**CURSO CISCO DE PYTHON**



***INICIO DIA 25/04/2023***

***#MaratónLATAM2023\_Python\_Essentials\_1\_IFSULDEMINAS***

**CISCO SKILLSFORALL** -> <https://skillsforall.com/pt/course/python-essentials-1?courseLang=pt-BR&instance_id=25c80f00-9e98-4eaa-a278-e7a8e7739403>

**CLASSROOM ->** <https://classroom.google.com/c/NTUyNzkxOTI2Njcz>

**COLAB PYTHON->** <https://colab.research.google.com/>

**MATERIA ESTRA(ALUNO):** <http://excript.com/python/atribuicao-multipla-python.html>

**FUNÇÕES:** <https://programadoresdepre.com.br/20-funcoes-python-que-voce-precisa-conhecer/>

**Fundamentos do Python 1**

Este curso faz parte do Coleções de Aprendizagem - Python

Aprenda conceitos fundamentais de programação de computadores e comece a desenvolver habilidades de codificação com a linguagem de programação Python

Usado por startups e gigantes da tecnologia como Google, Facebook, Netflix e muito mais, o Python oferece infinitas possibilidades para criar projetos de software de pequena e grande escala. Fácil de usar com código fácil de ler, o Python é uma ótima primeira linguagem de programação para aprender e não requer conhecimento prévio de programação. As habilidades em Python abrem você para carreiras em quase todos os setores e são necessárias se você quiser continuar em funções de desenvolvimento e engenharia de software mais avançadas e com salários mais altos, como engenheiro de software, administrador de sistemas e engenheiro de segurança.

Neste curso, você aprenderá habilidades sob demanda, como projetar, desenvolver e melhorar programas de computador, métodos para analisar problemas usando programação, melhores práticas de programação e muito mais. O curso também prepara você para a certificação PCEP – Certified Entry-Level Python Programmer (Exame PCEP-30-0x).

Aqui está o que você aprenderá:

* PE1: Módulo 1. Introdução à programação de computadores e Python
* PE1: Módulo 2. Tipos de dados em Python, variáveis, operadores e operações de I/O básicas
* PE1: Módulo 3. Valores Booleanos, Execução Condicional, Loops, Processamento de Listas e Listas, Operações Lógicas e bit a bit
* PE1: Módulo 4. Funções, Tuplas, Dicionários, Exceções e Processamento de Dados
* Fundamentos do Python 1 (PE1) Exame final do curso

Índice do Curso

**PE1: Módulo 1. Introdução à programação de computadores e Python**

1.0. Bem-vindo ao Fundamentos do Python 1

1.1. Seção 1 - Introdução à Programação

1.2. Seção 2 - Introdução a Python

1.3. Seção 3 - Download e Instalação do Python

1.4. Conclusão do Módulo 1 - Teste de Módulo

**PE1: Módulo 2. Tipos de dados em Python, variáveis, operadores e operações de I/O básicas**

2.1. Seção 1 - O Programa "Olá, mundo!"

2.2. Seção 2 - Literais Python

2.3. Seção 3 - Operadores - ferramentas de manipulação de dados

2.4. Seção 4 - Variáveis

2.5. Seção 5 - Comentários

2.6. Seção 6 - Interação com o usuário

2.7. Conclusão do Módulo 2 - TESTE DO MÓDULO

**PE1: Módulo 3. Valores Booleanos, Execução Condicional, Loops, Processamento de Listas e Listas, Operações Lógicas e bit a bit**

3.1. Seção 1 - Tomada de decisões em Python

3.2. Seção 2 - Loops no Python

3.3. Seção 3 - Operações lógicas e com bits no Python

3.4. Seção 4 - Listas

3.5. Seção 5 - Ordenando listas simples: o algoritmo de ordenação por bolhas

3.6. Seção 6 - Operações em listas

3.7. Seção 7 - Listas em aplicações avançadas

3.8. Conclusão do módulo 3 - Teste de módulo

**PE1: Módulo 4. Funções, Tuplas, Dicionários, Exceções e Processamento de Dados**

4.1. Seção 1 – Funções

4.2. Seção 2 – Como as funções se comunicam com seu ambiente

4.3. Seção 3 – Retornando o resultado de uma função

4.4. Seção 4 – Escopo em Python

4.5. Seção 5 – Criando funções multi paramétricas

4.6. Seção 6 – Tuplas e dicionários

4.7. Seção 7 – Exceções

4.8. Conclusão do Módulo 4 - Teste de Módulo



**Fundamentos do Python 1 (PE1) Exame final do curso**

* Projeto Final
* Teste Final
* Pesquisa de Final de Curso

# 1.0 Bem-vindo ao Fundamentos do Python 1

**1.0.1 Aprenda Python - o idioma de hoje e amanhã**

Este curso é o primeiro de uma série de dois cursos do Fundamentos do Python. Ele abrange tudo o que você precisa saber para começar a projetar, escrever, executar, depurar e melhorar os programas Python no nível básico. Ele também prepara você para o exame de certificação PCEP - Certified Entry-Level Python Programmer do Python Institute.

**1.0.2 Sobre o curso**

Bem-vindo ao Fundamentos do Python 1! Este curso foi projetado e desenvolvido pelo OpenEDG Python Institute em parceria com a Cisco Networking Academy.

O curso foi criado para qualquer pessoa que queira aprender Python e técnicas de programação modernas. Será particularmente atraente para:

* aspirantes a programadores e alunos interessados em aprender programação para diversão e tarefas relacionadas ao trabalho;
* alunos que desejam adquirir habilidades e conhecimentos fundamentais para uma função de nível básico como desenvolvedor de software, analista de dados ou testador;
* os profissionais do setor que desejam explorar tecnologias conectadas ao Python ou que o utilizam como base;
* líderes de equipe, gerentes de produto e gerentes de projeto que desejam entender a terminologia e os processos no ciclo de desenvolvimento de software para gerenciar e se comunicar com mais eficiência com as equipes de produção e desenvolvimento.
* Os alunos têm acesso a materiais práticos, questionários e avaliações para aprender a utilizar as habilidades e os conhecimentos adquiridos no curso e interagir com tarefas e situações de programação da vida real.

**1.0.3 Programas**

Neste curso, você aprenderá:

* os conceitos universais de programação de computadores;
* a sintaxe e a semântica da linguagem Python;
* habilidades práticas para resolver desafios típicos de implementação;
* como usar os elementos mais importantes da biblioteca padrão do Python;
* como instalar o ambiente de tempo de execução;
* como projetar, desenvolver, testar e depurar programas Python simples.

O curso está dividido em quatro módulos:

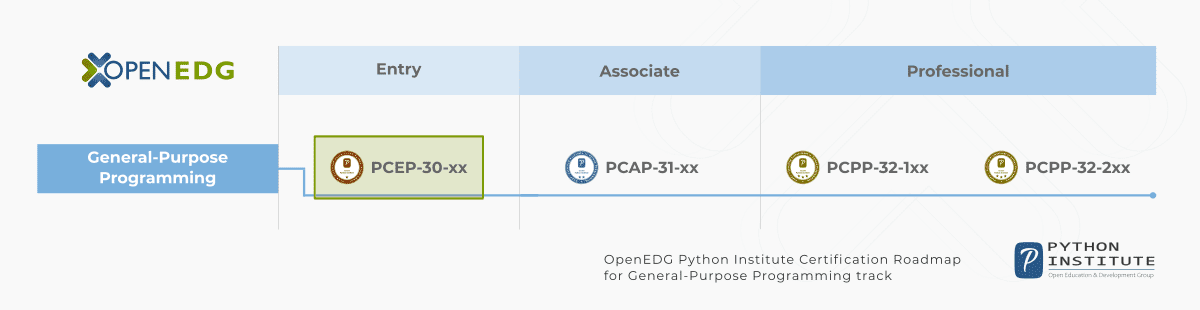
1. **Módulo 1 Introdução à programação de computadores e Python;**
2. **Módulo 2 Tipos de dados, variáveis, operações básicas de entrada e saída, operadores básicos;**
3. **Módulo 3 Valores Booleanos, Execução Condicional, Loops, Processamento de Listas e Listas, Operações Lógicas e bit a bit;**
4. **Módulo 4 Funções, Tuplas, Dicionários, Exceções e Processamento de Dados.**

**1.0.4 Prepare-se para o exame PCEP-30-0X**

Mergulhe na programação, aprenda Python do zero e prepare-se para a certificação PCEP - Certified Entry-Level Python Programmer

O Fundamentos do Python 1 está alinhado à certificação PCEP - Certified Entry-Level Python Programmer, uma credencial profissional que demonstra a compreensão do portador da sintaxe e da semântica da linguagem Python, bem como sua proficiência no uso dos elementos mais essenciais da linguagem, das ferramentas, e recursos para projetar, desenvolver e refatorar programas Python simples.

O detentor da certificação conhece a sintaxe da linguagem Python em um nível que lhes permite trabalhar com variáveis, operadores, mecanismos de fluxo de controle e funções, bem como entende os fundamentos do sistema de tipos de dados Python, tratamento de exceções, solução de problemas, depuração, e o ambiente de tempo de execução.



A certificação PCEP - Certified Python Programmer é uma etapa intermediária da certificação PCAP - Certified Associate in Python Programming e o ponto de partida para iniciar uma carreira em desenvolvimento de software, programação Python e tecnologias relacionadas.

Obter a certificação PCEP ajudará você a se destacar em relação a outros candidatos e a entrar na empresa.

Para obter mais informações sobre a certificação PCEP - Certified Entry-Level Python Programmer, acesse www.pythoninstitute.org <https://pythoninstitute.org/certification/pcep-certification-entry-level/>

***Então, você está pronto para começar sua jornada Python? Clique em Avançar para prosseguir até a Seção 1 e começar a aprender. Nos vemos lá!***

**1.1 Seção 1 - Introdução à programação**

Olá, bem-vindos à seção um! Começaremos aprendendo sobre alguns dos conceitos universais de programação, como lista de instruções, arquivo de origem, elementos de linguagem, compilação e interpretação. Vamos lá? Vamos começar!

**1.1.1 Como funciona um programa de computador?**

Um programa torna um computador utilizável. Sem um programa, um computador, mesmo o mais poderoso, não passa de um objeto. Da mesma forma, sem um jogador, um piano nada mais é do que uma caixa de madeira.

Os computadores são capazes de executar tarefas muito complexas, mas essa capacidade não é inata. A natureza de um computador é bem diferente.

Ele pode executar apenas operações extremamente simples. Por exemplo, um computador não consegue entender sozinho o valor de uma função matemática complicada, embora isso não esteja além dos limites das possibilidades no futuro próximo.

Os computadores contemporâneos só podem avaliar os resultados de operações muito fundamentais, como adicionar ou dividir, mas podem fazer isso com muita rapidez e podem repetir essas ações praticamente várias vezes.

Imagine que você quer saber a velocidade média que atingiu durante uma longa jornada. Você sabe a distância, você sabe o tempo, você precisa da velocidade.

Naturalmente, o computador poderá calcular isso, mas não está ciente de coisas como distância, velocidade ou tempo. Portanto, é necessário instruir o computador para:

* aceitar um número que represente a distância;
* aceitar um número que represente o tempo de viagem;
* divida o valor anterior pelo último e armazene o resultado na memória;
* exibem o resultado (representando a velocidade média) em um formato legível.

Essas quatro ações simples formam um programa. Obviamente, esses exemplos não são formalizados e estão muito longe do que o computador pode entender, mas são bons o suficiente para serem traduzidos para um idioma que o computador possa aceitar.

Linguagem é a palavra-chave.

**1.1.2 Linguagens Naturais vs Linguagens de Programação**

Uma **linguagem** é um meio (e uma ferramenta) para expressar e registrar pensamentos. Há muitos idiomas ao nosso redor. Alguns deles não exigem fala nem escrita, como a linguagem corporal; é possível expressar seus sentimentos mais profundos sem dizer uma palavra.

Outra linguagem que você usa todos os dias é a sua língua nativa, que você usa para manifestar sua vontade e refletir sobre a realidade. Os computadores também têm sua própria linguagem, chamada de linguagem de máquina, o que é muito rudimentar.

Um computador, mesmo o mais tecnicamente sofisticado, não tem nem um traço de inteligência. Você pode dizer que é como um cão bem treinado - responde apenas a um conjunto predeterminado de comandos conhecidos.

