as the most robust way
// to determine if a specific value is an Array .
}
```

jQuery offers utility methods to help you determine the type of an arbitrary value.

2.2.9 The this keyword

In JavaScript, as in most object-oriented programming languages, this is a special keyword that is used within methods to refer to the object on which a method is being invoked. The value of this is determined using a simple series of steps:

1. If the function is invoked using Function.call or Function.apply, this will be set to the first argument passed to call/apply. If the first argument passed to call/apply is null or undefined, this will refer to the global object (which is the window object in **Web browsers**).
2. If the function being invoked was created using Function.bind, this will be the first argument that was passed to bind at the time the function was created.
3. If the function is being invoked as a method of an object, this will refer to that object.
4. Otherwise, the function is being invoked as a standalone function not attached to any object, and this will refer to the global object.

Keyword

Web browser is a software application for accessing information on the World Wide Web. Each individual web page, image, and video is identified by a distinct URL, enabling browsers to retrieve and display them on the user's device.

A function invoked using Function.call

```
var myObject = {
sayHello : function () {
console . log ( 'Hi! My name is ' + this . myName );
} ,
myName : 'Rebecca '
};
var secondObject = {
myName : 'Colin '
};
myObject . sayHello (); // logs 'Hi! My name is Rebecca '
myObject . sayHello . call ( secondObject ); // logs 'Hi!
My name is Colin '
```

A function created using Function.bind

```
var myName = 'the global object ' ,
sayHello = function () {
```

```
console . log ( 'Hi! My name is ' + this . myName );
} ,
myObject = {
myName : 'Rebecca '
};
var myObjectHello = sayHello . bind ( myObject );
sayHello (); // logs 'Hi! My name is the global object '
myObjectHello (); // logs 'Hi! My name is Rebecca '
```

A function being attached to an object at runtime

```
var myName = 'the global object ' ,
sayHello = function () {
console . log ( 'Hi! My name is ' + this . myName );
} ,
myObject = {
myName : 'Rebecca '
} ,
secondObject = {
myName : 'Colin '
};
myObject . sayHello = sayHello ;
secondObject . sayHello = sayHello ;
sayHello (); // logs 'Hi! My name is the global object '
myObject . sayHello (); // logs 'Hi! My name is Rebecca '
secondObject . sayHello (); // logs 'Hi! My name is Colin '
```

Remember

When invoking a function deep within a long namespace, it is often tempting to reduce the amount of code you need to type by storing a reference to the actual function as a single, shorter variable.

It is important not to do this with instance methods as this will cause the value of this within the function to change, leading to incorrect code operation. For instance:

```
var myNamespace = {
myObject : {
sayHello : function () {
console . log ( 'Hi! My name is ' + this . myName );
} ,
myName : 'Rebecca '
}
};
var hello = myNamespace . myObject . sayHello ;
hello (); // logs 'Hi! My name is undefined '
```

You can, however, safely reduce everything up to the object on which the method is invoked:

```
var myNamespace = {
myObject : {
sayHello : function () {
console . log ( 'Hi! My name is ' + this . myName );
} ,
myName : 'Rebecca '
}
};
var obj = myNamespace . myObject ;
obj . sayHello (); // logs 'Hi! My name is Rebecca '
```

2.2.10 Scope

“Scope” refers to the variables that are available to a piece of code at a given time. A lack of understanding of scope can lead to frustrating debugging experiences.

When a variable is declared inside of a function using the var keyword, it is only available to code inside of that function — code outside of that function cannot access the variable. On the other hand, functions defined inside that function will have access to to the declared variable.

Furthermore, variables that are declared inside a function without the var keyword are not local to the function — JavaScript will traverse the scope chain all the way up to the window scope to find where the variable was previously defined. If the variable wasn’t previously defined, it will be defined in the global scope, which can have extremely unexpected consequences;

Functions have access to variables defined in the same scope

```
var foo = 'hello ';
var sayHello = function () {
console . log ( foo );
};
```

```
sayHello (); // logs 'hello '
console . log ( foo ); // also logs 'hello '
```

Code outside the scope in which a variable was defined does not have access to the variable

```
var sayHello = function () {
var foo = 'hello ';
console . log ( foo );
};
sayHello (); // logs 'hello '
console . log ( foo ); // doesn 't log anything
```

Variables with the same name can exist in different scopes with different values

```
var foo = 'world ';
var sayHello = function () {
var foo = 'hello ';
console . log ( foo );
};
sayHello (); // logs 'hello '
console . log ( foo ); // logs 'world '
```

Functions can "see" changes in variable values after the function is defined

```
var myFunction = function () {
var foo = 'hello ';
var myFn = function () {
console . log ( foo );
};
foo = 'world ';
return myFn ;
};
var f = myFunction ();
f (); // logs 'world ' -- uh oh
```

Scope insanity

```
// a self - executing anonymous function
( function () {
```

```
var baz = 1;
var bim = function () { alert ( baz ); };
bar = function () { alert ( baz ); };
})();
console . log ( baz ); // baz is not defined outside of the function
bar (); // bar is defined outside of the anonymous function
// because it wasn 't declared with var ; furthermore ,
// because it was defined in the same scope as baz ,
// it has access to baz even though other code
// outside of the function does not
bim (); // bim is not defined outside of the anonymous function ,
// so this will result in an error
```

Closures

Closures are an extension of the concept of scope — functions have access to variables that were available in the scope where the function was created. If that's confusing, don't worry: closures are generally best understood by example.

In Functions can see changes in variable values after the function is defined", we saw how functions have access to changing variable values. The same sort of behavior exists with functions defined within loops — the function "sees" the change in the variable's value even after the function is defined, resulting in all clicks alerting 5.

How to lock in the value of i?

```
/* this won 't behave as we want it to; */
/* every click will alert 5 */
for ( var i =0; i <5; i ++) {
$( ' <p> click me </p > '). appendTo ( 'body '). click ( function () {
alert (i);
});
}
```

Locking in the value of i with a closure

```
/* fix : 'close ' the value of i inside
createFunction , so it won 't change */
var createFunction = function (i) {
return function () { alert (i); };
};
for ( var i =0; i <5; i ++) {
$( ' <p> click me </p > '). appendTo ( 'body '). click ( createFunction (i ));
}
```

Closures can also be used to resolve issues with the this keyword, which is unique to each scope:

Using a closure to access inner and outer object instances simultaneously

```
var outerObj = {
myName : 'outer ' ,
outerFunction : function () {
// provide a reference to outerObj
// through innerFunction 's closure
var self = this ;
var innerObj = {
myName : 'inner ' ,
innerFunction : function () {
// logs 'outer inner '
console . log ( self . myName , this . myName );
}
};
innerObj . innerFunction ();
console . log ( this . myName ); // logs 'outer '
}
};
outerObj . outerFunction ();
```

This mechanism can be particularly useful when dealing with callbacks, though in those cases, it is often better to use Function.bind, which will avoid any overhead associated with scope traversal.

Did You Know?

Since 1996, the IIS web-server has supported Microsoft's implementation of server-side Javascript -- JScript—in ASP and .NET pages.

SUMMARY

- The syntax of a computer language is the set of rules that defines the combinations of symbols that are considered to be correctly structured statements or expressions in that language. This applies both to programming languages, where the document represents source code, and to markup languages, where the document represents data.
- The syntax of a language defines its surface form. Text-based computer languages are based on sequences of characters, while visual programming languages are based on the spatial layout and connections between symbols (which may be textual or graphical).
- JavaScript is a case-sensitive language. This means that the language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.
- The modern JavaScript is a "safe" programming language. It does not provide low-level access to memory or CPU, because it was initially created for browsers which do not require it.
- JavaScript's abilities in the browser are limited for the sake of the user's safety. The aim is to prevent an evil webpage from accessing private information or harming the user's data.
- The name of a variable, function, or property is known as an identifier in JavaScript. Identifiers consist of letters and numbers, but they cannot include any symbol outside of $ and _, and cannot begin with a number.
- JavaScript is a rich and expressive language in its own right. While it will be of particular value to people with no programming experience, even people who have used other programming languages may benefit from learning about some of the peculiarities of JavaScript.
- JavaScript operators are used to assign values, compare values, perform arithmetic operations, and more.
- Arrays are zero-indexed lists of values. They are a handy way to store a set of related items of the same type (such as strings), though in reality, an array can include multiple types of items, including other arrays.
- "Scope" refers to the variables that are available to a piece of code at a given time. A lack of understanding of scope can lead to frustrating debugging experiences.

KNOWLEDGE CHECK

1. **JavaScript Code is written inside file having extension __________.**
   a. .jvs
   b. .javascript
   c. .js
   d. .jsc
2. **Why JavaScript is called as Lightweight Programming Language ?**
   a. because JS is available free of cost.
   b. because JS is client side scripting
   c. because we can add programming functionality inside JS
   d. because JS can provide programming functionality inside but up to certain extend.
3. **Choose appropriate Option(s) : JavaScript is also called as _____________.**
   a. None of These
   b. Server Side Scripting Language
   c. Client Side Scripting Language
   d. Browser Side Scripting Language
4. **Local Browser used for validations on the Web Pages uses __________.**
   a. Java
   b. CSS
   c. HTML
   d. JS
5. **JavaScript Code can be called by using _________.**
   a. Function / Method
   b. RMI
   c. Triggering Event
   d. Preprocessor
6. **Which of the following is the correct syntax to print a page using JavaScript?**
   a. print();
   b. print();
   c. print();
   d. print();

7. Which of the following syntax is correct to refer to an external script called "LFC.js"?

a. <script source="LFC.js">

b. <script ref="LFC.js">

c. <script src="LFC.js">

d. <script type="LFC.js">

8. Which of the following syntax can be used to write "Hello World" in an alert box?

a. alertBox("Hello World");

b. msgBox("Hello World");

c. alert("Hello World");

d. msg("Hello World");

REVIEW QUESTIONS

1. What can in-browser JavaScript do?
2. Define the syntax and code structure in JavaScript.
3. Explain the various types of operators.
4. Write a program on conditional code.
5. Discuss on the JavaScript loops.

Check Your Result

1. (c)	2. (d)	3. (c)	4. (d)	5. (c)
6. (b)	7. (c)	8. (c)		

REFERENCES

1. Douglas Crockford. Javascript: The world's most misunderstood programming language. http://javascript.crockford.com/javascript.html, 2001. Accessed May 24, 2008.
2. Douglas Crockford. JavaScript: The world's most misunderstood programming language has become the world's most popular programming language. http://javascript.crockford.com/popular.html, 2008. Accessed May 24, 2008.
3. http://www.damits.ac.in/library_doc/javascript_tutorial.pdf
4. https://autotelicum.github.io/Smooth-CoffeeScript/literate/js-intro.pdf
5. https://javascript.info/array
6. https://javascript.info/intro#what-can-in-browser-javascript-do
7. https://whatis.techtarget.com/definition/loop
8. https://www.computerhope.com/jargon/r/reseword.htm
9. https://www.digitalocean.com/community/tutorials/understanding-syntax-and-code-structure-in-javascript
10. https://www.techopedia.com/definition/3959/syntax
11. https://www.tutorialspoint.com/javascript/javascript_ifelse.htm
12. https://www.w3schools.com/jsref/jsref_operators.asp
13. Maximiliano Firtman. jQuery Mobile: Up and Running. O'Reilly. 2012. ISBN 1-4493-9765-4.
14. Tim Berners-Lee, Roy Fielding, and Larry Masinter. Uniform resource identifier (URI): Generic syntax. http://tools.ietf.org/html/rfc3986, January 2005. Accessed on July 7, 2010.

CHAPTER 3

BUILT IN FUNCTIONS

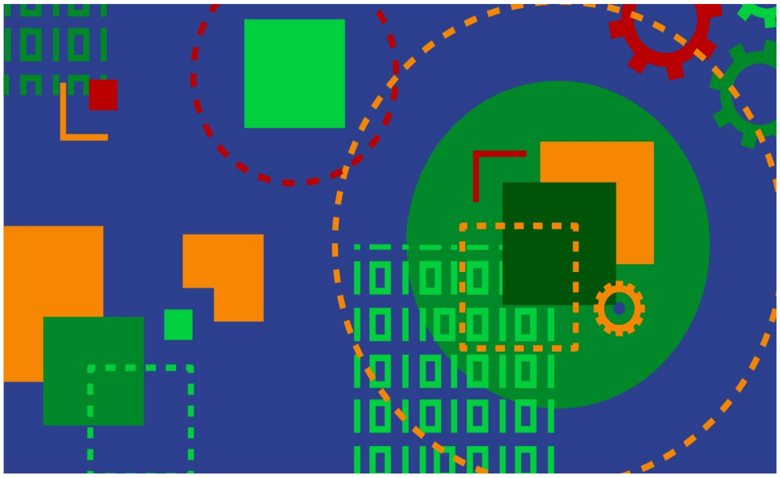
Node.js is a tumor on the programming community, in that not only is it completely braindead, but the people who use it go on to infect other people who cannot think for themselves.

– Ted Dziuba

LEARNING OBJECTIVES

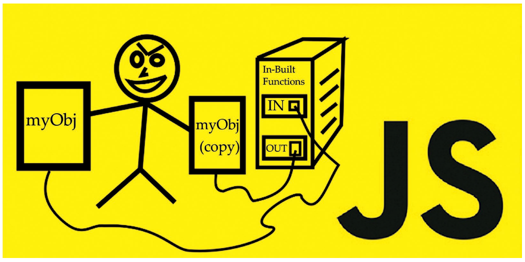
After studying this chapter, you will be able to:

1. Understand functions in JavaScript
2. Discuss about JavaScript built-in functions
3. Understand date and time in JavaScriptt



INTRODUCTION

A function is a block of code that performs an action or returns a value. Functions are custom code defined by programmers that are reusable, and can therefore make your programs more modular and efficient. JavaScript has several "top-level" built-in functions. JavaScript also has four built-in objects: Array, Date, Math, and String. Each object has special-purpose properties and methods associated with it. JavaScript also has constructors for Boolean and Number types.



In JavaScript, functions are objects. You can work with functions as if they were objects. For example, you can assign functions to variables, to array elements, and to other objects. They can also be passed around as arguments to other functions or be returned from those functions.

3.1 UNDERSTAND FUNCTIONS IN JAVASCRIPT

A function (also known as a method) is a self-contained piece of code that performs a particular "function", then returns a value. You can recognize a function by its format — it is a piece of descriptive text, followed by open and close brackets.

Like this:

```
function displayWelcomeMessage() {
    ...code goes here...
}
```

The code that goes into a function can be as simple or as complex as you need it to be. Regardless of the code's complexity inside the function, when you call the function, you do not need to know anything about the code inside.

All you need to know is the name of the function, and any arguments that you might need to supply. For example, take JavaScript's built-in alert() function. You can call that function from within your code without knowing how the function is written.

All you need to know is what the function *does* and how to call it. In this section, we will learn several ways to define a function, call a function, and use function parameters in JavaScript.

3.1.1 Defining a Function

Functions are defined, or declared, with the function keyword. .Below is the syntax for a function in JavaScript

```
function nameOfFunction() {
    // Code to be executed
}
```

The declaration begins with the function keyword, followed by the name of the function. Function names follow the same rules as variables — they can contain letters, numbers, underscores and dollar signs, and are frequently written in camel case. The name is followed by a set of **parentheses**, which can be used for optional parameters. The code of the function is contained in curly brackets, just like a for statement or an if statement.

In our first example, we will make a function declaration to print a greeting statement to the console.

```
// Initialize greeting function
function greet() {
    console.log(“Hello, World!”);
}
```

Here we have the code to print Hello, World! to the console contained inside the greet() function. However, nothing will happen and no code will execute until we invoke, or call the function. You can invoke a function by writing the name of the function followed by the parentheses.

```
// Invoke the function
greet();
```

Now we will put those together, defining our function and invoking it.

```
greet.js
// Initialize greeting function
function greet() {
    console.log(“Hello, World!”);
}
// Invoke the function
greet();
```

Keyword

Parentheses allow a writer to provide additional information. The parenthetical material might be a single word, a fragment, or multiple complete sentences.

Did You Know?

In 1995, Netscape Communications recruited Brendan Eich with the goal of embedding the Scheme programming language into its Netscape Navigator. Before he could get started, Netscape Communications collaborated with Sun Microsystems to include in Netscape Navigator Sun’s more static programming language Java, in order to compete with Microsoft for user adoption of Web technologies and platforms.

With the call for greet();, the function will run and we will receive the Hello, World! as the program's output.

```
Output
Hello, World!
```

Now we have our greet() code contained in a function, and can reuse it as many times as we want. Using parameters, we can make the code more dynamic.

3.1.2 Function Parameters

JavaScript is a functional language meaning that functions are the primary modular units of execution. Functions are obviously very important in JavaScript. When talking about functions, the terms parameters and arguments are often interchangeably used as if it were one and the same thing but there is a very subtle difference.

- Parameters are variables listed as a part of the function definition.
- Arguments are values passed to the function when it is invoked.

In our greet.js file, we created a basic function that prints Hello, World to the console. Using parameters, we can add additional functionality that will make the code more flexible. Parameters are input that get passed into functions as names and behave as local variables.

When a user logs in to an application, we may want the program to greet them by name, instead of just saying, "Hello, World!". We will add a parameter into our function, called name, to represent the name of the person being greeted.

```
// Initialize custom greeting function
function greet(name) {
    console.log(`Hello, ${name}!`);
}
```

The name of the function is greet, and now we have a single parameter inside the parentheses. The name of the parameter follows the same rules as naming a variable. Inside of the function, instead of a static string consisting of Hello, World, we have a template literal string containing our parameter, which is now behaving as a local variable.

You will notice we have not defined our name parameter anywhere. We assign it a value when we invoke our function. Assuming our user is named Sammy, we will call the function and place the username as the argument. The argument is the actual value that gets passed into the function, in this case it is the string "Sammy".

```
// Invoke greet function with "Sammy" as the argument
greet("Sammy");
```

The value of "Sammy" is being passed into the function through the name parameter. Now every time name is used throughout the function, it will represent the "Sammy" value. Here is the whole code.

```
greetSammy.js
// Initialize custom greeting function
function greet(name) {
    console.log(`Hello, ${name}!`);
}
// Invoke greet function with "Sammy" as the argument
greet("Sammy");
```

When we run the program above, we will receive the following output.

```
Output
Hello, Sammy!
```

Now we have an example of how a function can be reused. In a real world example, the function would pull the username from a database instead of directly supplying the name as an argument value.

In addition to parameters, variables can be declared inside of functions. These variables are known as local variables, and will only exist inside the *scope* of their own function block. Variable scope determines variables' accessibility; variables that are defined inside of a function are not accessible from outside of the function, but they can be used as many times as their function is used throughout a program.

3.1.3 Returning Values

More than one parameter can be used in a function. We can pass multiple values into a function and return a value. We will create a function to find the sum of two values, represented by x and y.

```
sum.js
// Initialize add function
function add(x, y) {
    return x + y;
}

// Invoke function to find the sum
add(9, 7);
```

In the program above, we defined a function with the parameters x and y, and then passed the values of 9 and 7 to the function. When we run the program, we will receive the sum of those numbers as the output.

Output

```
16
```

In this case, with 9 and 7 passed to the sum() function, the program returned 16.

When the return keyword is used, the function ceases to execute and the value of the expression is returned. Although in this case the browser will display the value in the console, it is not the same as using console.log() to print to the console. Invoking the function will output the value exactly where the function was invoked. This value can be used immediately or placed into a variable.

3.1.4 Function Expressions

In the last section, we used a function declaration to get the sum of two numbers and return that value. We can also create a function expression by assigning a function to a variable. Using our same add function example, we can directly apply the returned value to a variable, in this case sum.

functionExpression.js

```
// Assign add function to sum constant
const sum = function add(x, y) {
    return x + y;
}

// Invoke function to find the sum
sum(20, 5);
```

Output

```
25
```

Now the sum constant is a function. We can make this expression more concise by turning it into an anonymous function, which is an unnamed function. Currently, our function has the name add, but with function expressions it is not necessary to name the function and the name is usually omitted.

anonymousExpression.js

```
// Assign function to sum constant
const sum = function(x, y) {
    return x + y;
}
// Invoke function to find the sum
sum(100, 3);
Output
103
```

In this example, we have removed the name of the function, which was add, and turned it into an anonymous function. A named function expression could be used to aid in debugging, but it is usually omitted.

3.1.5 Arrow Functions

So far, we have gone through how to define functions using the function keyword. However, there is a newer, more concise method of defining a function known as arrow function expressions as of ECMAScript 6. Arrow functions, as they are commonly known, are represented by an equals sign followed by a greater than sign: =>. Arrow functions are always anonymous functions and a type of function expression. We can create a basic example to find the product of two numbers.

```
arrowFunction.js
// Define multiply function
const multiply = (x, y) => {
    return x * y;
}

// Invoke function to find product
multiply(30, 4);
Output
120
```

Instead of writing out the keyword function, we use the => arrow to indicate a function. Otherwise, it works similarly to a regular function expression. In the case of only one parameter, the parentheses can be excluded. In this example, we are squaring x, which only requires one number to be passed as an argument. The parentheses have been omitted.

```
// Define square function
const square = x => {
    return x * x;
}
```

```
// Invoke function to find product
square(8);
Output
64
```

With these particular examples that only consist of a return statement, arrow functions allow the syntax to be reduced even further. If the function is only a single line return, both the curly brackets and the return statement can be omitted, .as seen in the example below

```
// Define square function
const square = x => x * x;
// Invoke function to find product
square(10);
Output
100
```

Remember

In the case of no parameters, an empty set of parentheses () is required in the arrow functions.

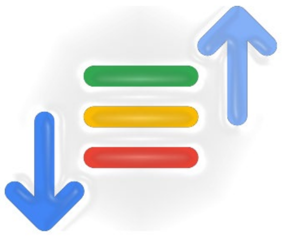
All three of these types of **syntax** result in the same output. It is generally a matter of preference or company coding standards to decide how you will structure your own functions. In this section, we covered function declarations and function expressions, returning values from functions, assigning function values to variables.

Keyword

Syntax is the set of rules, principles, and processes that govern the structure of sentences in a given language, usually including word order.

3.2 JAVASCRIPT BUILT-IN FUNCTIONS

JavaScript, first introduced by Netscape, changed the static fate of the HTML web pages. Before JavaScript came into existence, HTML pages were just used to render static content. Inserting JavaScript to a web page, can give it a significant degree of interactivity and functionality. It lets you control the behavior of the web browser and how the elements of the web page are displayed. Most web browsers have a built in JavaScript interpreter. When a browser downloads an html file that contains JavaScript, the JavaScript interpreter reacts to any script.



JavaScript is an object oriented programming language. It supports the concept of objects in the form of attributes. If an object attribute consists of function, then it is a method of that object, or if an object attribute consists of values, then it is a property of that object.

Some browsers encounter problems using the example from the local drive. Chrome displays a blank page when you access Customers2.XML from the local drive. To test this technique in a way that works for most browsers, copy the files to your web server and then access the XML file from the web server.

3.2.1 Number Methods

The Number object contains only the default methods that are part of every object's definition. The Boolean object corresponds to the number primitive type.

Sr. No	Method & Description
1	**constructor()** Returns the function that created this object's instance. By default, this is the Number object.
2	**toExponential()** Forces a number to display in exponential notation, even if the number is in the range in which JavaScript normally uses standard notation.
3	**toFixed()** Formats a number with a specific number of digits to the right of the decimal.
4	**toLocaleString()** Returns a string value version of the current number in a format that may vary according to a browser's locale settings.

5	**toPrecision()** Defines how many total digits (including digits to the left and right of the decimal) to display of a number.
6	**toString()** Returns the string representation of the number's value.
7	**valueOf()** Returns the number's value.

3.2.2 Boolean Methods

The Boolean object represents a primitive Boolean value. Here is a list of each method and its description.

Sr. No	**Method & Description**
1	**toSource()** Returns a string containing the source of the Boolean object; you can use this string to create an equivalent object.
2	**toString()** Returns a string of either "true" or "false" depending upon the value of the object.
3	**valueOf()** Returns the primitive value of the Boolean object.

3.2.3 String Methods

Here is a list of each method and its description.

Sr. No	**Method & Description**
1	**charAt()** Returns the character at the specified index.
2	**charCodeAt()** Returns a number indicating the Unicode value of the character at the given index.

3	**concat()** Combines the text of two strings and returns a new string.
4	**indexOf()** Returns the index within the calling String object of the first occurrence of the specified value, or -1 if not found.
5	**lastIndexOf()** Returns the index within the calling String object of the last occurrence of the specified value, or -1 if not found.
6	**localeCompare()** Returns a number indicating whether a reference string comes before or after or is the same as the given string in sort order.
7	**length()** Returns the length of the string.
8	**match()** Used to match a regular expression against a string.
9	**replace()** Used to find a match between a regular expression and a string, and to replace the matched substring with a new substring.
10	**search()** Executes the search for a match between a regular expression and a specified string.
11	**slice()** Extracts a section of a string and returns a new string.
12	**split()** Splits a String object into an array of strings by separating the string into substrings.
13	**substr()** Returns the characters in a string beginning at the specified location through the specified number of characters.
14	**substring()** Returns the characters in a string between two indexes into the string.
15	**toLocaleLowerCase()** The characters within a string are converted to lower case while respecting the current locale.

16	**toLocaleUpperCase()** The characters within a string are converted to upper case while respecting the current locale.
17	**toLowerCase()** Returns the calling string value converted to lower case.
18	**toString()** Returns a string representing the specified object.
19	**toUpperCase()** Returns the calling string value converted to uppercase.
20	**valueOf()** Returns the primitive value of the specified object.

3.2.4 String HTML wrappers

A **String** is an object representing a series of characters. Here is a list of each method which returns a copy of the string wrapped inside the appropriate HTML tag.

Sr. No	Method & Description
1	**anchor()** Creates an HTML anchor that is used as a hypertext target.
2	**big()** Creates a string to be displayed in a big font as if it were in a <big> tag.
3	**blink()** Creates a string to blink as if it were in a <blink> tag.
4	**bold()** Creates a string to be displayed as bold as if it were in a <b> tag.
5	**fixed()** Causes a string to be displayed in fixed-pitch font as if it were in a <tt> tag
6	**fontcolor()** Causes a string to be displayed in the specified color as if it were in a <font color=”color”> tag.

7	**fontsize()** Causes a string to be displayed in the specified font size as if it were in a <font size="size"> tag.
8	**italics()** Causes a string to be italic, as if it were in an <i> tag.
9	**link()** Creates an HTML hypertext link that requests another URL.
10	**small()** Causes a string to be displayed in a small font, as if it were in a <small> tag.
11	**strike()** Causes a string to be displayed as struck-out text, as if it were in a <strike> tag.
12	**sub()** Causes a string to be displayed as a subscript, as if it were in a <sub> tag
13	**sup()** Causes a string to be displayed as a superscript, as if it were in a <sup> tag

3.2.5 Array Methods

JavaScript arrays are a special kind of object, and are created dynamically. An array object contains a number of variables. The number of variables may be zero, in which case the array is said to be empty. The variables contained in an array have no names; instead they are referenced by array access expressions that use nonnegative integer index values. These variables are called the *components* of the array. If an array has *n* components, we say *n* is the *length* of the array; the components of the array are referenced using integer indices from 0 to *n*-1, inclusive.

Unlike Java, the components of an array do not necessarily have the same type. An array component can itself be an array, to create essentially multi-dimensional arrays. If, starting from any array type, one considers its component type, and then (if that is also an array type) the component type of that type, and so on, eventually one must reach a component type that is not an array type; the components at this level of the data structure are called the *elements* of the original array.

Here is a list of each method and its description.

Sr. No	Method & Description
1	**concat()** Returns a new array comprised of this array joined with other array(s) and/or value(s).
2	**every()** Returns true if every element in this array satisfies the provided testing function.
3	**filter()** Creates a new array with all of the elements of this array for which the provided filtering function returns true.
4	**forEach()** Calls a function for each element in the array.
5	**indexOf()** Returns the first (least) index of an element within the array equal to the specified value, or -1 if none is found.
6	**join()** Joins all elements of an array into a string.
7	**lastIndexOf()** Returns the last (greatest) index of an element within the array equal to the specified value, or -1 if none is found.
8	**map()** Creates a new array with the results of calling a provided function on every element in this array.
9	**pop()** Removes the last element from an array and returns that element.
10	**push()** Adds one or more elements to the end of an array and returns the new length of the array.
11	**reduce()** Apply a function simultaneously against two values of the array (from left-to-right) as to reduce it to a single value.
12	**reduceRight()** Apply a function simultaneously against two values of the array (from right-to-left) as to reduce it to a single value.

13	**reverse()** Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first.
14	**shift()** Removes the first element from an array and returns that element.
15	**slice()** Extracts a section of an array and returns a new array.
16	**some()** Returns true if at least one element in this array satisfies the provided testing function.
17	**toSource()** Represents the source code of an object
18	**sort()** Sorts the elements of an array.
19	**splice()** Adds and/or removes elements from an array.
20	**toString()** Returns a string representing the array and its elements.
21	**unshift()** Adds one or more elements to the front of an array and returns the new length of the array.

3.2.6 Date Methods

The Date object provides a system-independent **abstraction** of dates and times. Dates may be constructed from a year, month, day of the month, hour, minute, and second, and those six components, as well as the day of the week, may be extracted from a date. Dates may also be compared and converted to a readable string form. A Date is represented to a precision of one millisecond.The way JavaScript handles dates is very similar to the way Java handles dates: both languages have many of the same date methods, and both store dates internally as the number of milliseconds since January 1, 1970 00:00:00. Dates prior to 1970 are not allowed.

Keyword

Abstraction is a fundamental concept to computer science and software development. The process of abstraction can also be referred to as modeling and is closely related to the concepts of theory and design.

Here is a list of each method and its description.

Sr. No	Method & Description
1	**Date()** Returns today's date and time
2	**getDate()** Returns the day of the month for the specified date according to local time.
3	**getDay()** Returns the day of the week for the specified date according to local time.
4	**getFullYear()** Returns the year of the specified date according to local time.
5	**getHours()** Returns the hour in the specified date according to local time.
6	**getMilliseconds()** Returns the milliseconds in the specified date according to local time.
7	**getMinutes()** Returns the minutes in the specified date according to local time.
8	**getMonth()** Returns the month in the specified date according to local time.
9	**getSeconds()** Returns the seconds in the specified date according to local time.
10	**getTime()** Returns the numeric value of the specified date as the number of milliseconds since January 1, 1970, 00:00:00 UTC.
11	**getTimezoneOffset()** Returns the time-zone offset in minutes for the current locale.

12	**getUTCDate()** Returns the day (date) of the month in the specified date according to universal time.
13	**getUTCDay()** Returns the day of the week in the specified date according to universal time.
14	**getUTCFullYear()** Returns the year in the specified date according to universal time.
15	**getUTCHours()** Returns the hours in the specified date according to universal time.
16	**getUTCMilliseconds()** Returns the milliseconds in the specified date according to universal time.
17	**getUTCMinutes()** Returns the minutes in the specified date according to universal time.
18	**getUTCMonth()** Returns the month in the specified date according to universal time.
19	**getUTCSeconds()** Returns the seconds in the specified date according to universal time.
20	**getYear()** **Deprecated** - Returns the year in the specified date according to local time. Use getFullYear instead.
21	**setDate()** Sets the day of the month for a specified date according to local time.
22	**setFullYear()** Sets the full year for a specified date according to local time.
23	**setHours()** Sets the hours for a specified date according to local time.

24	**setMilliseconds()** Sets the milliseconds for a specified date according to local time.
25	**setMinutes()** Sets the minutes for a specified date according to local time.
26	**setMonth()** Sets the month for a specified date according to local time.
27	**setSeconds()** Sets the seconds for a specified date according to local time.
28	**setTime()** Sets the Date object to the time represented by a number of milliseconds since January 1, 1970, 00:00:00 UTC.
29	**setUTCDate()** Sets the day of the month for a specified date according to universal time.
30	**setUTCFullYear()** Sets the full year for a specified date according to universal time.
31	**setUTCHours()** Sets the hour for a specified date according to universal time.
32	**setUTCMilliseconds()** Sets the milliseconds for a specified date according to universal time.
33	**setUTCMinutes()** Sets the minutes for a specified date according to universal time.
34	**setUTCMonth()** Sets the month for a specified date according to universal time.
35	**setUTCSeconds()** Sets the seconds for a specified date according to universal time.

36	**setYear()** **Deprecated -** Sets the year for a specified date according to local time. Use setFullYear instead.
37	**toDateString()** Returns the "date" portion of the Date as a human-readable string.
38	**toGMTString()** **Deprecated -** Converts a date to a string, using the Internet GMT conventions. Use toUTCString instead.
39	**toLocaleDateString()** Returns the "date" portion of the Date as a string, using the current locale's conventions.
40	**toLocaleFormat()** Converts a date to a string, using a format string.
41	**toLocaleString()** Converts a date to a string, using the current locale's conventions.
42	**toLocaleTimeString()** Returns the "time" portion of the Date as a string, using the current locale's conventions.
43	**toSource()** Returns a string representing the source for an equivalent Date object; you can use this value to create a new object.
44	**toString()** Returns a string representing the specified Date object.
45	**toTimeString()** Returns the "time" portion of the Date as a human-readable string.
46	**toUTCString()** Converts a date to a string, using the universal time convention.
47	**valueOf()** Returns the primitive value of a Date object.

3.2.7 Date Static Methods

In addition to the many instance methods listed previously, the Date object also defines two static methods. These methods are invoked through the Date( ) constructor itself.

Sr. No	Method & Description
1	**Date.parse( )** Parses a string representation of a date and time and returns the internal millisecond representation of that date.
2	**Date.UTC( )** Returns the millisecond representation of the specified UTC date and time.

3.2.8 Math Methods

The built-in *Math* object has properties and methods for mathematical constants and functions, respectively. Here is a list of each method and its description.

Sr. No	Method & Description
1	**abs()** Returns the absolute value of a number.
2	**acos()** Returns the arccosine (in radians) of a number.
3	**asin()** Returns the arcsine (in radians) of a number.
4	**atan()** Returns the arctangent (in radians) of a number.
5	**atan2()** Returns the arctangent of the quotient of its arguments.
6	**ceil()** Returns the smallest integer greater than or equal to a number.

7	**cos()** Returns the cosine of a number.
8	**exp()** Returns E^N, where N is the argument, and E is Euler's constant, the base of the natural logarithm.
9	**floor()** Returns the largest integer less than or equal to a number.
10	**log()** Returns the natural logarithm (base E) of a number.
11	**max()** Returns the largest of zero or more numbers.
12	**min()** Returns the smallest of zero or more numbers.
13	**pow()** Returns base to the exponent power, that is, base exponent.
14	**random()** Returns a pseudo-random number between 0 and 1.
15	**round()** Returns the value of a number rounded to the nearest integer.
16	**sin()** Returns the sine of a number.
17	**sqrt()** Returns the square root of a number.
18	**tan()** Returns the tangent of a number.
19	**toSource()** Returns the string "Math".

3.2.9 RegExp Methods

Here is a list of each method and its description.

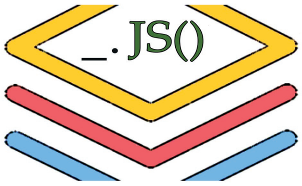
Sr. No	Method & Description
1	**exec()** Executes a search for a match in its string parameter.
2	**test()** Tests for a match in its string parameter.
3	**toSource()** Returns an object literal representing the specified object; you can use this value to create a new object.
4	**toString()** Returns a string representing the specified object.

3.2.10 How to use JavaScript's built-in functions to program with HTML

Remember

Simply displaying data with locale in mind does not perform any data conversion. For example, the strings you create will not suddenly appear in French if you natively speak English. There's nothing magic about locale-specific methods. All that these methods do is change the presentation of the data as you provide it.

JavaScript gives you access a number of built-in functions, including the prompt() function, which lets you ask the user for written input. As with the confirm() function, you provide a text prompt to ask the user to provide a value. On return, you set the output of the function equal to the prompt() function and use the data in your application.



Methods help you perform specific tasks with certain types of **data**. Using these methods makes it easier for you to create robust applications. Of all the methods provided by objects, these methods are the most common and likely the most important for many situations:

- length(): Returns the number of something. For example, when working with a string, length() returns the number of characters in the string. Likewise, when working with an array, length() returns the number of elements in the array. This method also appears as a property in some cases.

- toLocaleString(): Outputs the value of an object as a locale-specific string. For example, when the locale uses a comma for the decimal point, the viewer will see a comma, rather than a period, even if you use a period in your code. It is essential that you provide this support for people from other countries that visit your site.
- toString(): Outputs the value of the object as a string. This method is often used for display purposes.
- valueOf(): Returns a native version of the object value. You need this method in situations where an object could cause problems. For example, when saving data to disk, you want the value, not the object, stored.

Keyword

Data is measured, collected and reported, and analyzed, whereupon it can be visualized using graphs, images or other analysis tools.

JavaScript also includes the concept of global functions. These functions are available without regard to any object from any place you use JavaScript on a page. The following list provides an overview of the most common global functions:

- decodeURI(): Decodes a Uniform Resource Identifier (URI).
- decodeURIComponent(): Decodes a URI component, rather than the entire URI.
- URIs normally have between three or five standard components:
  - *Protocol:* The set of transport rules used to access the resource, such as HTTP, HTTPS, FTP, SMTP, or NNTP.
  - *Host:* The name of the server used to provide access to the resource, such as blog.johnmuellerbooks.com.
  - *Port number:* The port used to access the resource. In general, you do not provide this component because most sites use standard ports, which are assumed by the browser. For example, HTTP relies on port 80 for communication. When the server uses port 80, you do not need to include the port number as part of the URI.
  - *Path:* The fully defined location of the resource on the server. In some cases, you do not provide

a path, which means that the server provides the resource found on the default path.

- *Query string:* Name and value pairs that define additional information required to obtain the resource you want on the server.

- encodeURI(): Encodes a URI.
- encodeURIComponent(): Encodes a URI component rather than the entire URI.
- escape(): Encodes a string using the same techniques used for a URI. For example, escape("This string is encoded!") outputs This%20string%20is%20 encoded%21.
- eval(): Accepts a string that contains a script and then executes the string content as a script. Many developers use this function to create self-modifying applications that provide good flexibility.
    - Using evaluated code opens your application to potential security problems through injection attacks.
    - Debugging evaluated code is incredibly hard because none of the normal debugging tools will work.
    - Evaluated code runs more slowly because the browser cannot compile and then cache it.
- isFinite(): Returns true when a value is a finite, legal number.
- isNaN(): Returns true when a value is an illegal number.
- Number(): Changes an object's value to a native number.
- parseFloat(): Parses a string and returns a floating point number.
- parseInt(): Parses a string and returns an integer.
- String(): Converts an object's value to a string. This function provides the same output as the toString() method provided by most objects.
- unescape(): Decodes an encoded string by using the same techniques used for a URI.

3.3 UNDERSTANDING DATE AND TIME IN JAVASCRIPT

Date and time are a regular part of our everyday lives and therefore feature prominently in computer programming. In JavaScript, you might have to create a website with a calendar, a train schedule, or an interface to set up appointments. These applications need to show relevant times based on the user's current **time zone**, or perform calculations around arrivals and departures or start and end times. Additionally, you might need to use JavaScript to generate a report at a certain time every day, or filter through currently open restaurants and establishments.

To achieve all of these objectives and more, JavaScript comes with the built in Date object and related methods. This section will go over how to format and use date and time in .JavaScript

Keyword

Time zone
is a region of the globe that observes a uniform standard time for legal, commercial, and social purposes.

3.3.1 The Date Object

The Date object is a built-in object in JavaScript that stores the date and time. It provides a number of built-in methods for formatting and managing that data. By default, a new Date instance without arguments provided creates an object corresponding to the current date and time. This will be created according to the current computer's system settings.

To demonstrate JavaScript's Date, let's create a variable and assign the current date to it. This section is being written on Wednesday, October 18th in London (GMT), so that is the current date, time, and time zone that is represented below.

