पायथन प्रोग्रामिंग

अभय मेहरोत्रा

भाषा प्रकाशन

पायथन प्रोग्रामिंग

पायथन प्रोग्रामिंग

अभय मेहरोत्रा

भाषा प्रकाशन
नई दिल्ली – 110002

I.S.B.N. : 978-81-323-6944-8

प्रथम संस्करण : 2022

भाषा प्रकाशन

22, प्रकाशदीप बिल्डिंग, अंसारी रोड,
दरियागंज, नई दिल्ली – 110002

द्वारा वर्ल्ड टेक्नोलॉजीज नई दिल्ली के सहयोग से प्रकाशित

अनुक्रम

1

पायथन प्रोग्रामिंग लैंग्वेज का परिचय

पायथन एक प्रोग्रामिंग लैंग्वेज है जिसका उपयोग ग्राफिक यूजर इंटरफेस अनुप्रयोगों, वेबसाइटों और वेब अनुप्रयोगों को विकसित करने के लिए किया जाता है। पायथन लैंग्वेज के विभिन्न संस्करण पायथन 1.0, पायथन 2.0 और पायथन 3.0 हैं। यह एक परिचयात्मक अध्याय है जो इन पायथन प्रोग्रामिंग लैंग्वेज की इंस्टॉलेशन प्रोसेस का संक्षेप में परिचय देता है।

इसमें पायथन की एक व्याख्या की गई है और यह उच्च-स्तरीय, सामान्य-उद्देश्य वाली प्रोग्रामिंग लैंग्वेज है। इसे गुइडो वैन रोसुम द्वारा बनाया गया और पहली बार 1991 में जारी किया गया था, पायथन का डिजाइन दर्शन महत्वपूर्ण व्हाइटस्पेस के उल्लेखनीय उपयोग के साथ कोड रीडेबिलिटी पर जोर देता है। इसकी लेंग्वेज निर्माण और वस्तु-उन्मुख दृष्टिकोण का उद्देश्य छोटे और बड़े पैमाने की परियोजनाओं के लिए प्रोग्रामर को स्पष्ट, तार्किक कोड लिखने में मदद करना है।

पायथन गतिशील रूप से टाइप किया गया है और इसका इस्तेमाल गार्बेज- कलेक्ट करने के लिए किया जाता है। यह प्रक्रियात्मक, वस्तु-उन्मुख और कार्यात्मक प्रोग्रामिंग सहित कई प्रोग्रामिंग प्रतिमानों का समर्थन करता है। अपने व्यापक मानक लाइब्रेरी के कारण पायथन को अक्सर "बैटरी इनक्लूड" लैंग्वेज के रूप में वर्णित किया जाता है।

सबसे पहले पायथन की कल्पना 1980 के दशक के अंत में एबीसी लैंग्वेज के उत्तराधिकारी के रूप में की गई थी। 2000 में जारी पायथन 2.0 ने सूची की समझ और संदर्भ चक्र एकत्र करने में सक्षम गार्बेज संग्रह प्रणाली जैसी सुविधाएं पेश की थी। इसके बाद 2008 में जारी किया गया पायथन 3.0, उस लैंग्वेज का एक प्रमुख संशोधन था जो पूरी तरह से अनुकूल नहीं है और बहुत से पायथन 2 कोड पायथन 3 पर अनमॉडिफाइड नहीं चल पाते हैं।

पायथन 2 लैंग्वेज, यानी पायथन 2.7.x, 1 जनवरी, 2020 को "सनसेटिंग" है (विस्तार के बाद; पहली बार 2015 के लिए योजना बनाई गई), और स्वयंसेवकों की पायथन टीम सुरक्षा मुद्दों को ठीक नहीं करेगी या उसके बाद इसे अन्य तरीकों से उसमें सुधार नहीं करेगी। इसके बाद एंड-ऑफ़-लाइफ़ के साथ केवल पायथन 3.5.x और उसके बाद के संस्करण का समर्थन किया जाएगा।

कई ऑपरेटिंग सिस्टम के लिए पायथन इंटरप्रेटर उपलब्ध होते हैं। प्रोग्रामर्स का एक वैश्विक समुदाय सी-पायथन का विकास और रखरखाव करता है, जो एक खुला स्रोत संदर्भ कार्यान्वयन है। इसके लिए एक गैर-लाभकारी संगठन, पायथन सॉफ्टवेयर फाउंडेशन, पायथन और सीपीथन विकास के लिए संसाधनों का प्रबंधन और निर्देशन करता है।

पायथन एक बहु-प्रतिमान प्रोग्रामिंग लैंग्वेज है। ऑब्जेक्ट-ओरिएंटेड प्रोग्रामिंग और स्ट्रक्चर्ड प्रोग्रामिंग पूरी तरह से समर्थित हैं और इसकी कई विशेषताएं कार्यात्मक प्रोग्रामिंग और पहलू-उन्मुख प्रोग्रामिंग (मेटाप्रोग्रामिंग और मेटाऑब्जेक्ट्स (मैजिक मेथड्स) सहित) का समर्थन करती हैं। कई अन्य प्रतिमान एक्सटेंशन के माध्यम से इसमें समर्थित होते हैं, जिसमें अनुबंध द्वारा डिजाइन और तर्क प्रोग्रामिंग शामिल हैं।

पायथन गतिशील टाइपिंग और संदर्भ गणना के संयोजन और मेमोरी प्रबंधन के लिए एक चक्र-डिटेक्टिंग गार्बेज संग्रहकर्ता का उपयोग करता है। इसमें गतिशील नाम संकल्प (देर से बाध्यकारी) भी शामिल है, जो प्रोग्राम निष्पादन के दौरान विधि और वेरिएबल नामों को बांधता है।

पायथन का डिज़ाइन लिस्प परंपरा में कार्यात्मक प्रोग्रामिंग के लिए कुछ समर्थन प्रदान करता है। इसमें फिल्टर, मैप और रिड्यूस फंक्शन शामिल हैं; इसकी सूची समझ, शब्दकोश, सेट, और जनरेटर अभिव्यक्तियाँ। मानक लाइब्रेरी में दो मॉड्यूल (इटरटूल्स और फंकटूल्स) हैं जो हास्केल और मानक एमएल से उधार लिए गए कार्यात्मक उपकरणों को लागू करते हैं।

इसके डाक्यूमेंट्स में लैंग्वेज के मूल दर्शन को संक्षेप में प्रस्तुत किया गया है। पायथन का ज़ेन (पीईपी 20), जिसमें सूत्र शामिल हैं जैसे:

- सुंदर बदसूरत से बेहतर है।
- स्पष्ट निहित से बेहतर है.
- सरल जटिल से बेहतर है।
- जटिल जटिलता से बेहतर है।
- इसमें पठनीयता मायने रखती है।

इसकी सभी कार्यक्षमता को इसके मूल में निर्मित करने के बजाय, पायथन को अत्यधिक एक्स्टेंसिबल होने के लिए डिज़ाइन किया गया था। इस कॉम्पैक्ट मॉड्यूलरिटी ने इसे मौजूदा अनुप्रयोगों में प्रोग्राम करने योग्य इंटरफेस जोड़ने के साधन के रूप में विशेष रूप से लोकप्रिय बना दिया है। एक बड़े मानक लाइब्रेरी और आसानी से एक्स्टेंसिबल इंटरप्रेटर के साथ एक छोटी कोर लैंग्वेज की वैन रॉसम की दृष्टि एबीसी के साथ उनकी निराशा से उत्पन्न हुई थी, जिसने विपरीत दृष्टिकोण को स्वीकार किया था।

पायथन डेवलपर्स को उनकी कोडिंग पद्धति में एक विकल्प देते हुए एक सरल, कम-अव्यवस्थित वाक्यविन्यास और व्याकरण के लिए प्रयास करता है। पर्ल के "इसे करने के एक से अधिक तरीके हैं" आदर्श वाक्य के विपरीत, पायथन एक "एक होना चाहिए और अधिमानतः केवल एक - इसे करने का स्पष्ट तरीका" डिजाइन फिलोसोफी होती है। पायथन सॉफ्टवेयर फाउंडेशन के फेलो और पायथन पुस्तक लेखक एलेक्स मार्टेली लिखते हैं कि "किसी चीज को 'चतुर' के रूप में वर्णित करना पायथन संस्कृति में प्रशंसा नहीं माना जा सकता है।"

पायथन के डेवलपर्स समय से पहले अनुकूलन से बचने का प्रयास करते हैं और सीपायथन संदर्भ कार्यान्वयन के गैर-महत्वपूर्ण भागों में पैच को अस्वीकार करते हैं जो स्पष्टता की कीमत पर गति में मामूली वृद्धि की पेशकश करता है। जब इसकी गति महत्वपूर्ण होती है, तो एक पायथन प्रोग्रामर समय-महत्वपूर्ण कार्यों को सी जैसी लैंग्वेजों में लिखे गए विस्तार मॉड्यूल में स्थानांतरित कर सकता है, या पाय-पाय का उपयोग करता है, जो समय-समय पर संकलक होता है। इसके लिए साइथन भी उपलब्ध है, जो एक पायथन लिपि का सी में अनुवाद करता है और सीधे सी-स्तरीय एपीआई कॉल को पायथन इंटरप्रेटर में करता है।

पायथन के डेवलपर्स का एक महत्वपूर्ण लक्ष्य इसका उपयोग करने में रूचि को बनाए रखना है। यह लैंग्वेज के नाम-ब्रिटिश कॉमेडी समूह मोंटी पायथन के लिए एक श्रद्धांजलि या ट्यूटोरियल और संदर्भ सामग्री के लिए कभी-कभी चंचल दृष्टिकोण में परिलक्षित होता है, जैसे उदाहरण जो स्पैम और एग्स को संदर्भित करते हैं (एक प्रसिद्ध मोंटी पायथन स्केच से) ऐसा मानक फू और बार के बजाय इसका इस्तेमाल किया जाता है।

पायथन समुदाय में एक सामान्य नवविज्ञान पाइथोनिक है, जिसमें कार्यक्रम शैली से संबंधित अर्थों की एक विस्तृत श्रृंखला हो सकती है। कहा जाता है कि कोड पाइथोनिक होता है और इसके बारे में यह कहना है कि यह पायथन मुहावरों का अच्छी तरह से उपयोग करता है, यह स्वाभाविक है और लैंग्वेज में प्रवाह दिखाता है, कि यह पायथन के न्यूनतम दर्शन और पठनीयता पर जोर देता है। इसके विपरीत, जिस कोड को समझना मुश्किल होता है या किसी अन्य प्रोग्रामिंग लैंग्वेज से किसी न किसी ट्रांसक्रिप्शन की तरह पढ़ता है उसे अनपाइथोनिक कहा जाता है।

उपयोगकर्ताओं और पायथन के प्रशंसक, विशेष रूप से जिन्हें जानकार या अनुभवी माना जाता है, उन्हें अक्सर पाइथोनिस्टस कहा जाता है।

सिंटैक्स एंड सिमेंटिक्स

पायथन को आसानी से पढ़ने योग्य भाषा माना जाता है। इसका स्वरूप स्पष्ट रूप से अव्यवस्थित होता है और यह अक्सर अंग्रेजी कीवर्ड का उपयोग करता है जहां अन्य भाषाएं विराम चिह्न का उपयोग करती हैं। कई अन्य भाषाओं के विपरीत, यह ब्लॉकों को परिसीमित करने के लिए घुंघराले कोष्ठक का उपयोग नहीं करता है और स्टेटमेंट के बाद अर्धविराम वैकल्पिक होते हैं। इसमें सी या पास्कल की तुलना में कम वाक्यात्मक अपवाद और विशेष मामले होते हैं।

इंडेंटेशन

पाइथन ब्लॉक को परिसीमित करने के लिए कर्ली ब्रैकेट या कीवर्ड के बजाय व्हाइटस्पेस इंडेंटेशन का उपयोग करता है। कुछ कथनों के बाद इंडेंटेशन में वृद्धि होती है; इंडेंटेशन में कमी वर्तमान ब्लॉक के अंत का संकेत देती है। इस प्रकार कार्यक्रम की दृश्य संरचना सटीक रूप से कार्यक्रम की शब्दार्थ संरचना का प्रतिनिधित्व करती है। इस सुविधा को कभी-कभी ऑफ-साइड नियम भी कहा जाता है, जिसे कुछ अन्य लैंग्वेजों में साझा किया जाता है, लेकिन अधिकांश लैंग्वेजों में इंडेंटेशन का कोई अर्थ अर्थ नहीं होता है।

स्टेटमेंट्स और कंट्रोल फ्लो

पायथन में निम्न स्टेटमेंट्स शामिल हैं:

- असाइनमेंट स्टेटमेंट (टोकन '=', चिह्न के बराबर)। यह पारंपरिक अनिवार्य प्रोग्रामिंग लैंग्वेजों की तुलना में अलग तरह से संचालित होता है और यह मौलिक तंत्र (पायथन के वेरिएबल के संस्करण की प्रकृति सहित) लैंग्वेज की कई अन्य विशेषताओं को प्रकाशित करता है। सी में असाइनमेंट, उदाहरण के लिए, x = 2, "टाइप किए गए वेरिएबल नाम x को संख्यात्मक मान 2 की एक प्रति प्राप्त करता है" में अनुवाद करता है। (दाएं हाथ) मान को एक आवंटित भंडारण स्थान में कॉपी किया जाता है जिसके लिए (बाएं हाथ) वेरिएबल नाम प्रतीकात्मक पता होता है। घोषित प्रकार के लिए वेरिएबल को आवंटित मेमोरी काफी बड़ी (संभावित रूप से काफी बड़ी) है। पायथन असाइनमेंट के सबसे सरल मामले में, एक ही उदाहरण का उपयोग करते हुए, x = 2, "(जेनेरिक) नाम x में अनुवाद करता है, संख्यात्मक (int) प्रकार के मान 2 के एक अलग, गतिशील रूप से आवंटित ऑब्जेक्ट का संदर्भ प्राप्त करता है। इस नाम को वस्तु से बांधना कहा जाता है। चूंकि नाम के भंडारण स्थान में संकेतित मान नहीं है, इसलिए इसे एक वेरिएबल कहना अनुचित है। इसके नाम बाद में किसी भी समय बहुत भिन्न प्रकार की वस्तुओं के लिए रिबाउंड हो सकते हैं, जिसमें स्ट्रिंग्स, प्रक्रियाएं, डेटा और विधियों के साथ जटिल ऑब्जेक्ट आदि शामिल होते हैं। एक से अधिक नामों के लिए एक सामान्य मूल्य के क्रमिक असाइनमेंट, जैसे, x = 2; y = 2; z = 2 का परिणाम (अधिकतम) तीन नामों और एक संख्यात्मक वस्तु को भंडारण आवंटित करने में होता है, जिससे सभी तीन नाम बंधे होते हैं। चूंकि एक नाम एक सामान्य संदर्भ धारक होता है इसलिए इसके साथ एक निश्चित डेटा प्रकार को जोड़ना अनुचित है। हालांकि एक निश्चित समय पर एक नाम किसी वस्तु के लिए बाध्य होता है, यह इसका ही एक स्वरूप होता है ; इस प्रकार गतिशील टाइपिंग होती है।

- `i f` स्टेटमेंट, जो सशर्त रूप से कोड के एक ब्लॉक को निष्पादित करता है, साथ में `else` और `elif` (अन्य-अगर का संकुचन)।

  (`else-if` का संकुचन)।

- `for` के लिय स्टेटमेंट, जो एक पुनरावृत्त वस्तु पर पुनरावृति करता है, संलग्न ब्लॉक द्वारा उपयोग के लिए प्रत्येक तत्व को स्थानीय वेरिएबल में कैप्चर करता है।

- `while` स्टेटमेंट, जो कोड के एक ब्लॉक को तब तक निष्पादित करता है जब तक कि उसकी स्थिति सत्य है।

- `try` स्टेटमेंट, जो इसके संलग्न कोड ब्लॉक में उठाए गए अपवादों को पकड़ने और संभालने की अनुमति देता है के अलावा खंड; यह भी सुनिश्चित करता है कि सफाई कोड a आखिरकार ब्लॉक विल ब्लॉक कैसे निकलता है इस पर ध्यान दिए बिना हमेशा चलाएं।

- `raise` एक निर्दिष्ट अपवाद को बढ़ाने या एक पकड़े गए अपवाद को फिर से उठाने के लिए इस्तेमाल किया गया स्टेटमेंट होता है।

- `class` स्टेटमेंट, जो निष्पादित करता है कोड का एक ब्लॉक और ऑब्जेक्ट-ओरिएंटेड प्रोग्रामिंग में उपयोग के लिए अपने स्थानीय नामस्थान को एक वर्ग से जोड़ता है।

- `def` स्टेटमेंट, जो किसी फ़ंक्शन या विधि को परिभाषित करता है।

- `with` सितंबर 2006 को जारी पायथन 2.5 से स्टेटमेंट, जो एक संदर्भ प्रबंधक के भीतर एक कोड ब्लॉक संलग्न करता है (उदाहरण के लिए, कोड के ब्लॉक को चलाने से पहले लॉक प्राप्त करना और बाद में लॉक जारी करना, या फ़ाइल खोलना और फिर इसे बंद करना), अनुमति देना रिसोर्स एक्विजिशन इज़ इनिशियलाइज़ेशन (RAII) जैसा व्यवहार और एक सामान्य कोशिश/आखिरकार मुहावरे की जगह लेता है।

- `pass` स्टेटमेंट, जो एनओपी के रूप में कार्य करता है। खाली बनाने के लिए वाक्य रचना की आवश्यकता हैकोड खंड मैथा।

- `assert` स्टेटमेंट, डिबगिंग के दौरान उपयोग की जाने वाली स्थितियों की जांच करने के लिए उपयोग किया जाता है।

- `yield` विवरण, जो जनरेटर फ़ंक्शन से एक मान देता है। पायथन 2.5 से उपज एक ऑपरेटर है। इस फॉर्म का उपयोग कोरआउटिन को लागू करने के लिए किया जाता है।

- `import` स्टेटमेंट, जो मॉड्यूल आयात करने के लिए उपयोग किया जाता है जिनके कार्यों या वेरिएबल का उपयोग वर्तमान कार्यक्रम में किया जा सकता है। आयात का उपयोग करने के तीन तरीके हैं: `import <mod- ule name> [as <alias>]` or `from <module name> import * or from <module name> import <definition 1> [as <alias 1>], <definition 2> [as <alias 2>], ....`

- `print` स्टेटमेंट में बदल दिया गया था प्रिंट () फंक्शन पायथन 3 में है।

पायथन टेल कॉल ऑप्टिमाइज़ेशन या प्रथम श्रेणी निरंतरता का समर्थन नहीं करता है और गुइडो वैन रोसुम के अनुसार, यह कभी नहीं होता है। हालांकि कोरआउटिन जैसी कार्यक्षमता के लिए बेहतर समर्थन 2.5 में प्रदान किया जाता है, पायथन के जनरेटर का विस्तार करके ऐसा किया जाता है। 2.5 से पहले, जनरेटर आलसी पुनरावृत्त थे; सूचना जनरेटर के बाहर अप्रत्यक्ष रूप से पारित की गई थी। पायथन 2.5 से जानकारी को जनरेटर फ़ंक्शन में वापस भेजना संभव होता है और पायथन 3.3 से जानकारी को कई स्टैक स्तरों के माध्यम से पारित किया जा सकता है।

एक्सप्रेशन

कुछ पायथन एक्सप्रेशन सी और जावा जैसी लैंग्वेजों के समान होती हैं, जबकि कुछ ऐसी नहीं हैं:

- जोड़, घटाव और गुणा एक ही है, लेकिन भाग का व्यवहार अलग है। पायथन में दो प्रकार के विभाजन होते हैं। वे फ्लोर डिवीजन (या पूर्णांक डिवीजन) // और फ्लोटिंग पॉइंट / डिवीजन हैं। पायथन ने घातांक के लिए ** ऑपरेटर भी जोड़ा।

- पायथन 3.5 से, नया @ इंफिक्स ऑपरेटर पेश किया गया था। इसका उद्देश्य मैट्रिक्स गुणन के लिए NumPy जैसे पुस्तकालयों द्वारा उपयोग किया जाना है।

- पायथन 3.8 से, सिंटैक्स: =, जिसे 'वालरस ऑपरेटर' कहा जाता है, पेश किया गया था। यह एक बड़े व्यंजक के भाग के रूप में चरों को मान प्रदान करता है।

- पायथन में == मूल्य से तुलना करता है जावा, जो मूल्य द्वारा संख्याओं की तुलना करता है और संदर्भ द्वारा वस्तुओं की तुलना करता है। (जावा में वस्तुओं पर मूल्य तुलना के साथ किया जा सकता है () विधि।) पायथन के ऑपरेटर का उपयोग वस्तु पहचान (संदर्भ द्वारा तुलना) की तुलना करने के लिए किया जा सकता है। पायथन में तुलना को जंजीर में बांधा जा सकता है, उदाहरण के लिए `a <= b <= c`.

- पायथन शब्दों का उपयोग करता है `and, or, not` के लिए इसके बूलियन ऑपरेटरों के बजाय प्रतीकात्मक `&&, ||, !` जावा और सी का उपयोग किया गया है।

- पायथन में एक प्रकार की अभिव्यक्ति होती है जिसे सूची समझ कहा जाता है। पायथन 2.4 विस्तारित सूची की समझ को एक अधिक सामान्य अभिव्यक्ति में एक जनरेटर अभिव्यक्ति कहा जाता है।

- बेनामी फ़ंक्शंस लैम्ब्डा एक्सप्रेशन का उपयोग करके कार्यान्वित किए जाते हैं; हालांकि, ये सीमित हैं कि शरीर केवल एक अभिव्यक्ति हो सकता है।

- सशर्त अभिव्यक्ति पायथन में के रूप में लिखा गया है एक्स अगर सी और वाई (से ऑपरेंड के क्रम में भिन्न सी ? एक्स: वाईऑपरेटर आम करने के लिए कई अन्य लैंग्वेजएँ)।

- पायथन सूचियों और टुपल्स के बीच अंतर करता है। सूचियां `[1, 2, 3]` के रूप में लिखी जाती हैं, परिवर्तनशील होती हैं और इन्हें शब्दकोशों की कुंजियों के रूप में उपयोग नहीं किया जा सकता है (पायथन में शब्दकोश कुंजियाँ अपरिवर्तनीय होनी चाहिए)। टुपल्स (1, 2, 3) के रूप में लिखे गए हैं, यह अपरिवर्तनीय हैं और इस प्रकार शब्दकोशों की कुंजी के रूप में उपयोग किया जा सकता है, बशर्ते ट्यूपल के सभी तत्व अपरिवर्तनीय हों। + ऑपरेटर का उपयोग दो टुपल्स को जोड़ने के लिए किया जा सकता है, जो सीधे उनकी सामग्री को संशोधित नहीं करता है, बल्कि एक नया टपल उत्पन्न करता है जिसमें दोनों प्रदान किए गए टुपल्स के तत्व होते हैं। इस प्रकार, प्रारंभिक रूप से (1, 2, 3) के बराबर वेरिएबल t दिया गया है, t = t + (4, 5) निष्पादित करने से पहले t + (4, 5) का मूल्यांकन होता है, जो (1, 2, 3, 4, 5), जिसे बाद में t को सौंपा जाता है, जिससे t की सामग्री को प्रभावी ढंग से "संशोधित" किया जाता है, जबकि टपल वस्तुओं की अपरिवर्तनीय प्रकृति के अनुरूप होता है।

- पायथन में सीक्वेंस अनपैकिंग की सुविधा होती है, जहां कई एक्सप्रेशन, प्रत्येक का मूल्यांकन किसी भी चीज (एक वेरिएबल , एक लिखने योग्य संपत्ति, आदि) के लिए किया जा सकता है, उसी तरह से जुड़े होते हैं जो टपल लिटरल बनाते हैं और, एक पूरे के रूप में, डाल दिए जाते हैं असाइनमेंट स्टेटमेंट में बराबर चिह्न के बाईं ओर। स्टेटमेंट बराबर चिह्न के दाहिने हाथ की ओर एक चलने योग्य वस्तु की अपेक्षा करता है जो प्रदान किए गए लिखने योग्य अभिव्यक्तियों के समान मूल्यों का उत्पादन करता है और इसके माध्यम से पुनरावृत्त करेगा, प्रत्येक उत्पादित मानों को बाईं ओर संबंधित अभिव्यक्ति में निर्दिष्ट करता है।

- पायथन में एक "स्ट्रिंग फोर्मेट" ऑपरेटर है %. यह समान कार्य करता है printf प्रारूप स्ट्रिंग सी में, उदाहरण के लिए "स्पैम=%s एग=%d"% ("ब्लाह", 2) का मूल्यांकन करता है `"spam=%s eggs=%d".` पायथन 3 और 2.6+ में, इसके द्वारा पूरक किया गया थाप्रारूप() str वर्ग की विधि, `e.g. "spam={0} eggs={1}".format("blah", 2). Python 3.6 added "f-strings": blah = "blah"; eggs = 2; f'spam={blah} eggs={eggs}'.`

- पायथन में विभिन्न प्रकार के स्ट्रिंग अक्षर होते हैं:

  - एकल या दोहरे उद्धरण चिह्नों द्वारा सीमांकित स्ट्रिंग्स। यूनिक्स शेल, पर्ल और पर्ल-प्रभावित लैंग्वेजों के विपरीत, एकल उद्धरण चिह्न और दोहरे उद्धरण चिह्न समान रूप से कार्य करते हैं। दोनों प्रकार के स्ट्रिंग बैकस्लैश (\) को एस्केप कैरेक्टर के रूप में उपयोग करते हैं। स्ट्रिंग इंटरपोलेशन पायथन 3.6 में "स्वरूपित स्ट्रिंग अक्षर" के रूप में उपलब्ध होता है।

- ट्रिपल-क्युटेड स्ट्रिंग्स, जो तीन सिंगल या डबल कोट की श्रृंखला के साथ शुरू और समाप्त होती हैं वे कई पंक्तियों और कार्य कर सकते हैं जैसे यहां गोले, पर्ल और रूबी में डाक्यूमेंट्स ।
- रॉ स्ट्रिंग किस्में, स्ट्रिंग अक्षर को r के साथ उपसर्ग करके दर्शाती हैं। भागने के दृश्यों की व्याख्या नहीं की जाती है; इसलिए रॉ तार उपयोगी होते हैं जहां शाब्दिक बैकस्लैश सामान्य होते हैं, जैसे नियमित अभिव्यक्ति और विंडोज-शैली पथ। C# में "@-कोटिंग" की तुलना करें।

- पायथन में सूचियों पर सरणी अनुक्रमणिका और सरणी स्लाइसिंग अभिव्यक्तियां हैं, जिन्हें के रूप में दर्शाया गया है एक चाबी], एक [शुरू: बंद करो] या ए [प्रारंभ: रोकें: वेरिएबल ण]। इंडेक्स शून्य-आधारित और नकारात्मक इंडेक्स हैं- es अंत के सापेक्ष हैं। स्लाइस स्टार्ट इंडेक्स से तत्वों को ऊपर ले जाते हैं, लेकिन स्टॉप इंडेक्स को शामिल नहीं करते हैं। तीसरा स्लाइस पैरामीटर, जिसे स्टेप या स्ट्राइड कहा जाता है, तत्वों को छोड़ने और उलटने की अनुमति देता है। उदाहरण के लिए, स्लाइस इंडेक्स को छोड़ा जा सकता हैए[:] पूरी सूची की एक प्रति लौटाता है। स्लाइस का प्रत्येक तत्व एक उथली प्रति है।

सामान्य लिस्प, स्कीम या रूबी जैसी भाषाओं के विपरीत, पायथन में अभिव्यक्तियों और बयानों के बीच अंतर को सख्ती से लागू किया जाता है। यह कुछ कार्यक्षमता को डुप्लिकेट करने की ओर जाता है। उदाहरण के लिए:

- सूची की समझ बनाम लूप के लिए।
- सशर्त अभिव्यक्ति बनाम if ब्लॉक।
- `eval ()` बनाम `exec ()` अंतर्निहित कार्य (पायथन 2 में, exec एक स्टेटमेंट है); पूर्व अभिव्यक्ति के लिए होती है इसके बाद वाले बयानों के लिए होता है।

स्टेटमेंट एक अभिव्यक्ति का हिस्सा नहीं हो सकता है, इसलिए सूची और अन्य समझ या लैम्ब्डा एक्सप्रेशन, सभी एक्सप्रेशन में स्टेटमेंट नहीं हो सकते हैं। इसका एक विशेष मामला यह है कि एक असाइनमेंट स्टेटमेंट जैसे a = 1 एक सशर्त स्टेटमेंट की सशर्त अभिव्यक्ति का हिस्सा नहीं बन सकता है। यह एक क्लासिक सी त्रुटि से बचने का उपाए है, एक असाइनमेंट ऑपरेटर के लिए = एक समानता ऑपरेटर के लिए == शर्तों में: अगर (सी = 1) {...} वाक्य रचनात्मक रूप से मान्य है (लेकिन शायद अनपेक्षित) सी कोड लेकिन अगर c = 1: ... पायथन में सिंटैक्स त्रुटि का कारण बनता है।

तरीके

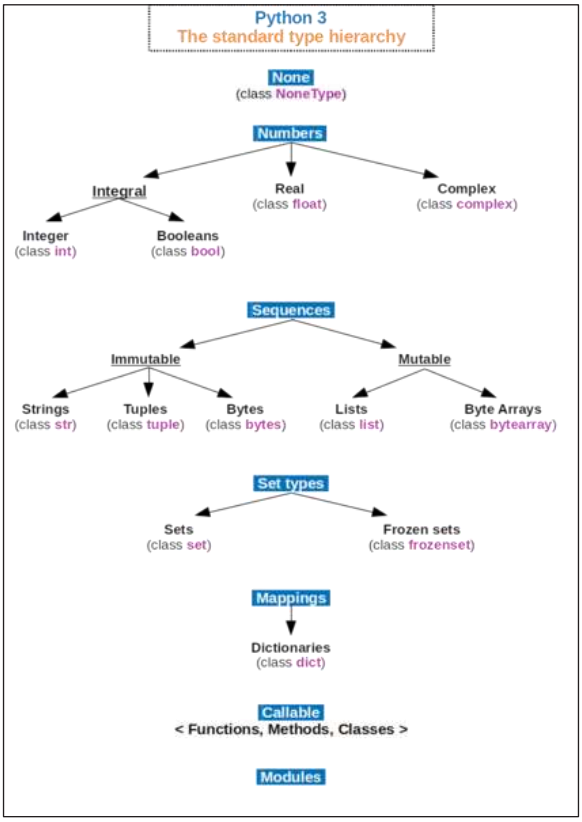
वस्तुओं पर विधियों वस्तु के वर्ग से जुड़े कार्य होते हैं यह वाक्यविन्यास उदाहरण है। method (argument), सामान्य तरीकों और कार्यों के लिए, `class.method (instance, argument)` के लिए वाक्य रचनात्मक है। कुछ अन्य ऑब्जेक्ट-ओरिएंटेड प्रोग्रामिंग लैंग्वेजों (जैसे, `C++`, `Java`, `Objective-C`, या `Ruby`) में निहित स्वयं (या यह) के विपरीत, पायथन विधियों में इंस्टेंस डेटा तक पहुँचने के लिए एक स्पष्ट स्व पैरामीटर है।

टाइपिंग

पायथन डक टाइपिंग का उपयोग करता है और इसमें ऑब्जेक्ट टाइप किये जाते हैं लेकिन अनटाइप्ड वेरिएबल नाम होते हैं। संकलन समय पर टाइप की बाधाओं की जांच नहीं की जाती है; बल्कि, किसी वस्तु पर संचालन विफल हो सकता है, यह दर्शाता है कि दी गई वस्तु उपयुक्त प्रकार की नहीं है। गतिशील रूप से टाइप किए जाने के बावजूद, पायथन दृढ़ता से टाइप किया गया है, जो कि अच्छी तरह से परिभाषित नहीं होते हैं (उदाहरण के लिए, एक स्ट्रिंग में एक संख्या जोड़ना) के बजाय चुपचाप उन्हें समझने का प्रयास करने के अलावा किया जाता है।

पायथन प्रोग्रामर को कक्षाओं का उपयोग करके अपने स्वयं के प्रकारों को परिभाषित करने की अनुमति देता है, जिनका उपयोग अक्सर वस्तुउन्मुख कार्यकर्म के लिए किया जाता है।

कक्षाओं के नए उदाहरणों का निर्माण वर्ग को कॉल करके किया जाता है (उदाहरण के लिए, SpamClass () या EggsClass ()), और कक्षाएं मेटाक्लास प्रकार के उदाहरण हैं (स्वयं का एक उदाहरण), मेटाप्रोग्रामिंग और प्रतिबिंब की अनुमति देता है।



पायथन 3 में मानक प्रकार पदानुक्रम।

संस्करण 3.0 से पहले, पायथन में दो प्रकार की कक्षाएं थीं जिसमें पुरानी शैली और नई शैली शामिल थी। यह दोनों शैलियों का सिंटैक्स समान है इसमें अंतर यह है कि क्या वर्ग वस्तु को प्रत्यक्ष या अप्रत्यक्ष रूप से विरासत में मिला है (सभी नई शैली की कक्षाएं वस्तु से विरासत में मिली हैं और प्रकार के उदाहरण हैं)। पायथन 2.2 के बाद के पायथन 2 के संस्करणों में, दोनों प्रकार की कक्षाओं का उपयोग किया जा सकता है। पायथन 3.0 में पुरानी शैली की कक्षाओं को समाप्त कर दिया गया था।

दीर्घकालिक योजना क्रमिक टाइपिंग का समर्थन करता है और पायथन 3.5 से लैंग्वेज का सिंटैक्स स्थिर प्रकारों को निर्दिष्ट करने की अनुमति देता है लेकिन डिफ़ॉल्ट कार्यान्वयन, सी-पायथन में उनकी जांच नहीं की जाती है। माईपी नामक एक प्रयोगात्मक वैकल्पिक स्थिर प्रकार चेकर संकलन-समय प्रकार की जांच का समर्थन करता है।

तालिका: पायथन 3 के अंतर्निहित प्रकारों का सारांश।

तरीके	अस्थिरता	विवरण	सिंटैक्स उदाहरण
`bool`	अडिग	बूलियन मान	`True False`

`bytearray`	परिवर्तनशील	बाइट्स का क्रम	`bytearray(b'Some ASCII')`
			`bytearray(b"SomeASCII")`
			`bytearray([119, 105, 107, 105])`

bytes	अडिग	बाइट्स का क्रम	b'Some ASCII' b"Some ASCII" bytes([119, 105, 107, 105])
complex	अडिग	जटिल वास्तविक और काल्पनिक भागों वाली संख्या	3+2.7j
dict	परिवर्तनशील	कुंजी और मूल्य जोड़े के सहयोगी सरणी (या शब्दकोश); मिश्रित प्रकार (कुंजी और मान) हो सकते हैं, कुंजियाँ एक हैशेबल प्रकार की होनी चाहिए	{'key1': 1.0, 3: False}
ellipsis[a]	अडिग	एक इलिप्सिस प्लेसहोल्डर को NumPy सरणियों में एक इंडेक्स के रूप में इस्तेमाल किया जाना है	... Ellipsis
float	अडिग	फ़्लोटिंग पॉइंट नंबर, सिस्टम-परिभाषित परिशुद्धता	3.1415927
frozenset	अडिग	अनियंत्रित सेट, इसमें शामिल हैकोई डुप्लिकेट नहीं; मिश्रित प्रकार हो सकते हैं, यदि हैशबल है	frozenset([4.0, 'string', True])
int	अडिग	असीमित परिमाण का पूर्णांक	42
list	परिवर्तनशील	सूची में मिश्रित प्रकार हो सकते हैं	[4.0, 'string', True]
NotImplementedType[a]	अडिग	मूल्य की अनुपस्थिति का प्रतिनिधित्व करने वाली वस्तु।	None
NoneType[a]	अडिग	एक प्लेसहोल्डर जिसे असमर्थित ऑपरेंड प्रकारों को इंगित करने के लिए अतिभारित ऑपरेटरों से वापस किया जा सकता है।	Not Implemented
set	परिवर्तनशील	अनियंत्रित सेट, इसमें शामिल हैकोई डुप्लिकेट नहीं; मिश्रित प्रकार हो सकते हैं, यदि हैशबल है	{4.0, 'string', True}
str	अडिग	एक वर्ण स्ट्रिंग: यूनिकोड कोडपॉइंट्स का अनुक्रम	'Wikipedia' "Wikipedia" """Spanning multiple
tuple	अडिग	शामिल कर सकते हैं मिश्रित प्रकार	(4.0, 'string', True)
range	अडिग	आमतौर पर विशिष्ट संख्या में लूपिंग के लिए उपयोग की जाने वाली संख्याओं का एक क्रम के लिये छोरों	range(1, 10) range(10,-5,-2)

^a नाम से सीधे पहुंच योग्य नहीं है।

गणित

पायथन में अंकगणितीय ऑपरेटरों के लिए सामान्य प्रतीक होते हैं (+, -, *, /), और शेष ऑपरेटर % (जहां शेष ऋणात्मक हो सकता है, जैसे 4% -3 == -2)। इसमें यह भी है **घातांक के लिए, उदाहरण के लिए 5**3 == 125 तथा 9**0.5 == 3.0, और एक नया मैट्रिक्स गुणा करें @ ऑपरेटर संस्करण में शामिल होता है यानी ये सभी ऑपरेटर पारंपरिक गणित की तरह काम करते हैं; समान पूर्वता नियमों के साथ, ऑपरेटरों infix (या - अतिरिक्त रूप से एकात्मक हो सकते हैं)। इसके अतिरिक्त,इसमें एक यूनरी ऑपरेटर (~) है, जो अनिवार्य रूप से अपने एक तर्क के सभी बिट्स को उलट देता है। पूर्णांकों के लिए इसका अर्थ ~x=-x-1 है। अन्य ऑपरेटरों में बिटवाइज़ शिफ्ट ऑपरेटर x << y शामिल हैं, जो x को बाएं y स्थानों पर स्थानांतरित करता है, जो x*(2**y), और x >> y के समान है, जो x को दाईं ओर y स्थानों पर स्थानांतरित करता है, जैसा कि एक्स // (2 ** y)।

समय के साथ विभाजन का व्यवहार काफी बदल गया है ताकि पूर्णांकों के बीच विभाजन सटीक फ़्लोटिंग पॉइंट परिणाम उत्पन्न करे:

- पायथन 2.1 और पहले के सी डिवीजन व्यवहार का उपयोग करते हैं। ऑपरेटर पूर्णांक विभाजन है यदि दोनों ऑपरेंड पूर्णांक हैं और फ्लोटिंग-पॉइंट डिवीजन अन्यथा। पूर्णांक विभाजन 0 की ओर चक्कर लगाता है, उदाहरण के लिए 7/3 == 2 और `-7/3 == -2`।
- पायथन 2.2 पूर्णांक विभाजन को नकारात्मक अनंत की ओर गोल करने के लिए बदलता है, उदाहरण के लिए 7/3 == 2 और -7/3 == -3। फ्लोर डिवीजन // ऑपरेटर पेश किया गया है। तो `7//3 == 2, -7//3 == -3, 7.5//3 == 2.0` तथा -7.5//3 == -3.0। से `__future__ import division` कारण विभाजन के लिए पायथन 3.0 नियमों का उपयोग करने के लिए एक मॉड्यूल की तरह होता है
- पायथन 3.0 परिवर्तन / हमेशा फ़्लोटिंग-पॉइंट डिवीजन होने के लिए, उदाहरण के लिए 5/2 == 2.5.

पायथन शब्दों में, / संस्करण 3.0 से पहले क्लासिक डिवीजन है, / संस्करणों में 3.0 और उच्चतर सही डिवीजन है, और // फ्लोर डिवीजन है।

अधिकांश लैंग्वेजों से अलग होने के बावजूद, नकारात्मक अनंतता की ओर बढ़ना, निरंतरता जोड़ता है। उदाहरण के लिए इसका अर्थ है कि समीकरण (a + b)//b == a//b + 1 हमेशा सत्य होता है। इसका यह भी अर्थ है कि समीकरण b*(a//b) + a%b == a, a के धनात्मक और ऋणात्मक दोनों मानों के लिए मान्य होता है। हालांकि, इस समीकरण की वैधता को बनाए रखने का मतलब है कि एक% बी का परिणाम, जैसा कि अपेक्षित है, आधे खुले अंतराल [0, बी) में है, जहां बी एक सकारात्मक पूर्णांक है, इसे अंतराल में स्थित होना चाहिए (बी) , 0] जब b ऋणात्मक हो पायथन प्रदान करता है a एक फ्लोट को निकटतम पूर्णांक में गोल करने के लिए गोल कार्य। टाई-ब्रेकिंग के लिए, 3 से पहले के संस्करण राउंड-अवे-फ्रॉम-जीरो: राउंड (0.5) का उपयोग करते हैं1.0 है, राउंड (-0.5) -1.0 है। अजगर3 राउंड टू इवन का उपयोग करता है: गोल (1.5) 2 है, गोल(2.5) 2 है।

पायथन कई समानता संबंधों के साथ बूलियन अभिव्यक्तियों की अनुमति देता है जो गणित में सामान्य उपयोग के अनुरूप होते हैं। उदाहरण के लिए, व्यंजक a < b < c परीक्षण करता है कि a, b से छोटा है और b, c से छोटा है। सी-व्युत्पन्न लैंग्वेजएं इस अभिव्यक्ति की अलग-अलग व्याख्या करती हैं: सी में, अभिव्यक्ति पहले एक <बी का मूल्यांकन करेगी, जिसके परिणामस्वरूप 0 या 1 होगा, और उस परिणाम की तुलना सी के साथ की जाती है।

मनमाने-सटीक अंकगणित के लिए पायथन के पास व्यापक अंतर्निहित समर्थन है। पूर्णांकों को मशीन-समर्थित अधिकतम निश्चित-परिशुद्धता (आमतौर पर 32 या 64 बिट्स) से, अजगर प्रकार के int से संबंधित, मनमाने ढंग से सटीक, पायथन प्रकार लंबे से संबंधित, जहां आवश्यक हो, से पारदर्शी रूप से स्विच किया जाता है। उत्तरार्द्ध में उनके शाब्दिक प्रतिनिधित्व में "एल" प्रत्यय होता है। (पायथन 3 में, `int` और `long` प्रकार के बीच का अंतर समाप्त कर दिया गया था; यह व्यवहार अब पूरी तरह से int वर्ग द्वारा समाहित है।) दशमलव प्रकार/वर्ग मॉड्यूल दशमलव में (संस्करण 2.4 के बाद से) मनमाने ढंग से परिशुद्धता के लिए दशमलव फ़्लोटिंग पॉइंट नंबर प्रदान करता है और कई गोलाई मोड। मॉड्यूल भिन्नों में भिन्न प्रकार (संस्करण 2.6 के बाद से) परिमेय संख्याओं के लिए मनमाना परिशुद्धता प्रदान करता है।

पायथन के व्यापक गणित लाइब्रेरी और तीसरे पक्ष के लाइब्रेरी NumPy के कारण, जो मूल क्षमताओं को और बढ़ाता है, इसे अक्सर संख्यात्मक डेटा प्रोसेसिंग और हेरफेर जैसी समस्याओं में सहायता के लिए एक वैज्ञानिक स्क्रिप्टिंग लैंग्वेज के रूप में उपयोग किया जाता है।

पायथन प्रोग्रामिंग के उदाहरण

नमस्ते विश्व कार्यक्रम:

```
print('Hello, world!')
```

एक सकारात्मक पूर्णांक के भाज्य की गणना करने के लिए कार्यक्रम:

```
n = int(input('Type a number, then its factorial will be printed:
')) if n < 0:
      raise ValueError('You must enter a positive
number') fact = 1
i = 2
while i <= n:
      fact = fact * i
      i += 1
print(fact)
```

लाइब्रेरी

पायथन के बड़े मानक लाइब्रेरी , जिसे आमतौर पर इसकी सबसे बड़ी ताकत के रूप में उद्धृत किया जाता है, यह कई कार्यों के लिए उपयुक्त उपकरण प्रदान करता है। इंटरनेट से जुड़े अनुप्रयोगों के लिए, कई मानक प्रारूप और प्रोटोकॉल जैसे एमआईएमई और एचटीटीपी समर्थित हैं। इसमें ग्राफिकल यूजर इंटरफेस बनाने, रिलेशनल डेटाबेस से जुड़ने, छद्म यादृच्छिक संख्याएं उत्पन्न करने, मनमानी-सटीक दशमलव के साथ अंकगणित, नियमित अभिव्यक्तियों में हेरफेर करने और यूनिट परीक्षण के लिए मॉड्यूल शामिल होते हैं।

मानक लाइब्रेरी के कुछ हिस्से विनिर्देशों द्वारा कवर किए गए हैं (उदाहरण के लिए, वेब सर्वर गेटवे इंटरफेस (डब्लूएसजीआई) कार्यान्वयन वस ग्रिफ पीईपी 333 का पालन करता है), लेकिन अधिकांश मॉड्यूल नहीं हैं। वे उनके कोड, आंतरिक दस्तावेज़ीकरण और परीक्षण सूट (यदि आपूर्ति की जाती है) द्वारा निर्दिष्ट किए जाते हैं। हालांकि, अधिकांश मानक लाइब्रेरी क्रॉस-प्लेटफ़ॉर्म पायथन कोड होते हैं, केवल कुछ मॉड्यूल को भिन्न कार्यान्वयन के लिए बदलने या फिर से लिखने की आवश्यकता होती है।

नवंबर 2019 तक, पायथन पैकेज इंडेक्स (पेपाई), थर्ड-पार्टी पायथन सॉफ्टवेयर के लिए आधिकारिक रिपॉजिटरी में 200,000 से अधिक पैकेज शामिल हैं, जिनमें कार्यक्षमता की एक विस्तृत श्रृंखला है, जिसमें शामिल हैं:

- चित्रात्मक उपयोगकर्ता इंटरफेस
- वेब फ्रेमवर्क
- मल्टीमीडिया
- डेटाबेस
- नेटवर्किंग
- टेस्ट फ्रेमवर्क
- स्वचालन
- वेब स्क्रेपिंग

- प्रलेखन
- सिस्टम एडमिनिस्ट्रेशन
- वैज्ञानिक कंप्यूटिंग
- टेक्स्ट प्रोसेसिंग
- इमेज प्रोसेसिंग

कार्यान्वयन

संदर्भ कार्यान्वयन

सी-पायथन पायथन का संदर्भ कार्यान्वयन है। यह C में लिखा गया है, कई चुनिंदा C99 सुविधाओं के साथ C89 मानक को पूरा करता है। यह पाइथन प्रोग्राम को एक इंटरमीडिएट बाइटकोड में संकलित करता है जिसे बाद में इसकी वर्चुअल मशीन द्वारा निष्पादित किया जाता है। सी-पायथन को C और देशी पायथन के मिश्रण में लिखे गए एक बड़े मानक लाइब्रेरी के साथ वितरित किया जाता है। यह विंडोज और अधिकांश आधुनिक यूनिक्स जैसी प्रणालियों सहित कई प्लेटफार्मों के लिए उपलब्ध है। प्लेटफॉर्म पोर्टेबिलिटी इसकी शुरुआती प्राथमिकताओं में से एक थी।

अन्य कार्यान्वयन

`PyPy`, पायथन 2.7 और 3.5 का एक तेज़, आज्ञाकारी इंटरप्रेटर है। इसका जस्ट-इन-टाइम कंपाइलर सी-पायथन पर एक महत्वपूर्ण गति सुधार लाता है।

स्टैकलेस पायथन सी-पायथन का एक महत्वपूर्ण फोर्क है जो माइक्रोथ्रेड्स को लागू करता है; यह सी मेमोरी स्टैक का उपयोग नहीं करता है और इस प्रकार यह बड़े पैमाने पर समवर्ती कार्यक्रमों की अनुमति देता है। इतन ही नहीं `PyPy` का एक स्टैकलेस संस्करण भी है।

माइक्रोपायथन और सर्किटपायथन माइक्रोकंट्रोलर्स के लिए अनुकूलित पायथन 3 वेरिएंट हैं। इसमें लेगो माइंडस्टॉर्म EV3 शामिल है।

रस्टपायथन रस्ट में लिखा गया एक पायथन 3 इंटरप्रेटर है।

असमर्थित कार्यान्वयन

अन्य जस्ट-इन-टाइम पायथन कंपाइलर विकसित किए गए हैं, लेकिन अब असमर्थित हैं:

- गूगल ने 2009 में अनलेडैन स्वालो नाम से एक प्रोजेक्ट शुरू किया था, जिसका उद्देश्य एलएलवीएम का उपयोग करके पायथन इंटरप्रेटर को पांच गुना तेज करना और हजारों कोर तक स्केल करने के लिए इसकी मल्टीथ्रेडिंग क्षमता में सुधार करना था।
- साइको एक जस्ट-इन-टाइम विशेषज्ञता वाला कंपाइलर है जो सी-पायथन के साथ एकीकृत होता है और रनटाइम पर बाइटकोड को मशीन कोड में बदल देता है। उत्सर्जित कोड कुछ डेटा प्रकारों के लिए विशिष्ट है और मानक पायथन कोड से तेज़ होता है।

2005 में, नोकिया ने श्रृंखला 60 मोबाइल फोन के लिए एक पायथन इंटरप्रेटर जारी किया जिसका नाम PyS60 था। इसमें सी-पायथन कार्यान्वयन से कई मॉड्यूल और सिम्बियन ऑपरेटिंग सिस्टम के साथ एकीकृत करने के लिए कुछ अतिरिक्त मॉड्यूल को शामिल किया गया था । S60 प्लेटफॉर्म के सभी वेरिएंट्स पर चलने के लिए प्रोजेक्ट को अप-टू-डेट रखा गया था और भी कई थर्ड-पार्टी मॉड्यूल उपलब्ध थे। Nokia N900, जीटीके विजेट लाइब्रेरी के साथ पायथन को भी सपोर्ट करता है, जिससे प्रोग्राम को लक्ष्य डिवाइस पर लिखा और चलाया जा सकता है।

लैंग्वेज के लिए अन्य क्रॉस-कंपाइलर

उच्च-स्तरीय ऑब्जेक्ट भाषाओं के लिए कई कंपाइलर होते हैं, ये या तो अप्रतिबंधित पायथन, पायथन का एक प्रतिबंधित उपसमुच्चय, या स्रोत भाषा के रूप में पायथन के समान लैंग्वेज होती है:

- जायथन जावा बाइट कोड में संकलित करता है, जिसे तब प्रत्येक जावा वर्चुअल मशीन कार्यान्वयन द्वारा निष्पादित किया जा सकता है। यह पायथन प्रोग्राम से जावा क्लास लाइब्रेरी फ़ंक्शंस के उपयोग को भी सक्षम बनाता है।
- .नेट कॉमन लैंग्वेज रनटाइम पर पायथन प्रोग्राम चलाने के लिए आयरनपीथन एक समान दृष्टिकोण का अनुसरण करता है।
- आरपायथन लैंग्वेज को सी, जावा बाइटकोड या सामान्य इंटरमीडिएट लैंग्वेज में संकलित किया जा सकता है और इसका उपयोग पायथन के पाय-पाय इंटरप्रेटर के निर्माण के लिए किया जाता है।
- पे-जेएस पायथन को जावास्क्रिप्ट में संकलित करता है।
- साइथन पायथन को सी और सी ++ में संकलित करता है।
- पायथन को मशीन कोड में संकलित करने के लिए नुम्बा एलएलवीएम का उपयोग करता है।
- पाइथ्रन ने पायथन को C++ में संकलित किया जाता है।
- कुछ हद तक पाइरेक्स (2010 में नवीनतम रिलीज़) और शेड स्किन (2013 में नवीनतम रिलीज़) क्रमशः C और C++ में संकलित होते हैं।
- गूगल का ग्रम्पी पायथन को गो के लिए संकलित करता है।
- माय-एचडीएल पायथन को वीएचडीएल में संकलित करता है।
- नुइटका ने पायथन को C++ में संकलित किया है।

प्रदर्शन

एक गैर-संख्यात्मक (संयोजन-अल) कार्यभार पर विभिन्न पायथन कार्यान्वयन की एक प्रदर्शन तुलना यूरोसाइपी'13 में प्रस्तुत की गई थी।

विकास

पायथन का विकास बड़े पैमाने पर पायथन एन्हांसमेंट प्रस्ताव (पीईपी) प्रक्रिया के माध्यम से किया जाता है, जो प्रमुख नई सुविधाओं को प्रस्तावित करने, मुद्दों पर सामुदायिक इनपुट एकत्र करने और पायथन डिजाइन निर्णयों का डॉक्यूमेंटिंग करने के लिए प्राथमिक तंत्र होता है। पायथन कोडिंग शैली पीईपी 8 में शामिल होता है। बकाया पीईपी की समीक्षा की जाती है और पायथन समुदाय और स्टीयरिंग काउंसिल द्वारा टिप्पणी की जाती है।

लैंग्वेज की वृद्धि सी-पायथन संदर्भ कार्यान्वयन के विकास से मेल खाती है। मेलिंग सूची पायथन-देव लैंग्वेज के विकास के लिए प्राथमिक मंच है। विशिष्ट मुद्दों पर पायथन.ओआरजी पर बनाए गए राउंडअप बग ट्रैकर में चर्चा की गई है। विकास मूल रूप से एक स्व-होस्ट किए गए स्रोत-कोड रिपॉजिटरी पर हुआ है, जो मर्क्यूरियल चला रहा था, जब तक कि जनवरी 2017 में पायथन गिटहब में स्थानांतरित नहीं होता।

सी-पायथन की सार्वजनिक रिलीज़ तीन प्रकारों में आती है, जो इस बात से अलग है कि संस्करण संख्या के किस भाग में वृद्धि की गई है:

- पिछड़े-असंगत संस्करण, जहां कोड के टूटने की उम्मीद होती है और मैन्युअल रूप से पोर्ट करने की आवश्यकता होती है। संस्करण संख्या का पहला भाग बढ़ा हुआ है। ये रिलीज़ बहुत कम होते हैं—उदाहरण के लिए, संस्करण 3.0 2.0 के 8 साल बाद जारी किया गया था।

- प्रमुख या "सुविधा" रिलीज़, लगभग हर 18 महीने में और काफी हद तक संगत होती हैं लेकिन नई सुविधाएं पेश करती हैं। संस्करण संख्या का दूसरा भाग बढ़ा हुआ होता है। प्रत्येक प्रमुख संस्करण को जारी होने के बाद कई वर्षों तक बगफिक्स द्वारा समर्थित किया जाता है।

- बगफिक्स रिलीज़, जो कोई नई सुविधाएं पेश नहीं करती हैं, लगभग हर 3 महीने में होती हैं और तब बनाई जाती हैं जब पिछली रिलीज़ के बाद से पर्याप्त संख्या में बग्स को अपस्ट्रीम में ठीक किया जाता है। इन रिलीज में सुरक्षा कमजोरियों को भी पैच किया जाता है। इस संस्करण संख्या का तीसरा और अंतिम भाग बढ़ा हुआ होता है।

दिसंबर 2019 में पायथन 3.9 अल्फा 1 की उम्मीद की गई थी, लेकिन अंतिम संस्करण के लिए रिलीज की तारीख इस बात पर निर्भर करती है कि रिलीज की तारीखों के लिए कौन सा नया प्रस्ताव चर्चा के तहत तीन मसौदा प्रस्तावों के साथ अपनाया गया था और एक वार्षिक ताल एक विकल्प होता है।

कई अल्फा, बीटा और रिलीज-उम्मीदवारों को पूर्वावलोकन के रूप में और अंतिम रिलीज से पहले परीक्षण के लिए भी जारी किया जाता है। यद्यपि प्रत्येक रिलीज़ के लिए एक मोटा शेड्यूल होता है, कोड तैयार नहीं होने पर उन्हें अक्सर देरी हो जाती है। पायथन की विकास टीम विकास के दौरान बड़ी इकाई परीक्षण सूट चलाकर और बिल्डबॉट निरंतर एकीकरण प्रणाली का उपयोग करके कोड की स्थिति की निगरानी करती है।