Os comandos que ele reconhece são muito simples. Podemos imaginar que o computador responde a pedidos como "pegue esse número, divida por outro e salve o resultado".

Um conjunto completo de comandos conhecidos é chamado de **lista de instruções**, às vezes abreviada para **IL**. Diferentes tipos de computadores podem variar de acordo com o tamanho das ILs, e as instruções podem ser completamente diferentes em modelos diferentes.

Observação: linguagens de máquina são desenvolvidas por seres humanos.

Nenhum computador é capaz de criar um novo idioma no momento. No entanto, isso pode mudar em breve. Assim como as pessoas usam vários idiomas diferentes, as máquinas também têm muitos idiomas diferentes. A diferença, no entanto, é que as linguagens humanas se desenvolveram naturalmente.

Além disso, eles ainda estão evoluindo, e novas palavras são criadas todos os dias à medida que palavras antigas desaparecem. Esses idiomas são chamados de linguagens naturais.

**1.1.3 O que faz uma linguagem?**

Lista de seções expansíveis. Selecione cada botão para expandir o conteúdo.

Podemos dizer que cada linguagem (máquina ou natural, não importa) consiste nos seguintes elementos:

* **um alfabeto**

um conjunto de símbolos usado para criar palavras de um determinado idioma (por exemplo, o alfabeto latino para inglês, o alfabeto cirílico para russo, o kanji para japonês e assim por diante)

* **uma lexis**

(também conhecido como dicionário), um conjunto de palavras que o idioma oferece aos usuários (por exemplo, a palavra "computador" vem do dicionário de português, enquanto "cmoptrue" não; o termo "chat" está presente nos dicionários de inglês e francês , mas com significados diferentes)

* **uma sintaxe**

um conjunto de regras (formais ou informais, escritas ou intuitivamente) usadas para determinar se uma determinada sequência de palavras forma uma sentença válida (por exemplo, "eu sou uma píton" é uma frase sintaticamente correta, enquanto "eu uma píton estou" não é)

* **semântica**

um conjunto de regras que determina se uma determinada frase faz sentido (por exemplo, "comi uma rosquinha" faz sentido, mas "uma rosquinha me comeu" não faz)

**1.1.4 Linguagem da máquina vs. linguagem de alto nível**

A **IL** é, na verdade, o ***alfabeto de uma linguagem de máquina***. Este é o conjunto de símbolos mais simples e primário que podemos usar para dar comandos a um computador. É a língua nativa do computador.

Infelizmente, essa língua nativa está muito longe de ser a língua nativa humana. Nós dois (computadores e seres humanos) precisamos de algo mais, uma linguagem comum para computadores e seres humanos, ou uma ponte entre os dois mundos diferentes.

Precisamos de uma linguagem na qual os humanos possam escrever seus programas e de uma linguagem que os computadores possam usar para executar os programas, uma linguagem que é muito mais complexa do que a linguagem de máquina e, no entanto, muito mais simples do que a linguagem natural.

Essas linguagens são frequentemente chamadas de linguagens de programação de alto nível. Eles são pelo menos um pouco semelhantes aos naturais, pois usam símbolos, palavras e convenções legíveis para os seres humanos. Essas linguagens permitem que seres humanos expressem comandos para computadores que são muito mais complexos do que aqueles oferecidos por ILs.

Um programa escrito em uma linguagem de programação de alto nível é chamado de código-fonte (em contraste com o código de máquina executado por computadores). Da mesma forma, o arquivo que contém o código-fonte é chamado de arquivo-fonte.

**1.1.5 Compilação vs. interpretação**

A programação de computadores é o ato de compor os elementos da linguagem de programação selecionada na ordem que causará o efeito desejado. O efeito pode ser diferente em cada caso específico.

- Depende da imaginação, do conhecimento e da experiência do programador.

Obviamente, essa composição precisa ser correta em muitos sentidos:

* **em ordem alfabética** - um programa precisa ser escrito em um script reconhecível, como romano, cirílico etc.
* **lexicamente -** cada linguagem de programação tem seu dicionário e você precisa dominá-lo; felizmente, é muito mais simples e menor do que o dicionário de qualquer linguagem natural;
* **sintaticamente** - cada idioma tem suas regras e elas devem ser obedecidas;
* **semanticamente** - o programa tem que fazer sentido.

Infelizmente, um programador também pode cometer erros com cada um dos quatro sentidos acima. Cada um deles pode tornar o programa completamente inútil.

Vamos supor que você tenha escrito um programa com sucesso. Como convencer o computador a executá-lo? Você precisa tornar o programa em linguagem de máquina. Felizmente, a tradução pode ser feita por um computador, tornando todo o processo rápido e eficiente.

**Há duas maneiras diferentes de transformar um programa de uma linguagem de programação de alto nível em linguagem de máquina:**

Clique nas imagens para saber mais sobre as diferenças entre compilação e interpretação.



**Compilação -** o programa de origem é convertido uma vez (no entanto, esse ato deve ser repetido cada vez que você modifica o código-fonte) ao obter um arquivo (por exemplo, um arquivo an .exe se o código for executado no MS Windows) contendo a máquina código. Agora você pode distribuir o arquivo em todo o mundo; o programa que executa essa conversão é chamado de compilador ou traduzir.



**Interpretação -** você (ou qualquer usuário do código) pode traduzir o programa de origem toda vez que ele tiver que ser executado. O programa que executa esse tipo de transformação é chamado de interpretador, pois interpreta o código toda vez que se destina a ser executado. Isso também significa que você não pode apenas distribuir o código-fonte como está, porque o usuário final também precisa que o intérprete o execute.

Devido a algumas razões fundamentais, uma determinada linguagem de programação de alto nível foi projetada para se enquadrar em uma dessas duas categorias.

Existem muito poucas linguagens que podem ser compiladas e interpretadas. Normalmente, uma linguagem de programação é projetada com esse fator na mente de seus construtores - será compilada ou interpretada?

**1.1.6 O que o interpretador faz?**

Vamos supor mais uma vez que você escreveu um programa. Agora, ele existe como um arquivo de computador: um programa de computador é na verdade um pedaço de texto, então o código-fonte geralmente é colocado em arquivos de texto.

Observação: ele precisa ser texto puro, sem nenhuma decoração, como fontes diferentes, cores, imagens incorporadas ou outras mídias. Agora você precisa chamar o intérprete e permitir que ele leia o arquivo de origem.

**O intérprete lê o código-fonte de uma forma comum na cultura ocidental:** da parte superior para a inferior e da esquerda para a direita. Há algumas exceções - elas serão abordadas mais adiante neste curso.

* Primeiro de tudo, o intérprete verifica se todas as linhas subsequentes estão corretas (usando os quatro aspectos abordados anteriormente).
* Se o compilador encontrar um erro, ele terminará o trabalho imediatamente. O único resultado nesse caso é uma mensagem de erro.
* O intérprete informará onde o erro está localizado e o que o causou. No entanto, essas mensagens podem ser enganosas, pois o intérprete não é capaz de seguir suas intenções exatas e pode detectar erros a alguma distância de suas causas reais.

Por exemplo, se você tentar usar uma entidade de nome desconhecido, ela causará um erro, mas o erro será descoberto no local em que tenta usar a entidade, e não onde o nome da nova entidade foi introduzido.

Em outras palavras, o motivo real geralmente está localizado um pouco mais cedo no código, por exemplo, no lugar onde você tinha que informar o intérprete que você usaria a entidade do nome.

* Se a linha parece boa, o intérprete tenta executá-la (observação: cada linha é geralmente executada separadamente, para que o "leitura-verificação-execute" do grupo possa ser repetido muitas vezes - mais vezes do que o número real de linhas no arquivo de origem , pois algumas partes do código podem ser executadas mais de uma vez).

Também é possível que uma parte significativa do código seja executada com sucesso antes que o intérprete encontre um erro. Esse é o comportamento normal neste modelo de execução.

Você pode perguntar agora: qual é o melhor? O modelo de "compilação" ou o modelo de "interpretação"? Não há uma resposta óbvia. Se houvesse, um desses modelos teria deixado de existir há muito tempo. Ambos têm suas vantagens e desvantagens.

**1.1.7 Compilação vs. Interpretação - Vantagens e Desvantagens**

|  |  |  |
| --- | --- | --- |
|  | **COMPILAÇÃO** | **INTERPRETAÇÃO** |
| **Vantagens** | * **✓** a execução do código convertido é mais rápida; * **✓** apenas o usuário precisa ter o compilador - o usuário final pode usar o código sem ele; * **✓** o código traduzido é armazenado usando linguagem de máquina - como é muito difícil de entender, suas próprias invenções e truques de programação provavelmente permanecerão em segredo. | * **✓** você pode executar o código assim que tiver concluído - não há fases adicionais de conversão; * **✓** o código é armazenado usando linguagem de programação, e não linguagem de máquina - isso significa que pode ser executado em computadores que usam linguagens de máquina diferentes; você não compila seu código separadamente para cada arquitetura. |
| **Desvantagens** | * **❌** a compilação em si pode ser um processo muito demorado - talvez você não consiga executar o código imediatamente após fazer uma alteração; * **❌** você precisa ter tantos compiladores quanto plataformas de hardware nas quais deseja que seu código seja executado. | * **❌** não espere que a interpretação aumente seu código a alta velocidade - seu código vai compartilhar a capacidade do computador com o intérprete, então não pode ser muito rápido; * **❌** Você e o usuário final precisam ter o intérprete para executar o código. |

O que tudo isso significa para você?

* **Python** - como você deve ter certeza - é uma **linguagem interpretada**. Isso significa que ele herda todas as vantagens e desvantagens descritas. Obviamente, ele adiciona alguns de seus recursos exclusivos aos dois conjuntos.
* Se você quiser programar em Python, precisará do **interpretador Python.** Você não poderá executar seu código sem ele. Felizmente, o Python é gratuito. Essa é uma das vantagens mais importantes.

Devido a razões históricas, as linguagens projetadas para serem utilizadas na maneira de **interpretação** são frequentemente chamadas de ***linguagens de script,*** enquanto os programas de origem **codificados** usando-as são chamados de ***scripts***. Ok, vamos conhecer Python.

**1.2 Seção 2 - Introdução a Python**

**1.0.1 Aprenda Python - o idioma de hoje e amanhã**

Bem-vindo à seção dois. Aqui, aprenderemos um pouco sobre a história do Python, as diferentes versões e implementações do Python e o impacto que o Python teve na programação moderna. Vamos iniciar.

**1.2.1 Python - uma ferramenta, não um réptil**

**O que é Python?**

Python é uma linguagem de programação amplamente usada, interpretada, orientada a objeto e de alto nível com semântica dinâmica, usada para programação de uso geral.

E embora você possa conhecer a píton como uma cobra grande, o nome da linguagem de programação Python vem de uma antiga série de comédia da BBC chamada Monty Python's Flying Circus.

No auge do sucesso, a equipe de Monty Python estava realizando seus esboços para viver audiências em todo o mundo, inclusive no Hollywood Bowl.

Como Monty Python é considerado um dos dois principais elementos essenciais para um programador (o outro é pizza), o criador do Python nomeou a linguagem em homenagem ao programa de TV.

**1.2.2 Quem criou Python?**

Um dos recursos surpreendentes do Python é o fato de que ele é realmente o trabalho de uma pessoa. Normalmente, novas linguagens de programação são desenvolvidas e publicadas por grandes empresas que empregam muitos profissionais e, devido às regras de direitos autorais, é muito difícil nomear qualquer uma das pessoas envolvidas no projeto. Python é uma exceção.

Não há muitos idiomas cujos autores sejam conhecidos pelo nome. Python foi ***criada por Guido van Rossum, nascida em 1956 em Haarlem, na Holanda.*** Claro, Guido van Rossum não desenvolveu e evoluiu todos os componentes do Python.



A velocidade com que o Python se espalhou pelo mundo é resultado do trabalho contínuo de milhares (muitas vezes anônimos) programadores, testadores, usuários (muitos deles não são especialistas em TI) e entusiastas, mas é preciso dizer que os a primeira idéia (a semente da qual Python brotou) veio à tona - o de Guido.