```
now.js
// Set variable to current date and time
const now = new Date();
// View the output
now;
```

Output

```
Wed Oct 18 2017 12:41:34 GMT+0000 (UTC)
```

Looking at the output, we have a date string containing the following:

Day of the Week	Month	Day	Year	Hour	Minute	Second	Time zone
Wed	Oct	18	2017	12	41	34	GMT+0000 (UTC)

Remember

Encoding replaces whitespace characters, such as a space, with whitespace equivalent values, such as %20. In addition, Unicode characters that would normally cause parsing problems, such as those with diacritical marks, are replaced with their Unicode equivalents.

The date and time is broken up and printed in a way that we can understand as humans. JavaScript, however, understands the date based on a timestamp derived from Unix time, which is a value consisting of the number of milliseconds that have passed since midnight on January 1st, 1970. We can get the timestamp with the getTime() method.

```
// Get the current timestamp
now.getTime();
Output
1508330494000
```

The large number that appears in our output for the current timestamp represents the same value as above, October 18th, 2017.

Epoch time, also referred to as zero time, is represented by the date string 01 January, 1970 00:00:00 Universal Time (UTC), and by the 0 timestamp. We can test this in the browser by creating a new variable and assigning to it a new Date instance based on a timestamp of 0.

```
epoch.js
// Assign the timestamp 0 to a new variable
const epochTime = new Date(0);
epochTime;
Output
01 January, 1970 00:00:00 Universal Time (UTC)
```

Epoch time was chosen as a standard for computers to measure time by in earlier days of programming, and it is the method that JavaScript uses. It is important to understand the concept of both the timestamp and the date string, as both may be used depending on the settings and purpose of an application.

So far, we learned how to create a new Date instance based on the current time, and how to create one based on a timestamp. In total, there are four formats by which you can create a new Date in JavaScript. In addition to the current time default and timestamp, you can also use a date string, or specify particular dates and times.

Date Creation	**Output**
new Date()	Current date and time
new Date(timestamp)	Creates date based on milliseconds since Epoch time
new Date(date string)	Creates date based on date string
new Date(year, month, day, hours, minutes, seconds, milliseconds)	Creates date based on specified date and time

To demonstrate the different ways to refer to a specific date, we will create new Date objects that will represent July 4th, 1776 at 12:30pm GMT in three different ways.

```
usa.js
// Timestamp method
new Date(-6106015800000);
// Date string method
new Date("January 31 1980 12:30");
// Date and time method
new Date(1776, 6, 4, 12, 30, 0, 0);
```

The three examples above all create a date containing the same information.

You will notice the timestamp method has a negative number; any date prior to Epoch time will be represented as a negative number.

In the date and time method, our seconds and milliseconds are set to 0. If any number is missing from the Date creation, it will default to 0. However, the order cannot be changed, so keep that in mind if you decide to leave off a number. You may also notice that the month of July is represented by 6, not the usual 7. This is because the date and time numbers start from 0, as most counting in programming does. See the next section for a more detailed chart.

Keyword

Date object is used to work with dates and times. You create an instance of the Date object with the «new» keyword.

3.3.2 Retrieving the Date with get

Once we have a date, we can access all the components of the date with various built-in methods. The methods will return each part of the date relative to the local time zone. Each of these methods starts with get, and will return the relative number. Below is a detailed table of the get methods of the **Date object**.

Date/Time	Method	Range	Example
Year	getFullYear()	YYYY	1970
Month	getMonth()	0-11	0 = January
Day (of the month)	getDate()	1-31	1 = 1st of the month
Day (of the week)	getDay()	0-6	0 = Sunday
Hour	getHours()	0-23	0 = midnight
Minute	getMinutes()	0-59	

Second	getSeconds()	0-59	
Millisecond	getMilliseconds()	0-999	
Timestamp	getTime()	Milliseconds since Epoch time	

Let's make a new date, based on July 31, 1980, and assign it to a variable.

harryPotter.js

```
// Initialize a new birthday instance
const birthday = new Date(1980, 6, 31);
```

Now we can use all our methods to get each date component, from year to millisecond.

getDateComponents.js

```
birthday.getFullYear();      // 1980
birthday.getMonth();         // 6
birthday.getDate();          // 31
birthday.getDay();           // 4
birthday.getHours();         // 0
birthday.getMinutes();       // 0
birthday.getSeconds();       // 0
birthday.getMilliseconds();  // 0
birthday.getTime();          // 333849600000 (for GMT)
```

Sometimes it may be necessary to extract only part of a date, and the built-in get methods are the tool you will use to achieve this. For an example of this, we can test the current date against the day and month of October 3rd to see whether it is October 3rd or not.

oct3.js

```

// Get today's date
const today = new Date();

// Compare today with October 3rd
if (today.getDate() === 3 && today.getMonth() === 9) {
  console.log("It is October 3rd.");
} else {
  console.log("It is not October 3rd.");
}
```

Output

```
It is not October 3rd.
```

Since, at the time of writing, it is not October 3rd, the console reflects that. The built-in Date methods that begin with get allow us to access date components that return the number associated with what we are retrieving from the instantiated object.

3.3.3 Modifying the Date with set

For all the get methods that we learned about above, there is a corresponding set method. Where get is used to retrieve a specific component from a date, set is used to modify components of a date. Below is a detailed chart of the set methods of the Date object.

Date/Time	**Method**	**Range**	**Example**
Year	setFullYear()	YYYY	1970
Month	setMonth()	0-11	0 = January
Day (of the month)	setDate()	1-31	1 = 1st of the month
Day (of the week)	setDay()	0-6	0 = Sunday
Hour	setHours()	0-23	0 = midnight
Minute	setMinutes()	0-59	
Second	setSeconds()	0-59	
Millisecond	setMilliseconds()	0-999	
Timestamp	setTime()	Milliseconds since Epoch time	

We can use these set methods to modify one, more, or all of the components of a date. For example, we can change the year of our birthday variable from above to be 1997 instead of 1980.

harryPotter.js

```
// Change year of birthday date
birthday.setFullYear(1997);

birthday;
```

Output

```
Thu Jul 31 1997 00:00:00 GMT+0000 (UTC)
```

We see in the example above that when we call the birthday variable we receive the new year as part of the output. The built-in methods beginning with set let us modify different parts of a Date object.

3.3.4 Date Methods with UTC

The get methods discussed above retrieve the date components based on the user's local time zone settings. For increased control over the dates and times, you can use the getUTC methods, which are exactly the same as the get methods, except they calculate the time based on the UTC (Coordinated Universal Time) standard. Below is a table of the UTC methods for the JavaScript Date object.

Date/Time	**Method**	**Range**	**Example**
Year	getUTCFullYear()	YYYY	1970
Month	getUTCMonth()	0-11	0 = January
Day (of the month)	getUTCDate()	1-31	1 = 1st of the month
Day (of the week)	getUTCDay()	0-6	0 = Sunday
Hour	getUTCHours()	0-23	0 = midnight
Minute	getUTCMinutes()	0-59	
Second	getUTCSeconds()	0-59	
Millisecond	getUTCMilliseconds()	0-999	

To test the difference between local and UTC get methods, we can run the following code.

UTC.js

```
// Assign current time to a variable
const now = new Date();

// Print local and UTC timezones
console.log(now.getHours());
console.log(now.getUTCHours());
```

Running this code will print out the current hour, and the hour of the UTC time zone. If you are currently in the UTC time zone the numbers that are output from running the program above will be the same. UTC is useful in that it provides an international time standard reference and can therefore keep your code consistent across time zones if that is applicable to what you are developing.

CASE STUDY

JAVASCRIPT DESIGN PATTERNS

Background

Documenting the path, struggle and findings as I am doing Udacity's JavaScript Design Patterns course. The starting project was to build a Cat clicker where every time you click on a cat, the number next to it is incremented. I hate cats, so I did Butterfly clicker (couldn't find cute elephants pics). First it was just one cat, then requirements changed and it was 2 cats and we were asked not to hard-code data.

At this point, my code looked like this:

```
<div id="main">
    <h1>Butterfly Clicker</h1>
    <div id="insertButterflies">
    </div>
</div>
#main { max-width: 80%; margin: 0 auto; }
#insertButterflies label { display: block; margin: 10px 0; font-weight: bold; }
$(document).ready(function () {
    buttterFlyClicker.init();
});

var buttterFlyClicker = {
   catObjects: [{ "name": "Orange", "imageUrl": "images/Orange.jpg", "clickCount":
7 }, { "name": "Browny", "imageUrl": "images/Browny.jpg", "clickCount": 23 }],
    init: function(){
        for (var i = 0; i < buttterFlyClicker.catObjects.length; i++) {
            var butterfly = buttterFlyClicker.catObjects[i];
        $("#insertButterflies").append('<div><h2>' + butterfly.name + '</h2><img src="'
+ butterfly.imageUrl + '" onclick=\"buttterFlyClicker.updateCount(this);\" /><label>' +
butterfly.clickCount + '</label></div>');
        }
    },
    updateCount: function (id) {
        var lbl = $(id).next();
        var num = parseInt($(lbl).html());
        num++;
        $(lbl).html(num);
    }
};
```

Here the instructor pointed out a bug that we can come across with respect to closures.

Let's say you have an array of elements and you are looping through them and want to alert the number when that number element is clicked

```
var nums = [1,2,3];


for (var i = 0; i < nums.length; i++) {
    var num = nums[i];
    var elem = document.createElement('div');
    elem.textContent = num;


    elem.addEventListener('click', function() {
        alert(num);
    });
    document.body.appendChild(elem);
};
```

We might expect to see 1, 2 and 3 alerted but we get the last number 3 all the time we click.

This is because the value of num keeps changing in the for loop but the event handler function is executed when the element is clicked by which num would have been set to last element 3.

The fix involves use of closures. We can create an inner scope to hold the value of num at the exact time the event handler is added.

```
elem.addEventListener('click', (function(numCopy) {
    return function() {
        alert(numCopy)
    };
})(num));
```

The outer function is immediately invoked by passing num into numCopy, so even if num changes numCopy won't be affected.

Next step the requirements changed again, this time to have a list of cats and clicking on any to update the display area with the selected cat. At this point my code was working but not scalable/readable, involved inline event listeners and not meeting industry standards. I needed to learn better organizational techniques so my applications are stable, bug-free, cleanly written and they scale well and are extensible.

Spaghetti code is easily avoidable once we know how to avoid it. The problem is things get really messy when you connect things together. And an application is ultimately all about connecting pieces of code to each other. But if we connect all the pieces to all the other pieces, suddenly you can't move anything around anymore.

Separation of Concerns:

We can separate our code into fundamentally different pieces. No matter how the size of the application, programmers like to organize everything into buckets: Model, View and Controller

Model: All the data is stored here which includes data from the server and from the user.

View: This is all the stuff user see and interacts with includes DOM elements, input elements, buttons and images. This is user's interface to the application both for giving and ready data.

Controller: Connects Model and View. Controller provides the separation of concerns that is needed when we're building applications. This holds things together but also keeps them separate enough to allow changes in individual pieces without disturbing the rest i.e., view can be changed without disturbing the model or we can change the way we store data in the model without disturbing the view.

At this point we started with this pizza repository code to learn separation on concerns.

Using this I attempted to redo my butterfly clicker code and will be noting each point.

<meta charset="utf-8">

The editors generally plug this in and we kind of know what it is but can't really tell the proper definition. This meta tag specifies what character set is your website written with.

Definition of UTF-8: UTF-8 (U from Universal Character Set + Transformation Format—8-bit) is a character encoding capable of encoding all possible characters (called code points) in Unicode. The encoding is variable-length and uses 8-bit code units.

<meta name="viewport" content="width=device-width, initial-scale=1">

This is used to make your site responsive i.e., the browser will (probably) render the width of the page at the width of its own screen. So if that screen is 320px wide, the browser window will be 320px wide, rather than way zoomed out and showing 960px (or whatever that device does by default, in lieu of a responsive meta tag).

JavaScript Templating

JavaScript Templating is useful when you have dynamic content but don't have server-side templating available. It also arguably minimize the amount of data returned to the client (ex. data as an object or JSON instead of the entire markup) making sites faster and servers more responsive by lowering bandwidth and load.

SUMMARY

- A function is a block of code that performs an action or returns a value. Functions are custom code defined by programmers that are reusable, and can therefore make your programs more modular and efficient. JavaScript has several "top-level" built-in functions. JavaScript also has four built-in objects: Array, Date, Math, and String.
- A function (also known as a method) is a self-contained piece of code that performs a particular "function", then returns a value. You can recognize a function by its format — it is a piece of descriptive text, followed by open and close brackets.
- JavaScript is a functional language meaning that functions are the primary modular units of execution. Functions are obviously very important in JavaScript.
- JavaScript, first introduced by Netscape, changed the static fate of the HTML web pages. Before JavaScript came into existence, HTML pages were just used to render static content. Inserting JavaScript to a web page, can give it a significant degree of interactivity and functionality.
- JavaScript is an object oriented programming language. It supports the concept of objects in the form of attributes. If an object attribute consists of function, then it is a method of that object, or if an object attribute consists of values, then it is a property of that object.
- JavaScript arrays are a special kind of object, and are created dynamically. An array object contains a number of variables. The number of variables may be zero, in which case the array is said to be empty. The variables contained in an array have no names; instead they are referenced by array access expressions that use nonnegative integer index values. These variables are called the *components* of the array.
- The Date object provides a system-independent **abstraction** of dates and times. Dates may be constructed from a year, month, day of the month, hour, minute, and second, and those six components, as well as the day of the week, may be extracted from a date. Dates may also be compared and converted to a readable string form. A Date is represented to a precision of one millisecond.
- JavaScript gives you access a number of built-in functions, including the prompt() function, which lets you ask the user for written input. As with the confirm() function, you provide a text prompt to ask the user to provide a value. On return, you set the output of the function equal to the prompt() function and use the data in your application.

KNOWLEDGE CHECK

1. **Functions are invoked as functions or as methods with an ....**
   a. invocation statement
   b. invocation expression
   c. invocation function
   d. invocation method
2. **An .... consists of a function expression that evaluates to a function object followed by an open parenthesis, a comma separated list of zero or more argument expressions and a close parenthesis.**
   a. invocation statement
   b. invocation expression
   c. invocation function
   d. invocation method
3. **If a function or method invocation preceded by the keyword new, then it is a ....**
   a. constructor invocation
   b. new invocation
   c. indirect invocation
   d. direct invocation
4. **In .... you can invoke any function as a method of any object, even if it is not actually a method of that object.**
   a. constructor invocation
   b. new invocation
   c. indirect invocation
   d. direct invocation
5. **Both call( ) and apply( ) methods allow you to explicitly specify the .... value for the invocation.**
   a. this
   b. me
   c. that
   d. new

6. **Which one of the following is used for the calling a function or a method in the JavaScript:**
   a. Property Access Expression
   b. Functional expression
   c. Invocation expression
   d. Primary expression
7. **Which of the following function of the String object returns the character in the string starting at the specified position via the specified number of characters?**
   a. slice()
   b. split()
   c. substr()
   d. search()
8. **Which of the following number object function returns the value of the number?**
   a. toString()
   b. valueOf()
   c. toLocaleString()
   d. toPrecision()

REVIEW QUESTIONS

1. What do you understand by function in JavaScript?
2. How to define function parameters.
3. What do you mean by number methods?
4. Discuss about string HTML wrappers method.
5. Define the array methods in JavaScript.
6. Explain date and time in JavaScript. Briefly.

Check Your Result

1. (b)	2. (b)	3. (a)	4. (c)	5. (a)
6. (c)	7. (c)	8. (b)		

REFERENCES

1. Eich, Brendan (21 June 2011). "New JavaScript Engine Module Owner". *brendaneich.com*. Retrieved 16 July 2016.
2. Flanagan, David. *JavaScript - The definitive guide* (6 ed.). p. 1. JavaScript is part of the triad of technologies that all Web developers must learn: HTML to specify the content of web pages, CSS to specify the presentation of web pages, and JavaScript to specify the behavior of web pages.
3. http://hepunx.rl.ac.uk/~adye/jsspec11/builtin.htm
4. http://www.programming-free.com/2012/07/javascript-built-in-functions-with.html
5. https://codeburst.io/parameters-arguments-in-javascript-eb1d8bd0ef04
6. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects
7. https://javascript.info/native-prototypes
8. https://medium.com/@_bengarrison/javascript-es6-exploring-the-new-built-in-methods-b62583b0a8e6
9. https://medium.freecodecamp.org/here-are-the-new-built-in-methods-and-functions-in-javascript-8f4d2fd794fa
10. https://openclassrooms.com/en/courses/3523231-learn-to-code-with-javascript/3685856-write-functions
11. https://www.digitalocean.com/community/tutorials/how-to-define-functions-in-javascript
12. https://www.digitalocean.com/community/tutorials/understanding-date-and-time-in-javascript
13. https://www.dofactory.com/tutorial/javascript-function-objects
14. https://www.dummies.com/web-design-development/html/how-to-use-javascripts-built-in-functions-to-program-with-html/
15. https://www.scribd.com/document/237949356/Javascript-Bibliography
16. https://www.tutorialride.com/javascript/javascript-built-in-functions.htm
17. https://www.tutorialspoint.com/javascript/javascript_builtin_functions.htm
18. https://www.w3schools.com/jsref/jsref_obj_global.asp
19. McCracken, Harry (16 September 2010). "The Unwelcome Return of "Best Viewed with Internet Explorer"". *technologizer.com*. Retrieved 16 July 2016.
20. Robert Nyman, Getters And Setters With JavaScript – Code Samples And Demos, Robertnyman.com, published 29 May 2009, accessed 2 January 2010.
21. Severance, Charles (February 2012). "JavaScript: Designing a Language in 10 Days". *Computer*. IEEE Computer Society. 45 (2): 7–8. doi:10.1109/MC.2012.57. Retrieved 23 March 2013.

CHAPTER 4

HTML FORMS

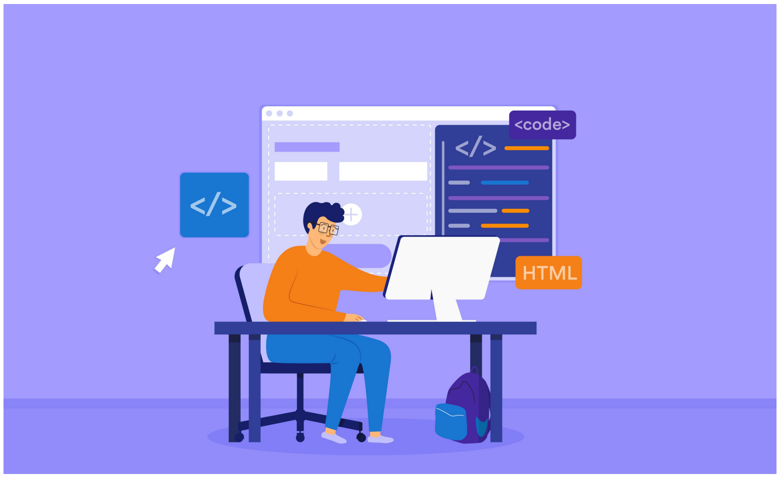
"When you decide to put your business online it is a little bet tricky step for novice computer users because they want to keep data safe & secure. This problem developed from companies which did not take security seriously."

– Mohamed Saad

LEARNING OBJECTIVES

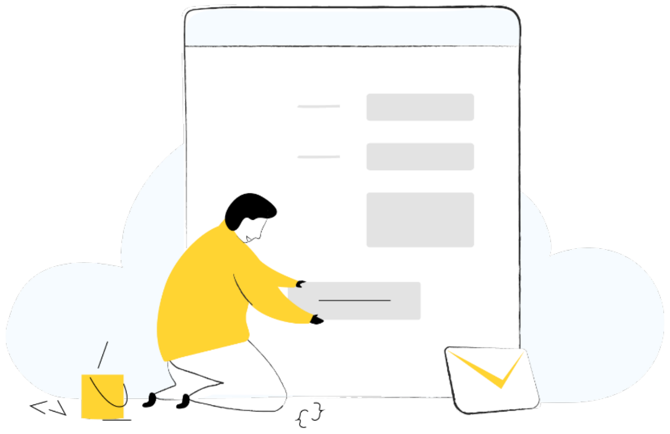
After studying this chapter, you will be able to:

1. Understand forms basics
2. Identify Html form elements
3. Discuss about styling html forms



INTRODUCTION

HTML Forms are one of the main points of interaction between a user and a web site or application. They allow users to send data to the web site. Most of the time that data is sent to the web server, but the web page can also intercept it to use it on its own. An HTML Form is made of one or more widgets. Those widgets can be text fields (single line or multiline), select boxes, buttons, checkboxes, or radio buttons. Most of the time those widgets are paired with a label that describes their purpose — properly implemented labels are able to clearly instruct both sighted and blind users on what to enter into a form input.



The main difference between a HTML form and a regular HTML document is that most of the time, the data collected by the form is sent to a web server. In that case, you need to set up a web server to receive and process the data.

4.1 FORMS BASICS

Web forms are used by virtually all websites for a wide range of purposes. Users of forums and social networks use forms to add content and interact with other users. Websites that can be customized to create a personalized experience, such as customizable newsfeeds, use forms to allow users to control the content that appears on the page. And nearly every website uses forms to allow website visitors to contact the organization or person administering the website. Web forms are made possible by the integration of multiple technologies:

- HTML to create the form fields and labels and accept user input.
- CSS to style the presentation of the form.
- JavaScript to validate form input and provide Ajax-enabled interactions.
- Server-side languages such as PHP to process form data.

HTML Form is a document which stores information of a user on a web server using interactive controls. An HTML form contains different kind of information such as username, password, contact number, email id etc.

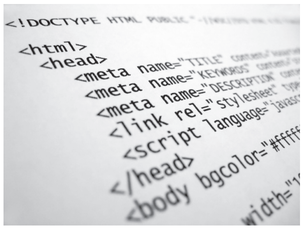
The elements used in an HTML form are check box, input box, radio buttons, submit buttons etc. Using these elements the information of an user is submitted on a web server.

The **form** tag is used to create an HTML form.

Example of an HTML Form:

```
<!DOCTYPE html>
<html>
<body>

<form>
  Username:<br>
  <input type="text" name="username">
  <br>
  Email id:<br>
  <input type="text" name="email_id">
  <br><br>
  <input type="submit" value="Submit">
</form>

</body>
</html>
```

The basic tags used in the actual HTML of forms are form, input, textarea, select and option.



```
<!DOCTYPE HTML PUBLIC "-
<html>
  <head>
    <meta name="TITLE" cont
    <meta name="KEYWORDS" conten
    <meta name="DESCRIPTION" con
    <link rel="stylesheet" typ
    <script language="javas
  </head>
  <body bgcolor="#ffff
    width="1
```

4.1.1 Form

form defines the form and within this tag, if you are using a form for a user to submit information (which we are assuming at this level), an actionattribute is needed to tell the form where its contents will be sent to.

The method attribute tells the form how the data in it is going to be sent and it can have the value get, which is default, and latches the form information onto a **web address**, or post, which (essentially) invisibly sends the form's information.

Keyword

Web address, or domain name, is an address where you can be found online.

get is used for shorter chunks of non-sensitive information - you might see the information you have submitted in a web site's search to appear in the web address of its search results page, for example, post is used for lengthier, more secure submissions, such as in contact forms.

So a form element will look something like this:

<form action="processingscript.php" method="post">

</form>

4.1.2 Input

The input tag is the daddy of the form world. It can take a multitude of guises, the most common of which are outlined below:

- <input **type="text"**> or simply <input> is a standard textbox. This can also have a value attribute, which sets the initial text in the textbox.
- <input **type="password"**> is similar to the textbox, but the characters typed in by the user will be hidden.

- <input **type="checkbox"**> is a checkbox, which can be toggled on and off by the user. This can also have a checked attribute (<input type="checkbox" **checked**> - the attribute doesn't require a value), and makes the initial state of the check box to be switched on, as it were.
- <input **type="radio"**> is similar to a checkbox, but the user can only select one radio button in a group. This can also have a checked attribute.
- <input **type="submit"**> is a button that when selected will submit the form. You can control the text that appears on the submit button with the value attribute, for example <input type="submit" value="Ooo. Look. Text on a button. Wow">.

Remember

Like img and br tags, the input tag, which doesn't surround any content, doesn't require a closing tag.

4.1.3 Textarea

textarea is, basically, a large, multi-line textbox. The anticipated number of rows and columns can be defined with rows and cols attributes, although you can manipulate the size to your heart's content using CSS.

```
<textarea rows="5" cols="20">A big load of text</textarea>
```

Any text you choose to place between the opening and closing tags (in this case "a big load of text") will form the initial value of the text area.

4.1.4 Select

The select tag works with the option tag to make drop-down select boxes.

```
<select>
    <option>Option 1</option>
    <option>Option 2</option>
    <option value="third option">Option 3</option>
</select>
```

When the form is submitted, the value of the selected option will be sent. This value will be the text between the selected opening and closing option tag unless an explicit value is specified with the value attribute, in which case this will be sent instead. So, in the above example, if the first item is selected, "Option 1" will be sent, if the third item is selected, "third option" will be sent.

Similar to the checked attribute of checkboxes and radio buttons, an option tag can also have a selected attribute, to start off with one of the items already being selected, eg. <option selected>Rodent</option> would pre-select "Rodent" from the items.

Keyword

Attribute is a characteristic of a page element, such as a font. An HTML user can set font attributes, such as size and color, to different values.

4.1.5 Names

All of the tags mentioned above will look very nice presented on the page but if you hook up your form to a form-handling script, they will all be ignored. This is because the form fields need **names**. So to all of the fields, the attribute name needs to be added, for example <input type=»text» name=»talkingsponge»>.

A contact form for Noah's Ark, for example, might look something like the one below. (Note: this form will not work unless there is a "contactus.php" file, which is stated in the action **attribute** of the form tag, to handle the submitted data).

```
<form action="contactus.php" method="post">
    <p>Name:</p>
    <p><input type="text" name="name" value="Your name"></p>


    <p>Species:</p>
    <p><input name="species"></p>
  <!-- remember: 'type="text"' isn't actually necessary -->


    <p>Comments: </p>
  <p><textarea name="comments" rows="5" cols="20">Your comments</textarea></p>


    <p>Are you:</p>
  <p><input type="radio" name="areyou" value="male"> Male</p>
```

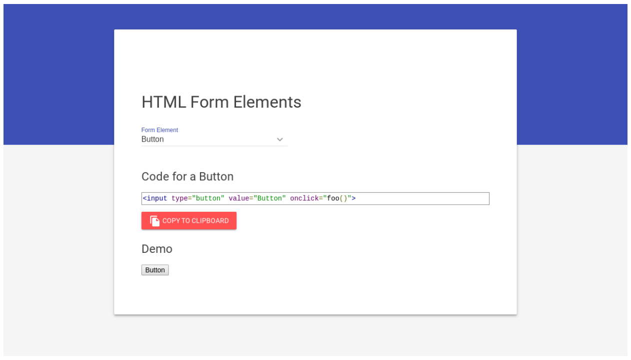
```
        <p><input type="radio" name="areyou" value="female"> Female</p>
          <p><input type="radio" name="areyou" value="hermaphrodite"> An
hermaphrodite</p>
        <p><input type="radio" name="areyou" value="asexual"> Asexual</p>


        <p><input type="submit"></p>


    </form>
```

4.2 HTML FORM ELEMENTS

HTML forms are an awesome and professional tool for collecting user data. However, these forms, just like everything else in HTML, wouldn't work without some proper HTML form elements.

HTML forms can help you with getting to know your users and their suggestions for you. With a list of HTML form elements, you will be able to create an excellent user experience on your website, maintain quality and add functionality. There is nothing better while developing a website than having satisfied users and proper results.

HTML form elements are used to capture user input. There are many different types of form elements such as the text box, check box, drop down, submit button, and much more.


HTML Form Elements

Form Element: Button

Code for a Button

```
<input type="button" value="Button" onclick="foo()">
```

Take a look at an example form with a few input elements below:

The form elements above are the most common used form elements. We will go into further detail on each of these form elements:

- Text Box Input
- Password Input

- Text Area
- Select Drop Down
- Check Box
- Radio Input
- File Input
- Submit Button

4.2.1 Text Box Input

The text box input is used to capture text such as a name, **email, address**, or any other type of text. To create an HTML text box we will use the input tag and specify that the type attribute to be text.

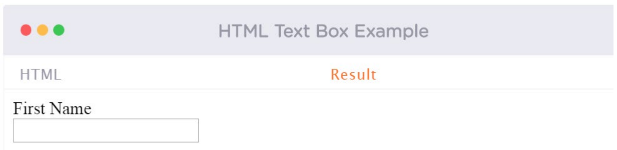
> **Keyword**
>
> **Email address** identifies an email box to which messages are delivered.

Take a look at how to create a text box in HTML:

<input type="text" name="first_name">

Notice the name attribute. This is the name that will be sent to the server when the form is submitted, so our server side code would get this text box value by referencing the first_nameform element.

Let's take another quick example of how a text box looks :in HTML



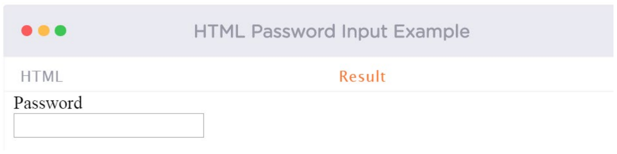
Notice we have text above the text box which says First Name, this is referred to as a label and is a way of describing what the user should input in the form element. A basic HTML label will look like the following:

<label for="first_name">First Name</label>

<label> tags are very common when using HTML forms. Notice the forattribute contains the value first_name this matches the name of your input. So since our input has a name of name="first_name" our label will have an attribute of for="first_name". Easy Peasy, right. Let's move on to the password input.

4.2.2 Password Input

The password input is essentially the same thing as the Text Box input; however, it does not show the text as the user types. Try it out in the example below:



And this is how you will create a password input:

<input type="password" name="password">

Remember

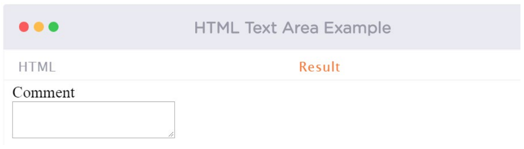
The characters in a password field are masked (shown as asterisks or circles).

4.2.3 Text Area

To create a Text Area where a large amount of text can be entered by the user we can use the <textarea> tag. Take a look at an example of how to create a Text Area in HTML:

<textarea name="comment"></textarea>

And a Text Area would look like the following:



4.2.4 Select Drop Down

A Drop Down is a specific list of options that a user can choose when selecting a Drop Down menu. To create a Select Dropdown in HTML you would represent this inside of <select></select> tags, then each option that you want to allow your user, you will specify with an <option></option> tag inside the select tags. Take a look at the following example:

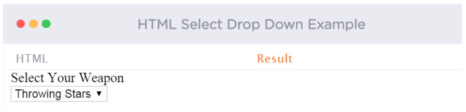
<select name="weapon">

<option value="throwing_stars">Throwing Stars</option>

<option value="sword">Sword</option>

```
    <option value="staff">Staff</option>
</select>
```

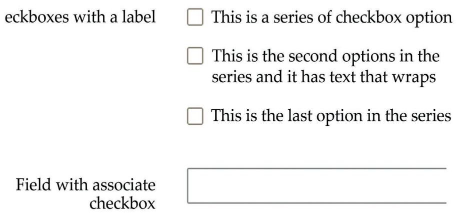
And this will give you the following result:



Select Your Weapon
Throwing Stars ▾

4.2.5 Check Box

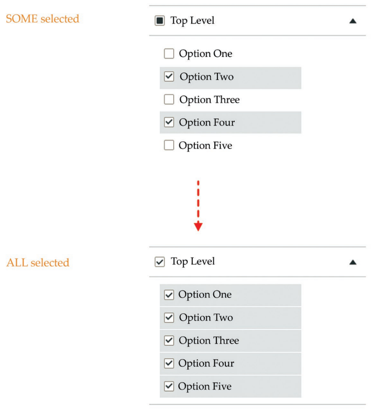
Checkboxes should be used in cases where one or many options may be selected. A checkboxes may also be used in cases where the user may wish to 'opt in' or enable an action or setting. Checkboxes may be used in relation to other form elements as well.


eckboxes with a label
This is a series of checkbox option
This is the second options in the series and it has text that wraps
This is the last option in the series
Field with associate checkbox

Remember

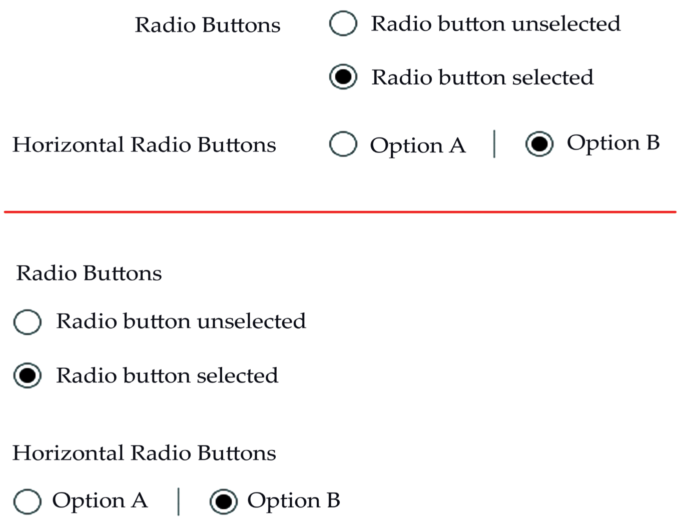
Checkboxes let a user select ZERO or MORE options of a limited number of choices.

If there is a *parent-child* relationship of associated checkboxes there should be some visual distinction for the user between "some" checked and "all" checked, in the event that the child elements are hidden from view as in an expandable section.



4.2.6 Radio Input

Radio buttons should be used in cases of ‘yes’ or ‘no’. The pair of radio buttons may be arranged either vertically or horizontally. Radio buttons may often have a default or preferred selection. This preferred selection should be the first button of the pair whenever possible. In some instances when only a single selection may be made but there are several options to choose from, all of the radio button in the set may appear un-selected or ‘empty’. Once a user has made a selection however the selected state should be evident and be visible henceforth, even if the user changes the selection - it cannot be returned to the initial ‘un-selected’ state.

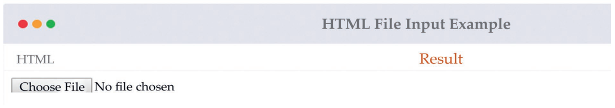


4.2.7 File Input

A File input is used to allow users to upload images or files. To create a File input element we will use the input element and give it a type of file like the following:

```
<input type=”file” name=”image”>
```

This will give us the following results:



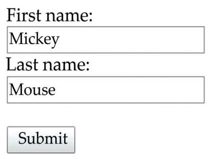
When clicking on the Choose File button above the user will then be prompted with a window to specify which file they would like to upload.

4.2.8 Submit Button

Lastly we are going to need a way to submit our form with all the data. This is what the submit button is used for. In order to use the submit button we will use another input element with a type of submit:

```
<input type=”submit” value=”Submit The Form”>
```

And our Submit button will look like the following:



First name:

Mickey

Last name:

Mouse

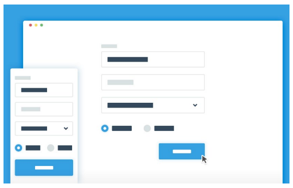
That’s a lot of form element. But don’t get overwhelmed they are all really easy to use and integrate. Allowing user input with forms is very common and it allows for our web site to provide more functionality for our user.

Keyword

User Agent Style sheets simply refer to the default styles that browsers apply to web pages. It is of the lowest importance considering User Styles and Author styles will overwrite these. Each browser is a little different in how it displays “unstyled” html.

4.3 STYLING HTML FORMS

To style elements that are easy to style with CSS, you should not face any difficulties, since they mostly behave like any other HTML element. However, the **user-agent style sheet** of every browser can be a little inconsistent, so there are a few tricks that can help you style them in an easier way.

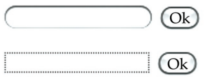


4.3.1 Search Fields

Search boxes are the only kind of text fields that can be a little tricky to style. On WebKit based browsers (Chrome, Safari, etc.), you’ll have to tweak it with the -webkit-appearance proprietary property.

Example

```
<form>
  <input type="search">
</form>
input[type=search] {
  border: 1px dotted #999;
  border-radius: 0;
  -webkit-appearance: none;
}
```



Chrome 25 / Mac OSX

As you can see on this screenshot of the search field on Chrome, the two fields have a border set as in our example. The first field is rendered without using the -webkit-appearance property, whereas the second is rendered using -webkit-appearance:none. This difference is noticeable.

Did You Know?

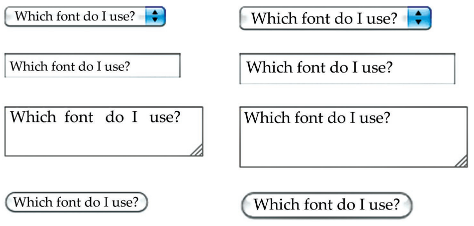
Forms are usually combined with programs written in various programming language to allow developers to create dynamic web sites. The most popular languages include both client-side and/ or server-side languages.

4.3.2 Fonts and Text

CSS font and text features can be used easily with any widget (and yes, you can use @font-face with form widgets). However, browsers' behaviors are often inconsistent. By default, some widgets do not inherit font-family and font-sizefrom their parents. Many browsers use the system default appearance instead. To make your forms' appearance consistent with the rest of your content, you can add the following rules to your stylesheet:

```
button, input, select, textarea {
  font-family : inherit;
  font-size   : 100%;
}
```

The screenshot below shows the difference; on the left is the default rendering of the element in Firefox on Mac OS X, with the platform's default font style in use. On the right are the same elements, with our font harmonization style rules applied.



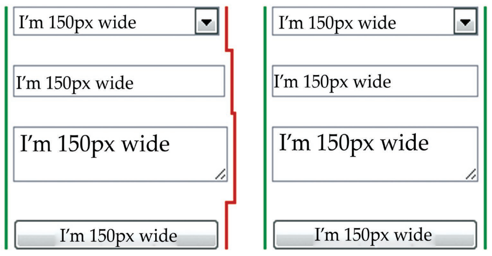
Firefox 16 / Mac OSX

There's a lot of debate as to whether forms look better using the system default styles, or customized styles designed to match your content. This decision is yours to make, as the designer of your site, or Web application.

4.3.3 Box Model

All text fields have complete support for every property related to the CSS box model (width, height, padding, margin, and border). As before, however, **browsers** rely on the system default styles when displaying these widgets. It is up to you to define how you wish to blend them into your content. If you want to keep the native look and feel of the widgets, you'll face a little difficulty if you want to give them a consistent size. This is because each widget has their own rules for border, padding and margin. So if you want to give the same size to several different widgets, you have to use the box-sizing property:

```
input, textarea, select, button {
  width : 150px;
  margin: 0;
  -webkit-box-sizing: border-box; /* For legacy WebKit based browsers */
   -moz-box-sizing: border-box; /* For legacy (Firefox <29) Gecko based browsers */
         box-sizing: border-box;
}
```



Chrome 22 / Windows 7

In the screenshot above, the left column is built without box-sizing, while the right column uses this property with the value border-box. Notice how this lets us ensure that all of the elements occupy the same amount of space, despite the platform's default rules for each kind of widget.

4.3.4 Positioning

Positioning of HTML form widgets is generally not a problem; however, there are two elements you should take special note of:

legend

The <legend> element is okay to style, except for positioning. In every browser, the <legend> element is positioned on top of the top border of its <fieldset>parent. There is absolutely no way to change it to be positioned within the HTML flow, away from the top border. You can, however, position it absolutely or relatively, using the position property. But otherwise it is part of the fieldset border. Because the <legend> element is very important for accessibility reasons, it will be spoken by assistive technologies as part of the label of each form element inside the fieldset, it's quite often paired with a title, and then hidden in an accessible way.

textarea

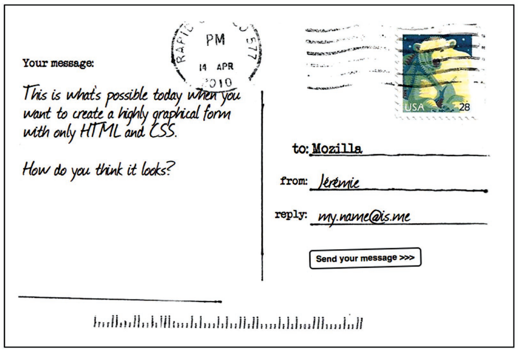
```
HTML
<fieldset>
  <legend>Hi!</legend>
  <h1>Hello</h1>
</fieldset>
CSS
legend {
  width: 1px;
  height: 1px;
  overflow: hidden;
}
```

By default, all browsers consider the <textarea> element to be an inline block, aligned to the text bottom line. This is rarely what we actually want to see. To change from inline-block to block, it›s pretty easy to use the display property. But if you want to use it inline, it›s common to change the vertical alignment:

```
textarea {
  vertical-align: top;
}
```

Example

Let's look at a concrete example of how to style an HTML form. This will help make a lot of these ideas clearer. We will build the following "postcard" contact form:


Your message:
This is what's possible today when you want to create a highly graphical form with only HTML and CSS.
How do you think it looks?
PM
14 APR
2010
USA 28
to: Mozilla
from: Jérémie
reply: my.name@is.me
Send your message >>>

If you want to follow along with this example, make a local copy of our postcard-start.html file, and follow the below instructions.

The HTML

The HTML is only slightly more involved than the example we used earlier; it just has a few extra IDs and a title.