पायथन डेवलपर्स के समुदाय ने तीसरे पक्ष के पायथन पुस्तकालयों के आधिकारिक भंडार, पायथन पैकेज इंडेक्स (पीईपीआई) में 86,000 से अधिक सॉफ्टवेयर मॉड्यूल का योगदान दिया है।

पाइकॉन पर प्रमुख शैक्षणिक सम्मेलन पाइकॉन होता है। पाइलडीज जैसे विशेष पायथन मेंटरिंग प्रोग्राम भी हैं।

नामकरण

पायथन का नाम ब्रिटिश कॉमेडी समूह मोंटी पायथन से लिया गया है, जिसे पायथन निर्माता गुइडो वैन रोसुम ने लैंग्वेज विकसित करते समय आनंद लिया था। मोंटी पायथन संदर्भ अक्सर पायथन कोड और संस्कृति में दिखाई देते हैं; उदाहरण के लिए, पायथन साहित्य में अक्सर उपयोग किए जाने वाले मेटासिंटेक्टिक वेरिएबल पारंपरिक फू और बार के बजाय स्पैम और एग होते हैं। आधिकारिक पायथन प्रलेखन में मोंटी पायथन रूटीन के विभिन्न संदर्भ भी शामिल हैं।

उपसर्ग Py- का उपयोग यह दिखाने के लिए किया जाता है कि कुछ पायथन से संबंधित है। पायथन अनुप्रयोगों या पुस्तकालयों के नामों में इस उपसर्ग के उपयोग के उदाहरणों में शामिल हैं Pygame, SDL को पायथन से बांधना (आमतौर पर गेम बनाने के लिए उपयोग किया जाता है); पीईक्यूटी और पीईजीटीके, जो क्रमशः क्यूटी और जीटीके को पायथन से बांधते हैं; और PyPy, एक पायथन कार्यान्वयन जो मूल रूप से पायथन में लिखा गया था।

एपीआई डॉक्यूमेंटेशन जेनरेटर

पायथन एपीआई डॉक्यूमेंटेशन जनरेटर में शामिल हैं:

- स्फिंक्स

- एपिडोक
- हैडरडॉक
- पाइडोक

उपयोग

पायथन सॉफ्टवेयर की सूची

2003 के बाद से पायथन ने लगातार टिओब प्रोग्रामिंग कम्युनिटी इंडेक्स में शीर्ष दस सबसे लोकप्रिय प्रोग्रामिंग भाषाओं में स्थान दिया है, जहां दिसंबर 2018 तक यह तीसरी सबसे लोकप्रिय लैंग्वेज (जावा और सी के बाद) है। इसे 2007, 2010 और 2018 में प्रोग्रामिंग लैंग्वेज ऑफ द ईयर चुना गया था।

एक अनुभवजन्य अध्ययन में पाया गया कि स्क्रिप्टिंग लैंग्वेज जैसे कि पायथन, पारंपरिक लैंग्वेजों की तुलना में अधिक उत्पादक हैं, जैसे कि सी और जावा, प्रोग्रामिंग समस्याओं के लिए स्ट्रिंग हेरफेर और एक शब्दकोश में खोज और निर्धारित किया कि मेमोरी खपत अक्सर "जावा से बेहतर" थी और यह सी या सी ++ से ज्यादा अच्छी थी।

पायथन का उपयोग करने वाले सबसे बड़े संगठनों में विकिपीडिया, गूगल, याहू!, सर्न, नासा, फेसबुक, अमेज़न, इंस्टाग्राम, स्पॉटिफाय और आईएलएम और आईटीए जैसी कुछ छोटी सस्थाएं शामिल थीं। सोशल न्यूज नेटवर्किंग साइट रेडिट पूरी तरह से पायथन में लिखी गई है।

पायथन वेब अनुप्रयोगों के लिए एक स्क्रिप्टिंग लैंग्वेज के रूप में काम कर सकता है, उदाहरण के लिए अपाचे वेब सर्वर के लिए मोड_वेस्गी के माध्यम से इसका इस्तेमाल करता है। वेब सर्वर गेटवे इंटरफेस के साथ इन अनुप्रयोगों को सुविधाजनक बनाने के लिए एक मानक एपीआई विकसित किया गया है। डीजांगो, पाएलोंस, पिरामिड, टर्बोगियर्स, वेब2पे, टोरंडो, फ्लास्क, बॉटल और जोप जैसे वेब फ्रेमवर्क जटिल अनुप्रयोगों के डिजाइन और रखरखाव में डेवलपर्स का समर्थन करते हैं। अजाक्स-आधारित अनुप्रयोगों के क्लाइंट-साइड को विकसित करने के लिए पेजस और आयरनपायथन का उपयोग किया जा सकता है। एसक्यूएलअल्काई को रिलेशनल डेटाबेस में डेटा मैपर के रूप में इस्तेमाल किया जा सकता है। ट्विस्टेड कंप्यूटर के बीच संचार को प्रोग्राम करने के लिए एक ढांचा है और ड्रॉपबॉक्स द्वारा इसका उपयोग (उदाहरण के लिए) किया जाता है।

नमपे, स्कीपाय और मेटप्लोलिब जैसे लाइब्रेरी वैज्ञानिक कंप्यूटिंग में पायथन के प्रभावी उपयोग की अनुमति देते हैं, विशेष लाइब्रेरी जैसे कि बायोपायथन और एसट्राफी' डोमेन-विशिष्ट कार्यक्षमता प्रदान करते हैं। सेजमैथ एक गणितीय सॉफ्टवेयर है जिसमें एक नोटबुक इंटरफ़ेस है जिसे पायथन में प्रोग्राम किया जा सकता है: इसके लाइब्रेरी में गणित के कई पहलुओं को शामिल किया गया है, जिसमें बीजगणित, कॉम्बिनेटरिक्स, संख्यात्मक गणित, संख्या सिद्धांत और कलन शामिल होते हैं।

पायथन को कई सॉफ्टवेयर उत्पादों में एक स्क्रिप्टिंग लैंग्वेज के रूप में सफलतापूर्वक एम्बेड किया गया है, जिसमें अबाकस जैसे परिमित तत्व विधि सॉफ्टवेयर, फ्रीकैड जैसे 3 डी पैरामीट्रिक मॉडलर, 3 डी एनिमेशन पैकेज जैसे 3 डी मैक्स, ब्लेंडर, सिनेमा 4 डी, लाइटवेव, हौदिनी, माया, मोडो शामिल हैं। मोशनबिल्डर, सॉफ्टिमेज, विजुअल इफेक्ट्स कंपोजिटर न्यूक, 2डी इमेजिंग प्रोग्राम जैसे जीआईएमपी, इंकस्केप, स्क्रिबस और पेंट शॉप प्रो और म्यूजिकल नोटेशन प्रोग्राम जैसे स्कोरराइटर और कैपेला शामिल हैं। जीएनयू डीबगर सी ++ कंटेनर जैसी जटिल संरचनाओं को दिखाने के लिए पाइथन को एक सुंदर प्रिंटर के रूप में उपयोग करता है। एसरी, अर्कगिस में स्क्रिप्ट लिखने के लिए पायथन को सबसे अच्छे विकल्प के रूप में बढ़ावा देता है। यह कई वीडियो गेम में भी इस्तेमाल किया गया है और गूगल ऐप इंजन में उपलब्ध तीन प्रोग्रामिंग भाषाओं में से पहली के रूप में अपनाया गया है, अन्य दो जावा और गो हैं।

टेन्सर फ्लो, केरस और स्किकिट-लर्न जैसे पुस्तकालयों की मदद से पायथन का उपयोग आमतौर पर कृत्रिम बुद्धिमत्ता परियोजनाओं में किया जाता है। मॉड्यूलर आर्किटेक्चर, सरल सिंटैक्स और रिच टेक्स्ट प्रोसेसिंग टूल्स के साथ एक स्क्रिप्टिंग लैंग्वेज के रूप में, पायथन का उपयोग अक्सर प्राकृतिक लैंग्वेज प्रसंस्करण के लिए किया जाता है।

कई ऑपरेटिंग सिस्टम में एक मानक घटक के रूप में पायथन को शामिल किया गया है। यह अधिकांश लिनक्स वितरणों, एमिगाऑएस 4, फ्रीबीएसडी (पैकेज के रूप में), नेटबीएसडी, ओपनबीएसडी (पैकेज के रूप में) और मैकऑस के साथ शिप करता है और कमांड लाइन (टर्मिनल) से उपयोग किया जा सकता है। कई लिनक्स वितरण पायथन में लिखे गए इंस्टॉलर का उपयोग करते हैं: उबंटू यूबिकिटी इंस्टॉलर का उपयोग करता है, जबकि रेड हैट लिनक्स और फेडोरा एनाकोंडा इंस्टॉलर का उपयोग करते हैं। जेनटू लिनक्स अपने पैकेज प्रबंधन सिस्टम, पोर्टेज में पायथन का उपयोग करता है।

सूचना सुरक्षा उद्योग में पायथन का व्यापक रूप से उपयोग किया जाता है, जिसमें शोषण विकास भी शामिल है।

वन लैपटॉप प्रति चाइल्ड एक्सओ के लिए अधिकांश शुगर सॉफ्टवेयर, जिसे अब शुगर लैब्स में विकसित किया गया है, इसे पायथन में लिखा गया है। रास्पबेरी पाई सिंगल-बोर्ड कंप्यूटर प्रोजेक्ट ने पायथन को अपनी मुख्य उपयोगकर्ता-प्रोग्रामिंग भाषा के रूप में अपनाया गया है।

लिब्रे ऑफिस में पायथन शामिल है और जावा को पायथन के साथ बदलने की कोशिश करता है। इसका पायथन स्क्रिप्टिंग प्रदाता 7 फरवरी 2013 से संस्करण 4.0 के बाद से एक मुख्य विशेषता है।

पायथन से प्रभावित लैंग्वेज

पायथन के डिजाइन और फिलोसोफी ने कई अन्य प्रोग्रामिंग लैंग्वेजों को प्रभावित किया है:

- बू इंडेंटेशन, एक समान सिंटैक्स और एक समान ऑब्जेक्ट मॉडल का उपयोग करता है।
- कोबरा इंडेंटेशन और इसी तरह के सिंटैक्स का उपयोग करता है और इसका "पावती" डाक्यूमेंट्स पायथन को उन लैंग्वेजों में सबसे पहले सूचीबद्ध करता है जिन्होंने इसे प्रभावित किया हो। हालांकि, कोबरा सीधे अनुबंध-दर-अनुबंध, इकाई परीक्षण और वैकल्पिक स्थिर टाइपिंग का समर्थन करता है।
- कॉफ़ीस्क्रिप्ट, एक प्रोग्रामिंग लैंग्वेज है जो जावास्क्रिप्ट को क्रॉस-कंपाइल करती है और यह पायथन-इन-स्पायर्ड सिंटैक्स है।
- ईसीएमए स्क्रिप्ट ने पायथन से इटरेटर और जेनरेटर उधार लिया है।
- इसे "पायथन जैसी गतिशील लैंग्वेज में काम करने की गति" के लिए डिज़ाइन किया गया है और यह स्लाइसिंग एरेज़ के लिए समान सिंटैक्स साझा करता है।
- ग्रोवी पायथन डिजाइन दर्शन को जावा में लाने की इच्छा से बनाया गया था।
- जूलिया को "टू मैक्रोज़ के साथ और सामान्य प्रोग्रामिंग के लिए पायथन के रूप में प्रयोग करने योग्य और सी के रूप में तेज़ होना चाहिए" के साथ डिजाइन किया गया था। जूलिया को या से कॉल करना संभव है; पेकॉल.जेएल और एक पायथन पैकेज के साथ पेजूलिया, दूसरी दिशा में पायथन से कॉल करने की अनुमति देता है।
- कोटलिन पायथन के समान एक इंटरैक्टिव शेल के साथ एक कार्यात्मक प्रोग्रामिंग लैंग्वेज है। हालांकि, मानक जावा पुस्तकालयों तक पहुंच के साथ कोटलिन दृढ़ता से टाइप किया गया है।
- रूबी के निर्माता, युकिहिरो मात्सुमोतो ने कहा है: "मैं एक स्क्रिप्टिंग लैंग्वेज चाहता था जो पर्ल से अधिक शक्तिशाली हो और पायथन की तुलना में अधिक ऑब्जेक्ट-ओरिएंटेड हो। इसलिए मैंने अपनी लैंग्वेज खुद डिजाइन करने का फैसला किया है।"
- ऐप्पल द्वारा विकसित एक प्रोग्रामिंग लैंग्वेज स्विफ्ट में कुछ पायथन-प्रेरित वाक्यविन्यास है।

- जीडी स्क्रिप्ट, गतिशील रूप से टाइप की गई प्रोग्रामिंग लैंग्वेज, जिसका उपयोग वीडियो-गेम बनाने के लिए किया जाता है। यह कुछ मामूली अंतरों के साथ बिल्कुल पायथन के समान होता है।

पायथन की विकास प्रथाओं का अन्य भाषाओं द्वारा भी अनुकरण किया गया है। उदाहरण के लिए टीसीएल और एरलांग में भाषा में बदलाव (पायथन में, एक पीईपी) के औचित्य और आसपास के मुद्दों का वर्णन करने वाले डाक्यूमेंट्स की आवश्यकता का अभ्यास भी किया जाता है।

पायथन ने 2007, 2010 और 2018 में टीआईओबीई की प्रोग्रामिंग लैंग्वेज ऑफ द ईयर पुरस्कार प्राप्त किया था। यह पुरस्कार उस लैंग्वेज को दिया जाता है, जिसकी लोकप्रियता में वर्ष के दौरान सबसे अधिक वृद्धि हुई है, जैसा कि टीआईओबीई इंडेक्स द्वारा मापा जाता है।

पायथन को कैसे इंस्टॉल करें

मैक और लिनक्स के साथ पायथन इंस्टॉल हो जाता है, लेकिन अगर आप विंडोज का उपयोग कर रहे हैं तो आपको इसे स्वयं इंस्टॉल करना होगा। यदि आप मैक या लिनक्स कंप्यूटर का उपयोग कर रहे हैं, तो आप नवीनतम सुविधाओं तक पहुँच सुनिश्चित करने के लिए नवीनतम संस्करण इंस्टॉल कर सकते हैं।

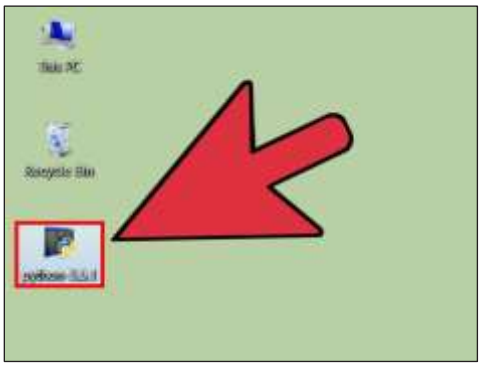
तरीका 1: विंडोज़

1. आप सबसे पायथन वेबसाइट पर जाएं: आप पाइथन वेबसाइट (python.org/downloads) से पाइथन के साथ आरंभ करने के लिए आवश्यक सभी चीजें डाउनलोड कर सकते हैं। वेबसाइट को स्वचालित रूप से पता लगाना चाहिए कि आप विंडोज का उपयोग कर रहे हैं और विंडोज इंस्टॉलर के लिंक प्रस्तुत करते हैं।

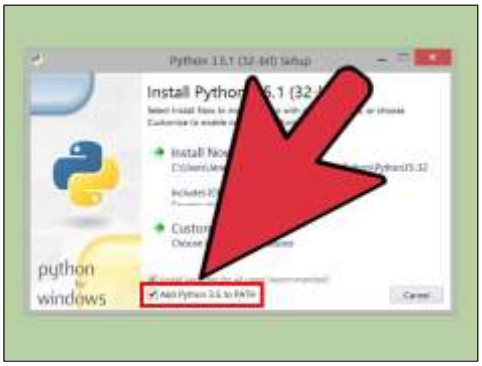
2. आप तय करें कि आप कौन सा संस्करण इंस्टॉल करना चाहते हैं: वर्तमान में पायथन के दो संस्करण उपलब्ध हैं: 3.x.x और 2.7.10। पायथन दोनों को डाउनलोड करने के लिए उपलब्ध कराता है, लेकिन नए उपयोगकर्ताओं को 3.x.x . चुनना चाहिए।

संस्करण. 2.7.10 डाउनलोड करें यदि आप लीगेसी पायथन कोड के साथ काम करने जा रहे हैं या उन प्रोग्रामों और पुस्तकालयों के साथ जिन्होंने अभी तक 3.x.x को नहीं अपनाया है।

- यह मार्गदर्शिका मान लेगी कि आप 3.xx . इंस्टॉल कर रहे हैं



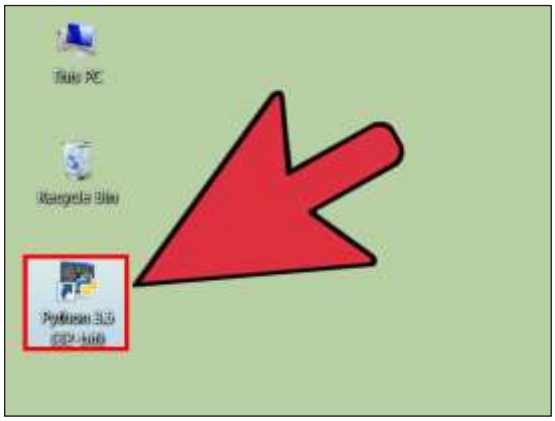
3. इंस्टॉलर को डाउनलोड करने के बाद चलाएं: अपने इच्छित संस्करण के लिए बटन पर क्लिक करने से इसके लिए इंस्टॉलर डाउनलोड हो जाएगा। डाउनलोड होने के बाद इस इंस्टॉलर को चला सकते हैं।



4. "पायथन 3.5 को पाथ में जोड़ें" बॉक्स को चेक करें: यह आपको सीधे कमांड प्रॉम्प्ट से पायथन चलाने की अनुमति देगा।

5. "इंस्टॉल नाउ" पर क्लिक करें: यह अपनी सभी डिफ़ॉल्ट सेटिंग्स के साथ पायथन को इंस्टॉल करेगा, जो कि अधिकांश उपयोगकर्ताओं के लिए ठीक होना चाहिए।

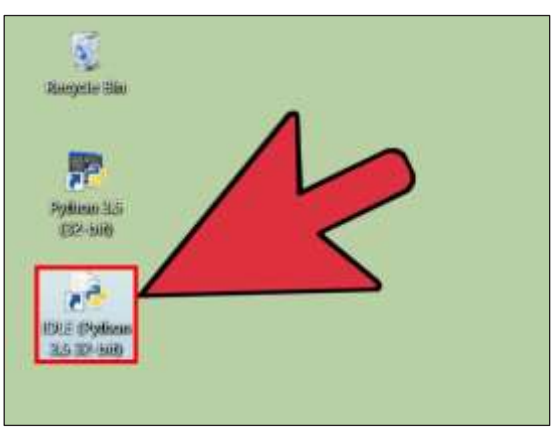
- यदि आप कुछ कार्यों को अक्षम करना चाहते हैं, तो स्थापना निर्देशिका बदलें या डीबगर इंस्टॉल करें, इसके बजाय "कस्टोमाइज इंस्टालेशन" पर क्लिक करें और फिर बक्से को चेक या अनचेक करें।



6. पायथन इंटरप्रेटर खोलें: यह सत्यापित करने के लिए कि पायथन इंस्टॉल है और सही तरीके से काम कर रहा है, नए इंस्टॉल इंटरप्रेटर को खोलें। स्टार्ट बटन पर क्लिक करें और इसे जल्दी से खोलने के लिए "पायथन" टाइप करें।

7. एक परीक्षण स्क्रिप्ट आज़माएं: पायथन एक कमांड लाइन के लिए खुलेगा। निम्न कमांड टाइप करें और "हैलो वर्ल्ड!" प्रदर्शित करने के लिए ↵ एंटर दबाएं। स्क्रीन पर:

दबाएँ "हैलो वर्ल्ड!" प्रदर्शित करने के लिए दर्ज करें स्क्रीन पर: प्रिंट ('हैलो वर्ल्ड!')



8. आईडीएलई विकास परिवेश खोलें: पायथन आईडीएलई नामक एक विकास परिवेश के साथ आता है। यह आपको स्क्रिप्ट चलाने, परीक्षण करने और डीबग करने की अनुमति देता है। आप स्टार्ट मेन्यू खोलकर और "निष्क्रिय" खोजकर जल्दी से आईडीएलई लॉन्च कर सकते हैं।

9. पायथन सीखना जारी रखें: अब जब आपने इंस्टॉल कर लिया है कि पायथन स्थापित है और काम कर रहा है, तो आप इसका उपयोग करना सीखना शुरू कर सकते हैं। पायथन का उपयोग करने के तरीके सीखने के कुछ सुझावों के लिए पायथन में प्रोग्रामिंग कैसे शुरू करें देखें।

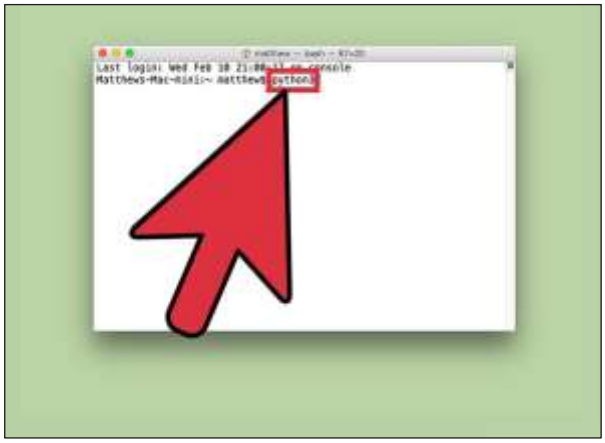
विधि 2: मैक

1. तय करें कि आप पायथन 3.x.x इंस्टोल करना चाहते हैं: ओएस एक्स के सभी संस्करण पहले से स्थापित पायथन 2.7 के साथ आते हैं। यदि आपको पायथन के नए संस्करण की आवश्यकता नहीं है, तो आपको कुछ भी स्थापित करने की आवश्यकता नहीं है। यदि आप पायथन के नवीनतम संस्करणों तक पहुंच चाहते हैं, तो आप 3.x.x इंस्टॉल करना चाहेंगे।

    - यदि आप केवल पायथन के शामिल संस्करण का उपयोग करना चाहते हैं, तो आप एक टेक्स्ट एडिटर में स्क्रिप्ट बना सकते हैं और उन्हें टर्मिनल के माध्यम से चला सकते हैं।

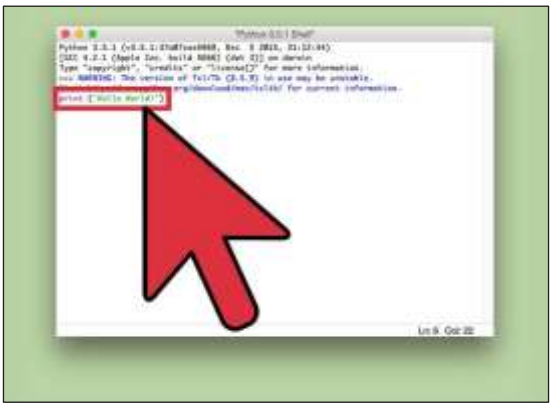
2. पायथन वेबसाइट से पायथन 3.xx फ़ाइलें डाउनलोड करें: अपने मैक पर जाएं (पायथन.org/downloads। इसे आपके ऑपरेटिंग सिस्टम का पता लगाना चाहिए और मैक इंस्टॉलेशन फ़ाइलें दिखानी चाहिए। यदि ऐसा नहीं होता है, तो "मैक ओएस एक्स" लिंक पर क्लिक करें।

3. पायथन को इंस्टॉल करने के लिए डाउनलोड की गई पीकेजी फ़ाइल पर डबल-क्लिक करें: पायथन को इंस्टॉल करने के लिए संकेतों का पालन करें। अधिकांश उपयोगकर्ता केवल डिफ़ॉल्ट सेटिंग्स का उपयोग कर सकते हैं।



4. प्रक्षेपण टर्मिनल में पायथन: यह सत्यापित करने के लिए कि इंस्टॉलेशन ठीक होता है, टर्मिनल लॉन्च करें और पायथन3 टाइप करें। यह पायथन 3.xx इंटरफ़ेस शुरू करना चाहिए और संस्करण प्रदर्शित करना चाहिए।

5. आईडीएलई विकास वातावरण खोलें: यह प्रोग्राम आपको पायथन स्क्रिप्ट लिखने और परीक्षण करने की अनुमति देता है। आप इसे एप्लिकेशन फ़ोल्डर में पा सकते हैं।

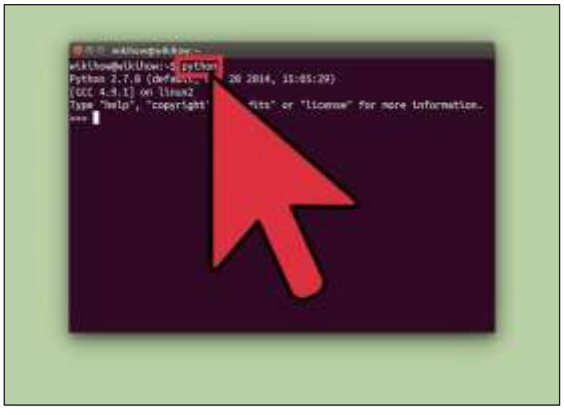


6. एक परीक्षण स्क्रिप्ट आज़माएं: आईडीएलई एक टर्मिनल स्क्रीन के समान वातावरण खोलेगा। निम्न कमांड टाइप करें और "हैलो वर्ल्ड!" प्रदर्शित करने के लिए ↵ एंटर दबाएं:

```
print('Hello world!')
```

7. पायथन का उपयोग करना शुरू करें: अब जब पायथन इंस्टॉल हो गया है, तो आप इसका उपयोग प्रोग्राम करना सीखने के लिए शुरू कर सकते हैं। पायथन शुरुआती के लिए अधिक निर्देशों के लिए पायथन में प्रोग्रामिंग कैसे शुरू करें यह देखें।

तरीका 3: लिनक्स



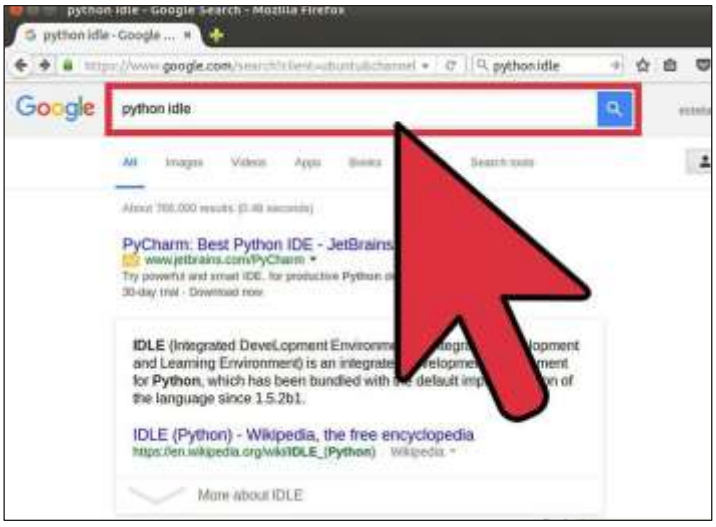
1. आपके द्वारा पहले से इंस्टॉल किए गए पायथन के संस्करण की जाँच करें: लिनक्स का लगभग हर वितरण पायथन स्थापित के साथ आता है। आप टर्मिनल खोलकर और अजगर टाइप करके देख सकते हैं कि आपके पास कौन सा संस्करण होता है।

2. उबंटू में नवीनतम संस्करण इंस्टॉल करें: टर्मिनल विंडो खोलें और सूडो एपीटी-गेट इंस्टोल पायथन टाइप करें।

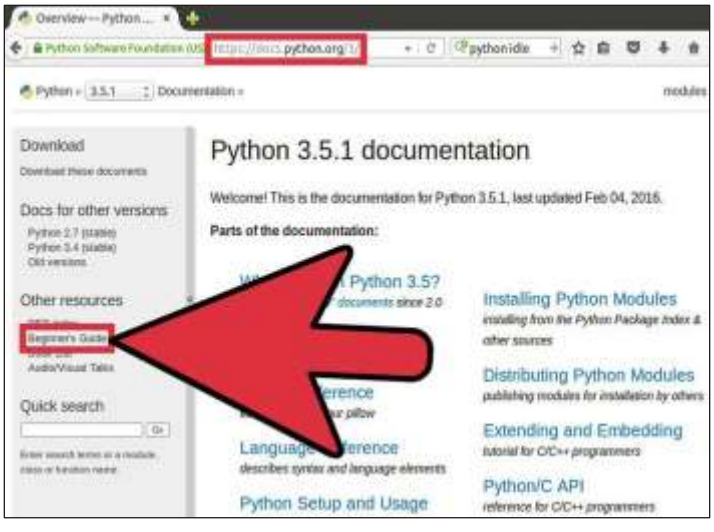
- आप एप्लिकेशन विंडो में स्थित उबंटू के ऐड / रिमूव एप्लिकेशन ऐप का उपयोग करके भी पायथन को इंस्टॉल कर सकते हैं।

3. रेड हेट और फेडोरा में नवीनतम संस्करण इंस्टॉल करें: टर्मिनल विंडो खोलें और सूडो यम इंस्टोल पायथन टाइप करें।

4. आर्क लिनक्स में नवीनतम संस्करण इंस्टॉल करें: रूट उपयोगकर्ता के रूप में लॉग इन करें। पॅकमैन-एस पायथन टाइप करें।



5. आईडीएलई परिवेश डाउनलोड करें: यदि आप पायथन विकास परिवेश का उपयोग करना चाहते हैं, तो आप इसे अपने वितरण के सॉफ़्टवेयर प्रबंधक का उपयोग करके प्राप्त कर सकते हैं। पैकेज को खोजने और इंस्टॉल करने के लिए बस "पायथन आइडल" खोजें।



6. पायथन सीखने में प्रोग्राम कैसे करें: अब जब आपके पास पायथन का नवीनतम संस्करण इंस्टॉल हो गया है, तो आप सीखना शुरू कर सकते हैं कि इसे प्रोग्राम में कैसे उपयोग किया जाए। पायथन सीखने के कुछ सुझावों के लिए पायथन में प्रोग्रामिंग कैसे शुरू करें देखें।

विंडोज़ पर पायथन 2 कैसे इंस्टॉल करें

आप सबसे पहले, इसकी आधिकारिक वेबसाइट से पायथन 2.7 का नवीनतम संस्करण डाउनलोड करें। यदि आप यह सुनिश्चित करना चाहते हैं कि आप पूरी तरह से अप-टू-डेट संस्करण इंस्टॉल कर रहे हैं, तो पायथन.ओआरजी वेब साइट के होम पेज से डाउनलोड> विंडोज लिंक पर क्लिक करें।

विंडोज संस्करण एक एमएसआई पैकेज के रूप में प्रदान किया जाता है। इसे मैन्युअल रूप से स्थापित करने के लिए, बस फ़ाइल पर डबल-क्लिक करें। एमएसआई पैकेज प्रारूप विंडोज प्रशासकों को अपने मानक उपकरणों के साथ स्थापना को स्वचालित करने की अनुमति देता है।

डिज़ाइन के अनुसार, पायथन एक निर्देशिका में इंस्टॉल होता है जिसमें संस्करण संख्या एम्बेडेड होती है, उदाहरण के लिए पायथन संस्करण 2.7 पर इंस्टॉल होगा सी:\पायथन27\, ताकि आप बिना किसी विरोध के एक ही सिस्टम पर पायथन के कई संस्करण रख सकें। बेशक, केवल एक इंटरप्रेटर पायथन फ़ाइल प्रकारों के लिए डिफ़ॉल्ट एप्लिकेशन हो सकता है। यह स्वचालित रूप से पाथ एनवायरनमेंट वेरिएबल को संशोधित नहीं करता है, ताकि आप हमेशा इस पर नियंत्रण रखें कि पायथन की कौन सी प्रति चलाई जाती है।

हर बार पाइथन इंटरप्रेटर के लिए पूर्ण पथ नाम टाइप करना जल्दी से थकाऊ हो जाता है, इसलिए अपने डिफ़ॉल्ट पायथन संस्करण के लिए निर्देशिकाओं को पाथ में जोड़ें। यह मानते हुए कि आपका पायथन इंस्टॉलेशन C:\पायथन27\ में है, इसे अपने पाथ में जोड़ें:

```
C:\Python27\;C:\Python27\Scripts\
```

आप इसे पावरशेल में निम्नलिखित चलाकर आसानी से कर सकते हैं:

```
[Environment]::SetEnvironmentVariable("Path", "$env:Path;C:\Python27\;
C:\Python27\Scripts\", "User")
```

यह भी स्थापना प्रक्रिया के दौरान एक विकल्प है।

दूसरी (Scripts) निर्देशिका को कुछ पैकेज स्थापित होने पर कमांड फाइल प्राप्त होती है, इसलिए यह एक बहुत ही उपयोगी जोड़ है। पायथन का उपयोग करने के लिए आपको कुछ और स्थापित या कॉन्फ़िगर करने की आवश्यकता नहीं है। विशेष रूप से, आपको हमेशा Setuptools स्थापित करना चाहिए, क्योंकि यह आपके लिए अन्य तृतीय-पक्ष पायथन पुस्तकालयों का उपयोग करना बहुत आसान बनाता है।

सेटअपटूल्स + पिप

दो सबसे महत्वपूर्ण तृतीय-पक्ष पायथन पैकेज सेटअपटूल और पीआईपी हैं।

एक बार इंस्टॉल हो जाने पर, आप इसे डाउनलोड कर सकते हैं, एक ही कमांड के साथ किसी भी आज्ञाकारी पायथन सॉफ्टवेयर उत्पाद को इंस्टॉल और अनइंस्टॉल करें। यह आपको इस नेटवर्क स्थापना क्षमता को अपने स्वयं के पायथन सॉफ़्टवेयर में बहुत कम काम के साथ जोड़ने में सक्षम बनाता है।

पायथन 2.7.9 और बाद में (पायथन 2 श्रृंखला पर), और पायथन 3.4 और बाद में डिफ़ॉल्ट रूप से पाइप को इसमें शामिल करें।

यह देखने के लिए कि क्या पाइप इंस्टॉल है, कमांड प्रॉम्प्ट खोलें और इसे चलाएं

```
command -v pip
```

पाइप इंस्टॉल करने के लिए, आधिकारिक पाइप इंस्टालेशन गाइड का पालन करें - यह स्वचालित रूप से सेटअपटूल का नवीनतम संस्करण इंस्टॉल करेगा।

वर्चुअल एनवायरनमेंट

वर्चुअल एनवायरनमेंट विभिन्न प्रोजेक्ट्स के लिए आवश्यक डिपेंडेंसीज़ को अलग-अलग जगहों पर रखने के लिए, उनके लिए वर्चुअल पायथन वातावरण बनाकर एक उपकरण होता है। यह "प्रोजेक्ट एक्स संस्करण 1.x पर निर्भर करता है, लेकिन प्रोजेक्ट वाई को 4.x" की जरूरत होती है और आपकी वैश्विक साइट-पैकेज निर्देशिका को साफ और प्रबंधनीय रखता है।

विंडोज़ पर पायथन 3 कैसे इंस्टॉल करें

सबसे पहले, चॉकलेटी के लिए इंस्टॉलेशन निर्देशों का पालन करें। यह विंडोज 7+ के लिए एक सामुदायिक सिस्टम पैकेजर मैनेजर है। (यह ओएस एक्स पर होमब्रेव की तरह है।)

एक बार हो जाने के बाद, पायथन 3 को स्थापित करना बहुत सरल है, क्योंकि चॉकलेटी पायथन 3 को डिफ़ॉल्ट के रूप में धकेलता है।

```
choco install python
```

एक बार जब आप इस कमांड को चला लेते हैं, तो आपको सीधे कंसोल से पायथन लॉन्च करने में सक्षम होना चाहिए। (चॉकलेट शानदार है और स्वचालित रूप से आपके पथ में पायथन जोड़ता है।)

सेटअपटूल्स + पिप

दो सबसे महत्वपूर्ण तृतीय-पक्ष पायथन पैकेज सेटअपटूल और पाइप हैं, जो आपको एक ही कमांड के साथ किसी भी आज्ञाकारी पायथन सॉफ्टवेयर उत्पाद को डाउनलोड, इंस्टॉल और अनइंस्टॉल करने देते हैं। यह आपको इस नेटवर्क स्थापना क्षमता को अपने स्वयं के पायथन सॉफ़्टवेयर में बहुत कम काम के साथ जोड़ने में सक्षम बनाता है।

पायथन 3 के सभी समर्थित संस्करणों में पाइप शामिल है, इसलिए सुनिश्चित करें कि यह अद्यतित है:

```
python -m pip install -U pip
```

पिपेनव और वर्चुअल एनवायरनमेंट

अगला कदम पिपेनव को इंस्टॉल करना है, ताकि आप निर्भरता इंस्टॉल कर सकें और आभासी वातावरण का प्रबंधन कर सकें।

वर्चुअल एनवायरनमेंट विभिन्न प्रोजेक्ट्स के लिए आवश्यक डिपेंडेंसीज़ को अलग-अलग जगहों पर रखने के लिए, उनके लिए वर्चुअल पायथन वातावरण बनाकर एक उपकरण है। यह "प्रोजेक्ट एक्स संस्करण 1.x पर निर्भर करता है, लेकिन प्रोजेक्ट वाई को 4.x" की जरूरत है, और आपकी वैश्विक साइट-पैकेज निर्देशिका को साफ और प्रबंधनीय रखता है।

मैक पर पायथन 2 कैसे इंस्टॉल करें

पायथन को इंस्टॉल करने से पहले, आपको एक सी कंपाइलर इंस्टॉल करना होगा। इसका सबसे तेज़ तरीका `xcode-select --install` चलाकर एक्स कोड कमांड लाइन टूल्स को इंस्टॉल करना है। आप मैक एप स्टोर से एक्सकोड का पूर्ण संस्करण, या न्यूनतम लेकिन अनौपचारिक OSX-GCC-Installerpack-age से भी डाउनलोड कर सकते हैं।

यदि आपके पास पहले से ही एक्सकोड इंस्टॉल है, तो OSX-GCC-Installer इंस्टॉल न करें। संयोजन में, सॉफ़्टवेयर उन समस्याओं का कारण बन सकता है जिनका निदान करना मुश्किल है।

यदि आप एक्सकोड की एक नई स्थापना करते हैं, तो आपको टर्मिनल पर `xcode-select --install` चलाकर कमांडलाइन टूल भी जोड़ने होंगे।

जबकि ओएस एक्स बड़ी संख्या में यूनिक्स उपयोगिताओं के साथ आता है, लिनक्स सिस्टम से परिचित लोगों को एक प्रमुख घटक गायब दिखाई देगा: एक सभ्य पैकेज प्रबंधक। होमेब्रेव इस शून्य को भरता है।

होमेब्रेव इंस्टॉल करने के लिए, खोलें टर्मिनल या आपका पसंदीदा ओएस एक्स टर्मिनल एमुलेटर और चलाएं

```
$ /usr/bin/ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Home-
brew/install/master/install)"
```

स्क्रिप्ट बताएगी कि यह क्या बदलाव करेगा और इंस्टॉलेशन शुरू होने से पहले आपको संकेत देगा। होमेब्रेव स्थापित करने के बाद, अपने PATH एनवायरनमेंट वेरिएबल के शीर्ष पर होमेब्रेव निर्देशिका डालें। आप अपनी `~/.profile` फ़ाइल के नीचे निम्न पंक्ति जोड़कर ऐसा कर सकते हैं

```
export PATH="/usr/local/bin:/usr/local/sbin:$PATH"
```

अब, हम कर सकते हैं पायथन 2.7 इंस्टॉल करें:

```
$ brew install python@2
```

क्योंकि paythan@2 हैएक "केग", हमें अपना अद्यतन करने की आवश्यकता है पथ फिर से, हमारी नई स्थापना को इंगित करने के लिए:

```
export PATH="/usr/local/opt/python@2/libexec/bin:$PATH"
```

होमेब्रेव निष्पादन योग्य नाम देता है को पायथन2 ताकि आप अभी भी कर सकें निष्पादन योग्य के माध्यम से सिस्टम पायथन चलाएं अजगर.

```
$ python -V # होमेब्रेव installed Python 3 interpreter (if installed)

$ python2 -V # होमेब्रेव installed Python 2 interpreter

# $ python3 -V # होमेब्रेव installed Python 3 interpreter (if installed)
```

सेटअपटूल एंड पिप

होमेब्रेव आपके लिए सेटअपटूल और पिप इंस्टॉल करता है।

सेटअपटूल आपको एक ही कमांड (`easy_install`) के साथ नेटवर्क (आमतौर पर इंटरनेट) पर किसी भी आज्ञाकारी पायथन सॉफ्टवेयर को डाउनलोड और इंस्टॉल करने में सक्षम बनाता है। यह आपको इस नेटवर्क स्थापना क्षमता को अपने स्वयं के पायथन सॉफ़्टवेयर में बहुत कम काम के साथ जोड़ने में सक्षम बनाता है।

रंज पायथन पैकेज को आसानी से इंस्टॉल करने और प्रबंधित करने के लिए एक उपकरण है, जिसकी सिफारिश ऊपर की गई है आसान_ इंस्टॉल करें। यह easy_install से बेहतर हैकई तरीकों से, और सक्रिय रूप से बनाए रखा जाता है।

```
$ pip2 -V # pip pointing to the Homebrew installed Python 2
interpreter

$ pip -V # pip pointing to the Homebrew installed Python 3
interpreter (if installed)
```

वर्चुअल एनवायरनमेंट

एक वर्चुअल एनवायरनमेंट (आमतौर पर 'वर्चुअलएन्व' के रूप में जाना जाता है) अलग-अलग जगहों पर अलग-अलग प्रोजेक्ट्स के लिए आवश्यक डिपेंडेंसीज को बनाए रखने के लिए वर्चुअल पायथन वातावरण बनाने का एक उपकरण है। यह "प्रोजेक्ट एक्स संस्करण 1.x पर निर्भर करता है, लेकिन प्रोजेक्ट वाई को 4.x" की जरूरत है और आपकी वैश्विक साइट-पैकेज निर्देशिका को साफ और प्रबंधनीय रखता है।

मैक पर पायथन 3 कैसे इंस्टॉल करें

पायथन को इंस्टॉल करने से पहले, आपको जीसीसी इंस्टॉल करना होगा। जीसीसी को एक्सकोड, छोटे कमांड लाइन टूल्स (एक ऐप्पल खाता होना चाहिए) या उससे भी छोटा ओएसएक्स-जीसीसी-इन-स्टालर पैकेज डाउनलोड करके प्राप्त किया जा सकता है।

यदि आपके पास पहले से ही एक्सकोड इंस्टॉल है, तो `OSX-GCC-Installer` इंस्टॉल न करें। इसके संयोजन में सॉफ़्टवेयर उन समस्याओं का कारण बन सकता है जिनका निदान करना मुश्किल है।

यदि आप एक्सकोड की एक नई स्थापना करते हैं, तो आपको टर्मिनल पर xcode-select --install चलाकर कमांडलाइन टूल भी जोड़ने होंगे।

जबकि ओएस एक्स बड़ी संख्या में यूनिक्स उपयोगिताओं के साथ आता है, लिनक्स सिस्टम से परिचित लोगों को एक प्रमुख घटक गायब दिखाई देगा: एक पैकेज मैनेजर। होमेब्रेव इस शून्य को भरता है।

होमेब्रेव इंस्टॉल करने के लिए, खोलें टर्मिनल या आपका पसंदीदा ओएस एक्स टर्मिनल एमुलेटर और चलाएं

```
$ ruby -e "$(curl -fsSL https://raw.githubusercontent.com/होमेब्रेव/in-
stall/master/install)"
```

स्क्रिप्ट बताएगी कि यह क्या बदलाव करेगा और इंस्टॉलेशन शुरू होने से पहले आपको संकेत देगा। होमेब्रेव स्थापित करने के बाद, अपने `PATH` environment वेरिएबल के शीर्ष पर होमेब्रेव निर्देशिका डालें। आप अपनी ~/.profile फ़ाइल के नीचे निम्न पंक्ति जोड़कर ऐसा कर सकते हैं

```
export PATH="/usr/local/opt/ python /libexec/bin:$PATH"
```

If you have OS X 10.12 (Sierra) or older use this line instead

```
export PATH=/usr/local/bin:/usr/local/sbin:$PATH
```

हम पायथन 3 इंस्टॉल कर सकते हैं:

```
$ brew install python
```

इसमें एक या दो मिनट का समय लगेगा।

पिप

होमेब्रेव आपके लिए होमेब्रेव'ड पायथन 3 की ओर इशारा करते हुए पाइप इंस्टॉल करता है।

पायथन 3 से काम करना

इस बिंदु पर, आपके पास सिस्टम पायथन 2.7 उपलब्ध है, संभावित रूप से पायथन 2 का होमेब्रेव संस्करण इंस्टॉल है, और पायथन 3 का होमेब्रेव संस्करण भी है।

```
$ python
```

लॉन्च करेंगे होमेब्रेव-इंस्टॉल पायथन 3 इंटरप्रेटर।

```
$ python2
```

लॉन्च करेंगे होमेब्रेव-इंस्टॉल पायथन 2 इंटरप्रेटर (यदि कोई हो)।

```
$ python3
```

लॉन्च करेंगे होमेब्रेव-इंस्टॉल पायथन 3 इंटरप्रेटर।

यदि पायथन 2 का होमेब्रेव संस्करण इंस्टॉल है तो pip2 पायथन 2 को इंगित करेगा। यदि पायथन 3 का होमेब्रेव संस्करण इंस्टॉल है, तो पिप पायथन 3 को इंगित करेगा।

बाकी गाइड मान लेंगे कि अजगर पायथन 3 का संदर्भ देता है।

```
# Do I have a Python 3 installed?
$ Python --version
Python 3.7.1 # Success!
```

पिपेनव एंड वर्चुअल एनवायरनमेंट

अगला कदम पिपेनव को इंस्टॉल करना है, ताकि आप निर्भरता इंस्टॉल कर सकें और आभासी वातावरण का प्रबंधन कर सकें।

वर्चुअल एनवायरनमेंट विभिन्न प्रोजेक्ट्स के लिए आवश्यक डिपेंडेंसीज़ को अलग-अलग जगहों पर रखने के लिए, उनके लिए वर्चुअल पायथन वातावरण बनाकर एक उपकरण है। यह "प्रोजेक्ट एक्स संस्करण 1.x पर निर्भर करता है, लेकिन प्रोजेक्ट वाई को 4.x" की जरूरत है, और आपकी वैश्विक साइट-पैकेज निर्देशिका को साफ और प्रबंधनीय रखता है।

पायथन 2 लिनक्स कैसे इंस्टॉल करें

सेंट एस, रेडहैट इंटरपरिसे (आरएचईएल) और यूबंटू के नवीनतम संस्करण पायथन के साथ आते हैं 2.7 बॉक्स से बाहर।

यह देखने के लिए कि आपने पायथन का कौन सा संस्करण इंस्टॉल किया है, कमांड प्रॉम्प्ट खोलें और चलाएँ:

```
$ python 2 --version
```

हालांकि, पायथन 3 की बढ़ती लोकप्रियता के साथ, कुछ वितरण, जैसे कि फेडोरा, पहले से इंस्टॉल पायथन 2 के साथ नहीं आते हैं। आप अपने वितरण पैकेज प्रबंधक के साथ पायथन2 पैकेज इंस्टॉल कर सकते हैं:

```
$ sudo dnf install python2
```

पायथन का उपयोग करने के लिए आपको कुछ और इंस्टॉल या कॉन्फ़िगर करने की आवश्यकता नहीं है। यह कहने के बाद। विशेष रूप से, आपको हमेशा सेटअपटूल और पिप इंस्टॉल करना चाहिए, क्योंकि यह आपके लिए अन्य तृतीय-पक्ष पायथन पुस्तकालयों का उपयोग करना बहुत आसान बनाता है।

सेटअपटूल और पिप

दो सबसे महत्वपूर्ण तृतीय-पक्ष पायथन पैकेज सेटअपटूल और पीआईपी हैं।

एक बार इंस्टॉल हो जाने पर, आप डाउनलोड कर सकते हैं, एक ही कमांड के साथ किसी भी आज्ञाकारी पायथन सॉफ्टवेयर उत्पाद को इंस्टॉल और अनइंस्टॉल करें। यह आपको इस नेटवर्क स्थापना क्षमता को अपने स्वयं के पायथन सॉफ़्टवेयर में बहुत कम काम के साथ जोड़ने में सक्षम बनाता है।

पायथन 2.7.9 और बाद में (पायथन 2 श्रृंखला पर), और पायथन 3.4 और बाद में डिफ़ॉल्ट रूप से पाइप शामिल करें। यह देखने के लिए कि क्या पाइप

इंस्टॉल है, कमांड प्रॉम्प्ट खोलें और चलाएं

```
$ command -v pip
```

पाइप इंस्टॉल करने के लिए, आधिकारिक पाइप स्थापना मार्गदर्शिका का पालन करें - यह स्वचालित रूप से सेटअपटूल का नवीनतम संस्करण इंस्टॉल करेगा।

वर्चुअल एनवायरनमेंट

वर्चुअल एनवायरनमेंट विभिन्न प्रोजेक्ट्स के लिए आवश्यक डिपेंडेंसीज़ को अलग-अलग जगहों पर रखने के लिए, उनके लिए वर्चुअल पायथन वातावरण बनाकर एक उपकरण है। यह "प्रोजेक्ट एक्स संस्करण 1.x पर निर्भर करता है, लेकिन प्रोजेक्ट वाई को 4.x" की जरूरत है, और आपकी वैश्विक साइट-पैकेज निर्देशिका को साफ और प्रबंधनीय रखता है।

2

पायथन में प्रवाह नियंत्रण और कार्य

प्रवाह नियंत्रण सशर्त बयान, लूप और फ़ंक्शन कॉल शामिल होते हैं। पायथन में विभिन्न लूप और फ़ंक्शंस में से कुछ हैं, इफ़-एल्स स्टेटमेंट, लूप के लिए है जबकि लूप, बिल्ट-इन फ़ंक्शंस, यूज़र-डिफ़ाइंड फ़ंक्शंस, इत्यादि के लिए इस्तेमाल किया जाता है। इन प्रवाह नियंत्रण की आसान समझ प्रदान करने के लिए इस अध्याय को सावधानीपूर्वक लिखा गया है और ये पायथन में कार्य करता है।

इफ़-एल्स स्टेटमेंट का इस्तेमाल कैसे करें

निर्णय लेने की आवश्यकता तब होती है जब हम किसी कोड को केवल तभी निष्पादित करना चाहते हैं जब एक निश्चित शर्त पूरी हो। इस निर्णय लेने के लिए

पायथन में `if…elif…else` स्टेटमेंट का उपयोग किया जाता है।

अगर पायथन स्टेटमेंट सिंटैक्स है, तब

```
If test expression: statement
        (s)
```

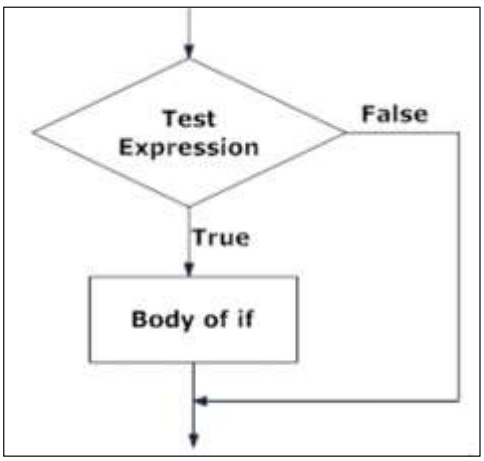
यहां, प्रोग्राम परीक्षण अभिव्यक्ति का मूल्यांकन करता है और यह केवल तभी कथन निष्पादित करता है जब टेक्स्ट एक्सप्रेशन सही होता है।

यदि टेक्स्ट एक्सप्रेशन गलत है, तो स्टेटमेंट निष्पादित नहीं होते हैं।

पायथन में इफ स्टेटमेंट का मुख्य भाग इंडेंटेशन द्वारा इंगित किया जाता है। बॉडी इंडेंटेशन से शुरू होती है और पहली अनइंडेंट लाइन अंत को चिह्नित करती है।

पायथन गैर-शून्य मानों की व्याख्या करता है `true`, `None` और 0 के रूप में व्याख्या की जाती है false.