**1.2.3 Um projeto de programação de hobby**

As circunstâncias em que o Python foi criado são um pouco intrigantes. De acordo com Guido van Rossum:

Em dezembro de 1989, eu estava procurando por um projeto de programação "hobby" que me mantivesse ocupado durante a semana por volta do Natal. Meu escritório (...) ficava fechado, mas eu tinha um computador em casa e não muito mais. Decidi escrever um intérprete para a nova linguagem de script em que tenho pensado ultimamente: um descendente da ABC que agradaria a hackers Unix/C. Eu escolhi o Python como um título de trabalho para o projeto, por ser um pouco irreverente (e um grande fã de Circo Voador de Monty Python). Guido van Rossum

**Objetivos do Python**

Em 1999, Guido van Rossum definiu seus objetivos para o Python:

* uma linguagem **fácil e intuitiva**, tão eficiente quanto a dos grandes concorrentes;
* **código aberto**, para que qualquer pessoa possa contribuir com seu desenvolvimento;
* código tão **compreensível** quanto o simples inglês;
* **adequado para tarefas diárias**, permitindo tempos de desenvolvimento curtos.

Cerca de 20 anos depois, fica claro que todas essas intenções foram cumpridas. Algumas fontes dizem que Python é a linguagem de programação mais popular do mundo, enquanto outras afirmam que é a segunda ou a terceira.

De qualquer forma, ele ainda ocupa uma posição alta entre os dez primeiros da Popularidade de Linguagem de Programação PYPL (<http://pypl.github.io/PYPL.html>) e do Índice da Comunidade de Programação TIOBE (<https://www.tiobe.com/tiobe-index/> ).

Python não é mais uma linguagem jovem. **É maduro e confiável.** Não é de admirar. É uma estrela no firmamento de programação, e o tempo gasto aprendendo Python é um investimento muito bom.

**1.2.4 O que torna o Python tão especial?**

O que torna o Python tão especial?

**Por que Python?**

Como é que os programadores, novos e antigos, experientes e iniciantes, querem usá-lo? Como aconteceu que as grandes empresas adotaram o Python e implementaram seus principais produtos usando-o?

Há muitas razões - listamos algumas delas já, mas vamos enumerá-las novamente de uma forma mais prática:

* **É fácil de aprender** - o tempo necessário para aprender Python é mais curto do que para muitas outras linguagens; isso significa que é possível iniciar a programação real mais rapidamente;
* **é fácil de ensinar** - a carga de trabalho de ensino é menor do que a necessária em outros idiomas; isso significa que o professor pode colocar mais ênfase nas técnicas de programação geral (independentes do idioma), não desperdiçando energia em truques exóticos, estranhas exceções e regras incompreensíveis;
* **É fácil de usar para escrever novos softwares -** muitas vezes é possível escrever código mais rápido ao usar Python;
* **É fácil de entender** - também é mais fácil entender o código de outra pessoa com mais rapidez, se for escrito em Python;
* **É fácil de obter, instalar e implantar** – Python é gratuito, aberto e multiplataforma; nem todas as linguagens podem se orgulhar disso.



**1.2.5 Rivais do Python?**

O Python tem dois concorrentes diretos, com propriedades e predisposições comparáveis. São eles:

* **Perl** - uma linguagem de script criada originalmente por **Larry Wall**;
* **Ruby** - uma linguagem de script criada por **Yukihiro Matsumoto**.

O primeiro é mais tradicional e mais conservador do que o Python, e se assemelha a algumas das linguagens antigas derivadas da linguagem de **programação clássica C**.

Em contrapartida, o último é mais inovador e mais cheio de novas idéias do que o Python. O próprio Python está em algum lugar entre essas duas criações.

A Internet está cheia de fóruns com discussões infinitas sobre a superioridade de um desses três sobre os outros, caso você queira saber mais sobre cada um deles.

**1.2.6 Onde podemos ver Python em ação?**

Vemos isso todos os dias e quase em todos os lugares. É amplamente utilizado para implementar serviços de Internet complexos, como mecanismos de pesquisa, armazenamento em nuvem e ferramentas, mídias sociais e assim por diante. Sempre que você usa qualquer um desses serviços, você está muito próximo do Python, embora não saiba.

Muitas ferramentas de desenvolvimento são implementadas em Python. Cada vez mais aplicativos de uso diário estão sendo escritos em Python. Muitos cientistas abandonaram ferramentas proprietárias caras e mudaram para o Python. Muitos testadores de projeto de TI começaram a usar o Python para realizar procedimentos de teste repetíveis. A lista é longa.



**1.2.7 Por que não Python?**

Apesar da crescente popularidade do Python, ainda há alguns nichos nos quais o Python está ausente ou é raramente visto:

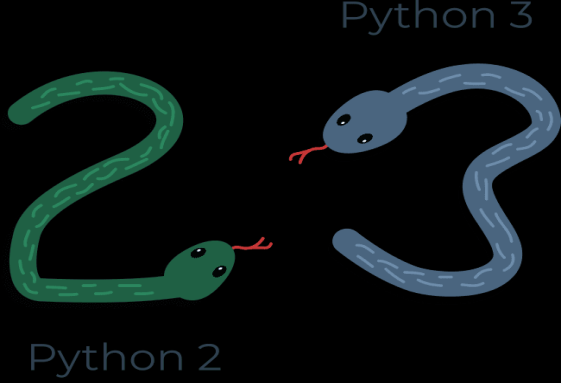
* **programação de baixo nível (às vezes chamada de programação "próxima ao metal"):** se você quiser implementar um driver ou mecanismo gráfico extremamente eficaz, não usará Python;
* **aplicativos para dispositivos móveis:** embora esse território ainda esteja esperando para ser conquistado pelo Python, provavelmente acontecerá algum dia.

**1.2.8 Há mais de um python**

**Python 2 vs. Python 3**

Existem dois tipos principais de Python, chamados Python 2 e Python 3.

**Python 2** é uma versão mais antiga do Python original. Seu desenvolvimento foi intencionalmente interrompido, embora isso não signifique que não há atualizações para ele. Pelo contrário, as atualizações são emitidas regularmente, mas não pretendem modificar a linguagem de forma significativa. Eles preferem corrigir bugs e falhas de segurança recém-descobertos. O caminho de desenvolvimento do Python 2 já chegou a um impasse, mas o próprio Python 2 ainda está vivo.



**Python 3** é a versão mais recente (ou para ser mais preciso, a atual) da linguagem. Ela está seguindo seu próprio caminho evolutivo, criando seus próprios padrões e hábitos.

Essas duas versões do Python não são compatíveis entre si. Os scripts Python 2 não serão executados em um ambiente Python 3 e vice-versa. Se você quiser que o código Python 2 antigo seja executado por um intérprete Python 3, a única solução possível é reescrevê-lo, não do zero, é claro, pois grandes partes do código podem permanecer intocadas, mas você precisa revisar todo o código para encontrar todas as incompatibilidades possíveis. Infelizmente, esse processo não pode ser totalmente automatizado.

É muito difícil, muito demorado, muito caro e muito arriscado migrar uma aplicação antiga do Python 2 para uma nova plataforma, e é até possível que a reescrita do código introduza novos bugs nela. É mais fácil e mais sensato deixar esses sistemas em paz e melhorar o intérprete atual, em vez de tentar trabalhar dentro do código-fonte que já está em funcionamento.

O Python 3 não é apenas uma versão melhor do Python 2 - é uma linguagem completamente diferente, embora muito semelhante ao seu antecessor. Quando você os observa à distância, eles parecem ser os mesmos, mas quando você olha de perto, você percebe muitas diferenças.

Importante: Se você estiver modificando uma solução Python antiga, é muito provável que ela tenha sido codificada em Python 2. Essa é a razão pela qual o Python 2 ainda está em uso. Há muitos aplicativos Python 2 para descartá-lo completamente.

**Observação**

Se você vai começar um novo projeto Python, deve usar o Python 3, e esta é a versão do Python que será usada durante este curso.

É importante lembrar que pode haver diferenças menores ou maiores entre as versões subsequentes do Python 3 (por exemplo, o Python 3.6 introduziu chaves de dicionário ordenadas por padrão na implementação do CPython) - a boa notícia é que todas as versões mais recentes do Python 3 são compatíveis com as versões anteriores do Python 3. Sempre que significativo e importante, sempre tentaremos destacar essas diferenças no curso.

Todos os exemplos de código que você encontrará durante o curso foram testados em Python 3.4, Python 3.6, Python 3.7, Python 3.8 e Python 3.9.

**1.2.9 Implementações de Python**

Além de Python 2 e Python 3, há mais de uma versão de cada.

Seguindo a página wiki do Python (<https://wiki.python.org/moin/PythonImplementations> ) , uma implementação do Python se refere a "um programa ou ambiente, que oferece suporte para a execução de programas escritos na linguagem Python, conforme representado pela implementação de referência do CPython".

A implementação tradicional do Python, chamada CPython, é a versão de referência da linguagem de computação Python de Guido van Rossum, e na maioria das vezes é chamada apenas de "Python". Quando você ouve o nome CPython, ele provavelmente é usado para diferenciá-lo de outras implementações alternativas não tradicionais.

Mas, primeiro, as coisas. Existem Pythons mantidos por pessoas reunidas em torno do **PSF (Python Software Foundation) (**[**https://www.python.org/psf-landing/**](https://www.python.org/psf-landing/) **),** uma comunidade que tem como objetivo desenvolver, melhorar, expandir e popularizar o Python e seu ambiente. O presidente do PSF é o próprio Guido von Rossum e, por essa razão, esses Pythons são chamados de **canônicos**. Eles também são considerados **Pythons de referência**, pois qualquer outra implementação da linguagem deve seguir todos os padrões estabelecidos pelo PSF.

Guido van Rossum usou a linguagem de programação "C" para implementar a primeira versão de sua linguagem, e essa decisão ainda está em vigor. Todos os Pythons provenientes do PSF são escritos na linguagem "C". Há muitas razões para essa abordagem. Uma delas (provavelmente a mais importante) é que, graças a ela, o Python pode ser facilmente portado e migrado para todas as plataformas com a capacidade de compilar e executar programas de linguagem "C" (praticamente todas as plataformas têm esse recurso, o que abre muitas expansões oportunidades para o Python).

É por isso que a implementação do PSF é conhecida como **CPython**. Este é o Python mais influente entre todos os Pythons do mundo.

This component is a flipcard comprised of flippable cards containing display image. Select the front face image to flip to the back face of these card to display associated text.

Clique nas imagens para saber mais sobre os membros da família Python e algumas das implementações alternativas mais populares do Python.