```
<form>
  <h1>to: Mozilla</h1>

  <div id="from">
    <label for="name">from:</label>
    <input type="text" id="name" name="user_name">
  </div>
  <div id=”reply”>
    <label for=”mail”>reply:</label>
    <input type=”email” id=”mail” name=”user_email”>
  </div>

  <div id=”message”>
    <label for=”msg”>Your message:</label>
   <textarea id=”msg” name=”user_message”></textarea>
  </div>

  <div class=”button”>
    <button type=”submit”>Send your message</button>
  </div>
</form>
```

Add the above code into the body of your HTML.

Keyword

Coding refers to creating computer programming code.

Organizing Your Assets

Before we start **coding**, we need three additional assets:

- The postcard background
- A typewriter font
- A handdrawn font

Your fonts need some more processing before you start:

- Go to the fontsquirrel Webfont Generator.
- Using the form, upload both your font files and generate a webfont kit. Download the kit to your computer.
- Unzip the provided zip file.
- Inside the unzipped contents you will find two .woff files and two .woff2files. Copy these four files into a directory called fonts, in the same directory as before. We are using two different files for each font to maximise browser compatibility.

The CSS

Now we can dig into the CSS for the example. Add all the code blocks shown below inside the <style> element, one after another.

First, we prepare the ground by defining our @font-face rules, all the basics on the <body> element, and the <form> element:

```
@font-face {
    font-family: 'handwriting';
    src: url('fonts/journal-webfont.woff2') format('woff2'),
         url('fonts/journal-webfont.woff') format('woff');
    font-weight: normal;
    font-style: normal;
}

@font-face {
    font-family: 'typewriter';
    src: url('fonts/veteran_typewriter-webfont.woff2') format('woff2'),
         url('fonts/veteran_typewriter-webfont.woff') format('woff');
    font-weight: normal;
    font-style: normal;
}
body {
  font   : 21px sans-serif;

  padding : 2em;
  margin  : 0;

  background : #222;
}

form {
  position: relative;

  width  : 740px;
  height : 498px;
  margin : 0 auto;

  background: #FFF url(background.jpg);
}
```

Now we can position our elements, including the title and all the form elements:

```
h1 {
  position : absolute;
  left : 415px;
  top  : 185px;

  font : 1em "typewriter", sans-serif;
}

#from {
  position: absolute;
  left : 398px;
  top  : 235px;
}

#reply {
  position: absolute;
  left : 390px;
  top  : 285px;
}

#message {
  position: absolute;
  left : 20px;
  top  : 70px;
}
```

That's where we start working on the form elements themselves. First, let's ensure that the <label>s are given the right font:

```
label {
  font : .8em "typewriter", sans-serif;
}
```

The text fields require some common rules. Simply put, we remove their borders and backgrounds, and redefine their padding and margin:

```
input, textarea {
  font    : .9em/1.5em "handwriting", sans-serif;

  border  : none;
  padding : 0 10px;
  margin  : 0;
  width   : 240px;

  background: none;
}
```

Remember

Depending on browser support, the url field can be automatically validated when submitted.

When one of these fields gains focus, we highlight them with a light grey, transparent, background. Note that it's important to add the outline property, in order to remove the default focus highlight added by some browsers:

```
input:focus, textarea:focus {
  background    : rgba(0,0,0,.1);
  border-radius: 5px;
  outline       : none;
}
```

Now that our text fields are complete, we need to adjust the display of the single and multiple line text fields to match, since they won't typically look the same using the defaults.

The single-line text field needs some tweaks to render nicely in Internet Explorer. Internet Explorer does not define the height of the fields based on the natural height of the font (which is the behavior of all other browsers). To fix this, we need to add an explicit height to the field, as follows:

```
input {
    height: 2.5em; /* for IE */
    vertical-align: middle; /* This is optional but it makes legacy IEs look better */
}
```

<textarea> elements default to being rendered as a block element. The two important things here are the resize and overflow properties. Because our design is a fixed-size design, we will use the resize property to prevent users from resizing our multi-line text field. The overflow property is used to make the field render more consistently across browsers. Some browsers default to the value auto, while some default to the value scroll. In our case, it's better to be sure every one will use auto:

```
textarea {
  display : block;

  padding : 10px;
  margin  : 10px 0 0 -10px;
  width   : 340px;
  height  : 360px;

  resize  : none;
  overflow: auto;
}
```

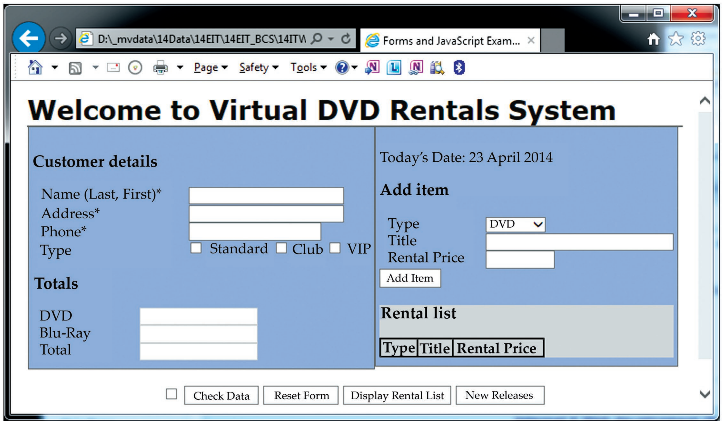
The <button> element is really convenient with CSS; you can do whatever you want, even using pseudo-elements:

```
button {
  position     : absolute;
  left         : 440px;
  top          : 360px;

  padding      : 5px;

  font         : bold .6em sans-serif;
  border       : 2px solid #333;
  border-radius: 5px;
  background   : none;

  cursor       : pointer;
-webkit-transform: rotate(-1.5deg);
   -moz-transform: rotate(-1.5deg);
    -ms-transform: rotate(-1.5deg);
     -o-transform: rotate(-1.5deg);
        transform: rotate(-1.5deg);
}

button:after {
  content: " >>>";
}

button:hover,
button:focus {
  outline    : none;
  background: #000;
  color     : #FFF;
}
```

CASE STUDY

CREATING A FORM

The first step to form validation is of course creating the actual form: For this case study we will be using a DVD Hire business as an example.



Today's Date: 23 April 2014

Type: DVD

Design Notes

1. Create the following files

- vmv_js-index.html = main html file
- vmv_js-style.css = cascading style sheet file
- vmv_js-index.js = File containing JavaScripts
- vmv_js-NewReleases.html = html page describing New Releases

2. Add the following to the head section of the main html file (To link the css and js files)

```
<link rel="stylesheet" type="text/css" href="vmv_js-style.css">
<script type="text/javascript" src="vmv_js-index.js"> </script>
```

3. "Welcome to .." is h1
4. "Customer details", "Totals", etc.. are h4
5. Use the following styles to vmv_js-index.css. Floating BOTH left will make sure that the two columns flow correctly.

```
.col1 {
   background: #93A5C4;  float:left;
```

```
    padding: 10px 5px 10px 5px;
    border: 1px solid #666;}
.col2 {
    background: #93A5C4; float:left;
    padding: 10px 5px 10px 5px;
    border: 1px solid #666;     }
```

6. Use a style to space out the field names (e.g. "Name (Last, First)")

```
.lbl120 { padding-left: 10px; width: 120px; float:left;}
```

7. Put txt in front of the field names and use CamelCase, for example

```
<input name="txtFullName" type="text" style="width: 186px">
```

8. Put btn in front of the button names and use Camel Case, for example:

```
<input name="btnChkForm" type="button" value="Check Data" onclick="fnCheckData()">
```

9. For the buttons *onclick* call the following functions

Check Data = fnCheckData()

Display Rental List = fnDisplayForm()

- New Release = fnNewReleases()

10. Today's Date will be called using a function fnDisplayTodaysDate();

11. The Add Item button will be called using a function called fnAddItem();

12. Use the following code for the Rental List table

```
<h4>Rental list</h4>
<table id="myTable" border="1" style=" border-collapse: collapse;">
  <tr><th>Type</th><th>Title</th><th>Rental Price</th></tr>
</table>
```

Notes

The JavaScript is used to Validate the Form data which is done before sending to the server so cuts down on Server traffic. To actually work with the data we would sent the Form data to an Internet Server which would process that data using a Server Side Language (such as PHP or ASP.NET) and this would put the data into a DataBase (e.g. MySQL, MS-SQL, Oracle).

SUMMARY

- HTML Forms are one of the main points of interaction between a user and a web site or application. They allow users to send data to the web site. Most of the time that data is sent to the web server, but the web page can also intercept it to use it on its own.
- An HTML Form is made of one or more widgets. Those widgets can be text fields (single line or multiline), select boxes, buttons, checkboxes, or radio buttons. Most of the time those widgets are paired with a label that describes their purpose — properly implemented labels are able to clearly instruct both sighted and blind users on what to enter into a form input.
- Web forms are used by virtually all websites for a wide range of purposes. Users of forums and social networks use forms to add content and interact with other users.
- HTML Form is a document which stores information of a user on a web server using interactive controls. An HTML form contains different kind of information such as username, password, contact number, email id etc.
- textarea is, basically, a large, multi-line textbox. The anticipated number of rows and columns can be defined with rows and cols attributes, although you can manipulate the size to your heart's content using CSS.
- HTML forms are an awesome and professional tool for collecting user data. However, these forms, just like everything else in HTML, wouldn't work without some proper HTML form elements.
- HTML form elements are used to capture user input. There are many different types of form elements such as the text box, check box, drop down, submit button, and much more.
- The text box input is used to capture text such as a name, email, address, or any other type of text. To create an HTML text box we will use the input tag and specify that the type attribute to be text.

KNOWLEDGE CHECK

1. **In HTML form <input type="text"> is used for**
   a. One line text
   b. Block of text
   c. One paragraph
   d. None
2. **The attribute of <form> tag**
   a. Method
   b. Action
   c. Both (a)&(b)
   d. None of these
3. **Correct HTML tag for the largest heading is**
   a. <head>
   b. <h6>
   c. <heading>
   d. <h1>
4. **For defining a submit button which tag is used?**
   a. <button>
   b. <submit button>
   c. <submit>
   d. <action submit>
5. **When form data contains sensitive or personal information than which method is more preferable?**
   a. Get method
   b. Post method
   c. Host method
   d. Put method
6. **Action attribute in HTML forms specifies that**
   a. Which HTTP method is used
   b. Which action is going on
   c. Where to submit form
   d. The auto completion of form

7. **Choose the correct option.**
   a. HTML form elements are used for taking user input.
   b. HTML form elements are defined inside <for> tag.
   c. HTML form elements can be of different types.
   d. All of these.
8. **Which one of the following is a form element?**
   a. text box.
   b. radio button.
   c. submit button.
   d. All of these.
9. **Choose the incorrect option.**
   a. radio button allows to choose only one option from the given options.
   b. default option can be chosen using attribute "selected" in radio button
   c. default option can be chosen using attribute "checked" in radio button
   d. checkbox allows to choose one or more than one options from the given options.
10. **How more than one option can be selected in drop down?**
   a. Use of multiple attribute inside <option> tag.
   b. Use of multiple attribute inside <select> tag.
   c. use of multiple attribute inside <text> tag.
   d. It is not possible to select more than one option in drop down.

REVIEW QUESTIONS

1. Write down the main purpose of HTML forms.
2. Write the difference between radio buttons and checkboxes.
3. Is the size of the text a user can enter in a text field limited by the value of the SIZE attribute in the <INPUT> tag?
4. What attribute must be included in a <TEXTAREA> tag to ensure a horizontal scroll bar is included in a multiline text field?
5. How do you set the initial state of a check box?

Check Your Result

1. (a)	2. (c)	3. (d)	4. (c)	5. (b)
6. (c)	7. (d)	8. (d)	9. (b)	10. (b)

REFERENCES

1. "Forms – HTML5". w3.org. W3C. Retrieved 2014-02-20.
2. "HTML/Elements/label". w3.org wiki.
3. "input type=color – color-well control". w3.org. W3C. Retrieved 2014-10-31.
4. "Radio Buttons". Windows Dev Center. Retrieved 14 September 2016.
5. Garofalo, Josh. "Intro to Online Forms and Form Builders". Blitzen Blog.
6. http://jkorpela.fi/forms/present.html
7. http://www.htmldog.com/guides/html/beginner/forms/
8. http://www-db.deis.unibo.it/courses/TW/DOCS/w3schools/html/html_form_input_types.asp.html
9. https://devdocs.magento.com/guides/v2.2/pattern-library/getting-user-input/form_elements/form_elements.html
10. https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms
11. https://html.com/forms/
12. https://marksheet.io/html-forms.html
13. Jarrett and Gaffney. Forms That Work: Designing Web Forms For Usability, Morgan Kaufmann, 2008.
14. Jumašev, Alex. "The history of a radio-button". JitBit Founders Blog. Retrieved 14 September 2016.
15. Satrom, Brandon (November 2011). "Better Web Forms with HTML5 Forms". MSDN Magazine. Microsoft. Retrieved 2014-02-20.
16. Wroblewski, Luke and, Spool, Jared. Web Form Design: Filling in the Blanks, Rosenfeld Media, 2008.

CHAPTER 5

HTML DOM

"JavaScript's global scope is like a public toilet. You can't avoid going in there, but try to limit your contact with surfaces when you do."

– Dmitry Baranovskiy

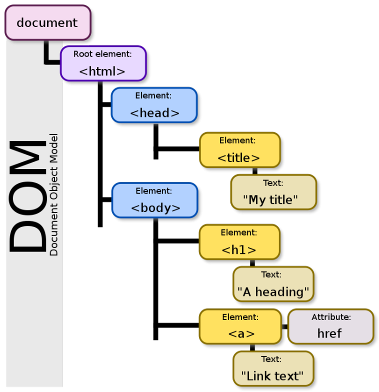
LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Discuss the HTML DOM methods and HTML DOM document
2. Learn HTML DOM elements and changing HTML
3. How to changing CSS and HTML DOM animation
4. Explain the HTML DOM events and HTML DOM eventlistener
5. Describe the HTML DOM navigation and HTML DOM elements (nodes)

INTRODUCTION

The Document Object Model (DOM) is a cross-platform and language-independent interface that treats an XML or HTML document as a tree structure wherein each node is an object representing a part of the document. The DOM represents a document with a logical tree. Each branch of the tree ends in a node, and each node contains objects. DOM methods allow programmatic access to the tree; with them one can change the structure, style or content of a document. Nodes can have event handlers attached to them. Once an event is triggered, the event handlers get executed.



The principal standardization of the DOM was handled by the World Wide Web Consortium (W3C), which last developed a recommendation in 2004. WHATWG took over the development of the standard, publishing it as a living document. The W3C now publishes stable snapshots of the WHATWG standard.

5.1 HTML DOM METHODS

HTML DOM methods are **actions** you can perform (on HTML Elements).

HTML DOM properties are **values** (of HTML Elements) that you can set or change.

Remember

The innerHTML property can be used to get or change any HTML element, including <html> and <body>.

5.1.1 The DOM Programming Interface

The HTML DOM can be accessed with JavaScript (and with other programming languages).

.In the DOM, all HTML elements are defined as **objects**

The programming interface is the properties and methods of each object.

A **property** is a value that you can get or set (like changing the content of an HTML element).

A **method** is an action you can do (like add or deleting an HTML element).

Example

The following example changes the content (the innerHTML) of the <p> element with id=”demo”:

Example

```
<html>
<body>
<p id=”demo”></p>
<script>
document.getElementById(“demo”).innerHTML = «Hello World!»;
</script>
</body>
</html>
```

In the example above, getElementById is a **method**, while innerHTML is a **property**.

5.1.2 The getElementById Method

The most common way to access an HTML element is to use the id of the element.

In the example above the getElementById method used id=”demo” to find the element.

5.1.3 The innerHTML Property

The easiest way to get the content of an element is by using the innerHTML property.

The innerHTML property is useful for getting or replacing .the content of **HTML elements**

Keyword

HTML element is an individual component of an HTML document or web page, once this has been parsed into the Document Object Model.

5.2 HTML DOM DOCUMENT

HTML DOM document object is the owner of all other objects in your web page.

5.2.1 The HTML DOM Document Object

The document object represents your web page.

If you want to access any element in an HTML page, you always start with accessing the document object.

Below are some examples of how you can use the document object to access and manipulate HTML.

5.2.2 Finding HTML Elements

Method	Description
document.getElementById(id)	Find an element by element id
document.getElementsByTagName(name)	Find elements by tag name
document.getElementsByClassName(name)	Find elements by class name

5.2.3 Changing HTML Elements

Method	**Description**
element.innerHTML = new html content	Change the inner HTML of an element
element.attribute = new value	Change the attribute value of an HTML element
element.setAttribute(attribute, value)	Change the attribute value of an HTML element
element.style.property = new style	Change the style of an HTML element

5.2.4 Adding and Deleting Elements

Method	Description
document.createElement(element)	Create an HTML element
document.removeChild(element)	Remove an HTML element

document.appendChild(element)	Add an HTML element
document.replaceChild(element)	Replace an HTML element
document.write(text)	Write into the HTML output stream

5.2.5 Adding Events Handlers

Method	Description
document.getElementById(id).onclick = function(){code}	Adding event handler code to an onclick event

5.2.6 Finding HTML Objects

The first HTML DOM Level 1 (1998), defined 11 HTML objects, object collections, and properties. These are still valid in HTML5.

Later, in HTML DOM Level 3, more objects, collections, and properties were added.

Property	**Description**	**DOM**
document.anchors	Returns all <a> elements that have a name attribute	1
document.applets	Returns all <applet> elements (Deprecated in HTML5)	1
document.baseURI	Returns the absolute base URI of the document	3
document.body	Returns the <body> element	1
document.cookie	Returns the document's cookie	1
document.doctype	Returns the document's doctype	3
document.documentElement	Returns the <html> element	3
document.documentMode	Returns the mode used by the browser	3
document.documentURI	Returns the URI of the document	3
document.domain	Returns the domain name of the document server	1
document.domConfig	Obsolete. Returns the DOM configuration	3
document.embeds	Returns all <embed> elements	3
document.forms	Returns all <form> elements	1

document.head	Returns the <head> element	3
document.images	Returns all <img> elements	1
document.implementation	Returns the DOM implementation	3
document.inputEncoding	Returns the document's encoding (character set)	3
document.lastModified	Returns the date and time the document was updated	3
document.links	Returns all <area> and <a> elements that have a href attribute	1
document.readyState	Returns the (loading) status of the document	3
document.referrer	Returns the URI of the referrer (the linking document)	1
document.scripts	Returns all <script> elements	3
document.strictErrorChecking	Returns if error checking is enforced	3
document.title	Returns the <title> element	1
document.URL	Returns the complete URL of the document	1

```
<html>
<body>

<script type="text/javascript">
document.write("Hello World!");
</script>

</body>
</html>
```

5.3 HTML DOM ELEMENTS

This page you how to find and access HTML elements in an HTML page.

5.3.1 Finding HTML Elements

Often, with JavaScript, you want to manipulate HTML elements. To do so, you have to find the elements first. There are

:a couple of ways to do this

- Finding HTML elements by id
- Finding HTML elements by tag name
- Finding HTML elements by class name
- Finding HTML elements by CSS selectors
- Finding HTML elements by HTML object collections

5.3.2 Finding HTML Element by Id

The easiest way to find an HTML element in the DOM, is by using the element id.

This example finds the element with id=”intro”:

Example

```
var myElement = document.getElementById(“intro”);
```

If the element is found, the method will return the element as an object (in myElement).

If the element is not found, myElement will contain null.

5.3.3 Finding HTML Elements by Tag Name

This example finds all <p> elements:

Example

```
var x = document.getElementsByTagName(“p”);
```

This example finds the element with id=”main”, and then finds all <p> elements inside “main”:

Example

```
var x = document.getElementById(“main”);
    var y = x.getElementsByTagName(“p”);
```

5.3.4 Finding HTML Elements by Class Name

If you want to find all HTML elements with the same class name, use getElementsByClassName().

This example returns a list of all elements with class=”intro”.

Example

```
var x = document.getElementsByClassName(“intro”);
```

Finding elements by class name does not work in Internet Explorer 8 and earlier versions.

Keyword

Internet Explorer is a series of graphical web browsers developed by Microsoft and included in the Microsoft Windows line of operating systems, starting in 1995.

5.3.5 Finding HTML Elements by CSS Selectors

If you want to find all HTML elements that matches a specified CSS selector (id, class names, types, attributes, values of attributes, etc), use the querySelectorAll() method.

This example returns a list of all <p> elements with .“class=”intro

Example

```
var x = document.querySelectorAll(“p.intro”);
```

The querySelectorAll() method does not work in **Internet Explorer** 8 and earlier versions.

5.3.6 Finding HTML Elements by HTML Object Collections

This example finds the form element with id=”frm1”, in the forms collection, and displays all element values:

Example

```
var x = document.forms[“frm1”];
    var text = “”;
    var i;
    for (i = 0; i < x.length; i++) {
;“<text += x.elements[i].value + “<br
    }
    document.getElementById(“demo”).innerHTML = text;
```

5.4 CHANGING HTML

The HTML DOM allows JavaScript to change the content of HTML elements.

5.4.1 Changing the HTML Output Stream

In JavaScript, document.write() can be used to write directly to the HTML output stream:

Example

```
<!DOCTYPE html>
    <html>
    <body>
    <script>
    document.write(Date());
    </script>

    </body>
    </html>
```

5.4.2 Changing HTML Content

The easiest way to modify the content of an HTML element is by using the innerHTML property.

To change the content of an HTML element, use this syntax:

```
document.getElementById(id).innerHTML = new HTML
```

This example changes the content of a <p> element:

Example

```
<html>
<body>
<p id=”p1”>Hello World!</p>
<script>
document.getElementById(“p1”).innerHTML = “New text!”;
</script>
```

```
</body>
</html>
```

Example explained:

- The HTML document above contains a <p> element with id="p1"
- We use the HTML DOM to get the element with id="p1"
- A JavaScript changes the content (innerHTML) of that element to "New text!"

This example changes the content of an <h1> element:

Example

```
<!DOCTYPE html>
<html>
<body>

<h1 id="id01">Old Heading</h1>

<script>
var element = document.getElementById("id01");
element.innerHTML = "New Heading";
</script>

</body>
</html>
```

Keyword

JavaScript often abbreviated as JS, is a high-level, interpreted programming language.

Example explained:

- The HTML document above contains an <h1> element with id="id01"
- We use the HTML DOM to get the element with id="id01"
- A **JavaScript** changes the content (innerHTML) of that element to "New Heading"

5.4.3 Changing the Value of an Attribute

To change the value of an HTML attribute, use this syntax:

document.getElementById(*id*).*attribute* = *new value*

This example changes the value of the src attribute of an <img> element:

Example

```
<!DOCTYPE html>
    <html>
    <body>

    <img id="myImage" src="smiley.gif">

    <script>
    document.getElementById("myImage").src = "landscape.jpg";
    </script>

    </body>
    </html>
```

5.5 CHANGING CSS

The HTML DOM allows JavaScript to change the style of HTML elements.

5.5.1 Changing HTML Style

To change the style of an HTML element, use this syntax:

document.getElementById(id).style.property = new style

The following example changes the style of a <p> element:

Example

```
<html>
<body>

<p id="p2">Hello World!</p>
```

```
<script>
document.getElementById("p2").style.color = "blue";
</script>

    <p>The paragraph above was changed by a script. </p>

    </body>
    </html>
```

5.5.2 Using Events

The HTML DOM allows you to execute code when an event occurs.

Events are generated by the browser when "things happen" to HTML elements:

- An element is clicked on
- The page has loaded
- Input fields are changed

This example changes the style of the HTML element with id="id1", when the user clicks a button:

Example

```
<!DOCTYPE html>
    <html>
    <body>
    <h1 id="id1">My Heading 1</h1>
    <button type="button"
    onclick="document.getElementById('id1').style.color = 'red'">
    Click Me!</button>

    </body>
    </html>
```

5.6 HTML DOM ANIMATION

Learn to create HTML animations using JavaScript.

5.6.1 A Basic Web Page

To demonstrate how to create HTML animations with JavaScript, we will use a simple web page:

Example

```
<!DOCTYPE html>
<html>
<body>

<h1>My First JavaScript Animation</h1>

<div id="animation">My animation will go here</div>

</body>
</html>
```

5.6.2 Create an Animation Container

All animations should be relative to a container element.

Example

```
<div id ="container">
  <div id ="animate">My animation will go here</div>
</div>
```

5.6.3 Style the Elements

The container element should be created with style = "position: relative".

The animation element should be created with style = "position: absolute".

Example

```
#container {
    width: 400px;
    height: 400px;
    position: relative;
    background: yellow;
}
#animate {
    width: 50px;
    height: 50px;
    position: absolute;
    background: red;
}
```

5.6.4 Animation Code

JavaScript animations are done by programming gradual changes in an element's style.

The changes are called by a timer. When the timer interval is small, the animation looks continuous.

The basic code is:

Example

```
var id = setInterval(frame, 5);
function frame() {
    if (/* test for finished */) {
        clearInterval(id);
    } else {
        /* code to change the element style */
    }
}
```

5.6.5 Create the Animation Using JavaScript

Example

```
function myMove() {
    var elem = document.getElementById("animate");
    var pos = 0;
    var id = setInterval(frame, 5);
    function frame() {
        if (pos == 350) {
            clearInterval(id);
        } else {
            pos++;
            elem.style.top = pos + 'px';
            elem.style.left = pos + 'px';
        }
    }
}
```

5.7 HTML DOM EVENTS

HTML DOM allows JavaScript to react to HTML events:

5.7.1 Reacting to Events

A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.

To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute:

onclick=*JavaScript*

Examples of HTML events:

- When a user clicks the mouse
- When a web page has loaded
- When an image has been loaded
- When the mouse moves over an element

- When an input field is changed
- When an HTML form is submitted
- When a user strokes a key

In this example, the content of the <h1> element is changed when a user clicks on it:

Example

```
<!DOCTYPE html>
<html>
<body>
<h1 onclick=”this.innerHTML = ‘Ooops!’”>Click on this text!</h1>
</body>
</html>
```

In this example, a function is called from the event handler:

Example

```
<!DOCTYPE html>
<html>
<body>

<h1 onclick=”changeText(this)”>Click on this text!</h1>

<script>
function changeText(id) {
    id.innerHTML = “Ooops!”;
}
</script>

</body>
</html>
```

5.7.2 HTML Event Attributes

To assign events to HTML elements you can use event attributes.

Example

Assign an onclick event to a button element:

<button onclick=”displayDate()”>Try it</button>

In the example above, a function named displayDate will be executed when the button is clicked.

5.7.3 Assign Events Using the HTML DOM

The HTML DOM allows you to assign events to HTML elements using JavaScript:

Example

Assign an onclick event to a button element:

<script>

document.getElementById(“myBtn”).onclick = displayDate;

</script>

In the example above, a function named displayDate is assigned to an HTML element with the id=”myBtn”.

The function will be executed when the button is clicked.

Keyword

Web Page is a document that is suitable for the World Wide Web and web browsers.

5.7.4 The onload and onunload Events

The onload and onunload events are triggered when the user enters or leaves the page.

The onload event can be used to check the visitor’s browser type and browser version, and load the proper version of the **web page** based on the information.

The onload and onunload events can be used to deal with cookies.

Example

<body onload=”checkCookies()”>

5.8 HTML DOM EVENTLISTENER

The addEventListener() method attaches an event handler to the specified element.

5.8.1 The addEventListener() method

Example

Add an event listener that fires when a user clicks a button:

document.getElementById("myBtn").addEventListener("click", displayDate);

The addEventListener() method attaches an event handler to the specified element.

The addEventListener() method attaches an event handler to an element without overwriting existing event handlers.

You can add many event handlers to one element.

You can add many event handlers of the same type to one element, i.e two "click" events.

You can add event listeners to any DOM object not only HTML elements. i.e the window object.

The addEventListener() method makes it easier to control how the event reacts to bubbling.

When using the addEventListener() method, the JavaScript is separated from the HTML markup, for better readability and allows you to add event listeners even when you do not control the HTML markup.

You can easily remove an event listener by using the removeEventListener() method.

5.8.2 Syntax

element.addEventListener(event, function, useCapture);

The first parameter is the type of the event (like "click" or "mousedown").

The second parameter is the function we want to call when the event occurs.

The third parameter is a boolean value specifying whether to use event bubbling or event capturing. This parameter is optional.

5.8.3 Add an Event Handler to an Element

Example

Alert "Hello World!" when the user clicks on an element:

element.addEventListener("click", function(){ alert("Hello World!"); });

You can also refer to an external "named" function:

Example

Alert “Hello World!” when the user clicks on an element:

```
element.addEventListener(“click”, myFunction);
function myFunction() {
    alert (“Hello World!”);
}
```

5.8.4 Add Many Event Handlers to the Same Element

The addEventListener() method allows you to add many events to the same element, without overwriting existing events:

Example

```
element.addEventListener(“click”, myFunction);
element.addEventListener(“click”, mySecondFunction);
```

:You can add events of different types to the same element

Example

```
element.addEventListener(“mouseover”, myFunction);
element.addEventListener(“click”, mySecondFunction);
element.addEventListener(“mouseout”, myThirdFunction);
```

Keyword

HTML is the standard markup language for creating web pages and web applications.

5.8.5 Add an Event Handler to the Window Object

The addEventListener() method allows you to add event listeners on any HTML DOM object such as HTML elements, the **HTML** document, the window object, or other objects that support events, like the xmlHttpRequest object.

Example

Add an event listener that fires when a user resizes the window:

```
window.addEventListener("resize", function(){
    document.getElementById("demo").innerHTML = sometext;
});
```

5.8.6 Passing Parameters

When passing parameter values, use an "anonymous function" that calls the specified function with the parameters:

Example

```
element.addEventListener("click", function(){ myFunction(p1, p2); });
```

5.8.7 Event Bubbling or Event Capturing?

There are two ways of event propagation in the HTML DOM, bubbling and capturing.

Event propagation is a way of defining the element order when an event occurs. If you have a <p> element inside a <div> element, and the user clicks on the <p> element, which element's "click" event should be handled first?

In bubbling the inner most element's event is handled first and then the outer: the <p> element's click event is handled first, then the <div> element's click event.

In capturing the outer most element's event is handled first and then the inner: the <div> element's click event will be handled first, then the <p> element's click event.

With the addEventListener() method you can specify the propagation type by using the "useCapture" parameter:

```
addEventListener(event, function, useCapture);
```

The default value is false, which will use the bubbling propagation, when the value is set to true, the event uses the capturing propagation.

Example

```
document.getElementById("myP").addEventListener("click", myFunction, true);
document.getElementById("myDiv").addEventListener("click", myFunction, true);
```

5.8.8 The removeEventListener() method

The removeEventListener() method removes event handlers that have been attached with the addEventListener() method:

Example

element.removeEventListener("mousemove", myFunction);

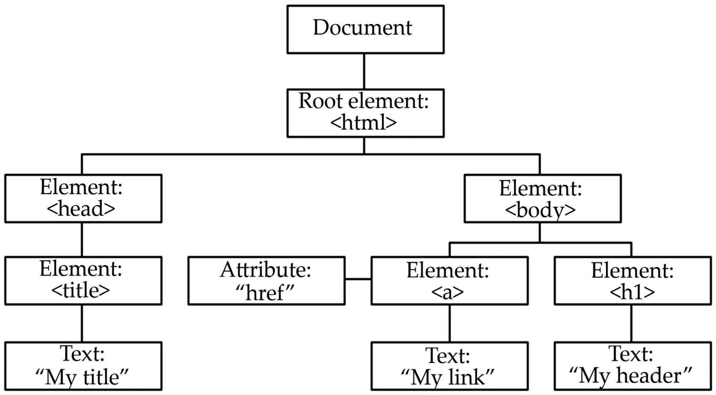
5.9 HTML DOM NAVIGATION

With the HTML DOM, you can navigate the node tree using node relationships.

5.9.1 DOM Nodes

According to the W3C HTML DOM standard, everything in an HTML document is a **node**:

- The entire document is a document node
- Every HTML element is an element node
- The text inside HTML elements are text nodes
- Every HTML attribute is an attribute node (deprecated)
- All comments are comment nodes



New nodes can be created, and all nodes can be modified .or deleted

5.9.2 Node Relationships

The nodes in the node tree have a hierarchical relationship to each other.

The terms parent, child, and sibling are used to describe the relationships.

Did You Know?

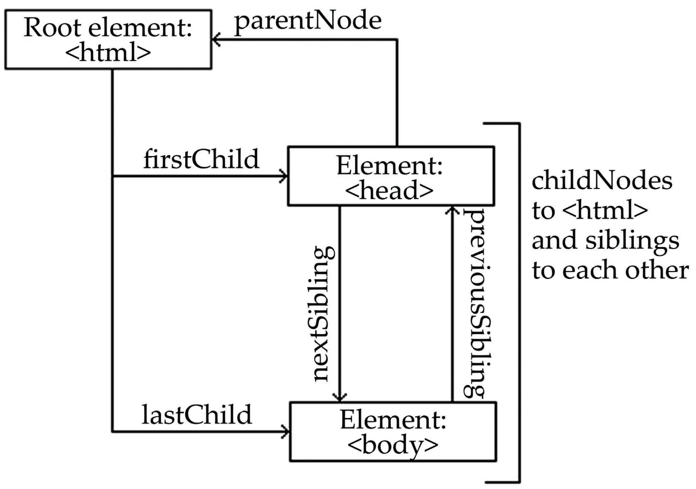
The history of the Document Object Model is intertwined with the history of the "browser wars" of the late 1990s between Netscape Navigator and Microsoft Internet Explorer, as well as with that of JavaScript and JScript, the first scripting languages to be widely implemented in the JavaScript engines of web browsers.

Keyword

Node is a basic unit of a data structure, such as a linked list or tree data structure.

- In a node tree, the top node is called the root (or root node)
- Every node has exactly one parent, except the root (which has no parent)
- A node can have a number of children
- Siblings (brothers or sisters) are nodes with the same parent

```
<html>

  <head>
      <title>DOM Tutorial</title>
  </head>

  <body>
      <h1>DOM Lesson one</h1>
      <p>Hello world!</p>
  </body>

</html>
```



From the HTML above you can read:

- <html> is the root node
- <html> has no parents
- <html> is the parent of <head> and <body>

- <head> is the first child of <html>
- <body> is the last child of <html>

and:

- <head> has one child: <title>
- <title> has one child (a text node): "DOM Tutorial"
- <body> has two children: <h1> and <p>
- <h1> has one child: "DOM Lesson one"
- <p> has one child: "Hello world!"
- <h1> and <p> are siblings

5.9.3 Child Nodes and Node Values

A common error in DOM processing is to expect an element node to contain text.

Example:

```
<title id="demo">DOM Tutorial</title>
```

The element node <title> (in the example above) does not contain text.

It contains a text node with the value "DOM Tutorial".

The value of the text node can be accessed by the node's innerHTML property:

```
var myTitle = document.getElementById("demo").innerHTML;
```

Accessing the innerHTML property is the same as accessing the nodeValue of the first child:

```
var myTitle = document.getElementById("demo").firstChild.nodeValue;
```

Accessing the first child can also be done like this:

```
var myTitle = document.getElementById("demo").childNodes[0].nodeValue;
```

All the (3) following examples retrieves the text of an <h1> element and copies it into a <p> element:

Example

```
<html>
<body>

<h1 id="id01">My First Page</h1>
```

```
<p id=”id02”></p>
<script>
document.getElementById(“id02”).innerHTML = document.getElementById(“id01”).
innerHTML;
</script>

</body>
</html>
```

Example

```
<html>
<body>

<h1 id=”id01”>My First Page</h1>
<p id=”id02”></p>

<script>
document.getElementById(“id02”).innerHTML = document.getElementById(“id01”).
firstChild.nodeValue;
</script>
</body>
</html>
```

Example