पायथन स्टेटमेंट फ़्लोचार्ट



कार्यवाही अगर बयान का।

उदाहरण: पायथन अगर स्टेटमेंट है

```
# If the number is positive, we print an appropriate message
num = 3
if num > 0:
      print(num, "is a positive number.")
print("This is always printed.")
num = -1
if num > 0:
print(num, "is a positive number.")
print("This is also always printed.") जब
```

आप प्रोग्राम चलाते हैं, तो आउटपुट होगा:

```
3 is a positive number
This is always printed
This is also always printed.
```

उपरोक्त उदाहरण में, `num > 0` परीक्षण व्यंजक है। if का मुख्य भाग तभी निष्पादित होत

है जब यह `True` का मूल्यांकन करता है।

जब वेरिएबल संख्या 3 के बराबर होती है, तो परीक्षण अभिव्यक्ति सत्य होती है और अगर के शरीर के अंदर का शरीर निष्पादित होता है।

यदि वेरिएबल संख्या -1 के बराबर है, तो परीक्षण अभिव्यक्ति गलत है और यदि शरीर के अंदर का शरीर छोड़ दिया गया है।

प्रिंट () स्टेटमेंट इफ ब्लॉक (अनइंडेंटेड) के बाहर आता है। इसलिए, इसे परीक्षण अभिव्यक्ति की परवाह किए बिना निष्पादित किया जाता है।

पायथन इफ़...एल्स स्टेटमेंट

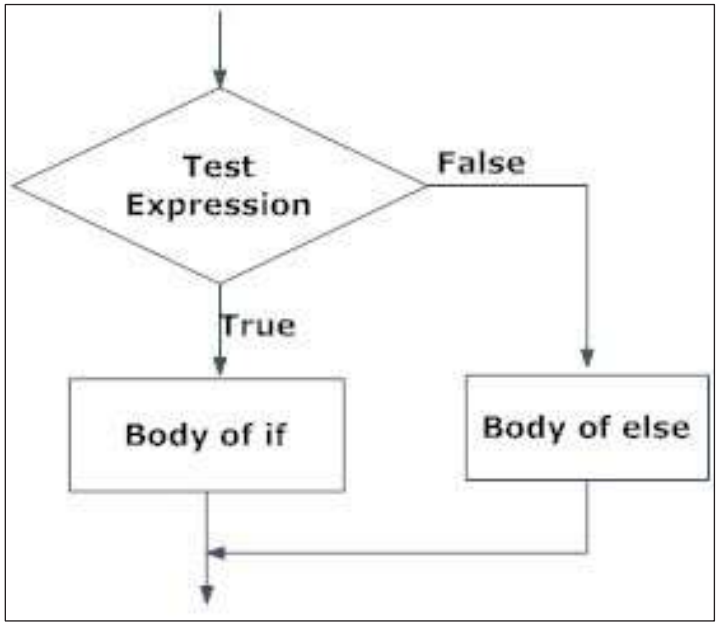
सिंटैक्स इफ़-...एल्स

```
if test expression:
      Body of if
else:
      Body of else
```

if..`else` स्टेटमेंट टेस्ट एक्सप्रेशन का मूल्यांकन करता है और अगर टेस्ट कंडीशन ट्रू होने पर ही बॉडी को एक्सीक्यूट करता है।

इसमें शर्त है अगर `false`, की बॉडी निष्पादित किया जाता है। इंडेंटेशन का उपयोग ब्लॉक को अलग करने के लिए किया जाता है।

पायथन इफ़..एल्स फ़्लोचार्ट



कार्यवाही पायथन का अगर.. और फ़्लोचार्ट।

if...एल्स (इफ़ एल्स) का उदाहरण:

```
# Program checks if the number is positive or negative
# And displays an appropriate message
num = 3
# Try these two variations as well.
# num = -5
# num = 0
if num >= 0:
      print("Positive or Zero")
else:
      print("Negative number")
```

उपरोक्त उदाहरण में, जब संख्या 3 के बराबर होती है, तो परीक्षण अभिव्यक्ति सत्य होती है और इफ़ का शरीर निष्पादित होता है और अन्य का एल्स छोड़ दिया जाता है।

अगर अंक -5 के बराबर है, परीक्षण अभिव्यक्ति झूठी है और अन्य के शरीर को निष्पादित किया जाता है और यदि का शरीर छोड़ दिया जाता है।

अगर अंक 0 के बराबर है, परीक्षण अभिव्यक्ति सत्य है और अगर का शरीर निष्पादित किया जाता है और अन्य का शरीर छोड़ दिया जाता है।

पायथन if...elif...else (इफ़...एलिफ...एल्स) स्टेटमेंट

if...elif...else (इफ़...एलिफ...एल्स) का सिंटैक्स

```
if test expression:
      Body of if
elif test expression:
      Body of elif
else:
      Body of else
```

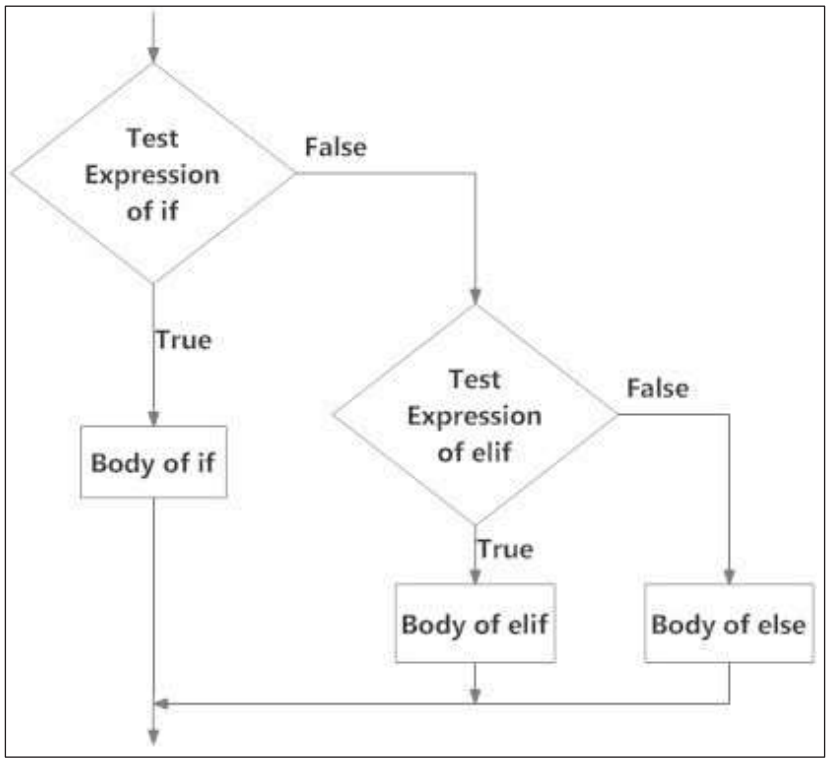
एलिफ और एल्स के लिए छोटा है। यह हमें कई अभिव्यक्तियों की जांच करने की अनुमति देता है।

यदि शर्त अगर है झूठा, यह अगले एलिफ ब्लॉक वगैरह की स्थिति की जांच करता है। यदि सभी शर्तें फाल्स हैं, तो अन्य के बॉडी को निष्पादित किया जाता है।

कई में से केवल एक ब्लॉक `if...elif...else` ब्लॉक को स्थिति के अनुसार निष्पादित किया जाता है।

if ब्लॉक में केवल एक और ब्लॉक हो सकता है। लेकिन इसमें कई एलिफ ब्लॉक हो सकते हैं।

if...elif...else (इफ़...एलीफ...एल्स) का फ़्लोचार्ट



इफ़...एलिफ...एल्स का ऑपरेशन स्टेटमेंट।

if...elif...else (इफ़...एलिफ...एल्स) के उदाहरण

```
# In this program,
# we check if the number is positive or
# negative or zero and
# display an appropriate message
num = 3.4
# Try these two variations as well:
# num = 0
# num = -4.5
if num > 0:
      print("Positive number")
elif num == 0:
      print("Zero")
else:
      print("Negative number")
```

जब वेरिएबल अंक पॉजिटिव है, तब यह पॉजिटिव संख्या को छापता है।

अगर अंक 0 के बराबर है तो यह जीरो लिखता है।

अगर अंक नेगेटिव है तो यह नेगेटिव संख्या को लिखता है।

पायथन नेस्टेड इफ़ के उदाहरण

हमारे पास एक इफ़...एलिफ...एल्स स्टेटमेंट दूसरे के अंदर हो सकता है इफ़...एलिफ...एल्स स्टेटमेंट है। इसे कंप्यूटर प्रोग्रामिंग में नेस्टिंग भी कहा जाता है।

इनमें से किसी भी कथन को एक दूसरे के अंदर नेस्टेड बनाया जा सकता है। नेस्टिंग के स्तर का पता लगाने का एकमात्र तरीका इंडेंटेशन है। यह भ्रमित करने वाला हो सकता है, इसलिए यदि हम कर सकते हैं तो इससे बचना चाहिए।

पायथन नेस्टेड इफ़ के उदाहरण

```
# In this program, we input a number
# check if the number is positive or
# negative or zero and display
# an appropriate message
# This time we use nested if
```

```
num = float(input("Enter a number: "))
if num >= 0:
      if num == 0:
      print("Zero")
else:
      print("Positive number")
else:
      print("Negative number")
```

आउटपुट 1

```
Enter a number: 5
Positive number
```

आउटपुट 2

```
Enter a number: -1
Negative number
```

आउटपुट 3

```
Enter a number: 0
Zero
```

लूप के लिए उपयोग कैसे करें

पायथन में लूप के लिए एक अनुक्रम (सूची, टपल, स्ट्रिंग) या अन्य पुनरावृत्त वस्तुओं पर पुनरावृति करने के लिए उपयोग किया जाता है। एक अनुक्रम पर पुनरावृति को ट्रैवर्सल कहा जाता है।

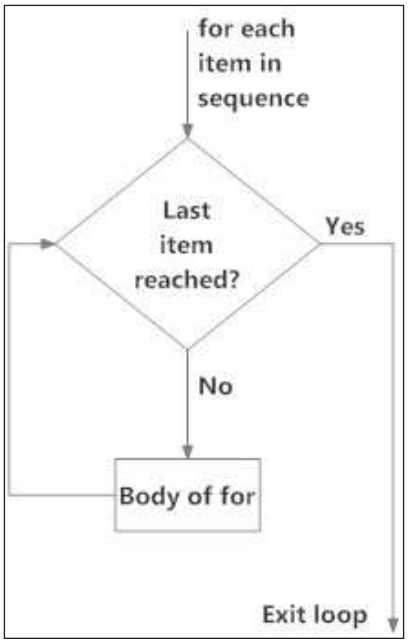
लूप के लिए सिंटेक्स

```
for val in sequence:
      Body of for
```

यहां, वैल वह वेरिएबल है जो प्रत्येक पुनरावृत्ति पर अनुक्रम के अंदर आइटम का समझ लेता है।

यह लूप तब तक जारी रहता है जब तक हम अनुक्रम में अंतिम आइटम तक नहीं पहुंच जाते। इंडेंटेशन का उपयोग करके लूप के शरीर को शेष कोड से अलग किया जाता है।

लूप के लिए फ़्लोचार्ट



लूप के लिए संचालन।

उदाहरण: लूप के लिए पायथन।

```
# Program to find the sum of all numbers stored in a list
# List of numbers
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]
# variable to store the sum
sum = 0
# iterate over the list
for val in numbers:
      sum = sum+val
# Output: The sum is 48
print("The sum is", sum)
```

जब आप प्रोग्राम चलाते हैं, तो आउटपुट होगा:

```
The sum is 48
```

रेंज () फ़ंक्शन

हम रेंज () फ़ंक्शन का उपयोग करके संख्याओं का एक क्रम उत्पन्न कर सकते हैं। रेंज (10) 0 से 9 (10 नंबर) तक की संख्याएं उत्पन्न करता है।

हम स्टार्ट, स्टॉप और स्टेप साइज को `range(start,stop,step size)` के रूप में भी परिभाषित कर सकते हैं। स्टेप साइज़ यदि प्रदान नहीं किया गया है तो यह डिफ़ॉल्ट रूप से 1 हो जाता है।

यह फ़ंक्शन सभी मानों को मेमोरी में संग्रहीत नहीं करता है, यह अक्षम होता है। तो यह स्टार्ट, स्टॉप, स्टेप साइज को याद रखता है और चलते-फिरते अगला नंबर जेनरेट करता है।

इस फ़ंक्शन को सभी वस्तुओं को आउटपुट करने के लिए बाध्य करने के लिए, हम फ़ंक्शन सूची () का उपयोग कर सकते हैं।

निम्नलिखित उदाहरण इसे अधिक स्पष्ट करता है:

```
# Output: range(0, 10)
print(range(10))
# Output: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(list(range(10)))
# Output: [2, 3, 4, 5, 6, 7]
print(list(range(2, 8)))
# Output: [2, 5, 8, 11, 14, 17]
print(list(range(2, 20, 3)))
```

हम लूप के लिए रेंज () फ़ंक्शन का उपयोग संख्याओं के अनुक्रम के माध्यम से पुनरावृति करने के लिए कर सकते हैं। अनुक्रमण का उपयोग करते हुए अनुक्रम के बावजूद इसे पुनरावृति करने के लिए लेन () फ़ंक्शन के साथ जोड़ा जा सकता है। यहां एक उदाहरण है।

```
# Program to iterate through a list using indexing
genre = [‘pop’, ‘rock’, ‘jazz’]
# iterate over the list using index
for i in range(len(genre)):
      print(“I like”, genre[i])
```

जब आप प्रोग्राम चलाते हैं, तो आउटपुट होगा:

```
I like pop
I like rock
I like jazz
```

लूप के लिए एल्स के साथ

लूप के लिए एक वैकल्पिक अन्य ब्लॉक भी हो सकता है। अन्य भाग निष्पादित किया जाता है यदि अनुक्रम में आइटम लूप निकास के लिए उपयोग किए जाते हैं।

लूप को रोकने के लिए ब्रेक स्टेटमेंट का उपयोग किया जा सकता है। ऐसे मामले में, अन्य भाग की उपेक्षा की जाती है।

इसलिए, यदि कोई ब्रेक नहीं होता है तो यह लूप के अन्य भाग के लिए चलता है।

इसे स्पष्ट करने के लिए यहां एक उदाहरण दिया गया है।

```
digits = [0, 1, 5]
for i in digits:
      print(i)
else:
      print("No items left.")
```

जब आप प्रोग्राम चलाते हैं, तो आउटपुट यह होगा:

```
0
1
5
No items left.
```

यहां, लूप के लिए सूची के आइटम तब तक प्रिंट करता है जब तक कि लूप समाप्त नहीं हो जाता। जब लूप समाप्त हो जाता है, तो यह कोड के ब्लॉक को अन्य में निष्पादित करता है और प्रिंट करता है:

```
No items left.
```

लूप के दौरान उपयोग कैसे करें

जब तक परीक्षण अभिव्यक्ति (स्थिति) सही है, तब तक पायथन में लूप का उपयोग कोड के एक ब्लॉक पर पुनरावृति करने के लिए किया जाता है।

हम आम तौर पर इस लूप का उपयोग तब करते हैं जब हम पहले से नहीं जानते कि कितनी बार पुनरावृति करनी है।

पाइथन में लूप के समय का सिंटैक्स

```
while test_expression:
Body of while
```

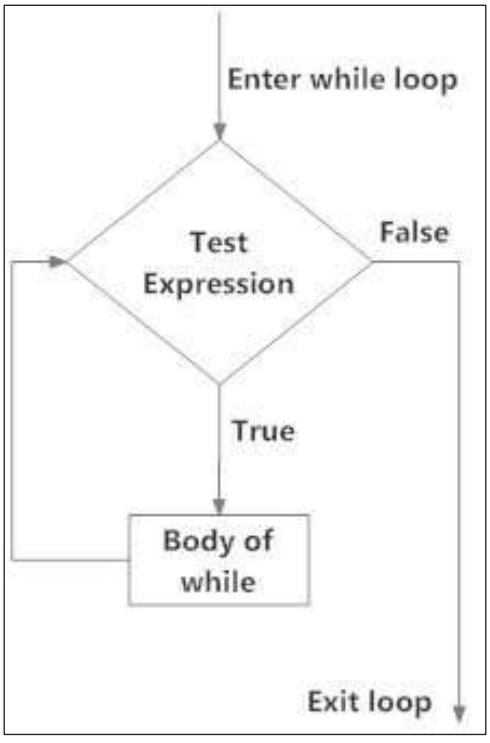
जबकि लूप में पहले टेस्ट एक्सप्रेशन चेक किया जाता है। लूप का मुख्य भाग तभी दर्ज किया जाता है जब टेस्ट_एक्स- अभिव्यक्ति का मूल्यांकन करता है ट्रू। एक पुनरावृत्ति के बाद, परीक्षण अभिव्यक्ति की फिर से जाँच की जाती है। यह प्रक्रिया तब तक जारी रहती है जब तक कि टेस्ट_एक्सप्रेशन का मूल्यांकन गलत नहीं हो जाता।

पायथन में, जबकि लूप का शरीर इंडेंटेशन के माध्यम से निर्धारित किया जाता है।

बॉडी इंडेंटेशन से शुरू होती है और पहली अनइंडेंटेड लाइन अंत को चिह्नित करती है।

पायथन किसी भी गैर-शून्य मान को सत्य के रूप में व्याख्या करता है। कोई नहीं और 0 की व्याख्या false के रूप में की जाती है।

व्हाइल लूप का फ़्लोचार्ट



कार्यवाही जबकि लूप।

उदाहरण: पायथन व्हाइल लूप।

```
# Program to add natural
# numbers upto
# sum = 1+2+3+...+n
# To take input from the user,
# n = int(input("Enter n: "))
n = 10
# initialize sum and counter
sum = 0
i = 1
while i <= n:
      sum = sum + i
      i = i+1 # update counter
# print the sum
```

```
print("The sum is", sum)
```

जब आप प्रोग्राम चलाते हैं, तो आउटपुट यह होता है:

```
Enter n: 10

The sum is 55
```

उपरोक्त कार्यक्रम में, परीक्षण अभिव्यक्ति तब तक सत्य होगी जब तक कि हमारा काउंटर वेरिएबल i से कम है या n के बराबर (हमारे कार्यक्रम में 10) होता है।

हमें लूप की बॉडी में काउंटर वेरिएबल के मान को बढ़ाने की जरूरत होती है। यह बहुत महत्वपूर्ण है (और ज्यादातर भुला दिया गया)। ऐसा करने में विफल होने पर एक अनंत लूप (कभी न खत्म होने वाला लूप) बन जाता है।

आखिरकार परिणाम प्रदर्शित होता है।

एल्स के साथ व्हाइल लूप

लूप के समान ही, हमारे पास लूप के साथ एक वैकल्पिक अन्य ब्लॉक भी हो सकता है।

एल्स भाग को निष्पादित किया जाता है यदि लूप में स्थिति फाल्स का मूल्यांकन करती है।

जबकि लूप को ब्रेक स्टेटमेंट के साथ समाप्त किया जा सकता है। ऐसे मामले में एल्स भाग की उपेक्षा की जाती है। इसलिए थोड़ी देर के लूप अन्यथा अंश यदि कोई विराम नहीं होता है तो चलता है और स्थिति फाल्स है।

इसे स्पष्ट करने के लिए यहां एक उदाहरण दिया गया है।

```
# Example to illustrate
# the use of else statement
# with the while loop
counter = 0
while counter < 3:
      print("Inside loop")
      counter = counter + 1
else:
      print("Inside else")
```

आउटपुट

```
Inside loop
Inside loop
Inside loop
Inside loop
```

यहां, हम स्ट्रिंग इनसाइड लूप को तीन बार प्रिंट करने के लिए काउंटर वेरिएबल का उपयोग करते हैं।

आगे पुनरावृत्ति, जबकि स्थिति बन जाती है `false` इसलिए अन्यथा भाग निष्पादित किया जाता है।

ब्रेक और कंटिन्यू का उपयोग कैसे करें

पायथन में बयानों को तोड़ना और जारी रखना सामान्य लूप के प्रवाह को बदल सकता है।

लूप्स कोड के एक ब्लॉक पर पुनरावृति करते हैं जब तक कि परीक्षण अभिव्यक्ति फाल्स न हो, लेकिन कभी-कभी हम परीक्षण अभिव्यक्ति की जांच किए बिना वर्तमान पुनरावृत्ति या यहां तक कि पूरे लूप को समाप्त करना चाहते हैं।

इन मामलों में ब्रेक एंड कंटिन्यू स्टेटमेंट का उपयोग किया जाता है।

पायथन ब्रेक स्टेटमेंट

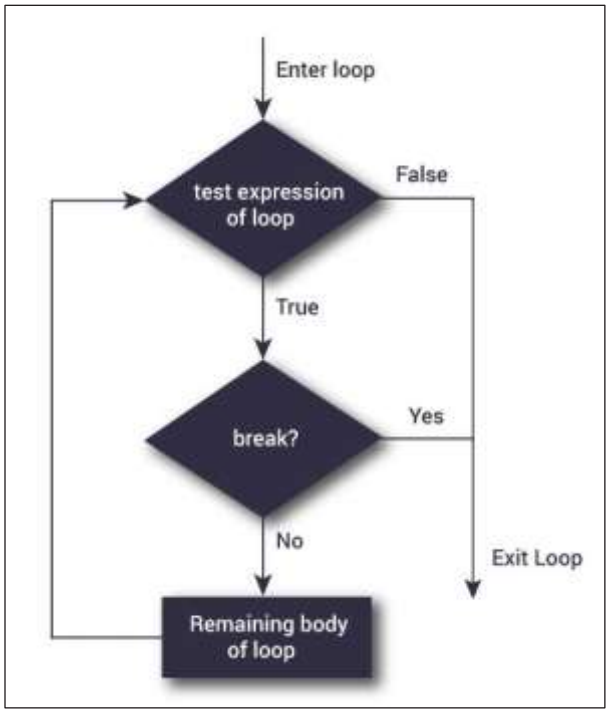
ब्रेक स्टेटमेंट उस लूप को समाप्त कर देता है जिसमें यह होता है। कार्यक्रम का नियंत्रण लूप की बॉडी के तुरंत बाद स्टेटमेंट में प्रवाहित होता है।

यदि ब्रेक स्टेटमेंट नेस्टेड लूप (दूसरे लूप के अंदर लूप) के अंदर है, तो ब्रेक अंतरतम लूप को समाप्त कर देगा।

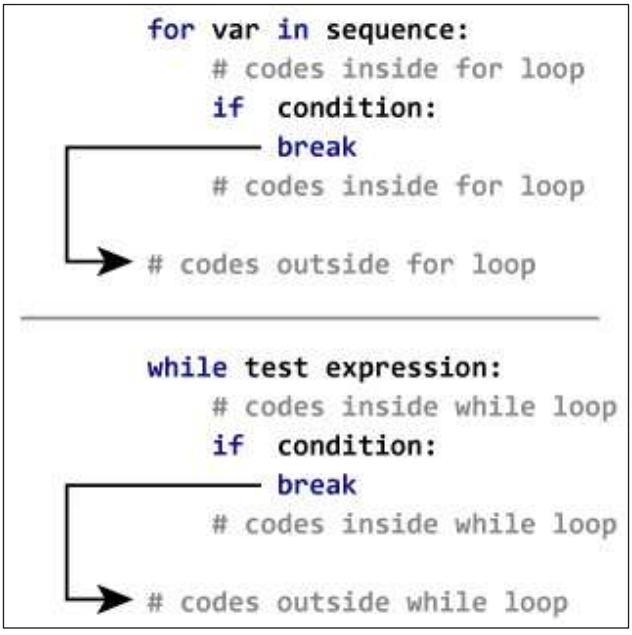
ब्रेक का सिंटेक्स

```
break
```

ब्रेक का फ़्लोचार्ट



लूप के लिए और लूप के दौरान ब्रेक स्टेटमेंट का कार्य नीचे दिखाया गया है।




```
# Use of break statement inside loop
for val in "string":
      if val == "i":
            break
       print(val)
print("The end")
```

आउटपुट

```
s
t
r
The end
```

इस कार्यक्रम में, हम "स्ट्रिंग" अनुक्रम के माध्यम से पुनरावृति करते हैं। इसमें हम जांचते हैं कि क्या अक्षर "i" है, जिस पर हम लूप से टूटते हैं। इसलिए, हम अपने आउटपुट में देखते हैं कि "i" तक के सभी अक्षर प्रिंट हो जाते हैं। उसके बाद, लूप समाप्त हो जाता है।

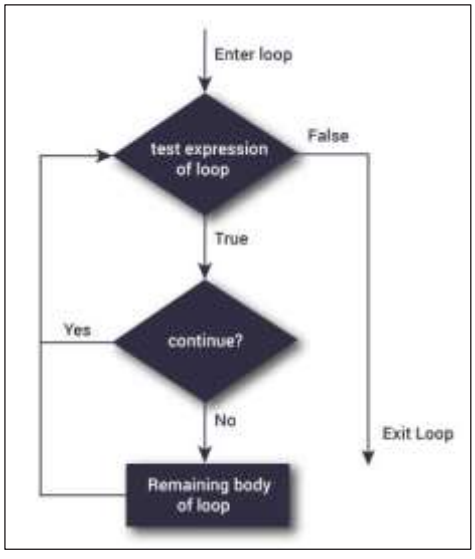
पायथन कंटिन्यू स्टेटमेंट

जारी बयान का उपयोग केवल वर्तमान पुनरावृत्ति के लिए लूप के अंदर शेष कोड को छोड़ने के लिए किया जाता है। लूप समाप्त नहीं होता है लेकिन अगले पुनरावृत्ति के साथ जारी रहता है।

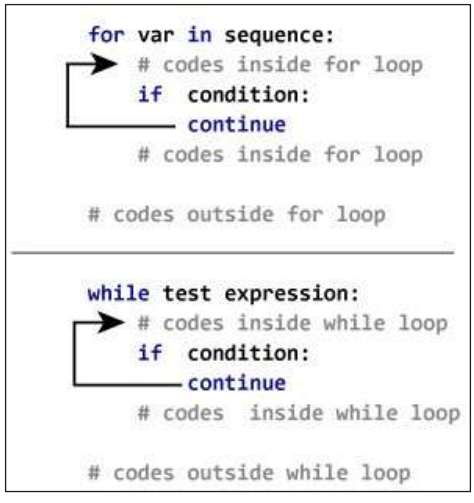
कंटिन्यू का सिंटेक्स

continue

कंटिन्यू का फ़्लोचार्ट



लूप के लिए और व्हाइल लूप में कंटिन्यू स्टेटमेंट का कार्य नीचे दिखाया गया है।




```
# Program to show the use of continue statement inside loops
for val in "string":
      if val == "i":
            continue
      print(val)
print("The end")
```

आउटपुट

```
s
t
r
n
g
The end
```

यह प्रोग्राम उपरोक्त उदाहरण के समान है सिवाय इसके कि ब्रेक स्टेटमेंट को बदल दिया गया है जारी रखें।

यदि स्ट्रिंग "i" है, तो हम लूप के साथ जारी रखते हैं, बाकी ब्लॉक को निष्पादित नहीं करते हैं। इसलिए, हम अपने आउटपुट में देखते हैं कि "i" को छोड़कर सभी अक्षर प्रिंट हो जाते हैं।

पास स्टेटमेंट का उपयोग कैसे करें

पायथन प्रोग्रामिंग में, पास एक अशक्त कथन है। एक टिप्पणी और पास के बीच का अंतर पायथन में कथन यह है कि, जबकि दुभाषिया एक टिप्पणी को पूरी तरह से अनदेखा करता है, पास को अनदेखा नहीं किया जाता है। हालांकि, पास निष्पादित होने पर कुछ नहीं होता है। इसका परिणाम नो ऑपरेशन (एनओपी) में होता है।

पास का सिंटेक्स

```
pass
```

हम आम तौर पर इसे प्लेसहोल्डर के रूप में उपयोग करते हैं।

मान लीजिए कि हमारे पास एक लूप या फ़ंक्शन है जो अभी तक लागू नहीं हुआ है, लेकिन हम इसे भविष्य में लागू करना चाहते हैं। उनके पास खाली शरीर नहीं हो सकता। इंटरप्रेटर शिकायत करेगा। तो, हम का उपयोग करते हैंउत्तीर्ण एक शरीर बनाने के लिए बयान जो कुछ भी नहीं करता है।

उदाहरण: पास स्टेटमेंट।

```
# pass is just a placeholder for
# functionality to be added later.
sequence = {'p', 'a', 's', 's'}
for val in sequence:
      pass
```

यही काम हम किसी खाली फंक्शन या क्लास में भी कर सकते हैं।

फंक्शन का उपयोग कैसे करें

पायथन में, फंक्शन संबंधित कथनों का एक समूह है जो एक विशिष्ट कार्य करता है।

फंक्शन हमारे प्रोग्राम को छोटे और मॉड्यूलर भागों में तोड़ने में मदद करते हैं। जैसे-जैसे हमारा कार्यक्रम बड़ा और बड़ा होता जाता है, फंक्शन इसे और अधिक व्यवस्थित और प्रबंधनीय बनाते हैं।

इसके अलावा, यह दोहराव से बचता है और कोड को पुन: प्रयोज्य बनाता है।

फंक्शन का सिंटैक्स

```
def function_name(parameters):
      “””docstring”””
      statement(s)
```

ऊपर दिखाई गई एक फ़ंक्शन की परिभाषा है जिसमें निम्नलिखित घटक शामिल होते हैं।

- कीवर्ड `def` फ़ंक्शन हेडर की शुरुआत को चिह्नित करता है।
- इसे विशिष्ट रूप से पहचानने के लिए एक फ़ंक्शन नाम दिया गया है। फ़ंक्शन नामकरण पायथन में पहचानकर्ता लिखने के समान नियमों का पालन करता है।
- पैरामीटर (आर्गुमेंट्स) जिसके माध्यम से हम किसी फ़ंक्शन को मान देते हैं। वे वैकल्पिक होते हैं।
- फ़ंक्शन हेडर के अंत को चिह्नित करने के लिए एक कोलन (:)।
- फ़ंक्शन क्या करता है इसका वर्णन करने के लिए वैकल्पिक दस्तावेज़ स्ट्रिंग (डॉकस्ट्रिंग)।
- एक या अधिक मान्य पायथन स्टेटमेंट जो फंक्शन बॉडी बनाते हैं। कथनों में समान इंडेंटेशन स्तर (आमतौर पर 4 रिक्त स्थान) होना चाहिए।
- एक वैकल्पिक `return` को स्टेटमेंट फ़ंक्शन से एक मान बापस किया जाता है।

फंक्शन का एक उदाहरण

```
def greet(name):
      “””This function greets to
      the person passed in as
       parameter”””
      print(“Hello, “ + name + “. Good morning!”)
```

पायथन में किसी फ़ंक्शन को कैसे कॉल करें

एक बार जब हम एक फ़ंक्शन को परिभाषित कर लेते हैं, तो हम इसे किसी अन्य फ़ंक्शन, प्रोग्राम या यहां तक कि पायथन प्रॉम्प्ट से कॉल कर सकते हैं। किसी फ़ंक्शन को कॉल करने के लिए हम केवल उपयुक्त पैरामीटर के साथ फ़ंक्शन का नाम टाइप करते हैं।

```
>>> greet('Paul')
Hello, Paul. Good morning!
```

आउटपुट देखने के लिए उपरोक्त कोड को पायथन शेल में चलाने का प्रयास करें।

डॉकस्ट्रिंग

फ़ंक्शन हेडर के बाद पहली स्ट्रिंग को डॉकस्ट्रिंग कहा जाता है और दस्तावेज़ीकरण स्ट्रिंग के लिए छोटा है। इसका उपयोग संक्षेप में यह समझाने के लिए किया जाता है कि कोई फंक्शन क्या करता है।

हालांकि हालांकि वैकल्पिक, प्रलेखन एक अच्छा प्रोग्रामिंग अभ्यास है। जब तक आपको याद न हो कि आपने पिछले सप्ताह रात के खाने के लिए क्या खाया था, हमेशा अपने कोड का दस्तावेजीकरण करें।

उपरोक्त उदाहरण में, हमारे पास फंक्शन हेडर के ठीक नीचे एक डॉकस्ट्रिंग है। हम आम तौर पर ट्रिपल कोट्स का उपयोग करते हैं ताकि डॉकस्ट्रिंग कई लाइनों तक विस्तारित हो सके। यह स्ट्रिंग हमारे लिए इस प्रकार उपलब्ध है और यह दस्तावेज़ फंक्शन की विशेषता है।

उदाहरण के लिए:

आउटपुट देखने के लिए निम्नलिखित को पायथन शेल में चलाने का प्रयास करें।

```
>>> print(greet.  doc  )
This function greets to
     the person passed into the
     name parameter
```

रिटर्न स्टेटमेंट

रिटर्न स्टेटमेंट का उपयोग किसी फ़ंक्शन से बाहर निकलने और उस स्थान पर वापस जाने के लिए किया जाता है जहां से इसे बुलाया गया था।

रिटर्न का सिंटेक्स

```
return [expression_list]
```

इस स्टेटमेंट में अभिव्यक्ति हो सकती है जिसका मूल्यांकन किया जाता है और मूल्य वापस कर दिया जाता है। अगर स्टेटमेंट में कोई एक्सप्रेशन नहीं है या रिटर्न स्टेटमेंट ही किसी फंक्शन के अंदर मौजूद नहीं है, तो फंक्शन किसी भी ऑब्जेक्ट को वापस नहीं करता।

उदाहरण के लिए:

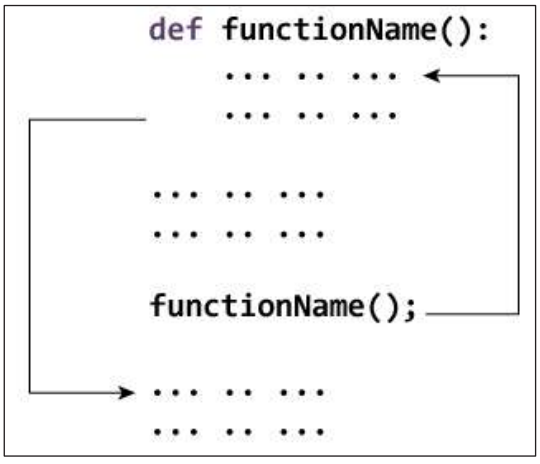
```
>>> print(greet("May"))
Hello, May. Good morning!
```

यहां, None लौटाया गया मूल्य है।

रिटर्न का उदाहरण

```
def absolute_value(num):
      “””This function returns the absolute
      value of the entered number”””
if num >= 0:
            return num
      else:
            return -num
# Output: 2
print(absolute_value(2))
# Output: 4
print(absolute_value(-4))
```

पायथन में फंक्शन कैसे काम करता है



वेरिएबल का दायरा और जीवनकाल

एक वेरिएबल का दायरा एक कार्यक्रम का हिस्सा है जहां वेरिएबल को मान्यता दी जाती है। किसी फ़ंक्शन के अंदर परिभाषित पैरामीटर और वेरिएबल बाहर से दिखाई नहीं देते हैं। इसलिए, उनके पास एक स्थानीय दायरा है।

एक वेरिएबल का जीवनकाल वह अवधि है जिसके दौरान वेरिएबल मेमोरी में बाहर निकलता है। किसी फ़ंक्शन के अंदर वेरिएबल का जीवनकाल तब तक होता है जब तक फ़ंक्शन निष्पादित होता है।

एक बार जब हम फंक्शन से लौटते हैं तो वे नष्ट हो जाते हैं। इसलिए, एक फ़ंक्शन याद नहीं रखता है अपने पिछले कॉल से एक वेरिएबल का मान होता है।

किसी फ़ंक्शन के अंदर एक वेरिएबल के दायरे को चित्रित करने के लिए यहां एक उदाहरण दिया गया है।

```
def my_func():
      x = 10
      print("Value inside function:",x)
x = 20
my_func()
print("Value outside function:",x)
```

आउटपुट

```
Value inside function: 10
Value outside function: 20
```

यहां, हम देख सकते हैं कि का मान x प्रारंभ में 20 है। भले ही फ़ंक्शन `my_func()` बदल गया हो इसका मूल्य x से 10 तक, इसने फ़ंक्शन के बाहर के मान को प्रभावित नहीं करता है।

ऐसा इसलिए है क्योंकि फ़ंक्शन के अंदर वेरिएबल x एक से अलग (फ़ंक्शन के लिए स्थानीय) है बाहर। हालांकि उनके नाम समान होते हैं, वे अलग-अलग दायरे वाले दो अलग-अलग वेरिएबल होते हैं।

दूसरी ओर फ़ंक्शन के बाहर के वेरिएबल अंदर से दिखाई दे रहे हैं। यह उनका एक वैश्विक दायरा है।

हम इन मानों को फ़ंक्शन के अंदर से पढ़ सकते हैं लेकिन उन्हें बदल (लिख) नहीं सकते। फ़ंक्शन के बाहर वेरिएबल के मान को संशोधित करने के लिए, उन्हें वैश्विक कीवर्ड का उपयोग करके `global` वेरिएबल के रूप में घोषित किया जाना चाहिए।

फंक्शन के प्रकार

मूल रूप से, हम कार्यों को निम्नलिखित दो प्रकारों में विभाजित कर सकते हैं:

- बिल्ट-इन फंक्शन - फंक्शन जो पायथन में निर्मित होते हैं।
- उपयोगकर्ता-परिभाषित कार्य - स्वयं उपयोगकर्ताओं द्वारा परिभाषित कार्य।

रिकर्सन कैसे करें

रिकर्सन अपने आप में किसी चीज को परिभाषित करने की प्रक्रिया है।

एक भौतिक दुनिया का उदाहरण दो समानांतर दर्पणों को एक दूसरे के सामने रखना होता है। उनके बीच की कोई भी वस्तु पुनरावर्ती रूप से परावर्तित होती है।

पायथन रिकर्सिव फंक्शन

हम जानते हैं कि पायथन में एक फ़ंक्शन अन्य कार्यों को कॉल कर सकता है। फ़ंक्शन के लिए स्वयं को कॉल करना भी संभव है। इस प्रकार के निर्माण को पुनरावर्ती कार्य कहा जाता है।

एक पूर्णांक का भाज्य ज्ञात करने के लिए पुनरावर्ती फलन का एक उदाहरण निम्नलिखित है।

किसी संख्या का भाज्य 1 से उस संख्या तक के सभी पूर्णांकों का गुणनफल होता है। उदाहरण के लिए, 6 का भाज्य (6 के रूप में निरूपित!) 1*2*3*4*5*6 = 720 है।

रिकर्सिव फंक्शन का उदाहरण

```
# An example of a recursive function to
# find the factorial of a number
def calc_factorial(x):
      "”"This is a recursive function
      to find the factorial of an integer"""
if x == 1:
            return 1
      else:
            return (x * calc_factorial(x-1))
num = 4
print("The factorial of", num, "is", calc_factorial(num))
```

उपरोक्त उदाहरण में `calc_factorial()` एक पुनरावर्ती कार्य है क्योंकि यह स्वयं को कॉल करता है।

जब हम इस फ़ंक्शन को एक सकारात्मक पूर्णांक के साथ कहते हैं, तो यह को कम करके खुद को पुनरावर्ती रूप से कॉल करेगा।

प्रत्येक फ़ंक्शन कॉल नंबर 1 के फैक्टोरियल के साथ संख्या को तब तक गुणा करता है जब तक कि संख्या एक के बराबर न हो जाए। इस पुनरावर्ती कॉल को निम्नलिखित वेरिएबलणों में समझाया जा सकता है।

```
calc_factorial(4)                  # 1st call with 4
4 * calc_factorial(3)              # 2nd call with 3
4 * 3 * calc_factorial(2)          # 3rd call with 2
4 * 3 * 2 * calc_factorial(1)      # 4th call with 1
4 * 3 * 2 * 1                      # return from 4th call as number=1
4 * 3 * 2                          # return from 3rd call
4 * 6                              # return from 2nd call
24                                 # return from 1st call
```

जब संख्या घटकर 1 हो जाती है तो हमारा पुनरावर्तन समाप्त हो जाता है। इसे आधार स्थिति कहा

प्रत्येक रिकर्सिव फ़ंक्शन में मूल शर्त होनी चाहिए जो रिकर्सन को रोकती है या फिर फ़ंक्शन खुद को अनंत कहते हैं।

रिकर्सन के लाभ

- पुनरावर्ती कार्य कोड को साफ और सुरुचिपूर्ण बनाते हैं।
- रिकर्सन का उपयोग करके एक जटिल कार्य को सरल उप-समस्याओं में विभाजित किया जा सकता है।
- कुछ नेस्टेड पुनरावृत्ति का उपयोग करने की तुलना में रिकर्सन के साथ अनुक्रम पीढ़ी आसान होती है।

रिकर्सन के नुकसान

- कभी-कभी रिकर्सन के पीछे तर्क का पालन करना मुश्किल होता है।
- रिकर्सिव कॉल महंगे (अक्षम) होते हैं क्योंकि वे बहुत अधिक मेमोरी और समय लेते हैं।
- पुनरावर्ती कार्यों को डीबग करना कठिन है।

एनोनिमस / लैम्ब्डा फंक्शन का प्रयोग कैसे करें

पायथन में लैम्ब्डा फ़ंक्शन में निम्नलिखित सिंटैक्स होता है।

पायथन में लैम्ब्डा फंक्शन का सिंटैक्स

```
lambda arguments: expression
```

लैम्ब्डा फ़ंक्शंस में कई तर्क हो सकते हैं लेकिन केवल एक अभिव्यक्ति हो सकती है। अभिव्यक्ति का मूल्यांकन किया जाता है और उसे वापस कर दिया जाता है। लैम्ब्डा फ़ंक्शंस का उपयोग जहां भी फ़ंक्शन ऑब्जेक्ट्स की आवश्यकता होती है, वहां किया जा सकता है।

पायथन में लैम्ब्डा फंक्शन का उदाहरण

यहां लैम्ब्डा फ़ंक्शन का एक उदाहरण दिया गया है जो इनपुट मान को दोगुना करता है।

```
# Program to show the use of lambda functions
double = lambda x: x * 2
# Output: 10
print(double(5))
```

उपरोक्त प्रोग्राम में, लैम्ब्डा `x: x * 2` लैम्ब्डा फंक्शन है। यहां x तर्क है और `x * 2` वह अभिव्यक्ति है जिसका मूल्यांकन किया जाता है और वापस किया जाता है।

इस फ़ंक्शन का कोई नाम नहीं है। यह एक फ़ंक्शन ऑब्जेक्ट देता है जिसे पहचानकर्ता डबल को असाइन किया जाता है। अब हम इसे एक सामान्य कार्य कह सकते हैं। बयान:

```
double = lambda x: x * 2
```

लगभग समान है:

```
def double(x):
    return x * 2
```

पायथन में लैम्ब्डा फंक्शन का उपयोग

हम लैम्ब्डा फ़ंक्शन का उपयोग तब करते हैं जब हमें थोड़े समय के लिए एक अनाम फ़ंक्शन की आवश्यकता होती है।

पायथन में, हम आम तौर पर इसे उच्च-क्रम फ़ंक्शन (एक फ़ंक्शन जो अन्य कार्यों को तर्क के रूप में लेता है) के तर्क के रूप में उपयोग करते हैं। लैम्ब्डा फ़ंक्शंस का उपयोग अंतर्निहित फ़ंक्शंस जैसे फ़िल्टर (), मैप () आदि के साथ किया जाता है।

फ़िल्टर के साथ प्रयोग () करने के उदाहरण

पायथन में `filter()` फ़ंक्शन एक फ़ंक्शन और एक सूची को तर्क के रूप में लेता है।

फ़ंक्शन को सूची में सभी आइटमों के साथ बुलाया जाता है और एक नई सूची लौटा दी जाती है जिसमें आइटम शामिल होते हैं जिसके लिए फ़ंक्शन का मूल्यांकन करता है सत्य.

यहां एक सूची से केवल सम संख्याओं को फ़िल्टर करने के लिए `filter()` फ़ंक्शन का उपयोग करने का एक उदाहरण दिया गया है।

```
# Program to filter out only the even items from a list

my_list = [1, 5, 4, 6, 8, 11, 3, 12]

new_list = list(filter(lambda x: (x%2 == 0), my_list))

# Output: [4, 6, 8, 12]

print(new_list)
```

मानचित्र के साथ प्रयोग करने () का उदाहरण

पायथन में फ़िल्टर () फ़ंक्शन एक फ़ंक्शन और एक सूची को तर्क के रूप में लेता है।

फ़ंक्शन को सूची में सभी आइटमों के साथ बुलाया जाता है और एक नई सूची लौटा दी जाती है जिसमें आइटम शामिल होते हैं प्रत्येक आइटम के लिए उस फ़ंक्शन द्वारा लौटाया गया।

सूची में सभी आइटम को दोगुना करने के लिए मानचित्र () फ़ंक्शन का एक उदाहरण उपयोग यहां दिया गया है।

```
# Program to double each item in a list using map()

my_list = [1, 5, 4, 6, 8, 11, 3, 12]
```

```
new_list = list(map(lambda x: x * 2, my_list))

# Output: [2, 10, 8, 12, 16, 22, 6, 24]

print(new_list)
```

ग्लोबल, लोकल और नॉन-लोकल वेरिएबल का उपयोग कैसे करें

ग्लोबल वेरिएबल

पायथन में, फ़ंक्शन के बाहर या वैश्विक दायरे में घोषित एक वेरिएबल को वैश्विक वेरिएबल के रूप में जाना जाता है। इसका मतलब है कि ग्लोबल वेरिएबल को फंक्शन के अंदर या बाहर एक्सेस किया जा सकता है।

आइए एक उदाहरण देखें कि पायथन में एक वैश्विक वेरिएबल कैसे बनाया जाता है।

एक वैश्विक वेरिएबल बनाएं

```
x = "global"

def foo():

      print("x inside :", x)

foo()

print("x outside:", x)
```

जब हम कोड चलाते हैं, तो वसीयत आउटपुट होगा:

```
x inside : global

x outside: global
```

उपरोक्त कोड में, हमने बनाया x को वैश्विक वेरिएबल के रूप में और वैश्विक वेरिएबल x . को मुद्रित करने के लिए `foo()` को परिभाषित किया गया है. अंत में, हम कॉल करते हैं `foo()` जो प्रिंट करेगा का मूल्य एक्स. क्या होगा यदि आप . का मान बदलना चाहते हैंएक्स एक फंक्शन के अंदर? एक्स = "वैश्विक"

```
x = "global"

def foo():

      x = x * 2

      print(x)

foo()
```

जब हम कोड चलाते हैं, तो वसीयत आउटपुट होगा:

```
UnboundLocalError: local variable 'x' referenced before assignment
```

आउटपुट एक त्रुटि दिखाता है क्योंकि पायथन x को स्थानीय वेरिएबल के रूप में मानता है और x को भी परिभाषित नहीं किया गया है के भीतर `foo()`.

इस काम को करने के लिए हम ग्लोबल कीवर्ड का उपयोग करते हैं, अधिक जानने के लिए पायथन ग्लोबल कीवर्ड पर जाएं।

लोकल वेरिएबल

फ़ंक्शन के शरीर के अंदर या स्थानीय दायरे में घोषित एक वेरिएबल को स्थानीय वेरिएबल के रूप में जाना जाता है।

एक्सेस के बाहर स्थानीय वेरिएबल तक पहुंचना

```
def foo():
      y = "local"
foo()
print(y)
```

जब हम कोड चलाते हैं, तो वसीयत आउटपुट होगा:

```
NameError: name 'y' is not defined
```

आउटपुट एक त्रुटि दिखाता है, क्योंकि हम एक स्थानीय वेरिएबल y को वैश्विक दायरे में एक्सेस करने का प्रयास कर रहे हैं जबकि लोकल वेरिएबल केवल अंदर काम करता है `foo()` या लोकल दायरा है।

आइए एक उदाहरण देखें कि पायथन में एक स्थानीय वेरिएबल कैसे बनाया जाता है।

एक लोकल वेरिएबल बनाएं

आम तौर पर, हम स्थानीय वेरिएबल बनाने के लिए फ़ंक्शन के अंदर एक वेरिएबल घोषित करते हैं।

```
def foo():
      y = "local"
      print(y)
foo()
```

जब हम कोड चलाते हैं, तो यह आउटपुट होगा:

```
local
```

आइए पहले की समस्या पर एक नज़र डालते हैं जहां x एक ग्लोबल वेरिएबल था और हम इसे संशोधित करना चाहते थे- fy एक्स के भीतर `foo()`.

ग्लोबल और लोकल वेरिएबल

यहां, हम दिखाएंगे कि एक ही कोड में वैश्विक वेरिएबल और स्थानीय वेरिएबल का उपयोग कैसे करें।

समान कोड में ग्लोबल और लोकल वेरिएबल का उपयोग करें

```
x = "global"
```

```
def foo():

      global x
      y = "local"
      x = x * 2
      print(x)
      print(y)
foo()
```

जब हम कोड चलाते हैं, तो रिटर्न आउटपुट मिलता है:

```
global global
local
```

उपरोक्त कोड में, हम x को वैश्विक और y को foo() में स्थानीय चर के रूप में घोषित करते हैं। फिर, हम वैश्विक चर x को संशोधित करने के लिए गुणन ऑपरेटर * का उपयोग करते हैं और हम x और y दोनों को प्रिंट करते हैं।

foo () को कॉल करने के बाद, x का मान वैश्विक वैश्विक हो जाता है क्योंकि हमने x * 2 का उपयोग दो बार वैश्विक प्रिंट करने के लिए किया था। उसके बाद हम लोकल वेरिएबल y यानी लोकल की वैल्यू प्रिंट करते हैं।

समान नाम के साथ ग्लोबल वेरिएबल और लोकल वेरिएबल

```
x = 5
def foo():
      x = 10
      print("local x:", x)
foo()
print("global x:", x)
```

जब हम कोड चलाते हैं, तो हमें आउटपुट मिलता है:

```
local x: 10
global x: 5
```

उपरोक्त कोड में, हमने वैश्विक वेरिएबल और स्थानीय वेरिएबल दोनों के लिए समान नाम x का उपयोग किया है। जब हम एक ही वेरिएबल को प्रिंट करते हैं तो हमें अलग-अलग परिणाम मिलते हैं क्योंकि वेरिएबल को दोनों स्कोप में घोषित किया जाता है, यानी `foo()` के अंदर लोकल स्कोप और `foo()` के बाहर ग्लोबल स्कोप।

जब हम वेरिएबल को के अंदर प्रिंट करते हैं `foo()` यह आउटपुट करता है स्थानीय एक्स: 10, यह वेरिएबल का स्थानीय दायरा कहलाता है।

इसी तरह, जब हम वेरिएबल को के बाहर प्रिंट करते हैं `foo()`, यह आउटपुट करता है वैश्विक एक्स: 5, यह कहा जाता है वेरिएबल का वैश्विक दायरा।

नॉन-लोकल वेरिएबल

गैर-स्थानीय वेरिएबल का उपयोग नेस्टेड फ़ंक्शन में किया जाता है जिसका स्थानीय दायरा परिभाषित नहीं होता है। इसका मतलब है कि वेरिएबल न तो स्थानीय और न ही वैश्विक दायरे में हो सकता है।

आइए एक उदाहरण देखें कि पायथन में एक वैश्विक वेरिएबल कैसे बनाया जाता है।

नॉन-लोकल वेरिएबल बनाने के लिए हम नॉन-लोकल कीवर्ड का उपयोग करते हैं।

एक नॉन-लोकल वेरिएबल का निर्माण करें

```
def outer():
      x = "local"
      def inner():
            nonlocal x
            x = "nonlocal"
            print("inner:", x)
      inner()
      print("outer:", x)
outer()
```

जब हम कोड चलाते हैं, तो वसीयत आउटपुट होगा:

```
inner: nonlocal
outer: nonlocal
```

उपरोक्त कोड में एक नेस्टेड फ़ंक्शन है `inner()` . हम प्रयोग करते हैं गैर स्थानीय गैर बनाने के लिए कीवर्ड स्थानीय वेरिएबल। आंतरिक () फ़ंक्शन को किसी अन्य फ़ंक्शन बाहरी () के दायरे में परिभाषित किया गया है।

यदि हम गैर-स्थानीय वेरिएबल का मान बदलते हैं, तो परिवर्तन स्थानीय वेरिएबल में दिखाई देते हैं।

ग्लोबल कीवर्ड का उपयोग कैसे करें

पायथन में, वैश्विक कीवर्ड आपको वर्तमान दायरे के बाहर वेरिएबल को संशोधित करने की अनुमति देता है। यह है एक वैश्विक वेरिएबल बनाने और स्थानीय संदर्भ में वेरिएबल में परिवर्तन करने के लिए उपयोग किया जाता है।

वैश्विक कीवर्ड के नियम

पायथन में वैश्विक कीवर्ड के लिए बुनियादी नियम हैं:

- जब हम किसी फ़ंक्शन के अंदर एक वेरिएबल बनाते हैं, तो यह डिफ़ॉल्ट रूप से स्थानीय होता है।

- जब हम किसी फ़ंक्शन के बाहर एक वेरिएबल परिभाषित करते हैं, तो यह डिफ़ॉल्ट रूप से वैश्विक होता है। आपको करने की ज़रूरत नहीं है उपयोग ग्लोबल कीवर्ड।
- किसी फंक्शन के अंदर ग्लोबल वेरिएबल को पढ़ने और लिखने के लिए हम ग्लोबल कीवर्ड का उपयोग करते हैं।
- किसी फ़ंक्शन के बाहर वैश्विक कीवर्ड के उपयोग का कोई प्रभाव नहीं पड़ता है।

वैश्विक कीवर्ड का उपयोग

आइए एक उदाहरण लेते हैं।

एक फंक्शन के अंदर से ग्लोबल वेरिएबल तक पहुंचना

```
c = 1 # global variable
def add():
      print(c)
add()
```

जब हम उपरोक्त प्रोग्राम चलाते हैं, तो आउटपुट होगा:

```
1
```

हालाँकि, हमारे पास कुछ परिदृश्य हो सकते हैं जहाँ हमें किसी फ़ंक्शन के अंदर से वैश्विक चर को संशोधित करने की आवश्यकता होती है।

फंक्शन के अंदर से ग्लोबल वेरिएबल को संशोधित करना

```
c = 1 # global variable
def add():
      c = c + 2 # increment c by 2
      print(c)
add()
```

जब हम उपरोक्त प्रोग्राम चलाते हैं, तो आउटपुट एक त्रुटि दिखाता है:

```
UnboundLocalError: local variable 'c' referenced before assignment
```

ऐसा इसलिए है क्योंकि हम केवल वैश्विक वेरिएबल का उपयोग कर सकते हैं, लेकिन इसे फ़ंक्शन के अंदर से संशोधित नहीं कर सकते। इसका समाधान है

वैश्विक कीवर्ड का उपयोग करना।

ग्लोबल का उपयोग करके एक फंक्शन के अंदर से ग्लोबल वेरिएबल को बदलना

```
c = 0 # global variable
def add():
```

```
    global c
    c = c + 2 # increment by 2


    print("Inside add():", c)
add()
print("In main:", c)
```

जब हम उपरोक्त प्रोग्राम चलाते हैं, तो आउटपुट होगा:

```
Inside add(): 2
In main: 2
```

उपरोक्त कार्यक्रम में, हम c को ऐड () फ़ंक्शन के अंदर एक वैश्विक कीवर्ड के रूप में परिभाषित करते हैं।

फिर, हम वेरिएबल c को 1 से बढ़ाते हैं, यानी `c = c + 2`। उसके बाद, हम ऐड () फ़ंक्शन को कॉल करते हैं। अंत में, हम वैश्विक वेरिएबल c को प्रिंट करते हैं।

जैसा कि हम देख सकते हैं, फ़ंक्शन के बाहर वैश्विक वेरिएबल पर भी परिवर्तन हुआ, c = 2।

सार्वत्रिक वेरिएबल पायथन मॉड्यूल के पार

पायथन में, हम वैश्विक वेरिएबल रखने के लिए एक एकल मॉड्यूल `config.py` बनाते हैं और एक ही प्रोग्राम के भीतर पायथन मॉड्यूल में जानकारी साझा करते हैं।

यहां बताया गया है कि हम वैश्विक वेरिएबल को पायथन मॉड्यूल में कैसे साझा कर सकते हैं।

पायथन मॉड्यूल में एक वैश्विक वेरिएबल साझा करें

वैश्विक चरों को संग्रहीत करने के लिए एक config.py फ़ाइल बनाएँ:

```
a = 0
b = "empty"
```

वैश्विक वेरिएबल बदलने के लिए एक update.py फ़ाइल बनाएँ:

```
import config
config.a = 10
config.b = "alphabet"
```

मान में परिवर्तन का परीक्षण करने के लिए एक main.py फ़ाइल बनाएँ:

```
import config
import update
print(config.a)
print(config.b)
```

जब हम main.py फ़ाइल चलाते हैं तब आउटपुट होगा:

```
10
alphabet
```

उपरोक्त में, हम तीन फाइलें बनाते हैं: config.py, update.py और main.py।

मॉड्यूल config.py a और b के वैश्विक चरों को संग्रहीत करता है। Update.py फ़ाइल में, हम config.py मॉड्यूल आयात करते हैं और a और b के मानों को संशोधित करते हैं। इसी तरह, main.py फ़ाइल में हम config.py और update.py मॉड्यूल दोनों को आयात करते हैं। अंत में, हम वैश्विक वेरिएबल के मूल्यों को प्रिंट और परीक्षण करते हैं कि वे बदले गए हैं या नहीं।

नेस्टेड फंक्शन में ग्लोबल

यहां बताया गया है कि आप नेस्टेड फ़ंक्शन में ग्लोबल वेरिएबल का उपयोग कैसे कर सकते हैं।

नेस्टेड फ़ंक्शन में एक ग्लोबल वेरिएबल का प्रयोग करना

```
def foo():
      x = 20
def bar():
      global x
      x = 25
print("Before calling bar: ", x)
      print("Calling bar now")
      bar()
      print("After calling bar: ", x)
foo()
print("x in main : ", x)
```

आउटपुट है:

```
Before calling bar: 20
Calling bar now
After calling bar: 20
```

```
x in main : 25
```

उपरोक्त कार्यक्रम में हम नेस्टेड फ़ंक्शन बार () के अंदर वैश्विक वेरिएबल घोषित करते हैं। `foo` के अंदर () फंक्शन , x का वैश्विक कीवर्ड का कोई प्रभाव नहीं होता है।

बार() को कॉल करने से पहले और बाद में वेरिएबल x स्थानीय वैरिएबल यानी x = 20 का मान लेता है। `foo()` फ़ंक्शन के बाहर, वेरिएबल x बार() फ़ंक्शन यानी x = 25 में परिभाषित मान लेगा।

ऐसा इसलिए है होता क्योंकि हमने बार के अंदर ग्लोबल वैरिएबल बनाने के लिए x में ग्लोबल कीवर्ड का इस्तेमाल किया है () फंक्शन (स्थानीय दायरा)।

यदि हम बार () फ़ंक्शन के अंदर कोई परिवर्तन करते हैं, तो परिवर्तन स्थानीय दायरे से बाहर दिखाई देते हैं, अर्थात `foo ()`.