|  |  |
| --- | --- |
| https://skillsforall.com/content/pe1/1.0/m1/course/pt-BR/assets/2f6ec470920f1cebe2d423d059be0da58b9e9088.png | **Cython** é uma das possíveis soluções para a mais dolorosa das características da Python - a falta de eficiência. Cálculos matemáticos grandes e complexos podem ser facilmente codificados em Python (muito mais fácil do que em "C" ou em qualquer outra linguagem tradicional), mas a execução de código resultante pode ser extremamente demorada.  Como essas duas contradições são reconciliadas? Uma solução é escrever suas idéias matemáticas usando Python, e quando você tiver certeza absoluta de que seu código está correto e produz resultados válidos, você pode convertê-lo em "C". Certamente, "C" será executado muito mais rápido do que Python puro.  Isso é o que o Cython se destina a fazer para **converter automaticamente o código Python** (limpo e claro, mas não muito rápido) **em código "C"** (complicado e falante, mas ágil). |
| https://skillsforall.com/content/pe1/1.0/m1/course/pt-BR/assets/7391f3e83666d75ca9942ee3d61a5d4603a65b88.png | Outra versão do Python é chamada **Jython**.  "J" é para "Java". Imagine um Python escrito em Java em vez de C. Isso é útil, por exemplo, se você desenvolver sistemas grandes e complexos escritos inteiramente em Java e quiser adicionar alguma flexibilidade Python a eles. O CPython tradicional pode ser difícil de integrar a esse ambiente, pois C e Java vivem em mundos completamente diferentes e não compartilham muitas ideias comuns.  O **Jython pode se comunicar com a infraestrutura Java atual com mais eficiência.** É por isso que alguns projetos acham útil e necessário.  Observação: a implementação atual do Jython segue os padrões do Python 2. Até o momento, não há nenhum Jython em conformidade com o Python 3. |
| https://skillsforall.com/content/pe1/1.0/m1/course/pt-BR/assets/7028625f2a5a77b4b7014ac76a8f4bc8f07a0829.png | O logotipo do **PyPy** é um jogo de quebra-cabeça. Você pode resolver o problema? Isso significa: um Python dentro de um Python. Em outras palavras, ele representa um ambiente Python escrito em uma linguagem semelhante a Python chamada **RPython (Python restrito**). Na verdade, é um subconjunto do Python.  O código fonte do PyPy não é executado na maneira de interpretação, mas, em vez disso, **é traduzido para a linguagem de programação C e executado separadamente**.  Isso é útil porque se você quiser testar qualquer novo recurso que possa ser (mas não precise ser) introduzido na implementação principal do Python, é mais fácil verificar isso com o PyPy do que com o CPython. É por isso que o PyPy é mais uma ferramenta para as pessoas que desenvolvem o Python do que para o restante dos usuários.  Isso não torna o PyPy menos importante ou menos sério do que o CPython, é claro.  Além disso, o PyPy é compatível com a linguagem Python 3.  Há muito mais Pythons diferentes no mundo. Você os encontrará se olhar, mas este curso se concentrará no CPython. |
| https://skillsforall.com/content/pe1/1.0/m1/course/pt-BR/assets/59cfd19e1241c018d1f24b29d877baaef220e09b.png | O **MicroPython** é uma implementação eficiente de software de código aberto do Python 3, que é **otimizada para ser executada em microcontroladores**. Ele inclui um pequeno subconjunto da biblioteca padrão do Python, mas é amplamente embalado com um grande número de recursos, como prompt interativo ou números inteiros de precisão arbitrária, bem como módulos que dão ao programador acesso ao hardware de nível inferior.  Originalmente criado por Damien George, um programador australiano que, no ano de 2013, realizou uma campanha bem-sucedida no Kickstarter e lançou a primeira versão do MicroPython com uma placa de desenvolvimento baseada em STM32F4 chamada pyboard.  Em 2017, o MicroPython foi usado para criar o CircuitPython, outra linguagem de programação de código aberto executada no hardware do microcontrolador, que é um derivado da linguagem MicroPython. |

**1.3 Seção 3 - Download e instalação do Python**

Bem-vindo à seção três, onde falaremos sobre as formas de obter, instalar e configurar o Python no seu computador local. Esta seção é opcional, pois durante o curso você poderá iniciar, testar e experimentar todos os seus programas Python no **Edube Interactive TM**, o ambiente de programação que integramos à plataforma de aprendizagem e a esses recursos de estudo. Ainda assim, se você pode baixar e instalar o Python em sua máquina local, é altamente recomendável.

**1.3.1 Comece sua jornada Python**

**Obter o Python e como usá-lo**

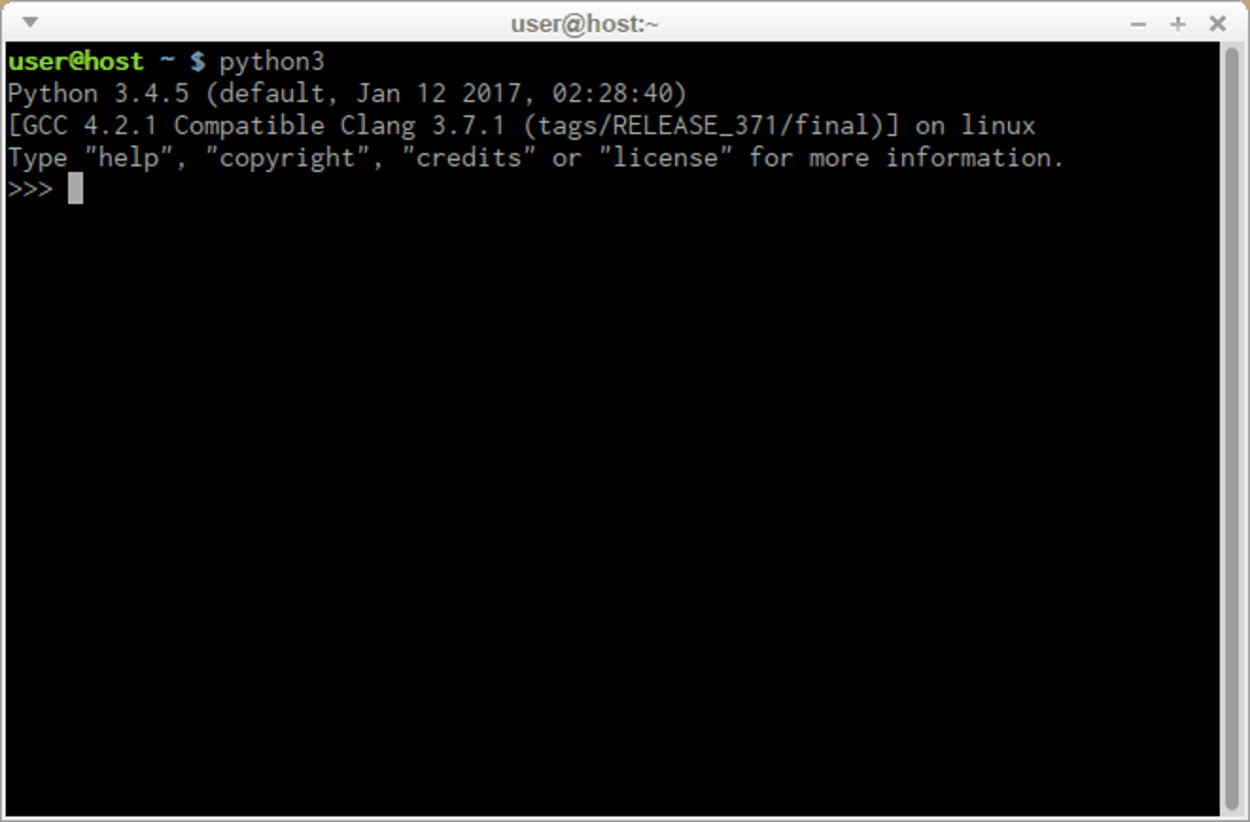
Existem várias maneiras de obter sua própria cópia do Python 3, dependendo do sistema operacional que você usa.

Os usuários de Linux provavelmente já têm o Python instalado - esse é o cenário mais provável, pois a infraestrutura do Python é usada intensivamente por muitos componentes de SO Linux.

Por exemplo, alguns distribuidores podem associar suas ferramentas específicas ao sistema e muitas dessas ferramentas, como gerenciadores de pacotes, geralmente são escritas em Python. Algumas partes dos ambientes gráficos disponíveis no mundo Linux também podem usar Python.

Se você for um usuário de Linux, abra o terminal/console e digite:

no prompt do shell, pressione Enter e aguarde. Se você vir algo assim:



Então você não precisa fazer mais nada.

Se o Python 3 estiver ausente, consulte a documentação do Linux para descobrir como usar o gerenciador de pacotes para baixar e instalar um novo pacote - o que você precisa se chama python3 ou o nome começa com isso.

Todos os usuários não Linux podem baixar uma cópia em <https://www.python.org/downloads/> .

**1.3.2 Como baixar, instalar e configurar o python**

Como o navegador informa ao site que você entrou no sistema operacional utilizado, a única etapa a ser seguida é clicar na versão apropriada do Python desejada.

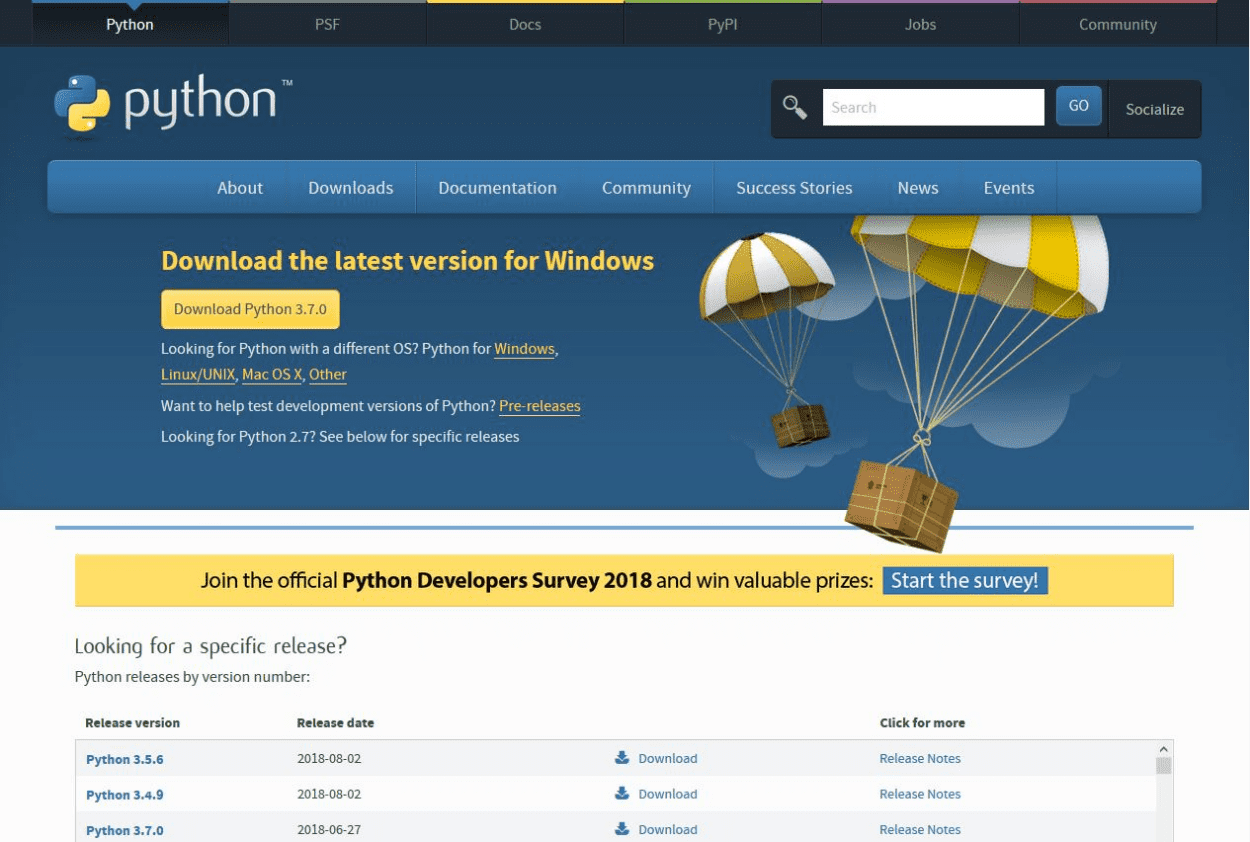
Nesse caso, selecione Python 3. O site oferece sempre a versão mais recente.

Se você é um usuário do **Windows**, inicie o arquivoownload.exe e siga todas as etapas.

Deixe as configurações padrão que o instalador sugere por enquanto, com uma exceção - observe a caixa de seleção chamada Adicionar Python 3.x ao PATH e marque-a.

Isso facilitará as coisas.

Se você for um usuário do **macOS**, uma versão do Python 2 pode já ter sido pré-instalada no seu computador, mas como trabalharemos com o Python 3, você ainda precisará baixar e instalar o arquivo relevantes.pkg no site do Python.



**1.3.3 Começando seu trabalho com Python**

Agora que você tem o Python 3 instalado, é hora de verificar se ele funciona e fazer o primeiro uso dele.

Esse será um procedimento muito simples, mas deve ser suficiente para convencê-lo de que o ambiente Python é completo e funcional.

Há muitas maneiras de utilizar o Python, especialmente se você for um desenvolvedor de Python.