```
<html>
<body>

<h1 id=”id01”>My First Page</h1>
<p id=”id02”>Hello!</p>

<script>
document.getElementById(“id02”).innerHTML = document.getElementById(“id01”).
childNodes[0].nodeValue;
</script>
```

```
</body>
</html>
```

5.9.4 DOM Root Nodes

There are two special properties that allow access to the full document:

- document.body - The body of the document
- document.documentElement - The full document

Example

```
<html>
<body>

<p>Hello World!</p>
<div>
<p>The DOM is very useful!</p>
<p>This example demonstrates the <b>document.body</b> property.</p>
</div>

<script>
alert(document.body.innerHTML);
</script>
</body>
</html>
```

Example

```
<html>
<body>

<p>Hello World!</p>
<div>
<p>The DOM is very useful!</p>
<p>This example demonstrates the <b>document.documentElement</b> property.</
p>
```

```
</div>

<script>
alert(document.documentElement.innerHTML);
</script>

</body>
</html>
```

5.9.5 The nodeName Property

The nodeName property specifies the name of a node.

- nodeName is read-only
- nodeName of an element node is the same as the tag name
- nodeName of an attribute node is the attribute name
- nodeName of a text node is always #text
- nodeName of the document node is always #document

Example

```
<h1 id="id01">My First Page</h1>
<p id="id02"></p>

<script>
document.getElementById("id02").innerHTML = document.getElementById("id01").
nodeName;
</script>
```

5.9.6 The nodeValue Property

The nodeValue property specifies the value of a node.

- nodeValue for element nodes is undefined
- nodeValue for text nodes is the text itself
- nodeValue for attribute nodes is the attribute value

5.9.7 The nodeType Property

The nodeType property is read only. It returns the type of a node.

Example

```
<h1 id="id01">My First Page</h1>
p id="id02"></p>

<script>
document.getElementById("id02").innerHTML = document.getElementById("id01").
nodeType;
</script>
```

5.10 HTML DOM ELEMENTS (NODES)

Adding and Removing Nodes (HTML Elements)

5.10.1 Creating New HTML Elements (Nodes)

To add a new element to the HTML DOM, you must create the element (element node) first, and then append it to an existing element.

Example

```
<div id="div1">
<p id="p1">This is a paragraph. </p>
<p id="p2">This is another paragraph. </p>
</div>

<script>
var para = document.createElement("p");
var node = document.createTextNode("This is new.");
para.appendChild(node);
var element = document.getElementById("div1");
element.appendChild(para);
</script>
```

Example Explained

This code creates a new <p> element:

```
var para = document.createElement("p");
```

To add text to the <p> element, you must create a text node first. This code creates a text node:

```
var node = document.createTextNode("This is a new paragraph.");
```

Then you must append the text node to the <p> element:

```
para.appendChild(node);
```

Finally you must append the new element to an existing element.

This code finds an existing element:

```
var element = document.getElementById("div1");
```

This code appends the new element to the existing element:

```
element.appendChild(para);
```

5.10.2 Creating new HTML Elements - insertBefore()

The appendChild() method in the previous example, appended the new element as the last child of the parent.

If you don't want that you can use the insertBefore() method:

Example

```
<div id="div1">
<p id="p1">This is a paragraph. </p>
<p id="p2">This is another paragraph. </p>
</div>
<script>
var para = document.createElement("p");
var node = document.createTextNode("This is new.");
para.appendChild(node);

var element = document.getElementById("div1");
var child = document.getElementById("p1");
element.insertBefore(para, child);
</script>
```

5.10.3 Removing Existing HTML Elements

To remove an HTML element, you must know the parent of the element:

Example

```
<div id="div1">
<p id="p1">This is a paragraph. </p>
<p id="p2">This is another paragraph. </p>
</div>

<script>
var parent = document.getElementById("div1");
var child = document.getElementById("p1");
parent.removeChild(child);
</script>
```

5.10.4 Replacing HTML Elements

To replace an element to the HTML DOM, use the replaceChild() method:

Example

```
<div id="div1">
<p id="p1">This is a paragraph. </p>
<p id="p2">This is another paragraph. </p>
</div>
<script>
var para = document.createElement("p");
var node = document.createTextNode("This is new.");
para.appendChild(node);
var parent = document.getElementById("div1");
var child = document.getElementById("p1");
parent.replaceChild(para, child);
</script>
```

CASE STUDY

DOCUMENT OBJECT MODEL

The history of the Document Object Model, known as the DOM, is tightly coupled with the beginning of the JavaScript and JScript scripting languages.

JavaScript

The LiveScript language was designed at Netscape Communications to make the Java support in Netscape Navigator more accessible to non-Java programmers. LiveScript, like any scripting language, is a loosely-typed language. It is intended for a large audience of Web designers and developers.

In December 1995, LiveScript was renamed JavaScript and released as part of Netscape Navigator 2.0. Except for marketing purposes, JavaScript has nothing to do with the Java language developed and maintained by Sun Microsystems. The Web community started then to manipulate the content of Web documents, in order to bring interactivity and typography to the formerly static Web.

JScript

In July 1996, Microsoft released Internet Explorer 3.0 with a port of JavaScript called JScript.

ECMAScript

In June 1997, ECMA adopted a hybrid version of the scripting languages called ECMAScript. The International Organization for Standardization (ISO) followed suit in 1998. Unfortunately, ECMAScript arrived too late for the 4.0 releases of Netscape Navigator and Internet Explorer. Each introduced their own document object model, DHTML and dHTML, that came to be called Dynamic HTML.

ECMA-262, released in December 1999, is still not followed by Microsoft and their Internet Explorer. Netscape claims to support ECMA-262 in Netscape Navigator versions 6 and 7.

The World Wide Web Consortium

In 1994, Tim Berners-Lee, inventor of the World Wide Web, created the World Wide Web Consortium (W3C) to lead the Web to its full potential. At the beginning of 1997, the companies involved in this consortium — including Netscape Communications and Microsoft — decided to find a consensus around their object models to access and manipulate documents. While trying to stay as backward compatible as possible with the original browser object models, the W3C's Document Object Model (DOM) provided a better object representation of HTML documents.

HTML and XML

In 1996, a new markup language, the Extensible Markup Language (XML), was developed in the W3C as well. Meant to remove the HTML language's extensibility restrictions, the idea of developing an object model for XML quickly became another goal of the DOM effort.

SUMMARY

- The Document Object Model (DOM) is a cross-platform and language-independent interface that treats an XML or HTML document as a tree structure wherein each node is an object representing a part of the document. The DOM represents a document with a logical tree. Each branch of the tree ends in a node, and each node contains objects.
- HTML DOM methods are actions you can perform (on HTML Elements).
- HTML DOM properties are **values** (of HTML Elements) that you can set or change.
- A property is a value that you can get or set (like changing the content of an HTML element).
- HTML DOM document object is the owner of all other objects in your web page.
- The HTML DOM allows JavaScript to change the content of HTML elements.
- The HTML DOM allows JavaScript to change the style of HTML elements.
- JavaScript animations are done by programming gradual changes in an element's style.

KNOWLEDGE CHECK

1. **What is the reason for avoiding the attributes property in the HTML DOM?**
   a. Found unnecessary
   b. Attributes don't have attributes
   c. Attributes have attributes
   d. None of the mentioned
2. **What is the purpose of the method nodeMap.setNamedItem()?**
   a. Sets ID
   b. Sets attribute node
   c. Sets element name
   d. None of the mentioned
3. **How is everything treated in HTML DOM?**
   a. Node
   b. Attributes
   c. Elements
   d. All of the mentioned
4. **What does the NamedNodeMap object represent in the HTML DOM?**
   a. Unordered collection of elements
   b. Unordered collection of attributes
   c. Unordered collection of nodes
   d. All of the mentioned
5. **What is the purpose of the Attr object in the HTML DOM?**
   a. Used to focus on a particular part of the HTML page
   b. HTML Attribute
   c. Used to arrange elements
   d. None of the mentioned
6. **What is DOM?**
   a. Dynamic Object Model
   b. Document Object Model
   c. Distributed Object Model
   d. None of these

7. In how many defferent parts is the DOM divided ?

a. 2

b. 4

c. 3

d. 1

8. You can find the element you want to manipulate in ________ way ?

a. getElementById()

b. getElementsByTagName()

c. All of these

d. None of these

9. Every node has some properties that contain some information about the node. The properties are__________ .

a. nodeName

b. nodeValue

c. nodeType

d. All of these.

10. The History object is actually a ________ object.

a. JavaScript

b. HTML DOM

c. XML DOM

d. Core DOM

REVIEW QUESTIONS

1. Use HTML DOM to change the value of the input field.
2. Use HTML DOM to change the value of the image's SRC attribute.
3. Change the (text) color of the p element to "red".
4. Use the eventListener to assign an onclick event to the button element.
5. Write the convenient function removeElement which removes the DOM node it is given as an argument from its parent node.

Check Your Result

1. (b)	2. (b)	3. (a)	4. (d)	5. (b)
6. (b)	7. (c)	8. (c)	9. (d)	10. (a)

REFERENCES

1. David Flanagan. JavaScript: The Definitive Guide, Sixth Edition. O'Reilly. 2011.
2. Douglas Crockford. JavaScript: The good parts. O'Reilly Media, 2008.
3. Douglas Crockford. Javascript: The world's most misunderstood programming language.
4. Douglas Crockford. JSLint: The JavaScript verifier.http://www.jslint.com/, 2002. Accessed August 4, 2008.
5. Jonathan Stark. *Building iPhone Apps with HTML, CSS, and JavaScript*. O'Reilly. 2010.
6. Maximiliano Firtman. Programming the Mobile Web. O'Reilly. 2010.

CHAPTER 6

COOKIES

"The strength of JavaScript is that you can do anything. The weakness is that you will."

– Reg Braithwaite

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Understand the basics to reading/writing cookies with JavaScript
2. Explain the setting different cookie kinds in JavaScript

INTRODUCTION

A cookie is an item of data that a web server saves to your computer's hard disk via a web browser. A cookie is a piece of data that is stored on your computer to be accessed by your browser. You also might have enjoyed the benefits of cookies knowingly or unknowingly. Have you ever saved your Facebook password so that you do not have to type it each and every time you try to login? If yes, then you are using cookies. Cookies are saved as key/value pairs.

A computer cookie, also referred to as an "HTTP cookie," is a small text file that contains a unique ID tag, placed on the user's computer by a website. In this file, various information can be stored, from pages visited on the site, to information voluntarily given to the site. These tiny files provide practical benefits to both users and website operators, and generally make surfing the net a smoother experience than it otherwise would be. Nevertheless, privacy advocates tend to be wary of them, since many users are unaware of exactly what information is collected, and how the information may be used or shared.

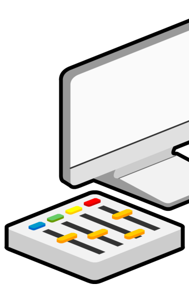
The communication between a web browser and server happens using a stateless protocol named HTTP. Stateless protocol treats each request independent. So, the server does not keep the data after sending it to the browser. But in many situations, the data will be required again. Here come cookies into a picture. With cookies, the web browser will not have to communicate with the server each time the data is required. Instead, it can be fetched directly from the computer.

Keyword

Web browser is a software application for accessing information on the World Wide Web.

6.1 BASICS TO READING/WRITING COOKIES WITH JAVASCRIPT

Cookies are relatively small text files that a **web browser** embeds on a user's computer. Cookies allow otherwise stateless HTTP communications to emulate state (i.e., memory). Cookies are being replaced by somewhat newer technologies such as local storage and session storage; however, cookies are still widely used by many major websites today. For that reason alone, it is a good idea for you to familiarize yourself with how cookies work. Additionally, it is fun to see how you yourself can use JavaScript to read from and write to the your browser's cookie API.



One user registration ends after completing many pages. But how to maintain users' session information across all the web pages.

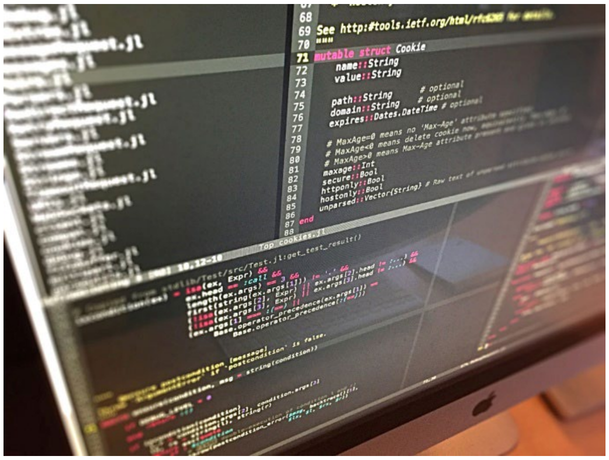
Cookies were originally invented by Netscape to give 'memory' to web servers and browsers. The HTTP protocol, which arranges for the transfer of web pages to your browser and browser requests for pages to servers, is *state-less*, which means that once the server has sent a page to a browser requesting it, it does not remember a thing about it. So if you come to the same web page a second, third, hundredth or millionth time, the server once again considers it the very first time you ever came there. This can be annoying in a number of ways. The server cannot remember if you identified yourself when you want to access protected pages, it cannot remember your user preferences, it cannot remember anything. As soon as personalization was invented, this became a major problem. Cookies were invented to solve this problem. There are other ways to solve it, but cookies are easy to maintain and very versatile.

Did You Know?

The computer cookie dates back to 1994. In that year, it was adapted as a tool for the World Wide Web by Leo Montulli from a similar technique, called "magic cookie," which was used in UNIX® systems. This is also the origin of the term itself. It was not for another couple of years, however, that the cookies became widely known to the general public.

6.1.1 Using Cookies in JavaScript

Cookies are the name given to the small text files your browser stores on your computer, which contain information relevant to the sites you have visited in the past. Using JavaScript you can write to these text files and then extract data from them whenever your reader returns to your site. Cookies are variables that can be stored on a user's computer and be picked up by any other web pages in the correct domain and path. Cookies are set to expire after a certain length of time. They are limited to storing string values only. Be warned that many users (including me) will not permit cookies on their computers. Do not make your web sites rely on them. The reason for this is that many web sites only use cookies as part of advert tracing systems, which they use to track your movement around the Internet. We would not want anyone to follow me around a city while you was shopping, taking notes of every shop we visit and whether you look in the lingerie section, as that would be an invasion of our privacy. Many people feel the same applies to the Internet. You may find it helps to firstly display a message saying what you are going to use the cookie for, for example to store a username and password for the next time they visit the site.



```
See http:#tools.ietf.org/html/rfc6265 for details.
"""
mutable struct Cookie
    name::String
    value::String

    path::String            # optional
    domain::String          # optional
    expires::Dates.DateTime # optional

    # MaxAge=0 means no 'Max-Age' attribute specified.
    # MaxAge<0 means delete cookie now, equivalently 'Max-Age: 0'
    # MaxAge>0 means Max-Age attribute present and given in seconds
    maxage::Int
    secure::Bool
    httponly::Bool
    hostonly::Bool
    unparsed::Vector{String} # Raw text of unparsed attribute-value pairs
end
```

Note also that European law requires sites to gain explicit permission before using cookies, unless those cookies are essential to the operation of the site (such as a shopping basket). Some browsers will have a limit to how many cookies they can store, usually 300 cookies or more, of which there may be 20 cookies per domain name. A total of 4 KB (after encoding) of cookies can be stored for any domain or path. The document.cookie object is a string representation of all cookies available to the current web page. The document.cookie object is somewhat unusual in that when you assign string values to it, it does not become the value that you assign to it. Instead, it takes a part of that value and appends its current value to that, so making a string containing several pairs of variableName=variableValue.

Why we use Cookies

Cookies are necessary because the HTTP protocol that is used to transfer webpages around the web is *state-less*. This means that web servers cannot remember information about users throughout their travels, and so everyone becomes anonymous. If you ever return to a site you have visited previously, you are treated as if it was your first visit. This is especially unsatisfactory for sites which ask their users to log in — if you leave and return just a few minutes later, you will have to log in again. The server does not remember anything about your visit or your preferred settings. So, cookies were invented to give memory, of a sort, to web servers.

What Kinds of Data Can Be Stored in a Cookie?

A cookie is basically a string of text characters not longer than 4 KB. Cookies are set in name=value pairs, separated by semi-colons. For example, a cookie might be a string like the following:

"theme=blue; max-age=60; path=/; domain=thesitewizard.com"

This example cookie has 4 variable/value pairs:

- **max-age**, which is set to 60,
- **path**, which is set to the slash character "/",
- **domain**, which is set to "thesitewizard.com",
- and **theme**, which is set to "blue".

The variables "max-age", "path" and "domain" are special variable names that are recognized by the browser to control things like the lifespan of the cookie and the URLs for which the cookie is valid. Only the "theme" variable in your example contains the real data that you wish to set. You can create any variable name you want, and set it to whatever value you wish, subject to the following constraints:

- max-age

Cookies have, by default, a lifespan of the current browser session. As soon as your visitor closes his browser, your cookie disappears. To make it last longer, you will need to set the max-age variable to contain the number of seconds (yes, seconds) you want the cookie to last.

For example, if you want your cookie to last 30 days, set it to 2,592,000. Actually instead of pre-calculating this and putting it into your script, you can have the JavaScript interpreter calculate it for you at run time, and simply encode it as

"theme=blue; max-age=" + 60*60*24*30 + "; path=/; domain=thesitewizard.com"

This is superior to writing a huge number that you will forget the meaning of in the future.

- path

By default cookies are valid only for web pages in the directory of the current web page that stored them, as well as its descendants. That is, if a cookie is set by http://example.com/abc/webpage.html, it will be valid for http://example.com/abc/yet-another-page.html as well as http://example.com/abc/Sub-Folder/index.html, but not for http://example.com/index.html.

If you want the cookie to be valid in some other directory, say, http://example.com/special/, you will need to set the path variable to contain the value "/special". If you want the cookie to be valid everywhere on your site, set it to the root of your web directory, that is, "/".

- domain

Another special variable name that you may want to take note of is the domain variable. Cookies set in sub-domains like www.example.com will only be valid for that subdomain. If you want it to be valid for all sub-domains of example.com, you will need to set the domain to point to "example.com". The cookie will then be valid for "www.example.com", "blog.example.com", and whatever other subdomains that you may have.

Remember that for security reasons, if your domain is example.com, browsers will not accept a cookie for a different domain, like google.com.

- secure

There's another variable that has special meaning: secure. This variable should not be assigned any value. Including it means that the cookie will only be sent if your visitor is visiting your website over a secure connection.

- expires

The expires variable is obsolete although still supported by today's browsers. Use the max-age variable instead, since it is easier to use. Be careful not to use "expires" as a variable name to store your data as well.

- No spaces, commas, semi-colons

Your cookie values cannot have any embedded whitespaces, commas or semi-colons. If you have any, they must be converted to its "encoded" equivalent. The easiest way to do this is to use the encodeURIComponent() function to encode it, and the decodeURIComponent() function to decode it when you read the cookie.

Expanding on your earlier example, if you want to set a "theme" variable to "blue theme", you can do it this way:

"theme=" + encodeURIComponent("blue theme") + "; max-age=" + 60*60*24*30 + "; path=/; domain=thesitewizard.com"

Of course in the above case, since there is only one space character to encode, you can do it manually as "blue%20 theme" as well.

- Cookie Limits

Although different browsers may implement different limits for cookies, the bare minimum that they are supposed to support is as follows:

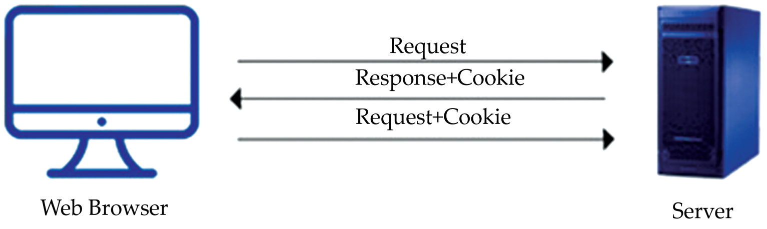
- Cookie length: 4 KB. The total length of your string, including all the variables with special meaning, should not be more than 4,096 characters.
- Maximum number of cookies per web server: 20.
- Total number of cookies supported by the browser: 300. This includes cookies stored by other websites.

6.1.2 Structure of a Cookie

Cookies are no more than simple text files — usually found in your browser cache, or 'Temporary Internet files' — which contain one or more entries. Each entry is made up of

- A name-value pair which stores whatever data you want to save.
- An expiry date, after which time the entry will be deleted.
- The web domain and path that the entry should be associated with.

You can use JavaScript to read or write a new entry to the cookie file. The process of creating an entry is often referred to as 'writing a cookie', but this is misleading. The cookie is the text file which *contains* all of your entries, while the individual entries themselves hold the data. Each domain name on the web can have a cookie file associated with it, and each cookie can hold multiple entries.



When you request a file from a server that you have used previously, the data in the relevant cookie is sent to the server along with your request. This way, server-side scripts, such as those written in Perl or » **PHP**, can read your cookie and figure out whether you have permission to view a certain page, for instance. Cookies can also be used for somewhat more malicious purposes, usually by advertising companies to track your behavior online. Most modern browsers include good measures that allow you to block cookies from certain sites, so which sites you disclose information to is now at your discretion.

The name-value pair part of the entry is very similar to declaring a variable — when you want to retrieve information you ask for the value that is associated with a name that you provide. The expiry date is expressed in an unfriendly UTC format; though fortunately there are methods for generating a suitable date. If a date is not set, the entry is deleted when you close your browser.

The domain and path that you associate your cookie with have to be part of the same domain that your site belongs to. For instance, you can set your cookie to be active for www.yourhtmlsource.com, the default; or to yourhtmlsource.com, which will cover any and all subdomains you set up for the site. You cannot, however, set it to yahoo.com, for obvious security reasons. Using the path you can restrict a cookie to be valid for only a certain directory. Usually you will want it to be available to any page in the domain, so this is set to /, the root directory.

6.1.3 Setting, Reading and Erasing Cookies

The document has an object in JavaScript called document.cookie, which is used to read and retrieve cookie data. It is a repository of Strings (though not an array). You can create new entries, read out an existing name-value pair, or erase an entry through JavaScript. To create a cookie for your site, you write

document.cookie =

“testvalue1=Yes; expires=Fri, 13 Jul 2004 05:28:21 UTC; path=/”;

The whole entry is supplied as one quoted String with segments set apart with semicolons — first the name-value pair, then the expiry date in the correct format, and finally the path. This syntax is fixed, and you should not go rearranging the elements. Write this entry now. To test out the cookie›s contents, we can use a simple script like

alert(document.cookie);

Which will yield this result. You may see another value in there too, which is used by our Stylesheet-switcher. Now we can write another entry to the cookie, using a different name, as so:

document.cookie =

“testvalue2=Nah; expires=Fri, 13 Jul 2004 05:28:21 UTC; path=/”;

Checking on the cookie›s contents now, we can see that our first value is still in there. Had we used the same name, the first value would have been overwritten; but since we used a different name, the new entry has been added in with the first.

Erasing cookie entries is easy — just set a new value and give an expiry date before today, as in

document.cookie =

“testvalue2=Whatever; expires=Fri, 13 Jul *2001* 05:28:21 UTC; path=/”;

You can also give the entry an expiry date of -1, and it will be erased immediately. Erase test values 1 and 2 now, if you have the heart.

6.1.4 Convenient Scripts

To easily tinker with cookies ourselves, we will be using some great scripts which were originally coded by » Scott Andrew. They will take much of the pain out of the process; especially reading the values out of a cookie, which is a bit complicated. Here are the functions:

```
function createCookie(name, value, days)
{
  if (days) {
    var date = new Date();
    date.setTime(date.getTime()+(days*24*60*60*1000));
    var expires = “; expires=”+date.toGMTString();
    }
  else var expires = “”;
  document.cookie = name+”=”+value+expires+”; path=/”;
}

function readCookie(name)
{
  var ca = document.cookie.split(‘;’);
  var nameEQ = name + “=”;
  for(var i=0; i < ca.length; i++) {
    var c = ca[i];
    while (c.charAt(0)==’ ‘) c = c.substring(1, c.length); //delete spaces
    if (c.indexOf(nameEQ) == 0) return c.substring(nameEQ.length, c.length);
    }
  return null;
}

function eraseCookie(name)
{
  createCookie(name, “”, -1);
}
```

These are some nicely coded scripts, and do not require too much explanation. The function we use to create cookies takes three arguments, which make up the name-value pair and the amount of days to retain the cookie. The last argument is converted into a valid date by adding its value in hours to the current time before being annexed into the line which creates the cookie.

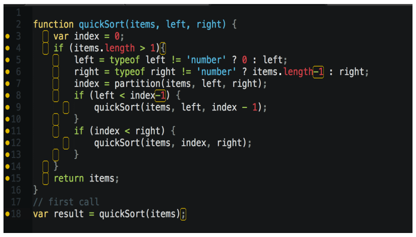
The cookie reading function is the most difficult one here. First it splits the available cookie String (what we have been reading out in the alert earlier on this page) at every occurrence of the separating semicolon. This creates a new array, with each index holding an entry pair. We **loop** through these looking for the String 'name='. When we find this, we read out whatever else makes up this index, which will be the value associated with the name we passed to the function at the beginning. Erasing an entry is easy — simply recreate a cookie with its expiration date set to -1.

Keyword

Loop is a programming function that iterates a statement or condition based on specified boundaries.

6.1.5 How to set Cookies with JavaScript

Cookies are an important part of modern browsers. Without them, we could not browse websites that require authentication, such as social networks since we would be asked for our password on every page we would browse. We would not be able to write a simple e-mail, or purchase stuff online. Website usage would be limited to browsing only static websites. In this section we will focus only on creating and editing cookies using jQuery.



```

function quickSort(items, left, right) {
    var index = 0;
    if (items.length > 1){
        left = typeof left != 'number' ? 0 : left;
        right = typeof right != 'number' ? items.length-1 : right;
        index = partition(items, left, right);
        if (left < index-1) {
            quickSort(items, left, index - 1);
        }
        if (index < right) {
            quickSort(items, index, right);
        }
    }
    return items;
}
// first call
var result = quickSort(items);
```

Step 1 - HTML first

We create a DIV and inside it we add two messages. One that will be displayed only once, when the page is loaded, and the other will be displayed after the first one was shown. Whether to show the first or second message is the job of CSS:

```
<div class=”message  change-message--on-load  hide--second”>
  <p>This message is displayed only the first time you visit this page. Refresh your page to hide it!</p>
  <p>This is shown only after the before message was shown in the last visit. Even when you refresh the page, the browser remembers your option.</p>
</div>
```

Step 2 - CSS

With CSS we tell the browser to hide the first message if the div’s class is .hide--first or hide the second message if the div’s class is .hide--second:

```
.hide--first > *:first-child {
  display: none;
}
.hide--second > *:last-child {
display: none;
}
```

Step 3 - Initializing

For faster loading times, add your JavaScript to the bottom of the page, before closing the </body> tag. First, we need to call jQuery.

```
<script type=”text/javascript” src=”http://ajax.googleapis.com/ajax/libs/jquery/1.6.4/jquery.min.js”></script>
```

Next we call the cookie script. Make sure you add the correct URL pointing to the script.

```
<script type=”text/javascript” src=”cookie.js”></script>
```

Below it, we add an empty script tag and we can start coding:

```
<script type=”text/JavaScript”>
</script>
```

Step 4 - JavaScript

First we add the cookie code. There are two parts to this: one that checks if cookie exists, and the other part is the one that adds the cookie. First, we will check for the cookie. If the cookie is true, hide the initial message and show the other one (with CSS) by changing the class of the <div>:

```
if ($.cookie('hide-after-load') == 'yes') {
  $('.change-message--on-load').removeClass('hide--second');
  $('.change-message--on-load').addClass('hide--first');
}
```

Before we close the script tag, we have to add the cookie that will hide the first message. We add it to the end, because if we were to add it before the code that checks (see above) the cookie, it would hide the first message from the start. Adding it at the end ensures that the message will be hidden the next time the page is loaded. It is set to expire in 7 days.

```
$.cookie('hide-after-load', 'yes', {expires: 7 });
```

The complete code for the first example:

```
<script type="text/JavaScript">
    $(document).ready(function() {
            if ($.cookie('hide-after-load') == 'yes') {
              $('.change-message--on-load').removeClass('hide--second');
              $('.change-message--on-load').addClass('hide--first');
            }

            $.cookie('hide-after-load', 'yes', {expires: 7 });
    });
</script>
```

Step 5 - Add cookie on click

In the demo page, you saw that the second container would hide the first message and show the other one, only after you clicked on the "×" icon. First we need to add an empty href tag for the icon to our HTML:

<p>You can only hide this message, by clicking the &times; on the right of this box <a href="#" class="close" title="Hide This Message">&times;</a></p>

To position the "×" icon to the top-right of the container, we use the absolute inside relative container trick in the CSS:

```
.message {
    position: relative
}
```

```
.close {
  color: #f00;
  position: absolute;
  text-transform: lowercase;
  right: 20px;
  font-size: 1.5em;
  top: 10px;
  line-height: 1;
  border: none !important;
}
```

Inside the JavaScript tag, we add the code to do something once the icon is clicked. In this case, return nothing so that the URL is not populated with the empty href hash (#) symbol:

```
$('.close').click(function() {
  return false;
})
```

Once the user has clicked on the icon, we need to check whether the parent container of the icon is displaying the first message or the second. If it displays the first message, hide it by changing its class. Finally, we also add a cookie with the variable yes so that this option is remembered next time.

```
if (!$('.change-message--on-click').is('hide--first')) {
  $('.change-message--on-click').removeClass('hide--second');
  $('.change-message--on-click').addClass('hide--first');

  // add cookie setting that user has clicked
  $.cookie('hide-after-click', 'yes', {expires: 7 });
}
```

The complete script when clicking the icon looks like this:

```
  $('.close').click(function() {
    if (!$('.change-message--on-click').is('hide--first')) {
      $('.change-message--on-click').removeClass('hide--second');
      $('.change-message--on-click').addClass('hide--first');
      $.cookie('hide-after-click', 'yes', {expires: 7 });
    }
    return false;
  })
```

But wait, we are not done! We also need to check if the clicked cookie has the yes variable attached to it. If it does, show the second message, and hide the first:

```
if ($.cookie('hide-after-click') == 'yes') {
  $('.change-message--on-click').removeClass('hide--second');
  $('.change-message--on-click').addClass('hide--first');
}
```

The complete code for the second example:

```
<script type="text/JavaScript">
  $(document).ready(function() {
    // COOKIES
    // if the cookie is true, hide the initial message and show the other one
    if ($.cookie('hide-after-click') == 'yes') {
      $('.change-message--on-click').removeClass('hide--second');
      $('.change-message--on-click').addClass('hide--first');
    }

    // when clicked on "X" icon do something
    $('.close').click(function() {
      // check that "X" icon was not cliked before (hidden)
      if (!$('.change-message--on-click').is('hide--first')) {
        $('.change-message--on-click').removeClass('hide--second');
        $('.change-message--on-click').addClass('hide--first');

        // add cookie setting that user has clicked
        $.cookie('hide-after-click', 'yes', {expires: 7 });
      }
      return false;
    })
  });
<script/>
```

Cookies can be used in many ways. Now you know how to create your own Hellobar. You could take it a step further and figure out how to authenticate users (remember login details) and save entire sessions in the cookies (sign up process does not get lost in case you refresh the page).

6.1.6 Privacy and Legislation

Cookies can only be read by the site that created them, or a site 'underneath' the site that created them. This prevents other websites from stealing cookies. When cookies were fairly new, there was a lot of controversy about their ability to track users browsing 'all around the web'. This is not really the case, due to sites only being able to read their own cookies; however, for affiliated companies, such as advertising companies, it is true that cookies could be placed in banners such that any site showing a banner could aid the banner company in tracking every website the banner viewer visited on their network.

Keyword

Privacy is the ability of an individual or group to seclude themselves, or information about themselves, and thereby express themselves selectively.

Therefore as more paranoid users may feel the need to disable their cookies, Composr does not require them: session details may be relayed by URL in Composr. The obvious disadvantage is that automatic login is not possible in this situation, and there is an additional disadvantage that JavaScript will be thought to be disabled, as Composr needs to use cookies to detect it. The first problem may be ameliorated by the web browser 'auto-fill' feature, which can be used to automatically remember how forms, such as login forms, were filled in. The developers recommend that users do have cookies enabled, but that they possibly disable 'third party cookies' if they are concerned about **privacy** so that advertisers can not track the advertising sites that they view.

EU legislation

The EU require tracking cookies be declared for organizations operating inside the EU. The Composr "Cookie notice" configuration option implements this. It is not on by default because Composr's cookies are heavily minimized to what we consider reasonable compliance without special notice. However, use of something like Adsense strictly requires you enable the cookie notice.

20 things you did not know about cookies (for programmers)

1. All cookies have an expiry time. A cookie that has expired in the past will be deleted once the web browser is closed. Cookies can also be deleted explicitly.
2. It is this expiry behavior that leads to 'session cookies'. Session cookies have no actual definition beyond that they are defined to expire in the past from the very time that they are created. The emergent behavior is that they act as temporary cookies, existent only for a browser session.
3. Session cookie XSS prevention security is lost if web browser tabs are used: the cookies do not expire because the browser is never closed.
4. Cookie expiry time is measured in GMT UNIX timestamp seconds, hence client/server time is theoretically the same – but care is still needed as computer clocks may be fast/slow
5. A third-party cookie is a cookie that is set onto a domain name that the main page document cannot read. This is possible only because a document may reference images on other servers, and these images may themselves set cookies (any URL can generate cookies). Some browser privacy settings disable these.
6. While cookies are sometimes disabled by people for privacy concerns, session cookies are usually allowed as an exception – so it is not the end of the world if session cookies are required – but it is better to be able to store session IDs in the URL. Some popular websites do require cookies to be enabled though (Tesco.com, New York Times, …).
7. Full cookie data is sent with all web requests that the cookies are scoped under, even image requests – so it is inefficient to store a lot of cookies.
8. Cookies must be set against a domain name. This is either done by putting .domain as the cookie domain, or by leaving the domain blank when setting the cookie; the domain should never be defined as domain as it will not work properly.
9. Cookies work with an elaborate but confusing precedence system. Only cookies underneath a matched domain/path combination will be sent to a server URL (for privacy reasons), and they will be given precedence based on 'most specific gets priority'. To change a cookie the server must set it against the domain/path combination it was created with. The variables a cookie was defined under are not available server-side, which means that anyone modifying the cookie must know these in advance, or guess wildly.
10. There is a legitimate privacy concern with cookies when ads are concerned. Banner rotations run from centralized sites, and hence have the ability to

effectively track users from this centralised site but with regard any site that they visit that uses the rotation. Nevertheless, such tracking could happen regardless of cookies, via server logging and cross-server messaging – so blaming cookies is simplistic.

11. Microsoft made a great extension to Netscape's original cookie spec, allowing 'HTTP only' cookies (cookies that JavaScript cannot read). Use of this prevents XSS many vulnerabilities.
12. At the protocol level, cookies are sent to the server in a single Cookie HTTP header, but set from the server using individual Set-Cookie headers.
13. Cookies just store names and values, and never any data that a web server or normal JavaScript would not have been able to discern – because it is the web server or browser that sets the cookie.
14. JavaScript provides its cookie support by a virtual variable, document.cookie. The variable can be set and read, but the process is not actually direct.
15. A common server-side coding mistake is to set a cookie and then refer to the cookie value within the same server response – yet the cookie would not have been activated until the response had been sent.
16. Some web servers (including Apache) restrict cookie data length, refusing to server data if the length is exceeded.
17. Cookie names should not contain certain special characters like '=' as these have special meanings within HTTP and there is no standard escaping mechanism for cookie names. (Unexpected bugs may happen if you attempt to set such cookies)
18. Cookies were invented by Netscape, not by the usual standards bodies (the IETF or W3C).
19. On some web servers it is not possible to set a cookie at the same time as doing an HTTP redirect.
20. The name 'cookie' was given for no particular reason, but is the origin of endless bad jokes.

6.1.7 Advantages and Disadvantages of using Cookies

A cookie is a small bit of text that accompanies requests and pages as they go between the Web server and browser.

Advantages of using cookies

Here are some of the advantages of using cookies to store session state.

- Cookies are simple to use and implement.

- Occupies less memory, do not require any server resources and are stored on the user's computer so no extra burden on server.
- We can configure cookies to expire when the browser session ends (session cookies) or they can exist for a specified length of time on the client's computer (persistent cookies).
- Cookies persist a much longer period of time than Session state.

Disadvantages of using cookies

Here are some of the disadvantages:

- Cookies are not secure as they are stored in clear text they may pose a possible security risk as anyone can open and tamper with cookies. You can manually encrypt and decrypt cookies, but it requires extra coding and can affect application performance because of the time that is required for encryption and decryption
- Several limitations exist on the size of the cookie text (4kb in general), number of cookies(20 per site in general), etc.
- User has the option of disabling cookies on his computer from browser's setting.
- Cookies will not work if the security level is set to high in the browser.
- Users can delete a cookies.
- Users browser can refuse cookies, so your code has to anticipate that possibility.
- Complex type of data not allowed (e.g. dataset etc.). It allows only plain text (i.e. cookie allows only string content)

6.2 SETTING DIFFERENT COOKIE KINDS IN JAVASCRIPT

A cookie is a named piece of data, created and used by a certain website for a certain viewing user, and sent from the user's web browser to the web server each time a page is viewed. In this section we will show you how to create different kind of cookies, in JavaScript.

Originally there was no way to identify a user on the web that was viewing a website with a user that had previously visited, unless they had an account on the website and logged in each time. It was possible to identify a user within a visit, without them being logged-in, by storing additional information in URLs: however this is unwieldy. Cookies were designed to resolve this problem, and another one:

- it would allow server-side web applications to identify a specific user by the computer they accessed with
- it would allow **client-side** web applications to have a memory, which was otherwise impossible

Keyword

Client-side refers to operations that are performed by the client in a client–server relationship in a computer network.

6.2.1 Session Cookie –First Cookie

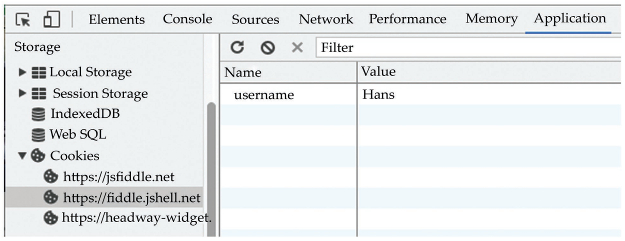
A session identifies a user, even if they are not a member. It is a unique number attached to a user and stored in a 'session cookie' or their URLs. A session cookie is a special kind of cookie that is automatically deleted when a user closes their web browser. We will start with the most basic cookie: the Session cookie. This would be simplest cookie you can create and is referred to as a "Session Cookie" since it does not live outside of a session, so it gets deleted automatically when you close the browser. Information of a cookie is stored in value pairs, and it is most simple form would look something like this:

username=Hans;

Storing sensitive information like usernames and such, in a cookie, is not really a BAD idea – here it just serves as an example. Below the JavaScript code which you can paste in JSFiddle;

document.cookie = "username=Hans";

Obviously, we did not see anything exciting happen, so use the web developer tools of your browser to see that the cookie has actually been set. You can use the "Inspect" option from Google Chrome on the JSFiddle page as well.

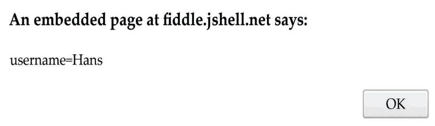


Name	Value
username	Hans

If you would prefer to not use the browsers web developer tools, you could also use an alert, which would look something like this:

```
document.cookie = "username=Hans";
alert(document.cookie);
```

The result should be a popup, showing something like this:



An embedded page at fiddle.jshell.net says:

username=Hans

Alrighty then we have the basics kind-a under control, so now time to do some more advanced things by adding options. These options are stored as value pairs as well, in the same document.cookie property. Let's go through them one at a time.

Sessions have the following advantages over conventional cookies:

- they allow remembering of guests
- they can be used to force explicit login for a member
- they can be used even when cookies are disabled

Note that IP addresses could never be used instead of sessions because they are often shared between multiple users, and because a single user's dynamic IP address may often change. Composr has an added layer of security, in that it only allows a session cookie to work if it was created for a similar IP address: this reduces the security risk of 'session stealing' if a hacker somehow managed to find another user's session (which should not be possible in itself).

Remember

Session for guests are actually using normal cookies, not session cookies. This is because they may need to be identified between visits if Composr has been extended with features such as a shopping cart system.

6.2.2 Persistent Cookie – A Cookie that survives closing the browser

A persistent cookie is a data file capable of providing websites with user preferences, settings and information for future visits. Persistent cookies provide convenient and rapid access to familiar objects, which enhances the user experience (UX). A persistent cookie is also known as a stored or permanent cookie. When users visit a website and set choices or preferences, persistent cookies may be used to remember these options. Persistent cookies can help isolate and identify a specific client and are capable of traversing a user's path toward a website. They are stored as text files on the hard drive of a computer and usually have expiration dates of one to two years. Persistent cookies facilitate setting the following preferences:

- Favorites or internal bookmarks
- User authentication
- Login details
- Menu preferences
- Theme selection, if applicable
- Language preferences

Persistent cookies are also capable of providing the browsing behavior of users.

As mentioned before, we can make a cookie persistent by defining it is expiration date and time, which would look like something like the example below:

username=Hans; Expires=Sun, 22 Oct 2017 08:00:00 UTC;

This will also be used to delete a cookie, by setting the expiration date to a date in the past – your browser will remove the cookie based on that alone.

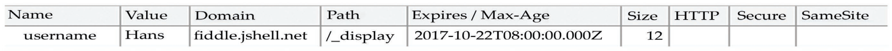
- The date format for cookies must be in a UTC/GMT format (i.g. Sun, 22 Oct 2017 08:00:00 UTC), so correct for your timezone if needed.
- **UTC** versus **GMT**: both indicate the same time, however: UTC is a time standard, whereas GMT is a timezone. UTC is preferred.

OK, ready for an example – here it can be very helpful to look at the earlier mentioned "Inspect" (web developer) tools, since it lists these properties as well. Our JavaScript will look something like this (remember to change the date here if you are trying this after Oct 22nd 2017 8 AM UTC):

Example

```
document.cookie = "username=Hans; Expires=Sun, 22 Oct 2017 08:00:00 UTC;";
alert(document.cookie);
```

And when inspecting the cookies with Google Chrome (for example), we will see something like this:



Name	Value	Domain	Path	Expires / Max-Age	Size	HTTP	Secure	SameSite
username	Hans	fiddle.jshell.net	/_display	2017-10-22T08:00:00.000Z	12			

Here we can see the name, value, domain, path, expiration date/time, size (number of characters of name + value), HTTPOnly, Secure and SameSite. The alert on the other hand only shows "username=Hans".

A few things to note here, and the alert() message kind-a points in that direction as well this:

- document.cookie only holds the name-value pairs that we are allowed to see,
- properties of a value pair (like expiration) is not returned to use when we read document.cookie.

The fact that we can see this info in the web developer tool, is because these tools are part of your browser and allow looking into the inner workings – which is great for developers and for us while we are learning more about cookies.

Remember

Whether the cookie must use a secure connection (https://). If this value is TRUE, the cookie can be transferred only across a secure connection. The default is FALSE.

6.2.3 Secure Cookie – Which only works when HTTPS is being used

A secure cookie, also known as httpOnly cookie, is a type of cookie that only works with HTTP/HTTPS and does not work for scripting languages like JavaScript. Since it is only used in storing information and used for hypertext transfer protocol requests and data over the **internet**, exploits and hacks made through scripting are unable to access them. So a secure cookie's main benefit is that it can stop theft through cross-site scripting (XSS).

Keyword

Internet is the global system of interconnected computer networks that use the Internet protocol suite (TCP/IP) to link devices worldwide.

Making a cookie secure is pretty easy as well, we simply add the option "Secure".