मॉड्यूल का उपयोग कैसे करें

मॉड्यूल एक फ़ाइल को संदर्भित करता है जिसमें पायथन स्टेटमेंट और परिभाषाएं होती हैं।

एक फ़ाइल जिसमें पायथन कोड होता है, उदाहरण के लिए: `example.py`, को मॉड्यूल कहा जाता है और इसका मॉड्यूल नाम उदाहरण होता है।

हम बड़े फंक्शन को छोटी प्रबंधनीय और संगठित फाइलों में तोड़ने के लिए मॉड्यूल का उपयोग करते हैं। फर- थर्मोर, मॉड्यूल कोड की पुन: प्रयोज्यता प्रदान करते हैं।

हम अपने सबसे अधिक उपयोग किए जाने वाले कार्यों को एक मॉड्यूल में परिभाषित कर सकते हैं और उनकी परिभाषाओं को विभिन्न कार्यक्रमों में कॉपी करने के बजाय इसे आयात कर सकते हैं।

आइए एक मॉड्यूल बनाएं। निम्नलिखित टाइप करें और इसे `example.py` के रूप में सहेजें।

```
# Python Module example

def add(a, b):

      “””This program adds two

      numbers and return the result”””

      result = a + b

      return result
```

यहां हमने उदाहरण नामक मॉड्यूल के अंदर एक फ़ंक्शन `add ()` को परिभाषित किया है। फंक्शन में लेता है दो संख्याएँ और उनकी राशि लौटाता है।

पायथन में मॉड्यूल आयात कैसे करें

हम एक मॉड्यूल के अंदर परिभाषाओं को दूसरे मॉड्यूल या पायथन में इंटरेक्टिव दुभाषिया में आयात कर सकते हैं।

हम ऐसा करने के लिए `import` कीवर्ड का उपयोग करते हैं। हमारे पहले से परिभाषित मॉड्यूल उदाहरण को आयात करने के लिए हम पायथन प्रॉम्प्ट में निम्नलिखित टाइप करते हैं।

```
>>> import example
```

यह उदाहरण में परिभाषित कार्यों के नाम सीधे वर्तमान प्रतीक तालिका में दर्ज नहीं करता है। यह केवल वहां मॉड्यूल नाम उदाहरण में प्रवेश करता है।

मॉड्यूल नाम का उपयोग करके हम डॉट का उपयोग करके फ़ंक्शन तक पहुंच सकते हैं। ऑपरेटर। उदाहरण के लिए:

```
>>> example.add(4,5.5)
9.5
```

पायथन में एक टन मानक मॉड्यूल उपलब्ध होते हैं।

आप पायथन मानक मॉड्यूल की पूरी सूची देख सकते हैं और वे किस लिए हैं। ये फ़ाइलें उस स्थान के अंदर लिब निर्देशिका में हैं जहां आपने पायथन स्थापित किया था।

मानक मॉड्यूल उसी तरह `import` किए जा सकते हैं जैसे हम अपने उपयोगकर्ता-परिभाषित मॉड्यूल को आयात करते हैं।

मॉड्यूल आयात करने के कई तरीके हैं। वे निम्नानुसार सूचीबद्ध हैं।

पायथन आयात विवरण

हम इम्पोर्ट विवरण का उपयोग करके एक मॉड्यूल आयात कर सकते हैं और डॉट ऑपरेटर का उपयोग करके इसके अंदर की परिभाषाओं तक पहुंच सकते हैं। यहां एक उदाहरण है।

```
# import statement example
# to import standard module math
import math
print("The value of pi is", math.pi)
```

जब आप प्रोग्राम चलाते हैं, तो आउटपुट होगा

```
The value of pi is 3.141592653589793
```

नामकरण के साथ आयात करें

हम एक मॉड्यूल का नाम बदलकर आयात कर सकते हैं:

```
# import module by renaming it
import math as m
print("The value of pi is", m.pi)
```

हमने इसका नाम बदल दिया है `math` मॉड्यूल के रूप में `m`। यह हमें कुछ मामलों में टाइपिंग समय बचा सकता है।

ध्यान दें कि गणित नाम को हमारे दायरे में मान्यता नहीं मिली है। इसलिए, `math.pi` अमान्य है, `m.pi` सही कार्यान्वयन है।

इम्पोर्ट स्टेटमेंट से पायथन

हम पूरे मॉड्यूल को आयात किए बिना मॉड्यूल से विशिष्ट नाम आयात कर सकते हैं। यहां एक उदाहरण है।

```
# import only pi from math module
```

```
from math import pi

print("The value of pi is", pi)
```

हमने मॉड्यूल से केवल विशेषता pi आयात किया है।

ऐसे मामले में हम डॉट ऑपरेटर का उपयोग नहीं करते हैं। हम निम्नानुसार कई विशेषताओं को आयात कर सकते थे।

```
>>> from math import pi,e

>>> pi

3.141592653589793

>>> e

2.718281828459045
```

सभी नाम इम्पोर्ट करें

हम निम्नलिखित निर्माण का उपयोग करके मॉड्यूल से सभी नाम (परिभाषाएं) आयात कर सकते हैं।

```
# import all names from the standard module math

from math import *

print("The value of pi is", pi)
```

हमने गणित मॉड्यूल से सभी परिभाषाएं आयात कीं। यह अंडरस्कोर के साथ शुरुआत करने वालों को छोड़कर सभी नाम बनाता है, जो हमारे दायरे में दिखाई देता है।

एस्टरिस्क (*) प्रतीक के साथ सब कुछ आयात करना एक अच्छा प्रोग्रामिंग अभ्यास नहीं है। इससे पहचानकर्ता के लिए डुप्लिकेट परिभाषाएं हो सकती हैं। यह हमारे कोड की पठनीयता को भी बाधित करता है।

पायथन मॉड्यूल सर्च पाथ

मॉड्यूल आयात करते समय, पायथन कई स्थानों पर देखता है। इंटरप्रेटर पहले `sys.path` में परिभाषित निर्देशिकाओं की सूची में एक अंतर्निर्मित मॉड्यूल की तलाश करता है (यदि नहीं मिला)। इसी क्रम में तलाश की जा रही है।

- वर्तमान निर्देशिका।
- `PYTHONPATH` (निर्देशिका की सूची के साथ एक पर्यावरण वेरिएबल)।
- स्थापना-निर्भर डिफ़ॉल्ट निर्देशिका।

```
>>> import sys

>>> sys.path

['',

'C:\\Python33\\Lib\\idlelib',
```

```
'C:\\Windows\\system32\\python33.zip',
'C:\\Python33\\DLLs',
'C:\\Python33\\lib',
'C:\\Python33',
'C:\\Python33\\lib\\site-packages']
```

हम अपना पथ जोड़ने के लिए इस सूची को संशोधित कर सकते हैं।

एक मॉड्यूल को रिलोड करना

पायथन इंटरप्रेटर सत्र के दौरान केवल एक बार मॉड्यूल इम्पोर्ट करता है। यह चीजों को और अधिक कुशल बनाता है। यह कैसे काम करता है यह दिखाने के लिए यहां एक उदाहरण दिया गया है।

मान लीजिए कि हमारे पास `my_module` नामक मॉड्यूल में निम्न कोड है।

```
# This module shows the effect of
# multiple imports and reload
print("This code got executed")
```

हम कई आयातों का प्रभाव देखते हैं।

```
>>> import my_module
This code got executed
>>> import my_module
>>> import my_module
```

हम देख सकते हैं कि हमारा कोड केवल एक बार निष्पादित हुआ है। कहने का तात्पर्य यह है कि हमारा मॉड्यूल केवल एक बार आयात किया गया था।

अब अगर कार्यक्रम के दौरान हमारा मॉड्यूल बदल जाता है, तो हमें इसे फिर से लोड करना होगा। एक तरह से करने के लिएयह इंटरप्रेटर को पुनरारंभ करना होगा। लेकिन इससे ज्यादा मदद नहीं मिलती है।

पायथन ऐसा करने का एक साफ तरीका प्रदान करता है। हम `imp mod` के अंदर रीलोड () फ़ंक्शन का उपयोग कर सकते हैं- एक मॉड्यूल को पुनः लोड करने के लिए `ule`। इस तरह किया जाता है।

```
>>> import imp
>>> import my_module
This code got executed
>>> import my_module
>>> imp.reload(my_module)
```

```
This code got executed
<module 'my_module' from '.\\my_module.py'>
```

डिर () बिल्ट-इन फंक्शन

हम उपयोग कर सकते हैं `dir ()` मॉड्यूल के अंदर परिभाषित नामों का पता लगाने के लिए कार्य करता है।

उदाहरण के लिए हमने मॉड्यूल उदाहरण में एक फ़ंक्शन ऐड () परिभाषित किया है जो हमारे पास शुरुआत में था।

```
>>> dir(example)
['  builtins  ',
'  cached  ',
'  doc  ',
'  file_  ,
'  initializing  ',
'  loader  ',
'  name  ',
'  package  ',
'add']
```

यहां, हम नामों की एक क्रमबद्ध सूची (जोड़ के साथ) देख सकते हैं। अन्य सभी नाम जो एक अंडर से शुरू होते हैं- स्कोर मॉड्यूल से जुड़ी डिफ़ॉल्ट पायथन विशेषताएँ हैं (हमने उन्हें स्वयं परिभाषित नहीं किया है)। उदाहरण के लिए,____नाम विशेषता में मॉड्यूल का नाम होता है।

```
>>> import example
>>> example.  name__
'example'
```

हमारे वर्तमान नेमस्पेस में परिभाषित सभी नामों को `dir ()` फ़ंक्शन का उपयोग करके पता लगाया जा सकता है- किसी भी तर्क से बाहर।

```
>>> a = 1
>>> b = "hello"
>>> import math
>>> dir()
['  builtins  ', '  doc  ', '  name  ', 'a', 'b', 'math', 'pyscripter']
```

पैकेज का उपयोग कैसे करें

हम आमतौर पर अपनी सभी फाइलों को अपने कंप्यूटर में एक ही स्थान पर स्टोर नहीं करते हैं। हम आसान पहुँच के लिए निर्देशिकाओं के एक सुव्यवस्थित पदानुक्रम का उपयोग करते हैं।

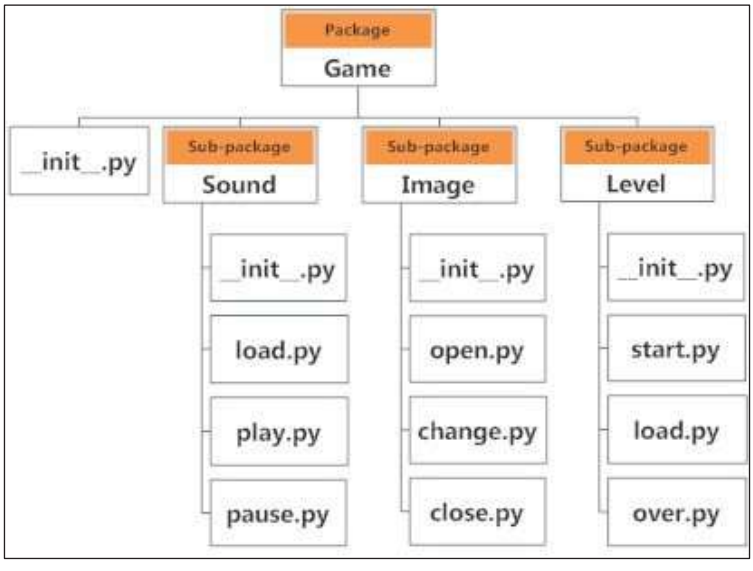
समान फ़ाइलें एक ही निर्देशिका में रखी जाती हैं, उदाहरण के लिए, हम सभी गीतों को "संगीत" निर्देशिका में रख सकते हैं। इसके अनुरूप, पायथन में निर्देशिकाओं के लिए पैकेज और फाइलों के लिए मॉड्यूल हैं।

जैसे-जैसे हमारा एप्लिकेशन प्रोग्राम बहुत सारे मॉड्यूल के साथ आकार में बड़ा होता जाता है, हम समान मॉड्यूल को एक पैकेज में और विभिन्न मॉड्यूल को विभिन्न पैकेजों में रखते हैं। यह एक परियोजना (कार्यक्रम) को प्रबंधित करने में आसान और अवधारणात्मक रूप से स्पष्ट बनाता है।

इसी तरह, एक निर्देशिका में उप-निर्देशिकाएं और फाइलें हो सकती हैं, एक पायथन पैकेज में उप-पैकेज और मॉड्यूल हो सकते हैं।

निर्देशिका में नाम की एक फ़ाइल होनी चाहिए `init .py` ताकि पायथन इसे एक पैकेज के रूप में मान सके। इस फाइल को खाली छोड़ा जा सकता है लेकिन हम आम तौर पर इस फाइल में उस पैकेज के लिए इनिशियलाइज़ेशन कोड डालते हैं।

यहां एक उदाहरण है, मान लीजिए कि हम एक गेम विकसित कर रहे हैं, नीचे पैकेज और मॉड्यूल का एक संभावित संगठन नीचे दिए गए चित्र में दिखाया जा सकता है।



पैकेज से मॉड्यूल इम्पोर्ट करना

हम डॉट (.) ऑपरेटर का उपयोग करके पैकेज से मॉड्यूल आयात कर सकते हैं।

उदाहरण के लिए, यदि उपरोक्त उदाहरण में प्रारंभ मॉड्यूल आयात करना चाहते हैं, तो यह निम्नानुसार किया जाता है।

```
import Game.Level.start
```

अब अगर इस मॉड्यूल में `select_difficulty()` नाम का एक फंक्शन है, तो हमें इसे संदर्भित करने के लिए पूरे नाम का उपयोग करना चाहिए।

```
Game.Level.start.select_difficulty(2)
```

यदि यह निर्माण लंबा लगता है, तो हम मॉड्यूल को पैकेज उपसर्ग के बिना निम्नानुसार इम्पोर्ट कर सकते हैं।

```
from Game.Level import start
```

अब हम फंक्शन को इस प्रकार से कॉल कर सकते हैं।

```
start.select_difficulty(2)
```

फिर भी एक पैकेज के भीतर एक मॉड्यूल बनाने के लिए आवश्यक फ़ंक्शन (या वर्ग या वेरिएबल) को आयात करने का एक और तरीका इस प्रकार होता है।

```
from Game.Level.start import select_difficulty
```

अब हम सीधे इस फ़ंक्शन को कॉल कर सकते हैं।

```
select_difficulty(2)
```

हालांकि यह आसान है, इस विधि की अनुशंसा नहीं की जाती है। पूर्ण नाम स्थान का उपयोग करने से भ्रम से बचा जाता है और दो समान पहचानकर्ता नामों को टकराने से रोकता है।

पैकेज आयात करते समय, पायथन `sys.path` में परिभाषित निर्देशिकाओं की सूची में दिखता है, जैसे कि मॉड्यूल खोज पथ के लिए होता है।

3

पायथन में डेटा के प्रकार

पायथन कई प्रकार के डेटा का समर्थन करता है जैसे कि पूर्णांक, लंबे पूर्णांक, फ्लोट वेरिएबल, बूलियन प्रकार, तार, टपल, शब्दकोश और जटिल संख्या। इस अध्याय में विस्तृत विषय पायथन में इन डेटा प्रकारों के बारे में बेहतर परिप्रेक्ष्य प्राप्त करने में मदद करेंगे।

नंबर डेटाटाइप का उपयोग कैसे करें

पायथन पूर्णांकों, फ्लोटिंग पॉइंट नंबरों और जटिल संख्याओं का समर्थन करता है। उन्हें `int` के रूप में परिभाषित किया गया है, पानी पर तैरना तथा पायथन में जटिल वर्ग है।

पूर्णांक और फ़्लोटिंग बिंदु दशमलव बिंदु की उपस्थिति या अनुपस्थिति से अलग होते हैं। 5 पूर्णांक है जबकि 5.0 एक अस्थायी बिंदु संख्या है।

सम्मिश्र संख्याएँ `x + yj` के रूप में लिखी जाती हैं, जहाँ `x` वास्तविक भाग है और `y` काल्पनिक है अंश।

हम टाइप () फ़ंक्शन का उपयोग यह जानने के लिए कर सकते हैं कि एक वेरिएबल या मूल्य किस वर्ग से संबंधित है और यह जांचने के लिए कि क्या यह किसी विशेष वर्ग से संबंधित है।

```
a = 5

# Output: <class 'int'>

print(type(a))

# Output: <class 'float'>

print(type(5.0))

# Output: (8+3j)

c = 5 + 3j

print(c + 3)

# Output: True

print(isinstance(c, complex))
```

जबकि पूर्णांक किसी भी लंबाई के हो सकते हैं, एक अस्थायी बिंदु संख्या केवल 15 दशमलव स्थान तक ही सटीक होती है (16 वां स्थान गलत है)।

जिन संख्याओं का हम प्रतिदिन व्यवहार करते हैं वे दशमलव (आधार 10) संख्या प्रणाली हैं। लेकिन कंप्यूटर प्रोग्रामर्स (आमतौर पर एम्बेडेड प्रोग्रामर) को बाइनरी (बेस 2), हेक्साडेसिमल (बेस 16) और ऑक्टल (बेस 8) नंबर सिस्टम के साथ काम करने की आवश्यकता होती है।

पायथन में, हम उस संख्या से पहले एक उपसर्ग को उचित रूप से रखकर इन संख्याओं का प्रतिनिधित्व कर सकते हैं। निम्न तालिका इन उपसर्गों को सूचीबद्ध करती है।

तालिका: पायथन संख्याओं के लिए संख्या प्रणाली उपसर्ग।

संख्या प्रणाली	उपसर्ग
बायनरी	'0b' or '0B'
अष्टभुजाकार	'0o' or '0O'
हेक्साडेसिमल	'0x' or '0X'

यहां कुछ उदाहरण दिए गए हैं:

```
# Output: 107
print(0b1101011)
# Output: 253 (251 + 2)
print(0xFB + 0b10)
# Output: 13
print(0o15)
```

जब आप प्रोग्राम चलाते हैं, तो आउटपुट होगा:

```
107
253
13
```

कन्वर्शन टाइप

हम एक प्रकार की संख्या को दूसरे प्रकार में बदल सकते हैं। इसे कॉएरसिओन के रूप में भी जाना जाता है।

यदि ऑपरेंड में से कोई एक फ़्लोट है, तो जोड़, घटाव ज़बरदस्ती पूर्णांक (स्वचालित रूप से) तैरने के लिए संचालन करता है।

```
>>> 1 + 2.0
3.0
```

हम ऊपर देख सकते हैं कि 1 (पूर्णांक) को जोड़ने के लिए 1.0 (फ्लोट) में ज़बरदस्ती किया गया है और परिणाम भी एक फ्लोटिंग पॉइंट नंबर होता है।

हम स्पष्ट रूप से प्रकारों के बीच कनवर्ट करने के लिए इंट (), फ्लोट () और कॉम्प्लेक्स () जैसे अंतर्निहित कार्यों का भी उपयोग कर सकते हैं। ये फ़ंक्शन स्ट्रिंग्स से भी परिवर्तित हो सकते हैं।

```
>>> int(2.3)
2
```

```
>>> int(-2.8)
-2
>>> float(5)
5.0
>>> complex('3+5j')
(3+5j)
```

फ्लोट से पूर्णांक में कनवर्ट करते समय, संख्या को छोटा कर दिया जाता है (पूर्णांक जो शून्य के करीब होता है)।

पायथन डेसीमल

पायथन बिल्ट-इन क्लास फ्लोट कुछ गणनाएँ करता है जो हमें विस्मित कर सकती हैं। हम सभी जानते हैं कि 1.1 और 2.2 का योग 3.3 है, लेकिन पायथन असहमत लगता है।

```
>>> (1.1 + 2.2) == 3.3
False
```

यह पता चला है कि फ्लोटिंग-पॉइंट नंबर कंप्यूटर हार्डवेयर में बाइनरी अंशों के रूप में लागू होते हैं, क्योंकि कंप्यूटर केवल बाइनरी (0 और 1) को समझता है। इस कारण से, अधिकांश दशमलव भिन्न जिन्हें हम जानते हैं, हमारे कंप्यूटर में सटीक रूप से संग्रहीत नहीं किए जा सकते हैं।

आइए हम एक उदाहरण लेते हैं। हम भिन्न 1/3 को दशमलव संख्या के रूप में प्रदर्शित नहीं कर सकते। यह 0.333333333... देगा जो असीम रूप से लंबा है, और हम केवल इसका अनुमान लगा सकते हैं।

दशमलव अंश 0.1 निकलता है, जिसके परिणामस्वरूप 0.00011001100110011001 का एक असीम रूप से लंबा बाइनरी अंश होगा ... और हमारा कंप्यूटर केवल इसकी एक सीमित संख्या को संग्रहीत करता है।

यह केवल लगभग 0.1 होगा लेकिन कभी भी बराबर नहीं होगा। इसलिए, यह हमारे कंप्यूटर हार्डवेयर की सीमा है और पायथन में कोई त्रुटि नहीं है।

```
>>> 1.1 + 2.2
3.3000000000000003
```

इस समस्या को दूर करने के लिए, हम दशमलव मॉड्यूल का उपयोग कर सकते हैं जो पायथन के साथ आता है। जबकि फ़्लोटिंग पॉइंट नंबरों में 15 दशमलव स्थानों तक सटीकता होती है, दशमलव मॉड्यूल में उपयोगकर्ता सेटटेबल परिशुद्धता होती है।

```
import decimal
# Output: 0.1
print(0.1)
#Output: Decimal('0.1000000000000000055511151231257827021181583404541015625')
print(decimal.Decimal(0.1))
```

इस मॉड्यूल का उपयोग तब किया जाता है जब हम दशमलव गणना करना चाहते हैं जैसे हमने स्कूल में सीखा था।

यह महत्व को भी बरकरार रखता है। हम जानते हैं कि 25.50 किग्रा 25.5 किग्रा से अधिक सटीक है क्योंकि इसमें एक की तुलना में दो महत्वपूर्ण दशमलव स्थान हैं।

```
from decimal import Decimal as D

# Output: Decimal('3.3')

print(D('1.1') + D('2.2'))

# Output: Decimal('3.000')

print(D('1.2') * D('2.50'))
```

उपरोक्त उदाहरण में अनुगामी शून्यों पर ध्यान दें।

हम पूछ सकते हैं, फ्लोट के बजाय हर बार दशमलव को लागू क्यों नहीं करते? मुख्य कारण दक्षता है। फ़्लोटिंग पॉइंट ऑपरेशंस किए जाते हैं जो दशमलव ऑपरेशंस की तुलना में तेज़ होते हैं।

फ्लोट के बजाए दशमलव का उपयोग कब करें?

हम आम तौर पर निम्नलिखित मामलों में दशमलव का उपयोग करते हैं।

- जब हम वित्तीय अनुप्रयोग कर रहे होते हैं जिन्हें सटीक दशमलव प्रतिनिधित्व की आवश्यकता होती है।
- जब हम आवश्यक परिशुद्धता के स्तर को नियंत्रित करना चाहते हैं।
- जब हम महत्वपूर्ण दशमलव स्थानों की धारणा को लागू करना चाहते हैं।
- जब हम चाहते हैं कि ऑपरेशन उसी तरह किया जाए जैसे हमने स्कूल में किया था।

पायथन फ्रैक्शन

पायथन अपने भिन्न मॉड्यूल के माध्यम से भिन्नात्मक संख्याओं को शामिल करते हुए संचालन प्रदान करता है।

एक भिन्न में एक अंश और एक हर होता है, जो दोनों पूर्णांक होते हैं। इस मॉड्यूल में परिमेय संख्या अंकगणित के लिए समर्थन करता है।

हम भिन्न भिन्न तरीकों से भिन्न वस्तुएं बना सकते हैं।

```
import fractions

# Output: 3/2

print(fractions.Fraction(1.5))

# Output: 5

print(fractions.Fraction(5))

# Output: 1/3

print(fractions.Fraction(1,3))
```

फ्लोट से भिन्न बनाते समय, हमें कुछ असामान्य परिणाम मिल सकते हैं। यह अपूर्ण बाइनरी फ्लोटिंग पॉइंट नंबर प्रतिनिधित्व के कारण है।

सौभाग्य से, फ्रैक्शन हमें स्ट्रिंग के साथ भी तुरंत चालू करने की अनुमति देता है। दशमलव संख्याओं का उपयोग करते समय यह पसंदीदा विकल्प है।

```
import fractions
# As float
# Output: 2476979795053773/2251799813685248
print(fractions.Fraction(1.1))
# As string
# Output: 11/10
print(fractions.Fraction('1.1'))
```

यह डेटाटाइप सभी बुनियादी कार्यों का समर्थन करता है। यहां कुछ उदाहरण दिए गए हैं।

```
from fractions import Fraction as F
# Output: 2/3
print(F(1,3) + F(1,3))
# Output: 6/5
print(1 / F(5,6))
# Output: False
print(F(-3,10) > 0)
# Output: True
print(F(-3,10) < 0)
```

पायथन गणित

पायथन विभिन्न गणित जैसे त्रिकोणमिति, लघुगणक, संभाव्यता और सांख्यिकी आदि को पूरा करने के लिए गणित और यादृच्छिक जैसे मॉड्यूल प्रदान करता है।

```
import math
# Output: 3.141592653589793
print(math.pi)
# Output: -1.0
print(math.cos(math.pi))
```

```
# Output: 22026.465794806718
print(math.exp(10))
# Output: 3.0
print(math.log10(1000))
# Output: 1.1752011936438014
print(math.sinh(1))
# Output: 720
print(math.factorial(6))
import random
# Output: 16
print(random.randrange(10,20))
x = ['a', 'b', 'c', 'd', 'e']
# Get random choice
print(random.choice(x))
# Shuffle x
random.shuffle(x)
# Print the shuffled x
print(x)
# Print random element
print(random.random())
```

सूची डेटाटाइप का उपयोग कैसे करें

पायथन प्रोग्रामिंग में, सभी वस्तुओं (तत्वों) को एक वर्ग ब्रैकेट [ ] के अंदर, अल्पविराम से अलग करके एक सूची बनाई जाती है।

इसमें किसी भी संख्या में आइटम हो सकते हैं और वे विभिन्न प्रकार (पूर्णांक, फ्लोट, स्ट्रिंग आदि) के हो सकते हैं।

```
# empty list
my_list = []
# list of integers
```

```
my_list = [1, 2, 3]

# list with mixed datatypes

my_list = [1, "Hello", 3.4]
```

साथ ही, एक सूची में एक आइटम के रूप में दूसरी सूची भी हो सकती है। इसे नेस्टेड सूची कहा जाता है।

```
# nested list

my_list = ["mouse", [8, 4, 6], ['a']]
```

सूची से तत्वों तक कैसे पहुँचें

ऐसे कई तरीके हैं जिनसे हम किसी सूची के तत्वों तक पहुँच प्राप्त कर सकते हैं।

सूची सूचकांक

सूची में किसी आइटम तक पहुँचने के लिए हम इंडेक्स ऑपरेटर [] का उपयोग कर सकते हैं। इंडेक्स 0 से शुरू होता है। इसलिए, 5 तत्वों वाली सूची में 0 से 4 तक इंडेक्स होता है।

किसी अन्य तत्व तक पहुंचने का प्रयास कर रहा है कि यह इंडेक्स एरर उठाएगा। सूचकांक एक पूर्णांक होना चाहिए। हम फ्लोट या `TypeError` का उपयोग नहीं कर सकते, इसका परिणाम `TypeError` में होगा।

नेस्टेड सूची को नेस्टेड इंडेक्सिंग का उपयोग करके एक्सेस किया जाता है

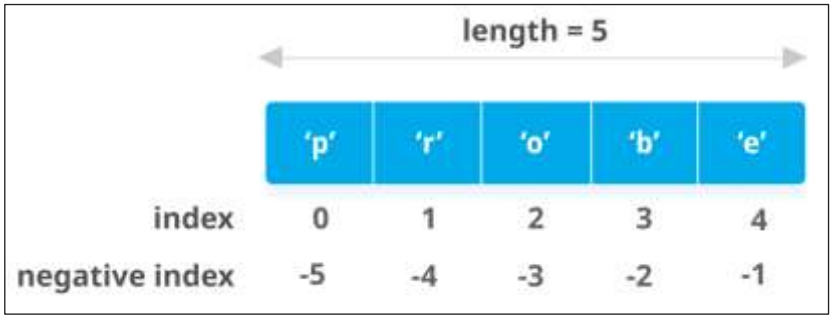
```
my_list = ['p','r','o','b','e']

# Output: p

print(my_list[0])

# Output: o

print(my_list[2])

# Output: e

print(my_list[4])

# Error! Only integer can be used for indexing


# my_list[4.0]

# Nested List

n_list = ["Happy", [2,0,1,5]]

# Nested indexing

# Output: a
print(n_list[0][1])
```

```
# Output: 5

print(n_list[1][3])
```

नेगेटिव इंडेक्सिंग

पायथन अपने अनुक्रमों के लिए नकारात्मक अनुक्रमण की अनुमति देता है। -1 का सूचकांक अंतिम आइटम को संदर्भित करता है, -2 से दूसरे अंतिम आइटम और इसी तरह।

```
my_list = ['p','r','o','b','e']

# Output: e

print(my_list[-1])

# Output: p print(my_list[-5])
```

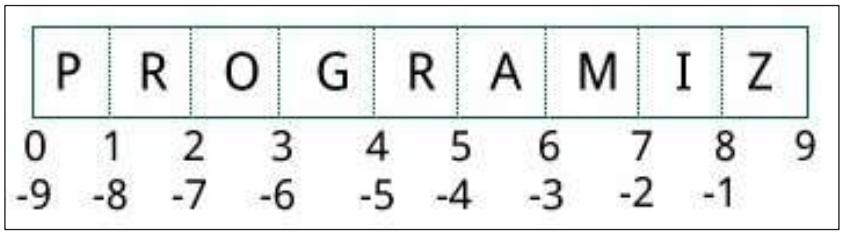


पायथन में सूचियों को कैसे काटें

हम स्लाइसिंग ऑपरेटर (कोलन) का उपयोग करके सूची में कई मदों तक पहुंच सकते हैं।

```
my_list = ['p','r','o','g','r','a','m','i','z']

# elements 3rd to 5th

print(my_list[2:5])

# elements beginning to 4th

print(my_list[:-5])

# elements 6th to end

print(my_list[5:])

# elements beginning to end

print(my_list[:])
```

नीचे दिखाए गए अनुसार तत्वों के बीच सूचकांक पर विचार करके स्लाइसिंग की सबसे अच्छी कल्पना की जा सकती है। इसलिए यदि हम किसी श्रेणी तक पहुंचना चाहते हैं, तो हमें दो सूचकांकों की आवश्यकता है जो उस हिस्से को सूची से हटा देंगे।



किसी सूची में तत्वों को कैसे बदलें या जोड़ें

सूची परिवर्तनशील होती है, अर्थात्, उनके तत्वों को स्ट्रिंग या टपल के विपरीत बदला जा सकता है। हम किसी आइटम या आइटम की

श्रेणी को बदलने के लिए असाइनमेंट ऑपरेटर (=) का उपयोग कर सकते हैं।

```
# mistake values

odd = [2, 4, 6, 8]

# change the 1st item

odd[0] = 1

# Output: [1, 4, 6, 8]

print(odd)

# change 2nd to 4th items

odd[1:4] = [3, 5, 7]

# Output: [1, 3, 5, 7]

print(odd)
```

हम सूची में एक आइटम का उपयोग करके जोड़ सकते हैं `append()` विधि या `extend()` का उपयोग करके कई आइटम जोड़ें
तरीका।

```
odd = [1, 3, 5]

odd.append(7)

# Output: [1, 3, 5, 7]

print(odd)

odd.extend([9, 11, 13])

# Output: [1, 3, 5, 7, 9, 11, 13]

print(odd)
```

हम दो सूचियों को संयोजित करने के लिए + ऑपरेटर का भी उपयोग कर सकते हैं। इसे यूनियन टन भी कहते हैं। * ऑपरेटर दी गई संख्या लिए एक सूची

दोहराता है।

```
odd = [1, 3, 5]

# Output: [1, 3, 5, 9, 7, 5]

print(odd + [9, 7, 5])

#Output: [“re”, “re”, “re”]

print([“re”] * 3)
```

इसके अलावा, हम इंसर्ट () विधि का उपयोग करके एक आइटम को वांछित स्थान पर सम्मिलित कर सकते हैं या एक सूची के खाली स्लाइस में इसे निचोड़कर कई आइटम सम्मिलित कर सकते हैं।

```
odd = [1, 9]

odd.insert(1,3)

# Output: [1, 3, 9]

print(odd)

odd[2:2] = [5, 7]

# Output: [1, 3, 5, 7, 9]

print(odd)
```

किसी सूची से तत्वों को कैसे हटाएं

हम कीवर्ड `del` का उपयोग करके सूची से एक या अधिक आइटम हटा सकते हैं। यह सूची को पूरी तरह से हटा भी सकता है।

```
my_list = [‘p’,’r’,’o’,’b’,’l’,’e’,’m’]

# delete one item

del my_list[2]

# Output: [‘p’, ‘r’, ‘b’, ‘l’, ‘e’, ‘m’]

print(my_list)

# delete multiple items

del my_list[1:5]

# Output: [‘p’, ‘m’]

print(my_list)

# delete entire list

del my_list

# Error: List not defined
```

```
print(my_list)
```

हम दिए गए आइटम को हटाने के लिए `remove ()` विधि का उपयोग कर सकते हैं या दिए गए इंडेक्स पर किसी आइटम को हटाने के लिए `pop ()` विधि का उपयोग कर सकते हैं।

यदि अनुक्रमणिका प्रदान नहीं की जाती है तो `pop()` विधि अंतिम आइटम को हटा देती है और वापस कर देती है। यह हमें सूचियों को स्टैक के रूप में लागू करने में मदद करता है (पहले, अंतिम डेटा संरचना में)।

हम का भी उपयोग कर सकते हैं सूची खाली करने के लिए clear() विधि। my_list =

```
my_list = ['p','r','o','b','l','e','m']
my_list.remove('p')
# Output: ['r', 'o', 'b', 'l', 'e', 'm']
print(my_list)
# Output: 'o'
print(my_list.pop(1))
# Output: ['r', 'b', 'l', 'e', 'm']
print(my_list)
# Output: 'm'
print(my_list.pop())
# Output: ['r', 'b', 'l', 'e']
print(my_list)
my_list.clear()
# Output: []
print(my_list)
```

अंत में, हम तत्वों के एक टुकड़े को एक खाली सूची निर्दिष्ट करके सूची में आइटम भी हटा सकते हैं।

```
>>> my_list = ['p','r','o','b','l','e','m']
>>> my_list[2:3] = []
>>> my_list
['p', 'r', 'b', 'l', 'e', 'm']
>>> my_list[2:5] = []
>>> my_list
['p', 'r', 'm']
```

पायथन सूची के तरीके

पायथन प्रोग्रामिंग में सूची वस्तु के साथ उपलब्ध तरीके सारणीबद्ध हैं।

उन्हें `list.method()` के रूप में एक्सेस किया जाता है। कुछ विधियों का पहले ही उपयोग किया जा चुका है।

पायथन सूची के तरीके
परिशिष्ट () - सूची के अंत में एक तत्व जोड़ें
विस्तार () - एक सूची के सभी तत्वों को दूसरी सूची में जोड़ें
इन्सर्ट () - परिभाषित इंडेक्स पर एक आइटम डालें
हटाएं () - सूची से किसी आइटम को हटाता है
पॉप () - दिए गए इंडेक्स पर एक तत्व को हटाता है और लौटाता है
क्लियर() - सूची से सभी आइटम हटा देता है
अनुक्रमणिका () - पहले मिलान किए गए आइटम का सूचकांक लौटाता है
गिनती () - तर्क के रूप में पारित वस्तुओं की संख्या की गिनती देता है
सॉर्ट () - सूची में आइटम को आरोही क्रम में क्रमबद्ध करें
रिवर्स () - सूची में वस्तुओं के क्रम को उलट दें
कॉपी () - सूची की उथली प्रति लौटाता है

पायथन सूची विधियों के कुछ उदाहरण:

```
my_list = [3, 8, 1, 6, 0, 8, 4]

# Output: 1

print(my_list.index(8))

# Output: 2

print(my_list.count(8))

my_list.sort()

# Output: [0, 1, 3, 4, 6, 8, 8]

print(my_list)

my_list.reverse()

# Output: [8, 8, 6, 4, 3, 1, 0]

print(my_list)
```

सूची समझ: नई सूची बनाने का सुरुचिपूर्ण तरीका

पायथन में मौजूदा सूची से एक नई सूची बनाने के लिए सूची की समझ एक सुंदर और संक्षिप्त तरीका है।

लिस्ट कॉम्प्रिहेंशन में एक एक्सप्रेशन होता है जिसके बाद स्क्वायर ब्रैकेट्स के अंदर स्टेटमेंट के लिए होता है। यहां एक सूची बनाने के लिए एक उदाहरण दिया गया है जिसमें प्रत्येक आइटम 2 की शक्ति बढ़ा रहा है।

```
pow2 = [2 ** x for x in range(10)]
# Output: [1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
print(pow2)
```

यह कोड इसके बराबर है:

```
pow2 = []
for x in range(10):
    pow2.append(2 ** x)
```

एक सूची समझ में वैकल्पिक रूप से अधिक हो सकते हैं `for` या अगर बयान। एक वैकल्पिक अगरबयान नई सूची के लिए आइटम फ़िल्टर कर सकते हैं। यहां कुछ उदाहरण दिए गए हैं।

```
>>> pow2 = [2 ** x for x in range(10) if x > 5]
>>> pow2
[64, 128, 256, 512]
>>> odd = [x for x in range(20) if x % 2 == 1]
>>> odd
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
>>> [x+y for x in ['Python ','C '] for y in ['Language','Programming']]
['Python Language', 'Python Programming', 'C Language', 'C Programming']
```

पायथन में अन्य सूचि संचालन

लिस्ट मेम्बरशिप टेस्ट

कीवर्ड का उपयोग करके हम जांच सकते हैं कि कोई आइटम सूची में मौजूद है या नहीं।

```
my_list = ['p','r','o','b','l','e','m']
# Output: True
print('p' in my_list)
```

```
# Output: False
print('a' in my_list)
# Output: True
print('c' not in my_list)
```

एक सूची के माध्यम से पुनरावृति

लूप के लिए हम सूची में प्रत्येक आइटम के माध्यम से पुनरावृति कर सकते हैं।

```
for fruit in ['apple','banana','mango']:
      print("I like",fruit)
```

टपल का प्रयोग कैसे करें

पायथन में एक टपल एक सूची के समान है। दोनों के बीच अंतर यह है कि एक बार असाइन किए जाने के बाद हम एक टपल के तत्वों को बदल नहीं सकते हैं, जबकि एक सूची में, तत्वों को बदला जा सकता है।

एक टपल बनाना

सभी वस्तुओं (तत्वों) को कोष्ठक () के अंदर, अल्पविराम से अलग करके एक टपल बनाया जाता है। कोष्ठक वैकल्पिक हैं, हालांकि, उनका उपयोग करना एक अच्छा अभ्यास है।

एक टपल में किसी भी संख्या में आइटम हो सकते हैं और वे विभिन्न प्रकार के हो सकते हैं (पूर्णांक, फ्लोट, सूची, स्ट्रिंग, आदि)।

```
# Empty tuple
my_tuple = ()
print(my_tuple) # Output: ()
# Tuple having integers
my_tuple = (1, 2, 3)
print(my_tuple) # Output: (1, 2, 3)
# tuple with mixed datatypes
my_tuple = (1, "Hello", 3.4)
print(my_tuple) # Output: (1, "Hello", 3.4)
# nested tuple
my_tuple = ("mouse", [8, 4, 6], (1, 2, 3))
```

```
# Output: ("mouse", [8, 4, 6], (1, 2, 3))
print(my_tuple)
```

कोष्ठक का उपयोग किए बिना भी एक टपल बनाया जा सकता है। इसे टपल पैकिंग के रूप में जाना जाता है।

```
my_tuple = 3, 4.6, "dog"
print(my_tuple) # Output: 3, 4.6, "dog"
# tuple unpacking is also possible
a, b, c = my_tuple
print(a) # 3
print(b) # 4.6
print(c) # dog
```

एक तत्व के साथ टपल बनाना थोड़ा मुश्किल है।

कोष्ठक में एक तत्व होना पर्याप्त नहीं है। हमें यह इंगित करने के लिए एक अनुगामी अल्पविराम की आवश्यकता होगी कि यह वास्तव में एक टपल होता है।

```
my_tuple = ("hello")
print(type(my_tuple)) # <class 'str'>
# Creating a tuple having one element
my_tuple = ("hello",)
print(type(my_tuple)) # <class 'tuple'>
# Parentheses is optional
my_tuple = "hello",
print(type(my_tuple)) # <class 'tuple'>
```

टपल तत्वों तक पहुंच

ऐसे कई तरीके हैं जिनसे हम टपल के तत्वों तक पहुच सकते हैं।

1. इंडेक्सिंग

हम इंडेक्स ऑपरेटर का उपयोग कर सकते हैं [] किसी आइटम को टपल में एक्सेस करने के लिए जहां इंडेक्स 0 से शुरू होता है।

तो, 6 तत्वों वाले टपल में 0 से 5 तक के सूचकांक होंगे। टपल के बाहर किसी तत्व तक पहुँचने का प्रयास (उदाहरण के लिए, 6, 7,...) एक `IndexError` बढ़ाएगा।

सूचकांक एक पूर्णांक होना चाहिए; इसलिए हम फ्लोट या अन्य प्रकारों का उपयोग नहीं कर सकते। इसके परिणामस्वरूप टाइप एरर हो सकता है।

```
my_tuple = ('p','e','r','m','i','t')
print(my_tuple[0]) # 'p'
print(my_tuple[5]) # 't'
# IndexError: list index out of range
# print(my_tuple[6])
# Index must be an integer
# TypeError: list indices must be integers, not float
# my_tuple[2.0]
# nested tuple
n_tuple = ("mouse", [8, 4, 6], (1, 2, 3))
# nested index
print(n_tuple[0][3]) # 's'
print(n_tuple[1][1]) # 4
```

2. नेगेटिव इंडेक्सिंग

पायथन अपने अनुक्रमों के लिए नकारात्मक अनुक्रमण की अनुमति देता है।

-1 का सूचकांक अंतिम आइटम को संदर्भित करता है, -2 से दूसरे अंतिम आइटम और इसी तरह से होते हैं।

```
my_tuple = ('p','e','r','m','i','t')
# Output: 't'
print(my_tuple[-1])
# Output: 'p'
print(my_tuple[-6])
```

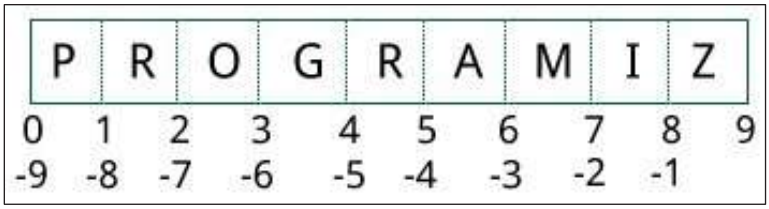
3. स्लाइसिंग

हम स्लाइसिंग ऑपरेटर - कोलन ":" का उपयोग करके टपल में कई मदों तक पहुंच सकते हैं।

```
my_tuple = ('p','r','o','g','r','a','m','i','z')
# elements 2nd to 4th
# Output: ('r', 'o', 'g')
print(my_tuple[1:4])
# elements beginning to 2nd
```

```
# Output: ('p', 'r')

print(my_tuple[:-7])

# elements 8th to end

# Output: ('i', 'z')

print(my_tuple[7:])

# elements beginning to end

# Output: ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i',

'z') print(my_tuple[:])
```

नीचे दिखाए गए अनुसार तत्वों के बीच सूचकांक पर विचार करके स्लाइसिंग की सबसे अच्छी कल्पना की जा सकती है।
इसलिए यदि हम किसी श्रेणी तक पहुंच प्राप्त करना चाहते हैं, तो हमें उस सूचकांक की आवश्यकता है जो टपल से भाग को काट देती है।



टपल को बदलना

सूचियों के विपरीत, टपल अपरिवर्तनीय हैं।

इसका मतलब है कि एक बार असाइन किए जाने के बाद टपल के तत्वों को बदला नहीं जा सकता है। लेकिन यदि तत्व स्वयं सूची की तरह एक परिवर्तनशील डेटाटाइप है, तो इसके नेस्टेड आइटम को बदला जा सकता है।

हम विभिन्न मूल्यों (पुनर्असाइनमेंट) के लिए एक टपल भी निर्दिष्ट कर सकते हैं।

```
my_tuple = (4, 2, 3, [6, 5])

# TypeError: 'tuple' object does not support item assignment

# my_tuple[1] = 9

# However, item of mutable element can be

changed my_tuple[3][0] = 9 # Output: (4, 2, 3,

[9, 5]) print(my_tuple)

# Tuples can be reassigned

my_tuple = ('p','r','o','g','r','a','m','i','z')

# Output: ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i',

'z') print(my_tuple)
```

हम इसका उपयोग कर सकते हैं + ऑपरेटर दो टपल को मिलाने के लिए। इसे यूनियन टन भी कहते हैं।

हम *ऑपरेटर का उपयोग करके दी गई संख्या के लिए तत्वों को टपल में दोहरा सकते हैं। दोनों + और * संचालन के परिणामस्वरूप एक नया टपल होता है।

```
# Concatenation
# Output: (1, 2, 3, 4, 5, 6)
print((1, 2, 3) + (4, 5, 6))
# Repeat
# Output: ('Repeat', 'Repeat', 'Repeat')
print(("Repeat",) * 3)
```

टपल को हटाना

जैसा कि ऊपर चर्चा की गई है, हम तत्वों को टपल में नहीं बदल सकते हैं। इसका मतलब यह भी है कि हम हटा नहीं सकते या टपल से आइटम हटा दें।

लेकिन कीवर्ड `del` का उपयोग करके टपल को पूरी तरह से हटाना संभव होता है।

```
my_tuple = ('p','r','o','g','r','a','m','i','z')
# can't delete items
# TypeError: 'tuple' object doesn't support item deletion
# del my_tuple[3]
# Can delete an entire tuple
del my_tuple
# NameError: name 'my_tuple' is not defined
print(my_tuple)
```

टपल करने के तरीके

आइटम जोड़ने या आइटम निकालने वाली विधियाँ टपल के साथ उपलब्ध नहीं हैं। केवल निम्नलिखित दो विधियाँ उपलब्ध हैं।

पायथन टपल विधि	
तरीका	विवरण
गिनती (एक्स)	मदों की संख्या लौटाता है एक्स
सूचकांक (एक्स)	पहले आइटम का सूचकांक लौटाता है जो के बराबर है एक्स

पायथन टपल विधियों के कुछ उदाहरण:

```
my_tuple = ('a','p','p','l','e',)
print(my_tuple.count('p')) # Output: 2
print(my_tuple.index('l')) # Output: 3
```

अन्य टपल संचालन

1. टपल सदस्यता परीक्षण

हम कीवर्ड का उपयोग करके जांच सकते हैं कि कोई आइटम टपल में मौजूद है या नहीं।

```
my_tuple = ('a','p','p','l','e',)
# In operation
# Output: True
print('a' in my_tuple)
# Output: False
print('b' in my_tuple)
# Not in operation
# Output: True
print('g' not in my_tuple)
```

2. टपल के माध्यम से पुनरावृति

लूप के लिए हम प्रत्येक आइटम के माध्यम से टपल में पुनरावृति कर सकते हैं।

```
# Output:
# Hello John
# Hello Kate
for name in ('John','Kate'):
    print("Hello",name)
```

टपल ओवर लिस्ट के लाभ

चूंकि टपल सूचियों से काफी मिलते-जुलते हैं, इसलिए दोनों का उपयोग समान स्थितियों में भी किया जाता है।

हालांकि, सूची में टपल को लागू करने के कुछ फायदे हैं। नीचे सूचीबद्ध कुछ मुख्य लाभ होते हैं:

- हम आम तौर पर विषम (अलग) डेटाटाइप के लिए टपल का उपयोग करते हैं और सजातीय (समान) डेटाटाइप के लिए सूची का उपयोग करते हैं।
- चूंकि टपल अपरिवर्तनीय होते हैं, टपल के माध्यम से पुनरावृति सूची की तुलना में तेज है। तो थोड़ा सा प्रदर्शन बढ़ावा है।
- अपरिवर्तनीय तत्वों वाले टपल को शब्दकोश की कुंजी के रूप में उपयोग किया जा सकता है। सूचियों के साथ, यह संभव नहीं है।
- यदि आपके पास डेटा है जो नहीं बदलता है, तो इसे टपल के रूप में लागू करने से यह गारंटी होगी कि यह लेखन-संरक्षित रहता है।

स्ट्रिंग का उपयोग कैसे करें

एक स्ट्रिंग वर्णों का एक क्रम होता है।

एक करैक्टर बस एक प्रतीक होता है। उदाहरण के लिए, अंग्रेजी भाषा में 26 वर्ण होते हैं।

कंप्यूटर वर्णों से नहीं निपटते, वे संख्याओं (बाइनरी) से निपटते हैं। भले ही आप अपनी स्क्रीन पर वर्ण देख सकते हैं, आंतरिक रूप से इसे 0 और 1 के संयोजन के रूप में संग्रहीत और हेरफेर किया जाता है।

किसी संख्या में वर्ण के इस रूपांतरण को एन्कोडिंग कहा जाता है और रिवर्स प्रक्रिया डिकोडिंग होती है। एएससीआईआई और यूनिकोड कुछ लोकप्रिय एन्कोडिंग हैं जिनका उपयोग किया जाता है।

पायथन में स्ट्रिंग यूनिकोड वर्ण का एक क्रम होता है। यूनिकोड को सभी भाषाओं में प्रत्येक वर्ण को शामिल करने और एन्कोडिंग में एकरूपता लाने के लिए पेश किया गया था।

पायथन में एक स्ट्रिंग कैसे बनाएं

एकल उद्धरण या दोहरे उद्धरण चिह्नों के अंदर वर्णों को संलग्न करके स्ट्रिंग्स बनाई जा सकती हैं। पाइथन में भी ट्रिपल कोट्स का उपयोग किया जा सकता है लेकिन आम तौर पर मल्टीलाइन स्ट्रिंग्स और डॉकस्ट्रिंग्स का प्रतिनिधित्व करने के लिए उपयोग किया जाता है।

```
# all of the following are equivalent
my_string = 'Hello'
print(my_string)
my_string = "Hello"
print(my_string)
my_string = '''Hello'''
print(my_string)
# triple quotes string can extend multiple lines
```

```
my_string = """Hello, welcome to
      the world of Python"""
print(my_string)
```

जब आप प्रोग्राम चलाते हैं, तो आउटपुट ये होता है:

```
Hello
Hello
Hello
Hello, welcome to
      the world of Python
```

एक स्ट्रिंग में कैरेक्टर तक कैसे पहुंचे

हम अनुक्रमण का उपयोग करके अलग-अलग वर्णों तक पहुँच सकते हैं और स्लाइसिंग का उपयोग करके वर्णों की एक श्रृंखला का उपयोग कर सकते हैं। इंडेक्स 0 से शुरू होता है। किसी कैरेक्टर को इंडेक्स रेंज से बाहर एक्सेस करने की कोशिश करने से `IndexError` बढ़ जाएगा। सूचकांक एक पूर्णांक होना चाहिए। हम फ्लोट या अन्य प्रकारों का उपयोग नहीं कर सकते हैं, इसका परिणाम `TypeError` में होगा।

पायथन अपने अनुक्रमों के लिए नकारात्मक अनुक्रमण की अनुमति देता है।

-1 का सूचकांक अंतिम आइटम को संदर्भित करता है, -2 से दूसरे अंतिम आइटम और इसी तरह। हम स्लाइसिंग ऑपरेटर (कोलन) का उपयोग करके एक स्ट्रिंग में कई मदों तक पहुंच सकते हैं।

```
str = 'programiz'
print('str = ', str)
#first character
print('str[0] = ', str[0])
#last character
print('str[-1] = ', str[-1])
#slicing 2nd to 5th character
print('str[1:5] = ', str[1:5])
#slicing 6th to 2nd last character
print('str[5:-2] = ', str[5:-2])
```

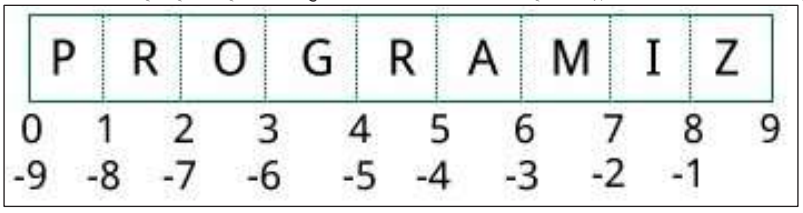
यदि हम इंडेक्स को सीमा से बाहर एक्सेस करने का प्रयास करते हैं या दशमलव संख्या का उपयोग करते हैं, तो हमें त्रुटियां मिलेंगी।

```
# index must be in range
>>> my_string[15]
```

```
...
IndexError: string index out of range
# index must be an integer
>>> my_string[1.5]
...
TypeError: string indices must be integers
```

नीचे दिखाए गए अनुसार तत्वों के बीच सूचकांक पर विचार करके स्लाइसिंग की सबसे अच्छी कल्पना की जा सकती है।

यदि हम किसी श्रेणी तक पहुँच प्राप्त करना चाहते हैं, तो हमें उस अनुक्रमणिका की आवश्यकता है जो स्ट्रिंग से भाग को काट देती है।



स्ट्रिंग को कैसे बदलें या हटाएं

स्ट्रिंग्स अपरिवर्तनीय होते हैं। इसका मतलब है कि एक स्ट्रिंग के तत्वों को एक बार असाइन किए जाने के बाद बदला नहीं जा सकता है। हम बस एक ही नाम के लिए अलग-अलग स्ट्रिंग्स को फिर से असाइन कर सकते हैं।

```
>>> my_string = 'programiz'
>>> my_string[5] = 'a'
...
TypeError: 'str' object does not support item assignment
>>> my_string = 'Python'
>>> my_string
'Python'
```

हम स्ट्रिंग से वर्णों को बदल या हटा नहीं सकते हैं। लेकिन स्ट्रिंग को पूरी तरह से हटाना संभव है इसके लिए आप कीवर्ड `del` का उपयोग कर सकते हैं।

```
>>> del my_string[1]
...
TypeError: 'str' object doesn't support item deletion
>>> del my_string
>>> my_string
```

```
...
NameError: name 'my_string' is not defined
```

पायथन स्ट्रिंग ऑपरेशंस

ऐसे कई ऑपरेशन हैं जो स्ट्रिंग के साथ किए जा सकते हैं जो इसे पायथन में सबसे अधिक उपयोग किए जाने वाले डेटाटाइप में से एक बनाता है।

दो या दो से अधिक स्ट्रिंग्स का संयोजन

दो या दो से अधिक धागों का एक ही में जुड़ना यूनियन टन कहलाता है।

+ ऑपरेटर इसे पायथन में करता है। बस दो स्ट्रिंग अक्षर को एक साथ लिखना भी उन्हें जोड़ता है।

* ऑपरेटर का उपयोग किसी निश्चित संख्या में स्ट्रिंग को दोहराने के लिए किया जा सकता है।

```
str1 = 'Hello'
str2 ='World!'
# using +
print('str1 + str2 = ', str1 + str2)
# using *
print('str1 * 3 =', str1 * 3)
```

दो स्ट्रिंग अक्षर को एक साथ लिखना भी उन्हें + ऑपरेटर की तरह जोड़ता है

यदि हम स्ट्रिंग्स को अलग-अलग लाइनों में जोड़ना चाहते हैं, तो हम कोष्ठक का उपयोग कर सकते हैं।

```
>>> # two string literals together
>>> 'Hello ''World!'
'Hello World!'
>>> # using parentheses
>>> s = ('Hello '
... 'World')
>>> s
'Hello World'
```

स्ट्रिंग के माध्यम से पुनरावृति

लूप के लिए हम इसके माध्यम से पुनरावृति कर सकते हैं एक स्ट्रिंग। यहाँ एक स्ट्रिंग में l› की संख्या गिनने के लिए एक उदाहरण दिया गया है।

```
count = 0
for letter in 'Hello World':
      if(letter == 'l'):
            count += 1
print(count,'letters found')
```

स्ट्रिंग मेंबरशिप टेस्ट

हम कीवर्ड का उपयोग करके यह जांच सकते हैं कि स्ट्रिंग के भीतर कोई उप स्ट्रिंग मौजूद है या नहीं।

```
>>> 'a' in 'program'
True
>>> 'at' not in 'battle'
False
```

पायथन के साथ काम करने के लिए अंतर्निहित कार्य

अनुक्रम के साथ काम करने वाले विभिन्न अंतर्निहित कार्य, स्ट्रिंग के साथ भी काम करते हैं।

आमतौर पर इस्तेमाल किए जाने वाले कुछ एन्यूमरेट () और लेन () हैं। एन्यूमरेट () फ़ंक्शन एक एन्यूमरेट ऑब्जेक्ट को फिर से चालू करता है। इसमें जोड़े के रूप में स्ट्रिंग में सभी वस्तुओं का सूचकांक और मूल्य होता है। यह पुनरावृत्ति के लिए उपयोगी हो सकता है।

इसी तरह, लेन () स्ट्रिंग की लंबाई (वर्णों की संख्या) देता है।

```
str = 'cold'
# enumerate()
list_enumerate = list(enumerate(str))
print('list(enumerate(str) = ', list_enumerate)
#character count
print('len(str) = ', len(str))
```

पायथन स्ट्रिंग फॉर्मेटिंग

अगर हम एक टेक्स्ट प्रिंट करना चाहते हैं जैसे - उसने कहा, "वहां क्या है?" - हम न तो सिंगल कोट या डबल कोट्स का उपयोग कर सकते हैं। इसका परिणाम `SyntaxError` होगा क्योंकि टेक्स्ट में ही सिंगल और डबल कोट्स दोनों होते हैं।

```
>>> print("He said, "What's there?"")
...
```

```
SyntaxError: invalid syntax
>>> print('He said, "What's there?"')
...
SyntaxError: invalid syntax
```

इस समस्या को हल करने का एक तरीका ट्रिपल कोट्स का उपयोग करना है। वैकल्पिक रूप से, हम एस्केप सीक्वेंस का उपयोग कर सकते हैं।

एस्केप सीक्वेंस बैकस्लैश से शुरू होता है और इसकी अलग तरह से व्याख्या की जाती है। यदि हम एक स्ट्रिंग का प्रतिनिधित्व करने के लिए एकल उद्धरण का उपयोग करते हैं, तो स्ट्रिंग के अंदर सभी एकल उद्धरणों से बचना चाहिए। डबल कोट्स के मामले में भी ऐसा ही है। यहां बताया गया है कि यह उपरोक्त पाठ का प्रतिनिधित्व करने के लिए कैसे किया जा सकता है।

```
# using triple quotes
print('''He said, "What's there?"''')
# escaping single quotes
print('He said, "What's \ there?"')
# escaping double quotes
print("He said, \"What's there?\"")
```

यहां पाइथन द्वारा समर्थित सभी एस्केप सीक्वेंस की सूची दी गई है।

पायथन में एस्केप सीक्वेंस

निकास का क्रम	विवरण
\newline	बैकस्लैश और न्यूलाइन पर ध्यान नहीं दिया गया
\\	बैकस्लैश
\'	एकल बोली
\"	दोहरे उद्धरण
\a	एएससीआईआई बेल
\b	एएससीआईआई बैकस्पेस
\f	एएससीआईआई फॉर्मफीड
\n	एएससीआईआई लाइनफीड
\r	एएससीआईआई कैरिज रिटर्न
\t	एएससीआईआई क्षैतिज टैब
\v	एएससीआईआई कार्यक्षेत्र टैब
\ooo	ऑक्टल वैल्यू वाला कैरेक्टर ooo
\xHH	हेक्साडेसिमल मान HH . के साथ वर्ण

यहां कुछ उदाहरण दिए गए हैं:

```
>>> print("C:\\Python32\\Lib")
C:\Python32\Lib
>>> print("This is printed\nin two lines")
This is printed
in two lines
>>> print("This is \x48\x45\x58 representation")
This is HEX representation
```

एस्केप अनुक्रम को अनदेखा करने के लिए रॉ स्ट्रिंग

कभी-कभी हम एक स्ट्रिंग के अंदर भागने के दृश्यों को अनदेखा करना चाह सकते हैं। ऐसा करने के लिए हम स्ट्रिंग के सामने r या R रख सकते हैं। इसका मतलब यह होगा कि यह एक कच्चा तार है और इसके अंदर किसी भी भागने के क्रम को नजरअंदाज कर दिया जाएगा।

```
>>> print("This is \x61 \ngood example")
This is a
good example
>>> print(r"This is \x61 \ngood example")
This is \x61 \ngood example
```

स्ट्रिंग्स को फ़ॉर्मेट करने के लिए फ़ॉर्मेट () विधि

`format()` विधि जो स्ट्रिंग ऑब्जेक्ट के साथ उपलब्ध है, स्ट्रिंग्स को स्वरूपित करने में बहुत बहुमुखी और शक्तिशाली है। फ़ॉर्मेट स्ट्रिंग्स में प्लेसहोल्डर या प्रतिस्थापन फ़ील्ड के रूप में घुंघराले ब्रेसिज़ {} होते हैं जो प्रतिस्थापित हो जाते हैं।

हम ऑर्डर निर्दिष्ट करने के लिए स्थितीय तर्क या कीवर्ड तर्क का उपयोग कर सकते हैं।

```
# default(implicit) order
default_order = "{}, {} and
{}".format('John','Bill','Sean') print('\n--- Default Order
---')
print(default_order)
# order using positional argument
positional_order = "{1}, {0} and {2}".format('John','Bill','Sean')
print('\n--- Positional Order ---')
```

```
print(positional_order)

# order using keyword argument

keyword_order = "{s}, {b} and

{j}".format(j='John',b='Bill',s='Sean') print('\n--- Keyword Order -

--')

print(keyword_order)
```

format () विधि में वैकल्पिक प्रारूप विनिर्देश हो सकते हैं। उन्हें कोलन का उपयोग करके फ़ील्ड नाम से अलग किया जाता है। उदाहरण के लिए, हम दिए गए स्थान में बाएं-औचित्य <, दाएं-औचित्य> या केंद्र ^ एक स्ट्रिंग कर सकते हैं। हम पूर्णांकों को बाइनरी, हेक्साडेसिमल आदि के रूप में भी प्रारूपित कर सकते हैं और फ्लोट को घातांक प्रारूप में गोल या प्रदर्शित किया जा सकता है। एक टन स्वरूपण है जिसका आप उपयोग कर सकते हैं। format () पद्धति के साथ उपलब्ध सभी स्ट्रिंग स्वरूपण के लिए यहां जाएं।

```
>>> # formatting integers

>>> "Binary representation of {0} is {0:b}".format(12)

'Binary representation of 12 is 1100'

>>> # formatting floats

>>> "Exponent representation: {0:e}".format(1566.345)

'Exponent representation: 1.566345e+03'

>>> # round off

>>> "One third is: {0:.3f}".format(1/3)

'One third is: 0.333'

>>> # string alignment

>>>

"|{:<10}|{:^10}|{:>10}|".format('butter','bread','ham')

'|butter | bread | ham|'
```

पुरानी शैली स्वरूपण

हम सी प्रोग्रामिंग भाषा में उपयोग की जाने वाली पुरानी स्प्रिंटफ () शैली की तरह स्ट्रिंग को भी प्रारूपित कर सकते हैं। हम उपयोग % ऑपरेटर इसे पूरा करने के लिए।

```
>>> x = 12.3456789

>>> print('The value of x is %3.2f' %x)

The value of x is 12.35

>>> print('The value of x is %3.4f' %x)

The value of x is 12.3457
```

सामान्य पायथन स्ट्रिंग के तरीके

स्ट्रिंग ऑब्जेक्ट के साथ कई विधियां उपलब्ध होती हैं। `format()` विधि जिसका हमने ऊपर उल्लेख किया है, उनमें से एक है। आमतौर पर इस्तेमाल की जाने वाली कुछ विधियां निम्न हैं `lower(), upper(), join(), split(), find(), replace()` आदि।

सेट्स का उपयोग कैसे करें

एक सेट वस्तुओं का एक अनियंत्रित संग्रह है। प्रत्येक तत्व अद्वितीय है (कोई डुप्लिकेट नहीं) और अपरिवर्तनीय होना चाहिए (जिसे बदला नहीं जा सकता)।

हालांकि, सेट ही परिवर्तनशील होते हैं। हम इसमें आइटम जोड़ या हटा सकते हैं।

समुच्चय का उपयोग गणितीय समुच्चय संचालन जैसे यूनियन , प्रतिच्छेदन, सममित अंतर आदि को करने के लिए किया जा सकता है।

एक सेट कैसे बनाएं?