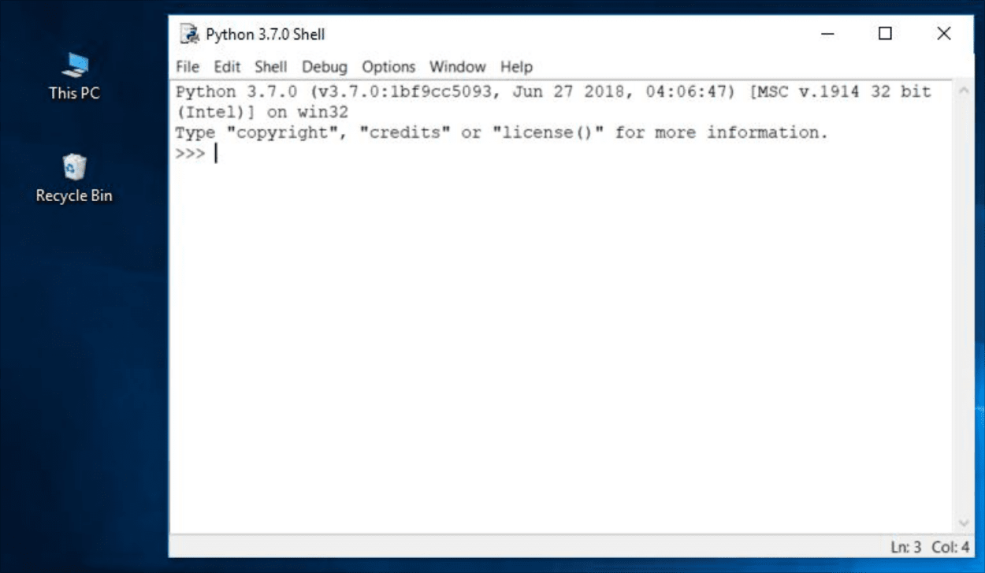
Para começar o trabalho, você precisa das seguintes ferramentas:

* um **editor** que ajudará você a escrever o código (ele deve ter alguns recursos especiais, não disponíveis em ferramentas simples); este editor dedicado oferecerá mais do que o equipamento padrão do SO;
* um **console** no qual é possível iniciar o código recém-escrito e interrompê-lo à força quando ele ficar fora de controle;
* uma ferramenta chamada **debugger**, capaz de iniciar o código passo a passo, o que permitirá que você inspecione o código a cada momento de execução.

Além de seus muitos componentes úteis, a instalação padrão do Python 3 contém um aplicativo muito simples, mas extremamente útil **chamado IDLE.**

**IDLE** significa **Integrated Development and Learning Environment** (Desenvolvimento e ambiente de aprendizado integrados).

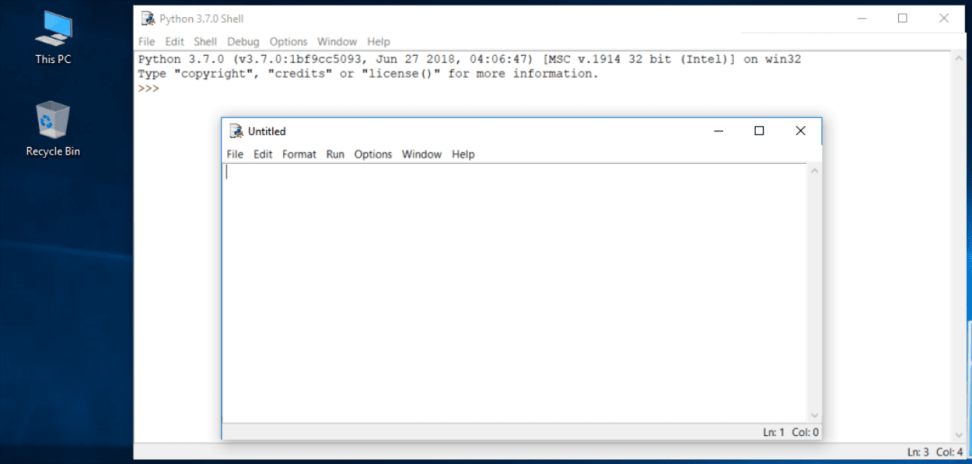
Navegue pelos menus do SO, encontre IDLE em algum lugar no Python 3.x e inicie-o. Isso é o que você deve ver:



**1.3.4 Seu primeiro programa antes do seu primeiro programa.**

Agora é a hora de escrever e executar o seu primeiro programa Python 3. Vai ser muito simples, por enquanto.

A primeira etapa é criar um novo arquivo de origem e preenchê-lo com o código. Clique em Arquivo no menu IDLE e selecione New file.



Como você pode ver, o IDLE abre uma nova janela para você. Você pode usá-lo para escrever e alterar seu código.

Esta é a janela do editor. Seu único objetivo é ser um local de trabalho no qual o código-fonte é tratado. Não confunda a janela do editor com a janela do shell. Eles desempenham funções diferentes.

A janela do editor não tem título, mas é uma boa prática começar o trabalho nomeando o arquivo de origem.

Clique em Arquivo (na nova janela), clique em Salvar como..., selecione uma pasta para o novo arquivo (a área de trabalho é um bom local para suas primeiras tentativas de programação) e escolha um nome para o novo arquivo.

Observação: não defina nenhuma extensão para o nome do arquivo que você usará. O Python precisa que seus arquivos tenham a extensão .py, então você deve confiar nos padrões da janela de diálogo. O uso da extensão padrão .py permite que o sistema operacional abra esses arquivos corretamente.

Agora, coloque apenas uma linha na janela do editor recém-aberta e nomeada.

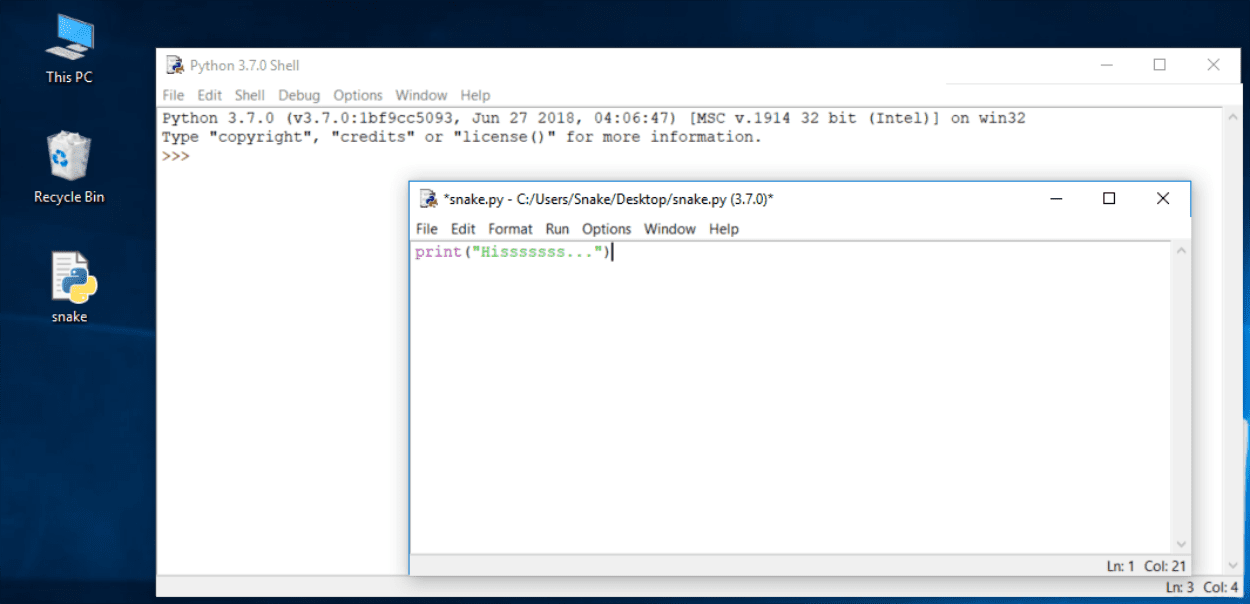
A linha fica assim:

*print*("Hisssssss...")

Você pode usar a área de transferência para copiar o texto no arquivo.

Não explicaremos o significado do programa no momento. Você encontrará uma discussão detalhada no próximo capítulo.

Veja mais de perto as aspas. Essas são as formas mais simples de aspas (neutras, diretas, mudas etc.) comumente usadas em arquivos de origem. Não tente usar aspas tipográficas (curvas, curvas, inteligentes, etc.), usadas por processadores de texto avançados, pois o Python não as aceita.



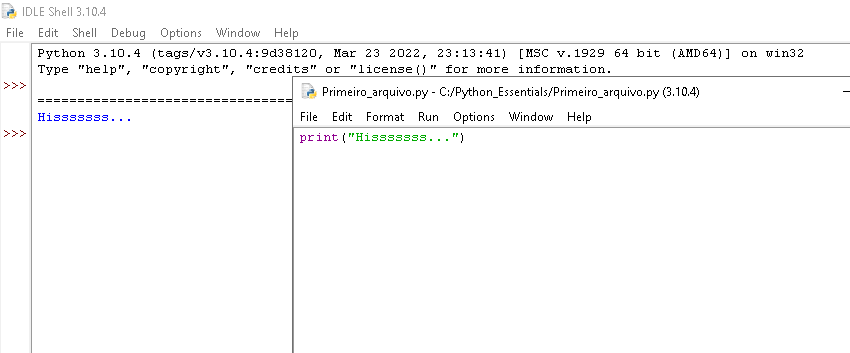
Salve o arquivo (Arquivo -> Salvar) e execute o programa (Executar -> Executar módulo).

Se tudo correr bem e não houver erros no código, a janela do console mostrará os efeitos causados pela execução do programa.

Nesse caso, o programa rs.

Tente executá-lo novamente. E mais uma vez.

Agora feche as duas janelas e volte para a área de trabalho.



**1.3.5 Como estragar e corrigir seu código**

Agora inicie o **IDLE** novamente.

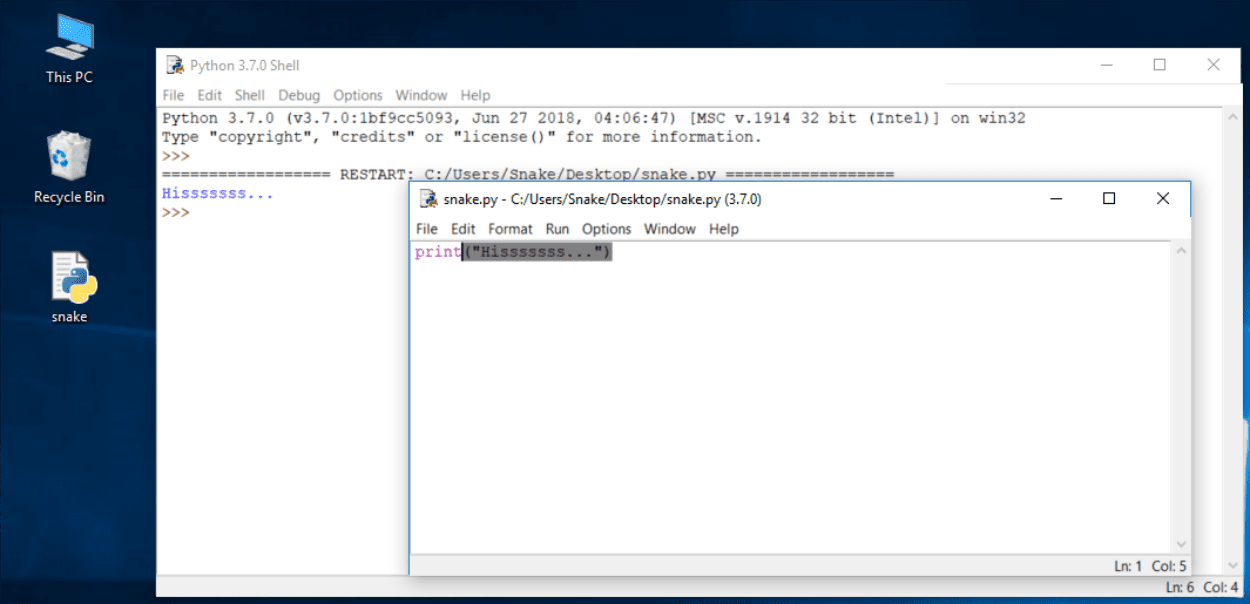
* Clique em Arquivo, Abrir, aponte para o arquivo que você salvou anteriormente e deixe que o IDLE o leia.
* Tente executá-lo novamente pressionando ***F5*** quando a janela do editor estiver ativa.
* Como você pode ver, o IDLE pode salvar seu código e recuperá-lo quando você precisar dele novamente.
* Ocioso contém um recurso adicional e útil.
* Primeiro, remova os parênteses de fechamento.
* Em seguida, insira o parênteses novamente.

Seu código deve se parecer com o seguinte:

Output

Hisssssss...

Cada vez que você colocar os parênteses de fechamento no programa, o IDLE exibirá a parte do texto limitada por um par de parênteses. Isso ajuda a lembrar de colocá-los em pares.