```
username=Hans; Secure;
```

This one can be a tricky one to test with a local file or your own web-server, since this one would only work over a HTTPS connection which typically is not the case. JSFiddle on the other hand uses HTTPS so it will work there;

```
document.cookie = "username=Hans; Secure;";
alert(document.cookie);
```

The "Inspect" option in Google Chrome, might require you to click the refresh button in the web developer tools (as indicated before). You will now see a checkmark in the "secure" column as well.

A secure cookie always has the secure attribute activated, so it is used mostly via HTTPS and securely transmitted with encrypted connections. The httpOnly flag in the secure cookie header ensures that JavaScript or any non-HTTP methods cannot access the cookie. The cookie works through the assistance of two headers: set-cookie and cookie. The job of the set-cookie header is to create a secured cookie on the user's system in response to an http request. While the cookie header is part of the application with an http request sent to the server to validate if there is a secure cookie that matches the domain and path requested. The secure attribute and httpOnly flag work together to ensure that the browser is able to restrict access to the secure cookie data from malicious scripts that may have infected the browser or the network. This mitigates many of the damages that many XSS attacks can cause, specifically those that target cookies.

6.2.4 HTTPOnly Cookie – The Cookie that can only be accessed by the web-server

HttpOnly is a flag added to cookies that tell the browser not to display the cookie through client-side scripts (document.cookie and others). The agenda behind HttpOnly is not to spill out cookies when an XSS flaw exists, as a hacker might be able to run their script but the fundamental benefit of having an XSS vulnerability (the ability steal cookies and hijack a currently established session) is lost.

When you set a cookie with the HttpOnly flag, it informs the browser that this special cookie should only be accessed by the server. Any try to access the cookie from client side script is strictly forbidden. Of course, this presumes you have: A modern web browser. HttpOnly cookies were first presented in Microsoft's Internet Explorer 6 SP1, and as of now, this has become a popular practice while setting session cookies.

The same goes for making a HTTPOnly cookie, we just add "HttpCookie" – before doing so, you might want to delete all cookies in the web developer tools of your browser.

```
document.cookie = "username=Hans; HttpOnly;";
                  alert(document.cookie);
```

Remember

Whether the cookie must use the HTTP protocol. If this value is TRUE, scripting languages such as JavaScript cannot access the cookie. The default is FALSE.

Now refreshing the cookies in the web developer tools should not show a cookie, and neither does the "alert()" … which is correct. The cookie exists, but we just are not allowed to access it with JavaScript.

6.2.5 SameSite Cookie – A Cookie only for this website

SameSite prevents the browser from sending this cookie along with cross-site requests. The main goal is mitigate the risk of cross-origin information leakage. It also provides some protection against cross-site request forgery attacks. SameSite cookies, if supported by the browser, knows two variants: strict and lax. Here strict is the preferred setting, since it closes down all cross domain access.

```
document.cookie = "username=Hans; SameSite = strict;";
alert(document.cookie);
```

When you want to allow "some" cross domain access:

```
document.cookie = "username=Hans; SameSite = lax;";
alert(document.cookie);
```

This again is a tricky one, since it might only show something with your local web-server. A file or JSFiddle might not show anything in the "alert()", but the web developer tools will show that the cookie actually does exist.

6.2.6 Cookie Domain – For Cookies that are only for a specific domain

The cookie domain is an important security feature, probably even more important than the secure flag. It tells the browser that this cookie must only be sent to matching domains. We can define a domain and path in which a cookie can be accessed. To limit a cookie to a certain domain, you can try the following two examples. The first one will use a random domain, example.com. The second example will use the domain jshell.net – which is the domain used by JSFiddle.

Note: when entering "example.com" as the domain, the cookie can only be accessed from that exact domain, but not from sub domains. When adding a period in front of the domain, for example ".example.com" then the cookie can be accessed from sub domains as well (i.e. forum.example.com, email.example.com, www.example.com, etc).

```
document.cookie = "username=Hans; domain = .example.com;";
alert(document.cookie);
```

You will notice that there is no cookie and the alert is not showing anything either – which again is correct since we are not allowed access to these cookies, since we are working in the wrong domain (not example.com).

document.cookie = "username=Hans; domain = .jshell.net;"; alert(document.cookie);

This second example will generate a cookie that is visible and the alert will show the cookie as well (when testing this in JSFiddle, since JSFiddle runs in the jshell.net domain).

6.2.7 Cookie Domain and Path – For Cookies that are only for a specific path and domain

We can narrow this down by adding a path, which would only allow access to this cookie if we are in the right domain and path (if Domain is not defined, the current domain will be used). The following example the following tells the browser to only access this cookie when we are in the path /secretdata (on the current domain). So if we would be working on the server with domain example.com, this would work for pages in for example http://example.com/secretdata and it is sub paths like for example **http://example.com/secretdata/morestuff**. Obviously this is a little trickier to test – and you might not need this ever.

It is recommended to add the "Path" to your cookies so the browser will not get confused too much. By default the cookie will be relative to the page path you are working with. But if you want to access the cookie from another page on your website, then the cookie might actually not be accessible. If the cookie should be readable throughout your entire dome then always add: "Path=/;".

document.cookie = "username=Hans; path = /secretdata;";

alert(document.cookie);

To illustrate how things get confusing, try the following code. The first "username" cookie will be relative to this page, the second "username" relative to the entire site. Since one was defined without path (relative to this page) and one with path (relative to the **website**), suddenly the browser has 2 cookies with the same name – which obviously might result in unexpected behavior when trying to read the cookie.

Keyword

Website is a collection of related web pages, including multimedia content, typically identified with a common domain name, and published on at least one web server.

```
document.cookie="username=Hans";
alert(document.cookie);
document.cookie="username=John; Path=/;";
alert(document.cookie);

document.cookie="username=Banana;";
alert(document.cookie);
```

In the example above: first we create the cookie username=Hans relative to this page and the alert confirms this. The next cookie however is defined as a new (!) cookies username=John with Path=\ (entire domain). The alert now actually shows 2 cookies with the same name: username=Hans; username=John. So which one do we use in our code? In the 3rd part we change the value of one of the page relative cookies, for example because we forgot to add the path, and guess what things become messy real quick. Here you can see how important it can be to properly define the path.

6.2.8 Combining options – A Cookie jar of options

All these options can be combined in a single line as well, separated by semi colons, even when it might make little or no sense:

document.cookie = "username=Hans;Path=/; Domain=email.example.com; Expires=Sun, 22 Oct 2017 08:00:00 UTC; Secure; HttpOnly; SameSite = strict;";

alert(document.cookie);

So this cookie holds the username=Hans value pair, can be used until Sunday October 22nd, 8 AM UTC (GMT), and only be accessed in the email.example.com domain with the path /secretdata, cross domain use is prohibited (SameSite), HTTPS is required (secure) and it is a HTTPOnly cookie (which can only be read by the server).

SUMMARY

- A cookie is an item of data that a web server saves to your computer's hard disk via a web browser. A cookie is a piece of data that is stored on your computer to be accessed by your browser.
- A computer cookie, also referred to as an "HTTP cookie," is a small text file that contains a unique ID tag, placed on the user 's computer by a website.
- Cookies are relatively small text files that a **web browser** embeds on a user's computer. Cookies allow otherwise stateless HTTP communications to emulate state (i.e., memory). Cookies are being replaced by somewhat newer technologies such as local storage and session storage; however, cookies are still widely used by many major websites today.
- Cookies are necessary because the HTTP protocol that is used to transfer webpages around the web is *state-less*. This means that web servers cannot remember information about users throughout their travels, and so everyone becomes anonymous.
- The document has an object in JavaScript called document.cookie, which is used to read and retrieve cookie data. It is a repository of Strings (though not an array).
- A cookie is a small bit of text that accompanies requests and pages as they go between the Web server and browser.
- A cookie is a named piece of data, created and used by a certain website for a certain viewing user, and sent from the user's web browser to the web server each time a page is viewed.
- A persistent cookie is a data file capable of providing websites with user preferences, settings and information for future visits. Persistent cookies provide convenient and rapid access to familiar objects, which enhances the user experience (UX). A persistent cookie is also known as a stored or permanent cookie.
- A secure cookie, also known as httpOnly cookie, is a type of cookie that only works with HTTP/HTTPS and does not work for scripting languages like JavaScript.
- HttpOnly is a flag added to cookies that tell the browser not to display the cookie through client-side scripts (document.cookie and others).

KNOWLEDGE CHECK

1. **Cookies were originally designed for**
   a. Client-side programming
   b. Server-side programming
   c. Both Client-side & Server-side programming
   d. None of the mentioned
2. **The Cookie manipulation is done using which property?**
   a. cookie
   b. cookies
   c. manipulate
   d. none of the mentioned
3. **Which of the following explains Cookies nature?**
   a. Non Volatile
   b. Volatile
   c. Intransient
   d. Transient
4. **Which attribute is used to extend the lifetime of a cookie?**
   a. higher-age
   b. increase-age
   c. max-age
   d. lifetime
5. **Which of the following defines the Cookie visibility?**
   a. document Path
   b. localStorage
   c. sessionStorage
   d. all of the mentioned
6. **Which of the following can be used to configure the scope of the Cookie visibility?**
   a. path
   b. domain
   c. both path and domain
   d. server

7. How can you set a Cookie visibility scope to localStorage?

a. /

b. %

c. *

d. //

8. Which of the following is a boolean cookie attribute?

a. bool

b. secure

c. `lookup

d. domain

9. Which of the following function is used as a consequence of not including semicolons, commas or whitespace in the Cookie value?

a. encodeURIComponent()

b. encodeURI()

c. encodeComponent()

d. encode()

10. What is the constraint on the data per cookie?

a. 2 KB

b. 1 KB

c. 4 KB

d. 3 KB

REVIEW QUESTIONS

1. Write the use of cookies in JavaScript.
2. What is the structure of a cookie?
3. Discuss about setting, reading and erasing cookies.
4. How to set cookies with JavaScript.
5. Define the privacy and legislation of cookies.
6. Write the advantages and disadvantages of using cookies.

Check Your Result

1. (b)	2. (a)	3. (d)	4. (c)	5. (d)
6. (c)	7. (a)	8. (b)	9. (a)	10. (c)

REFERENCES

1. Eckersley, Peter (17 May 2010). "How Unique Is Your Web Browser?" (PDF). *eff.org*. Electronic Frontier Foundation. Archived from the original (PDF) on 15 October 2014. Retrieved 23 July 2014.
2. http://clubmate.fi/setting-and-reading-cookies-with-javascript/
3. http://notes.corewebprogramming.com/student/JavaScript.pdf
4. http://www.cs.toronto.edu/~mashiyat/csc309/Lectures/Session_Cookies.pdf
5. http://www.dis.uniroma1.it/~damore/was/slides2015/cookies.pdf
6. http://www.howtocreate.co.uk/tutorials/javascript/cookies
7. http://www.javascripter.net/faq/settinga.htm
8. http://www.javascriptkit.com/javatutors/cookiedetect.shtml
9. http://www.splessons.com/lesson/javascript-cookies/
10. http://www.webtoolkit.info/javascript_cookies.html#.W6srF2gzbIU
11. https://alexcican.com/post/set-cookies-javascript/
12. https://appendto.com/2017/01/cookies-with-javascript/
13. https://developer.mozilla.org/en-US/docs/Web/API/Document/cookie
14. https://plainjs.com/javascript/utilities/set-cookie-get-cookie-and-delete-cookie-5/
15. https://resources.infosecinstitute.com/securing-cookies-httponly-secure-flags/#gref
16. https://stackoverflow.com/questions/14573223/set-cookie-and-get-cookie-with-javascript
17. https://tech.findmypast.com/creating-cookies-in-javascript/
18. https://wanago.io/2018/06/18/cookies-explaining-document-cookie-and-the-set-cookie-header/
19. https://way2tutorial.com/javascript/javascript_cookies.php
20. https://www.bitdegree.org/learn/what-are-cookies/
21. https://www.codexpedia.com/javascript/javascript-create-read-and-delete-cookies/
22. https://www.guru99.com/cookies-in-javascript-ultimate-guide.html
23. https://www.ics.uci.edu/~lopes/teaching/inf212W15/lectures/EPS-OOP-JavaScript.pdf
24. https://www.javatpoint.com/javascript-cookies
25. https://www.perlscriptsjavascripts.com/js/cookies.html
26. https://www.quackit.com/javascript/tutorial/javascript_functions.cfm
27. https://www.thesitewizard.com/javascripts/cookies.shtml
28. https://www.thonky.com/javascript-and-css-guide/set-cookie

29. https://www.tutorialspoint.com/javascript/javascript_cookies.htm
30. https://www.tweaking4all.com/web-development/generic-web-design/cookies-in-javascript/#UpdatingaCookiewithJavaScript
31. https://www.webcodeexpert.com/2013/03/what-is-cookie-advantages-and.html
32. https://www.wisegeek.com/what-are-computer-cookies.htm
33. https://www.yourhtmlsource.com/javascript/cookies.html

CHAPTER 7

JAVA SCRIPT: CLASSES AND OBJECTS

"Java is to JavaScript as ham is to hamster."

– Jeremy Keith

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Understand Java Script: Classes
2. Discuss about Class Body and Method Definitions
3. Identify JavaScript: Objects

```
if ($(window).scrollTop() > header1_initialDistance) {
    if (parseInt(header1.css('padding-top'), 10)
        header1.css('padding-top', '' + $(window).scrollTop()
    }
} else {
    header1.css('padding-top', '' + header1_initialPadding
}

if ($(window).scrollTop() > header2_initialDistance) {
    if (parseInt(header2.css('padding-top'), 10)
        header2.css('padding-top', '' + $(window).scrollTop()
    }
} else {
    .css('padding-top', '' + header2_initialPadding
```

INTRODUCTION

JavaScript classes, introduced in ECMA Script 2015, are primarily syntactical sugar over JavaScript's existing prototype-based inheritance. The class syntax does not introduce a new object-oriented inheritance model to JavaScript. JavaScript is a very flexible object-oriented language when it comes to syntax. In this chapter you can find three ways of defining and instantiating an object. Even if you have already picked your favorite way of doing it, it helps to know some alternatives in order to read other people's code. It's important to note that there are no classes in JavaScript. Functions can be used to somewhat simulate classes, but in general JavaScript is a class-less language. Everything is an object. And when it comes to inheritance, objects inherit from objects, not classes from classes as in the "class"-ical languages.

Keyword

Function Declaration defines a named function variable without requiring variable assignment.

7.1 JAVA SCRIPT: CLASSES

Classes are in fact "special functions", and just as you can define function expressions and **function declarations**, the class syntax has two components: class expressions and class declarations.

```
class Rectangle {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
}
```

Keyword

The **class expression** is one way to define a class in ECMAScript 2015. Similar to function expressions, class expressions can be named or unnamed. If named, the name of the class is local to the class body only. JavaScript classes use prototype-based inheritance.

Hoisting

An important difference between function declarations and class declarations is that function declarations are hoisted and class declarations are not. You first need to declare your class and then access it, otherwise code like the following will throw a Reference Error:

```
const p = new Rectangle(); // ReferenceError

class Rectangle {}
```

Class expressions

A **class expression** is another way to define a class. Class expressions can be named or unnamed. The name given to a named class expression is local to the class's body. (it can be retrieved through the class's (not an instance's) .name property, though)

```
// unnamed
let Rectangle = class {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
};
console.log(Rectangle.name);
// output: "Rectangle"

// named
let Rectangle = class Rectangle2 {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
};
console.log(Rectangle.name);
// output: "Rectangle2"
```

7.2 CLASS BODY AND METHOD DEFINITIONS

The body of a class is the part that is in curly brackets {}. This is where you define class members, such as methods or constructor.

Strict mode

The body of a class is executed in strict mode, i.e., code written here is subject to stricter syntax for increased performance,

Keyword

In computer science, a **syntax error** is an error in the syntax of a sequence of characters or tokens that is intended to be written in a particular programming language.

some otherwise silent errors will be thrown, and certain keywords are reserved for future versions of ECMAScript.

Constructor

The constructor method is a special method for creating and initializing an object created with a class. There can only be one special method with the name «constructor" in a class. A **Syntax Error** will be thrown if the class contains more than one occurrence of a constructor method. A constructor can use the super keyword to call the constructor of the super class.

7.2.1 Prototype Methods

See also method definitions.

```
class Rectangle {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
  // Getter
  get area() {
    return this.calcArea();
  }
  // Method
  calcArea() {
    return this.height * this.width;
  }
}

const square = new Rectangle(10, 10);

console.log(square.area); // 100
```

Static methods

The static keyword defines a static method for a class. Static methods are called without instantiating their class and cannot be called through a class instance. Static methods are often used to create utility functions for an application.

```
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
  }

  static distance(a, b) {
    const dx = a.x - b.x;
    const dy = a.y - b.y;

    return Math.hypot(dx, dy);
  }
}

const p1 = new Point(5, 5);
const p2 = new Point(10, 10);

console.log(Point.distance(p1, p2)); // 7.0710678118654755
```

7.2.2 Boxing with Prototype and Static Methods

When a static or prototype method is called without a value for this, the value will be undefined inside the method. This behavior will be the same even if the "use strict" directive isn't present, because code within the class body's syntactic boundary is always executed in **strict mode**.

```
class Animal {
  speak() {
    return this;
  }
  static eat() {
    return this;
  }
}
```

Keyword

Strict Mode is a new feature in ECMAScript 5 that allows you to place a program, or a function, in a "strict" operating context.

```
let obj = new Animal();
obj.speak(); // Animal {}
let speak = obj.speak;
speak(); // undefined

Animal.eat() // class Animal
let eat = Animal.eat;
eat(); // undefined
```

If the above is written using traditional function-based syntax, then autoboxing in method calls will happen in non–strict mode based on the initial *this* value. If the initial value is undefined, *this* will be set to the global object.

Autoboxing will not happen in strict mode, the value remains as passed.

```
function Animal() { }

Animal.prototype.speak = function() {
  return this;
}

Animal.eat = function() {
  return this;
}

let obj = new Animal();
let speak = obj.speak;
speak(); // global object

let eat = Animal.eat;
eat(); // global object
```

Instance Properties

Instance properties must be defined inside of class methods:

```
class Rectangle {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
}
```

Static class-side properties and prototype data properties must be defined outside of the ClassBody declaration:

```
Rectangle.staticWidth = 20;
Rectangle.prototype.prototypeWidth = 25;
```

7.2.3 Sub classing with extends

The extends keyword is used in class declarations or class expressions to create a class as a child of another class.

```
class Animal {
  constructor(name) {
    this.name = name;
  }

  speak() {
    console.log(this.name + ' makes a noise.');
  }
}

class Dog extends Animal {
  constructor(name) {
   super(name); // call the super class constructor and pass in the name parameter
  }

  speak() {
    console.log(this.name + ' barks.');
    }
  }

  let d = new Dog('Mitzie');
  d.speak(); // Mitzie barks.
```

If there is a constructor present in the subclass, it needs to first call super () before using "this".

One may also extend traditional function-based "classes":

```
function Animal (name) {
  this.name = name;
}

Animal.prototype.speak = function () {
  console.log(this.name + ' makes a noise.');
}

class Dog extends Animal {
  speak() {
    console.log(this.name + ' barks.');
  }
}

let d = new Dog('Mitzie');
d.speak(); // Mitzie barks.
```

Keyword

Those substances which have fixed geometrical shape are called **regular objects.**

Note that classes cannot extend regular (non-constructible) objects. If you want to inherit from a **regular object**, you can instead use Object.setPrototypeOf():

```
const Animal = {
  speak() {
    console.log(this.name + ' makes a noise.');
  }
};

class Dog {
  constructor(name) {
    this.name = name;
  }
}

// If you do not do this you will get a TypeError when you invoke speak
Object.setPrototypeOf(Dog.prototype, Animal);
let d = new Dog('Mitzie');
d.speak(); // Mitzie makes a noise.
```

Species

You might want to return Array objects in your derived array class MyArray. The species pattern lets you override default constructors.

For example, when using methods such as map() that returns the default constructor, you want these methods to return a parent Array object, instead of the MyArray object. The Symbol.species symbol lets you do this:

```
class MyArray extends Array {
  // Overwrite species to the parent Array constructor
  static get [Symbol.species]() { return Array; }
}

let a = new MyArray(1,2,3);
let mapped = a.map(x => x * x);

console.log(mapped instanceof MyArray); // false
console.log(mapped instanceof Array);   // true
```

Super class calls with super

The super keyword is used to call corresponding methods of super class. This is one advantage over prototype-based inheritance.

```
class Cat {
  constructor(name) {
    this.name = name;
  }

  speak() {
    console.log(`${this.name} makes a noise.`);
  }
}

class Lion extends Cat {
  speak() {
    super.speak();
    console.log(`${this.name} roars.`);
  }
}

let l = new Lion('Fuzzy');
l.speak();
// Fuzzy makes a noise.
// Fuzzy roars.
```

Mix-ins

Abstract subclasses or mix-ins are templates for classes. An ECMAScript class can only have a single superclass, so multiple inheritance from tooling classes, for example, is not possible. The functionality must be provided by the superclass.

A function with a superclass as input and a subclass extending that superclass as output can be used to implement mix-ins in ECMAScript:

```
let calculatorMixin = Base => class extends Base {
  calc() { }
};

let randomizerMixin = Base => class extends Base {
  randomize() { }
```

```
    };
```

A class that uses these mix-ins can then be written like this:

```
    class Foo { }
    class Bar extends calculatorMixin(randomizerMixin(Foo))
{ }
```

7.2.4 Using a Function

This is probably one of the most common ways. You define a normal **JavaScript function** and then create an object by using the new keyword. To define properties and methods for an object created using function(), you use the this keyword, as seen in the following example.

```
    function Apple (type) {
        this.type = type;
        this.color = "red";
        this.getInfo = getAppleInfo;
    }

    // anti-pattern! keep reading...
    function getAppleInfo() {
        return this.color + ' ' + this.type + ' apple';
    }
```

To instantiate an object using the Apple constructor function, set some properties and call methods you can do the following:

```
    var apple = new Apple('macintosh');
    apple.color = "reddish";
    alert(apple.getInfo());
```

Keyword

A **JavaScript function** is defined with the function keyword, followed by a name, followed by parentheses (). Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

Methods defined Internally

In the example above you see that the method getInfo() of the Apple "class" was defined in a separate function getAppleInfo(). While this works fine, it has one drawback – you may end up defining a lot of these functions and they are all in the "global namespece". This means you may have naming conflicts if you (or another library you are using) decide to create another function with the same name. The way to prevent pollution of the global namespace, you can define your methods within the constructor function, like this:

```
function Apple (type) {
    this.type = type;
    this.color = "red";
    this.getInfo = function() {
        return this.color + ' ' + this.type + ' apple';
    };
}
```

Using this syntax changes nothing in the way you instantiate the object and use its properties and methods.

Methods Added To the Prototype

A drawback of is that the method getInfo() is recreated every time you create a new object. Sometimes that may be what you want, but it's rare. A more inexpensive way is to add getInfo() to the *prototype* of the constructor function.

```
function Apple (type) {
    this.type = type;
    this.color = "red";
}

Apple.prototype.getInfo = function() {
    return this.color + ' ' + this.type + ' apple';
};
```

Remember

In the place of number, if you provide any non-number argument, then the argument cannot be converted into a number, it returns NaN (Not-a-Number).

7.2.5 Using Object Literals

Literals are shorter way to define objects and arrays in JavaScript. To create an empty object using you can do:

```
var o = {};
```

instead of the “normal” way:

```
var o = new Object();
```

For arrays you can do:

```
var a = [];
```

instead of:

```
var a = new Array();
```

So you can skip the class-like stuff and create an instance (object) immediately. Here’s the same functionality as described in the examples, but using object literal syntax this time:

```
var apple = {
    type: “macintosh”,
    color: “red”,
    getInfo: function () {
        return this.color + ‘ ‘ + this.type + ‘ apple’;
    }
}
```

In this case you don’t need to (and cannot) create an instance of the class, it already exists. So you simply start using this instance.

```
apple.color = “reddish”;
alert(apple.getInfo());
```

Such an object is also sometimes called singleton. In “classical” languages such as Java, singleton means that you can have only one single instance of this class at any time, you cannot create more objects of the same class. In JavaScript (no classes, remember?) this concept makes no sense anymore since all objects are singletons to begin with.

Singleton using a Function

The third way presented in this article is a combination of the other two you already saw. You can use a function to define a singleton object. Here’s the syntax:

```
var apple = new function() {
    this.type = “macintosh”;
    this.color = “red”;
    this.getInfo = function () {
        return this.color + ‘ ‘ + this.type + ‘ apple’;
    };
}
apple.color = "reddish";
alert(apple.getInfo());
```

new function(){...} does two things at the same time: define a function (an anonymous constructor function) and invoke it with new. It might look a bit confusing if you're not used to it and it's not too common, but hey, it's an option, when you really want a constructor function that you'll use only once and there's no sense of giving it a name.

7.3 JAVASCRIPT: OBJECTS

JavaScript is an **Object Oriented Programming (OOP)** language. A programming language can be called object-oriented if it provides four basic capabilities to developers −

- Encapsulation − the capability to store related information, whether data or methods, together in an object.
- Aggregation – the capability to store one object inside another object.
- Inheritance − the capability of a class to rely upon another class (or number of classes) for some of its properties and methods.
- Polymorphism − the capability to write one function or method that works in a variety of different ways.

Keyword

Object-oriented programming (OOP): Object-oriented programming (OOP) is a programming language model organized around objects rather than "actions" and data rather than logic.

Objects are composed of attributes. If an attribute contains a function, it is considered to be a method of the object, otherwise the attribute is considered a property.

7.3.1 Object Properties

Object properties can be any of the three primitive data types, or any of the abstract data types, such as another object. Object properties are usually variables that are used internally in the object's methods, but can also be globally visible variables that are used throughout the page.

The syntax for adding a property to an object is −

```
objectName.objectProperty = propertyValue;
```

For example – The following code gets the document title using the "title" property of the document object.

var str = document.title;

7.3.2 Object Methods

Methods are the functions that let the object do something or let something be done to it. There is a small difference between a function and a method – at a function is a standalone unit of statements and a method is attached to an object and can be referenced by the **this** keyword.

Methods are useful for everything from displaying the contents of the object to the screen to performing complex mathematical operations on a group of local properties and parameters.

For example – Following is a simple example to show how to use the **write()** method of document object to write any content on the document.

document.write("This is test");

User-Defined Objects

All user-defined objects and built-in objects are descendants of an object called **Object**.

The new Operator

The **new** operator is used to create an instance of an object. To create an object, the **new** operator is followed by the constructor method.

In the following example, the constructor methods are Object(), Array(), and Date(). These constructors are built-in JavaScript functions.

var employee = new Object();

var books = new Array("C++", "Perl", "Java");

var day = new Date("August 15, 1947");

The Object() Constructor

A constructor is a function that creates and initializes an object. JavaScript provides a special constructor function called **Object ()** to build the object. The return value of the **Object ()** constructor is assigned to a variable.

The variable contains a reference to the new object. The properties assigned to the object are not variables and are not defined with the **var** keyword.

Example 1

Try the following example; it demonstrates how to create an Object.

```
<html>
   <head>
      <title>User-defined objects</title>

      <script type="text/javascript">
         var book = new Object();   // Create the object
         book.subject = "Perl"; // Assign properties to the object
         book.author  = "Mohtashim";
      </script>

   </head>

   <body>

      <script type="text/javascript">
         document.write("Book name is : " + book.subject + "<br>");
         document.write("Book author is : " + book.author + "<br>");
      </script>

   </body>
</html>
```

Output

```
Book name is : Perl
Book author is : Mohtashim
```

Example 2

This example demonstrates how to create an object with a User-Defined Function. Here **this** keyword is used to refer to the object that has been passed to a function.

```
<html>
   <head>
```

```
  <title>User-defined objects</title>

    <script type="text/javascript">
      function book(title, author){
        this.title = title;
        this.author  = author;
      }
    </script>

  </head>
  <body>

    <script type="text/javascript">
      var myBook = new book("Perl", "Mohtashim");
      document.write("Book title is : " + myBook.title + "<br>");
      document.write("Book author is : " + myBook.author + "<br>");
    </script>

  </body>
</html>
```

Output

```
Book title is : Perl
Book author is : Mohtashim
```

Defining Methods for an Object

The previous examples demonstrate how the constructor creates the object and assigns properties. But we need to complete the definition of an object by assigning methods to it.

Example

Try the following example; it shows how to add a function along with an object.

```
<html>
```

```
    <head>
    <title>User-defined objects</title>

       <script type="text/javascript">
          // Define a function which will work as a method
          function addPrice(amount){
             this.price = amount;
          }

          function book(title, author){
             this.title = title;
             this.author  = author;
             this.addPrice = addPrice; // Assign that method as property.
          }
       </script>

    </head>
    <body>

       <script type="text/javascript">
          var myBook = new book("Perl", "Mohtashim");
          myBook.addPrice(100);

          document.write("Book title is : " + myBook.title + "<br>");
          document.write("Book author is : " + myBook.author + "<br>");
          document.write("Book price is : " + myBook.price + "<br>");
       </script>

    </body>
 </html>
```

Output

```
Book title is : Perl
Book author is : Mohtashim
Book price is : 100
```

The ‘with’ Keyword

The **‘with’** keyword is used as a kind of shorthand for referencing an object’s properties or methods. The object specified as an argument to **with** becomes the default object for the duration of the block that follows. The properties and methods for the object can be used without naming the object.

Syntax

The syntax for with object is as follows −

```
with (object){
   properties used without the object name and dot
}
```

Example

Try the following example.

```
<html>
   <head>
   <title>User-defined objects</title>

      <script type="text/javascript">
         // Define a function which will work as a method
         function addPrice(amount){
            with(this){
               price = amount;
            }
         }

         function book(title, author){
            this.title = title;
            this.author  = author;
            this.price = 0;
            this.addPrice = addPrice; // Assign that method as property.
         }
      </script>
   </head>
   <body>

      <script type=”text/javascript”>
         var myBook = new book(“Perl”, “Mohtashim”);
         myBook.addPrice(100);

         document.write(“Book title is : “ + myBook.title + “<br>”);
         document.write(“Book author is : “ + myBook.author + “<br>”);
         document.write(“Book price is : “ + myBook.price + “<br>”);
      </script>

   </body>
</html>
```

Output

```
Book title is : Perl
Book author is : Mohtashim
Book price is : 100
```

JavaScript Native Objects

JavaScript has several built-in or native objects. These objects are accessible anywhere in your program and will work the same way in any browser running in any **operating system**.

Here is the list of all important JavaScript Native Objects –

- JavaScript Number Object
- JavaScript Boolean Object
- JavaScript String Object
- JavaScript Array Object
- JavaScript Date Object
- JavaScript Math Object
- JavaScript RegExp Object

7.3.3 Working with Objects

JavaScript is designed on a simple object-based paradigm. An object is a collection of properties, and a property is an association between a name (or key) and a value. A property's value can be a function, in which case the property is known as a method. In addition to objects that are predefined in the browser, you can define your own objects. It chapter describes how to use objects, properties, functions, and methods, and how to create your own objects.

Objects overview

Objects in JavaScript, just as in many other programming languages, can be compared to objects in real life. The concept of objects in JavaScript can be understood with real life, tangible objects.

In JavaScript, an object is a standalone entity, with properties and type. Compare it with a cup, for example. A cup is an object, with properties. A cup has a color, a design, weight, a material it is made of, etc. The same way, JavaScript objects can have properties, which define their characteristics.

example

Objects and Properties

A JavaScript object has properties associated with it. A property of an object can be explained as a variable that is attached to the object. Object properties are basically the same as ordinary JavaScript variables, except for the attachment to objects. The properties of an object define the characteristics of the object. You access the properties of an object with a :simple dot-notation

    objectName.propertyName

Like all JavaScript variables, both the object name (which could be a normal variable) and property name are case sensitive. You can define a property by assigning it a value. For example, let's create an object named myCar and give it properties named make, model, and year as follows:

    var myCar = new Object();

Keyword

An **operating system (OS)** is system software that manages computer hardware and software resources and provides common services for computer programs.

```
myCar.make = 'Ford';
myCar.model = 'Mustang';
myCar.year = 1969;
```

Unassigned properties of an object are undefined (and not null).