सभी वस्तुओं (तत्वों) को घुंघराले ब्रेसिज़ {} के अंदर रखकर, अल्पविराम से अलग करके या अंतर्निहित फ़ंक्शन सेट () का उपयोग करके एक सेट बनाया जाता है।

इसमें किसी भी संख्या में आइटम हो सकते हैं और वे विभिन्न प्रकार के हो सकते हैं (पूर्णांक, फ्लोट, टपल, स्ट्रिंग आदि)। लेकिन एक सेट में एक परिवर्तनशील तत्व नहीं हो सकता है, जैसे सूची, सेट या शब्दकोश, इसके तत्व के रूप में होते हैं।

```
# set of integers
my_set = {1, 2, 3}
print(my_set)
# set of mixed datatypes
my_set = {1.0, "Hello", (1, 2, 3)}
print(my_set)
```

निम्नलिखित उदाहरणों को भी आजमा सकते हैं।

```
# set do not have duplicates
# Output: {1, 2, 3, 4}
my_set = {1,2,3,4,3,2}
print(my_set)
# set cannot have mutable items
# here [3, 4] is a mutable list
```

```
# If you uncomment line #12,
# this will cause an error.
# TypeError: unhashable type: 'list'
#my_set = {1, 2, [3, 4]}
# we can make set from a list
# Output: {1, 2, 3}
my_set = set([1,2,3,2])
print(my_set)
```

खाली सेट बनाना थोड़ा मुश्किल होता है।

खाली कर्ली ब्रेसिज़ {} पायथन में एक खाली शब्दकोश बना देंगे। बिना किसी तत्व के एक सेट बनाने के लिए हम बिना किसी तर्क के सेट () फ़ंक्शन का उपयोग करते हैं।

```
# initialize a with {}
a = {}
# check data type of a
# Output: <class 'dict'>
print(type(a))
# initialize a with set()
a = set()
# check data type of a
# Output: <class 'set'>
print(type(a))
```

पायथन में एक सेट कैसे बदलें

सेट परिवर्तनशील होते हैं। लेकिन चूंकि वे अव्यवस्थित होते हैं, इसलिए अनुक्रमण का कोई अर्थ नहीं है।

हम इंडेक्सिंग या स्लाइसिंग का उपयोग करके सेट के किसी तत्व को एक्सेस या बदल नहीं सकते हैं। सेट इसका समर्थन नहीं करता है।

हम का उपयोग करके एकल तत्व जोड़ सकते हैं `add()` विधि और कई तत्वों का उपयोग कर `up-date ()` विधि। `update()` विधि टपल , सूचियों, स्ट्रिंग्स या अन्य सेटों को अपने तर्क के रूप में ले सकती है। सभी मामलों में, डुप्लिकेट से बचा जाता है।

```
# initialize my_set
my_set = {1,3}
```

```
print(my_set)

# if you uncomment line 9,

# you will get an error

# TypeError: 'set' object does not support indexing

#my_set[0]

# add an element

# Output: {1, 2, 3}

my_set.add(2)

print(my_set)

# add multiple elements

# Output: {1, 2, 3, 4}

my_set.update([2,3,4])

print(my_set)

# add list and set

# Output: {1, 2, 3, 4, 5, 6, 8}

my_set.update([4,5], {1,6,8})

print(my_set)
```

जब आप प्रोग्राम चलाते हैं, तो आउटपुट ये होता है:

```
{1, 3}

{1, 2, 3}

{1, 2, 3, 4}

{1, 2, 3, 4, 5, 6, 8}
```

एक सेट से तत्वों को कैसे निकालें

एक विशेष आइटम को विधियों का उपयोग करके सेट से हटाया जा सकता है, `discard()` और `remove()`।

दोनों के बीच एकमात्र अंतर यह है कि, यदि आइटम मौजूद नहीं है, तो discard() का उपयोग करते समय सेट, यह अपरिवर्तित रहता है। लेकिन निकालें() ऐसी स्थिति में एक त्रुटि उत्पन्न करेगा।

निम्नलिखित उदाहरण इसे स्पष्ट करेगा।

```
# initialize my_set
```

```
my_set = {1, 3, 4, 5, 6}
print(my_set)
# discard an element
# Output: {1, 3, 5, 6}
my_set.discard(4)
print(my_set)
# remove an element
# Output: {1, 3, 5}
my_set.remove(6)
print(my_set)
# discard an element
# not present in my_set
# Output: {1, 3, 5}
my_set.discard(2)
print(my_set)
# remove an element
# not present in my_set
# If you uncomment line 27,
# you will get an error.
# Output: KeyError: 2
#my_set.remove(2)
```

इसी तरह, हम `pop ()` विधि का उपयोग करके किसी आइटम को हटा और वापस कर सकते हैं।

अनियंत्रित होने के कारण, यह निर्धारित करने का कोई तरीका नहीं है कि कौन सा आइटम पॉप किया जाएगा। यह पूरी तरह से मनमाना होता है।

हम `clear()` का उपयोग करके एक सेट से सभी आइटम्स को भी हटा सकते हैं।

```
# initialize my_set
# Output: set of unique elements
my_set = set("HelloWorld")
```

```
print(my_set)

# pop an element

# Output: random element

print(my_set.pop())

# pop another element

# Output: random element

my_set.pop()

print(my_set)

# clear my_set

#Output: set()

my_set.clear()

print(my_set)
```

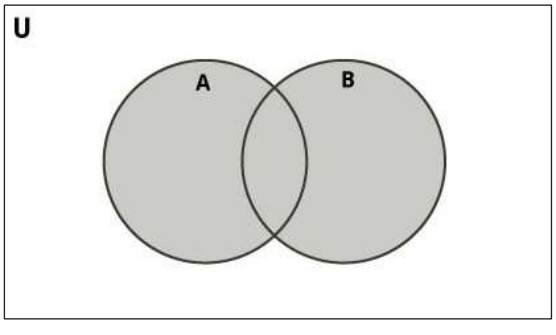
पायथन सेट ऑपरेशंस

समुच्चय का उपयोग गणितीय समुच्चय संचालन जैसे यूनियन , प्रतिच्छेदन, अंतर और सममित अंतर को पूरा करने के लिए किया जा सकता है। हम इसे ऑपरेटरों या विधियों के साथ कर सकते हैं।

आइए निम्नलिखित संक्रियाओं के लिए निम्नलिखित दो सेटों पर विचार करें:

```
>>> A = {1, 2, 3, 4, 5}

>>> B = {4, 5, 6, 7, 8}
```

सेट यूनियन



यूनियन ए तथा बी दोनों समुच्चयों के सभी तत्वों का समुच्चय होता है।

यूनियन का उपयोग करके किया जाता है | ऑपरेटर। यूनियन() विधि का उपयोग करके इसे पूरा किया जा सकता है।

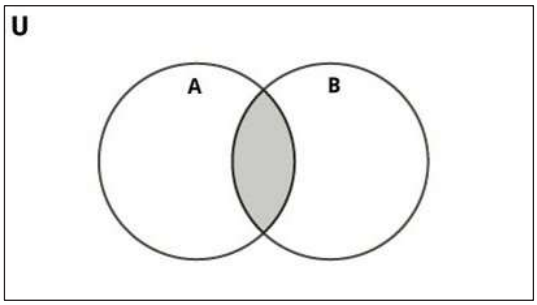
```
# initialize A and B
```

```
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
# use | operator
# Output: {1, 2, 3, 4, 5, 6, 7, 8}
print(A | B)
```

पायथन शेल पर निम्नलिखित उदाहरणों का प्रयास करें।

```
# use union function
>>> A.union(B)
{1, 2, 3, 4, 5, 6, 7, 8}
# use union function on B
>>> B.union(A)
{1, 2, 3, 4, 5, 6, 7, 8}
```

सेट इंटरसेक्शन



ए और बी का प्रतिच्छेदन तत्वों का एक समूह है जो दोनों सेटों में आम होता है।

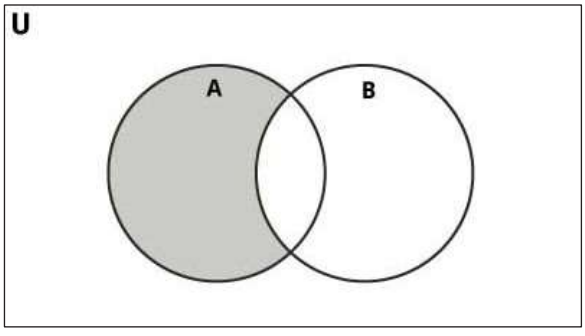
इंटरसेक्शन एंड ऑपरेटर का उपयोग करके किया जाता है। `intersection()` का उपयोग करके वही पूरा किया जा सकता है।

```
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
# use & operator
# Output: {4, 5}
print(A & B)
```

पायथन शेल पर निम्नलिखित उदाहरण आज़मा सकते हैं:

```
# use intersection function on A
>>> A.intersection(B)
{4, 5}
# use intersection function on B
>>> B.intersection(A)
{4, 5}
```

अंतर सेट करें



ए और बी का अंतर (ए - बी) तत्वों का एक सेट है जो केवल ए में है लेकिन बी में नहीं है। इसी तरह, बी - ए बी में तत्वों का एक सेट है लेकिन ए में नहीं।

अंतर - ऑपरेटर का उपयोग करके किया जाता है। विधि भिन्नता () का उपयोग करके इसे पूरा किया जा सकता है।

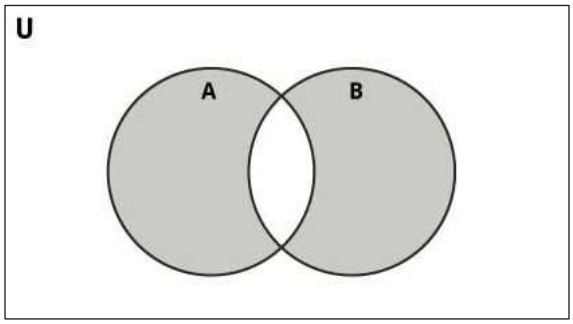
```
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
# use - operator on A
# Output: {1, 2, 3}
print(A - B)
```

पायथन शेल पर निम्नलिखित उदाहरण आज़माएं:

```
# use difference function on A
>>> A.difference(B)
{1, 2, 3}
# use - operator on B
```

```
>>> B - A
{8, 6, 7}
# use difference function on B
>>> B.difference(A)
{8, 6, 7}
```

सममित अंतर सेट करें



ए और बी का सममित अंतर ए और बी दोनों में तत्वों का एक समूह है, सिवाय उन तत्वों के जो सामान्य हैं दोनों मे।

^ ऑपरेटर का उपयोग करके सममित अंतर किया जाता है। मेथड `symmetric_difference()` का उपयोग करके इसे पूरा किया जा सकता है।

```
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
# use ^ operator
# Output: {1, 2, 3, 6, 7, 8}
print(A ^ B)
```

पायथन शेल पर निम्नलिखित उदाहरण आज़माएं:

```
# use symmetric_difference function on A
>>> A.symmetric_difference(B)
```

{1, 2, 3, 6, 7, 8}

```
# use symmetric_difference function on B
>>> B.symmetric_difference(A)
{1, 2, 3, 6, 7, 8}
```

विभिन्न पायथन सेट के तरीके

कई सेट विधियां होती हैं, जिनमें से कुछ हम पहले ही ऊपर उपयोग कर चुके हैं। यहां उन सभी विधियों की सूची दी गई है जो सेट ऑब्जेक्ट्स के साथ उपलब्ध हैं।

पायथन सेट मेथड्स	
तरीका	विवरण
जोड़ें()	सेट में एक तत्व जोड़ता है
स्पष्ट()	सेट से सभी तत्वों को हटाता है
कॉपी ()	सेट की एक प्रति लौटाता है
अंतर()	नए सेट के रूप में दो या दो से अधिक सेट का अंतर लौटाता है
डिफरेंस_अपडेट ()	इस सेट से दूसरे सेट के सभी तत्वों को हटाता है
रद्द करें()	यदि यह सदस्य है तो सेट से किसी तत्व को हटा देता है। (यदि तत्व सेट में नहीं है तो कुछ न करें)
चौराहा ()	दो सेटों के प्रतिच्छेदन को एक नए सेट के रूप में लौटाता है
चौराहा_अपडेट ()	अपने और दूसरे के प्रतिच्छेदन के साथ सेट को अपडेट करता है
इसडिसज्वाइंट ()	रिटर्न सत्य यदि दो सेटों में एक अशक्त चौराहा है
सबसेट ()	रिटर्न सत्य यदि किसी अन्य सेट में यह सेट है
इससुपरसेट ()	रिटर्न सत्य अगर इस सेट में एक और सेट है
पॉप()	एक मनमाना सेट तत्व को हटाता है और लौटाता है। चढ़ाईमुख्य त्रुटि अगर सेट खाली है
हटाना()	सेट से एक तत्व निकालता है। यदि तत्व सदस्य नहीं है, तो बढ़ाएँ ए मुख्य त्रुटि
सममित_अंतर ()	दो सेटों के सममित अंतर को एक नए सेट के रूप में लौटाता है
सममित_अंतर_अपडेट ()	अपने और दूसरे के सममित अंतर के साथ एक सेट को अपडेट करता है
यूनियन ()	एक नए सेट में सेट का यूनियन लौटाता है
अपडेट करें()	अपने और दूसरों के मिलन के साथ सेट को अपडेट करता है

अन्य सेट संचालन

सदस्यता परीक्षण सेट करें

हम कीवर्ड का उपयोग करके यह जांच सकते हैं कि कोई आइटम सेट में मौजूद है या नहीं।

```
# initialize my_set

my_set = set("apple")

# check if 'a' is present

# Output: True

print('a' in my_set)
```

```
# check if 'p' is present

# Output: False

print('p' not in my_set)
```

एक सेट के माध्यम से पुनरावृति

लूप के लिए उपयोग करके, हम सेट में प्रत्येक आइटम के बावजूद पुनरावृति कर सकते हैं।

```
>>> for letter in set("apple"):

... print(letter)

...

a

p

e

l
```

सेट के साथ अंतर्निहित कार्य

`all()`, `any()`, `enumerate()`, `len()`, `max()`, `min()`, `sorted()`, `sum()` जैसे अंतर्निहित कार्यों आदि आमतौर पर विभिन्न कार्यों को करने के लिए सेट के साथ उपयोग किए जाते हैं।

सेट के साथ अंतर्निहित कार्य	
फंक्शन	विवरण
`all()`	वापसी सत्य यदि समुच्चय के सभी अवयव सत्य हैं (या यदि समुच्चय रिक्त है)।
`any()`	वापसी सत्य यदि समुच्चय का कोई अवयव सत्य है। यदि सेट खाली है, तो वापस लौटेंझूठा।
`enumerate()`	एक एन्यूमरेट ऑब्जेक्ट लौटाएं। इसमें जोड़े के रूप में सेट की सभी वस्तुओं का सूचकांक और मूल्य होता है।
`len()`	सेट में लंबाई (आइटम की संख्या) लौटाएं।
`max()`	सेट में सबसे बड़ी वस्तु लौटाएं।
`min()`	सेट में सबसे छोटी वस्तु लौटाएं।
`sorted()`	सेट में तत्वों से एक नई सॉर्ट की गई सूची लौटाएं (सेट को स्वयं सॉर्ट नहीं करता है)।
`sum()`	सेट में सभी तत्वों का योग वापस करें।

पायथन फ्रोजनसेट

फ्रोज़नसेट एक नया वर्ग है जिसमें एक सेट की विशेषताएं हैं, लेकिन इसके तत्वों को एक बार असाइन किए जाने के बाद बदला नहीं जा सकता है। जबकि टपल अपरिवर्तनीय सूचियां हैं, फ्रोजनसेट अपरिवर्तनीय सेट होते हैं।

परिवर्तनशील होने वाले सेट अप्राप्य होते हैं, इसलिए उन्हें शब्दकोश कुंजी के रूप में उपयोग नहीं किया जा सकता है। दूसरी ओर, फ्रोजनसेट धोने योग्य होते हैं और एक शब्दकोश की कुंजी के रूप में उपयोग किए जा सकते हैं।

फ्रोजनसेट फ़ंक्शन फ्रोजनसेट () का उपयोग करके बनाया जा सकता है।

यह डेटाटाइप `copy()`, `difference()`, `intersection()`, `isdisjoint()`, `issubset()`, `issuperset()`, `symmetric_difference()` and `union()`. जैसी विधियों का समर्थन करता है। अपरिवर्तनीय होने के कारण इसमें ऐसे तरीके नहीं हैं जो तत्वों को जोड़ते या हटाते हैं।

```
# initialize A and B

A = frozenset([1, 2, 3, 4])

B = frozenset([3, 4, 5, 6])
```

इन उदाहरणों को पायथन शेल पर आज़माएं:

```
>>> A.isdisjoint(B)

False

>>> A.difference(B)

frozenset({1, 2})

>>> A | B

frozenset({1, 2, 3, 4, 5, 6})

>>> A.add(3)

...

AttributeError: 'frozenset' object has no attribute 'add'
```

शब्दकोश का प्रयोग कैसे करें

पायथन डिक्शनरी वस्तुओं का एक अनियंत्रित संग्रह होता है। जबकि अन्य मिश्रित डेटा प्रकारों में केवल एक तत्व के रूप में मूल्य होता है, एक शब्दकोश में एक कुंजी होती है: मूल्य जोड़ी है।

जब कुंजी ज्ञात हो जाती है तो शब्दकोशों को मूल्यों को पुनः प्राप्त करने के लिए अनुकूलित किया जाता है।

एक शब्दकोष कैसे बनाएं

डिक्शनरी बनाना उतना ही आसान है जितना कि कॉमा से अलग कर्ली ब्रेसिज़ {} के अंदर आइटम रखना। एक आइटम में एक कुंजी और संबंधित मान एक

जोड़ी, कुंजी: मान के रूप में व्यक्त किया जाता है।

जबकि मान किसी भी डेटा प्रकार के हो सकते हैं और दोहरा सकते हैं, कुंजियाँ अपरिवर्तनीय प्रकार की होनी चाहिए (स्ट्रिंग, संख्या या अपरिवर्तनीय तत्वों के साथ टपल) और अद्वितीय होनी चाहिए।

```
# empty dictionary

my_dict = {}
```

```
# dictionary with integer keys
my_dict = {1: 'apple', 2: 'ball'}
# dictionary with mixed keys
my_dict = {'name': 'John', 1: [2, 4, 3]}
# using dict()
my_dict = dict({1:'apple', 2:'ball'})
# from sequence having each item as a pair
my_dict = dict([(1,'apple'), (2,'ball')])
```

जैसा कि आप ऊपर देख सकते हैं, हम बिल्ट-इन फंक्शन `dict()` का उपयोग करके एक डिक्शनरी भी बना सकते हैं।

शब्दकोश से तत्वों तक कैसे पहुँचें

जबकि इंडेक्सिंग का उपयोग अन्य कंटेनर प्रकारों के साथ मूल्यों तक पहुंचने के लिए किया जाता है, शब्दकोश कुंजियों का उपयोग करता है। कुंजी का उपयोग या तो वर्गाकार कोष्ठक के अंदर या `get ()` विधि के साथ किया जा सकता है।

`get()` का उपयोग करते समय अंतर यह है कि यदि कुंजी नहीं है, तो यह `KeyError` के बजाय कोई नहीं लौटाता है मिलता है।

```
my_dict = {'name':'Jack', 'age': 26}
# Output: Jack
print(my_dict['name'])
# Output: 26
print(my_dict.get('age'))
# Trying to access keys which doesn't exist throws error
# my_dict.get('address')
# my_dict['address']
```

जब आप प्रोग्राम चलाते हैं, तो आउटपुट होगा:

```
Jack
26
```

शब्दकोश में तत्वों को कैसे बदलें या जोड़ें?

शब्दकोश परिवर्तनशील होते हैं। हम असाइनमेंट ऑपरेटर का उपयोग करके नए आइटम इसमें जोड़ सकते हैं और मौजूदा आइटम के मूल्य को बदल सकते हैं।

यदि कुंजी पहले से मौजूद है, तो मान अपडेट हो जाता है अन्यथा एक नई कुंजी: मूल्य जोड़ी को शब्दकोश में जोड़ा जाता है।

```
my_dict = {'name':'Jack', 'age': 26}

# update value

my_dict['age'] = 27

#Output: {'age': 27, 'name': 'Jack'}

print(my_dict)

# add item

my_dict['address'] = 'Downtown'

# Output: {'address': 'Downtown', 'age': 27, 'name':

'Jack'} print(my_dict)
```

जब आप प्रोग्राम चलाते हैं, तो आउटपुट होगा:

```
{'name': 'Jack', 'age': 27}

{'name': 'Jack', 'age': 27, 'address': 'Downtown'}
```

शब्दकोश से तत्वों को कैसे बदलें या हटाएं

हम `pop ()` विधि का उपयोग करके शब्दकोश में किसी विशेष आइटम को हटा सकते हैं। यह विधि प्रदान की गई कुंजी के साथ आइटम के रूप में हटा देती है और मान लौटाती है।

विधि, `popitem()` का उपयोग किसी मनमानी वस्तु (कुंजी, मान) को हटाने और वापस करने के लिए किया जा सकता है। `clear ()` विधि का उपयोग करके सभी वस्तुओं को एक बार में हटाया जा सकता है।

हम अलग-अलग आइटम या संपूर्ण डिक्शनरी को हटाने के लिए भी `del` कीवर्ड का उपयोग कर सकते हैं।

```
# create a dictionary

squares = {1:1, 2:4, 3:9, 4:16, 5:25}

# remove a particular item

# Output: 16

print(squares.pop(4))

# Output: {1: 1, 2: 4, 3: 9, 5: 25}

print(squares)

# remove an arbitrary item

# Output: (1, 1)

print(squares.popitem())
```

```
# Output: {2: 4, 3: 9, 5: 25}
print(squares)
# delete a particular item
del squares[5]
# Output: {2: 4, 3: 9}
print(squares)
# remove all items
squares.clear()
# Output: {}
print(squares)
# delete the dictionary itself
del squares
# Throws Error
# print(squares)
```

जब आप प्रोग्राम चलाते हैं, तो आउटपुट होगा:

```
16
{1: 1, 2: 4, 3: 9, 5: 25}
(1, 1)
{2: 4, 3: 9, 5: 25}
{2: 4, 3: 9}
{}
```

पायथन शब्दकोश के तरीके

शब्दकोश के साथ उपलब्ध विधियों को नीचे सारणीबद्ध किया गया है। उनमें से कुछ का पहले ही उपरोक्त उदाहरणों में उपयोग किया जा चुका है।

पायथन डिक्शनरी के तरीके	
तरीका	विवरण
`clear()`	शब्दकोश के रूप में सभी वस्तुओं को हटा दें।
`copy()`	शब्दकोश की उथली प्रति लौटाएं।

fromkeys(seq[, v])	से कुंजियों वाला एक नया शब्दकोश लौटाएं स्व-परीक्षा प्रश्न और मूल्य बराबर वी(डिफ़ॉल्ट करने के लिए कोई नहीं)
get(key[,d])	का मान लौटाएं चाभी. अगरचाभी बाहर नहीं निकलता, वापसी डी (डिफ़ॉल्ट करने के लिए कोई नहीं)
items()	शब्दकोश के आइटम (कुंजी, मान) का एक नया दृश्य लौटाएं।
keys()	शब्दकोश की कुंजियों का एक नया दृश्य लौटाएं।
pop(key[,d])	के साथ आइटम निकालें चाभी और इसका मान लौटाएं या डी अगर चाभी नहीं मिला। अगरडी प्रदान नहीं किया गया है और चाभी नहीं मिला, उठाता है मुख्य त्रुटि.
popitem()	एक मनमाना आइटम (कुंजी, मान) निकालें और वापस करें। जन्म देती हैमुख्य त्रुटि अगर शब्दकोश है खाली।
setdefault(key[, d])	अगर चाभी शब्दकोश में है, उसका मान लौटाएं। यदि नहीं, तो डालेंचाभी के मान के साथ डी और वापस डी (डिफ़ॉल्ट करने के लिए कोई नहीं)
update([other])	से कुंजी/मान जोड़े के साथ शब्दकोश को अपडेट करें अन्य, मौजूदा कुंजियों को अधिलेखित करना।
values()	शब्दकोश के मूल्यों का एक नया दृश्य लौटाएं

इन विधियों के उपयोग के कुछ उदाहरण यहां दिए गए हैं:

```
marks = {}.fromkeys(['Math','English','Science'], 0)
# Output: {'English': 0, 'Math': 0, 'Science':
0} print(marks)
for item in marks.items():
     print(item)
# Output: ['English', 'Math', 'Science']
list(sorted(marks.keys()))
```

पायथन डिक्शनरी कॉम्प्रिहेंशन

डिक्शनरी कॉम्प्रिहेंशन पायथन में एक पुनरावृत्त से नया शब्दकोश बनाने का एक सुंदर और संक्षिप्त तरीका है।

डिक्शनरी कॉम्प्रिहेंशन में एक एक्सप्रेशन पेयर (कुंजी: वैल्यू) होता है जिसके बाद फॉरस्टेटमेंट होता है कर्ली ब्रेसिज़ के अंदर {}।

प्रत्येक आइटम को एक संख्या और उसके वर्ग की एक जोड़ी होने के साथ एक शब्दकोश बनाने के लिए यहां एक उदाहरण दिया गया है।

```
squares = {x: x*x for x in range(6)}
# Output: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
print(squares)
```

यह कोड इसके बराबर है:

```
squares = {}
```

```
for x in range(6):
      squares[x] = x*x
```

एक शब्दकोश समझ में वैकल्पिक रूप से कथन के लिए या यदि अधिक हो सकता है,

एक वैकल्पिक अगर कथन नया शब्दकोश बनाने के लिए आइटम को फ़िल्टर कर सकता है।

केवल विषम वस्तुओं से शब्दकोश बनाने के लिए यहां कुछ उदाहरण दिए गए हैं।

```
odd_squares = {x: x*x for x in range(11) if x%2 == 1}
# Output: {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
print(odd_squares)
```

अन्य शब्दकोश संचालन

शब्दकोश सदस्यता परीक्षण

हम जांच सकते हैं कि कोई कुंजी शब्दकोश में है या नहीं। ध्यान दें कि सदस्यता परीक्षण केवल कुंजियों के लिए है, मूल्यों के लिए नहीं।

```
squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
# Output: True
print(1 in squares)
# Output: True
print(2 not in squares)
# membership tests for key only not value
# Output: False
print(49 in squares)
```

एक शब्दकोश के माध्यम से पुनरावृत्ति

लूप के लिए हम शब्दकोश में प्रत्येक कुंजी के माध्यम से पुनरावृति कर सकते हैं।

```
squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
for i in squares:
      print(squares[i])
```

शब्दकोश के साथ अंतर्निहित कार्य

`all()`, `any()`, `len()`, `cmp()`, `sorted()` आदि जैसे अंतर्निहित कार्यों को आमतौर पर विभिन्न कार्यों को करने के लिए शब्दकोश के साथ उपयोग किया जाता है।

शब्दकोश के साथ अंतर्निहित कार्य	
फंक्शन	विवरण
`all()`	वापसी `true` यदि शब्दकोश की सभी कुंजियाँ सत्य हैं (या यदि शब्दकोश खाली है)।
`any()`	वापसी `true` यदि शब्दकोश की कोई कुंजी सत्य है। यदि शब्दकोश खाली है, तो वापस लौटें `false`।
`len()`	शब्दकोश में लंबाई (वस्तुओं की संख्या) लौटाएं।
`cmp()`	दो शब्दकोशों की वस्तुओं की तुलना करता है।
`sorted()`	शब्दकोश में चाबियों की एक नई क्रमबद्ध सूची लौटाएं।

यहां कुछ उदाहरण दिए गए हैं जो डिक्शनरी के साथ काम करने के लिए बिल्ट-इन फ़ंक्शंस का उपयोग करते हैं।

```
squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}

# Output: 5

print(len(squares))

# Output: [1, 3, 5, 7, 9]

print(sorted(squares))
```

4

पायथन में ऑपरेटिंग फाइल्स

पायथन फाइलों को संभालने के लिए एक इन-बिल्ट फंक्शन प्रदान करता है और उपयोगकर्ताओं को फाइल बनाने, पढ़ने, लिखने और संचालित करने की अनुमति देता है। इसमें निर्देशिकाओं तक पहुंच और उपयोगकर्ता-परिभाषित अपवाद शामिल हैं। यह अध्याय विषय की व्यापक समझ प्रदान करने के लिए पायथन की ऑपरेटिंग फाइलों के इन पहलुओं की बारीकी से जांच करता है।

फ़ाइल संचालन का उपयोग कैसे करें

फ़ाइल संबंधित जानकारी संग्रहीत करने के लिए डिस्क पर एक नामित स्थान होता है। इसका उपयोग डेटा को नॉन-वोलाटाइल मेमोरी (जैसे हार्ड डिस्क) में स्थायी रूप से संग्रहीत करने के लिए किया जाता है।

चूंकि, रैंडम एक्सेस मेमोरी (रैम) अस्थिर होती है जो कंप्यूटर बंद होने पर अपना डेटा खो देती है, हम डेटा के भविष्य के उपयोग के लिए फाइलों का उपयोग करते हैं।

जब हम किसी फाइल से पढ़ना या लिखना चाहते हैं तो हमें उसे पहले खोलना होता है। जब हम ऐसा कर लेते हैं, तो इसे बंद करने की आवश्यकता होती है, ताकि फ़ाइल से जुड़े संसाधन मुक्त हो जाएं।

इसलिए, पायथन में, एक फ़ाइल ऑपरेशन निम्न क्रम में होता है।

- एक फ़ाइल खोलो
- पढ़ें या लिखें (ऑपरेशन करें)
- फ़ाइल बंद करें

फ़ाइल कैसे खोलें

फ़ाइल खोलने के लिए पायथन में एक अंतर्निहित फ़ंक्शन `open()` होता है। यह फ़ंक्शन एक फ़ाइल ऑब्जेक्ट देता है, जिसे हैंडल भी कहा जाता है, क्योंकि इसका उपयोग फ़ाइल को तदनुसार पढ़ने या संशोधित करने के लिए किया जाता है।

```
>>> f = open("test.txt") # open file in current directory
>>> f = open("C:/Python33/README.txt") # specifying full path
```

हम खोलते समय मोड निर्दिष्ट कर सकते हैं एक पंक्ति। मोड में, हम निर्दिष्ट करते हैं कि क्या हम फ़ाइल में 'r' पढ़ना चाहते हैं, 'w' लिखना चाहते हैं या 'a' जोड़ना चाहते हैं। हम यह भी निर्दिष्ट करते हैं कि क्या हम फ़ाइल को टेक्स्ट मोड या बाइनरी मोड में खोलना चाहते हैं।

डिफ़ॉल्ट पाठ मोड में पढ़ रहा है। इस मोड में, हमें फ़ाइल से पढ़ते समय तार मिलते हैं।

दूसरी ओर, बाइनरी मोड बाइट्स देता है और छवि या exe फ़ाइलों जैसी गैर-पाठ फ़ाइलों के साथ काम करते समय इस मोड का उपयोग किया जाता है।

पायथन फ़ाइल मोड	
तरीका	विवरण

'r'	पढ़ने के लिए एक फ़ाइल खोलें। (चूक जाना)
'w'	लिखने के लिए एक फाइल खोलें। यदि यह मौजूद नहीं है तो एक नई फ़ाइल बनाता है या मौजूद होने पर फ़ाइल को छोटा कर देता है।
'x'	अनन्य निर्माण के लिए एक फ़ाइल खोलें। यदि फ़ाइल पहले से मौजूद है, तो कार्रवाई विफल हो जाती है।
'a'	फ़ाइल को काटे बिना फ़ाइल के अंत में जोड़ने के लिए खोलें। यदि यह मौजूद नहीं है तो एक नई फ़ाइल बनाता है।
't'	टेक्स्ट मोड में खोलें। (चूक जाना)
'b'	बाइनरी मोड में खोलें।
'+'	अद्यतन करने के लिए फ़ाइल खोलें (पढ़ना और लिखना)

```
f = open("test.txt") # equivalent to 'r' or 'rt'
f = open("test.txt",'w') # write in text mode
f = open("img.bmp",'r+b') # read and write in binary mode
```

अन्य भाषाओं के विपरीत, वर्ण 'ए' 97 की संख्या को तब तक नहीं दर्शाता है जब तक कि इसका उपयोग करके एन्कोड नहीं किया जाता है एएससीआईआई (या अन्य समकक्ष एन्कोडिंग)।

इसके अलावा, डिफ़ॉल्ट एन्कोडिंग प्लेटफॉर्म पर निर्भर है। विंडोज़ में, यह `'cp1252'` है लेकिन `'utf-8'` है लिनक्स में।

इसलिए, हमें डिफ़ॉल्ट एन्कोडिंग पर भी भरोसा नहीं करना चाहिए अन्यथा हमारा कोड अलग-अलग प्लेटफॉर्म पर अलग-अलग व्यवहार करता है।

इसलिए, टेक्स्ट मोड में फाइलों के साथ काम करते समय, एन्कोडिंग प्रकार निर्दिष्ट करने की अत्यधिक अनुशंसा की जाती है।

```
f = open("test.txt",mode = 'r',encoding = 'utf-8')
```

पायथन का उपयोग करके किसी फ़ाइल को कैसे बंद करें

जब हम फ़ाइल के संचालन के साथ होते हैं, तो हमें फ़ाइल को ठीक से बंद करने की आवश्यकता होती है।

एक फ़ाइल को बंद करने से उन संसाधनों को मुक्त कर दिया जाएगा जो फ़ाइल से बंधे थे और पायथन `close ()` विधि का उपयोग करके ऐसा किया जाता है।

पाइथन में गैर-संदर्भित वस्तुओं को साफ करने के लिए एक कचरा संग्रहकर्ता होता है, लेकिन हमें फ़ाइल को बंद करने के लिए उस पर भरोसा नहीं करना चाहिए।

```
f = open("test.txt",encoding = 'utf-8')
# perform file operations
f.close()
```

यह तरीका पूरी तरह से सुरक्षित नहीं होता है। यदि कोई ऑपरेशन करते समय कोई अपवाद होता है फ़ाइल के साथ, कोड फ़ाइल को बंद किए बिना बाहर निकल जाता है। एक सुरक्षित तरीका यह है कि ट्राई अंत में ब्लॉक का उपयोग किया जाता है।

```
try:
      f = open("test.txt",encoding = 'utf-8')
      # perform file operations
finally:
      f.close()
```

इस तरह, हमें गारंटी दी जाती है कि अपवाद उठाए जाने पर भी फ़ाइल ठीक से बंद हो जाती है, जिससे प्रोग्राम प्रवाह रुक जाता है।

ऐसा करने का सबसे अच्छा तरीका के साथ कथन का उपयोग करता है। यह सुनिश्चित करता है कि फ़ाइल बंद हो जाती है जब अंदर के ब्लॉक से बाहर निकलता है।

हमें स्पष्ट रूप से `close ()` विधि को कॉल करने की आवश्यकता नहीं है। यह आंतरिक रूप से किया जाता है।

```
with open("test.txt",encoding = 'utf-8') as f:
      # perform file operations
```

पायथन का उपयोग करके फ़ाइल में कैसे लिखें

पायथन में एक फ़ाइल में लिखने के लिए, हमें इसे `'w'` लिखकर खोलना होगा, `'a'` या अनन्य जोड़ना होगा निर्माण `'x'` मोड।

हमें 'w' मोड से सावधान रहने की आवश्यकता है क्योंकि यदि यह पहले से मौजूद है तो यह फ़ाइल में अधिलेखित कर देगा। पिछले सभी डेटा मिटा दिए जाते हैं।

बाइट्स की एक स्ट्रिंग या अनुक्रम लिखना (बाइनरी फाइलों के लिए) `write ()` विधि का उपयोग करके किया जाता है। यह विधि फ़ाइल में लिखे गए वर्णों की संख्या लौटाती है।

```
with open("test.txt",'w',encoding = 'utf-8') as f:
      f.write("my first file\n")
      f.write("Thisfile\n\n")
      f.write("contains three lines\n")
```

यदि यह मौजूद नहीं है तो यह प्रोग्राम `'test.txt'` नाम की एक नई फ़ाइल बनाएगा। यदि यह अस्तित्व में है, तो यह अधिलेखित है- दस.

अलग-अलग पंक्तियों में अंतर करने के लिए हमें स्वयं न्यूलाइन वर्णों को शामिल करना चाहिए।

पायथन में फाइलें कैसे पढ़ें

पायथन में किसी फ़ाइल को पढ़ने के लिए हमें फ़ाइल को रीडिंग मोड में खोलना पड़ता है।

इस उद्देश्य के लिए विभिन्न विधियां उपलब्ध हैं। हम डेटा की `size` संख्या में पढ़ने के लिए `read (size)` विधि का उपयोग कर सकते हैं। यदि `size` पैरामीटर निर्दिष्ट नहीं है, तो यह फ़ाइल के अंत तक पढ़ता है और वापस आता है।

```
>>> f = open("test.txt",'r',encoding = 'utf-8')
>>> f.read(4) # read the first 4 data
'This'
>>> f.read(4) # read the next 4 data
' is '
>>> f.read() # read in the rest till end of file
'my first file\nThis file\ncontains three lines\n'
>>> f.read() # further reading returns empty
sting ''
```

हम देख सकते हैं कि `read ()` विधि `'\ n'` के रूप में नई लाइन लौटाती है। फ़ाइल के अंत तक पहुँच जाने के बाद, हम आगे पढ़ने पर खाली स्ट्रिंग प्राप्त करते हैं।

हम अपने वर्तमान फ़ाइल कर्सर (स्थिति) को `seek ()` विधि का उपयोग करके बदल सकते हैं। इसी तरह, `tell ()` विधि हमारी वर्तमान स्थिति (बाइट्स की संख्या में) लौटाती है।

```
>>> f.tell() # get the current file position
56
>>> f.seek(0) # bring file cursor to initial position
0
>>> print(f.read()) # read the entire file
This is my first file
This file
contains three lines
```

हम लूप के लिए एक फ़ाइल को लाइन-दर-लाइन पढ़ सकते हैं। यह कुशल और तेज दोनों होती है।

```
>>> for line in f:
... print(line, end = '')
...
This is my first file
Thisfile
contains three lines
```

फ़ाइल की पंक्तियों में ही एक न्यूलाइन वर्ण '\n' होता है।

इसके अलावा, प्रिंट करते समय दो नई पंक्तियों से बचने के लिए `print ()` अंतिम पैरामीटर है।

वैकल्पिक रूप से हम फ़ाइल की अलग-अलग पंक्तियों को पढ़ने के लिए `readline ()` विधि का उपयोग कर सकते हैं। यह विधि न्यूलाइन कैरेक्टर सहित, न्यूलाइन तक एक फाइल को पढ़ती है।

```
>>> f.readline()
'This is my first file\n'
>>> f.readline()
'This file\n'
>>> f.readline()
'contains three lines\n'
>>> f.readline()
''
```

अंत में, `readline ()` विधि संपूर्ण फ़ाइल की शेष पंक्तियों की एक सूची लौटाती है। ये सभी पढ़ते हैं- आईएनजी विधि फ़ाइल के अंत (ईओएफ) तक पहुंचने पर खाली मान लौटाती है।

```
>>> f.readlines()
['This is my first file\n', 'This file\n', 'contains three lines\n']
```

पायथन फ़ाइल मेथड्स

फ़ाइल ऑब्जेक्ट के साथ विभिन्न विधियाँ उपलब्ध होती हैं। उनमें से कुछ का उपयोग उपरोक्त उदाहरणों में किया गया है।

संक्षिप्त विवरण के साथ टेक्स्ट मोड में विधियों की पूरी सूची यहां दी गई है।

पायथन फ़ाइल तरीके	
तरीका	विवरण
`close()`	एक खुली फ़ाइल बंद करें। यदि फ़ाइल पहले ही बंद है तो इसका कोई प्रभाव नहीं पड़ता है।
`detach()`	अंतर्निहित बाइनरी बफर को से अलग करें टेक्स्टआईओबेस और इसे वापस करो।
`fileno()`	फ़ाइल की एक पूर्णांक संख्या (फ़ाइल डिस्क्रिप्टर) लौटाएँ।
`flush()`	फ़ाइल स्ट्रीम के राइट बफर को फ्लश करें।
`isatty()`	वापसी सत्य अगर फ़ाइल स्ट्रीम इंटरैक्टिव है।
`read(n)`	ज्यादा से ज्यादा पढ़ें एन वर्ण फ़ाइल बनाते हैं। फ़ाइल के अंत तक पढ़ता है अगर यह नकारात्मक है या कोई नहीं।
`readable()`	रिटर्न सत्य अगर फ़ाइल स्ट्रीम से पढ़ा जा सकता है।
`readline(n =-1)`	फ़ाइल से एक पंक्ति पढ़ें और वापस करें। ज्यादा से ज्यादा पढ़ता हैएन निर्दिष्ट होने पर बाइट्स।

readlines (n =-1)	फ़ाइल से लाइनों की सूची पढ़ें और वापस करें। ज्यादा से ज्यादा पढ़ता हैएन बाइट्स/चार-अभिनेता यदि निर्दिष्ट हैं।
seek (offset,from=SEEK_SET)	फ़ाइल की स्थिति को बदलें ओफ़्सेट बाइट्स, के संदर्भ में से (प्रारंभ, वर्तमान, अंत)।
seekable ()	रिटर्न सत्य अगर फ़ाइल स्ट्रीम यादृच्छिक अभिगम का समर्थन करती है।
tell()	वर्तमान फ़ाइल स्थान लौटाता है।
truncate (size=None)	फ़ाइल स्ट्रीम का आकार बदलें आकार बाइट्स। अगरआकार निर्दिष्ट नहीं है, वर्तमान स्थान का आकार बदलें।
writable()	रिटर्न सत्य अगर फ़ाइल स्ट्रीम को लिखा जा सकता है।
write (s)	स्ट्रिंग लिखें एस फ़ाइल में और लिखे गए वर्णों की संख्या लौटाएँ।
writelines (lines)	की एक सूची लिखें पंक्तियां फ़ाइल को।

निर्देशिका तक कैसे पहुंचे

यदि आपके पायथन प्रोग्राम में बड़ी संख्या में फाइलें हैं, तो आप चीजों को और अधिक प्रबंधनीय बनाने के लिए अपने कोड को विभिन्न निर्देशिकाओं में व्यवस्थित कर सकते हैं।

एक निर्देशिका या फ़ोल्डर फाइलों और उप निर्देशिकाओं का एक संग्रह होता है। पायथन में ओएस मॉड्यूल होता है, जो हमें निर्देशिकाओं (और फाइलों के साथ) के साथ काम करने के लिए कई उपयोगी तरीके प्रदान करता है।

वर्तमान निर्देशिका को कैसे प्राप्त करें

हम `getcwd ()` विधि का उपयोग करके वर्तमान कार्यशील निर्देशिका प्राप्त कर सकते हैं।

यह विधि वर्तमान कार्यशील निर्देशिका को एक स्ट्रिंग के रूप में लौटाती है। हम का भी उपयोग कर सकते हैं getcwdb() विधि इसे बाइट्स ऑब्जेक्ट के रूप में प्राप्त करने के लिए है।

```
>>> import os
>>> os.getcwd()
'C:\\Program Files\\PyScripter'
>>> os.getcwdb()
b'C:\\Program Files\\PyScripter'
```

अतिरिक्त बैकस्लैश का तात्पर्य एस्केप सीक्वेंस से है। `print ()` फ़ंक्शन इसे ठीक से प्रस्तुत करता है।

```
>>> print(os.getcwd())
C:\Program Files\PyScripter
```

निर्देशिका बदलना

हम `chdir ()` विधि का उपयोग करके वर्तमान कार्यशील निर्देशिका को बदल सकते हैं। जिस नए पथ को हम बदलना चाहते हैं, उसे इस पद्धति के लिए एक स्ट्रिंग के रूप में आपूर्ति की जानी चाहिए। पथ तत्वों को अलग करने के लिए हम फॉरवर्ड स्लैश (/) या बैकवर्ड स्लैश (\) दोनों का उपयोग कर सकते हैं।

बैकवर्ड स्लैश का उपयोग करते समय एस्केप सीक्वेंस का उपयोग करना सुरक्षित होता है।

```
>>> os.chdir('C:\\Python33')
>>> print(os.getcwd())
C:\Python33
```

निर्देशिका और फाइलों की सूची

एक निर्देशिका के अंदर सभी फाइलों और उप निर्देशिकाओं को `listdir ()` विधि का उपयोग करके जाना जा सकता है।

यह विधि एक पथ लेती है और उस पथ में उप निर्देशिकाओं और फ़ाइलों की एक सूची तैयार करके वापस देती है। यदि कोई पथ निर्दिष्ट नहीं होता है, तो यह वर्तमान कार्यशील निर्देशिका से वापस आ जाता है।

```
>>> print(os.getcwd())
C:\Python33
>>> os.listdir()
['DLLs',
'Doc',
'include',
'Lib',
'libs',
'LICENSE.txt',
'NEWS.txt',
'python.exe',
'pythonw.exe',
'README.txt',
'Scripts',
'tcl',
'Tools']
>>> os.listdir('G:\\')
```

```
['$RECYCLE.BIN',
'Movies',

'Music',

'Photos',

'Series',

'System Volume Information']
```

एक नई निर्देशिका बनाना

हम `mkdir ()` विधि का उपयोग करके एक नई निर्देशिका बना सकते हैं।

यह विधि नई निर्देशिका का पथ लेती है। यदि पूर्ण पथ निर्दिष्ट नहीं है, तो यह वर्तमान कार्यशील निर्देशिका में नई निर्देशिका बनाई जाती है।

```
>>> os.mkdir('test')

>>> os.listdir()

['test']
```

निर्देशिका या फ़ाइल का नाम बदलना

`rename ()` विधि किसी निर्देशिका या फ़ाइल का नाम बदल सकती है।

पहला तर्क पुराना नाम है और दूसरा तर्क के रूप में नया नाम आपूर्ति होना चाहिए।

```
>>> os.listdir()

['test']

>>> os.rename('test','new_one')

>>> os.listdir()

['new_one']
```

निर्देशिका या फ़ाइल को हटाना

किसी फ़ाइल को `remove ()` विधि का उपयोग करके हटाया (हटाया) जा सकता है। इसी तरह, `rmdir()`

यह विधि एक खाली निर्देशिका को हटा देती है।

```
>>> os.listdir()

['new_one', 'old.txt']

>>> os.remove('old.txt')

>>> os.listdir()

['new_one']

>>> os.rmdir('new_one')
```

```
>>> os.listdir()
[]
```

हालांकि इसमें ध्यान दें कि `rmdir()` विधि केवल खाली निर्देशिकाओं को हटा सकती है।

एक गैर-रिक्त निर्देशिका को हटाने के लिए हम शटिल मॉड्यूल के अंदर `rmtree()` विधि का उपयोग कर सकते हैं।

अपवाद तक कैसे पहुँचें

प्रोग्राम लिखते समय, हम अक्सर त्रुटियों का सामना करते हैं।

भाषा की उचित संरचना (वाक्यविन्यास) का पालन न करने के कारण होने वाली त्रुटि को वाक्य रचना त्रुटि या पार्सिंग त्रुटि कहा जाता है।

```
>>> if a < 3
      File "<interactive input>", line 1
      if a < 3
           ^
SyntaxError: invalid syntax
```

हम यहां देख सकते हैं कि `if` स्टेटमेंट में एक कोलन गायब होता है।

त्रुटियाँ रनटाइम पर भी हो सकती हैं और इन्हें अपवाद कहा जाता है। जैसे उदाहरण के लिए जब हम जिस फ़ाइल को खोलने का प्रयास करते हैं तो वह मौजूद नहीं होती है (`FileNotFoundError`), किसी संख्या को शून्य (`ZeroDivi- Error`) से विभाजित करते हुए, जिस मॉड्यूल को हम आयात करने का प्रयास करते हैं वह नहीं मिलता है (`ImportError`) आदि।