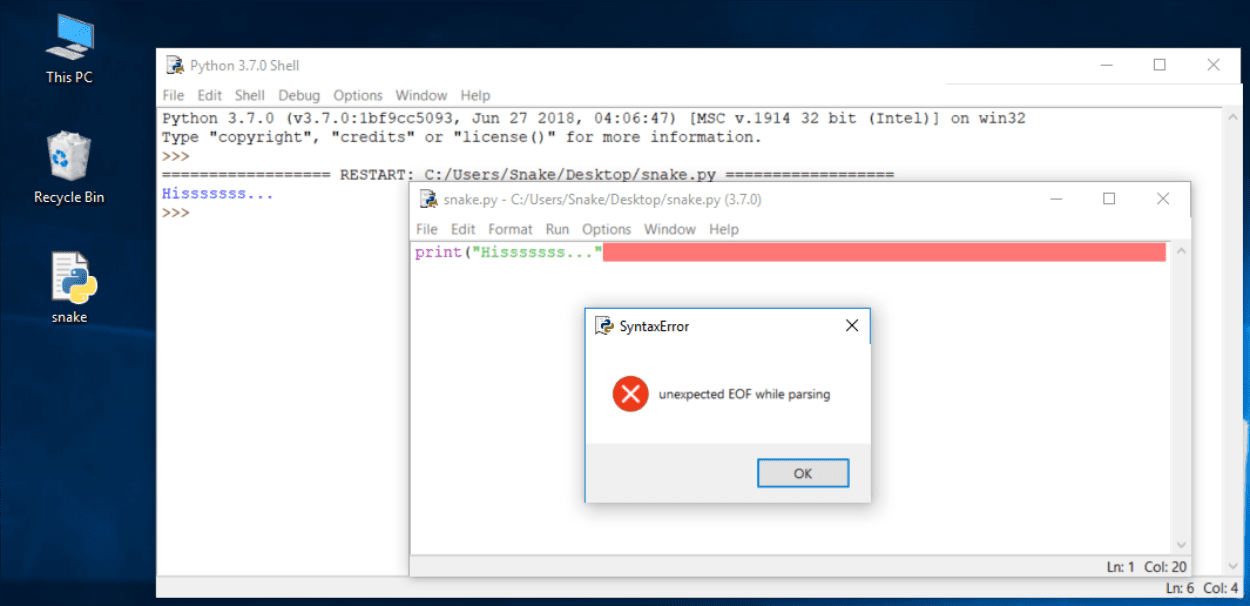
Remova os parênteses de fechamento novamente. o código está errado Ele contém um erro de sintaxe agora. O Ocioso não deve permitir que você o execute.

Tente executar o programa novamente. O Ocioso lembrará que você deve salvar o arquivo modificado. Siga as instruções.

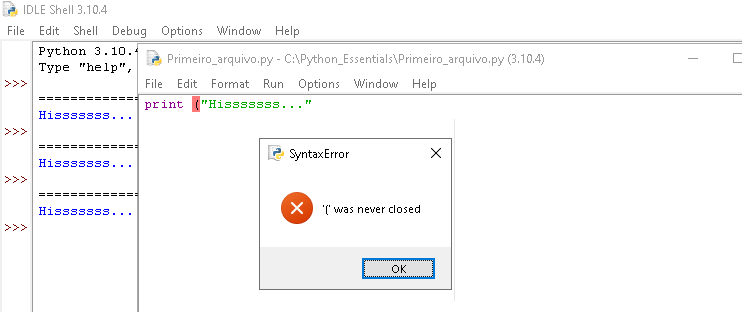
Observe todas as janelas com cuidado.

Uma nova janela aparece - Ele diz que o intérprete encontrou um EOF (fim de arquivo) embora (na sua opinião) o código deva conter um pouco mais de texto.

A janela do editor mostra claramente onde isso aconteceu.



ou

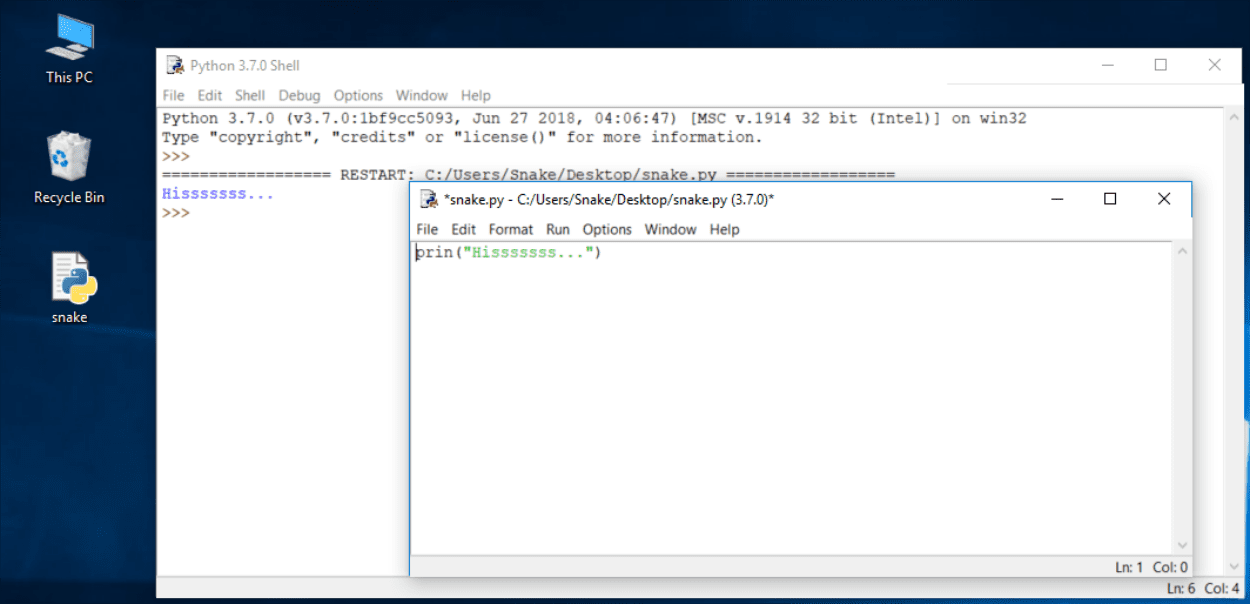


Corrija o código agora. Ele deve ficar assim:

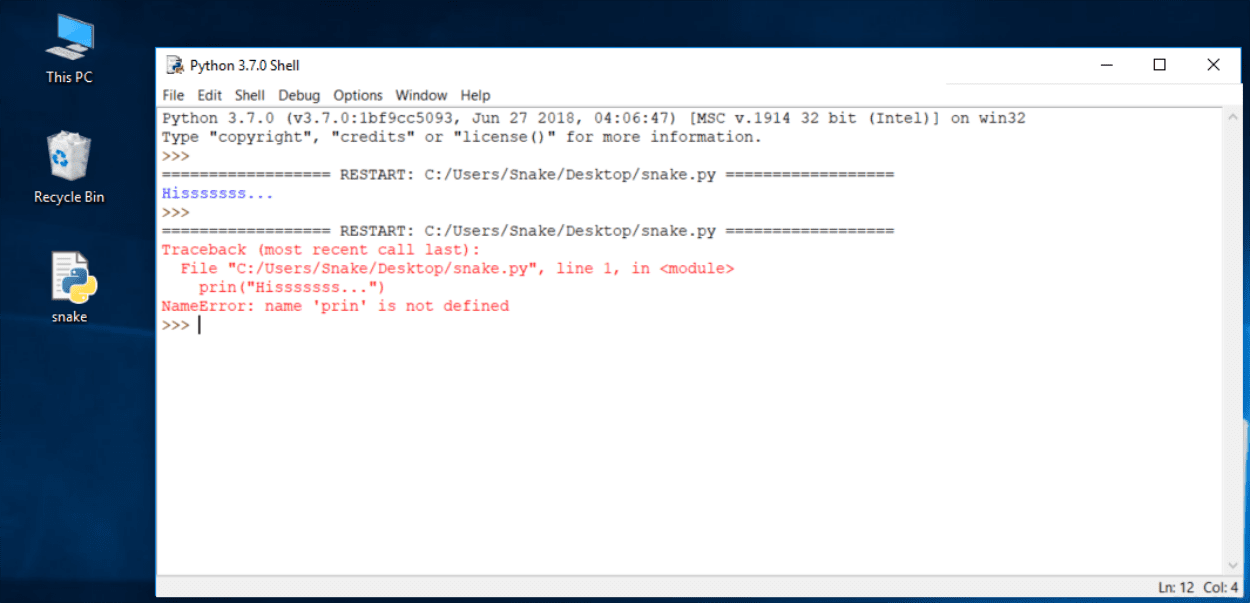
print("Hisssssss...")

Execute-o para ver se ele "assobia" novamente.

Vamos estragar o código mais uma vez. Remova uma letra da palavra print. Execute o código pressionando F5. O que ocorre agora? Como você pode ver, o Python não é capaz de reconhecer a instrução.



Você deve ter notado que a mensagem de erro gerada para o erro anterior é bem diferente da primeira.



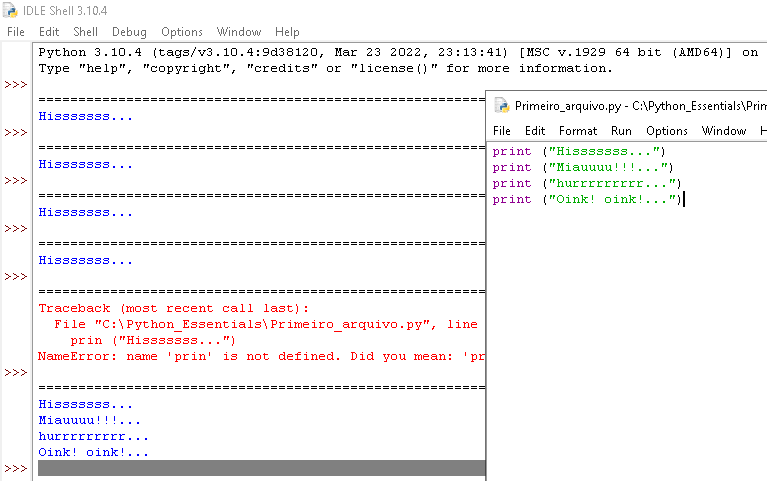
Isso ocorre porque a natureza do erro é diferente e o erro é descoberto em um estágio diferente de interpretação.

A janela do editor não fornecerá nenhuma informação útil sobre o erro, mas as janelas do console podem fornecer.

A mensagem (em vermelho) mostra (nas linhas subsequentes):

* **o retorno de rastreamento** (que é o caminho que o código percorre em diferentes partes do programa - você pode ignorá-lo por enquanto, pois está vazio em um código tão simples);
* **a localização do erro** (o nome do arquivo que contém o erro, o número da linha e o nome do módulo); observação: o número pode ser enganoso, pois o Python geralmente mostra o local onde primeiro nota os efeitos do erro, não necessariamente o erro em si;
* **o conteúdo da linha incorreta**; observação: a janela do editor do IDLE não mostra números de linha, mas exibe a localização atual do cursor no canto inferior direito; use-o para localizar a linha incorreta em um código-fonte longo;
* o nome do erro e uma breve explicação.

Experimente criar novos arquivos e executar seu código. Tente enviar uma mensagem diferente para a tela, por exemplo, **rugido!, miado**, ou até mesmo um **oink!.** Tente estragar e corrigir o código - veja o que acontece.



**1.4 Conclusão do módulo 1 - Teste de módulo**

**Teste do Módulo 1**

Muito bem! Você chegou ao final do Módulo 1 e concluiu um grande marco na sua formação em programação em Python. Aqui está um breve resumo das áreas de tópicos abordadas no Módulo 1:

* os fundamentos da programação de computadores, ou seja, como o computador funciona, como o programa é executado, como a linguagem de programação é definida e construída;
* a diferença entre compilação e interpretação;
* as informações básicas sobre Python e como ele está posicionado entre outras linguagens de programação e o que distingue suas diferentes versões;
* como obter, instalar e configurar o Python em sua máquina local.

Agora você está pronto para fazer o teste do módulo, o que ajudará a avaliar o que você aprendeu até agora.