```
myCar.color; // undefined
```

Properties of JavaScript objects can also be accessed or set using a bracket notation (for more details see property accessors). Objects are sometimes called associative arrays, since each property is associated with a string value that can be used to access it. So, for example, you could access the properties of the myCar object as follows:

```
myCar['make'] = 'Ford';
myCar['model'] = 'Mustang';
myCar['year'] = 1969;
```

An object property name can be any valid JavaScript string, or anything that can be converted to a string, including the empty string. However, any property name that is not a valid JavaScript identifier (for example, a property name that has a space or a hyphen, or that starts with a number) can only be accessed using the square bracket notation. This notation is also very useful when property names are to be dynamically determined (when the property name is not determined until runtime). Examples are as follows:

```
// four variables are created and assigned in a single go,
// separated by commas
var myObj = new Object(),
    str = 'myString',
    rand = Math.random(),
    obj = new Object();

myObj.type              = 'Dot syntax';
myObj['date created']   = 'String with space';
myObj[str]              = 'String value';
myObj[rand]             = 'Random Number';
myObj[obj]              = 'Object';
myObj['']               = 'Even an empty string';

console.log(myObj);
```

Please note that all keys in the square bracket notation are converted to String type, since objects in JavaScript can only have String type as key type. For example, in the above code, when the key obj is added to the myObj, JavaScript will call the obj.toString() method, and use this result string as the new key. You can also access properties by using a string value that is stored in a variable:

```
var propertyName = 'make';
myCar[propertyName] = 'Ford';

propertyName = 'model';
myCar[propertyName] = 'Mustang';
```

You can use the bracket notation with for...in to iterate over all the enumerable properties of an object. To illustrate how this works, the following function displays the properties of the object when you pass the object and the object›s name as arguments to the function:

```
function showProps(obj, objName) {
  var result = '';
  for (var i in obj) {
   // obj.hasOwnProperty() is used to filter out properties
from the object's prototype chain
    if (obj.hasOwnProperty(i)) {
      result += objName + '.' + i + ' = ' + obj[i] + '\n';
    }
  }
  return result;
}
```

So, the function call showProps(myCar, "myCar") would return the following:

```
myCar.make = Ford
myCar.model = Mustang
myCar.year = 1969
```

Did You Know?

Since the mid-2000s, additional server-side JavaScript implementations have been introduced, such as Node.js in 2009.

7.3.4 Enumerate the Properties of an Object

Starting with ECMAScript 5, there are three native ways to list/traverse object properties:

for...in loops

This method traverses all enumerable properties of an object and its prototype chain

Object.keys(o)

This method returns an array with all the own (not in the prototype chain) **enumerable properties'** names ("keys") of an object o.

Object.getOwnPropertyNames(o)

This method returns an array containing all own properties' names (enumerable or not) of an object o.

Before ECMAScript 5, there was no native way to list all properties of an object. However, this can be achieved with the following function:

```
function listAllProperties(o) {
    var objectToInspect;
    var result = [];


    for(objectToInspect = o; objectToInspect !== null;
objectToInspect = Object.getPrototypeOf(objectToInspect)) {
        result = result.concat(Object.getOwnPropertyNames
(objectToInspect));
    }


    return result;
}
```

This can be useful to reveal "hidden" properties (properties in the prototype chain which are not accessible through the object, because another property has the same name earlier in the prototype chain). Listing accessible properties only can easily be done by removing duplicates in the array.

Keyword

Enumerable properties are those properties whose internal enumerable flag is set to true, which is the default for properties created via simple assignment or via a property initializer (properties defined via Object. defineProperty and such default enumerable to false).

Creating New Objects

JavaScript has a number of predefined objects. In addition, you can create your own objects. You can create an object using an object initializer. Alternatively, you can first create a constructor function and then instantiate an object invoking that function in conjunction with the new operator.

Using object Initializers

In addition to creating objects using a constructor function, you can create objects using an object initializer. Using object initializers is sometimes referred to as creating objects with literal notation. "Object initializer" is consistent with the terminology used by C++.

The syntax for an object using an object initializer is:

```
var obj = { property_1:   value_1,   // property_# may be an identifier...
            2:            value_2,   // or a number...
            // ...,
            'property n': value_n }; // or a string
```

where obj is the name of the new object, each property_*i* is an identifier (either a name, a number, or a string literal), and each value_*i* is an expression whose value is assigned to the property_*i*. The obj and assignment is optional; if you do not need to refer to this object elsewhere, you do not need to assign it to a variable. (Note that you may need to wrap the object literal in parentheses if the object appears where a statement is expected, so as not to have the literal be confused with a block statement.)

Object initializers are expressions, and each object initializer results in a new object being created whenever the statement in which it appears is executed. Identical object initializers create distinct objects that will not compare to each other as equal. Objects are created as if a call to new Object() were made; that is, objects made from object literal expressions are instances of Object.

The following statement creates an object and assigns it to the variable x if and only if the expression cond is true:

```
if (cond) var x = {greeting: 'hi there'};
```

The following example creates myHonda with three properties. Note that the engineproperty is also an object with its own properties.

```
var myHonda = {color: 'red', wheels: 4, engine: {cylinders: 4, size: 2.2}};
```

You can also use object initializers to create arrays.

Using a constructor function

Alternatively, you can create an object with these two steps:

- Define the object type by writing a constructor function. There is a strong convention, with good reason, to use a capital initial letter.
- Create an instance of the object with new.

To define an object type, create a function for the object type that specifies its name, properties, and methods. For example, suppose you want to create an object type for cars. You want this type of object to be called Car, and you want it to have properties for make, model, and year. To do this, you would write the following function:

```
function Car(make, model, year) {
  this.make = make;
  this.model = model;
  this.year = year;
}
```

Remember

Add an image inside a container and add inputs (with a matching label) for each field. Wrap a <form> element around them to process the input. You can learn more about how to process input in our PHP tutorial.

Notice the use of this to assign values to the object's properties based on the values passed to the function.

Now you can create an object called mycar as follows:

```
var mycar = new Car('Eagle', 'Talon TSi', 1993);
```

This statement creates mycar and assigns it the specified values for its properties. Then the value of mycar.make is the string «Eagle», mycar.year is the integer 1993, and so on.

You can create any number of Car objects by calls to new. For example,

```
var kenscar = new Car('Nissan', '300ZX', 1992);
var vpgscar = new Car('Mazda', 'Miata', 1990);
```

An object can have a property that is itself another object. For example, suppose you define an object called person as follows:

```
function Person(name, age, sex) {
  this.name = name;
  this.age = age;
  this.sex = sex;
}
```

and then instantiate two new person objects as follows:

```
var rand = new Person('Rand McKinnon', 33, 'M');
var ken = new Person('Ken Jones', 39, 'M');
```

Then, you can rewrite the definition of Car to include an owner property that takes a person object, as follows:

```
function Car(make, model, year, owner) {
  this.make = make;
  this.model = model;
  this.year = year;
  this.owner = owner;
}
```

To instantiate the new objects, you then use the following:

```
var car1 = new Car('Eagle', 'Talon TSi', 1993, rand);
var car2 = new Car('Nissan', '300ZX', 1992, ken);
```

Notice that instead of passing a literal string or integer value when creating the new objects, the above statements pass the objects rand and ken as the arguments for the owners. Then if you want to find out the name of the owner of car2, you can access the following property:

```
car2.owner.name
```

Note that you can always add a property to a previously defined object. For example, the statement

```
car1.color = 'black';
```

adds a property color to car1, and assigns it a value of "black." However, this does not affect any other objects. To add the new property to all objects of the same type, you have to add the property to the definition of the Car object type.

Using the Object. Create Method

Objects can also be created using the Object.create() method. This method can be very useful, because it allows you to choose the prototype object for the object you want to create, without having to define a constructor function.

```
// Animal properties and method encapsulation
var Animal = {
  type: 'Invertebrates', // Default value of properties
  displayType: function() {  // Method which will display type of Animal
    console.log(this.type);
  }
};

// Create new animal type called animal1
var animal1 = Object.create(Animal);
animal1.displayType(); // Output:Invertebrates

// Create new animal type called Fishes
var fish = Object.create(Animal);
fish.type = 'Fishes';
fish.displayType(); // Output:Fishes
```

7.3.5 Inheritance

All objects in JavaScript inherit from at least one other object. The object being inherited from is known as the prototype, and the inherited properties can be found in the prototype object of the constructor.

Indexing Object Properties

You can refer to a property of an object either by its property name or by its ordinal index. If you initially define a property by its name, you must always refer to it by its name, and if you initially define a property by an index, you must always refer to it by its index.This restriction applies when you create an object and its properties with a constructor function (as we did previously with the Car object type) and when you define individual properties explicitly (for example, myCar.color = "red"). If you initially define an object property with an index, such as myCar[5] = "25 mpg", you subsequently refer to the property only as myCar[5].

The exception to this rule is array-like object reflected from HTML, such as the forms array-like object. You can always refer to objects in these array-like objects by either their ordinal number (based on where they appear in the document) or their name (if defined). For example, if the second <FORM> tag in a document has a NAME attribute of "myForm", you can refer to the form as document.forms[1] or document.forms["myForm"] or document.forms.myForm.

Defining properties for an object type

You can add a property to a previously defined object type by using the prototype property. This defines a property that is shared by all objects of the specified type, rather than by just one instance of the object. The following code adds a color property to all objects of type Car, and then assigns a value to the color property of the object car1.

```
Car.prototype.color = null;
car1.color = 'black';
```

Creating Objects with Constructor Functions

Object literal notation, such as var x = {}, is preferred if all you need is a single object and there is no need for multiple instances. However, if you need multiple instances, it is better to use a constructor function. Here is an example of a book constructor function.

```
function Book(isbn) {
    this.isbn = isbn;
    this.getIsbn = function () {
        return "Isbn is " + this.isbn;
    };
}
```

Properties, including methods, are assigned to the 'this' value in the function's body. In the above example a property and a function are assigned. Also notice that this function is capitalized (i.e. **B**ook); constructor functions are capitalized by convention in JavaScript

To create a new object with this function you use the new operator followed by a function invocation. A function that is invoked this way is called a constructor function whose main purpose is to create and initialize a new object. Here we are creating a new book object:

```
var book = new Book("901-3865");
alert(book.getIsbn());    // => Isbn is 901-3865
```

When new Book() is invoked, JavaScript creates a new empty Object and sets an internal property which specifies that the new object's prototype is Book, that is, the newly created object inherits the prototype of the function. It then passes the Book() function two arguments: the new object as this (as a hidden parameter) and the "901-3865" as isbn. The function, in turn, sets the object's isbn property to «901-3865» and also adds the getIsbn() method to the object. JavaScript returns the newly created object to the caller which then assigns the new object to the book variable.

Each time you invoke new Book(), a new getIsbn method is created which is a rather inefficient because the method is the same for all book instances. A better approach is to let all instances share a single method which can be accomplished by adding getIsbn to the prototype of Book rather than the Book function itself. Here is how this is done:

```
function Book(isbn) {
    this.isbn = isbn;
}
Book.prototype.getIsbn = function () {
    return “Isbn is “ + this.isbn;
};
var book = new Book(“901-3865”);
lert(book.getIsbn());    // => Isbn is 901-3865
```

SUMMARY

- JavaScript classes, introduced in ECMA Script 2015, are primarily JavaScript classes, introduced in ECMA Script 2015, are primarily syntactical sugar over JavaScript's existing prototype-based inheritance. The class syntax does not introduce a new object-oriented inheritance model to JavaScript. JavaScript is a very flexible object-oriented language when it comes to syntax.
- Classes are in fact "special functions", and just as you can define function expressions and function declarations, the class syntax has two components: class expressions and class declarations.
- A class expression is another way to define a class. Class expressions can be named or unnamed.
- The body of a class is the part that is in curly brackets {}. This is where you define class members, such as methods or constructor.
- The static keyword defines a static method for a class. Static methods are called without instantiating their class and cannot be called through a class instance. Static methods are often used to create utility functions for an application.
- A constructor is a function that creates and initializes an object. JavaScript provides a special constructor function called **Object ()** to build the object. The return value of the **Object ()** constructor is assigned to a variable.
- JavaScript is designed on a simple object-based paradigm. An object is a collection of properties, and a property is an association between a name (or key) and a value. A property's value can be a function, in which case the property is known as a method.d.

KNOWLEDGE CHECK

1. **An object's ……………………….. is a reference to another object from which properties are inherited.**
   a. Characteristics
   b. Prototype
   c. Class
   d. Extensible flag
2. **An object's …………………… is a string that categorizes the type of an object.**
   a. Characteristics
   b. Prototype
   c. Class
   d. Extensible flag
3. **An object's ……………………… specifies whether new properties may be added to the object.**
   a. Characteristics
   b. Prototype
   c. Class
   d3. Extensible flag
4. **A …………………….. is an object or class or objects defined by the ECMAScript specification which includes arrays, functions, dates and regular expressions.**
   a. native object
   b. host object
   c. user defined object
   d. remote object
5. **A …………………… object is any object created by the execution of JavaScript code.**
   a. native
   b. host
   c. user defined
   d. remote
6. **Every object contains three object attributes that are _______.**
   a. Prototype, class, object's extensible flag
   b. Prototype, class, objects' parameters

c. Class, parameters, object's extensible flag

d. Native object, Classes and Interfaces and Object's extensible flag

7. `The linkage of a set of prototype objects is known as______

a. prototype stack

b. prototype

c. prototype class

d. prototype chain

8. To know about an object, whether the object is a prototype (or a part of a prototype chain) of another object, the user can use_______

a. ==operator

b. equals() method

c. === operator

d. isPrototypeOf() method

REVIEW QUESTIONS

1. How to use classes in JavaScript.
2. How to create an object with a User-defined function.
3. How the constructor creates the object and assigns properties.
4. Focus on the object () constructor.
5. Discuss about JavaScript native objects.

Check Your Result

1. (b)	2. (c)	3. (d)	4. (a)	5. (c)
6. (a)	7. (d)	8. (d)		

REFERENCES

1. Bruce, Kim B. (2002). Foundations of Object-Oriented Languages: Types and Semantics. Cambridge, MA: MIT Press. ISBN 978-0-262-02523-2.
2. Jamrich, Parsons, June (2015-06-22). New perspectives computer concepts, 2016. Comprehensive. Boston, MA. ISBN 9781305271616. OCLC 917155105.
3. Thomas; Hunt. "Classes, Objects, and Variables". Programming Ruby: The Pragmatic Programmer's Guide. Ruby-Doc.org. Retrieved 2012-04-26.
4. Thomas; Hunt. "Classes and Objects". Programming Ruby: The Pragmatic Programmer's Guide. Ruby-Doc.org. Retrieved 2012-05-08.

CHAPTER 8

JAVASCRIPT BOM

"Technically, web browsers can control what users see, and sites using Javascript can overwrite anything coming from the original authors. Browsers heavily utilize Javascript to create an interactive Internet; sites like YouTube, Facebook, and Gmail could be crippled without it."

– — **Ben Shapiro**

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Browser Object Model
2. the windows objects
3. javascript history object
4. navigator object
5. location object
6. screen object

INTRODUCTION

BOM refers to the browser object model in JavaScript. BOM is used on the Windows screen and communicates with the browser. BOM refers to Windows objects in JavaScript.

Modern browsers have implemented the same methods and properties for JavaScript interactions, often referred to as BOM's methods and properties. A window object is automatically created by the browser.

Various types of BOM (Browser Object Model)

- Windows Object
- History Object
- Navigator Object
- Location Object
- Screen Object
- Storage Object

8.1 BROWSER OBJECT MODEL

Did You Know?

In 2020, an estimated 4.9 billion people used a browser.

The Browser Object Model (BOM) is used to interact with the browser.

The default object of browser is window means you can call all the functions of window by specifying window or directly. For example:

1. window.alert("hello javatpoint");

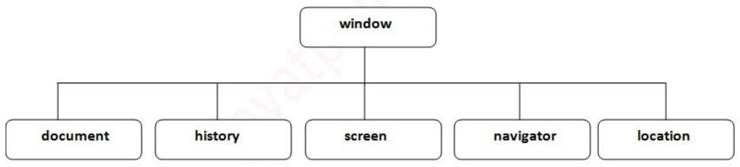
is same as:

1. alert("hello javatpoint");

You can use a lot of properties (other objects) defined underneath the window object like document, history, screen, navigator, location, inner Height, innerWidth,

Remember

The document object represents an html document. It forms DOM (Document Object Model).



8.2 THE WINDOWS OBJECTS

The window object represents a window in browser. An object of window is created automatically by the **browser**.

Window is the object of browser, it is not the object of javascript. The javascript objects are string, array, date etc.

Keyword

Browser is application software for accessing the World Wide Web.

8.2.1 Methods of Window Object

The important methods of window object are as follows:

Methods	**Description**
Alert ()	It displays the popup messages with the ok button.
Confirm ()	It displays the message on the alert box with the OK and cancel button.
Prompt ()	It gets input from the user to display a text message in the dialog box.
Open ()	Opens the current window.
Close ()	Closes the current window.
moveTO ()	Moves the current window.
resizeTo ()	Resizes the current window.
setTimeout ()	It performs an action after a specified time, like calling a function, evaluating expressions, etc.

Remember

If html document contains frame or iframe, browser creates additional window objects for each frame.

8.2.2 Example of Windows Object Methods

We can see the example for all methods, one by one.

The alert () method

This method is used to display an alert message to the user. It displays the alert box containing a **message** with the ok button to proceed. The alert () method takes a single parameter, which is the text displayed in the popup box.

Keyword

Message is an object of communication.

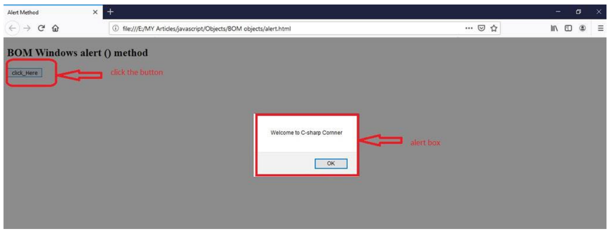
Example

Try it yourself:

1. <!DOCTYPE html>

```
<html>
<head>
    <meta charset="utf-8">

    <title>Alert Method</title>
</head>
<body>
    <h2>BOM Windows alert () method</h2>
    <script type="text/javascript">
        function atl()
        {
        alert("Welcome to C-sharp Cornner");  //alert method
        }
</script>
    <input type="button" value="click_Here" onclick="atl();">
</body>
</html>
```

Output


Alert Method
file:///E:/MY Articles/javascript/Objects/BOM objects/alert.html
click_Here
click the button
OK
alert box

BOM Windows alert () method

Welcome to C-sharp Cornner

The confirm () method

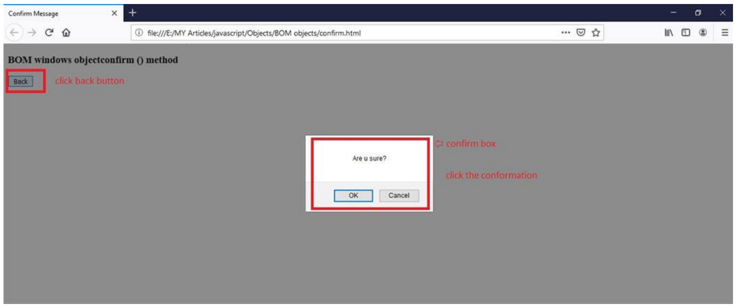
This method is used to display an alert message to the user and verify or accept something. It displays the alert box containing a message with two buttons, OK and cancel button to proceed. When the user clicks OK, the box returns true. If the user clicks Cancel, the box is invalid.

Example

Try it yourself:

```
<!DOCTYPE html>
<html>
<head>
   <meta charset=”utf-8”>
   <title>Confirm Message</title>
</head>
<body>
   <h3>BOM windows objectconfirm () method</h3>
<script type=”text/javascript”>
   function msg(){
   var confm= confirm(“Are u sure?”); //confirmation
      if(v==true)
      {
         alert(“ok”);
      }
      else
      {
         alert(“cancel”);
      }
      }
</script>
<input type="button" value="Back" onclick="msg();"/>
</body>
</html>
```

Output



BOM windows objectconfirm () method

Are u sure?

Keyword

User is a person who utilizes a computer or network service.

The Prompt () Method

The box on a line is often used to evaluate user input before entering a page. When the Instant Box pops up, the **user** must click OK or Cancel to enter the input value and proceed. It has a message and text field. When the user clicks OK, the box returns the input value. When the user clicks Cancel, it returns null.

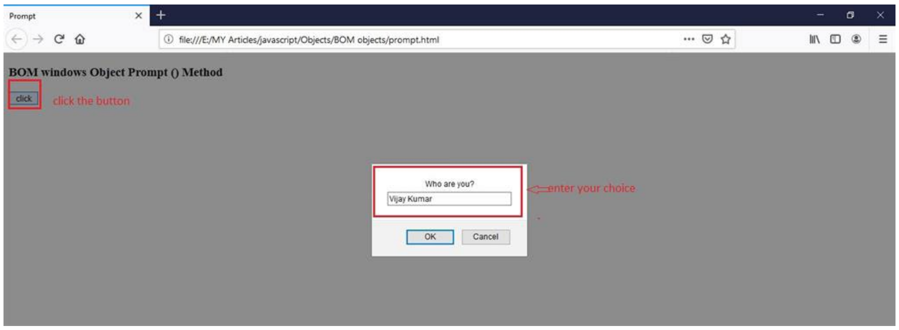
Example

Try it yourself:

```
<!DOCTYPE html>
<html>
<head>
   <title>Prompt </title>
</head>
```

```
<body>
   <h3>BOM windows Object Prompt () Method</h3>
<script type="text/javascript">
   function pmpt()
   {
      var input= prompt("Who are you?");// input box
      alert("I am "+input);
   }
</script>
 <input type="button" value="click" onclick="pmpt();">
</body>
</html>
```

Output


Prompt

file:///E:/MY Articles/javascript/Objects/BOM objects/prompt.html

BOM windows Object Prompt () Method

click

click the button

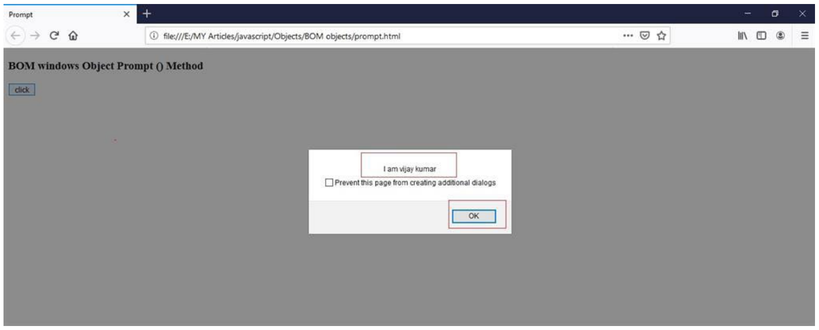
Who are you?

Vijay Kumar

OK Cancel

enter your choice

Afterward, enter a name and the dialog box will be displayed.


Prompt
file:///E:/MY Articles/javascript/Objects/BOM objects/prompt.html
BOM windows Object Prompt () Method
click
Prevent this page from creating additional dialogs
OK

I am vijay kumar

The Open () Method

This displays its content in a new tab or window. In the below example, click the button and it will redirect the new **window**.

Keyword

Window is an opening in a wall, door, roof, or vehicle that allows the passage of light and may also allow the passage of sound and sometimes air.

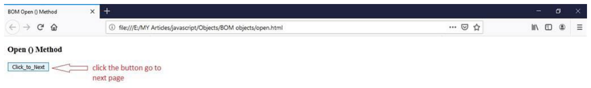
Example

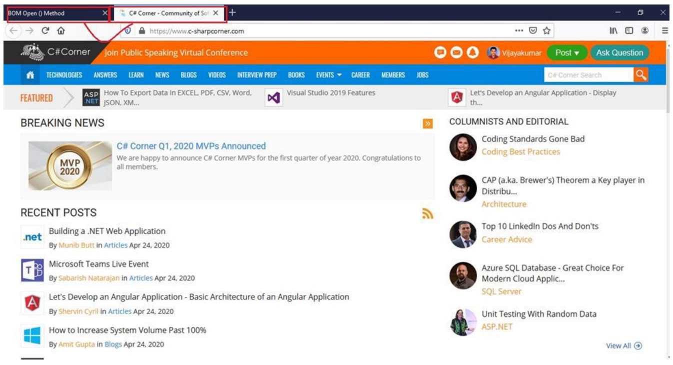
Try it yourself:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>BOM Open () Method</title>
</head>
<body>
    <h3>Open () Method</h3>
    <script type="text/javascript">
    function msg()
    {
   open("https:www.c-sharpcorner.com"); //open the link next window
    }
```

```
</script>
<input type="button" value="Click_to_Next" onclick="msg()"/>
</body>
</html>
```

Output


BOM Open () Method
file:///E:/MY Articles/javascript/Objects/BOM objects/open.html
Open () Method
Click_to_Next
click the button go to next page


BOM Open () Method
C# Corner - Community of Sof
https://www.c-sharpcorner.com
C#Corner
Join Public Speaking Virtual Conference
Vijayakumar
Post
Ask Question
TECHNOLOGIES ANSWERS LEARN NEWS BLOGS VIDEOS INTERVIEW PREP BOOKS EVENTS CAREER MEMBERS JOBS
C# Corner Search
FEATURED

How To Export Data In EXCEL, PDF, CSV, Word, JSON, XM...
Visual Studio 2019 Features
Let's Develop an Angular Application - Display th...

BREAKING NEWS

C# Corner Q1, 2020 MVPs Announced
We are happy to announce C# Corner MVPs for the first quarter of year 2020. Congratulations to all members.

RECENT POSTS

Building a .NET Web Application
By Munib Butt in Articles Apr 24, 2020

Microsoft Teams Live Event
By Sabarish Natarajan in Articles Apr 24, 2020

Let's Develop an Angular Application - Basic Architecture of an Angular Application
By Shervin Cyril in Articles Apr 24, 2020

How to Increase System Volume Past 100%
By Amit Gupta in Blogs Apr 24, 2020

COLUMNISTS AND EDITORIAL

Coding Standards Gone Bad
Coding Best Practices

CAP (a.k.a. Brewer's) Theorem a Key player in Distribu...
Architecture

Top 10 LinkedIn Dos And Don'ts
Career Advice

Azure SQL Database - Great Choice For Modern Cloud Applic...
SQL Server

Unit Testing With Random Data
ASP.NET

View All

The setTimeout () Method

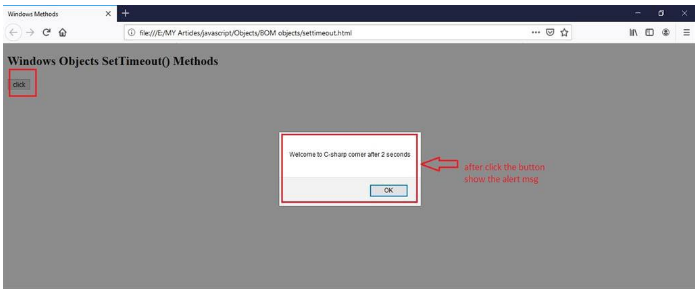
This method is performing the action on after the given milliseconds, like calling function, evaluating expressions etc.

Example

Try it yourself:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Windows Methods</title>
</head>
<body>
    <h2>Windows Objects SetTimeout() Methods </h2>
    <script type="text/javascript">
        function msg()
        {
            setTimeout(
                function()
                {
                    alert("Welcome to C-sharp corner after 2 seconds")
                },2000);
        }
    </script>
<input type="button" value="click" onclick="msg()">
</body>
</html>
```

Output



Windows Objects SetTimeout() Methods

Welcome to C-sharp corner after 2 seconds

Windows Methods
file:///E:/MY Articles/javascript/Objects/BOM objects/settimeout.html
click
OK
after click the button
show the alert msg

8.2.3 Properties in Windows Objects

Property	Descriptions
innerHeight	It returns the height of the browser window.
innerWidth	It returns the width of the browser window.
name	Specifies the name of the window.

8.2.4 Example of Windows Object Properties

The innerHeight and innerWidth

The windows object properties return the height and width of the browser content. We cannot change in height and width as these properties are read-only.

Syntax

```
windows.innerHeight
windows.innerWidth
```

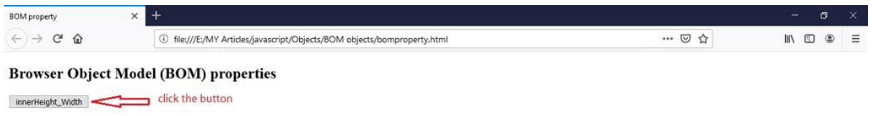
Example

Try it yourself,

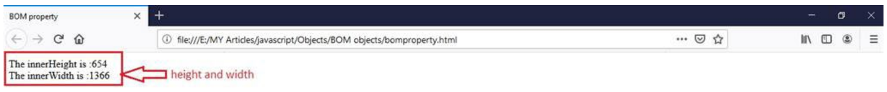
```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <title>BOM property</title>
</head>
<body>
   <h2>Browser Object Model (BOM) properties</h2>
   <script type="text/javascript">
```

```
        function prpt()
        {
         document.write("The innerHeight is :"+window.innerHeight+"<br>"); //inner
  height
         document.write("The innerWidth is :"+window.innerWidth); // inner width
        }
    </script>
    <input type="button" value="innerHeight_Width" onclick="prpt()">
</body>
</html>
```

Output


BOM property
file:///E:/MY Articles/javascript/Objects/BOM objects/bomproperty.html
innerHeight_Width
click the button

Browser Object Model (BOM) properties


BOM property
file:///E:/MY Articles/javascript/Objects/BOM objects/bomproperty.html
height and width

```
The innerHeight is :654
The innerWidth is :1366
```

The Name Property

The name property will return the name of the windows. This **property** is often used to change the name of the window after the window is created.

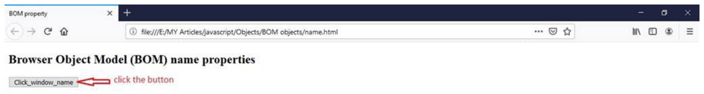
Example

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <title>BOM property</title>
</head>
<body>
   <h2>Browser Object Model (BOM) name properties</h2>
   <script type="text/javascript">
      function prt()
      {
       window.name = "c-sharp corner";
       document.write("The Name is :"+window.name); // windows name
      }
   </script>
     <input type="button" value="Click_window_name" onclick="prt()">
</body>
</html>
```

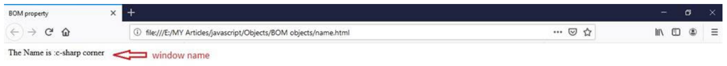
Keyword

Property is a special sort of class member, intermediate in functionality between a field (or data member) and a method.

Output


BOM property
file:///E:/MY Articles/javascript/Objects/BOM objects/name.html
Click_window_name
click the button

Browser Object Model (BOM) name properties


BOM property
file:///E:/MY Articles/javascript/Objects/BOM objects/name.html
window name

The Name is :c-sharp corner

8.3 JAVASCRIPT HISTORY OBJECT

The JavaScript history object represents an array of URLs visited by the user. By using this object, you can load previous, forward or any particular page.

The history object is the window property, so it can be accessed by:

1. window.history

Or,

1. history

Let's see the different usage of history object.

1. *history.back();//for previous page*
2. *history.forward();//for next page*
3. *history.go(2);//for next 2nd page*
4. *history.go(-2);//for previous 2nd page*

8.3.1 Property of JavaScript History Object

There are only 1 property of history object.

No.	Property	Description
1	length	returns the length of the history URLs.

8.3.2 Methods of JavaScript History Object

There are only 3 methods of history object.

No.	Method	Description
1	forward()	loads the next page.
2	back()	loads the previous page.
3	go()	loads the given page number.

8.4 NAVIGATOR OBJECT

Navigator object is used for browser detection. It can be used to get browser information such as version and browser type.

In the Navigator object has the two features,

- Methods and
- Properties

8.4.1 Methods in Navigator Object

The following table contains the methods of the Navigator object in JavaScript.

- Methods
- Description
- javaEnabled ()

  It Specifies whether or not Java is enabled in the browser.

- taintEnabled ()

  It Specifies whether the data field is enabled in the browser.

8.4.2 Examples of Navigator object Methods

We can see the examples of location methods one by one,

The javaEnabled () Method

The Navigator javaEnabled () method is used to return a Boolean value that indicates whether or not Java is enabled in a browser. If Java is enabled in the browser it returns true else it returns false.

Syntax

navigator.javaEnabled ()

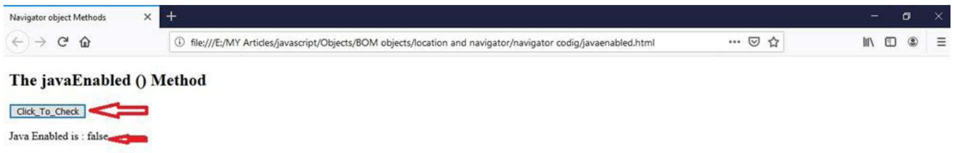
Example

The following programs illustrate the navigator javaEnabled () method in HTML.

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <title>Navigator object Methods</title>
</head>
<body>
   <h2>The javaEnabled () Method</h2>
   <button ondblclick="java()">Click_To_Check</button>
   <p id="javaEnabled"></p>
   <script type="text/javascript">
      function java() {
         var ch ="Java Enabled is : " + navigator.javaEnabled();
         document.getElementById("javaEnabled").innerHTML = ch;
      }
   </script>
</body>
```

```
</html>
```

Output



Navigator object Methods

file:///E:/MY Articles/javascript/Objects/BOM objects/location and navigator/navigator codig/javaenabled.html

The javaEnabled () Method

Java Enabled is : false

The taintEnabled ()Method

The Navigator taintEnable () method was best avoided in Javascript version 1.5, and was later removed to prevent future run-time errors. It returns a Boolean value false value, specifying whether the data-tainting feature is enabled in the browser.

Syntax

navigator.taintEnabled ()

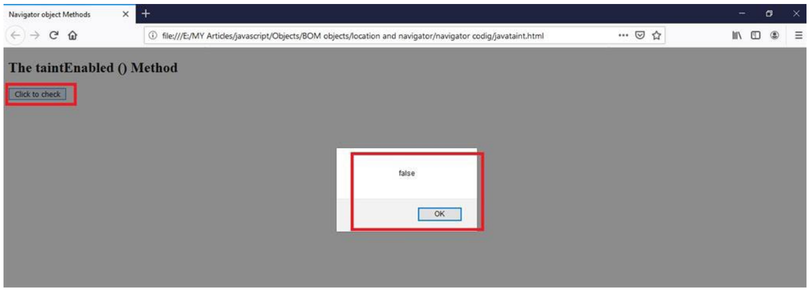
Example

The following programs illustrate the navigator taintEnabled () method in HTML.

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <title>Navigator object Methods</title>
</head>
<body>
   <h2>The taintEnabled () Method</h2>
    <input type="button"value="Click to check" onClick="taint()">

```

```
<script language="JavaScript">
    function taint() {
        var temp = navigator.taintEnabled();
        alert(window.navigator.taintEnabled());
    }
</script>
</body>
</html>
```

Output


Navigator object Methods
file:///E:/MY Articles/javascript/Objects/BOM objects/location and navigator/navigator codig/javataint.html
Click to check
OK

The taintEnabled () Method

false

8.4.3 Property in Navigator Object

The following table contains the properties of the Navigator object in JavaScript.

Properties	Descriptions
appName	It specifies the name of the browser.
appVersion	It specifies the version of the browser being.
appcodeName	It specifies code name of the browser
cookieEnabled	It specifies whether cookies are enabled in the browser or not
onLine	It returns true if the browser is online otherwise false.
geoLocation	It provides a geolocation object that can be used to track the user's status
language	It returns the browser language
platform	It returns which platform browser complied

8.4.4 Examples of Navigator Object Properties

We can see the examples of location properties one by one,

The appName property

It is used to return the name of the browser. This is a read-only property and the values it provides vary from browser to browser.

Syntax

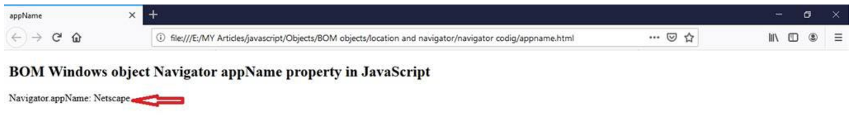
navigator.appName

Example

The following programs illustrate the navigator appName property in HTML.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>appName</title>
</head>
<body>
    <h2>BOM Windows object Navigator appName property in JavaScript</h2>
    <script>
        document.write("Navigator.appName: "+navigator.appName);
    </script>
</body>
</html>
```

Output



```
appName
file:///E:/MY Articles/javascript/Objects/BOM objects/location and navigator/navigator codig/appname.html

BOM Windows object Navigator appName property in JavaScript

Navigator.appName: Netscape
```

The appcodeName property

It is used to return the code name of the browser. This is a read-only property and all modern browsers provide.

Syntax

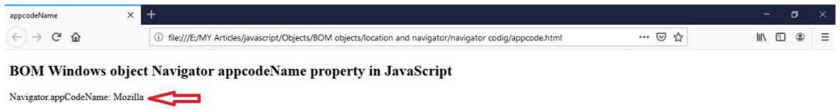
navigator.appcodeName

Example

The following programs illustrate the navigator appcodeName property in HTML.

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <title>appcodeName</title>
</head>
<body>
   <h2>BOM Windows object Navigator appcodeName property in JavaScript</h2>
   <script>
      document.write("Navigator.appCodeName: "+navigator.appCodeName);
   </script>
</body>
</html>
```

Output


appcodeName
file:///E:/MY Articles/javascript/Objects/BOM objects/location and navigator/navigator codig/appcode.html

BOM Windows object Navigator appcodeName property in JavaScript

Navigator.appCodeName: Mozilla

The appversion property

In Navigator appVersion specifies the version of the browser. It returns a string representing the version information of the browser.

Syntax

navigator.appversion

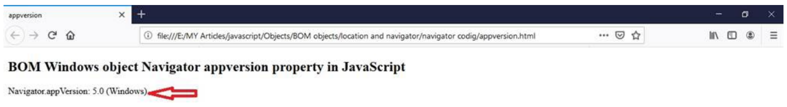
Example

The following programs illustrate the navigator appversion property in HTML.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>appversion</title>
</head>
<body>
    <h2>BOM Windows object Navigator appversion property in JavaScript</h2>
    <script>
        document.write("Navigator.appVersion: "+navigator.appVersion);
    </script>
</body>
```

```
</html>
```

Output


appversion
file:///E:/MY Articles/javascript/Objects/BOM objects/location and navigator/navigator codig/appversion.html

BOM Windows object Navigator appversion property in JavaScript

```
Navigator.appVersion: 5.0 (Windows)
```

The cookieEnabled property

It returns true if the Navigator cookie is enabled, otherwise it is false. It returns a Boolean value that indicates whether or not the browser has enabled cookies.

Syntax

navigator. cookieEnabled

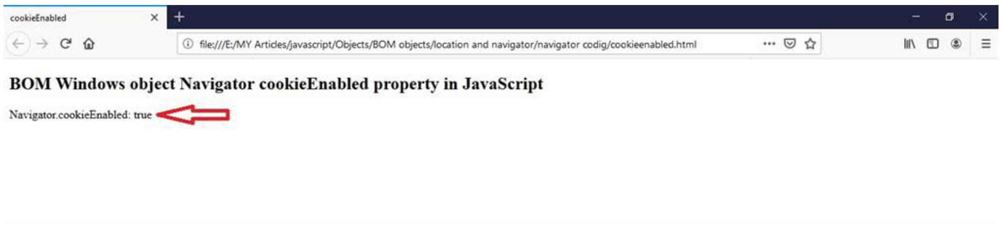
Example

The following programs illustrates the navigator cookieEnabled property in HTML.

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <title>cookieEnabled</title>
</head>
<body>
   <h2>BOM Windows object Navigator cookieEnabled property in JavaScript</h2>
   <script>
      document.write("Navigator.cookieEnabled: "+navigator.cookieEnabled);
   </script>
```

```
</body>
</html>
```

Output


cookieEnabled

file:///E:/MY Articles/javascript/Objects/BOM objects/location and navigator/navigator codig/cookieenabled.html

BOM Windows object Navigator cookieEnabled property in JavaScript

Navigator.cookieEnabled: true

The onLine property

The Navigator Online property is used to return a Boolean value indicating whether a browser is online or false.