जब भी इस प्रकार की रनटाइम त्रुटि होती है तो पायथन एक अपवाद वस्तु बनाता है। यदि ठीक से संभाला नहीं जाता है, तो यह उस त्रुटि के बारे में कुछ विवरणों के साथ उस त्रुटि के लिए एक ट्रेसबैक प्रिंट करता है कि वह त्रुटि क्यों हुई है।

```
>>> 1 / 0
Traceback (most recent call last):
      File "<string>", line 301, in runcode
      File "<interactive input>", line 1, in
<module> ZeroDivisionError: division by zero
>>> open("imaginary.txt")
Traceback (most recent call last):
      File "<string>", line 301, in runcode
      File "<interactive input>", line 1, in <module>
```

```
FileNotFoundError: [Errno 2] No such file or directory: 'imaginary.txt'
```

पायथन बिल्ट-इन अपवाद

अवैध संचालन अपवाद बढ़ा सकते हैं। पायथन में बहुत सारे अंतर्निहित अपवाद होते हैं जो संबंधित त्रुटियों के होने पर उठाए जाते हैं। हम सभी बिल्ट-इन अपवादों को `local ()` बिल्ट-इन फंक्शन्स का उपयोग करके निम्न प्रकार से देख सकते हैं।

```
>>> locals()['  builtins  ']
```

यह हमें अंतर्निहित अपवादों, कार्यों और विशेषताओं का एक शब्दकोश लौटाता है।

पायथन प्रोग्रामिंग में कुछ सामान्य बिल्ट-इन अपवादों के साथ-साथ इसके कारण होने वाली त्रुटि को नीचे सारणीबद्ध किया गया है।

पायथन बिल्ट-इन अपवाद	
अपवाद	त्रुटि का कारण
अभिकथन त्रुटि	उठाया जब ज़ोर कथन विफल रहता है।
विशेषता त्रुटि	विशेषता असाइनमेंट या संदर्भ विफल होने पर उठाया गया है।
ईओएफत्रुटि	उठाया जब इनपुट () फ़ंक्शन एंड-ऑफ़-फ़ाइल स्थिति को हिट करता है।
फ़्लोटिंगपॉइंट त्रुटि	फ्लोटिंग पॉइंट ऑपरेशन विफल होने पर उठाया जाता है।
जनरेटरबाहर निकलें	उठाएँ जब एक जनरेटर बंद करे() विधि कहा जाता है।
आयात त्रुटि	आयातित मॉड्यूल नहीं मिलने पर उठाया गया है।
अनुक्रमणिकात्रुटि	किसी अनुक्रम का सूचकांक सीमा से बाहर होने पर उठाया जाता है।
मुख्य त्रुटि	जब शब्दकोश में कोई कुंजी नहीं मिलती है तो उठाया जाता है।
कीबोर्ड इंटरप्ट	जब उपयोगकर्ता इंटरप्ट कुंजी दबाता है (Ctrl+c या हटाएं) तब उठाया जाता है।
मेमोरी एरर	जब कोई ऑपरेशन मेमोरी से बाहर हो जाता है तो उठाया जाता है।
नाम त्रुटि	स्थानीय या वैश्विक दायरे में कोई चर नहीं मिलने पर उठाया जाता है।
लागू नहीं किया गयात्रुटि	अमूर्त विधियों द्वारा उठाया गया।
ओएसत्रुटि	जब सिस्टम ऑपरेशन सिस्टम से संबंधित त्रुटि का कारण बनता है तो उठाया जाता है।
अतिप्रवाह त्रुटि	एक अंकगणितीय संक्रिया के परिणाम को प्रदर्शित करने के लिए बहुत बड़ा होने पर उठाया जाता है।
संदर्भ त्रुटि	जब एक कमजोर संदर्भ प्रॉक्सी का उपयोग कचरा एकत्रित संदर्भ तक पहुंचने के लिए किया जाता है तो उठाया जाता है।
रनटाइम त्रुटि	जब कोई त्रुटि किसी अन्य श्रेणी के अंतर्गत नहीं आती है तो उठाया जाता है।
स्टॉप इटरेशन	द्वारा उठाया अगला() यह इंगित करने के लिए कार्य करता है कि इटरेटर द्वारा वापस करने के लिए कोई और आइटम नहीं है।
वक्य रचना त्रुटि	सिंटैक्स त्रुटि का सामना करने पर पार्सर द्वारा उठाया गया है।
इंडेंटेशन त्रुटि	गलत इंडेंटेशन होने पर उठाया गया है।
टैब एरर	इंडेंटेशन में असंगत टैब और रिक्त स्थान होने पर उठाया गया है।
सिस्टम में गड़बड़ी	जब दुभाषिया आंतरिक त्रुटि का पता लगाता है तो उठाया जाता है।

सिस्टम से बाहर निकलें	द्वारा उठाया `sys.exit ()` समारोह।
त्रुटि प्रकार	जब कोई फ़ंक्शन या ऑपरेशन गलत प्रकार के ऑब्जेक्ट पर लागू होता है तो उठाया जाता है।
अनबाउंडलोकल एरर	जब किसी फ़ंक्शन या विधि में स्थानीय चर के लिए एक संदर्भ बनाया जाता है, लेकिन उस चर के लिए कोई मान बाध्य नहीं होता है।
यूनिकोड त्रुटि	यूनिकोड से संबंधित एन्कोडिंग या डिकोडिंग त्रुटि होने पर उठाया जाता है।
यूनिकोडएनकोडत्रुटि	एन्कोडिंग के दौरान यूनिकोड से संबंधित त्रुटि होने पर उठाया जाता है।
यूनिकोडडिकोडत्रुटि	डिकोडिंग के दौरान यूनिकोड से संबंधित त्रुटि होने पर उठाया जाता है।
यूनिकोड अनुवाद त्रुटि	अनुवाद के दौरान यूनिकोड से संबंधित त्रुटि होने पर उठाया जाता है।
वैल्यू एरर	तब उठाया जाता है जब किसी फ़ंक्शन को सही प्रकार का तर्क मिलता है लेकिन अनुचित मान।
ज़ीरोडिवीज़न त्रुटि	जब डिवीजन या मोडुलो ऑपरेशन का दूसरा ऑपरेंड शून्य होता है तो उठाया जाता है।

अपवाद-हैंडलिंग के माध्यम से कैसे प्राप्त करें

पायथन में कई अंतर्निहित अपवाद होता हैं जो आपके प्रोग्राम को एक त्रुटि उत्पन्न करने के लिए मजबूर करता है जब इसमें कुछ गलत हो जाता है।

जब ये अपवाद होते हैं, तो यह वर्तमान प्रक्रिया को रोक देता है और इसे कॉलिंग प्रक्रिया में तब तक भेजता है जब तक इसे संभाला नहीं जाता है। अगर संभाला नहीं गया, तो हमारा प्रोग्राम क्रैश हो सकता है।

उदाहरण के लिए यदि फ़ंक्शन `A`, फ़ंक्शन `B` को कॉल करता है, जो बदले में फ़ंक्शन `C` को कॉल करता है और एक अपवाद होता है
फंक्शन में `C` यदि इसे सी में संभाला नहीं जाता है, तो अपवाद `B` और फिर `A` के पास जाता है।

यदि इसे कभी संभाला नहीं जाता है, तो एक त्रुटि संदेश स्पीट हो जाता है और हमारा फंक्शन अचानक, अप्रत्याशित पड़ाव पर आ जाता है।

पायथन में अपवादों को पकड़ना

पायथन में, एक कोशिश कथन का उपयोग करके अपवादों को नियंत्रित किया जा सकता है।

एक क्रिटिकल ऑपरेशन जो अपवाद को बढ़ा सकता है उसे ट्राई क्लॉज के अंदर रखा जाता है और अपवाद को हैंडल करने वाला कोड क्लॉज को छोड़कर में लिखा जाता है।

यह हम पर निर्भर करता है कि अपवाद को पकड़ने के बाद हम कौन से ऑपरेशन करते हैं। ये रहा एक सरल उदाहरण है।

```
# import module sys to get the type of exception
import sys
randomList = ['a', 0, 2]
for entry in randomList:
      try:
            print("The entry is", entry)
```

```
            r = 1/int(entry)
            break
      except:
            print("Oops!",sys.exc_info()[0],"occured.")
            print("Next entry.")
            print()
print("The reciprocal of",entry,"is",r)
```

आउटपुट

```
The entry is a
Oops! <class 'ValueError'> occured.
Next entry.
The entry is 0
Oops! <class 'ZeroDivisionError' > occured.
Next entry.
The entry is 2
The reciprocal of 2 is 0.5
```

इस कार्यक्रम में हम तब तक लूप करते हैं जब तक उपयोगकर्ता एक पूर्णांक में प्रवेश नहीं करता है जिसमें एक वैध पारस्परिक होता है। वह भाग जो अपवाद का कारण बन सकता है उसे `try` ब्लॉक के अंदर रखा जाता है।

यदि कोई अपवाद नहीं होता है, तो ब्लॉक को छोड़कर उसे छोड़ दिया जाता है और सामान्य प्रवाह जारी रहता है। लेकिन अगर कोई अपवाद होता है, तो उसे छोड़कर ब्लॉक द्वारा पकड़ा जाता है।

यहां, हम `sys` मॉड्यूल के अंदर `ex_info ()` फ़ंक्शन का उपयोग करके अपवाद का नाम प्रिंट करते हैं और उपयोगकर्ता को फिर से प्रयास करने के लिए कहते हैं। हम देख सकते हैं कि मान 'a' और '1.3' वैल्यूएरर का कारण बनते हैं और जीरो डिवीज़न एरर का कारण बनते हैं।

पायथन में विशिष्ट अपवादों को पकड़ना

उपरोक्त उदाहरण में हमने अपवाद खंड में किसी अपवाद का उल्लेख नहीं किया है।

यह एक अच्छा प्रोग्रामिंग अभ्यास नहीं है क्योंकि यह सभी अपवादों को पकड़ लेगा और हर मामले को उसी तरह से संभालेगा। हम निर्दिष्ट कर सकते हैं कि कौन से अपवाद एक को छोड़कर क्लॉज पकड़े।

एक कोशिश खंड में उन्हें अलग तरीके से संभालने के लिए खंड को छोड़कर किसी भी संख्या में हो सकता है लेकिन अपवाद होने पर केवल एक ही निष्पादित किया जाता है।

हम एक अपवाद खंड में कई अपवादों को निर्दिष्ट करने के लिए मानों के टपल का उपयोग कर सकते हैं। यहाँ एक उदाहरण छद्म कोड है।

```
try:
      # do something
      pass
except ValueError:
      # handle ValueError exception
      pass
except (TypeError, ZeroDivisionError):
      # handle multiple exceptions
      # TypeError and ZeroDivisionError
      pass
except:
      # handle all other exceptions
      pass
```

अपवाद बढ़ाना

पायथन प्रोग्रामिंग में अपवाद तब उठाए जाते हैं जब रन टाइम में संबंधित त्रुटियां होती हैं, लेकिन हम कीवर्ड उठाकर इसे ताकत से बढ़ा सकते हैं।

हम यह स्पष्ट करने के लिए अपवाद के मूल्य में वैकल्पिक रूप से पास कर सकते हैं कि वह अपवाद क्यों उठाया गया था।

```
>>> raise KeyboardInterrupt
Traceback (most recent call last):
...
KeyboardInterrupt
>>> raise MemoryError("This is an argument")
Traceback (most recent call last):
...
MemoryError: This is an argument
>>> try:
... a = int(input("Enter a positive integer: "))
... if a <= 0:
```

```
... raise ValueError("That is not a positive number!")
... except ValueError as ve:
... print(ve)
...
Enter a positive integer: -2
That is not a positive number!
```

अंतिम कोशिश

पायथन में ट्राई स्टेटमेंट में एक वैकल्पिक अंत में क्लॉज हो सकता है। इस खंड को निष्पादित किया जाता है चाहे कुछ भी हो और आमतौर पर बाहरी संसाधनों को जारी करने के लिए उपयोग किया जाता है।

उदाहरण के लिए हम नेटवर्क के माध्यम से किसी दूरस्थ डेटा केंद्र से जुड़े हो सकते हैं या फ़ाइल के साथ काम कर सकते हैं या ग्राफिकल यूजर इंटरफेस (जीयूआई) के साथ काम कर सकते हैं।

इन सभी परिस्थितियों में, हमें एक बार उपयोग किए गए संसाधन को साफ करना चाहिए, चाहे वह सफल रहा हो या नहीं। निष्पादन की गारंटी के लिए इन क्रियाओं (फ़ाइल को बंद करना, जीयूआई या नेटवर्क से डिस्कनेक्ट करना) अंतिम खंड में किया जाता है।

इसे स्पष्ट करने के लिए फ़ाइल संचालन का एक उदाहरण यहां दिया गया है।

```
try:
      f = open("test.txt",encoding = 'utf-8')
      # perform file operations
finally:
      f.close()
```

इस प्रकार का निर्माण सुनिश्चित करता है कि अपवाद होने पर भी फ़ाइल बंद होती है।

उपयोगकर्ता द्वारा परिभाषित अपवाद को कैसे संभालें

पायथन में कई अंतर्निहित अपवाद होते हैं जो आपके प्रोग्राम को एक त्रुटि उत्पन्न करने के लिए मजबूर करते हैं जब इसमें कुछ गलत हो जाता है। हालांकि, कभी-कभी आपको अपने उद्देश्य को पूरा करने वाले कस्टम अपवाद बनाने की आवश्यकता हो सकती है।

पायथन में उपयोगकर्ता एक नया वर्ग बनाकर ऐसे अपवादों को परिभाषित कर सकते हैं। यह अपवाद वर्ग, प्रत्यक्ष या अप्रत्यक्ष रूप से, अपवाद वर्ग से प्राप्त किया जाता है। अधिकांश अंतर्निहित अपवाद भी इसी वर्ग से प्राप्त होते हैं।

```
>>> class CustomError(Exception):
```

```
... pass
...
>>> raise CustomError
Traceback (most recent call last):
...
_ main  .CustomError
>>> raise CustomError(“An error occurred”)
Traceback (most recent call last):
...
_ main  .CustomError: An error occurred
```

यहां, हमने कस्टमइरर नामक एक उपयोगकर्ता-परिभाषित अपवाद बनाया है जो अपवाद वर्ग से लिया गया है। इस नए अपवाद को अन्य अपवादों की तरह, वैकल्पिक त्रुटि संदेश के साथ बढ़ाएँ कथन का उपयोग करके उठाया जा सकता है।

जब हम एक बड़ा पायथन प्रोग्राम विकसित कर रहे होते हैं, तो हमारे प्रोग्राम द्वारा उठाए गए सभी उपयोगकर्ता-परिभाषित अपवादों को एक अलग फ़ाइल में रखना एक अच्छा अभ्यास होता है। कई मानक मॉड्यूल ऐसा करते हैं। वे अपने अपवादों को अलग-अलग अपवाद.py या error.py के रूप में परिभाषित करते हैं (आमतौर पर लेकिन हमेशा नहीं)।

उपयोगकर्ता-परिभाषित अपवाद वर्ग वह सब कुछ लागू कर सकता है जो एक सामान्य वर्ग कर सकता है, लेकिन हम आम तौर पर उन्हें सरल और संक्षिप्त बनाते हैं। अधिकांश कार्यान्वयन एक कस्टम बेस क्लास घोषित करते हैं और इस बेस क्लास से अन्य अपवाद वर्ग प्राप्त करते हैं। इस अवधारणा को निम्नलिखित उदाहरण में स्पष्ट किया गया है।

उदाहरण: पायथन में उपयोगकर्ता-परिभाषित अपवाद

इस उदाहरण में, हम यह बताएंगे कि प्रोग्राम में त्रुटियों को बढ़ाने और पकड़ने के लिए उपयोगकर्ता द्वारा परिभाषित अपवादों का उपयोग कैसे किया जा सकता है।

यह प्रोग्राम उपयोगकर्ता को एक संख्या दर्ज करने के लिए कहता है जब तक कि वे एक संग्रहीत संख्या का सही अनुमान न लगा लें। यह पता लगाने में उनकी मदद करने के लिए, संकेत दिया जाता है कि उनका अनुमान संग्रहीत संख्या से अधिक या कम है।

```
# define Python user-defined exceptions
class Error(Exception):
      “””Base class for other exceptions”””
      pass
class ValueTooSmallError(Error):
      “””Raised when the input value is too small”””
```

```
      pass
class ValueTooLargeError(Error):
      """Raised when the input value is too large"""
      pass
# our main program
# user guesses a number until he/she gets it right
# you need to guess this number
number = 10
while True:
      try:
            i_num = int(input("Enter a number: "))
      if i_num < number:
            raise ValueTooSmallError
      elif i_num > number:
            raise ValueTooLargeError
            break
      except ValueTooSmallError:
            print("This value is too small, try again!")
            print()
      except ValueTooLargeError:
            print("This value is too large, try again!")
            print()
print("Congratulations! You guessed it correctly.")
```

यहां इस कार्यक्रम का एक उदाहरण है।

```
Enter a number: 12
This value is too large, try again!
Enter a number: 0
This value is too small, try again!
```

```
Enter a number: 8
This value is too small, try again!
Enter a number: 10
Congratulations! You guessed it correctly.
```

यहां, हमने एरर नामक बेस क्लास को परिभाषित किया है।

अन्य दो अपवाद (`ValueTooSmallError` और `ValueTooLargeError`) जो वास्तव में हमारे कार्यक्रम द्वारा उठाए गए हैं, इस वर्ग से प्राप्त हुए हैं। पायथन प्रोग्रामिंग में उपयोगकर्ता द्वारा परिभाषित अपवादों को परिभाषित करने का यह मानक तरीका है, लेकिन आप केवल इस तरह तक ही सीमित नहीं हैं।

5

पायथन ऑब्जेक्ट्स और क्लासेस

एक वस्तु डेटा वेरिएबल और पायथन के फंक्शन का एक संग्रह है। क्लास उस कोड टेम्पलेट को संदर्भित करता है जो इन ऑब्जेक्ट्स को बनाता है। इस अध्याय को पायथन की वस्तुओं और वर्गों के साथ-साथ कई विरासतों की आसान समझ प्रदान करने के लिए सावधानीपूर्वक लिखा गया है।

वस्तु और वर्ग का उपयोग कैसे करें

पायथन एक ऑब्जेक्ट ओरिएंटेड प्रोग्रामिंग लैंग्वेज है। प्रक्रिया उन्मुख प्रोग्रामिंग के विपरीत, जहां कार्यों पर मुख्य जोर दिया जाता है, वस्तुओं पर वस्तु उन्मुख प्रोग्रामिंग स्ट्रेस होता है।

वस्तु केवल डेटा (वेरिएबल) और विधियों (फ़ंक्शंस) का एक संग्रह होता है जो उन डेटा पर कार्य करता है। और, वर्ग वस्तु के लिए एक खाका होता है।

हम क्लास को एक घर के स्केच (प्रोटोटाइप) के रूप में सोच सकते हैं। इसमें फर्श, दरवाजे, खिड़कियां आदि के बारे में सभी विवरण शामिल हैं। इन विवरणों के आधार पर हम घर बनाते हैं। घर वस्तु है।

जैसे, विस्तार से कई घर बनाए जा सकते हैं, हम एक वर्ग से कई वस्तुएँ बना सकते हैं। किसी ऑब्जेक्ट को क्लास का इंस्टेंस भी कहा जाता है और इस ऑब्जेक्ट को बनाने की प्रक्रिया को इंस्टेंटेशन कहा जाता है।

पायथन में एक क्लास को परिभाषित करना

जैसे फ़ंक्शन की परिभाषाएं कीवर्ड डेफ से शुरू होती हैं, इसी तरह पायथन में भी हम कीवर्ड क्लास का उपयोग करके एक क्लास को परिभाषित करते हैं।

पहली स्ट्रिंग को डॉकस्ट्रिंग कहा जाता है और इसमें क्लास के बारे में संक्षिप्त विवरण होता है। हालांकि अनिवार्य नहीं है, यह अनुशंसित ही हो।

यहां एक साधारण क्लास परिभाषा लिखित है।

```
class MyNewClass:
      '''This is a docstring. I have created a new
      class''' pass
```

एक वर्ग एक नया स्थानीय नाम स्थान बनाता है जहां उसकी सभी विशेषताएँ परिभाषित होती हैं। गुण डेटा या फ़ंक्शन हो सकते हैं।

इसमें विशेष गुण भी होते हैं जो डबल अंडरस्कोर से शुरू होते हैं (_) उदाहरण के लिए,__ द स्ता वे ज़ __हमें उस वर्ग का डॉकस्ट्रिंग देता है।

जैसे ही हम एक वर्ग को परिभाषित करते हैं, उसी नाम से एक नई क्लास वस्तु बनाई जाती है। यह वर्ग वस्तु हमें विभिन्न विशेषताओं तक पहुँचने के साथ-साथ उस वर्ग की नई वस्तुओं को त्वरित करने की अनुमति देता है।

```
class MyClass:
      "This is my second class"
      a = 10
      def func(self):
            print('Hello')
# Output: 10
print(MyClass.a)
# Output: <function MyClass.func at 0x0000000003079BF8>
print(MyClass.func)
# Output: 'This is my second class' print(MyClass.  doc  )
```

जब आप प्रोग्राम चलाते हैं, तो आउटपुट ये होगा:

```
10
<function 0x7feaa932eae8="" at="" myclass.func="">
This is my second class
```

पायथन में एक ऑब्जेक्ट को बनाना

हमने देखा कि विभिन्न विशेषताओं तक पहुँचने के लिए क्लास ऑब्जेक्ट का उपयोग किया जा सकता है।

इसका उपयोग उस वर्ग के नए ऑब्जेक्ट इंस्टेंस (तत्काल) बनाने के लिए भी किया जा सकता है। ऑब्जेक्ट बनाने की प्रक्रिया फ़ंक्शन कॉल के समान होती है।

```
>>> ob = MyClass()
```

यह `ob` नाम का एक नया इंस्टेंस ऑब्जेक्ट बनाता है। हम ऑब्जेक्ट नाम उपसर्ग का उपयोग करके वस्तुओं की विशेषताओं तक पहुंच सकते हैं।

गुण डेटा या विधि हो सकते हैं। किसी वस्तु की विधि उस वर्ग के संगत कार्य होती है।
कोई भी फ़ंक्शन ऑब्जेक्ट जो एक वर्ग की विशेषता होती है, उस वर्ग की वस्तुओं के लिए एक विधि को परिभाषित करता है।

कहने का मतलब यह है कि चूंकि `MyClass.func` एक फंक्शन ऑब्जेक्ट (क्लास की विशेषता) है, इसलिए `ob.func` एक मेथड ऑब्जेक्ट होता है।

```
class MyClass:
      "This is my second
```

```
    class” a = 10

    def func(self):

        print(‘Hello’)

# create a new MyClass

ob = MyClass()

# Output: <function MyClass.func at 0x000000000335B0D0>

print(MyClass.func)

# Output: <bound method MyClass.func of < main .MyClass object at
0x000000000332DEF0>>

print(ob.func)

# Calling function func()

# Output: Hello

ob.func()
```

आपने क्लास के अंदर फंक्शन डेफिनिशन में सेल्फ पैरामीटर पर ध्यान दिया होगा लेकिन, हमने बिना किसी तर्क के इस विधि को केवल `ob.func()` कहते हैं। यह अभी भी काम करता है।

ऐसा इसलिए होता है, क्योंकि जब भी कोई वस्तु अपनी मेथड को बुलाती है, तो वस्तु को ही पहले तर्क के रूप में पारित कर दिया जाता है।

इसलिए,`ob.func ()` अनुवाद My`Class.func (ob)` में करता है।

सामान्य तौर पर n तर्कों की सूची के साथ एक विधि को कॉल करना संबंधित फ़ंक्शन को एक तर्क सूची के साथ कॉल करने के बराबर होता है जो पहले तर्क से पहले मेथड की वस्तु को सम्मिलित करके बनाई गई होती है।

इन कारणों से, क्लास में फ़ंक्शन का पहला तर्क ऑब्जेक्ट ही होना चाहिए। इसे पारंपरिक रूप से `self` कहा जाता है। इसे अन्यथा नाम दिया जा सकता है लेकिन हम सम्मेलन का पालन करने की अत्यधिक अनुशंसा करते हैं।

अब आपको क्लास ऑब्जेक्ट, इंस्टेंस ऑब्जेक्ट, फंक्शन ऑब्जेक्ट, मेथड ऑब्जेक्ट और उनके अंतर से परिचित हो जाना चाहिए।

पायथन में कंस्ट्रक्टर्स

क्लास के कार्य जो डबल अंडरस्कोर से शुरू होते हैं (__) विशेष कार्य कहलाते हैं क्योंकि उनका विशेष अर्थ होता है।

एक विशेष का ब्याज है `__init__ ()` फंक्शन होता है। जब भी उस वर्ग की कोई नई वस्तु तत्काल होती है तो यह विशेष कार्य कहा जाता है।

ऑब्जेक्ट ओरिएंटेड प्रोग्रामिंग (ओओपी) में इस प्रकार के फंक्शन को कंस्ट्रक्टर भी कहा जाता है। हम आम तौर पर इसका इस्तेमाल सभी वेरिएबल्स को इनिशियलाइज़ करने के लिए करते हैं।

```
class ComplexNumber:

def   init  (self,r = 0,i = 0):
self.real = r
            self.imag = i
      def getData(self):
            print("{0}+{1}j".format(self.real,self.imag))
# Create a new ComplexNumber object
c1 = ComplexNumber(2,3)
# Call getData() function
# Output: 2+3j
c1.getData()
# Create another ComplexNumber object
# and create a new attribute 'attr'
c2 = ComplexNumber(5)
c2.attr = 10
# Output: (5, 0, 10)
print((c2.real, c2.imag, c2.attr))
# but c1 object doesn't have attribute 'attr'
# AttributeError: 'ComplexNumber' object has no attribute 'attr'
c1.attr
```

उपरोक्त उदाहरण में, हम सम्मिश्र संख्याओं को निरूपित करने के लिए एक नए वर्ग को परिभाषित करते हैं। इसके दो कार्य हैं, `_init ()` प्रति संख्या को ठीक से प्रदर्शित करने के लिए वेरिएबल (डिफ़ॉल्ट शून्य) और `getData()` प्रारंभ करते हैं।

उपरोक्त वेरिएबलण में ध्यान देने योग्य एक दिलचस्प बात यह है कि किसी वस्तु की विशेषताओं को फ्लाई पर बनाया जा सकता है। हमने ऑब्जेक्ट `c2` के लिए एक नया एट्रीब्यूट `attr` बनाया है और हम इसे भी पढ़ते हैं। लेकिन इसने ऑब्जेक्ट `c1` के लिए वह विशेषता नहीं बनाई गई है।

ऐट्रिब्यूट्स और ऑब्जेक्ट्स को हटाना

किसी ऑब्जेक्ट की किसी भी विशेषता को डेल स्टेटमेंट का उपयोग करके कभी भी हटाया जा सकता है। आउटपुट देखने के लिए पायथन शेल पर निम्न का प्रयास कर सकते हैं।

```
>>> c1 = ComplexNumber(2,3)
```

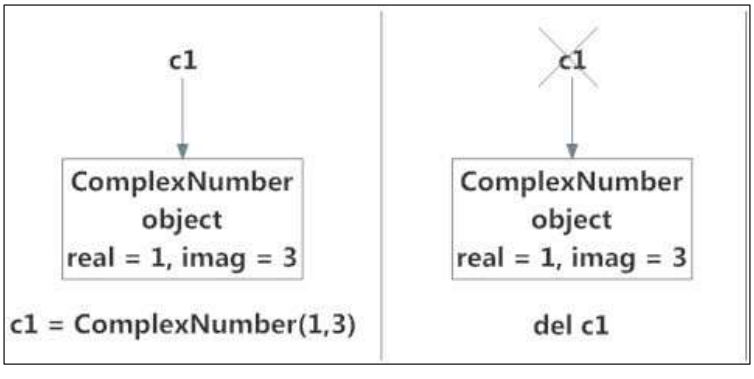
```
>>> del c1.imag
>>> c1.getData()
Traceback (most recent call last):
...
AttributeError: 'ComplexNumber' object has no attribute 'imag'
>>> del ComplexNumber.getData
>>> c1.getData()
Traceback (most recent call last):
...
AttributeError: 'ComplexNumber' object has no attribute 'getData'
```

हम डेल स्टेटमेंट का उपयोग करके ऑब्जेक्ट को स्वयं भी हटा सकते हैं।

```
>>> c1 = ComplexNumber(1,3)
>>> del c1
>>> c1
Traceback (most recent call last):
...
NameError: name 'c1' is not defined
```

दरअसल, यह उससे कहीं अधिक जटिल होता है। जब हम `c1 = ComplexNumber(1,3)` करते हैं, तो मेमोरी में एक नया इंस्टेंस ऑब्जेक्ट बन जाता है और `c1` नाम इसके साथ जुड़ जाता है।

कमांड डेल c1 पर, यह बाइंडिंग हटा दी जाती है और नाम c1 को संबंधित नाम स्थान से हटा दिया जाता है। हालाँकि वस्तु स्मृति में बनी रहती है और यदि कोई अन्य नाम इससे बंधा नहीं है, तो यह बाद में स्वतः नष्ट हो जाती है।



पायथन में गैर-संदर्भित वस्तुओं के इस स्वचालित विनाश को कचरा संग्रह भी कहा जाता है।

ऑब्जेक्ट-ओरिएंटेड प्रोग्रामिंग को अप्रोच कैसे करें

पायथन एक बहु-प्रतिमान प्रोग्रामिंग भाषा होती है। मतलब, यह विभिन्न प्रोग्रामिंग दृष्टिकोण का समर्थन करती है।

प्रोग्रामिंग समस्या को हल करने के लिए लोकप्रिय तरीकों में से एक ऑब्जेक्ट बनाना होता है। यह ज्ञात है ऑब्जेक्ट-ओरिएंटेड प्रोग्रामिंग (ओओपी) के रूप में होती है।

एक वस्तु की दो विशेषताएं होती हैं:

- ऐट्रिब्यूट्स
- बिहेवियर

आइए एक उदाहरण लेते हैं:तोता एक वस्तु है,

- नाम, उम्र, रंग गुण होते हैं।
- गायन, नृत्य व्यवहार होता है।

पायथन में ओओपी की अवधारणा पुन: प्रयोज्य कोड बनाने पर केंद्रित होती है। इस अवधारणा को ड्राई (डोंट रिपीट योरसेल्फ) के रूप में भी जाना जाता है।

पायथन में, ओओपी की अवधारणा कुछ बुनियादी सिद्धांतों का पालन करती है:

इनहेरिटेंस	मौजूदा वर्ग को संशोधित किए बिना एक नए वर्ग से विवरण का उपयोग करने की प्रक्रिया।
एनकैप्सुलेशन	किसी वर्ग के निजी विवरण को अन्य वस्तुओं से छिपाना।
पोलीमोर्फिस्म	विभिन्न डेटा इनपुट के लिए अलग-अलग तरीकों से सामान्य ऑपरेशन का उपयोग करने की अवधारणा।

क्लास

एक वर्ग वस्तु के लिए एक खाका होता है।

हम क्लास को लेबल वाले तोते के एक स्केच के रूप में सोच सकते हैं। इसमें नाम, रंग, आकार आदि के बारे में सभी विवरण शामिल होता है। इन विवरणों के आधार पर हम तोते के बारे में अध्ययन कर सकते हैं। यहाँ तोता एक वस्तु होता है।

तोते के क्लास के लिए ये उदाहरण हो सकता है:

```
class Parrot:

    pass
```

यहां, हम एक खाली क्लास `parrot` को परिभाषित करने के लिए `class` कीवर्ड का उपयोग करते हैं। class से हम उदाहरणों का निर्माण करते हैं। एक उदाहरण एक विशेष वर्ग से बनाई गई एक विशिष्ट वस्तु होती है।

ऑब्जेक्ट्स

एक ऑब्जेक्ट्स (उदाहरण) एक क्लास की तात्कालिकता होती है। जब क्लास को परिभाषित किया जाता है, तो केवल ऑब्जेक्ट्स का विवरण परिभाषित किया जाता है। इसलिए, कोई स्मृति या भंडारण आवंटित नहीं किया गया है।

तोता वर्ग की वस्तु का उदाहरण हो सकता है:

```
obj = Parrot()
```

यहां, `obj` तोता वर्ग की वस्तु होता है।

मान लीजिए हमारे पास तोते का विवरण है। अब हम यह दिखाने जा रहे हैं कि तोते के क्लास और ऑब्जेक्ट का निर्माण कैसे किया जाता है।

उदाहरण: पायथन में क्लास और ऑब्जेक्ट का निर्माण करना

```
class Parrot:
      # class attribute
      species = "bird"
      # instance attribute
      def   init  (self, name, age):
            self.name = name
            self.age = age
# instantiate the Parrot class
blu = Parrot("Blu", 10)
woo = Parrot("Woo", 15)
# access the class attributes
print("Blu is a {}".format(blu.  class  .species))
print("Woo is also a {}".format(woo.  class  .species))
# access the instance attributes
print("{} is {} years old".format( blu.name, blu.age))
print("{} is {} years old".format( woo.name, woo.age))
```

जब हम प्रोग्राम चलाते हैं, तो आउटपुट होगा:

```
Blu is a bird
Woo is also a bird
Blu is 10 years old
```

```
Woo is 15 years old
```

उपरोक्त फंक्शन में हम `parrot` नाम के साथ एक वर्ग बनाते हैं। फिर, हम विशेषताओं को परिभाषित करते हैं। गुण किसी वस्तु के लक्षण होते हैं।

फिर, हम `parrot` वर्ग के उदाहरण बनाते हैं। यहां, `blue` और `woo` हमारी नई वस्तुओं के संदर्भ (मूल्य) होते हैं।

फिर हम क्लास एट्रिब्यूट का उपयोग करके एक्सेस करते हैं `__class __.species`। क्लास एक वर्ग के सभी उदाहरणों के लिए विशेषताएं एक समान होती हैं। इसी तरह हम `blu.name` और `blu.age.` का उपयोग करके इंस्टेंस एट्रिब्यूट्स को एक्सेस करते हैं। हालांकि, किसी वर्ग के प्रत्येक उदाहरण के लिए आवृत्ति विशेषताएं भिन्न होती हैं।

क्लास और ऑब्जेक्ट्स के बारे में अधिक जानने के लिए पायथन क्लासेस और ऑब्जेक्ट्स पर जाएं।

तरीके

विधियां एक वर्ग के बॉडी के अंदर परिभाषित कार्य होती हैं। इनका उपयोग एक वस्तु के व्यवहार को परिभाषित करने के लिए किया जाता है।

उदाहरण: पायथन में मेथड का निर्माण करना

```
class Parrot:
      # instance attributes
      def   init  (self, name, age):
            self.name = name
            self.age = age
      # instance method
      def sing(self, song):
            return "{} sings {}".format(self.name, song)
      def dance(self):
            return "{} is now dancing".format(self.name)
# instantiate the object
blu = Parrot("Blu", 10)
# call our instance methods
print(blu.sing("'Happy'"))
print(blu.dance())
```

जब हम प्रोग्राम चलाते हैं, तो आउटपुट होगा:

```
Blu sings 'Happy'
```

```
Blu is now dancing
```

उपरोक्त फंक्शन में हम दो विधियों को परिभाषित करते हैं अर्थात `sing()` और `dance()`। इन्हें इंस्टेंस मेथड कहा जाता है क्योंकि इन्हें इंस्टेंस ऑब्जेक्ट यानी ब्लू पर बुलाया जाता है।

इनहेरिटेंस

वंशानुक्रम मौजूदा वर्ग के विवरण को संशोधित किए बिना उपयोग करने के लिए नया वर्ग बनाने का एक तरीका है। नवगठित वर्ग एक व्युत्पन्न वर्ग (या बाल वर्ग) है। इसी तरह, मौजूदा वर्ग एक बेस क्लास (या पैरेंट क्लास) होता है।

उदाहरण: पायथन में इनहेरिटेंस का इस्तेमाल

```
# parent class
class Bird:
      def   init  (self):
            print("Bird is ready")
      def whoisThis(self):
            print("Bird")
      def swim(self):
            print("Swim faster")
# child class
class Penguin(Bird):
      def   init  (self):
            # call super() function
      super().  init  ()
      print("Penguin is ready")
      def whoisThis(self):
            print("Penguin")
      def run(self):
            print("Run faster")
peggy = Penguin()
peggy.whoisThis() peggy.swim()
```

```
peggy.run()
```

जब हम इस प्रोग्राम को चलाते हैं, तो उसका आउटपुट ये होता है:

```
Bird is ready

Penguin is ready

Penguin

Swim faster

Run faster
```

उपरोक्त फंक्शन में हमने दो वर्ग बनाए अर्थात `bird` (पैरेंट क्लास) और `penguin` (चाइल्ड क्लास)। चाइल्ड क्लास को पैरेंट क्लास के फंक्शन विरासत में मिलते हैं। हम इसे `swim ()` विधि से देख सकते हैं। फिर से, चाइल्ड क्लास ने मूल वर्ग के व्यवहार को संशोधित किया है। इसे हम `whoisThis()` मेथड से देख सकते हैं। इसके अलावा, हम एक नई `run ()` विधि बनाकर मूल वर्ग के कार्यों का विस्तार करते हैं।

इसके अतिरिक्त, हम `__init__()` विधि से पहले `_init_()` फ़ंक्शन का उपयोग करते हैं। ऐसा इसलिए है क्योंकि हम `__init__()` विधि की सामग्री को मूल वर्ग से चाइल्ड क्लास में खींचना चाहते हैं

एनकैप्सुलेशन

ओओपी का उपयोग करके पायथन में हम विधियों और वेरिएबल तक पहुंच को प्रतिबंधित कर सकते हैं। यह डेटा को सीधे संशोधन से रोकता है जिसे एनकैप्सुलेशन कहा जाता है। पायथन में, हम उपसर्ग के रूप में अंडरस्कोर का उपयोग करके निजी विशेषता को निरूपित करते हैं अर्थात सिंगल "_" या डबल "".

उदाहरण: पायथन में डेटा एनकैप्सुलेशन

```
class Computer:
      def   init  (self):
            self.  maxprice = 900
      def sell(self):
            print("Selling Price: {}".format(self.  maxprice))
      def setMaxPrice(self, price):
            self.  maxprice = price
c = Computer()
c.sell()
# change the price
c.  maxprice = 1000
c.sell()
```

```
# using setter function

c.setMaxPrice(1000)

c.sell()
```

जब हम इस प्रोग्राम को चलाते हैं, तो आउटपुट ये होगा:

```
Selling Price: 900

Selling Price: 900

Selling Price: 1000
```

उपरोक्त फंक्शन में, हमने एक वर्ग को परिभाषित किया है संगणक। जब हम इसका प्रयोग करते हैं तो इस में `() computer` के अधिकतम विक्रय मूल्य को स्टोर करने की विधि होती है। हमने कीमत में बदलाव करने की कोशिश की। हालाँकि, हम इसे बदल नहीं सकते क्योंकि पायथन इसका व्यवहार करता है `__maxprice` निजी विशेषताओं के रूप में। मान बदलने के लिए, हमने एक सेटर फ़ंक्शन यानी `setMaxPrice()` का उपयोग किया, जो पैरामीटर के रूप में मूल्य को लेता है।

पोलीमोर्फिस्म

बहुरूपता एक क्षमता होती है (ओओपी में) एकाधिक रूपों (डेटा प्रकार) के लिए सामान्य इंटरफ़ेस का उपयोग करने के लिए इसका इस्तेमाल किया जाता है।

मान लीजिए हमें एक आकृति को रंगने की जरूरत है तो कई आकार विकल्प (आयत, वर्ग, वृत्त) हैं। हालांकि हम किसी भी आकृति को रंगने के लिए उसी विधि का उपयोग कर सकते हैं। इस अवधारणा को बहुरूपता कहा जाता है।

उदाहरण: पायथन में पोलीमोर्फिस्म का उपयोग करना

```
class Parrot:
      def fly(self):
            print("Parrot can fly")
      def swim(self):
            print("Parrot can't swim")
class Penguin:
      def fly(self):
            print("Penguin can't fly")
      def swim(self):
            print("Penguin can swim")
# common interface
def flying_test(bird):bird.fly()
```

```
#instantiate objects

blu = Parrot()

peggy = Penguin()

# passing the object

flying_test(blu)

flying_test(peggy)
```

जब हम उपरोक्त प्रोग्राम चलाते हैं, तो आउटपुट ये मिलता है:

```
Parrot can fly

Penguin can't fly
```

उपरोक्त फंक्शन में, हमने दो वर्गों को परिभाषित किया है `Parrot` और `Penguin` । उनमें से प्रत्येक के पास आम है विधि `fly ()` विधि होती है। हालांकि, उनके कार्य अलग होते हैं। बहुरूपता की अनुमति देने के लिए, हमने सामान्य इंटरफ़ेस बनाया है यानी flying_test() फ़ंक्शन जो किसी भी वस्तु को ले सकता है। फिर, हमने `flying_test()` फ़ंक्शन में `blu` और `peggy` ऑब्जेक्ट्स को पास किया, यह प्रभावी रूप से चलता है।

याद रखने योग्य मुख्य बिंदु

- प्रोग्रामिंग आसान और कुशल हो जाती है।
- वर्ग साझा करने योग्य होते हहिं, इसलिए कोड का पुन: उपयोग किया जा सकता है।
- प्रोग्रामर की उत्पादकता बढ़ जाती है।
- डेटा एब्स्ट्रैक्शन के साथ डेटा सुरक्षित और सुरक्षित होती है।

इनहेरिटेंस कैसे करें

वस्तु उन्मुख प्रोग्रामिंग में वंशानुक्रम एक शक्तिशाली विशेषता होती है।

यह एक नए वर्ग को परिभाषित करने के लिए संदर्भित करता है जिसमें मौजूदा वर्ग में बहुत कम या कोई संशोधन नहीं होता है। नया वर्ग है व्युत्पन्न कहा जाता है (या बच्चा) वर्ग और जिससे इसे विरासत में मिला है, उसे आधार (या माता-पिता) वर्ग कहा जाता है।

पायथन इनहेरिटेंस सिंटैक्स

```
class BaseClass:
      Body of base class
class DerivedClass(BaseClass):
      Body of derived class
```

व्युत्पन्न वर्ग बेस क्लास से सुविधाओं को इनहेरिट करता है, इसमें नई सुविधाएँ को जोड़ता है। इसका परिणाम है कोड की पुन: प्रयोज्यता करना।

पायथन में इनहेरिटेंस का उदाहरण

विरासत के उपयोग को प्रदर्शित करने के लिए, आइए हम एक उदाहरण लेते हैं।

एक बहुभुज 3 या अधिक भुजाओं वाली एक बंद आकृति होती है। मां लीजिये हमारे पास बहुभुज नामक एक वर्ग है जिसे इस प्रकार परिभाषित किया गया है।

```
class Polygon:
      def   init  (self, no_of_sides):
            self.n = no_of_sides
            self.sides = [0 for i in range(no_of_sides)]
      def inputSides(self):
            self.sides = [float(input("Enter side "+str(i+1)+" : ")) for i
            in range(self.n)]
      def dispSides(self):
            for i in range(self.n):
                  print("Side",i+1,"is",self.sides[i])
```

इस वर्ग में एक सूची के रूप में पक्षों की संख्या, n और प्रत्येक पक्ष के परिमाण को संग्रहीत करने के लिए डेटा विशेषताएँ होती हैं `sides`।

विधि `inputSides()` लेता है प्रत्येक पक्ष के परिमाण में होता है और इसी तरह, `dispSides()` इन्हें ठीक से प्रदर्शित करता है।

एक त्रिभुज एक बहुभुज है जिसमें 3 भुजाएँ होती हैं। तो, हम नामक एक वर्ग बना सकते हैं `Triangle` जो बहुभुज से विरासत में मिला है। यह Polygon वर्ग में उपलब्ध सभी विशेषताओं को त्रिभुज में आसानी से उपलब्ध कराता है। हमें उन्हें फिर से परिभाषित करने की आवश्यकता नहीं है (कोड पुन: प्रयोज्य)। Triangle को इस प्रकार परिभाषित किया गया है।

```
class Triangle(Polygon):
      def   init  (self):
            Polygon.  init  (self,3)
      def findArea(self):
            a, b, c = self.sides
            # calculate the semi-perimeter
            s = (a + b + c) / 2
            area = (s*(s-a)*(s-b)*(s-c)) ** 0.5
            print('The area of the triangle is %0.2f' %area)
```

हालांकि, Triangle के क्षेत्र को खोजने और प्रिंट करने के लिए वर्ग त्रिभुज में एक नया तरीका है `findArea()` ।
यहां एक उदाहरण रन के रूप में लिखा है।

```
>>> t = Triangle()
>>> t.inputSides()
Enter side 1 : 3
Enter side 2 : 5
Enter side 3 : 4
>>> t.dispSides()
Side 1 is 3.0
Side 2 is 5.0
Side 3 is 4.0
>>> t.findArea()
The area of the triangle is 6.00
```

हम देख सकते हैं कि, भले ही हमने तरीकों को परिभाषित नहीं किया हो `inputSides()` या `dispSides()` क्लास के लिए `Triangle`, हम उनका उपयोग करने में सक्षम थे।

यदि वर्ग में कोई विशेषता नहीं मिलती है, तो आधार वर्ग की खोज जारी रहती है। यह पुनरावर्ती रूप से दोहराता है, यदि आधार वर्ग स्वयं अन्य वर्गों से प्राप्त होता है।

पायथन में ओवरराइडिंग मेथड

उपरोक्त उदाहरण में ध्यान दें कि इस में () विधि को दोनों वर्गों में परिभाषित किया गया था, त्रिकोण साथ ही बहुभुज है। जब ऐसा होता है तो व्युत्पन्न वर्ग में विधि उस आधार वर्ग में ओवरराइड करती है। यह कहना है,इस में () त्रिभुज में बहुभुज में उसी पर वरीयता प्राप्त होती है।

आम तौर पर जब किसी आधार पद्धति को ओवरराइड करते हैं, तो हम परिभाषा को केवल बदलने के बजाय उसका विस्तार करते हैं। व्युत्पन्न वर्ग में से एक से बेस क्लास में विधि को कॉल करके भी ऐसा ही किया जा रहा है (कॉलिंगबहुभुज। इस में () से इस में () त्रिकोण में)।

अंतर्निहित फ़ंक्शन सुपर() का उपयोग करना एक बेहतर विकल्प होगा। तो, सुपर ()।इस में (3) बहुभुज के बराबर है। इस में (स्व, 3) और पसंद किया जाता है।

विरासत की जांच के लिए दो अंतर्निहित कार्य `isinstance` () और `issubclass ()` का उपयोग किया जाता है। फ़ंक्शन `isinstance ()` सही है यदि ऑब्जेक्ट वर्ग या उससे प्राप्त अन्य वर्गों का एक उदाहरण है। पायथन में प्रत्येक वर्ग को बेस क्लास ऑब्जेक्ट से विरासत में मिला है।

```
>>> isinstance(t,Triangle)
True
>>> isinstance(t,Polygon)
```

```
True
>>> isinstance(t,int)
False
>>> isinstance(t,object)
True
```

इसी तरह, `issubclass ()` का उपयोग क्लास इनहेरिटेंस की जांच के लिए किया जाता है।

```
>>> issubclass(Polygon,Triangle)
False
>>> issubclass(Triangle,Polygon)
True
>>> issubclass(bool,int)
True
```

मल्टीप्ल इनहेरिटेंस कैसे करें

पायथन में मल्टीप्ल इनहेरिटेंस

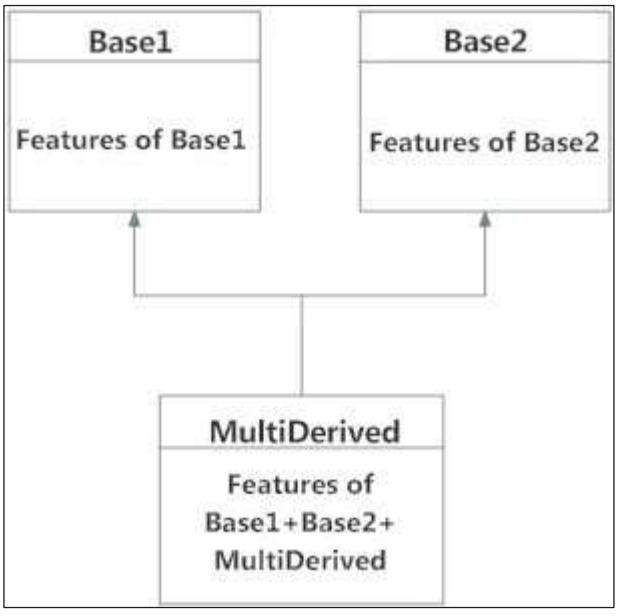
सी ++ की तरह, पायथन में एक से अधिक आधार वर्गों से एक वर्ग प्राप्त किया जा सकता है। इसे मल्टीपल इनहेरिटेंस कहा जाता है।

एकाधिक वंशानुक्रम में, सभी आधार वर्गों की विशेषताएं व्युत्पन्न वर्ग में विरासत में मिली हैं। एकाधिक वंशानुक्रम के लिए वाक्यविन्यास एकल वंशानुक्रम के समान होती हैं।

उदाहरण:

```
class Base1:
      pass
class Base2:
      pass
class MultiDerived(Base1, Base2):
      pass
```

यहां `MultiDerived` को `Base1` और `Base2` कक्षाओं से लिया गया है।



`MultiDerived` वर्ग `Base1` और `Base2` दोनों से इनहेरिट करता है।

पायथन में मल्टीप्ल इनहेरिटेंस

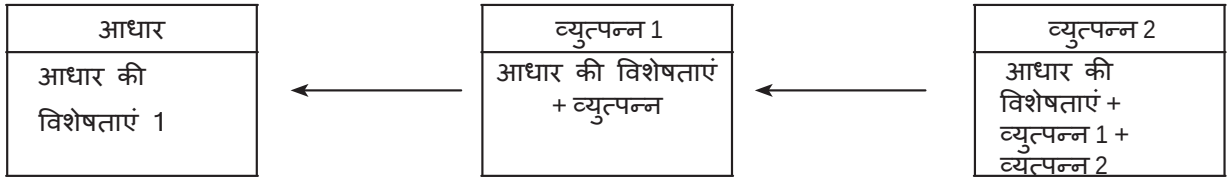
दूसरी ओर हम एक व्युत्पन्न वर्ग के रूप में भी इनहेरिट कर सकते हैं। इसे बहुस्तरीय विरासत कहा जाता है। यह पायथन में किसी भी गहराई का हो सकता है।

बहुस्तरीय वंशानुक्रम में, आधार वर्ग और व्युत्पन्न वर्ग की विशेषताएं नए व्युत्पन्न वर्ग में विरासत में मिली हैं।

संबंधित विज़ुअलाइज़ेशन के साथ एक उदाहरण नीचे दिया गया है।

```
class Base:
      pass
class Derived1(Base):
      pass
class Derived2(Derived1):
      pass
```

यहां, `Derived1` बेस से लिया गया है, और `Derived2, Derived1` से लिया गया है।



पायथन में विधि समाधान के आदेश

प्रत्येक वर्ग पायथन में वर्ग वस्तु से लिया गया है। यह पायथन में सबसे आधार का प्रकार होता है।

तो तकनीकी रूप से, अन्य सभी वर्ग या तो अंतर्निहित या उपयोगकर्ता-परिभाषित, व्युत्पन्न वर्ग होते हैं और सभी वस्तुएं हैं वस्तु वर्ग के उदाहरण हैं।

```
# Output: True

print(issubclass(list,object))

# Output: True

print(isinstance(5.5,object))

# Output: True

print(isinstance("Hello",object))
```

एकाधिक वंशानुक्रम परिदृश्य में, किसी भी निर्दिष्ट विशेषता को पहले वर्तमान वर्ग में खोजा जाता है। अगर नहीं मिला, खोज मूल क्लास ओं में गहराई-प्रथम, बाएँ-दाएँ फ़ैशन में एक ही क्लास को दो बार खोजे बिना जारी है।

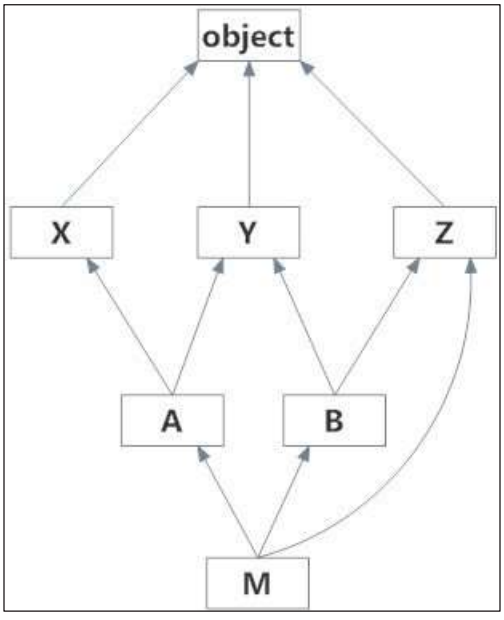
माल लीजिये `MultiDerived` वर्ग के उपरोक्त उदाहरण में खोज क्रम `[MultiDerived, Base1, Base2, object]` है। इस आदेश को `MultiDerived` क्लास का रेखीयकरण भी कहा जाता है और इस ऑर्डर को खोजने के लिए इस्तेमाल किए गए नियमों के सेट को मेथड रेजोल्यूशन ऑर्डर (एमआरओ) कहा जाता है।

एमआरओ स्थानीय वरीयता क्रम को रोकना चाहिए और एकरसता भी प्रदान करनी चाहिए। यह सुनिश्चित करता है कि एक वर्ग हमेशा अपने माता-पिता के सामने उपस्थित होता है और कई माता-पिता के मामले में, क्रम बेस क्लास के टपल के समान होता है।

किसी वर्ग के MRO को `__mro__` विशेषता या `mro()` विधि के रूप में देखा जा सकता है। पूर्व एक टपल देता है जबकि बाद वाला एक सूची देता है।

```
>>> MultiDerived.  mro__
(<class '  main  .MultiDerived'>,
     <class '  main  .Base1'>,
     <class '  main  .Base2'>,
     <class 'object'>)
>>> MultiDerived.mro()
[<class '  main  .MultiDerived'>,
     <class '  main  .Base1'>,
     <class '  main  .Base2'>,
     <class 'object'>]
```

यहां एमआरओ के साथ थोड़ा अधिक जटिल बहु वंशानुक्रम उदाहरण और इसका दृश्य दिखाया गया है।




```
class X: pass
class Y: pass
class Z: pass
class A(X,Y): pass
class B(Y,Z): pass
class M(B,A,Z): pass
# Output:
# [<class ' main .M'>, <class ' main .B'>,
# <class ' main .A'>, <class ' main .X'>,
# <class ' main .Y'>, <class ' main .Z'>,
# <class 'object'>]
print(M.mro())
```

6

पायथन में बुनियादी प्रोग्रामिंग

पायथन में बुनियादी प्रोग्रामिंग सुविधाओं का एक विस्तृत संग्रह मौजूद है जैसे कि प्रिंटिंग, गेम प्रोग्रामिंग, ज्यामितीय पैटर्न बनाना, सरल कैलकुलेटर का उपयोग करना, सरणियाँ, आदि। इस अध्याय में विस्तृत विषय पायथन की इन बुनियादी प्रोग्रामिंग विशेषताओं के बारे में बेहतर परिप्रेक्ष्य प्राप्त करने में मदद करेंगे।

हैलो वर्ल्ड कैसे लिखें

एक साधारण प्रोग्राम बनाना जो कुछ टेक्स्ट प्रदर्शित करता है, अक्सर कोई भी प्रोग्रामर पहली बार अपना वातावरण स्थापित करने के बाद सबसे पहला काम करता है। इसमें केवल कुछ सेकंड लगते हैं और यह प्रोग्रामर को पुष्टि कर सकता है कि पर्यावरण स्थापित है और काम कर रहा है। इन कार्यक्रमों को लिखते समय "नमस्ते दुनिया" सबसे आम बात लगती है।

वैसे भी आप स्क्रीन पर "हैलो वर्ल्ड" टेक्स्ट को आउटपुट करने के लिए पायथन के प्रिंट () फ़ंक्शन का उपयोग करके एक सरल "हैलो वर्ल्ड" प्रोग्राम बना सकते हैं।