O teste a seguir tem como base o que você acabou de aprender. Há dez perguntas no total e você precisa marcar pelo menos 70% para passar.

Boa sorte!

1. **Pergunta 1**

Questão de múltipla escolha

O que é código de máquina?

* Uma linguagem de programação de nível médio que consiste no código de montagem projetado para o processador do computador
* Uma linguagem de programação de alto nível que consiste em listas de instruções que os humanos podem ler e entender
* Uma linguagem de programação de baixo nível que consiste em dígitos hexadecimais que compõem instruções de linguagem de alto nível
* Uma linguagem de programação de baixo nível que consiste em dígitos/bits binários que o computador lê e entende

Obs: Linguagem de máquina (machine languale): **é a linguagem de mais baixo nível de entendimento pelo ser humano e a única, na verdade, entendida pelo processador (UCP)**. É constituída inteiramente de números, o que torna praticamente impossível entendê-la diretamente. Conceituamos também como linguagem de máquina binário é o decimal codificado em binário, em que os números decimais são codificados em formato binário. Cada algarismo decimal é codificado como um número binário de quatro dígitos, como está no exemplo abaixo: 0000 = 0. 0001 = 1

1. **Pergunta 2**

Questão de múltipla escolha

Quais são os quatro elementos fundamentais que compõem uma linguagem?

* Um alfabeto, fonética, fonologia e semântica
* Um alfabeto, um lexis, fonética e semântica
* Um alfabeto, um lexis, uma sintaxe e semânticas
* Um alfabeto, morfologia, fonética e semântica

1. **Pergunta 3**

Como você chama um arquivo que contém um programa escrito em uma linguagem de programação de alto nível?

* Um arquivo de máquina
* Um arquivo de destino
* Um arquivo de código
* Um arquivo de fonte

1. **Pergunta 4**

Questão de múltipla escolha

O que é verdadeiro sobre a compilação? (Selecione duas respostas)

* Você e o usuário final devem ter o compilador para executar o código
* Ela tende a ser mais lenta que a interpretação
* O código é convertido diretamente em código de máquina executável pelo processador
* Ela tende a ser mais rápida do que a interpretação

OBS: Qual é a diferença entre compilado é interpretado?

Ao utilizar o método de compilação, o compilador irá ler o código, fazer todas as análises sintáticas e demais processos, para, por fim, gerar um arquivo código-objeto ou um arquivo executável. Já na interpretação, nenhum arquivo ou código é gerado, e sim uma tradução instantânea, em tempo de execução.

1. **Pergunta 5**

Questão de múltipla escolha

Qual é a definição de um arquivo de script?

* É um arquivo de texto que contém sequências de zeros e uns
* É uma mensagem de erro gerada pelo interpretador
* É uma mensagem de erro gerada pelo compilador
* É um arquivo de texto que contém instruções que compõem um programa Python

Obs: Um script é um código de programação que reúne instruções para que o PC execute tarefas e pode ser utilizado para programar infinitas funções. O script é uma série de instruções para que o PC execute determinadas tarefas segundo programado

1. **Pergunta 6**

Questão de múltipla escolha

Selecione as afirmações verdadeiras. (Selecione duas respostas)

* Python é gratuito, de código aberto e multiplataforma
* Python 3 é retrocompatível com Python 2
* Python é uma boa opção para criar e executar testes para aplicações
* Python é uma boa escolha para programação de baixo nível, por exemplo, quando você deseja implementar um driver eficaz

1. **Pergunta 7**

Questão de múltipla escolha

O que é o CPython?

* É a implementação de referência padrão do Python, escrita na linguagem C
* É uma linguagem de programação que é um superconjunto da linguagem C, projetada para produzir desempenho semelhante ao Python com código escrito em C
* É uma linguagem de programação que é um superconjunto do Python, projetada para produzir desempenho em C com código escrito em Python
* É a implementação de referência padrão da linguagem C, escrita em Python

1. **Pergunta 8**

Questão de múltipla escolha

O que você chama de interpretador de linha de comando que permite interagir com o SO e executar scripts e comandos em Python?

* um compilador
* Jython
* Um Console
* Um editor

1. **Pergunta 9**

Questão de múltipla escolha

Qual é o comportamento esperado do seguinte programa?

print("Olá!")

* O programa gerará uma mensagem de erro na tela
* O programa exibirá "Olá!" na tela
* O programa emitirá um Olá! na tela
* O programa exibirá ("Olá!") na tela

1. **Pergunta 10**

Questão de múltipla escolha

Qual é o comportamento esperado do seguinte programa?

prin("Adeus!")

* O programa gerará uma mensagem de erro na tela
* O programa emitirá Adeus! na tela
* O programa exibirá ("Adeus!") na tela
* O programa exibirá "Adeus!" na tela

**2.1 Seção 1 - O Programa "Olá, mundo!"**

Bem-vindo ao módulo dois! Na primeira seção, vamos aprender sobre os elementos mais essenciais de sintaxe e semântica da linguagem Python, e usá-los para criar seu primeiro programa Python - "Olá, mundo!".

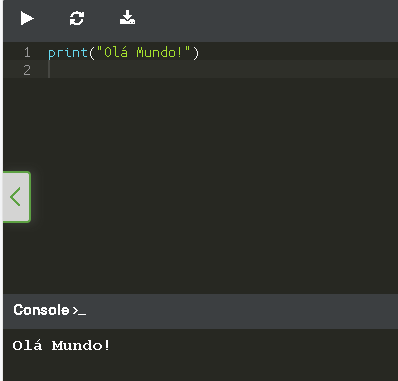
**2.1.1 Seu primeiro programa**

É hora de começar a escrever algum código Python real e funcional. Vai ser muito simples por enquanto.

Como mostraremos alguns conceitos e termos fundamentais, esses trechos de código não serão tão sérios ou complexos.

Execute o código na janela do editor. Se tudo correr bem aqui, você verá a linha de texto na janela do console.

Como alternativa, inicie o IDLE, crie um novo arquivo de origem Python, preencha-o com este código, nomeie o arquivo e salve-o. Agora execute-o. Se tudo correr bem, você verá o texto entre aspas na janela do console IDLE. O código que você executou deve parecer familiar. Vocês viram algo muito semelhante quando orientamos vocês pela configuração do ambiente IDLE.



Agora vamos passar algum tempo mostrando e explicando o que você realmente está vendo e por que isso se parece com isso.

Como você pode ver, este primeiro programa consiste nas seguintes partes:

* a palavra print;
* um parênteses de abertura;
* aspas (")
* uma linha de texto: Olá, Mundo!;
* outra aspa;
* outro parênteses.

Cada uma das opções acima desempenha um papel muito importante no código.

**2.1.2 A função print()**

Veja a linha de código abaixo:

print("Olá, Mundo!")

A **palavra print** que você pode ver aqui é um nome de **função**. Isso não significa que, sempre que a palavra aparece, sempre há um nome de função. O significado da palavra vem do contexto em que a palavra foi usada.

Você provavelmente já encontrou o termo função muitas vezes, durante as aulas de matemática. Você provavelmente também pode listar vários nomes de funções matemáticas, como senoidal ou log.

As funções do Python, no entanto, são mais flexíveis e podem conter mais conteúdo do que seus irmãos matemáticos.

Uma função (nesse contexto) é uma parte separada do código do computador capaz de:

* causar algum efeito (por exemplo, envie texto para o terminal, crie um arquivo, desenhe uma imagem, toque um som etc.); isso é algo totalmente inédito no mundo da matemática;
* avaliar um valor (por exemplo, a raiz quadrada de um valor ou o comprimento de um determinado texto) e retorná-lo como resultado da função; é isso que torna as funções do Python parentes de conceitos matemáticos.

Além disso, muitas funções Python podem fazer as duas coisas acima juntas.

**De onde vêm as funções?**

* Do próprio Python

Elas podem vir do próprio Python; a função print é desse tipo; essa função é um valor agregado recebido junto com o Python e seu ambiente (é integrado); você não precisa fazer nada de especial (por exemplo, pedir algo a alguém) se quiser fazer uso;

* Dos módulos

Elas podem vir de um ou mais dos complementos do Python chamados módulos; alguns dos módulos vêm com Python, outros podem exigir instalação separada. Seja qual for o caso, todos precisam estar explicitamente conectados ao seu código (mostraremos como fazer isso em breve);

* Do seu código

Você mesmo pode criá-las, colocando todas as funções que desejar e necessárias dentro do programa para torná-lo mais simples, claro e elegante.

O nome da função deve ser significativo (o nome da função de impressão é evidente).

Claro, se você vai usar qualquer função já existente, você não tem influência sobre seu nome, mas quando você começa a escrever suas próprias funções, você deve considerar cuidadosamente sua escolha de nomes.

**2.1.3 Argumentos de função**

Como dissemos antes, uma função pode ter:

* **um efeito;**
* **um resultado.**

Há também um terceiro componente de função muito importante ‒

* **o(s) argumento(s).**

As funções matemáticas geralmente levam um argumento. Por exemplo, sin(x) leva um x, que é a medida de um ângulo.

As funções Python, por outro lado, são mais versáteis. Dependendo das necessidades individuais, eles podem aceitar qualquer número de argumentos ‒ quantos forem necessários para realizar suas tarefas. Nota: quando dizemos qualquer número, que inclui zero ‒ algumas funções do Python não necessitam de argumento.

print("Olá Mundo!")

Apesar do número de argumentos necessários/fornecidos, as funções do Python exigem fortemente a **presença de um par de parênteses** ‒ abrindo e fechando, respectivamente.

Se você deseja entregar um ou mais argumentos para uma função, coloque-os dentro dos parênteses. Se você for usar uma função que não aceita nenhum argumento, ainda precisará dos parênteses.

Nota: para distinguir palavras comuns de nomes de funções, coloque um par de parênteses vazios após seus nomes, mesmo que a função correspondente precise de um ou mais argumentos. Essa é uma convenção padrão.

A função da qual estamos falando aqui é print().

A função print() em nosso exemplo tem algum argumento?

Claro que sim, mas o que são?

**Cadeia de caracteres como o argumento da função print()**

O único argumento entregue à função print() neste exemplo é uma **string**:

print("Olá Mundo!")

Como você pode ver, a string é delimitada por aspas - na verdade, as aspas formam a string - elas cortam uma parte do código e atribui um significado diferente a ele.

Você pode imaginar que as citações dizem algo como: o texto entre nós não é código. Não se destina a ser executado e você deve tomá-lo como está.

Quase tudo o que você coloca dentro das aspas será levado literalmente, não como código, mas como dados. Tente brincar com essa string específica - modifique-a, insira algum novo conteúdo, exclua parte do conteúdo existente.

Há mais de uma maneira de especificar uma string dentro do código do Python, mas por enquanto, porém, essa é suficiente.

Até agora, você aprendeu sobre duas partes importantes do código: a função e a string. Conversamos sobre eles em termos de sintaxe, mas agora é hora de discuti-los em termos de semântica.

**2.1.4 Chamada de função**

O nome da função (print neste caso), juntamente com os parênteses e os argumento(s), formam a invocação da função.

print("Olá Mundo!")

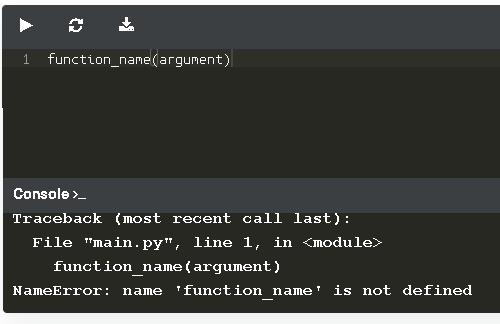
Vamos discutir isso em mais detalhes em breve, mas vamos esclarecer isso agora.

O que acontece quando o Python encontra uma invocação como esta abaixo?

function\_name(argument)

Vamos ver:

* Primeiro, o Python verifica se o nome especificado é legal (ele navega em seus dados internos para encontrar uma função atual do nome; se essa pesquisa falhar, o Python anula o código)
* segundo, o Python verifica se os requisitos da função para o número de argumentos permitem que você chame a função dessa maneira (por exemplo, se uma função específica exigir exatamente dois argumentos, qualquer invocação que forneça apenas um argumento será considerada errônea e abortará os execução)
* terceiro, o Python deixa seu código por um momento e salta para a função que você deseja chamar; é claro, ele também usa seus argumentos e os passa para a função;
* quarto, a função executa seu código, causa o efeito desejado (se houver), avalia o (s) resultado (s) desejado (s) (se houver) e termina sua tarefa;
* por fim, o Python retorna ao seu código (para o local imediatamente após a invocação) e retoma a execução.