Syntax

navigator.onLine

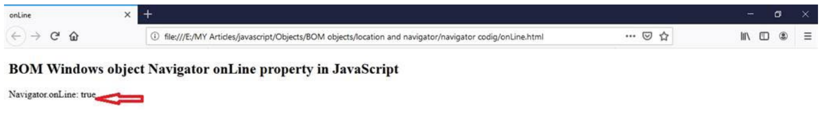
Example

The following programs illustrate the navigator onLine property in HTML.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>onLine</title>
</head>
<body>
    <h2>BOM Windows object Navigator onLine property in JavaScript</h2>
    <script>
```

```
        document.write("Navigator.onLine: "+navigator.onLine);
    </script>
</body>
</html>
```

Output



BOM Windows object Navigator onLine property in JavaScript

Navigator.onLine: true

The geoLocation property

The Navigator geoLocation property provides a geolocation object that can be used to track the user's status.

Syntax

navigator. geoLocation

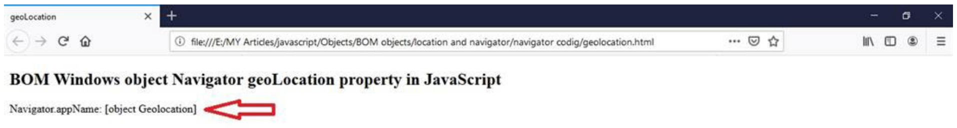
Example

The following programs illustrate the navigator geolocation property in HTML.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>geoLocation</title>
</head>
<body>
```

```
    <h2>BOM Windows object Navigator geoLocation property in JavaScript</h2>
    <script>
      document.write("Navigator.appName: "+navigator.geolocation);
    </script>
</body>
</html>
```

Output



geoLocation

file:///E:/MY Articles/javascript/Objects/BOM objects/location and navigator/navigator codig/geolocation.html

BOM Windows object Navigator geoLocation property in JavaScript

```
Navigator.appName: [object Geolocation]
```

The language property

TheNavigator language property is used to return the language version of the browser. E.g. en-us, fr, en, de.

Syntax

navigator. language

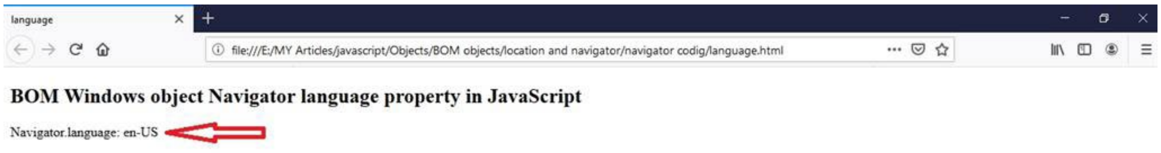
Example

The following programs illustrate the navigator language property in HTML.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>language</title>
```

```
</head>
<body>
    <h2>BOM Windows object Navigator language property in JavaScript</h2>
    <script>
        document.write("Navigator.language: "+navigator.language);
    </script>
</body>
</html>
```

Output


language
file:///E:/MY Articles/javascript/Objects/BOM objects/location and navigator/navigator codig/language.html

BOM Windows object Navigator language property in JavaScript

```
Navigator.language: en-US
```

The platform property

The Navigator Platform property is used to redirect the browser to the compiled site. It returns a string representing the browser's site. E.g. win32, win16, linuxi686

Syntax

navigator. platform

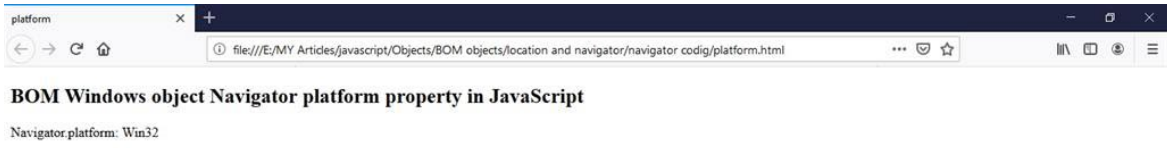
Example

The following programs illustrate the navigator platform property in HTML.

```
<!DOCTYPE html>
<html>
```

```
<head>
    <meta charset="utf-8">
    <title>platform</title>
</head>
<body>
    <h2>BOM Windows object Navigator platform property in JavaScript</h2>
    <script>
        document.write("Navigator.platform: "+navigator.platform);
    </script>
</body>
</html>
```

Output


platform
file:///E:/MY Articles/javascript/Objects/BOM objects/location and navigator/navigator codig/platform.html

BOM Windows object Navigator platform property in JavaScript

```
Navigator.platform: Win32
```

8.5 LOCATION OBJECT

The Location object window in JavaScript enables it to save the information about the current page location or address (URL), and redirect the browser to a new page.

A location object is part of the window object, it accesses through the windows. location. There is no general standard for the location object, but all major browsers support the location object.

The location object has two features:

- Methods and
- Properties

8.5.1 Methods in Location Object

The following table contains the methods of location objects in JavaScript.

Methods	**Description**
assign ()	It Loads a new document on a web page.
reload ()	Using location.href property, reload the current document.
replace ()	This replaces the current document with the specified new document. Not possible to go back to the previous document using your browser 'back' button.

8.5.2 Examples of Location object Methods

We can see the examples of location methods one by one:

The assign () Method

The location assign () method is used for loading a new document in a new page. It is not possible to go back to the previous document using your browser 'back' button.

Syntax

```
location.assign (URL)
```

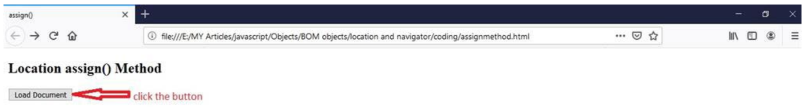
Example

The following programs illustrate the location assign () method in HTML.

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <title>assign()</title>
</head>
<body>
   <h2>Location assign() Method</h2>
```

```
    <button ondblclick=”assign()”></button>
    <script>
        function assign() {
            location.assign(“www.c-sharpcorner.com”); //loads a new page
        }
    </script>
</body>
</html>

```

Output



assign()

file:///E:/MY Articles/javascript/Objects/BOM objects/location and navigator/coding/assignmethod.html

Location assign() Method

The reload () Method

This method reloads the current document, used in the location.href property. It’s like the refresh button in the browser.

Syntax

```
location.reload(URL)
```

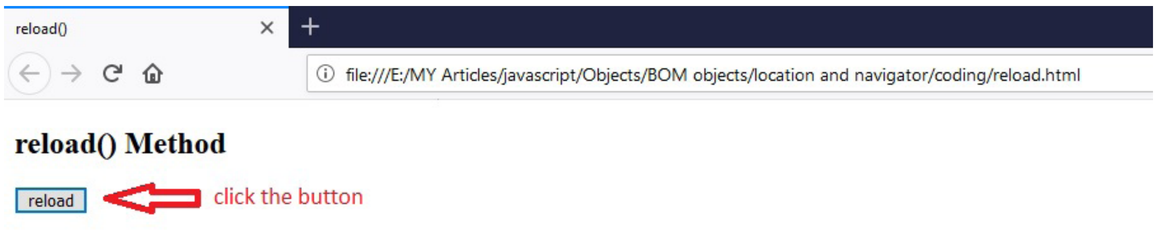
Example

The following programs illustrate the location reload () method in HTML.

```
<!DOCTYPE html>
```

```
<html>
<head>
    <meta charset="utf-8">
    <title>reload()</title>
</head>
<body>
    <h2>reload() Method</h2>
    <button onlclick="reload()">reload</button>
    <script>
        function reload() {
            location.reload(); //reload a page
        }
    </script>
</body>
</html>

```

Output


reload()
file:///E:/MY Articles/javascript/Objects/BOM objects/location and navigator/coding/reload.html
reload() Method
reload
click the button

The replace () Method

This replaces the current document with the specified new document. The replace () method in the location object is used to replace the current page with another page.

Syntax

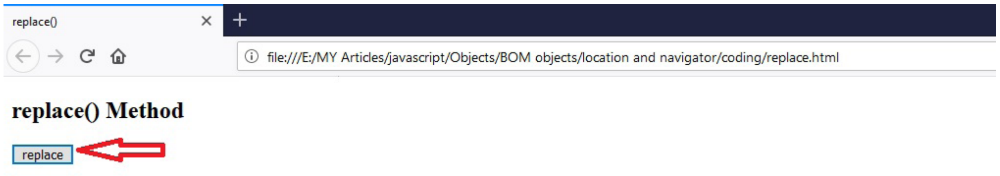
```
location.replace(URL)
```

Example

The following programs illustrate the location replace () method in HTML.

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <title>replace()</title>
</head>
<body>
   <h2>replace() Method</h2>
   <button onlclick="replace()">replace</button>
   <script>
      function replace() {
            location.replace("https://www.c-sharpcorner.com"); //replace the current page to new page
      }
   </script>
</body>
</html>

```

Output


replace()
file:///E:/MY Articles/javascript/Objects/BOM objects/location and navigator/coding/replace.html
replace() Method
replace

8.5.3 Property in Location Object

The following table contains the properties of location objects in JavaScript.

Properties	Description
herf	The location href property in HTML is used to set or redirect the current (URL) of the current page.
hash	The string beginning with # specifies an anchor name in an HTTP URL.
host	It refers to a string that includes the hostname and port strings
hostname	It Provides the domain name, sub-domain name and server name of the web host.
search	It specifies the search area of the URL.
port	It returns the port number of the current page.
pathname	It returns the pathname (URL) and file name (URL) of the current page.

8.5.4 Examples of Location object Properties

We can see the below examples of location object properties.

The href property

The location href property in HTML is used to set or redirect the current (URL) of the current page. The location href returns the full URL of the page and protocol.

Syntax

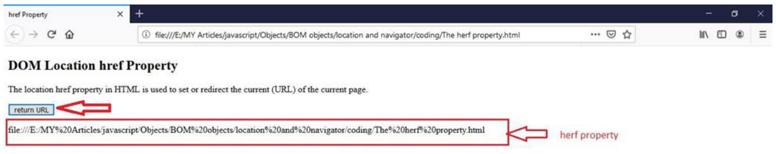
```
1.   location.href = URL
```

Example

The following programs illustrate the location href property.

```
<!DOCTYPE html>
<html>
<head>
   <title>href Property</title>
</head>
<body>
   <h2>DOM Location href Property</h2>
   <p>
      The location href property in HTML is used to set or redirect the current (URL) of the current page.
   </p>
   <button ondblclick="myhref()">return URL</button>
   <p id="value"></p>
   <script>
      function myhref() {
         var href = location.href;
         document.getElementById("value").innerHTML = herf; //get the location in the current document
      }
   </script>
</body>
</html>
```

Output



file:///E:/MY Articles/javascript/Objects/BOM objects/location and navigator/coding/The herf property.html

DOM Location href Property

The location href property in HTML is used to set or redirect the current (URL) of the current page.

file:///E:/MY%20Articles/javascript/Objects/BOM%20objects/location%20and%20navigator/coding/The%20herf%20property.html

The hash property

The location hash property string beginning with # specifies an anchor name in an HTTP URL.

Syntax

```
location.hash
```

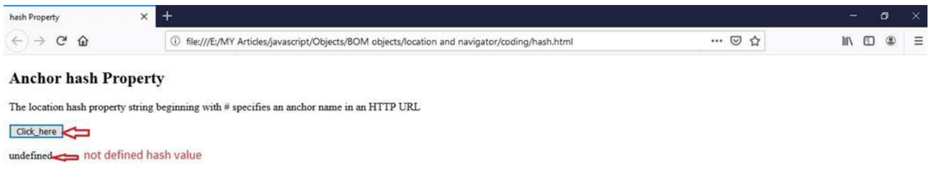
Example

The following programs illustrate the location hash property.

```
<!DOCTYPE html>
<html>
<head>
   <title>hash Property</title>
</head>
<body>
   <h2>Anchor hash Property</h2>
    <p id="hash">The location hash property string beginning with # specifies an anchor name in an HTTP URL</p>
<button onclick="myhash()">Click_here</button>
<p id="hash1">  </p>
<script>
```

```
    function myhash()
    {
        var has =document.getElementById("hash").hash;
        document.getElementById("hash1").innerHTML = has;;
    }
</script>
</body>
</html>
```

Output



```
hash Property
file:///E:/MY Articles/javascript/Objects/BOM objects/location and navigator/coding/hash.html

Anchor hash Property

The location hash property string beginning with # specifies an anchor name in an HTTP URL

Click_here

undefined
```

The hostname property

The location hostname property is used to return the hostname of the current URL. It contains the server name, domain name, IP address of a URL.

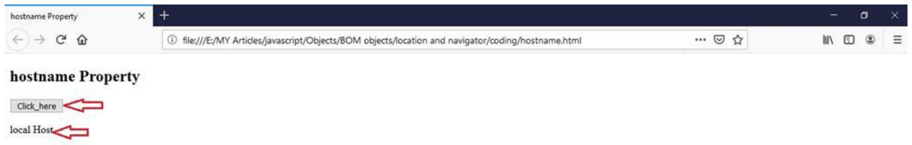
Example

The following programs illustrate the location hostname property.

```
<!DOCTYPE html>
<html>
<head>
    <title>hostname Property</title>
</head>
```

```
<body>
   <h2>hostname Property</h2>
<button onclick="host()">Click_here</button>
<p id="host1"> </p>
<script>
   function host()
   {
      var host2 =location.hostname;
      document.getElementById("host1").innerHTML = host2;;
   }
</script>
</body>
</html>
```

Output



hostname Property

file:///E:/MY Articles/javascript/Objects/BOM objects/location and navigator/coding/hostname.html

hostname Property

Click_here

local Host

The pathname property

The location pathname property is used to find the path name and file name of the current page.

Syntax

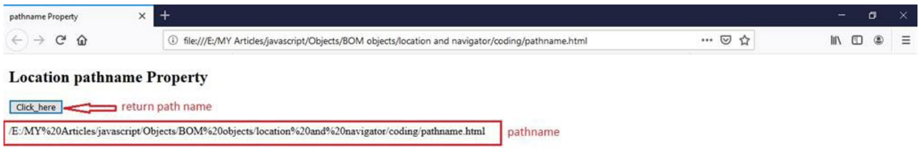
```
Location.pathname = path
```

Example

The following programs illustrate the location pathname property.

```
<!DOCTYPE html>
<html>
<head>
    <title>pathname Property</title>
</head>
<body>
    <h2>Location pathname Property</h2>
    <button ondblclick=”mypath()”>Click_here</button>
    <p id=”path”></p>
    <script>
        function mypath()
        {
            var p1 = location.pathname;
            document.getElementById(“path”).innerHTML = p1;
        }
    </script>
</body>
</html>
```

Output


pathname Property
file:///E:/MY Articles/javascript/Objects/BOM objects/location and navigator/coding/pathname.html
Click_here
return path name
pathname

Location pathname Property

```
/E:/MY%20Articles/javascript/Objects/BOM%20objects/location%20and%20navigator/coding/pathname.html
```

The Port Property

The location port property returns the port number of the current URL.

Syntax

```
location.port
location.port = 8090
```

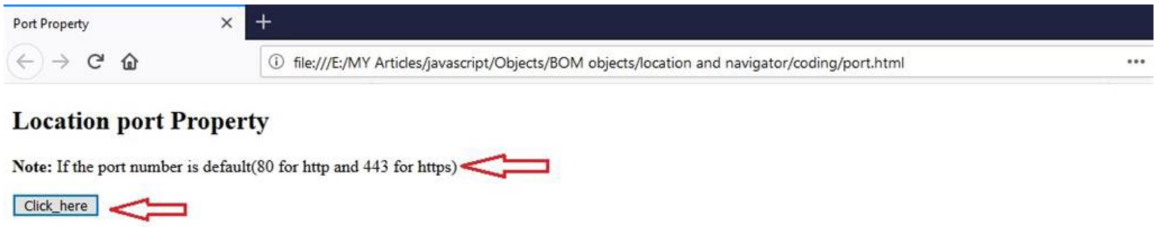
Example

The following programs illustrate the location hash property.

```
<!DOCTYPE html>
<html>
<head>
    <title>Port Property</title>
</head>
<body>
    <h2>Location port Property</h2>
     <p><b>Note: </b>If the port number is default (80 for http and 443 for https)</p>

    <button ondblclick="port()">Click_here</button>
    <p id="path"></p>
```

```
<script>
    function port()
    {
        var port2 = location.port;
        var p1 = location.port = 8080;
        document.getElementById("path").innerHTML = port2;
        document.write(p1);
    }
</script>
</body>
</html>
```

Output



Port Property

file:///E:/MY Articles/javascript/Objects/BOM objects/location and navigator/coding/port.html

Location port Property

Note: If the port number is default(80 for http and 443 for https)

Click_here

8.6 SCREEN OBJECT

The JavaScript screen object holds information of browser screen. It can be used to display screen width, height, colorDepth, pixelDepth etc.

The navigator object is the window property, so it can be accessed by:

```
window.screen
```

Or,

```
screen
```

8.6.1 Property of JavaScript Screen Object

There are many properties of screen object that returns information of the browser.

No.	Property	Description
1	width	returns the width of the screen
2	height	returns the height of the screen
3	availWidth	returns the available width
4	availHeight	returns the available height
5	colorDepth	returns the color depth
6	pixelDepth	returns the pixel depth.

8.6.2 Example of JavaScript Screen Object

Let's see the different usage of screen object.

```
<script>
document.writeln("<br/>screen.width: "+screen.width);
document.writeln("<br/>screen.height: "+screen.height);
document.writeln("<br/>screen.availWidth: "+screen.availWidth);
document.writeln("<br/>screen.availHeight: "+screen.availHeight);
document.writeln("<br/>screen.colorDepth: "+screen.colorDepth);
document.writeln("<br/>screen.pixelDepth: "+screen.pixelDepth);
</script>
```

Test it Now

```
screen.width: 1366
screen.height: 768
screen.availWidth: 1366
screen.availHeight: 728
screen.colorDepth: 24
screen.pixelD
```

SUMMARY

- BOM refers to the browser object model in JavaScript. BOM is used on the Windows screen and communicates with the browser. BOM refers to Windows objects in JavaScript.
- The window object represents a window in browser. An object of window is created automatically by the browser.
- The windows object properties return the height and width of the browser content. We cannot change in height and width as these properties are read-only.
- The name property will return the name of the windows. This property is often used to change the name of the window after the window is created.
- The JavaScript history object represents an array of URLs visited by the user. By using this object, you can load previous, forward or any particular page.
- Navigator object is used for browser detection. It can be used to get browser information such as version and browser type.
- The Navigator javaEnabled () method is used to return a Boolean value that indicates whether or not Java is enabled in a browser. If Java is enabled in the browser it returns true else it returns false.
- The Navigator taintEnable () method was best avoided in Javascript version 1.5, and was later removed to prevent future run-time errors. It returns a Boolean value false value, specifying whether the data-tainting feature is enabled in the browser.
- The Navigator geoLocation property provides a geolocation object that can be used to track the user’s status.
- The Location object window in JavaScript enables it to save the information about the current page location or address (URL), and redirect the browser to a new page.
- A location object is part of the window object, it accesses through the windows. location. There is no general standard for the location object, but all major browsers support the location object.
- The location assign () method is used for loading a new document in a new page. It is not possible to go back to the previous document using your browser ‘back’ button.
- The location href property in HTML is used to set or redirect the current (URL) of the current page. The location href returns the full URL of the page and protocol.
- The JavaScript screen object holds information of browser screen. It can be used to display screen width, height, colorDepth, pixelDepth etc.

KNOWLEDGE CHECK

1. **What is the full form of BOM?**
   a. Browser Object Method
   b. Browser Object Model
   c. Browser Oriented Method
   d. Browser Oriented M
2. **Which is window method is used to move the current window?**
   a. Move()
   b. Move to()
   c. Window move()
   d. Window.move to()
3. **The URL property belongs to which of the following object?**
   a. Document
   b. Element
   c. Location
   d. Event
4. **What does the location property represent?**
   a. Current DOM object
   b. Current URL
   c. Both DOM object and URL
   d. Document
5. **Which among the following is not a property of the Location object?**
   a. protocol
   b. host
   c. hostee
   d. hostname
6. **What is the return type of the hash property?**
   a. Query string
   b. Packets
   c. String
   d. Fragment identifier

7. **Which is the method that removes the current document from the browsing history before loading the new document?**
   a. modify()
   b. assign()
   c. replace()
   d. remove()
8. **Why is the replace() method better than the assign() method?**
   a. Reliable
   b. Highly manageable
   c. More efficient
   d. Handles unconditional loading
9. **What is the purpose of the assign() method?**
   a. Only loading
   b. Loading of window and display
   c. Displays already present window
   d. Unloading of window
10. **The history property belongs to which object?**
   a. Element
   b. Window
   c. History
   d. Location

REVIEW QUESTIONS

1. What is the full form of BOM browser?
2. What are different types of window object?
3. Which is the method of history object?
4. What is the navigator object?
5. What are two methods of location object?

Check Your Result

1. (b)	2. (d)	3. (a)	4. (b)	5. (c)
6. (d)	7. (c)	8. (d)	9. (b)	10. (c)

REFERENCES

1. Enzer, Larry (31 August 2018). "The Evolution of the Web Browsers". Monmouth Web Developers. Retrieved 31 August 2018.
2. Gillies, James; Cailliau, R. (2000). How the Web was Born: The Story of the World Wide Web. Oxford University Press. pp. 6. ISBN 0192862073.
3. Stewart, William. "Web Browser History". Archived from the original on 20 January 2011.

CHAPTER 9

JAVASCRIPT EVENTS

"JavaScript is the only language that I'm aware of that people feel they don't need to learn before they start using it."

— Douglas Crockford

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Discuss about javascript events
2. Describe the types of javascript event
3. Understand the javascript addEventListener()
4. Learn about javascript onclick event
5. Understand the javascript dblclick event
6. Know the javascript onload
7. Learn about javascript onresize event

```
<script>
    import sidebarController from './sidebarController.js';
    import mementoes from '../../store.js';
    import './sidebar.scss';
</script>

<div class="sidebar">
    <button class="add-memento" on:click={sidebarController [illegible]

    <ul class="mementoes">
        {#each $mementoes as memento}
            <li class="memento-item" [illegible]
                on:click={() => sidebarController [illegible]
                {memento.title}
            </li>
        {/each}
    </ul>
</div>
```

INTRODUCTION

The Javascript interacts with the documents HTML code using ***events***, which are triggered when a particular moment of interest happens in the document or the browser window.

Javascript Events makes the webpages interactive and responsive. These events are asynchronous (i.e it can occur anytime). Most events are triggered by user action but there are some exceptions like the event load.

When an event occurs there are some default action which the browser takes (like clicking on a link open up the location specified in attribute *href*).An event may be a click, mouseover, keystroke etc.

Javascript responds to events by calling a function which performs some task as defined in the function.

As of today, Javascript has three event models for programming events: the inline model, the Scripting Model and the DOM2 Model

Javascript can respond to the following type of events: Mouse Actions, Keyboard Actions, Form Actions, Page Loads, Time Intervals and Errors.

9.1 JAVASCRIPT EVENTS

The change in the state of an object is known as an Event. In html, there are various events which represents that some activity is performed by the user or by the browser. When javascript code is included in HTML, js react over these events and allow the execution. This process of reacting over the events is called Event Handling. Thus, js handles the HTML events via Event Handlers.

When a user clicks over the browser, add js code, which will execute the task to be performed on the event.

Some of the HTML events and their event handlers are:

- Mouse Events

Event Performed	Event Handler	Description
click	onclick	When mouse click on an element
mouseover	onmouseover	When the cursor of the mouse comes over the element
mouseout	onmouseout	When the cursor of the mouse leaves an element
mousedown	onmousedown	When the mouse button is pressed over the element
mouseup	onmouseup	When the mouse button is released over the element
mousemove	onmousemove	When the mouse movement takes place.

Click Event

```
<html>
<head> Javascript Events </head>
<body>
<script language="Javascript" type="text/Javascript">
   <!--
   function clickevent()
   {
      document.write("This is JavaTpoint");
   }
   //-->
</script>
<form>
<input type="button" onclick="clickevent()" value="Who's this?"/>
</form>
</body>
</html>
```

MouseOver Event

```
<html>
<head>
<h1> Javascript Events </h1>
</head>
<body>
<script language="Javascript" type="text/Javascript">
    <!--
    function mouseoverevent()
    {
        alert("This is JavaTpoint");
    }
    //-->
</script>
<p onmouseover="mouseoverevent()"> Keep cursor over me</p>
</body>
</html>
```

- Keyboard Events

Event Performed	Event Handler	Description
Keydown & Keyup	onkeydown & onkeyup	When the user press and then release the key

Keydown Event

```
<html>
<head> Javascript Events</head>
```

```
<body>
<h2> Enter something here</h2>
<input type="text" id="input1" onkeydown="keydownevent()"/>
<script>
<!--
   function keydownevent()
   {
      document.getElementById("input1");
      alert("Pressed a key");
   }
//-->
</script>
</body>
</html>
```

- Form Events

Event Performed	Event Handler	Description
focus	onfocus	When the user focuses on an element
submit	onsubmit	When the user submits the form
blur	onblur	When the focus is away from a form element
change	onchange	When the user modifies or changes the value of a form element

Focus Event

```
<html>
<head> Javascript Events</head>
```

```
<body>
<h2> Enter something here</h2>
<input type="text" id="input1" onfocus="focusevent()"/>
<script>
<!--
   function focusevent()
   {
      document.getElementById("input1").style.background=" aqua";
   }
//-->
</script>
</body>
</html>
```

- Window/Document Events

Event Performed	Event Handler	Description
load	onload	When the browser finishes the loading of the page
unload	onunload	When the visitor leaves the current webpage, the browser unloads it
resize	onresize	When the visitor resizes the window of the browser

Load Event

```
<html>
<head>Javascript Events</head>
</br>
<body onload="window.alert('Page successfully loaded');">
```

```
<script>
<!--
document.write(“The page is loaded successfully”);
//-->
</script>
</body>
</html>
```

9.2 JAVASCRIPT EVENT TYPES

These are the top 8 types of JavaScript Event discussed below:

9.2.1 User Interface Events

These occur as the result of any interaction with the browser window rather than the HTML page. In these events, we attach the event listener to the window object, not the document object. The various UI events are as follows.

- **load**: The load event fires when the webpage finishes loading. It can also fire on nodes of elements like *images, scripts, or objects.*
- **unload**: This event fires before the users leave the page, i.e., the webpage is unloading. Page unloading usually happens because a new page has been requested.
- **error**: This event fires when the browser encounters a JavaScript error or an asset that doesn't exist.
- **resize**: It fires when we resize the browser window. But browsers repeatedly fire this event, so avoid using this event to trigger complicated code; it might make the page less responsive.
- **scroll**: This event fires when the user scrolls up/down on the browser window. It can relate to the entire page or a specific element on the page.

9.2.2 Focus and Blur Events

These events fire when the HTML elements you can interact with gain/ lose focus. They are most commonly used in forms and especially helpful when you want to do the following tasks:

- To show tips or feedback to users as they interact with an element within a form. The tips are usually shown in the elements other than the one the user is interacting with.
- To trigger form validation as a user moves from one control to the next without waiting to submit the form.

The different focus and blur events are as follows:

- **focus**: This event fires, for a specific DOM node, when an element gains focus.
- **blur**: This fires, for a specific DOM node, when an element loses focus.
- **focusin**: This event is the same as the focus event. But Firefox doesn't yet support the focusin event.
- **focusout**: This is the same event as the blur event. This is a new event type in JavaScript, thus not supported in Firefox right now.

The focus and blur events use the capture approach, while the focusin and focusout events use both capture and bubble approach of the event flow.

Keyword

Touchscreen is the assembly of both an input ('touch panel') and output ('display') device.

9.2.3 Mouse Events

These events fire when the mouse moves or the user clicks a button. All the elements of the page support these events and use the bubbling approach. These actions work differently on **touchscreen** devices. Preventing the default behavior of mouse events can cause unexpected results. The various mouse events of JavaScript are as follows:

- click: This event fires when the user clicks on the primary mouse button (usually the left button). This event also fires if the user presses the Enter key on the keyboard when an element has focus.

Touch-screen: A tap on the screen acts like a single left-click.

- dblclick: This event fires when the user clicks the primary mouse button, in quick succession, twice.

Touch-screen: A double-tap on the screen acts like a double left-click.

Accessibility: You can add the above two events to any element, but it's better to apply it only on the items that are usually clicked, or it will not be accessible through keyboard navigation. All the mouse events discussed below cannot be triggered by the keyboard.

- mousedown: It fires when the user clicks down on any mouse button.

Touch-screen: You can use the **touchstart** event.

- Mouseup: It fires when the user releases a mouse button.

Touch-screen: You can use the **touchend** event.

We have separate **mousedown** and **mouseup** events to add drag-and-drop functionality or controls in game development. Don't forget a **click** event is the combination of **mousedown** and **mouseup** events.

- mouseover: It fires when the user moves the cursor, which was outside an element before, inside the element. We can say that it fires when we move the cursor over the element.
- mouseout: It fires when the user moves the cursor, which was inside an element before, outside the element. We can say that it fires when the cursor moves off the element.

The mouseover and mouseout events usually change the appearance of graphics on our webpage. A preferred alternative to this is to use the **CSS: hover** pseudo-class.

- mousemove: It fires when the user moves the cursor around the element. This event is frequently triggered.

9.2.4 Keyboard Events

These events fire on any kind of device when a user interacts with a keyboard.

- **input**: This event fires when the value of an **<input>** or a **<textarea>** changes (doesn't fire for deleting in IE9). You can use keydown as a fallback in older browsers.
- **keydown**: It fires when the user presses any key in the keyboard. If the user holds down the key, this event fires repeatedly.
- **keypress**: It fires when the user presses a key that results in printing a character on the screen. This event fires repeatedly if the user holds down the key. This event will not fire for the enter, tab, or arrow keys; the **keydown** event would.
- **keyup**: The keyup event fires when the user releases a key on the keyboard.

The keydown and keypress events fire before a character appears on the screen, the keyup fires after it shows.

To know the key pressed when you use the keydown and keypress events, the event object has a **keyCode** property. This property, instead of returning the letter for that key, returns the ASCII code of the lowercase for that key.

Did You Know?

Earlier, Herman Hollerith developed the first keypunch devices, which soon evolved to include keys for text and number entry akin to normal typewriters by the 1930s.

9.2.5 Form Events

These events are common while using forms on a **webpage**. In particular, we see the submit event mostly in form of validation (checking form values). As described in our tutorial; *Features of JavaScript*, if the users miss any required information or enter incorrect input, validation before sending the data to the server is faster. The list below explains the different form of events available to the user.

- **submit**: This event fires on the node representing the **<form>** element when a user submits a form.
- **change**: It fires when the status of various form elements change. This is a better option than using the click event because clicking is not the only way users interact with the form.
- **input**: The input event is very common with the <input> and the <textarea> elements.

We often use the focus and blur events with forms, but they are also available in conjunction with other elements like links.

Keyword

Webpage is a hypertext document provided by a website and displayed to a user in a web browser.

9.2.6 Mutation Events and Observers

Whenever the structure of the DOM tree changes, it triggers a mutation event. The change in the tree may be due to the addition or removal of a DOM node through your script. But these have an alternative that will replace them: mutation observers. The following are the numerous mutation events in JavaScript.

- **DOMNodeInserted**: It fires when the script inserts a new node in the DOM tree using *appendChild(), replaceChild(), insertBefore(), etc.*
- **DOMNodeRemoved**: This event fires when the script removes an existing node from the tree using *removeChild(), replaceChild(), etc.*

- **DOMSubtreeModified**: It fires when the structure of the DOM tree changes i.e. the above two events occur.
- **DOMNodeInsertedIntoDocument**: This event fires when the script inserts a node in the DOM tree as the descendant of another node already in the document.
- **DOMNodeRemovedFromDocument**: This event fires when the script removes a node from the DOM tree as the descendant of another node already in the document.

The problem with the mutation events is that lots of changes to your page can make your page feel slow or unresponsive. These can also trigger other event listeners, modifying DOM and leading to more mutation events firing. This is the reason for introducing mutation observers to the script.

Mutation observers wait until the script finishes its current task before reacting, then reports the changes in a batch (not one at a time). This reduces the number of events that fire when you change the DOM tree through your script. You can also specify which changes in the DOM you want them to react to.

9.2.7 HTML5 Events

These are the page-level events included in the versions of the HTML5 specialization. New events support more recent devices like phones and tablets. They respond to events such as gestures and movements. You will understand them better after you master the above concepts, thus they are not discussed for now. Work with the events below for now and when you are a better developer, you can search for other events available. The three HTML5 events we will learn are as follows:

- **DOMContentLoaded**: This event triggers when the DOM tree forms i.e. the script is loading. Scripts start to run before all the resources like *images, CSS, and JavaScript loads*. You can attach this event either to the **window** or the **document** objects.
- **hashchange**: It fires when the URL hash changes without refreshing the entire window. Hashes (#) link specific parts (known as **anchors**) within a page. It works on the **window** object; the event object contains both the **oldURL** and the **newURL** properties holding the URLs before and after the hashchange.
- **beforeunload**: This event fires on the window object just before the page unloads. This event should only be helpful for the user, not encouraging them to stay on the page. You can add a dialog box to your event, showing a message alerting the users like their changes are not saved.

9.2.8 CSS Events

These events trigger when the script encounters a CSS element. As CSS is a crucial part of web development, the developers decided to add these events to js to make working with CSS easier. Some of the most common CSS events are as follows:

- **transitionend**: This event fires when a CSS transition ends in a program. It is useful to notify the script of the end of transition so that it can take further action.
- **animationstart**: These events fire when CSS animation starts in the program.
- **animationiteration**: This event occurs when any CSS animation repeats itself. With this event, we can determine the number of times an animation iterates in the script.
- **animationend**: It fires when the CSS animation comes to an end in the program. This is useful when we want to act just after the **animation** process finishes.

Keyword

Animation is a method in which figures are manipulated to appear as moving images.

9.3 JAVASCRIPT ADDEVENTLISTENER()

The **addEventListener()** method is used to attach an **event handler** to a particular element. It does not override the existing event handlers. Events are said to be an essential part of the JavaScript. A web page responds according to the event that occurred. Events can be user-generated or generated by API's. An event listener is a JavaScript's procedure that waits for the occurrence of an event.

The addEventListener() method is an inbuilt function of JavaScript. We can add multiple event handlers to a particular element without overwriting the existing event handlers.

Keyword

Event handler is a callback subroutine that handles inputs received in a program (called a listener in Java and JavaScript).

9.3.1 Syntax

```
element.addEventListener(event, function, useCapture);
```

Although it has three parameters, the parameters *event* and *function* are widely used. The third parameter is optional to define. The values of this function are defined as follows.

9.3.2 Parameter Values

event: It is a required parameter. It can be defined as a string that specifies the event's name.

function: It is also a required parameter. It is a JavaScript function which responds to the event occur.

useCapture: It is an optional parameter. It is a Boolean type value that specifies whether the event is executed in the bubbling or capturing phase. Its possible values are **true** and **false**. When it is set to true, the event handler executes in the capturing phase. When it is set to false, the handler executes in the bubbling phase. Its default value is **false**.

Let's see some of the illustrations of using the addEventListener() method.

Remember

Do not use any prefix such as "on" with the parameter value. For example, Use "click" instead of using "onclick".

Example

It is a simple example of using the addEventListener() method. We have to click the given HTML button to see the effect.

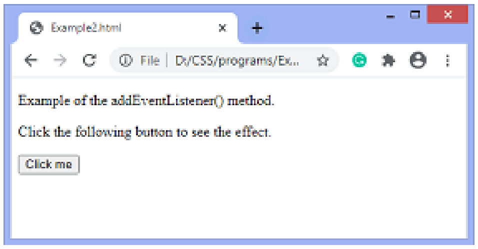
```
<!DOCTYPE html>
<html>
<body>
<p> Example of the addEventListener() method. </p>
<p> Click the following button to see the effect. </p>
<button id = “btn”> Click me </button>
<p id = “para”></p>
<script>
document.getElementById(“btn”).addEventListener(“click”, fun);
function fun() {
document.getElementById(“para”).innerHTML = “Hello World” + “<br>” + “Welcome to the javaTpoint.com”;
}
```

13. </script>

14. </body>

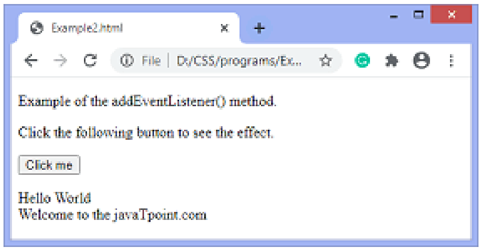
15. </html>

Output



```
Example of the addEventListener() method.

Click the following button to see the effect.
```

After clicking the given HTML button, the output will be -



```
Example of the addEventListener() method.

Click the following button to see the effect.

Hello World
Welcome to the javaTpoint.com
```

Now, in the next example we will see how to add many events to the same element without overwriting the existing events.

Example

In this example, we are adding multiple events to the same element.

1. <!DOCTYPE html>

2. <html>

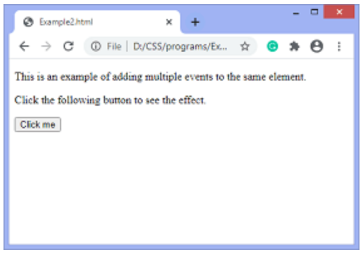
3. <body>

4. <p> This is an example of adding multiple events to the same element. **</p>**

5. <p> Click the following button to see the effect. **</p>**

```
<button id = “btn”> Click me </button>
<p id = “para”></p>
<p id = “para1”></p>
<script>
function fun() {
    alert(“Welcome to the javaTpoint.com”);
}

function fun1() {
   document.getElementById(“para”).innerHTML =  “This is second function”;

}
function fun2() {
   document.getElementById(“para1”).innerHTML =  “This is third function”;
}
var mybtn = document.getElementById(“btn”);
mybtn.addEventListener(“click”, fun);
mybtn.addEventListener(“click”, fun1);
mybtn.addEventListener(“click”, fun2);
</script>
</body>
</html>
```

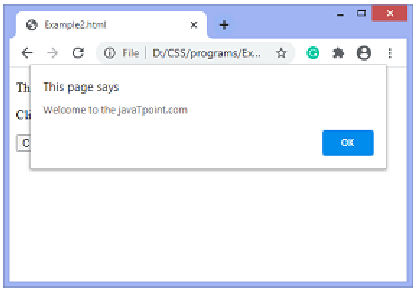
Output



This is an example of adding multiple events to the same element.

Click the following button to see the effect.

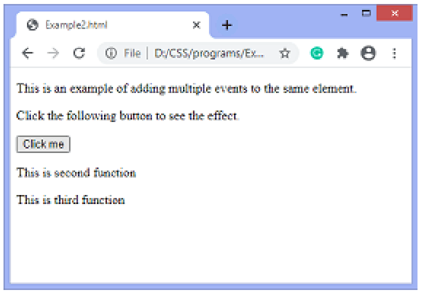
Now, when we click the button, an alert will be displayed. After clicking the given HTML button, the output will be -



This page says

Welcome to the javaTpoint.com

When we exit the alert, the output is -



This is an example of adding multiple events to the same element.

Click the following button to see the effect.

This is second function

This is third function

Example

In this example, we are adding multiple events of a different type to the same element.

1. <!DOCTYPE html>
2. **<html>**
3. **<body>**

```
<p> This is an example of adding multiple events of different type to the same element. </p>
<p> Click the following button to see the effect. </p>
<button id = "btn"> Click me </button>
<p id = "para"></p>
<script>
function fun() {
   btn.style.width = "50px";
   btn.style.height = "50px";
   btn.style.background = "yellow";
   btn.style.color = "blue";
}

function fun1() {
  document.getElementById("para").innerHTML =  "This is second function";

}
function fun2() {
   btn.style.width = "";
   btn.style.height = "";
   btn.style.background = "";
   btn.style.color = "";
}
var mybtn = document.getElementById("btn");
mybtn.addEventListener("mouseover", fun);
```

28. mybtn.addEventListener("click", fun1);

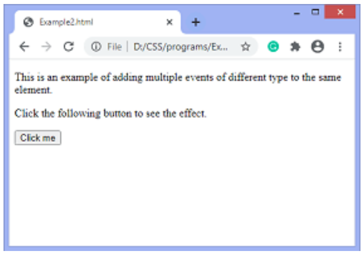
29. mybtn.addEventListener("mouseout", fun2);

30. </script>

31. </body>

32. </html>

Output



Example2.html

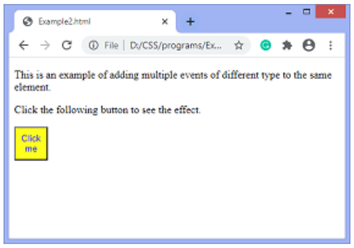
File | D:/CSS/programs/Ex...

This is an example of adding multiple events of different type to the same element.

Click the following button to see the effect.

Click me

When we move the cursor over the button, the output will be -



Example2.html

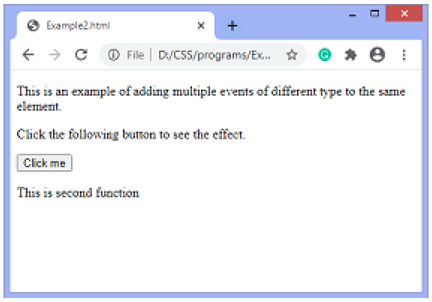
File | D:/CSS/programs/Ex...

This is an example of adding multiple events of different type to the same element.

Click the following button to see the effect.

Click me

After clicking the button and leave the cursor, the output will be -



Example2.html

File | D:/CSS/programs/Ex...

This is an example of adding multiple events of different type to the same element.

Click the following button to see the effect.

Click me

This is second function

9.3.3 Event Bubbling or Event Capturing

Now, we understand the use of the third parameter of JavaScript's addEventListener(), i.e., ***useCapture.***

In HTML DOM, Bubbling and Capturing are the two ways of event propagation. We can understand these ways by taking an example.

Suppose we have a div element and a paragraph element inside it, and we are applying the **"click"** event to both of them using the **addEventListener()** method. Now the question is on clicking the paragraph element, which element's click event is handled first.

So, in Bubbling, the event of paragraph element is handled first, and then the div element's event is handled. It means that in bubbling, the inner element's event is handled first, and then the outermost element's event will be handled.

In Capturing the event of div element is handled first, and then the paragraph element's event is handled. It means that in capturing the outer element's event is handled first, and then the innermost element's event will be handled.