प्रोग्राम लिखें

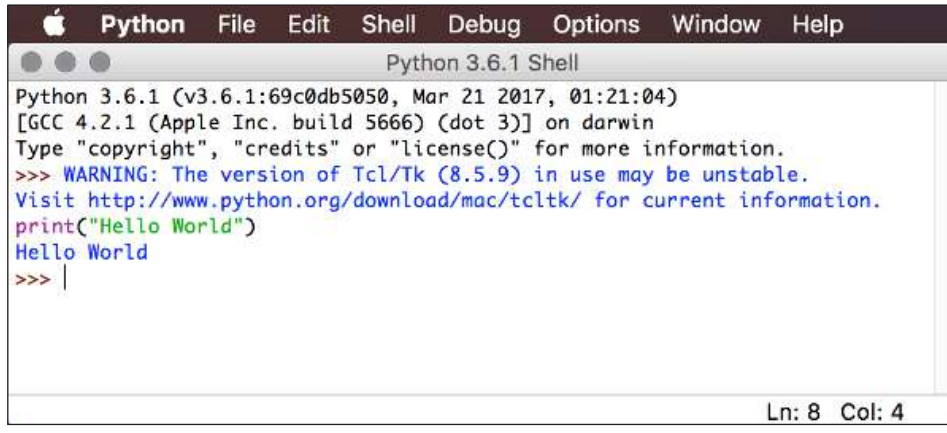
अपना पायथन संपादक खोलें (आईडीएलई ठीक है), और निम्नलिखित कोड दर्ज करें:

```
print("Hello World")
```

आपको यह निम्नलिखित आउटपुट दिखेगा:

```
Hello World
```

यह शायद आपके द्वारा बनाया गया सबसे सरल पायथन प्रोग्राम होगा, फिर भी यह अभी भी एक पायथन है प्रोग्राम है । किसी भी कंप्यूटर प्रोग्राम की तरह, आप इसे एक फ़ाइल में सहेज सकते हैं, फिर अधिक कार्यक्षमता में जोड़ सकते हैं बाद में अगर जरुरत हो तो इसका इस्तेमाल कर सकते हैं।



प्रिंट दर्ज करना ("नमस्ते World") में आईडीएलई का परिणाम होता है `Hello World` स्क्रीन पर प्रिंट हो जाता है।

अपने प्रोग्राम को संरक्षित करें

आगे बढ़ें और अपने प्रोग्राम को `hello.py` नाम की फाइल में सेव करें। यह आम तौर पर संपादक का उपयोग करके किया जाता है फ़ाइल > सेव या सिमिलर।

जब आप अपने पायथन प्रोग्राम को सहेजते हैं, तो एक `.py` फ़ाइल एक्सटेंशन का उपयोग करें। यह वह एक्सटेंशन है जिसका उपयोग पायथन फाइलें करती हैं।

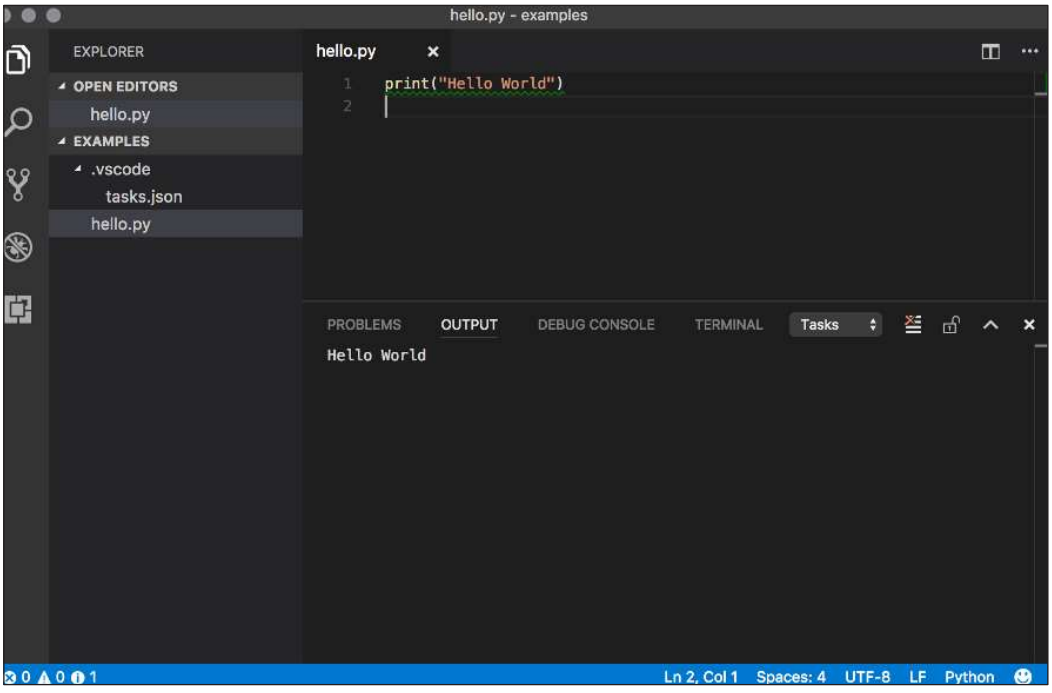
के साथ फ़ाइलें a .py एक्सटेंशन को टेक्स्ट एडिटर के साथ खोला और संपादित किया जा सकता है, लेकिन उन्हें चलाने के लिए एक पायथन दुभाषिया की आवश्यकता होती है।

यदि आप पायथन के साथ उपयोग के लिए कॉन्फ़िगर किए गए कोड संपादक का उपयोग करते हैं, तो यह देखेगा कि यह एक पायथन फ़ाइल है और उपयोग करें उपयुक्त सिंटैक्स हाइलाइटिंग, डिबगिंग, आदि कर सकते हैं।

अपने प्रोग्राम को रन करें

यहां प्रोग्राम को चलाने का तरीका बताया गया है:

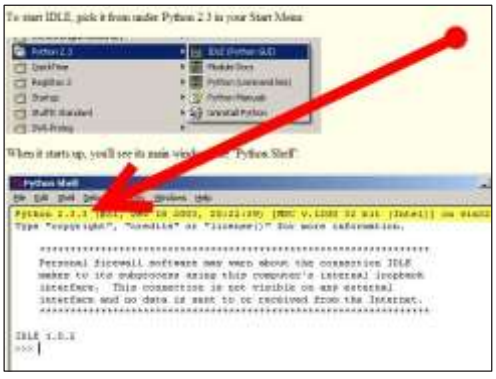
- अपनी फ़ाइल खोलें (यदि पहले से नहीं खुली है)। यह आमतौर पर फ़ाइल> ओपन या समान का उपयोग करके किया जाता है।
- फ़ाइल चलाएं। यह कैसे करना है यह आपके संपादक और मंच पर निर्भर करता है।
  - आईडीएलई में, रन> रन मॉड्यूल (शॉर्टकट F5) आज़माएं।
  - विजुअल स्टूडियो कोड में, विंडोज और लिनक्स पर Ctrl+Shift+B या Mac पर Cmd+Shift+B का उपयोग बिल्ड टास्क को चलाने के लिए करें (आपके टास्क रनर के माध्यम से कॉन्फ़िगर किया गया, जैसा कि हम चुनाव पर्यावरण का पता लगाया)।



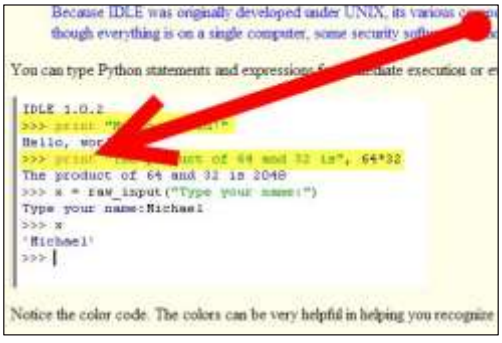
इस स्क्रीनशॉट में, विजुअल स्टूडियो कोड ने अभी-अभी hello.py फ़ाइल में प्रोग्राम चलाया गया है। यहां एक फ़ाइल खुली हुई है एक टैब में (शीर्ष फलक में), और इसका आउटपुट निचले फलक में प्रदर्शित होता है

पायथन में प्रिंट कैसे करें

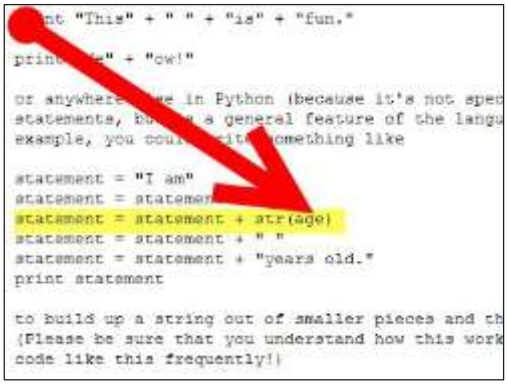
चरण



1. आप तय करें कि आप कौन सा पायथन चला रहे हैं: मैक और लिनक्स में पायथन 2 स्थापित होते हैं, लेकिन आपने इसे स्वयं स्थापित किया है या विंडोज मशीन पर हैं, यह पायथन 3 हो सकता है। जब आप एक पायथन शेल खोलते हैं (आईडीएलई विकास उपकरण के माध्यम से जो आता है) पायथन के साथ या मैक पर टर्मिनल में पायथन टाइप करके देख सकते हैं, उपयोग किए जा रहे पायथन की संस्करण संख्या आपको आपकी स्क्रीन पर दिख जाएगी।



2. अब आप प्रिंट में टाइप करें और उसके बाद एक स्पेस लिखें जो आप प्रिंट करना चाहते हैं। यदि आप पायथन का उपयोग कर रहे हैं मुद्रित की जाने वाली वस्तु के चारों ओर 3 कोष्ठक लगाना आवश्यक है।

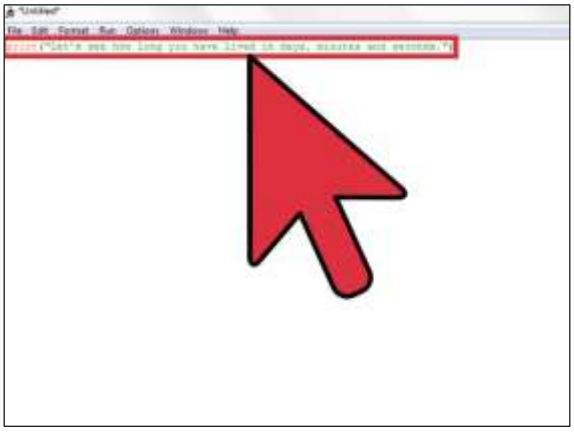


3. यदि आप एक साथ कई मदों को एक साथ मुद्रित करना चाहते हैं, तो इसका उपयोग करें + उन्हें एक साथ जोड़ने के लिए साइन करें और जोड़ने से पहले उन्हें बदलने के लिए +str फ़ंक्शन (आइटम को ब्रैकेट में कनवर्ट करने के लिए रखें) करें।

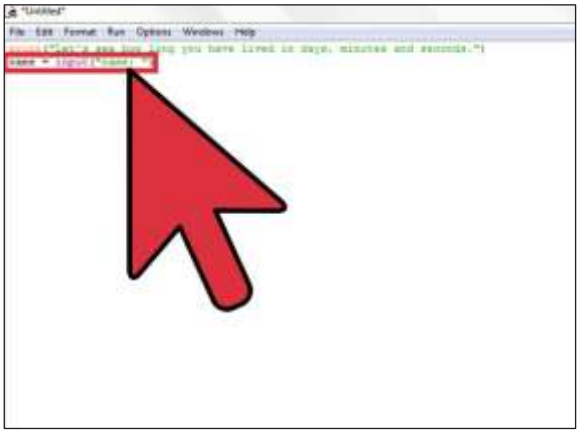
पायथन में एक बहुत ही सरल प्रोग्राम कैसे बनाएं

वेरिएबल ण

1. ctrl-N दबाकर या 'फाइल' और 'नई विंडो' पर जाकर पायथन शेल में एक नई विंडो खोलें।



2. एक परिचयात्मक वाक्य से शुरू करें। तो आपको प्रिंट फ़ंक्शन का उपयोग करना होगा। नीचे दिए गए कोड टाइप करें:

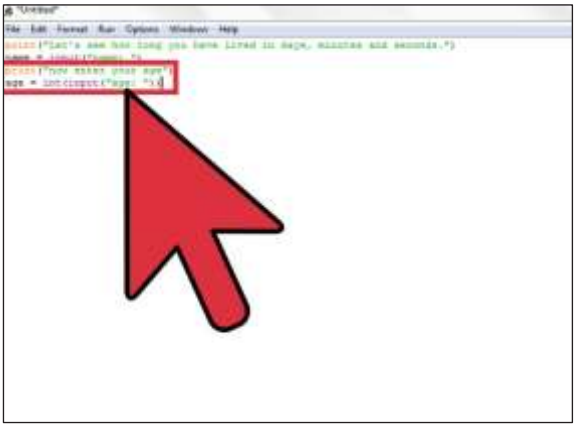


```
print("Let's see how long you have lived in days, minutes and seconds.")
```

3. उपयोगकर्ता का नाम पूछें। यह जानना अच्छा है कि उपयोगकर्ता का नाम क्या है, इसलिए इसे पंक्ति 2 में टाइप करें:

```
name = input("name: ")
```

- वेरिएबल "नाम" को अब उपयोगकर्ता के इनपुट से बदला जा सकता है।



4. अब आप उनकी उम्र पूछो। आपको उम्र जानने की जरूरत है, अब आप ऊपर जैसा ही काम करें, सिवाय इसके कि आपको करना है "इंट" फ़ंक्शन का उपयोग करें, क्योंकि उपयोगकर्ता इस तरह एक नंबर दर्ज करेगा:

```
print("now enter your age")
age = int(input("age: "))
```

- वेरिएबल "आयु" को अब उपयोगकर्ता के इनपुट से बदला जा सकता है।

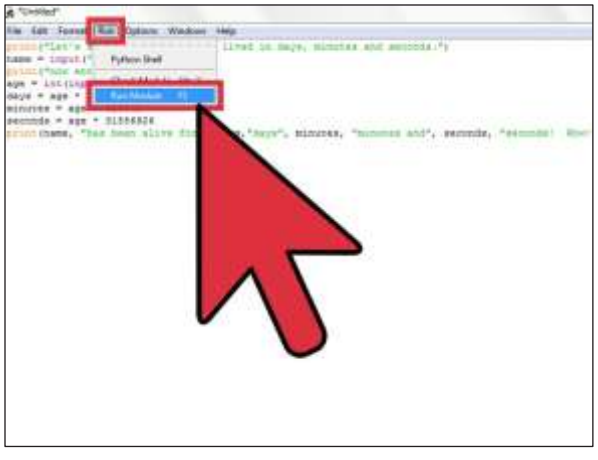
5. उपयोगकर्ता की दी गई उम्र का उपयोग करके रूपांतरण करें।

```
days = age * 365,
minutes = age * 525948
seconds = age * 31556926
```

- एक बार जब आप इसे लिख लेते हैं, तो उपयोगकर्ता की उम्र के इनपुट के आधार पर, पायथन स्वचालित रूप से दिनों, मिनटों और सेकंड के लिए मानों को बदल देता है।

6. इसके बाद आपम उपयोगकर्ता को उसकी जानकारी प्रदर्शित कर सकते हैं।

```
print(name, "has been alive for", days,"days", minutes, "minutes and",
seconds, "seconds! Wow!")
```



7. बधाई! आपने एक वास्तविक कार्यक्रम बनाया है जो एक उद्देश्य को पूरा करता है! इसे सेव करें और 'रन' और 'रन मॉड्यूल' पर जाकर इसे रन करें।

पायगेम के साथ पायथन में गेम कैसे प्रोग्राम करें

भाग 1. पायगेम स्थापित करना

- पायगेम डाउनलोड करें।
- इंस्टॉलर चलाएं।
- सत्यापित करें कि स्थापना ने काम किया है। एक पायथन टर्मिनल खोलें। "आयात पायगम" टाइप करें। यदि आपको कोई त्रुटि दिखाई नहीं देती है तो पायगेम को सफलतापूर्वक स्थापित किया गया था।

```
import pygame
```

भाग 2. एक मूल विंडो को सेट करना

- एक नई फ़ाइल खोलें।
- पायगेम इम्पोर्ट कैसे करें: पायगेम एक पुस्तकालय है जो ग्राफिक्स कार्यों तक पहुंच प्रदान करता है। अगर आप चाहते हैं ये फ़ंक्शन कैसे काम करते हैं, इस बारे में अधिक जानकारी नीचे डी गई है।

```
import pygame
from pygame.locals import *
```

- विंडो रिज़ॉल्यूशन सेट करें: आप स्क्रीन रिज़ॉल्यूशन के लिए एक वैश्विक वेरिएबल बना रहे होंगे ताकि गेम के कई हिस्सों में संदर्भित किया जा सके। फ़ाइल के शीर्ष पर ढूंढना भी आसान है, इसलिए इसे बाद में बदला जा सकता है। उन्नत परियोजनाओं के लिए, इस जानकारी को एक अलग फ़ाइल में रखना एक बेहतर विचार होगा।

```
resolution = (400,300)
```

- कुछ रंगों को परिभाषित करें: पायगेम में कलर होते हैं (आपबीजीए जो 0 और 255 के बीच के मानों में होता है। अल्फा मान (ए) वैकल्पिक है लेकिन अन्य कलर (लाल, नीला और हरा अनिवार्य हैं)।

```
white = (255,255,255)
black = (0,0,0)
red = (255,0,0)
```

- स्क्रीन को इनिशियलाइज़ करें: रिज़ॉल्यूशन वैरिएबल का उपयोग करें जिसे पहले परिभाषित किया गया था।

```
screen = pygame.display.set_mode(resolution)
```

- गेम लूप बनाएं: हमारे गेम के हर फ्रेम में कुछ क्रियाओं को दोहराएं। एक लूप बनाएं जो इन सभी क्रियाओं के माध्यम से हमेशा चक्र को दोहराएं।

```
while True:
```

- कलर पर्दा डालना।

```
screen.fill(white)
```

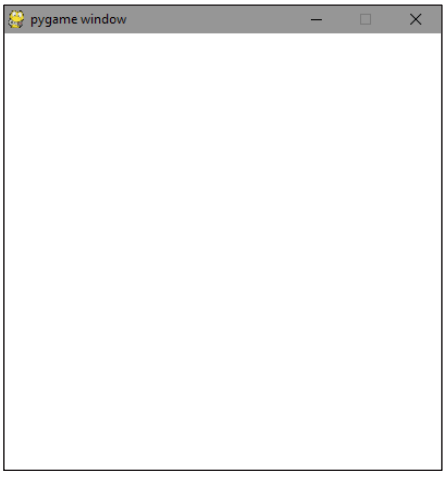
- स्क्रीन प्रदर्शित करें: यदि आप प्रोग्राम चलाते हैं, तो स्क्रीन सफेद हो जाएगी और फिर प्रोग्राम क्रैश हो जाएगा। ऐसा इसलिए है क्योंकि ऑपरेटिंग सिस्टम गेम में ईवेंट भेज रहा है और गेम उनके साथ कुछ नहीं कर रहा है। एक बार जब गेम को बहुत अधिक हैंडल न किए गए ईवेंट प्राप्त हो जाते हैं, तो यह क्रैश हो जाएगा।

```
while True:
    ...
    pygame.display.flip()
```

- इवेंट्स को संभालें: प्रत्येक फ्रेम में हुई सभी घटनाओं की एक सूची प्राप्त करें। आप ही जा रहे हैं देखभाल करने के लिए एक घटना के बारे में, छोड़ो घटना। यह तब होता है जब उपयोगकर्ता गेम विंडो बंद कर देता है।

यह हमारे प्रोग्राम को बहुत अधिक घटनाओं के कारण क्रैश होने से भी रोकता है।

```
while True:
      ...
      for event in pygame.event.get():
            if event.type == QUIT:
                  pygame.quit()
```



- कोशिश करके देखो! यहां बताया गया है कि कोड अब कैसा दिखना चाहिए:

```
import pygame
from pygame.locals import *
resolution = (400,300)
white = (255,255,255)
black = (0,0,0)
red = (255,0,0)
screen = pygame.display.set_mode(resolution)
while True:
      screen.fill(white)
      pygame.display.flip()
      for event in pygame.event.get():
            if event.type == QUIT:
                  pygame.quit()
```

भाग 3. गेम ऑब्जेक्ट कैसे बनाएं

- एक नया वर्ग और कंस्ट्रक्टर बनाएं: ऑब्जेक्ट के सभी गुण सेट करें। आप सभी संपत्तियों के लिए डिफ़ॉल्ट मान भी प्रदान कर रहे हैं।

```
class Ball:

        def init  (self, xPos = resolution[0] / 2, yPos = resolution[1]
      / 2, xVel = 1, yVel = 1, rad = 15):

            self.x = xPos

            self.y = yPos

            self.dx = xVel

            self.dy = yVel

            self.radius = rad

            self.type = "ball"
```

परिभाषित करें कि ऑब्जेक्ट को कैसे आकर्षित किया जाए: गेंद को एक सर्कल के रूप में खींचने के लिए और साथ ही ऑब्जेक्ट को आकर्षित करने के लिए फ़ंक्शन में सतह को पास करने के लिए कंस्ट्रक्टर में परिभाषित गुणों का उपयोग करें। सतह स्क्रीन ऑब्जेक्ट होगी जिसे पहले रिज़ॉल्यूशन का उपयोग करके बनाया गया था।

```
def draw(self, surface):

      pygame.draw.circle(surface, black, (self.x, self.y), self.radius)
```

- क्लास का एक उदाहरण बनाएं और साथ ही गेम लूप को प्रत्येक लूप में गेंद खींचने के लिए कहें।

```
ball = Ball()

while True:

      ...

      ball.draw(screen)
```

- ऑब्जेक्ट को मूव करें: एक फंक्शन बनाएं जो ऑब्जेक्ट की स्थिति को अपडेट करेगा। इस फ़ंक्शन को प्रत्येक गेम लूप में कॉल करें।

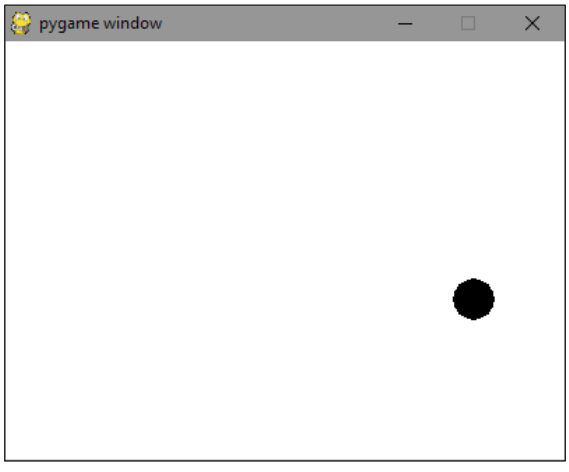
```
class Ball:

      ...

      def update(self):

            self.x += self.dx

            self.y += self.dy
```

- फ्रेम दर को सीमित करें: गेंद वास्तव में तेजी से आगे बढ़ेगी क्योंकि गेम लूप एक सेकंड में सैकड़ों बार चल रहा है। फ्रेम दर को 60 एफपीएस तक सीमित करने के लिए पायगम की घड़ी का प्रयोग करें।

```
clock = pygame.time.Clock()
while True:
    ...
    clock.tick(60)
```

- गेंद को स्क्रीन पर रखें: यदि गेंद स्क्रीन के किसी एक किनारे से टकराती है तो गेंद की दिशा को उलटने के लिए अपडेट फ़ंक्शन में चेक जोड़ें।

```
class Ball:
    ...
    def update(self):
        ...
    if (self.x <= 0 or self.x >= resolution[0]):
        self.dx *= -1
    if (self.y <= 0 or self.y >= resolution[1]):
        self.dy *= -1
```



- कोशिश करके देखो! यहां बताया गया है कि कोड अब कैसा दिखना चाहिए:

```
import pygame
from pygame.locals import *
resolution = (400,300)
white = (255,255,255)
```

इस में (स्वयं, xPos = संकल्प [0] / 2, yPos = संकल्प [1]

डीईएफ़

```
black = (0,0,0)
red = (255,0,0)
screen = pygame.display.set_mode(resolution)
class Ball:
        def init  (self, xPos = resolution[0] / 2, yPos = resolution[1]
      / 2, xVel = 1, yVel = 1, rad = 15):
            self.x = xPos
            self.y = yPos
            self.dx = xVel
            self.dy = yVel
            self.radius = rad
            self.type = "ball"
      def draw(self, surface):
            pygame.draw.circle(surface, black, (self.x, self.y), self.radius)
      def update(self):
            self.x += self.dx
            self.y += self.dy
      if (self.x <= 0 or self.x >= resolution[0]):
            self.dx *= -1
      if (self.y <= 0 or self.y >= resolution[1]):
            self.dy *= -1
ball = Ball()
clock = pygame.time.Clock()
while True:
      screen.fill(white)
      ball.draw(screen)
      ball.update()
      pygame.display.flip()
```

```
clock.tick(60)
for event in pygame.event.get():
    if event.type == QUIT:
        pygame.quit()
```

भाग 4. गेम का आयोजन करें

- सब कुछ व्यवस्थित करने के लिए कक्षाओं का उपयोग करें: खेल और अधिक जटिल होता जा रहा है। अपने कोड को व्यवस्थित करने के लिए वस्तु-उन्मुख तकनीकों का उपयोग करें।
- गेम लूप को एक क्लास में बनाएं: चूंकि हमारे गेम में अब आपके गेम ऑब्जेक्ट्स और फ़ंक्शंस सहित डेटा है, इसलिए यह आपके गेम लूप को क्लास में बदलने के लिए समझ में आता है।

```
class game():
```

- एक कंस्ट्रक्टर जोड़ें: यहां आप कुछ गेम ऑब्जेक्ट्स को इंस्टेंट करेंगे, हमारी स्क्रीन और क्लॉक बनाएंगे और पायगेम को इनिशियलाइज़ करेंगे। पाठ या ध्वनि जैसी कुछ विशेषताओं का उपयोग करने के लिए पायगेम को प्रारंभ करने की आवश्यकता होती है।

```
class game():
    def init  (self):
        pygame.init()
        self.screen = pygame.display.set_mode(resolution)
        self.clock = pygame.time.Clock()
```

- एक फंक्शन में इवेंट्स को संभालें।

```
class game():
    ...
    def handleEvents(self):
        for event in pygame.event.get():
            if event.type == QUIT:
                pygame.quit()
```

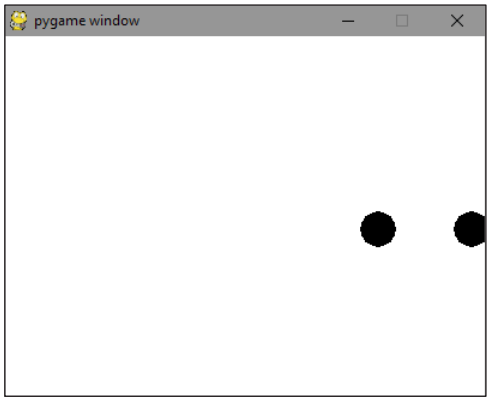
- गेम लूप को एक फंक्शन बनाएं: हर लूप में इवेंट हैंडलिंग फंक्शन को कॉल करें।

```
class game():
    ...
    def run(self):
        while True:
            self.handleEvents()
```

```
            self.screen.fill(white)
            self.clock.tick(60)
            pygame.display.flip()
```

- कई गेम ऑब्जेक्ट्स को हैंडल करें: अभी इस कोड को ड्रॉ को कॉल करना होता है और प्रत्येक फ्रेम पर हमारे ऑब्जेक्ट पर अपडेट करना होता है। यदि आपके पास बहुत सारी वस्तुएं हों तो यह गड़बड़ हो जाएगी। आइए अपनी वस्तु को एक सरणी में जोड़ें और फिर प्रत्येक लूप में सरणी में सभी वस्तुओं को अपडेट और ड्रा करें। अब आप आसानी से एक और वस्तु जोड़ सकते हैं और इसे एक अलग प्रारंभिक स्थिति दे सकते हैं।

```
class game():
        def init  (self):
            ...
            self.gameObjects = []
            self.gameObjects.append(Ball())
            self.gameObjects.append(Ball(100))
            ...
def run(self):
      while True:
            self.handleEvents()
            for gameObj in self.gameObjects:
                  gameObj.update()
            self.screen.fill(white)
            for gameObj in self.gameObjects:
                  gameObj.draw(self.screen)
                  self.clock.tick(60)
                  pygame.display.flip()
```



- आप कोशिश करके देख सकते हैं! यहां बताया गया है कि कोड अब कैसा दिखना चाहिए:

```
import pygame
from pygame.locals import *
resolution = (400,300)
white = (255,255,255)
black = (0,0,0)
red = (255,0,0)
screen = pygame.display.set_mode(resolution)
class Ball:
def init  (self, xPos = resolution[0] / 2, yPos = resolu- tion[1] /
2, xVel = 1, yVel = 1, rad = 15):
    self.x = xPos
    self.y = yPos
    self.dx = xVel
    self.dy = yVel
    self.radius = rad
    self.type = “ball”
def draw(self, surface):
pygame.draw.circle(surface, black, (self.x, self.y), self.ra- dius)
def update(self):
    self.x += self.dx
    self.y += self.dy
    if (self.x <= 0 or self.x >=
    resolution[0]): self.dx *= -1
if (self.y <= 0 or self.y >= resolution[1]):
    self.dy *= -1
class game():
def init  (self):
```

```
            pygame.init()
            self.screen = pygame.display.set_mode(resolution)
            self.clock = pygame.time.Clock()
            self.gameObjects = []
            self.gameObjects.append(Ball())
            self.gameObjects.append(Ball(100))
    def handleEvents(self):
        for event in pygame.event.get():
            if event.type == QUIT:
                pygame.quit()
    def run(self):
        while True:
            self.handleEvents()
    for gameObj in self.gameObjects:
        gameObj.update()
        self.screen.fill(white)
    for gameObj in self.gameObjects:
        gameObj.draw(self.screen)
        self.clock.tick(60)
        pygame.display.flip()
        game().run()
```

भाग 5. एक प्लेयर ऑब्जेक्ट को जोड़ना

- एक प्लेयर क्लास और कंस्ट्रक्टर बनाएं: आप एक और सर्कल बनाने जा रहे हैं जिसे माउस द्वारा नियंत्रित किया जाता है। कंस्ट्रक्टर में मानों को इनिशियलाइज़ करें। त्रिज्या एकमात्र महत्वपूर्ण मूल्य है।

```
class Player:

  def init  (self, rad = 20):
        self.x = 0
```

```
        self.y = 0
        self.radius = rad
```

- परिभाषित करें कि प्लेयर ऑब्जेक्ट को कैसे ड्रा करें: यह उसी तरह होने वाला है जैसे आपने अन्य गेम ऑब्जेक्ट को आकर्षित किया था।

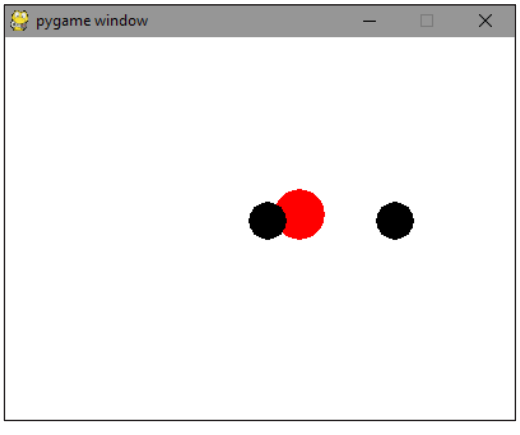
```
class Player:
    ...
    def draw(self, surface):
        pygame.draw.circle(surface, red, (self.x, self.y), self.ra-
        dius)
```

- प्लेयर ऑब्जेक्ट के लिए माउस नियंत्रण को जोड़ें: प्रत्येक फ़्रेम में माउस का स्थान जांचें और प्लेयर्स की ओब्जेक्ट्स की स्थिति को उस बिंदु पर सेट करें।

```
class Player:
    ...
    def update(self):
        cord = pygame.mouse.get_pos()
        self.x = cord[0]
        self.y = cord[1]
```

- गेमऑब्जेक्ट्स में प्लेयर ऑब्जेक्ट जोड़ें: एक नया प्लेयर इंस्टेंस बनाएं और इसे सूची में जोड़ें।

```
class game():
def init  (self):
    ...
    self.gameObjects.append(Player())
```



- आप भी कोशिश करके देखो! यहां बताया गया है कि कोड अब कैसा दिखना चाहिए:

```
import pygame

from pygame.locals import *

resolution = (400,300)

white = (255,255,255)

black = (0,0,0)

red = (255,0,0)

screen = pygame.display.set_mode(resolution)

class Ball:

        def init  (self, xPos = resolution[0] / 2, yPos = resolu-
      tion[1] / 2, xVel = 1, yVel = 1, rad = 15):

      self.x = xPos

      self.y = yPos

      self.dx = xVel

      self.dy = yVel

      self.radius = rad

      self.type = “ball”

    def draw(self, surface):

      pygame.draw.circle(surface, black, (self.x, self.y), self.ra-
      dius)

    def update(self):

      self.x += self.dx

      self.y += self.dy

    if (self.x <= 0 or self.x >= resolution[0]):

      self.dx *= -1

    if (self.y <= 0 or self.y >= resolution[1]):

      self.dy *= -1

class Player:

      def init  (self, rad = 20):
```

```
                self.x = 0
                self.y = 0
                self.radius = rad
                self.type = "player"
        def draw(self, surface):
                pygame.draw.circle(surface, red, (self.x, self.y), self.ra-
                dius)
        def update(self):
                cord = pygame.mouse.get_pos()
                self.x = cord[0]
                self.y = cord[1]
class game():
        def init  (self):
                pygame.init()
                self.screen = pygame.display.set_mode(resolution)
                self.clock = pygame.time.Clock()
                self.gameObjects = []
                self.gameObjects.append(Player())
                self.gameObjects.append(Ball())
                self.gameObjects.append(Ball(100))
        def handleEvents(self):
                for event in pygame.event.get():
                        if event.type == QUIT:
                                pygame.quit()
        def run(self):
                while True:
                        self.handleEvents()
        for gameObj in self.gameObjects:
                gameObj.update()
                self.screen.fill(white)
```

```
    for gameObj in self.gameObjects:
        gameObj.draw(self.screen)
        self.clock.tick(60)
        pygame.display.flip()
        game().run()
```

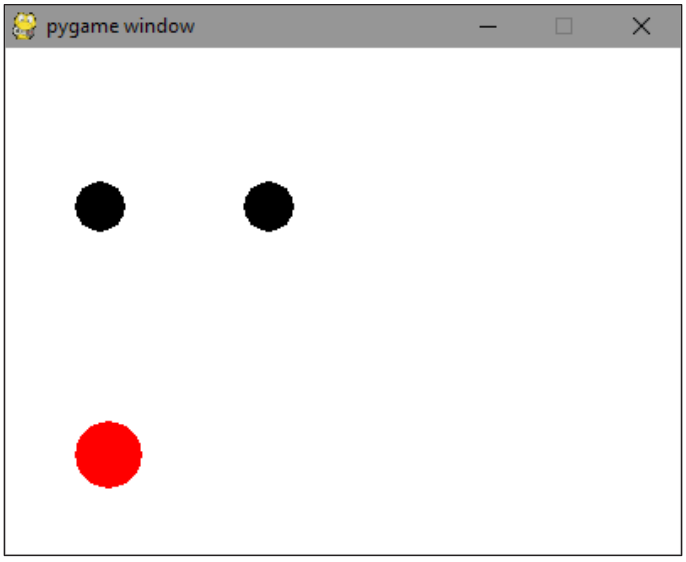
भाग 6. प्लेयर्स के साथ ऑब्जेक्ट का आदान-प्रदान करना

- अद्यतन कार्य बदलें: वस्तुओं को बातचीत करने के लिए, उन्हें एक दूसरे तक पहुंच की आवश्यकता होगी। गेमऑब्जेक्ट्स सूची में पास करने के लिए अपडेट में एक और पैरामीटर जोड़ें। आपको इसे प्लेयर ऑब्जेक्ट और बॉल ऑब्जेक्ट दोनों में जोड़ना होगा। यदि आपके पास बहुत सारी गेम ऑब्जेक्ट हैं, तो इनहेरिटेंस आपको अपने सभी मेथड सिग्नेचर को समान रखने में मदद कर सकता है।

```
class Ball:
    ...
    def update(self, gameObjects):
    ...
class Player:
    ...
    def update(self, gameObjects):
```

- प्लेयर और बॉल के बीच टकराव की जांच करें: सभी खेल वस्तुओं के माध्यम से जाएं और जांचें कि क्या ऑब्जेक्ट्स का प्रकार बॉल है। फिर दो वस्तुओं की त्रिज्या और दूरी सूत्र का उपयोग करके जाँच करें कि क्या वे टकरा रही हैं। मंडलियों पर टकराव की जाँच करना वास्तव में आसान होता है। यह सबसे बड़ा कारण है कि आपने इस खेल के लिए किसी अन्य आकार का उपयोग नहीं किया है।

```
class Player:
    ...
    def update(self, gameObjects):
        ...
        for gameObj in gameObjects:
            if gameObj.type == "ball":
                if (gameObj.x - self.x)**2 + (gameObj.y - self-
                .y)**2 <= (gameObj.radius + self.radius)**2:
```



- यदि प्लेयर "हिट" हो जाता है तो खेल समाप्त करें: चलो अभी के लिए खेल छोड़ दें।

```
if (gameObj.x - self.x)**2 + (gameObj.y - self.y)**2 <= (gameObj.radius
+ self.radius)**2:

    pygame.quit()
```

- कोशिश करके देखो! यहां बताया गया है कि कोड अब कैसा दिखना चाहिए:

```
import pygame

from pygame.locals import *

resolution = (400, 300)

white = (255,255,255)

black = (0,0,0)

red = (255,0,0)

screen = pygame.display.set_mode(resolution)

class Ball:

      def init  (self, xPos = resolution[0] / 2, yPos = resolution[1]
    / 2, xVel = 1, yVel = 1, rad = 15):

        self.x = xPos

        self.y = yPos

        self.dx = xVel

        self.dy = yVel
```

```
            self.radius = rad
            self.type = "ball"
      def draw(self, surface):
            pygame.draw.circle(surface, black, (self.x, self.y), self.ra-
            dius)
      def update(self, gameObjects):
            self.x += self.dx
            self.y += self.dy
      if (self.x <= 0 or self.x >= resolution[0]):
            self.dx *= -1
      if (self.y <= 0 or self.y >= resolution[1]):
            self.dy *= -1
class Player:
       def init  (self, rad = 20):
            self.x = 0
            self.y = 0
            self.radius = rad
            self.type = "player"
      def draw(self, surface):
            pygame.draw.circle(surface, red, (self.x, self.y), self.ra-
            dius)
      def update(self, gameObjects):
            cord = pygame.mouse.get_pos()
            self.x = cord[0]
            self.y = cord[1]
      for gameObj in gameObjects:
            if gameObj.type == "ball":
                  if (gameObj.x - self.x)**2 + (gameObj.y - self.y)**2 <=
                  (gameObj.radius + self.radius)**2:
                        pygame.quit()
```

```
class game():
        def init  (self):
            pygame.init()
            self.screen = pygame.display.set_mode(resolution)
            self.clock = pygame.time.Clock()
            self.gameObjects = []
            self.gameObjects.append(Player())
            self.gameObjects.append(Ball())
            self.gameObjects.append(Ball(100))
      def handleEvents(self):
            for event in pygame.event.get():
                  if event.type == QUIT:
                        pygame.quit()
      def run(self):
            while True:
                  self.handleEvents()
      for gameObj in self.gameObjects:
            gameObj.update(self.gameObjects)
            self.screen.fill(white)
      for gameObj in self.gameObjects:
            gameObj.draw(self.screen)
            self.clock.tick(60)
            pygame.display.flip()
game().run()
```

भाग 7. ऑब्जेक्ट बनाने के लिए गेम कंट्रोलर जोड़ना

- गेम कंट्रोलर क्लास बनाएं: गेम कंट्रोलर गेम को "रनिंग" करने के लिए जिम्मेदार होते हैं। यह हमारे खेल वर्ग से अलग है जो हमारी सभी वस्तुओं को चित्रित करने और अद्यतन करने के लिए जिम्मेदार होते हैं। खेल को कठिन बनाने के लिए नियंत्रक समय-समय पर स्क्रीन पर एक और गेंद जोड़ देता हैं। इसमें एक कंस्ट्रक्टर को जोड़ें और कुछ बुनियादी मूल्यों को इनिशियलाइज़ करें। अंतराल एक और गेंद जोड़े जाने से पहले का समय होगा।

```
class GameController:

        def init  (self, interval = 5):

            self.inter = interval

            self.next = pygame.time.get_ticks() + (2 * 1000)

            self.type = "game controller"
```

- अद्यतन फ़ंक्शन जोड़ें: यह जांच करेगा कि गेंद को जोड़े जाने के बाद या खेल की शुरुआत से कितना समय बीत चुका है। यदि समय अंतराल से अधिक है तो आप समय को रीसेट करेंगे और एक गेंद जोड़ेंगे।

```
class GameController:

      ...

      def update(self, gameObjects):

            if self.next < pygame.time.get_ticks():

                  self.next = pygame.time.get_ticks() + (self.inter * 1000)

                  gameObjects.append(Ball())
```

- गेंदों को रैंडम वेलोसिटी दें: खेल को हर बार अलग बनाने के लिए आपको रैंडम संख्याओं का उपयोग करना होगा। हालांकि, गेंदों का वेलोसिटी अब एक पूर्णांक के बजाय एक अस्थायी बिंदु संख्या है।

```
class GameController:

      ...

      def update(self, gameObjects):

            if self.next < pygame.time.get_ticks():

                  self.next = pygame.time.get_ticks() + (self.inter * 1000)

                  gameObjects.append(Ball(xVel=random()*2, yVel=ran-
                  dom()*2))
```

- ड्रा फ़ंक्शन को ठीक करें: ड्रा फ़ंक्शन फ़्लोट्स को स्वीकार नहीं करेगा। आइए गेंदों को खींचे जाने से पहले गेंद की स्थिति को पूर्णांक में बदलें।

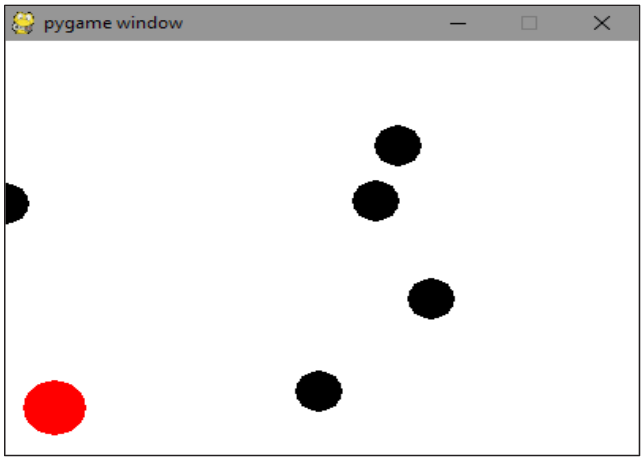
```
class Ball:

      ...

      def draw(self, surface):

                  pygame.draw.circle(surface, black, (int(self.x), int(-
                  self.y)), self.radius)
```

- गेम कंट्रोलर के लिए ड्रॉ विधि को परिभाषित करें: चूंकि यह एक गेम ऑब्जेक्ट है, मुख्य लूप इसे ड्रा करने का प्रयास करेगा। आपको ड्रॉ फ़ंक्शन को परिभाषित करने की आवश्यकता होगी जो कुछ भी नहीं करता है ताकि गेम क्रैश न हो।

```
class GameController:
    ...
    def draw(self, screen):
    pass
```

- गेम कंट्रोलर को गेमऑब्जेक्ट्स में जोड़ें और 2 गेंदों को हटा दें: गेम को अब हर पांच सेकंड में एक गेंद को स्पॉन करना चाहिए।

```
class game():
    def init  (self):
        ...
        self.gameObjects = [] self.gameObjects.append(GameController())
        self.gameObjects.append(Player())
```



- कोशिश करके देखें! यहां बताया गया है कि कोड अब कैसा दिखना चाहिए:

```
import pygame
from random import random
from pygame.locals import *
resolution = (400,300)
```

```
white = (255,255,255)

black = (0,0,0)

red = (255,0,0)

screen = pygame.display.set_mode(resolution)

class Ball:

        def init  (self, xPos = resolution[0] / 2, yPos = resolution[1]
      / 2, xVel = 1, yVel = 1, rad = 15):

            self.x = xPos

            self.y = yPos

            self.dx = xVel

            self.dy = yVel

            self.radius = rad

            self.type = “ball”

      def draw(self, surface):

            pygame.draw.circle(surface,   black,   (int(self.x),   int(-
            self.y)), self.radius)

      def update(self, gameObjects):

            self.x += self.dx

            self.y += self.dy

      if (self.x <= 0 or self.x >= resolution[0]):

            self.dx *= -1

      if (self.y <= 0 or self.y >= resolution[1]):

            self.dy *= -1

class Player:

        def init  (self, rad = 20):

            self.x = 0

            self.y = 0

            self.radius = rad

            self.type = “player”
```

```
    def draw(self, surface):
        pygame.draw.circle(surface, red, (self.x, self.y), self.ra-
        dius)
    def update(self, gameObjects):
        cord = pygame.mouse.get_pos()
        self.x = cord[0]
        self.y = cord[1]
    for gameObj in gameObjects:
        if gameObj.type == "ball":
            if (gameObj.x - self.x)**2 + (gameObj.y - self.y)**2 <=
            (gameObj.radius + self.radius)**2:
                pygame.quit()
class GameController:
    def init  (self, interval = 5):
        self.inter = interval
        self.next = pygame.time.get_ticks() + (2 * 1000)
        self.type = "game controller"
    def update(self, gameObjects):
        if self.next < pygame.time.get_ticks():
            self.next = pygame.time.get_ticks() + (self.inter * 1000)
            gameObjects.append(Ball(xVel=random()*2,      yVel=ran-
            dom()*2))
    def draw(self, screen):
        pass
class game():
    def init  (self):
        pygame.init()
        self.screen = pygame.display.set_mode(resolution)
        self.clock = pygame.time.Clock()
        self.gameObjects = []
```

```
            self.gameObjects.append(GameController())
            self.gameObjects.append(Player())
      def handleEvents(self):
            for event in pygame.event.get():
                  if event.type == QUIT:
                        pygame.quit()
      def run(self):
            while True:
                  self.handleEvents()
      for gameObj in self.gameObjects:
            gameObj.update(self.gameObjects)
            self.screen.fill(white)
      for gameObj in self.gameObjects:
            gameObj.draw(self.screen)
            self.clock.tick(60)
            pygame.display.flip()
game().run()
```

भाग 8. स्कोर और गेम ओवर को जोड़ना

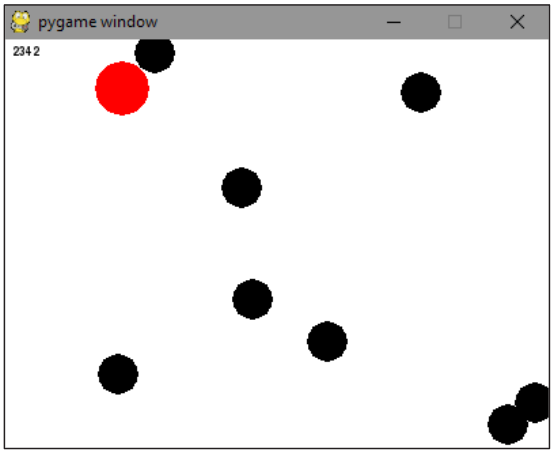
- गेम कंट्रोलर क्लास में स्कोर जोड़ें: एक फॉन्ट ऑब्जेक्ट और एक स्कोर वेरिएबल बनाएं। आप स्कोर प्रदर्शित करने के लिए हर फ्रेम में फॉन्ट खींचेंगे और अपडेट में हर फ्रेम में स्कोर बढ़ाएंगे।

```
class GameController:
        def init  (self, interval = 5):
            ...
            self.score = 0
            self.scoreText = pygame.font.Font(None, 12)
      def update(self, gameObjects):
            ...
      self.score += 1
            ...
            स्व.स्कोर += 1
```

```
    def draw(self, screen):
        screen.blit(self.scoreText.render(str(self.score),      True,
        black), (5,5))
```

- मॉडिफाई करें कि खेल कैसे समाप्त होता है: जब प्लेयर को टक्कर का पता चलता है तो चलो छोड़ दें। इसके बजाय आप प्लेयर में एक वैरिएबल सेट करेंगे जिसे गेम चेक कर सकता है। जब गेमओवर सेट हो, तो ऑब्जेक्ट को अपडेट करना बंद कर दें। यह सब कुछ स्थिर कर देगा ताकि खिलाड़ी देख सके कि क्या हुआ और अपने स्कोर की जांच कर सके। ध्यान दें कि ऑब्जेक्ट अभी भी खींचे जा रहे हैं, बस अपडेट नहीं किए गए हैं।

```
class Player:
    def init  (self, rad = 20):
        ...
        self.gameOver = False
    def update(self, gameObjects):
        ...
    for gameObj in gameObjects:
        if gameObj.type == "ball":
            if (gameObj.x - self.x)**2 + (gameObj.y - self.y)**2 <=
            (gameObj.radius + self.radius)**2:
                self.gameOver = True
class game():
    def init  (self):
        ...
        self.gameOver = False
    def run(self):
        while True:
            self.handleEvents()
    if not self.gameOver:
        for gameObj in self.gameObjects:
            gameObj.update(self.gameObjects)
    if gameObj.type == "player":
        self.gameOver = gameObj.gameOver
```



- आप भी कोशिश करके देख सकते हैं! यहां बताया गया है कि तैयार कोड अब कैसा दिखना चाहिए:

```
import pygame
from random import random
from pygame.locals import *
resolution = (400,300)
white = (255,255,255)
black = (0,0,0)
red = (255,0,0)
screen = pygame.display.set_mode(resolution)
class Ball:
        def init  (self, xPos = resolution[0] / 2, yPos = resolution[1]
    / 2, xVel = 1, yVel = 1, rad = 15):
            self.x = xPos
            self.y = yPos
            self.dx = xVel
            self.dy = yVel
            self.radius = rad
            self.type = "ball"
    def draw(self, surface):
            pygame.draw.circle(surface,  black,  (int(self.x),  int(-
            self.y)), self.radius)
```

```
      def update(self, gameObjects):
            self.x += self.dx
            self.y += self.dy
      if (self.x <= 0 or self.x >= resolution[0]):
            self.dx *= -1
      if (self.y <= 0 or self.y >= resolution[1]):
            self.dy *= -1
class Player:
        def init  (self, rad = 20):
            self.x = 0
            self.y = 0
            self.radius = rad
            self.type = "player"
            self.gameOver = False
      def draw(self, surface):
            pygame.draw.circle(surface, red, (self.x, self.y), self.ra-
            dius)
      def update(self, gameObjects):
            cord = pygame.mouse.get_pos()
            self.x = cord[0]
            self.y = cord[1]
      for gameObj in gameObjects:
            if gameObj.type == "ball":
                  if (gameObj.x - self.x)**2 + (gameObj.y - self.y)**2 <=
                  (gameObj.radius + self.radius)**2:
                        self.gameOver = True
class GameController:
        def init  (self, interval = 5):
            self.inter = interval
```

```
            self.next = pygame.time.get_ticks() + (2 *
            1000) self.type = "game controller"
            self.score = 0
            self.scoreText = pygame.font.Font(None, 12)
      def update(self, gameObjects):
            if self.next < pygame.time.get_ticks():
                  self.next = pygame.time.get_ticks() + (self.inter * 1000)
                  gameObjects.append(Ball(xVel=random()*2, yVel=ran-
                  dom()*2))
                  self.score += 1
      def draw(self, screen):
            screen.blit(self.scoreText.render(str(self.score),
            True, black), (5,5))
class game():
        def init  (self):
            pygame.init()
            self.screen = pygame.display.set_mode(resolution)
            self.clock = pygame.time.Clock()
            self.gameObjects = []
            self.gameObjects.append(GameController())
            self.gameObjects.append(Player())
            self.gameOver = False
      def handleEvents(self):
            for event in pygame.event.get():
                  if event.type == QUIT:
                        pygame.quit()
      def run(self):
            while True:
                  self.handleEvents()
```

```
        if not self.gameOver:
              for gameObj in self.gameObjects:
                    gameObj.update(self.gameObjects)
        if gameObj.type == "player":
              self.gameOver = gameObj.gameOver
              self.screen.fill(white)
        for gameObj in self.gameObjects:
              gameObj.draw(self.screen)
              self.clock.tick(60)
              pygame.display.flip()
game().run()
```

डीप लर्निंग के लिए पायथन एनवायरनमेंट कैसे सेट करें

यदि आपके पास सही संसाधन नहीं हैं, तो किसी विशेष प्लेटफॉर्म पर पायथन-आधारित मशीन लर्निंग (एमएल) वातावरण स्थापित करना कुछ चुनौतीपूर्ण हो सकता है। आपको पहले पायथन को इंस्टाल करना होगा और फिर कई अलग-अलग पैकेज को जोड़ना होगा। चीजों को आसान बनाने के लिए, यहां हम एनाकोंडा के माध्यम से एक पायथन एमएल विकास पर्यावरण की स्थापना और सेटअप के माध्यम से चलते हैं।

भाग 1. एनाकोंडा की स्थापना

- एनाकोंडा डाउनलोड करें। ये एनाकोंडा एक वैज्ञानिक वातावरण है जिसे पायथन के लिए डिज़ाइन किया गया है। ये मुफ्त है और प्रयोग करने में बेहद आसान है।
    - एनाकोंडा होमपेज पर जाएं। मेनू पर, 'एनाकोंडा' पर क्लिक करें और फिर ड्रॉप-डाउन सूची से 'डाउन-लोड' चुनें। यह आपको डाउनलोड पेज पर ले जाएगा। पायथन का चयन करें 3.5 और 'ग्राफिक इंस्टालर' चुनें
    - एनाकोंडा पायथन पैकेज अब आपके वर्कस्टेशन पर डाउनलोड किया जाना चाहिए
- सुनिश्चित करें कि आपके पास प्रशासनिक अधिकार हैं। इससे पहले कि हम इस चरण को शुरू करें, आपके पास अपने सिस्टम पर सॉफ़्टवेयर स्थापित करने के लिए पर्याप्त प्रशासनिक अधिकार होने चाहिए। एक बार पुष्टि हो जाने के बाद, आप इससे आगे बढ़ सकते हैं।
- एनाकोंडा स्थापित करें। डाउनलोड की गई फ़ाइल पर डबल-क्लिक करें। इसे इंस्टॉलेशन विज़ार्ड लॉन्च करना पड़ेगा। बस विज़ार्ड के अनुसार वेरिएबल ण दर वेरिएबल निर्देशों का पालन करें। यह काफी सरल और सीधी प्रक्रिया होती है। संस्थापन प्रक्रिया में आपकी हार्ड ड्राइव पर सभी 10 मिनट का समय 1 जीबी से थोड़ा अधिक होना चाहिए।

- सेटअप समाप्त करें। यह पुष्टि करेगा कि आपका एनाकोंडा पायथन वातावरण अद्यतित संस्करण है या नहीं। एनाकोंडा सॉफ्टवेयर एनाकोंडा नेविगेटर के नाम से जाने जाने वाले ग्राफिकल टूल्स के पूरे सूट के साथ आता है। आप एनाकोंडा नेविगेटर को एप्लिकेशन लॉन्चर के माध्यम से लॉन्च कर सकते हैं। एनाकोंडा नेविगेटर के बारे में अधिक जानने के लिए, आप यहां विस्तृत दस्तावेज देख सकते हैं। अभी के लिए, हम एनाकोंडा पर्यावरण के लिए कमांड लाइन से शुरुआत करने का सुझाव देते हैं जिसे कोंडा कहा जाता है।

भाग 2 कोंडा सेट करें

- सुनिश्चित करें कि कोंडा पहले से स्थापित है: कोंडा सरल, तेज और कुशल होता है। त्रुटि संदेशों को आसानी से देखा जा सकता है और आप आसानी से पुष्टि कर सकते हैं कि आपका काम करने का माहौल स्थापित है और इरादा के अनुसार काम कर रहा है।
    - इसके बाद टर्मिनल या कमांड लाइन विंडो खोलें। यह पुष्टि करने के लिए कि क्या कोंडा सही तरीके से स्थापित है या नहीं , टाइप करें:

```
conda -V

// You should see something similar to this
```

    - यह पुष्टि करने के लिए कि क्या पायथन सही तरीके से स्थापित है या नहीं, इसके लिए ये आप टाइप कर सकते हैं:

- `python -V`
    - आपको नीचे की रेखा कुछ इसी तरह की दिखनी चाहिए:

- `Python 3.5.2:: Anaconda 4.2.0 (x86_64)`
- इसे नवीनतम संस्करण में अपडेट करें: अपडेट की पुष्टि के लिए आपको कुछ अतिरिक्त पैकेज स्थापित करने पड़ सकते हैं। यह पुष्टि करने के लिए कि आपका कोंडा वातावरण नवीनतम संस्करण उपलब्ध है, टाइप करें-

भाग 3 सत्यापित करें कि साई-पाई पर्यावरण काम कर रहा है

- सेटअप साई-पाई: अब आपके साई-पाई परिवेश की पुष्टि करने का समय आ गया है। निम्नलिखित स्क्रिप्ट इष्टतम मशीन लर्निंग विकास के लिए आवश्यक प्रमुख साई-पाई पुस्तकालयों के संस्करण संख्या को प्रिंट करती है। कुछ बुनियादी पुस्तकालयों में शामिल हैं – साई-पाई, मेटप्लोटलिब, नम-पे, पांडास, स्किकिट-लर्न और स्टैट्स मॉडल्स।

- एक पायथन स्क्रिप्ट लिखें: इसके लिए आप या तो सीधे कमांड के बाद 'पायथन' टाइप करना चुन सकते हैं। या आप टेक्स्ट एडिटर का उपयोग कर सकते हैं और नीचे दी गई स्क्रिप्ट को अपने एडिटर में पेस्ट कर सकते हैं। हम दूसरे विकल्प की सलाह देते हैं।

```
# scipy
import scipy
print('scipy: %s' % scipy.  version  )
# numpy
```

```
import numpy
print('numpy: %s' % numpy.  version  )
# matplotlib
import matplotlib
print('matplotlib: %s' % matplotlib.  version  )
# pandas
import pandas
print('pandas: %s' % pandas.  version  )
# statsmodels
import statsmodels
print('statsmodels: %s' % statsmodels.  version  )
# scikit-learn
import sklearn
print('sklearn: %s' % sklearn.  version  )
```

- पायथन स्क्रिप्ट रन करें: स्क्रिप्ट को फ़ाइल नाम 'versions.py' से सेव करें। कमांड लाइन पर वापस, आप अपनी निर्देशिका को उस पथ में बदल सकते हैं जहां आपने स्क्रिप्ट फ़ाइल सेव की थी। इसके बाद आप टाइप कर सकते हैं -

```
python versions.py
# The output should look like this
scipy: 0.18.1
numpy: 1.11.1
matplotlib: 1.5.3
pandas: 0.18.1
statsmodels: 0.6.1
sklearn: 0.17.1
```

भाग 4. अपडेट स्किकिट-लर्न लाइब्रेरी

- समझें कि विज्ञान-सीखना क्या है। स्किकिट-लर्न पायथन में सबसे लोकप्रिय मशीन लाइब्रेरी होती है जिससे आप बच नहीं सकते है। यहां, हम पुस्तकालय को अद्यतन करने के लिए अनुसरण किए जाने वाले वेरिएबल पर जाते हैं।

- नवीनतम उपलब्ध संस्करण को शामिल करने के लिए स्किकिट-लर्न को अपडेट करें: एक विशिष्ट लाइब्रेरी को कोंडा कमांड लाइन के माध्यम से अपडेट किया जा सकता है। नीचे एक उदाहरण दिया गया है कि स्किकिट को कैसे अपडेट किया जाए- उपलब्ध नवीनतम संस्करण के बारे में जानें। एक बार टर्मिनल में टाइप करें-

```
conda update scikit-learn
```

  - आपके पास केवल एक विशिष्ट पुस्तकालय को अद्यतन करने का विकल्प भी है-

```
conda install -c anaconda scikit-learn=0.18.1
```

- पुष्टि करें कि इंस्टॉलेशन सफल रहा: यह पुष्टि करने के लिए कि क्या इंस्टॉलेशन सफलतापूर्वक पूरा हुआ और स्किकिट-लर्न अपडेट किया गया था, आप वर्जन.पी स्क्रिप को फिर से चला सकते हैं।-

```
python versions.py

# You should see the following output –

scipy: 0.18.1

numpy: 1.11.3

matplotlib: 1.5.3

pandas: 0.18.1

statsmodels: 0.6.1

sklearn: 0.18.1
```

भाग 5. डीप लर्निंग लाइब्रेरी को इंस्टॉल करें

- अपनी स्टैक आवश्यकताओं को समझें: यहां, हम पायथन पुस्तकालयों को स्थापित करने का प्रयास करते हैं जिनका उपयोग गहन सीखने के लिए किया जाता है। विशेष रूप से, टेंसरफ्लो, थेनो और केरस। हम गहन शिक्षण पहल के लिए केरस का उपयोग करने की सलाह देते हैं। यह ध्यान रखना महत्वपूर्ण है कि केरस को सिस्टम में केवल थीनो या टेन्सरफ्लो स्थापित करने की आवश्यकता होती है। ये दोनों अब इस्तेमाल नहीं किये जाते हैं। साथ ही, कुछ विंडो मशीनों को टेंसरफ्लो को स्थापित करने में समस्या का सामना करना पड़ सकता है।

- थेनो इंस्टोल करें: आप थेनो डीप लर्निंग लाइब्रेरी का उपयोग करके स्थापित कर सकते हैं-

```
conda install theano
```

- यदि आप टेंसरफ्लो डीप लर्निंग लाइब्रेरी का उपयोग करना चुनते हैं, तो आप निम्न कमांड का उपयोग कर सकते हैं (विंडोज को छोड़कर सभी) -

```
conda install theano
```

- आपके पास अपने प्लेटफ़ॉर्म के लिए पाइप और टेंसरफ़्लो के एक विशेष संस्करण का उपयोग करके इंस्टॉलेशन को पूरा करने का विकल्प हो सकता है। इसके बारे में अधिक जानने के लिए, टेंसरफ़्लो के लिए इंस्टॉलेशन निर्देशों पर जाएं।

- केरस इंस्टोल करें: आप केरस को कमांड के माध्यम से स्थापित कर सकते हैं-

```
pip install keras
```

- अपने स्टैक में अतिरिक्त टूल जोड़ें: आप कोड ट्रैकिंग, ऑटोमेशन और ऑटो-डॉक्यूमेंटेशन के लिए अपने डीप लर्निंग स्टैक में अतिरिक्त टूल जोड़ सकते हैं। आप अपनी टीम के लिए इसे आसान बनाने के लिए गहन शिक्षण एसडीके और निगरानी उपकरण एकीकृत कर सकते हैं।
- सुनिश्चित करें कि आपने सही संस्करण स्थापित किया है: एक बार पूरा हो जाने पर, पुष्टि करना एक अच्छा विचार है कि आपका वातावरण गहन सीखने के लिए ठीक से स्थापित है और इच्छित के अनुसार काम कर रहा है।
    - इसके लिए आप एक स्क्रिप्ट बनाने की आवश्यकता है जो प्रत्येक पुस्तकालय के लिए संस्करण संख्या को प्रिंट करती है, जैसा कि हमने पहले पूरा किया था साई-पाई वातावरण के लिए अभ्यास के समान है।

```
# theano

import theano

print('theano: %s' % theano.  version  )

#tensorflow

importtensorflow

print('tensorflow: %s' % tensorflow._ ersion___

# keras

import keras

print('keras: %s' % keras.  version  )
```

    - आप स्क्रिप्ट को इस रूप में सेव कर सकते हैं 'deep_versions.py' नाम की एक फ़ाइल और फिर स्क्रिप्ट का उपयोग करके चलाएँ:

```
python deep_versions.py

# You should see the following output -

theano: 0.8.2.dev-

901275534cbfe3fbbe290ce85d1abf8bb9a5b203 tensorflow: 0.12.1

Using TensorFlow backend.

keras: 1.2.1
```

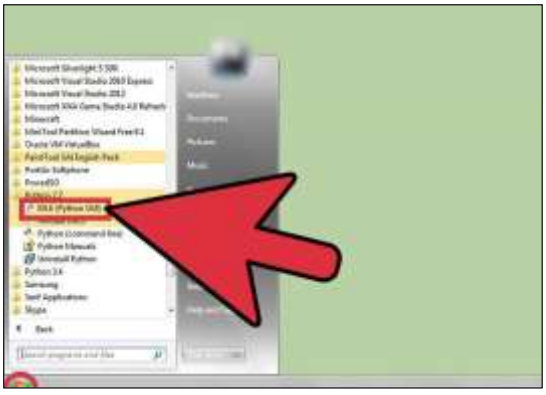
- पायथन सीखना शुरू करें: बस, अब आप अपने पसंदीदा डीप-लर्निंग स्टैक का उपयोग करके मॉडल बनाना शुरू कर सकते हैं या विकासशील मॉडल के साथ शुरुआत कर सकते हैं। जब आप अभी शुरुआत कर रहे हैं तो पायथन और डीप लर्निंग डराने वाला हो सकता है, लेकिन सरल मॉडल से शुरू करें और फिर अधिक जटिल मॉडल बनाना शुरू करें। आप गिटहब पर ढेरों उपयोगी संसाधन पा सकते हैं।

पायथन में जियोमेट्रिक पैटर्न कैसे प्रोग्राम करें

प्रोग्रामिंग कभी-कभी उबाऊ और थकाऊ हो सकती है, खासकर जब प्रोग्रामिंग कक्षाएं लेने की बात आती है। अक्सर आप जो प्रोग्राम बनाते हैं, वह सब कुछ एक ब्लैक लिटिल विंडो में आउटपुट करता है और वह यह है। प्रोग्रामिंग भाषा पायथन में आप टर्टल ग्राफिक्स नामक किसी चीज़ का उपयोग कर सकते हैं जो आपको कुछ बहुत अच्छे प्रोग्राम बनाने की अनुमति देता है। किसी भी कौशल स्तर का प्रोग्रामर इसका उपयोग पायथन में एक शांत जियोमेट्रिक पैटर्न बनाने के लिए कर सकता है।

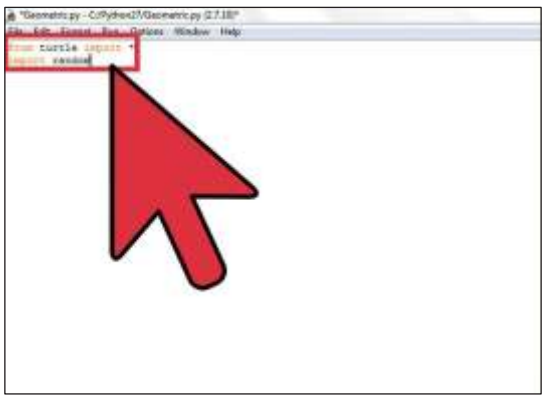
चरण

1. पायथन कंपाइलर डाउनलोड करें: संस्करण 2.7 डाउनलोड करना सुनिश्चित करें।

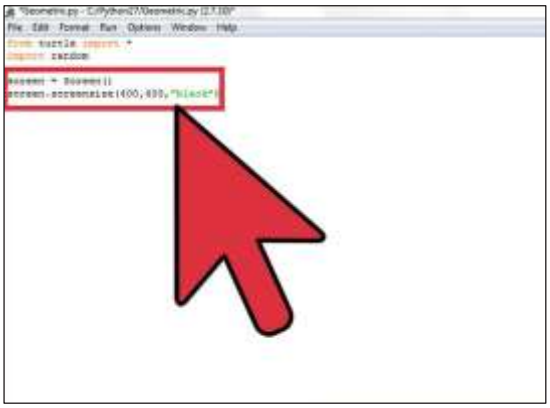


2. पायथन शैल खोलें: पायथन 2.7 फ़ोल्डर के अंतर्गत जाएं और "आईडीएलई (पायथन जीयूआई) पर क्लिक करें। इसे इस तरह एक पायथन शैल के साथ पॉप अप करना चाहिए।

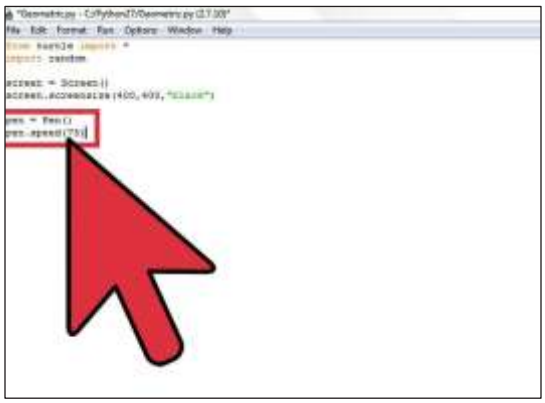
3. शेल से एक नई फ़ाइल प्रारंभ करें: इसके ऊपरी बाएं कोने में फ़ाइल पर क्लिक करें और ड्रॉप डाउन पर “नई फ़ाइल” पर क्लिक करें। यह एक बिना शीर्षक वाली फ़ाइल खोलेगा जहां आप अपना प्रोग्राम लिख सकते हैं।



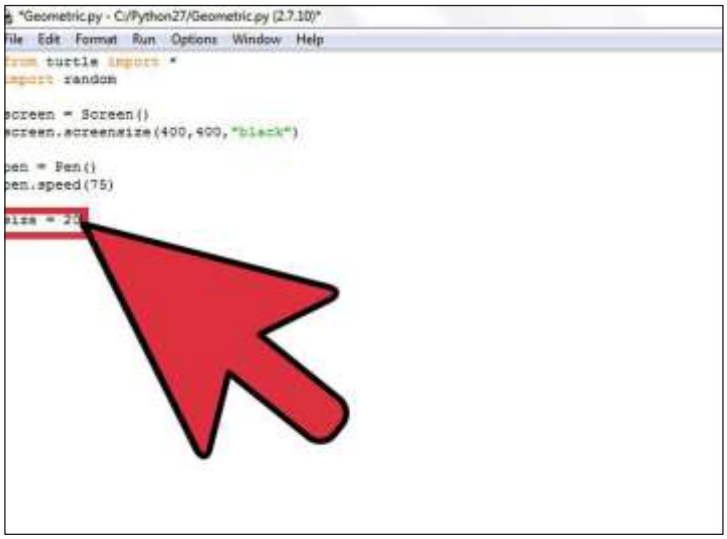
4. टर्टल ग्राफिक्स आयात करें: टर्टल ग्राफिक्स का उपयोग करने में सक्षम होने के लिए आपको उन्हें अपने कार्यक्रम में आयात करने की आवश्यकता होती है। आप इसे अपने कोड की पहली पंक्ति में देख सकते हैं। आप बस इस तरह "टर्टल आयात से *" टाइप करें। अपने प्रोग्राम को बेतरतीब ढंग से उत्पन्न कलर देने के लिए, अगली पंक्ति में आपको "इम्पोर्ट रैंडम" टाइप करना होगा।



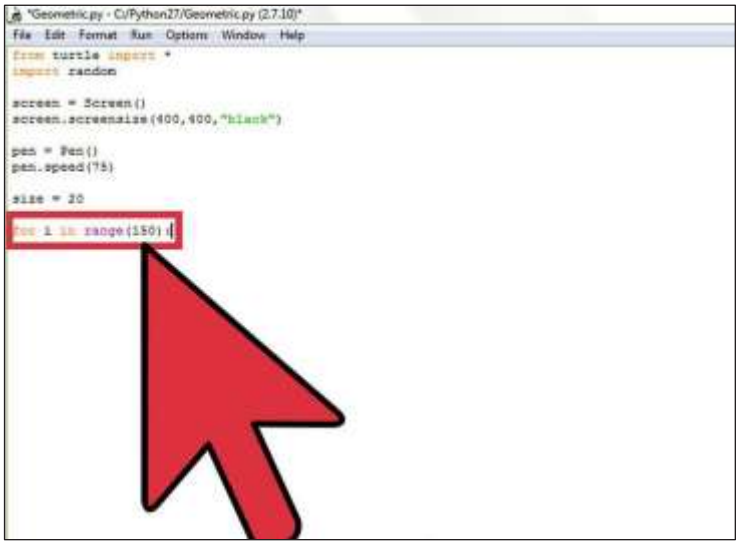
5. आप अपने प्रोग्राम में एक स्क्रीन बनाएं: अपने प्रोग्राम में ग्राफिक्स रखने के लिए आपको उन्हें प्रदर्शित करने के लिए एक स्क्रीन बनानी होगी। आप एक वैरिएबल बनाकर ऐसा कर सकते हैं (वेरिएबल स्क्रीन को नाम देना सबसे अच्छा है) और इसे फंक्शन के बराबर सेट करना होगा "स्क्रीन ()"। आपको स्क्रीन का आकार भी सेट करने की आवश्यकता हो सकती है। अगली पंक्ति में आप अपनी स्क्रीन के लिए बनाए गए वेरिएबल नाम लेते हैं और स्क्रीनसाइज फ़ंक्शन का उपयोग करते हैं। उदाहरण के लिए: स्क्रीन.स्क्रीनसाइज (400,400,"ब्लैक")। कोष्ठक के अंदर स्क्रीन की ऊंचाई, चौड़ाई और पृष्ठभूमि का कलर निर्धारित कर रहा है।



6. जियोमेट्रिक पैटर्न को आकर्षित करने के लिए एक पेन बनाएं: स्क्रीन के साथ पिछले चरण की तरह आप "पेन ()" फ़ंक्शन के बराबर एक वेरिएबल (भ्रम को बचाने के लिए सबसे अच्छा नामित पेन) सेट कर सकते हैं। अगली पंक्ति में आप अंतिम चरण के समान गति फ़ंक्शन का उपयोग करके पेन की गति निर्धारित कर सकते हैं, हालांकि ".स्क्रीनसाइज़" का उपयोग करने के बजाय आप ".स्पीड" का उपयोग करते हैं। कोष्ठक के अंदर आप गति निर्धारित करते हैं (जल्दी से पैटर्न बनाने के लिए, गति को 75 पर सेट करने का प्रयास करें)।

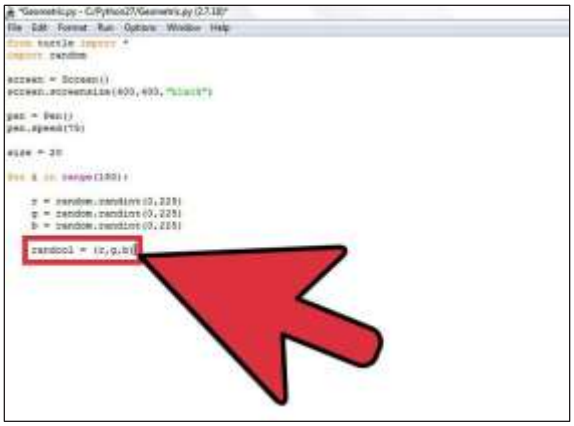


7. इसके बाद में एक वर्ग के आकार के रूप में उपयोग करने के लिए एक वेरिएबल बनाएं: इस प्रोग्राम में आपको जो कूल जियोमेट्रिक पैटर्न मिलता है, वह स्क्रीन में खींचे जा रहे कई वर्गों से निर्मित होता है। आपको "आकार" नामक एक वेरिएबल बनाने की जरूरत है तो इसे 20 के बराबर करना होगा जिसका उपयोग इन वर्गों के आकार को निर्धारित करने के लिए किया जाता है।

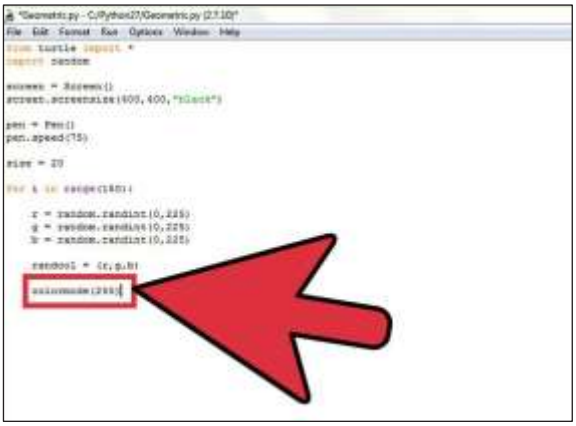


8. लूप के लिए बनाएं: वांछित जियोमेट्रिक पैटर्न प्राप्त करने के लिए आपको पेन बनाने वाले वर्ग रखने की आवश्यकता होती है, आप इसे पुनरावृत्ति के माध्यम से करते हैं जो कि लूप के लिए होती है। यह कोड की अगली पंक्ति पर "`for i in range(150):`" लिखकर किया जाता है। यह जो करता है वह प्रोग्राम को 150 बार चलाने के लिए सेटअप करता है, इस मामले के लिए इसका उपयोग 150 बार वर्गों को खींचने के लिए किया जाएगा, जिसके परिणामस्वरूप एक शांत जियोमेट्रिक पैटर्न होगा। (लूप बनाने के बाद के सभी वेरिएबल णों को लूप के अंदर होना चाहिए। यह केवल टैब कुंजी और इंडेंटिंग द्वारा किया जाता है। हालांकि लूप बनाने के बाद यह आपके लिए स्वचालित रूप से ऐसा करना चाहिए।)

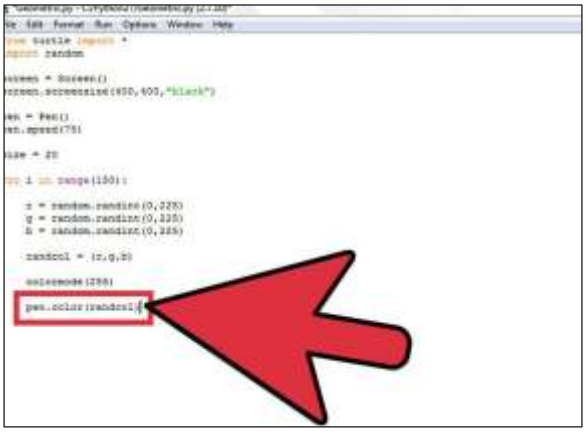
9. एक रैंडम कलर तैयार करें: पैटर्न को बेतरतीब ढंग से उत्पन्न कलर देने के लिए आपको निम्नलिखित कार्य करने होंगे। अगली पंक्ति में "r" नाम का एक वेरिएबल बनाएं और इसे " रैंडम.रैंडिंट (0,225) के बराबर सेट करें। वेरिएबल नामों "जी" और "बी" का उपयोग करके इस वेरिएबल ण को दो बार दोहराएं।



10. रैंडम कलर संग्रहीत करें: अब जब रैंडम संख्याएं उत्पन्न करने वाले तीन वेरिएबल हैं, तो आपको उन्हें एक वेरिएबल में संग्रहीत करने की आवश्यकता है। कोड की अगली पंक्ति में "रैंडकोल" नाम का एक वेरिएबल बनाएं और इसे "(आर,जी,बी)" के बराबर सेट करें।

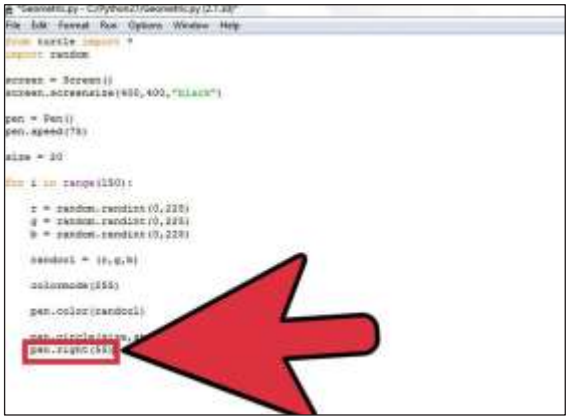


11. प्रोग्राम को कलर का उपयोग करने की अनुमति दें: अपने प्रोग्राम को उन रंगों तक पहुंच प्रदान करने के लिए जिन्हें आपने कोल- या फ़ंक्शन चलाया है। कलर फंक्शन को चलाने के लिए आपको बस कोड की अगली लाइन "कलर-मोड (255)" टाइप करना है और अगली लाइन पर जाना है।

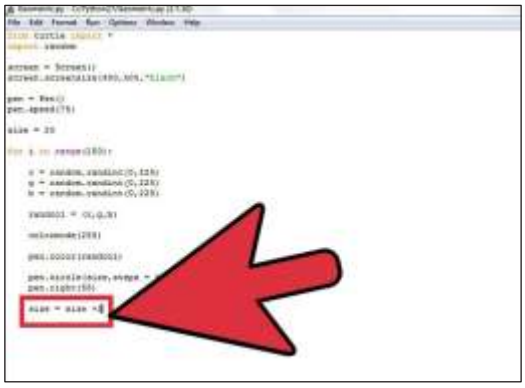


12. कलर सेट करें: पहले बनाए गए पेन का उपयोग करके आप उसका कलर सेट कर देंगे। आप इसे लिखकर "पेन.कलर (रैंडकोल)"। जब यह पैटर्न तैयार करेगा तो यह अब आपके पेन को एक रैंडम कलर कर देगा।

13. पेन को निर्देश दें: वांछित प्रभाव प्राप्त करने के लिए आपको "पेन.सर्कल (आकार, वेरिएबल ण) टाइप करना होगा = 4)"। वेरिएबल ण 7 में आपने एक वेरिएबल "आकार" बनाया है जिसका उपयोग यहां किया गया है। फिर "वेरिएबल ण = 4" भाग वर्ग बनाता है।



14. पेन को घुमाएं: कूल पैटर्न लूप के हर पुनरावृत्ति में पेन को घुमाने से आता है। आप कोड की अगली लाइन "पेन.राइट (55)" पर लिखकर पेन को घुमाते हैं। यह लूप के माध्यम से हर बार पेन को 55 डिग्री पर दाएं घुमाता है।



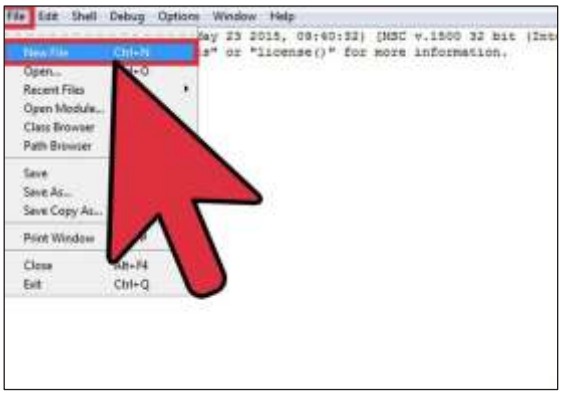
15. वर्ग का आकार बढाएं: शांत पैटर्न का एक हिस्सा यह है कि वर्ग बड़ा होता रहता है। आप कोड की अंतिम पंक्ति "आकार = आकार +3" लिखकर ऐसा कर सकते हैं। ऐसा करने पर हर बार लूप के माध्यम से यह वर्ग के आकार को 3 से बढ़ा देता है।

16. प्रोग्राम को चलाएं: आपका प्रोग्राम ऐसा दिखना चाहिए जैसे वह यहां दिखता है। कूल जियोमेट्रिक पैटर्न देखने के लिए आपको बस अपने कीबोर्ड पर "f5" हिट करना होगा।

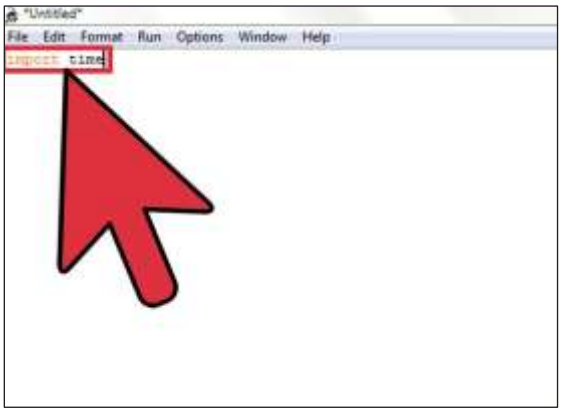
पायथन में एक काउंटडाउन प्रोग्राम कैसे बनाएं

चरण

1. अपना टेक्स्ट एडिटर खोलें।

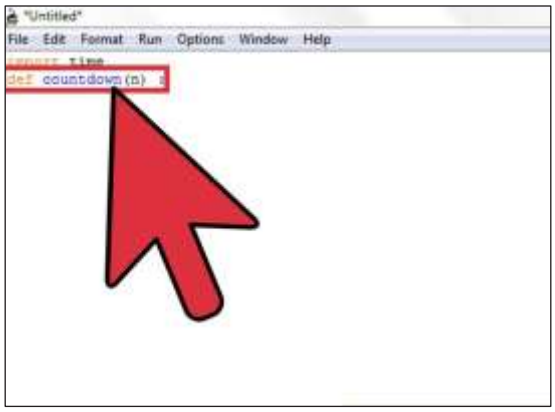


2. फ़ाइल मेनू पर जाएं और New Window पर क्लिक करें या बस Ctrl +N दबाएं।

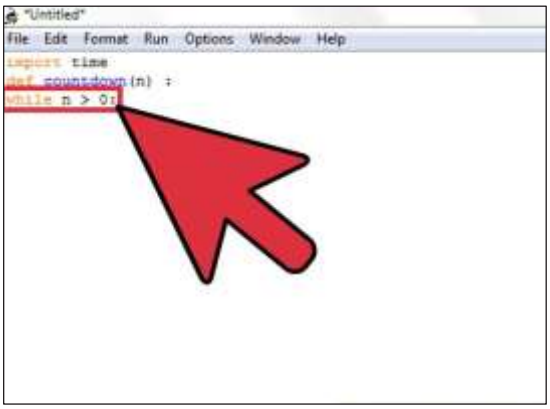


3. 'टाइम' मॉड्यूल आयात करें: ऐसा करने के लिए, "इम्पोर्ट टाइम" टाइप करें। यह टाइम मॉड्यूल आयात करेगा।

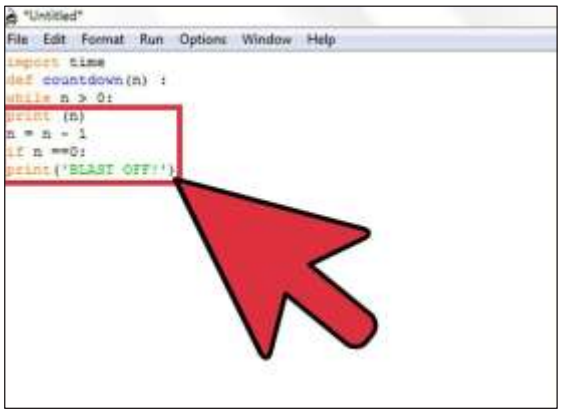
4. काउंटडाउन को परिभाषित करने के लिए 'डीईएफ' फ़ंक्शन का उपयोग करें: फ़ंक्शन को अपनी पसंद का नाम दें। 'उलटी गिनती' का उपयोग करेगा, "`def countdown(t):.`" Remember to leave four spaces after the colon.



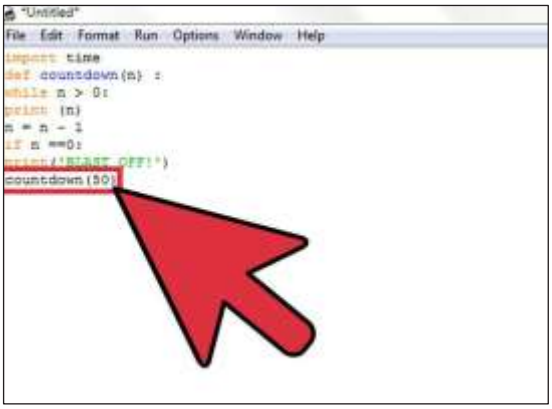
5. अपना काउंटडाउन लूप शुरू करने के लिए एक 'जबकि' फ़ंक्शन लिखें: "`while t > 0:`" कोड में टाइप करें, यह प्रोग्राम का कारण बनेगा, जबकि वर्णमाला को शून्य से अधिक के रूप में परिभाषित किया गया है, इस फ़ंक्शन को निष्पादित करें: "print(t)t = t -1." जैसे-जैसे फ़ंक्शन आगे बढ़ेगा और लूप चक्र पूरा करेगा, संख्या लगातार घटती जाएगी।



6. फिनिशिंग टच जोड़ें: 'ब्लास्ट ऑफ!' प्रिंट आउट करने के लिए निम्नलिखित कोड टाइप करें। जब काउंटडाउन शून्य पर पहुंच जाती है। " `if t == 0:print("BLAST OFF!").`"



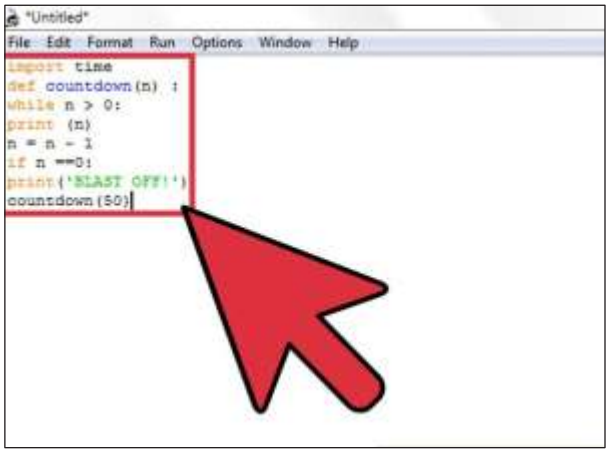
7. इसमें वह नंबर जोड़ें जिससे आप उलटी गिनती शुरू करना चाहते हैं: फ़ंक्शन 'उलटी गिनती' को कॉल करें और कोष्ठक में, अपनी इच्छित संख्या दर्ज करें। उदाहरण के लिए, यदि आप 50 सेकंड की देरी चाहते हैं, तो आपका कोड "`countdown(50).`" होना चाहिए।



8. अपना तैयार कोड जांचें: यह इस तरह दिखना चाहिए-

```
import time

#def countdown(n) :
```

```
#while n > 0:

#print (n)

#n = n - 1

#if n ==0:

#print('BLAST OFF!')

#countdown(50)

#

1.
```



एक साधारण कैलकुलेटर कैसे बनाएं

पायथन प्रोग्रामिंग भाषा संख्याओं के साथ काम करते समय और गणितीय अभिव्यक्तियों का मूल्यांकन करने के लिए उपयोग करने के लिए एक महान उपकरण है। इस गुण का उपयोग उपयोगी प्रोग्राम बनाने में किया जा सकता है।

हम अपना कैलकुलेटर बनाने के लिए गणित ऑपरेटरों, वेरिएबल , सशर्त बयानों, कार्यों का उपयोग करेंगे और उपयोगकर्ता इनपुट को संभालेंगे।

चरण 1: इनपुट के लिए संकेत उपयोगकर्ता

कैलकुलेटर तब सबसे अच्छा काम करते हैं जब एक मानव कंप्यूटर को हल करने के लिए समीकरण प्रदान करता है। हम अपना प्रोग्राम उस बिंदु पर लिखना शुरू करेंगे जहां मानव उन नंबरों में प्रवेश करता है, जिनके साथ वे चाहते हैं कि कंप्यूटर काम करे।

ऐसा करने के लिए, हम पायथन के अंतर्निहित इनपुट () फ़ंक्शन का उपयोग करेंगे जो कीबोर्ड से उपयोगकर्ता द्वारा उत्पन्न इनपुट को स्वीकार करता है। इनपुट () फ़ंक्शन के कोष्ठक के अंदर हम उपयोगकर्ता को संकेत देने के लिए एक स्ट्रिंग पास कर सकते हैं। हम उपयोगकर्ता के इनपुट को एक वेरिएबल में असाइन कर सकते हैं।

इस प्रोग्राम के लिए, हम चाहते हैं कि उपयोगकर्ता दो नंबरों को इनपुट करे, तो चलिए प्रोग्राम को दो नंबरों के लिए संकेत देते हैं। इनपुट मांगते समय, हमें अपनी स्ट्रिंग के अंत में एक स्पेस शामिल करना चाहिए ताकि उपयोगकर्ता के इनपुट और प्रॉम्प्टिंग स्ट्रिंग के बीच एक स्पेस हो।

```
number_1 = input('Enter your first number: ')

number_2 = input('Enter your second number: ')
```

अपनी दो लाइन लिखने के बाद हमें प्रोग्राम को चलाने से पहले उसे सेव कर लेना चाहिए। हम इस प्रोग्राम को Calculator.py कह सकते हैं और एक टर्मिनल विंडो में, हम अपने प्रोग्रामिंग वातावरण में python Calculator.py कमांड का उपयोग करके प्रोग्राम को चला सकते हैं। आपको प्रत्येक संकेत के जवाब में टर्मिनल विंडो में टाइप करने में सक्षम होना चाहिए।

```
Output

Enter your first number: 5

Enter your second number: 7
```

यदि आप इस प्रोग्राम को कई बार चलाते हैं और अपना इनपुट बदलते हैं, तो आप देखेंगे कि आप जो कुछ भी दर्ज कर सकते हैं तुम्हें चाहिएजब संकेत दिया जाए, तो शब्दों, प्रतीकों, रिक्त स्थान, या केवल एंटर कुंजी सहित। ऐसा इसलिए है क्योंकि इनपुट () डेटा को स्ट्रिंग्स के रूप में लेता है और यह नहीं जानता कि हम एक नंबर की तलाश कर रहे हैं।

हम 2 कारणों से इस प्रोग्राम में एक संख्या का उपयोग करना चाहेंगे: 1) प्रोग्राम को गणितीय गणना करने के लिए सक्षम करने के लिए, और 2) यह सत्यापित करने के लिए कि उपयोगकर्ता का इनपुट एक संख्यात्मक स्ट्रिंग होती है।

कैलकुलेटर की हमारी जरूरतों के आधार पर, हम इनपुट () फ़ंक्शन से आने वाली स्ट्रिंग को पूर्णांक या फ्लोट में बदलना चाह सकते हैं। हमारे लिए, पूर्ण संख्याएँ हमारे उद्देश्य के अनुकूल होते हैं, इसलिए हम इनपुट को पूर्णांक डेटा प्रकार में बदलने के लिए int () फ़ंक्शन में इनपुट () फ़ंक्शन को बदलेंगे।

calculator.py

```
number_1 = int(input('Enter your first number: '))

number_2 = int(input('Enter your second number:
```

अब, अगर हम दो पूर्णांक इनपुट करते हैं तो हम एक त्रुटि में नहीं चल सकते हैं:

```
Output

Enter your first number: 23

Enter your second number: 674
```

लेकिन, अगर हम अक्षर, प्रतीक, या कोई अन्य गैर-पूर्णांक दर्ज करते हैं, तो हम निम्नलिखित त्रुटि का सामना करेंगे:

```
Output

Enter your first number: sammy

Traceback (most recent call last):

      File "testing.py", line 1, in <module>

      number_1 = int(input('Enter your first number: '))
```

```
ValueError: invalid literal for int() with base 10: 'sammy'
```

अब तक, हमने उपयोगकर्ता इनपुट को पूर्णांक डेटा प्रकारों के रूप में संग्रहीत करने के लिए दो वेरिएबल सेट किए हैं। आप ऐसा कर सकते हैं इनपुट को फ़्लोट में बदलने के लिए भी प्रयोग करें।

चरण 2: ऑपरेटरों को जोड़ना

हमारा प्रोग्राम पूरा होने से पहले, हम कुल 4 गणितीय ऑपरेटर जोड़ेंगे: + जोड़ने के लिए, - घटाव के लिए, * गुणा के लिए, और / विभाजन के लिए।

जैसा कि हम अपने प्रोग्राम का निर्माण करते हैं, हम यह सुनिश्चित करना चाहते हैं कि प्रत्येक भाग सही ढंग से काम कर रहा है, इसलिए यहां हम जोड़ स्थापित करने के साथ शुरू करेंगे। हम दो नंबरों को एक प्रिंट फ़ंक्शन में जोड़ देंगे ताकि कैलकुलेटर का उपयोग करने वाला व्यक्ति आउटपुट देख सके।

calculater.py

```
number_1 = int(input('Enter your first number: '))

number_2 = int(input('Enter your second number:

')) print(number_1 + number_2)
```

आइए प्रोग्राम चलाते हैं और दो नंबर टाइप करते हैं जब यह सुनिश्चित करने के लिए कहा जाता है कि यह काम कर रहा है जैसा कि हम उम्मीद करते हैं:

```
Output

Enter your first number: 8

Enter your second number: 3

11
```

इसका आउटपुट हमें दिखाता है कि प्रोग्राम सही तरीके से काम कर रहा है, तो चलिए प्रोग्राम के पूरे रनटाइम के दौरान उपयोगकर्ता को पूरी तरह से सूचित करने के लिए कुछ और संदर्भ जोड़ते हैं। ऐसा करने के लिए, हम अपने टेक्स्ट को ठीक से प्रारूपित करने और प्रतिक्रिया प्रदान करने में हमारी सहायता के लिए स्ट्रिंग फॉर्मेटर्स का उपयोग करेंगे। हम चाहते हैं कि उपयोगकर्ता उन नंबरों के बारे में पुष्टि प्राप्त करे जो वे दर्ज कर रहे हैं और ऑपरेटर जो उत्पादित परिणाम के साथ उपयोग किया जा रहा है।

calculater.py

```
number_1 = int(input('Enter your first number: '))

number_2 = int(input('Enter your second number:

')) print('{} + {} = '.format(number_1, number_2))

print(number_1 + number_2)
```

अब, जब हम प्रोग्राम चलाते हैं तो हमारे पास अतिरिक्त आउटपुट होगा जो उपयोगकर्ता को प्रोग्राम द्वारा निष्पादित की जा रही गणितीय अभिव्यक्ति की पुष्टि करने देगा।

```
Output

Enter your first number: 90

Enter your second number: 717

90 + 717 =

807
```

स्ट्रिंग फॉर्मेटर्स का उपयोग करने से उपयोगकर्ताओं को अधिक प्रतिक्रिया मिलती है।

इस बिंदु पर, आप बाकी ऑपरेटरों को उसी प्रारूप के साथ प्रोग्राम में जोड़ सकते हैं जो हमारे पास है जोड़ने के लिए उपयोग किया जाता है:

calculater.py

```
number_1 = int(input('Enter your first number: '))

number_2 = int(input('Enter your second number:

'))

# Addition

print('{} + {} = '.format(number_1, number_2))

print(number_1 + number_2)

# Subtraction

print('{} - {} = '.format(number_1, number_2))

print(number_1 - number_2)

# Multiplication

print('{} * {} = '.format(number_1, number_2))

print(number_1 * number_2)

# Division

print('{} / {} = '.format(number_1, number_2))

print(number_1 / number_2)
```

हमने उपरोक्त कार्यक्रम में शेष ऑपरेटरों, -, *, और / को जोड़ा है। यदि हम इस बिंदु पर प्रोग्राम चलाते हैं, तो प्रोग्राम उपरोक्त सभी कार्यों को निष्पादित करेगा। हालांकि, हम प्रोग्राम को एक समय में केवल एक ऑपरेशन करने के लिए सीमित करना चाहते हैं। ऐसा करने के लिए, हम सशर्त बयानों का उपयोग करेंगे।

चरण 3: सशर्त विवरण को जोड़ना

हमारे कैलकुलेटर.पीई प्रोग्राम के साथ, हम चाहते हैं कि उपयोगकर्ता विभिन्न ऑपरेटरों में से चुनने में सक्षम हो। तो, आइए कार्यक्रम के शीर्ष पर कुछ जानकारी जोड़ने के साथ-साथ एक विकल्प बनाकर शुरू करें, ताकि व्यक्ति को पता चले कि क्या करना है।

हम ट्रिपल कोट्स का उपयोग करके कुछ अलग लाइनों पर एक स्ट्रिंग लिखेंगे:

```
'''
Please type in the math operation you would like to complete:

+ for addition

- for subtraction

* for multiplication

/ for division

'''
```

हम उपयोगकर्ताओं को अपनी पसंद बनाने के लिए प्रत्येक ऑपरेटर प्रतीकों का उपयोग कर रहे हैं, इसलिए यदि उपयोगकर्ता चाहता है कि विभाजन किया जाए, तो वे / टाइप करेंगे। हम जो भी प्रतीक चाहते हैं उसे चुन सकते हैं, हालांकि, जोड़ने के लिए 1 या घटाव के लिए बी का इस्तेमाल उन्हें करना है।

क्योंकि हम उपयोगकर्ताओं से इनपुट मांग रहे हैं, क्योंकि हम इनपुट () फ़ंक्शन का उपयोग करना चाहते हैं। हम स्ट्रिंग को इनपुट () फ़ंक्शन के अंदर रखेंगे और उस इनपुट के मान को एक वेरिएबल में पास करेंगे, जिसे हम ऑपरेशन नाम देंगे।

calculater.py

```
operation = input('''

कृपया वह गणित ऑपरेशन टाइप करें जिसे आप पूरा करना चाहते हैं:

+ for addition

- for subtraction

* for multiplication

/ for division

''')

number_1 = int(input('Enter your first number: '))

number_2 = int(input('Enter your second number:

')) print('{} + {} = '.format(number_1, number_2))

print(number_1 + number_2)

print('{} - {} = '.format(number_1, number_2))

print(number_1 - number_2)

print('{} * {} = '.format(number_1, number_2))

print(number_1 * number_2)
```

```
print('{} / {} = '.format(number_1, number_2))
print(number_1 / number_2)
```

इस पॉइंट पर, यदि हम अपना प्रोग्राम चलाते हैं, तो इससे कोई फ़र्क नहीं पड़ता कि हम पहले प्रॉम्प्ट पर क्या इनपुट करते हैं, तो चलिए प्रोग्राम में अपने कंडीशनल स्टेटमेंट जोड़ते हैं। इस वजह से कि हमने अपने प्रोग्राम को कैसे संरचित किया है, अगर कथन वह होगा जहां जोड़ किया जाता है, तो अन्य ऑपरेटरों में से प्रत्येक के लिए 3 अन्य-अगर या एलिफ कथन होंगे और अन्य कथन को संभालने के लिए रखा जाएगा त्रुटि अगर व्यक्ति ने एक ऑपरेटर प्रतीक इनपुट नहीं किया है।

calculater.py

```
operation = input('''
Please type in the math operation you would like to complete:
+ for addition
- for subtraction
* for multiplication
/ for division
''')
number_1 = int(input('Enter your first number: '))
number_2 = int(input('Enter your second number:
')) if operation == '+':
      print('{} + {} = '.format(number_1, number_2))
      print(number_1 + number_2)
elif operation == '-':
      print('{} - {} = '.format(number_1, number_2))
      print(number_1 - number_2)
elif operation == '*':
      print('{} * {} = '.format(number_1, number_2))
      print(number_1 * number_2)
elif operation == '/':
      print('{} / {} = '.format(number_1, number_2))
      print(number_1 / number_2)
```

```
else:

      print('You have not typed a valid operator, please run the program
      again.')
```

इस प्रोग्राम के माध्यम से चलने के लिए, पहले यह उपयोगकर्ता को एक ऑपरेशन प्रतीक डालने के लिए प्रेरित करता है। हम कहेंगे कि उपयोगकर्ता इनपुट * गुणा करने के लिए। इसके बाद, प्रोग्राम 2 नंबर मांगता है, और उपयोगकर्ता 58 और 40 इनपुट करता है। इस बिंदु पर, प्रोग्राम प्रदर्शन किए गए समीकरण और उत्पाद को दिखाता है।

```
Output

Please type in the math operation you would like to complete:

      + for addition

      - for subtraction

      * for multiplication

      / for division

      *

Please enter the first number: 58

Please enter the second number: 40

58 * 40 =

2320
```

इस वजह से कि हम प्रोग्राम की संरचना कैसे करते हैं, यदि उपयोगकर्ता पहली बार किसी ऑपरेशन के लिए पूछे जाने पर% दर्ज करता है, तो उन्हें नंबर दर्ज करने के बाद तक फिर से प्रयास करने के लिए फीडबैक प्राप्त नहीं होगा। आप विभिन्न स्थितियों से निपटने के लिए अन्य संभावित विकल्पों पर विचार करना चाह सकते हैं।

इस पॉइंट पर हमारे पास पूरी तरह कार्यात्मक प्रोग्राम है, लेकिन हम प्रोग्राम को फिर से चलाए बिना दूसरा या तीसरा ऑपरेशन नहीं कर सकते हैं, तो चलिए प्रोग्राम में कुछ और कार्यक्षमता जोड़ते हैं।

चरण 4: कार्यों को परिभाषित करें

उपयोगकर्ता जितनी बार चाहें प्रोग्राम को निष्पादित करने की क्षमता को संभालने के लिए, हम कुछ कार्यों को परिभाषित करते हैं। आइए पहले हमारे मौजूदा कोड ब्लॉक को एक फंक्शन में डालें। हम फ़ंक्शन का नाम कैलकुलेट () रखेंगे और फ़ंक्शन के भीतर ही इंडेंटेशन की एक अतिरिक्त परत जोड़ेंगे। यह सुनिश्चित करने के लिए कि प्रोग्राम चलता है, हम अपनी फ़ाइल के निचले भाग में फ़ंक्शन को भी कॉल करेंगे।

calculater.py

```
# Define our function

def calculate():

      operation = input('''
```

```
Please type in the math operation you would like to complete:
      + for addition
      - for subtraction
      * for multiplication
      / for division
''')
      number_1 = int(input('Please enter the first number: '))
      number_2 = int(input('Please enter the second number:
      '))
if operation == '+':
      print('{} + {} = '.format(number_1, number_2))
      print(number_1 + number_2)
elif operation == '-':
      print('{} - {} = '.format(number_1, number_2))
      print(number_1 - number_2)
elif operation == '*':
      print('{} * {} = '.format(number_1, number_2))
      print(number_1 * number_2)
elif operation == '/':
      print('{} / {} = '.format(number_1, number_2))
      print(number_1 / number_2)
else:
      print('You have not typed a valid operator, please run the program
      again.')
# Call calculate() outside of the function
calculate()
```

इसके बाद हम बनाते हैं एक दूसरा फ़ंक्शन जो अधिक स्टेटमेंट से बना है। कोड के इस ब्लॉक में हम उपयोगकर्ता को यह विकल्प देना चाहते हैं कि वे फिर से गणना करना चाहते हैं या नहीं। हम अपने कैलकुलेटर सशर्त स्टेटमेंट के आधार पर इसे आधार बना सकते हैं, लेकिन इस मामले में हमारे पास त्रुटियों को संभालने के लिए केवल एक विकल्प होगा पहला एलिफ और दूसरा भी यही होगा।

हम इस फ़ंक्शन को फिर से नाम देंगे () और इसे हमारे डीफ़ कैलकुलेट (): कोड ब्लॉक के नीचे जोड़ देंगे।

calculater.py

```
...

# Define again() function to ask user if they want to use the calculator
again

def again():

      # Take input from user

      calc_again = input(‘’’

Do you want to calculate again?

Please type Y for YES or N for NO.

’’’)

      # If user types Y, run the calculate()

      function if calc_again == ‘Y’:

            calculate()

      # If user types N, say good-bye to the user and end the

program elif calc_again == ‘N’:

      print(‘See you later.’)

      # If user types another key, run the function again

else:

      again()

# Call calculate() outside of the function

calculate()
```

यद्यपि ऊपर दिए गए अन्य कथन के साथ कुछ त्रुटि-प्रबंधन है, हम शायद ऊपरी-केस Y और N के अलावा एक लोअर-केस y और n को स्वीकार करने के लिए थोड़ा बेहतर कर सकते हैं। ऐसा करने के लिए, आइए जोड़ते हैं स्ट्रिंग फ़ंक्शन `str.upper ()`:

calculater.py

```
...

def again():

      calc_again = input(‘’’
```

```
Do you want to calculate again?

Please type Y for YES or N for

NO.''')

     # Accept 'y' or 'Y' by adding str.upper()

if calc_again.upper() == 'Y':

     calculate()

     # Accept 'n' or 'N' by adding str.upper()

elif calc_again.upper() == 'N':

     print('See you later.')

else:

     again()

...
```

इस पॉइंट पर, हमें कैलकुलेट () फ़ंक्शन के अंत में फिर से () फ़ंक्शन जोड़ना चाहिए ताकि हम उस कोड को ट्रिगर कर सकें, जो उपयोगकर्ता से पूछता है कि वे जारी रखना चाहते हैं या नहीं।

calculater.py

```
def calculate():

     operation = input('''

Please type in the math operation you would like to complete:

     + for addition

     - for subtraction

     * for multiplication

     / for division

''')

     number_1 = int(input('Please enter the first number: '))

     number_2 = int(input('Please enter the second number:

     '))

if operation == '+':

     print('{} + {} = '.format(number_1, number_2))

     print(number_1 + number_2)
```

```
elif operation == '-':
    print('{} - {} = '.format(number_1, number_2))
    print(number_1 - number_2)
elif operation == '*':
    print('{} * {} = '.format(number_1, number_2))
    print(number_1 * number_2)
elif operation == '/':
    print('{} / {} = '.format(number_1, number_2))
    print(number_1 / number_2)
else:
    print('You have not typed a valid operator, please run the program
    again.')
    # Add again() function to calculate() function
    again()
def again():
    calc_again = input('''
Do you want to calculate again?
Please type Y for YES or N for NO.
''')
if calc_again.upper() == 'Y':
    calculate()
elif calc_again.upper() == 'N':
    print('See you later.')
else:
    again()
calculate()
```

अब आप अपने प्रोग्राम को अपनी टर्मिनल विंडो में अजगर कैलकुलेटर के साथ चला सकते हैं और आप जितनी बार चाहें गणना करने में सक्षम होंगे।

चरण 5: कोड में सुधार करें

अब हमारे पास एक अच्छा, पूरी तरह कार्यात्मक प्रोग्राम मौजूद है। हालांकि, इस कोड को बेहतर बनाने के लिए आप और भी बहुत कुछ कर सकते हैं। आप एक वेलकम फंक्शन को जोड़ सकते हैं, उदाहरण के लिए जो प्रोग्राम के कोड के शीर्ष पर प्रोग्राम में लोगों का स्वागत करता है, जैसे:

```
def welcome():
      print('''
Welcome to Calculator
''')
...
# Don't forget to call the function
welcome()
calculate()
```

पूरे प्रोग्राम में अधिक त्रुटि-प्रबंधन शुरू करने के अवसर होते हैं। शुरुआत के लिए,आप यह सुनिश्चित कर सकते हैं कि प्रोग्राम चलता रहे, भले ही उपयोगकर्ता नंबर मांगे जाने पर प्लवक टाइप करता हो। जैसा कि प्रोग्राम अभी है, यदि संख्या_1 और संख्या_2 पूर्णांक नहीं होते हैं, तो उपयोगकर्ता को एक त्रुटि मिलेगी और प्रोग्राम चलना बंद हो जाता है। साथ ही, ऐसे मामलों में जब उपयोगकर्ता डिवीज़न ऑपरेटर (/) का चयन करता है और अपने दूसरे नंबर (नंबर_2) के लिए 0 टाइप करता है, तो उपयोगकर्ता को ज़ीरोडि-विज़न एरर: डिवीज़न बाय ज़ीरो एरर प्राप्त होगा। इसके लिए, आप कथन को छोड़कर प्रयास के साथ अपवाद हैंडलिंग का उपयोग करना चाह सकते हैं।

हमने खुद को 4 ऑपरेटरों तक सीमित कर दिया है, लेकिन आप अतिरिक्त ऑपरेटरों को जोड़ सकते हैं, जैसे:

```
...
      operation = input('''

Please type in the math operation you would like to complete:
+ for addition
- for subtraction
* for multiplication
/ for division
** for power
% for modulo
''')
...
```

...

```
# Don’t forget to add more conditional statements to solve for power
and modulo
```

इसके अतिरिक्त, आप लूप स्टेटमेंट के साथ प्रोग्राम के हिस्से को फिर से लिख सकते हैं।

त्रुटियों को संभालने और प्रत्येक कोडिंग प्रोजेक्ट को संशोधित करने और सुधारने के कई तरीके होते हैं। यह ध्यान रखना महत्वपूर्ण होता है कि किसी समस्या को हल करने का कोई एक सही तरीका नहीं है जिसे हम प्रस्तुत करते हैं।