```
addEventListener(event, function, useCapture);
```

We can specify the propagation using the ***useCapture*** parameter. When it is set to false (which is its default value), then the event uses bubbling propagation, and when it is set to true, there is the capturing propagation.

We can understand the *bubbling* and *capturing* using an illustration.

Keyword

Event bubbling is a type of event propagation where the event first triggers on the innermost target element, and then successively triggers on the ancestors (parents) of the target element in the same nesting hierarchy till it reaches the outermost DOM element or document object.

Example

In this example, there are two div elements. We can see the bubbling effect on the first div element and the capturing effect on the second div element.

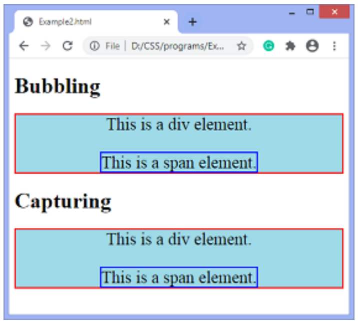
When we double click the span element of the first div element, then the span element's event is handled first than the div element. It is called *bubbling*.

But when we double click the span element of the second div element, then the div element's event is handled first than the span element. It is called *capturing*.

```
<!DOCTYPE html>
<html>
<head>
<style>
div{
background-color: lightblue;
border: 2px solid red;
font-size: 25px;
text-align: center;
}
span{
border: 2px solid blue;
}
</style>
</head>
<body>
<h1> Bubbling </h1>
<div id = “d1”>
This is a div element.
<br><br>
<span id = “s1”> This is a span element. </span>
</div>
<h1> Capturing </h1>
<div id = “d2”> This is a div element.
```

```
<br><br>
<span id = "s2"> This is a span element. </span>
</div>

<script>
document.getElementById("d1").addEventListener("dblclick", function(){alert('You have double clicked on div element')}, false);
document.getElementById("s1").addEventListener("dblclick", function(){alert('You have double clicked on span element')}, false);
document.getElementById("d2").addEventListener("dblclick", function(){alert('You have double clicked on div element')}, true);
document.getElementById("s2").addEventListener("dblclick", function(){alert('You have double clicked on span element')}, true);
</script>
</body>
</html>
```

Output



Bubbling

This is a div element.

This is a span element.

Capturing

This is a div element.

This is a span element.

We have to double click the specific elements to see the effect.

9.4 JAVASCRIPT ONCLICK EVENT

The onclick event generally occurs when the user clicks on an element. It allows the programmer to execute a JavaScript's function when an element gets clicked. This event can be used for validating a form, warning messages and many more.

Using JavaScript, this event can be dynamically added to any element. It supports all HTML elements except <html>, <head>, <title>, <style>, <script>, <base>, <iframe>, <bdo>, <br>, <meta>, and <param>. It means we cannot apply the onclick event on the given tags.

In HTML, we can use the onclick attribute and assign a JavaScript function to it. We can also use the JavaScript's addEventListener() method and pass a click event to it for greater flexibility.

9.4.1 Syntax

Now, we see the syntax of using the **onclick** event in HTML and in javascript (without **addEventListener()** method or by using the **addEventListener()** method).

9.4.2 In HTML

<element onclick = "fun()"**>**

9.4.3 In JavaScript

object.onclick = function() { myScript };

9.4.4 In JavaScript by using the addEventListener() method

object.addEventListener("click", myScript);

Let's see how to use **onclick** event by using some illustrations. Now, we will see the examples of using the **onclick** event in HTML, and in JavaScript.

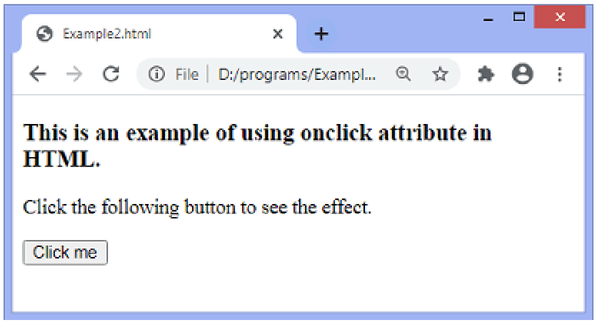
Example1 - Using onclick attribute in HTML

In this example, we are using the HTML **onclick** attribute and assigning a JavaScript's function to it. When the user clicks the given button, the corresponding function will get executed, and an alert dialog box will be displayed on the screen.

```
<!DOCTYPE html>
<html>
```

```
<head>
<script>
function fun() {
alert("Welcome to the javaTpoint.com");
}
</script>
</head>
<body>
<h3> This is an example of using onclick attribute in HTML. </h3>
<p> Click the following button to see the effect. </p>
<button onclick = "fun()">Click me</button>
</body>
</html>
```

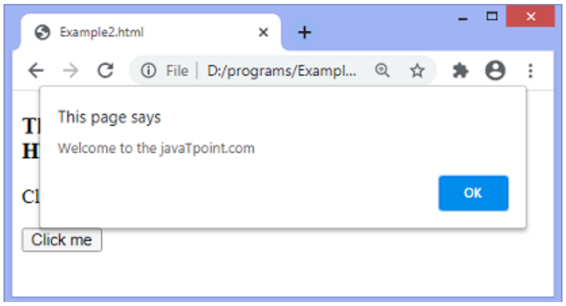
Output



This is an example of using onclick attribute in HTML.

Click the following button to see the effect.

After clicking the given button, the output will be -



This page says

Welcome to the javaTpoint.com

Example2 - Using JavaScript

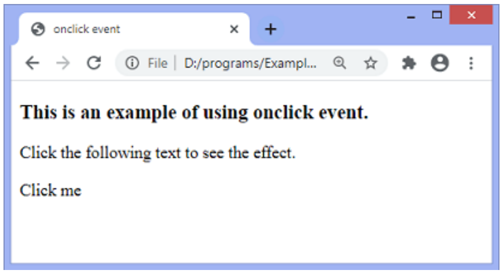
In this example, we are using JavaScript's **onclick** event. Here we are using the **onclick** event with the paragraph element.

When the user clicks on the paragraph element, the corresponding function will get executed, and the text of the paragraph gets changed. On clicking the **<p>** element, the background color, size, border, and color of the text will also get change.

```
<!DOCTYPE html>
<html>
<head>
<title> onclick event </title>
</head>
<body>
<h3> This is an example of using onclick event. </h3>
<p> Click the following text to see the effect. </p>
<p id = "para">Click me</p>
<script>
document.getElementById("para").onclick = function() {
fun()
};
function fun() {
```

```
document.getElementById("para").innerHTML = "Welcome to the javaTpoint.com";
document.getElementById("para").style.color = "blue";
document.getElementById("para").style.backgroundColor = "yellow";
document.getElementById("para").style.fontSize = "25px";
document.getElementById("para").style.border = "4px solid red";
}
</script>

</body>
</html>
```

Output

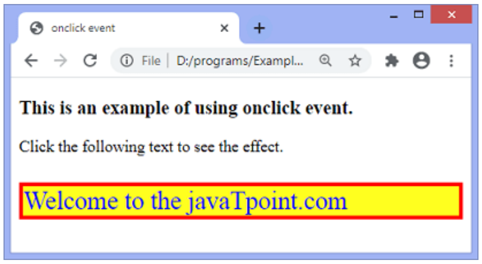


This is an example of using onclick event.

Click the following text to see the effect.

Click me

After clicking the text ***Click me,*** the output will be -



This is an example of using onclick event.

Click the following text to see the effect.

Welcome to the javaTpoint.com

Example3 - Using addEventListener() method

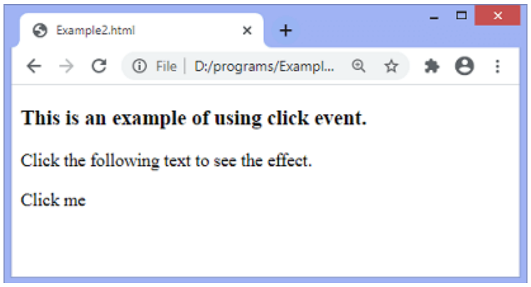
In this example, we are using JavaScript's **addEventListener()** method to attach a **click** event to the paragraph element. When the user clicks the paragraph element, the text of the paragraph gets changed.

On clicking the paragraph, the background color and font-size of elements will also change.

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<h3> This is an example of using click event. </h3>
<p> Click the following text to see the effect. </p>
<p id = "para">Click me</p>
<script>
document.getElementById("para").onclick = function() {
fun()
};
function fun() {
document.getElementById("para").innerHTML = "Welcome to the javaTpoint.
com";
document.getElementsByTagName("body")[0].style.color = "blue";
document.getElementsByTagName("body")[0].style.backgroundColor =
"lightgreen";
document.getElementsByTagName("body")[0].style.fontSize = "25px";
document.getElementById("para").style.border = "4px solid red";
}
```

```
20. </script>

21.

22. </body>

23. </html>
```

Output


Example2.html

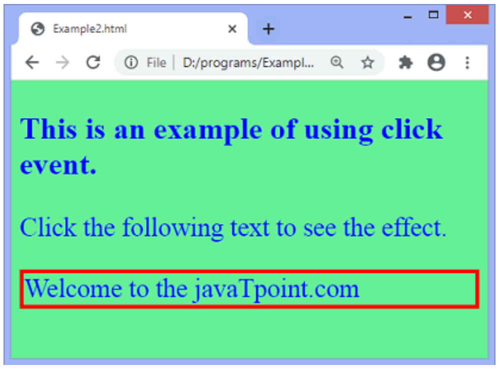
D:/programs/Exampl...

This is an example of using click event.

Click the following text to see the effect.

Click me

On clicking the text ***Click me***, the output will be -


Example2.html

D:/programs/Exampl...

This is an example of using click event.

Click the following text to see the effect.

Welcome to the javaTpoint.com

9.5 JAVASCRIPT DBLCLICK EVENT

The dblclick event generates an event on double click the element. The event fires when an element is clicked twice in a very short span of time. We can also use the JavaScript's addEventListener() method to fire the double click event.

In HTML, we can use the ondblclick attribute to create a double click event.

9.5.1 Syntax

Now, we see the syntax of creating double click event in HTML and in javascript (without using addEventListener() method or by using the addEventListener() method).

9.5.2 In HTML

```
<element ondblclick = "fun()">
```

9.5.3 In JavaScript

```
object.ondblclick = function() { myScript };
```

9.5.4 In JavaScript by using the addEventListener() method

```
object.addEventListener("dblclick", myScript);
```

Let's see some of the illustrations to understand the double click event.

Example - Using ondblclick attribute in HTML

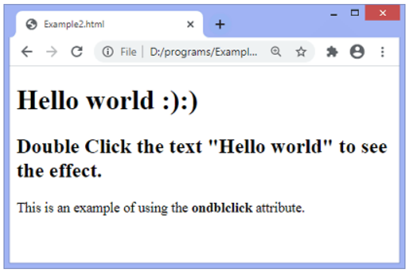
In this example, we are creating the double click event using the HTML **ondblclick** attribute.

```
<!DOCTYPE html>
<html>
<head>
</head>

<body>
<h1 id = "heading" ondblclick = "fun()"> Hello world :):) </h1>
<h2> Double Click the text "Hello world" to see the effect. </h2>
<p> This is an example of using the <b> ondblclick </b> attribute. </p>
<script>
```

11. function fun() {
12. document.getElementById("heading").innerHTML = " Welcome to the javaTpoint.com ";
13. }
14. **</script>**
15. **</body>**
16. **</html>**

Output

After the execution of the above code, the output will be -

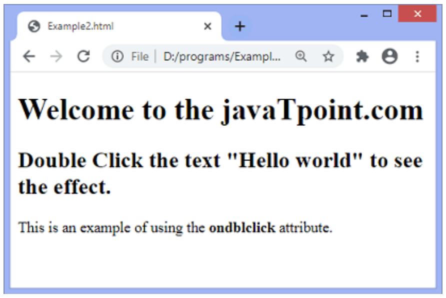


Hello world :):)

Double Click the text "Hello world" to see the effect.

This is an example of using the **ondblclick** attribute.

After double-clicking the text **"Hello world"**, the output will be -



Welcome to the javaTpoint.com

Double Click the text "Hello world" to see the effect.

This is an example of using the **ondblclick** attribute.

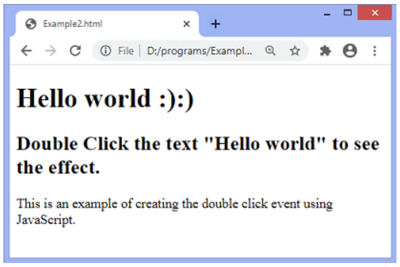
Now, we will see how to create double click event using JavaScript.

Example - Using JavaScript

1. <!DOCTYPE html>

```
<html>
<head>
</head>

<body>
<h1 id = "heading"> Hello world :):) </h1>
<h2> Double Click the text "Hello world" to see the effect. </h2>
<p> This is an example of creating the double click event using JavaScript. </p>
<script>
document.getElementById("heading").ondblclick = function() { fun() };
function fun() {
document.getElementById("heading").innerHTML = " Welcome to the javaTpoint.
com ";
}
</script>
</body>

</html>
```

Output

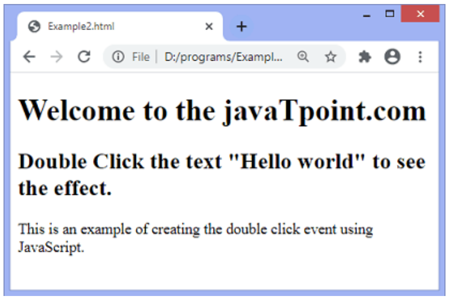


Hello world :):)

Double Click the text "Hello world" to see the effect.

This is an example of creating the double click event using JavaScript.

After double-clicking the text **“Hello world”**, the output will be -



Welcome to the javaTpoint.com

Double Click the text "Hello world" to see the effect.

This is an example of creating the double click event using JavaScript.

Example - Using JavaScript’s addEventListener() method

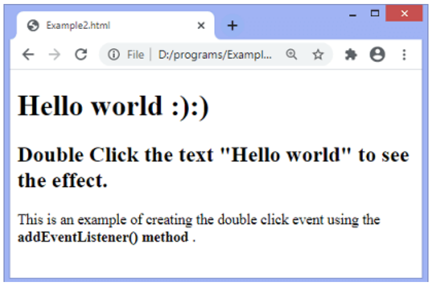
```
<!DOCTYPE html>
<html>
<head>
</head>

<body>
<h1 id = “heading”> Hello world :):) </h1>
<h2> Double Click the text “Hello world” to see the effect. </h2>
<p> This is an example of creating the double click event using the <b> addEventListener() method </b>. </p>
<script>
document.getElementById(“heading”).addEventListener(“dblclick”, fun);
function fun() {
document.getElementById(“heading”).innerHTML = “ Welcome to the javaTpoint.com “;
}
</script>
```

16. </body>

17.

18. </html>

Output



Example2.html

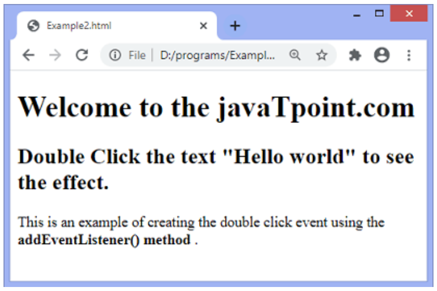
File | D:/programs/Exampl...

Hello world :):)

Double Click the text "Hello world" to see the effect.

This is an example of creating the double click event using the **addEventListener() method** .

After double-clicking the text **"Hello world"**, the output will be -



Example2.html

File | D:/programs/Exampl...

Welcome to the javaTpoint.com

Double Click the text "Hello world" to see the effect.

This is an example of creating the double click event using the **addEventListener() method** .

9.6 JAVASCRIPT ONLOAD

In JavaScript, this event can apply to launch a particular function when the page is fully displayed. It can also be used to verify the type and version of the visitor's browser. We can check what cookies a page uses by using the onload attribute.

In HTML, the onload attribute fires when an object has been loaded. The purpose of this attribute is to execute a script when the associated element loads.

In HTML, the onload attribute is generally used with the **<body>** element to execute a script once the content (including CSS files, images, scripts, etc.) of the webpage is completely loaded. It is not necessary to use it only with <body> tag, as it can be used with other HTML elements.

The difference between the **document.onload** and **window.onload** is: **document.onload** triggers before the loading of images and other external content. It is fired before the **window.onload**. While the **window.onload** triggers when the entire page loads, including CSS files, script files, images, etc.

9.6.1 Syntax

window.onload = fun()

Let's understand this event by using some examples.

Remember

onload attribute is not necessary to use it only with <body> tag, as it can be used with other HTML elements.

Example1

In this example, there is a div element with a height of 200px and a width of 200px. Here, we are using the **window.onload()** to change the background color, width, and height of the **div** element after loading the web page.

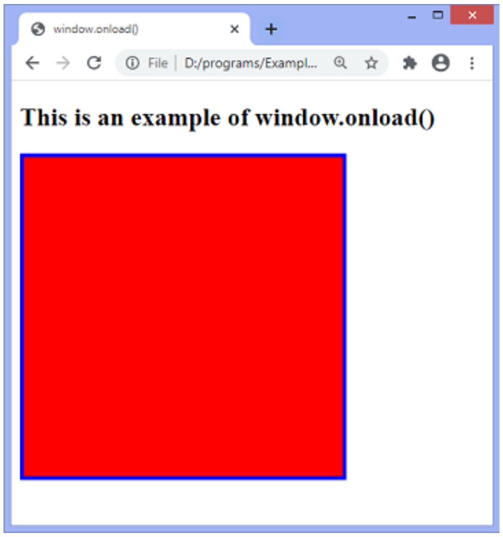
The background color is set to **'red'**, and width and height are set to **300px** each.

```
<!DOCTYPE html>
<html>
<head>
<meta charset = " utf-8">
<title> window.onload() </title>
<style type = "text/css">
#bg{
width: 200px;
height: 200px;
border: 4px solid blue;
}
</style>
<script type = "text/javascript">
```

```
window.onload = function(){
document.getElementById("bg").style.backgroundColor = "red";
document.getElementById("bg").style.width = "300px";
document.getElementById("bg").style.height = "300px";
}
</script>
</head>
<body>
<h2> This is an example of window.onload() </h2>
<div id = "bg"></div>
</body>
</html>
```

Output

After the execution of the code and loading of the page, the output will be -



```
This is an example of window.onload()
```

Example2

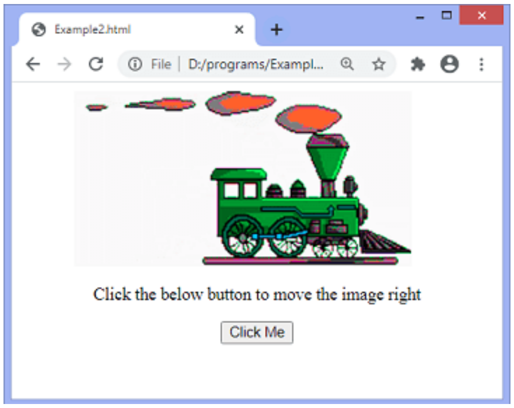
In this example, we are implementing a simple animation by using the properties of the DOM object and functions of javascript. We use the JavaScript function getElementById() for getting the DOM object and then assign that object into a global variable.

```
<html>
  <head>
    <script type = “text/javascript”>

        var img = null;
        function init(){
          img = document.getElementById(‘myimg’);
          img.style.position = ‘relative’;
          img.style.left = ‘50px’;
        }
        function moveRight(){
          img.style.left = parseInt(
          img.style.left) + 100 + ‘px’;
        }
        window.onload = init;

    </script>
  </head>

  <body>
    <form>
```

```
        <img id = “myimg” src = “train1.png” />
        <center>
          <p>Click the below button to move the image right</p>
        <input type = “button” value = “Click Me” onclick = “moveRight();” />
      </center>
      </form>
    </body>

</html>
```

Output

After the successful execution of the above code, the output will be -



Click the below button to move the image right

Now, there is an example in which we will use the HTML **onload** attribute and the JavaScript functions.

Example3

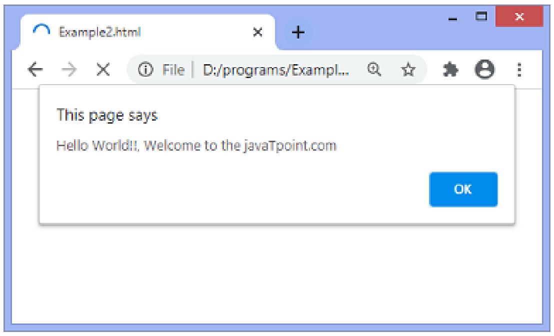
It is a simple example of using the HTML **onload** attribute with the function defined in JavaScript. In this example, the **alert()** function gets called whenever the document refresh.

```
<!DOCTYPE html>
```

```
<html>
<head>
<style>
</style>
<script>
  function fun() {
    alert("Hello World!!, Welcome to the javaTpoint.com");
  }
</script>
</head>
<body onload = "fun()">
<h1> Example of the HTML onload attribute </h1>
<p> Try to refresh the document to see the effect. </p>
</body>
</html>
```

Output

After the execution of the above code, the output will be -



This page says

Hello World!!, Welcome to the javaTpoint.com

9.7 JAVASCRIPT ONRESIZE EVENT

The onresize event in JavaScript generally occurs when the window has been resized. To get the size of the window, we can use the JavaScript's *window.outerWidth* and *window.outerHeight* events. We can also use the JavaScript's properties such as *innerWidth, innerHeight, clientWidth, ClientHeight, offsetWidth, offsetHeight* to get the size of an element.

In HTML, we can use the onresize attribute and assign a JavaScript function to it. We can also use the JavaScript's addEventListener() method and pass a resize event to it for greater flexibility.

9.7.1 Syntax

Now, we see the syntax of using the **onresize** event in HTML and in javascript (without **addEventListener()** method or by using the **addEventListener()** method).

9.7.2 In HTML

```
<element onresize = "fun()">
```

9.7.3 In JavaScript

```
object.onresize = function() { myScript };
```

9.7.4 In JavaScript by using the addEventListener() method

```
object.addEventListener("resize", myScript);
```

Let's see some of the illustrations to understand the **onresize** event.

Example

In this example, we are using the HTML **onresize** attribute. Here, we are using the ***window.outerWidth*** and ***window.outerHeight*** events of JavaScript to get the height and width of the window.

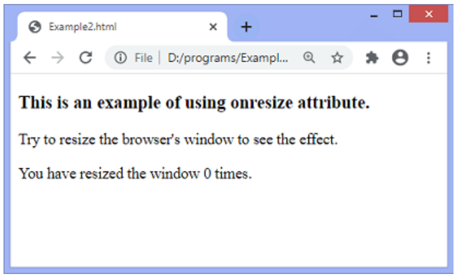
When the user resizes the window, the updated width and height of the window will be displayed on the screen. It will also display how many times the user tried to resize the window. When we change the height of the window, the updated height will change accordingly. Similarly, when we change the width of the window, the updated width will change accordingly.

```
<!DOCTYPE html>
```

```
<html>
<head>
<script>
var i = 0;

function fun() {
var res = “Width = “ + window.outerWidth + “<br>” + “Height = “ + window.outerHeight;
document.getElementById(“para”).innerHTML = res;

var res1 = i += 1;
document.getElementById(“s1”).innerHTML = res1;
}
</script>
</head>
<body onresize = “fun()”>
<h3> This is an example of using onresize attribute. </h3>
<p> Try to resize the browser’s window to see the effect. </p>

<p id = “para”> </p>
<p> You have resized the window <span id = “s1”> 0 </span> times.</p>
</body>
</html>
```

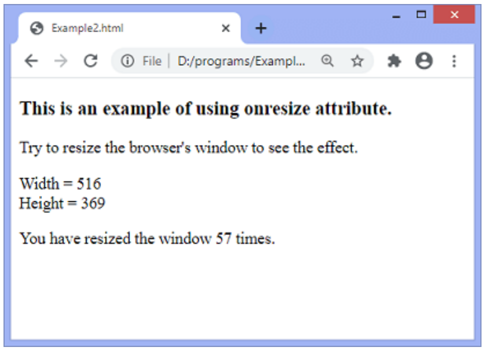
Output

After the execution of the above code, the output will be -



```
This is an example of using onresize attribute.

Try to resize the browser's window to see the effect.

You have resized the window 0 times.
```

When we try to resize the window, the output will be -



```
This is an example of using onresize attribute.

Try to resize the browser's window to see the effect.

Width = 516
Height = 369

You have resized the window 57 times.
```

Example - Using JavaScript

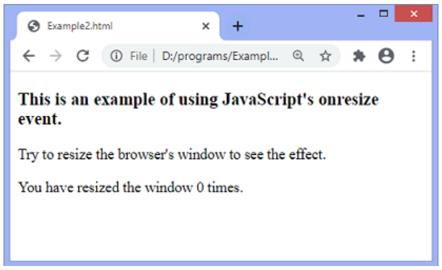
In this example, we are using JavaScript's **onresize** event.

1. <!DOCTYPE html>
2. **<html>**
3. **<head>**
4. **</head>**
5. **<body>**
6. **<h3>** This is an example of using JavaScript's onresize event. **</h3>**
7. **<p>** Try to resize the browser's window to see the effect. **</p>**

```

<p id = “para”> </p>
<p> You have resized the window <span id = “s1”> 0 </span> times.</p>
<script>
document.getElementsByTagName(“BODY”)[0].onresize = function() {fun()};
var i = 0;

function fun() {
var res = “Width = “ + window.outerWidth + “<br>” + “Height = “ + window.
outerHeight;
document.getElementById(“para”).innerHTML = res;

var res1 = i += 1;
document.getElementById(“s1”).innerHTML = res1;
}
</script>
</body>
</html>
```

Output

After the execution of the above code, the output will be -

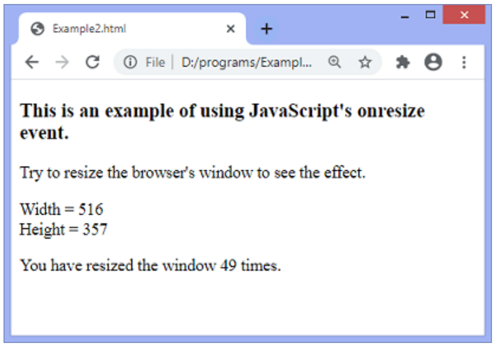


This is an example of using JavaScript's onresize event.

Try to resize the browser's window to see the effect.

You have resized the window 0 times.

When we try to resize the window, the output will be -



This is an example of using JavaScript's onresize event.

Try to resize the browser's window to see the effect.

```
Width = 516
Height = 357
```

You have resized the window 49 times.

Example - Using addEventListener() method

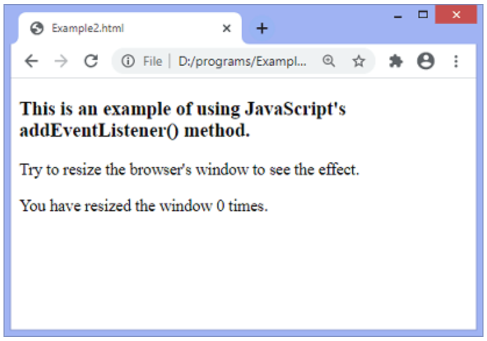
In this example, we are using JavaScript's **addEventListener()** method.

1. <!DOCTYPE html>
2. **<html>**
3. **<head>**
4. **</head>**
5. **<body>**
6. **<h3>** This is an example of using JavaScript's addEventListener() method. **</h3>**
7. **<p>** Try to resize the browser's window to see the effect. **</p>**
8.
9. **<p** id = "para"**> </p>**
10. **<p>** You have resized the window **<span** id = "s1"**>** 0 **</span>** times.**</p>**

```
<script>
window.addEventListener("resize", fun);
var i = 0;

function fun() {
var res = "Width = " + window.outerWidth + "<br>" + "Height = " + window.outerHeight;
document.getElementById("para").innerHTML = res;

var res1 = i += 1;
document.getElementById("s1").innerHTML = res1;
}
</script>
</body>
</html>
```

Output

After the execution of the above code, the output will be -

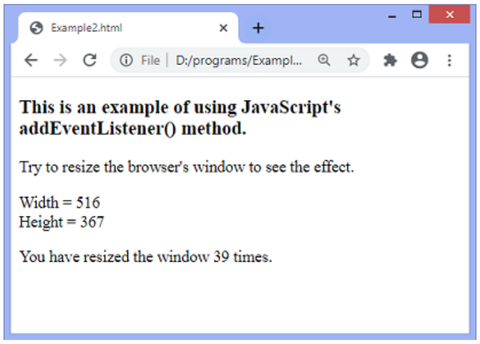


This is an example of using JavaScript's addEventListener() method.

Try to resize the browser's window to see the effect.

You have resized the window 0 times.

When we try to resize the window, the output will be -



This is an example of using JavaScript's addEventListener() method.

Try to resize the browser's window to see the effect.

Width = 516
Height = 367

You have resized the window 39 times.

ROLE MODEL

DOUGLAS CROCKFORD

Douglas Crockford is an American computer programmer and entrepreneur who is involved in the development of the JavaScript language. He popularized the data format JSON (JavaScript Object Notation), and has developed various JavaScript related tools such as JSLint and JSMin. He was a senior JavaScript architect at PayPal until 2019, and is also a writer and speaker on JavaScript, JSON, and related web technologies.

Education

Crockford earned a degree in Radio and Television from San Francisco State University in 1975. He took classes in FORTRAN and worked with a university lab's computer.

Career

Crockford purchased an Atari 8-bit computer in 1980 and wrote the game *Galahad and the Holy Grail* for the Atari Program Exchange (APX), which resulted in Chris Crawford hiring him at Atari, Inc. While at Atari, Crockford wrote another game, *Burgers!*, for APX and a number of experimental audio/visual demos that were freely distributed.

After Warner Communications sold the company, he joined National Semiconductor. In 1984 Crockford joined Lucasfilm, and later Paramount Pictures. He became known on video game oriented listservs in the early 1990s after he posted his memoir "The Expurgation of Maniac Mansion" to a videogaming bulletin board. The memoir documented his efforts to censor the computer game *Maniac Mansion* to Nintendo's satisfaction so that they could release it as a cartridge, and Crockford's mounting frustrations as Nintendo's demands became more obscure and confusing.

Together with Randy Farmer and Chip Morningstar, Crockford founded Electric Communities and was its CEO from 1994 to 1995. He was involved in the development of the programming language E.

Crockford was the founder of State Software (also known as Veil Networks) and its CTO from 2001 to 2002.

During his time at State Software, Crockford popularized the JSON data format, based upon existing JavaScript language constructs, as a lightweight alternative to XML. He obtained the domain name json.org in 2002, and put up his description of the format there. In July 2006, he specified the format officially, as RFC 4627.

"Good, not Evil"

In 2002, in reference to President George Bush's war on "evildoers", Crockford started releasing his JSMin software under a custom license, which he created by adding the requirement "The Software shall be used for Good, not Evil" to the open source MIT License. This clause was carried over to JSMin-PHP, a variation of JSMin by Ryan Grove. This software was hosted on Google Code until December 2009 when, due to the additional clause, Google determined that the license was not compliant with the definition of free and open source software, which does not permit any restriction on how software may be used. JSMin-PHP was forced to migrate to a new hosting provider.

Crockford's license has caused problems for some open source projects who mistook the license for an open source variant of the MIT license. Affected open source developers have asked Crockford to change the license, but he has continued to use it. He has, however, granted "IBM, its customers, partners, and minions" permission "to use JSLint for evil", a solution which appeared to satisfy IBM's lawyers.

In Media

Books

- Crockford is listed in the acknowledgements of the 1995 hardcover edition of *The Diamond Age*, by Neal Stephenson as *Douglas (Carl Hollywood) Crockford*.

SUMMARY

- The Javascript interacts with the documents HTML code using *events*, which are triggered when a particular moment of interest happens in the document or the browser window.
- Javascript Events makes the webpages interactive and responsive. These events are *asynchronous* (i.e it can occur anytime).Most events are triggered by user action but there are some exceptions like the event *load*
- When an event occurs there are some default action which the browser takes(like clicking on a link open up the location specified in attribute *href*). An event may be a click, mouseover, keystroke etc.
- The change in the state of an object is known as an Event. In html, there are various events which represents that some activity is performed by the user or by the browser. When javascript code is included in HTML, js react over these events and allow the execution. This process of reacting over the events is called Event Handling. Thus, js handles the HTML events via Event Handlers.
- The addEventListener() method is used to attach an event handler to a particular element. It does not override the existing event handlers. Events are said to be an essential part of the JavaScript. A web page responds according to the event that occurred. Events can be user-generated or generated by API's. An event listener is a JavaScript's procedure that waits for the occurrence of an event.
- The onclick event generally occurs when the user clicks on an element. It allows the programmer to execute a JavaScript's function when an element gets clicked. This event can be used for validating a form, warning messages and many more.
- The dblclick event generates an event on double click the element. The event fires when an element is clicked twice in a very short span of time. We can also use the JavaScript's addEventListener() method to fire the double click event.
- The onload attribute fires when an object has been loaded. The purpose of this attribute is to execute a script when the associated element loads.
- The onload attribute is generally used with the <body> element to execute a script once the content (including CSS files, images, scripts, etc.) of the webpage is completely loaded. It is not necessary to use it only with <body> tag, as it can be used with other HTML elements.
- The onresize event in JavaScript generally occurs when the window has been resized. To get the size of the window, we can use the JavaScript's *window.outerWidth* and *window.outerHeight* events. We can also use the JavaScript's properties such as *innerWidth, innerHeight, clientWidth, ClientHeight, offsetWidth, offsetHeight* to get the size of an element.

KNOWLEDGE CHECK

1. **In general, event handler is nothing but ____________**
   a. function
   b. interface
   c. event
   d. handler
2. **When will the browser invoke the handler?**
   a. Program begins
   b. Any event occurs
   c. Specified event occurs
   d. Webpage loads
3. **The process by which the browser decides which objects to trigger event handlers on is ____________**
   a. Event Triggering
   b. Event Listening
   c. Event Handling
   d. Event propagation
4. **Which form of event propagation handles the registered container elements?**
   a. Event Propagation
   b. Event Registration
   c. Event Capturing
   d. Default Actions
5. **In which events/scenarios, A function name gets optional in JavaScript?**
   a. When a function is defined as a looping statement
   b. When the function is called
   c. When a function is defined as expressions
   d. When the function is predefined

REVIEW QUESTIONS

1. Describe the types of JavaScript event.
2. What does addEventListener do in JavaScript?
3. What is onclick event in JavaScript?
4. What is dblclick in JavaScript?
5. Distinguish between event bubbling and event capturing.

Check Your Result

1. (a)	2. (c)	3. (d)	4. (c)	5. (c)

REFERENCES

1. Fedortsova, Irina (June 2012). “Concurrency in JavaFX”. JavaFX Documentation Home. Oracle. Retrieved 4 January 2018.
2. Mössenböck, Hanspeter (2002-03-25). “Advanced C#: Variable Number of Parameters” (PDF). Institut für Systemsoftware, Johannes Kepler Universität Linz, Fachbereich Informatik. p. 26. Retrieved 2011-08-05.
3. Samek, Miro (11 March 2009). “State Machines for Event-Driven Systems”. Retrieved 19 March 2013.
4. Vivek Gupta, Ethan Jackson, Shaz Qadeer and Sriram Rajamani (November 2012). “P: Safe Asynchronous Event-Driven Programming”. Microsoft Research. Retrieved 20 February 2017.

Index

Level: Beginner to Advanced
Subject: Computer and Information Science

Basic Computer Coding: Java Script

2nd Edition

JavaScript is a programming language that is primarily used by Web browsers to provide users with a dynamic and interactive experience. The majority of the functions and applications that make the Internet indispensable in modern life are written in JavaScript. While JavaScript is not the only client-side scripting language available on the Internet, it was one of the first and remains the most popular. Enterprising programmers have created JavaScript libraries, which are more concise languages built from JavaScript building blocks that are less complex and can be targeted for specific applications. JavaScript has become integral to the Internet experience as developers build increased interaction and complexity into their applications. Search engines, ecommerce, content management systems, responsive design, social media and phone apps would not be possible without it.

This edition contains nine chapters. Information is completely revised and new chapters are added. This book provides clear guidance on how to use approaches to writing JavaScript. A guide for beginners offers an overview of JavaScript basics and explains how to create Web pages, identify browsers, and integrate sound, graphics, and animation into Web applications.

ISBN 978-1-98467-603-0
00000
9 781984 676030

BIBLIOTEX
Digital Library

www.3ge-learning.com
email: info@3ge-learning.com