**2.1.5 LAB Trabalhando com a função print()**

Cenário

O comando print(), que é uma das diretrizes mais fáceis em Python, simplesmente imprime uma linha na tela.

Em seu primeiro laboratório:

* Use a função de print() para imprimir a linha Olá, Python! na tela. Use aspas duplas ao redor da string.
* Depois de fazer isso, use a função print() novamente, mas desta vez imprima seu primeiro nome.
* Remova as aspas duplas e execute o código. Assista à reação do Python. Que tipo de erro é gerado?
* Em seguida, remova os parênteses, coloque aspas duplas e execute o código novamente. Que tipo de erro é gerado desta vez?
* Experimente o máximo que puder. Altere aspas duplas para aspas simples, use várias funções print() na mesma linha e, em seguida, em linhas diferentes. Veja o que acontece.

**print ("Olá, Python")**

**print ("Iraê")**

**print "Iraê"**

**print ('Iraê')**

**print (Iraê)**

**print "Iraê"**

Console

Olá, Python

Iraê

Olá, Python

Traceback (most recent call last):

File "main.py", line 2, in <module>

print (Iraê)

NameError: name 'Iraê' is not defined

File "main.py", line 2

print "Iraê"

^

SyntaxError: Missing parentheses in call to 'print'. Did you mean print("Iraê")?

File "main.py", line 3

print "Iraê"

^

SyntaxError: Missing parentheses in call to 'print'. Did you mean print("Iraê")?

**2.1.6 A função print() e seus efeitos, argumentos e valores retornados**

Três perguntas importantes devem ser respondidas o mais rápido possível:

**1. Que efeito a função print() causa?**

O efeito é muito útil e muito espetacular. A função:

* aceita seus argumentos (pode aceitar mais de um argumento e também pode aceitar menos de um argumento)
* Converte-os em forma legível pelo homem, se necessário (como você pode suspeitar, as strings não exigem essa ação, pois a string já é legível)
* e envia os dados resultantes para o dispositivo de saída (geralmente o console); Em outras palavras, tudo o que você coloca na função print() aparecerá na tela.

Não é de admirar que, a partir de agora, você utilizará a print() muito intensamente para ver os resultados de suas operações e avaliações.

**2. Quais argumentos o print() espera?**

Qualquer. Mostraremos em breve que o print() é capaz de operar com praticamente todos os tipos de dados oferecidos pelo Python. Strings, números, caracteres, valores lógicos, objetos - qualquer um desses itens pode ser passado com sucesso para print().

**3. Qual valor a função print() retorna?**

Nenhum. Seu efeito é suficiente.

**2.1.7 Instruções**

Você já viu um programa de computador que contém uma chamada de função. Uma invocação de função é um dos muitos tipos possíveis de instrução Python.

Obviamente, qualquer programa complexo geralmente contém muito mais instruções do que uma. A questão é: como você acopla mais de uma instrução no código Python?

A sintaxe do Python é bastante específica nessa área. Ao contrário da maioria das linguagens de programação, o Python exige que não haja mais de uma instrução em uma linha.

Uma linha pode estar vazia (ou seja, pode não conter nenhuma instrução), mas não deve conter duas, três ou mais instruções. Isso é estritamente proibido.

**Nota:** Python faz uma exceção a esta regra ‒ permite que uma instrução se espalhe por mais de uma linha (o que pode ser útil quando seu código contém construções complexas).

Vamos expandir um pouco o código. Você pode vê-lo no editor abaixo. Execute-o e observe o que você vê no console.

**print("A pequenina aranha escalou a tromba d'água.")**

**print("Caiu a chuva e lavou a aranha.")**

**Console**

**A pequenina aranha escalou a tromba d'água.**

**Caiu a chuva e lavou a aranha.**

Esta é uma boa oportunidade para fazer algumas observações:

* o programa chama a função print() duas vezes, e você pode ver duas linhas separadas no console ‒ isso significa que print() começa sua saída a partir de uma nova linha cada vez que inicia sua execução; você pode mudar esse comportamento, mas também pode usá-lo a seu favor;
* cada invocação print() contém uma string diferente, como seu argumento, e o conteúdo do console a reflete ‒ isso significa que as instruções no código são executadas na mesma ordem em que foram colocadas no arquivo fonte; nenhuma instrução subseqüente é executada até que a anterior seja concluída (existem algumas exceções a esta regra, mas você pode ignorá-las por enquanto.)

Mudamos um pouco o exemplo - adicionamos uma Invocação de função vazia print(). Chamamos de vazio porque não entregamos nenhum argumento para a função.

Você pode vê-lo na janela do editor. Execute o código.

O que acontece?

**print("A pequenina aranha escalou a tromba d'água.")**

**print()**

**print("Caiu a chuva e lavou a aranha.")**

**Console**

**A pequenina aranha escalou a tromba d'água.**

**Caiu a chuva e lavou a aranha.**

Como você pode ver, a invocação de print() vazia não é tão vazia quanto você esperava – ela produz uma linha vazia ou (esta interpretação também está correta) gera uma nova linha.

Esta não é a única maneira de produzir uma nova linha no console de saída. Agora vamos mostrar-lhe outra maneira.

**2.1.8 Escape do Python e caracteres de novas linhas**

Modificamos o código novamente. Olhe com atenção.

Há duas mudanças muito sutis – inserimos um estranho par de caracteres dentro da rima. Eles se parecem com isso: \n.

print("A aranha pequenininha**\n**subiu a tromba d'água.")

print()

print("abaixo veio a chuva**\n**e lavou a aranha.")

**Console**

**A aranha pequenininha**

**subiu a tromba d'água.**

**abaixo veio a chuva**

**e lavou a aranha.**

Curiosamente, enquantovocê pode ver dois caracteres, o Python vê um.

A barra invertida (\) tem um significado muito especial quando usada dentro de strings – isso é chamadode caractere de escape.

A palavra escape deve ser entendida especificamente - significa que a série de caracteres na string escapa por um momento (um momento muito curto) para introduzir uma inclusão especial.

Em outras palavras, a barra invertida não significa nada em si, mas é apenas um tipo de anúncio de que o próximo caractere após a barra invertida também tem um significado diferente.

A letra n colocada após a barra invertida vem da palavra newline (nova linha).

Tanto a barra invertida quanto o n formam um símbolo especial chamado caractere de nova linha, que incita o console a iniciar uma nova linha de saída.

Execute o código. Seu console agora deve ficar assim:

Output

A aranha pequenininha

subiu a tromba d'água.

abaixo veio a chuva

e lavou a aranha.

Como você pode ver, duas novas linhas aparecem na rima do berçário, nos lugares onde foram usados os \n.

Esta convenção tem duas consequências importantes:

1. Se você quiser colocar apenas uma barra invertida dentro de uma string, não se esqueça de sua natureza de escape - você precisa dobrá-la. Por exemplo, uma invocação como esta causará um erro:

**print("\")**

enquanto este não vai:

print("\\")

2. Nem todos os pares de escape (a barra invertida juntamente com outro caractere) significam algo.

**Experimente seu código no editor, execute-o e veja o que acontece.**

**print("A aranha pequenininha\\nsubiu a tromba d'água.")**

**print()**

**print("abaixo veio a chuva\\ne lavou a aranha.")**

**Console**

**A aranha pequenininha\nsubiu a tromba d'água.**

**abaixo veio a chuva\ne lavou a aranha.**

**2.1.9 Usando vários argumentos**

Até agora, testamos o comportamento da função print() sem argumentos e com um argumento. Também vale a pena tentar alimentar a função print() com mais de um argumento.

Olhe para a janela do editor. Isso é o que vamos testar agora:

print("A aranha pequenininha" , "subiu" , "a tromba d'água.")

**Console**

**A aranha pequenininha subiu a tromba d'água.**

Há uma chamada de função print(), mas ela contém três argumentos. Todos eles são cadeias de caracteres.

Os argumentos são separados por vírgulas. Nós os cercamos de espaços para torná-los mais visíveis, mas não é realmente necessário, e não faremos mais isso.

Nesse caso, as vírgulas que separam os argumentos desempenham uma função completamente diferente da vírgula dentro da sequência. O primeiro faz parte da sintaxe do Python, enquanto o último deve ser exibido no console.

Se você olhar novamente o código, verá que não há espaços dentro das cadeias.

Execute o código e veja o que acontece.

A pequenina aranha escalou a tromba d'água.

Os espaços, removidos das cadeias, apareceram novamente. Você pode explicar por quê?

Duas conclusões emergem desse exemplo:

* uma função print() chamada com mais de um argumento gera todos eles em uma linha;
* a função print() coloca um espaço entre os argumentos gerados por sua própria iniciativa.

**2.1.10 Argumentos posicionais**

Agora que você sabe um pouco sobre os costumes da função print(), vamos mostrar-lhe como alterá-los.

Você deve ser capaz de prever a saída sem executar o código no editor.

**print("Meu nome é", "Python.")**

**print("Monty Python.")**

**Console**

**Meu nome é Python.**

**Monty Python.**

A forma como estamos passando os argumentos para a função print() é a mais comum em Python e é chamada de forma posicional. Esse nome vem do fato de que o significado do argumento é determinado pela sua posição (por exemplo, o segundo argumento será gerado após o primeiro, e não o contrário).

Execute o código e verifique se a saída corresponde às suas previsões.

**2.1.11 Argumentos de palavra-chave**

O Python oferece outro mecanismo para a transmissão de argumentos, que pode ser útil quando você quiser convencer a função print() a mudar um pouco o comportamento.

Não vamos explicar isso em detalhes agora. Planejamos fazer isso quando falarmos sobre funções. Por enquanto, queremos apenas mostrar como ele funciona. Fique à vontade para usá-lo em seus próprios programas.

O mecanismo é chamado de argumentos de **palavra-chave**. O nome decorre do fato de que o significado desses argumentos é obtido não de sua localização (posição), mas da palavra especial (palavra-chave) usada para identificá-los.

A função print() tem dois argumentos de palavra-chave que você pode usar para seus propósitos. O primeiro é chamado de end.

Na janela do editor, você pode ver um exemplo muito simples de como usar um argumento de palavra-chave.

**print("Meu nome é", "Python.", end=" ")**

**print("Monty Python.")**

**Console**

**Meu nome é Python. Monty Python.**

Para usá-lo, é necessário conhecer algumas regras:

* um argumento de palavra-chave consiste em três elementos: uma palavra-chave que identifica o argumento (end aqui); um sinal de igual (=); e um valor atribuído a esse argumento;
* qualquer argumento de palavra-chave deve ser colocado após o último argumento posicional (isso é muito importante)

Em nosso exemplo, utilizamos o argumento chave end e o definimos como uma sequência contendo um espaço.

Execute o código para ver como ele funciona.

O console agora deve estar mostrando o seguinte texto:

Meu nome é Python. Monty Python.

Como você pode ver, o argumento de palavra-chave end determina os caracteres que a função print() envia para a saída, uma vez que ela atinge o fim de seus argumentos de posição.

O comportamento padrão reflete a situação em que o argumento de palavra-chave end é usado implicitamente da seguinte maneira: **end="\n".**

E agora é hora de tentar algo mais difícil.

Se você olhar com atenção, verá que usamos o argumento end, mas a string atribuída a ele está vazia (não contém nenhum caractere).

O que vai acontecer agora? Execute o programa no editor para descobrir.

print("Meu nome é ", end="")

print("Monty Python.")

**Console**

**Meu nome é Monty Python.**

Como o argumento end foi definido como nada, a função print() também não gera nada, uma vez que seus argumentos posicionais se esgotam.

O console agora deve mostrar o seguinte texto:

Meu nome é Monty Python.

Nota: nenhum caracter de nova linha foi enviado para a saída.

A string atribuída ao argumento da palavra-chave end pode ter qualquer comprimento. Experimente se quiser.

Dissemos anteriormente que a função print() separa seus argumentos em saída com espaços. Esse comportamento também pode ser alterado.

O argumento da palavra -chave que pode fazer isso é nomeado **sep** (como no separador).

Veja o código no editor e execute-o.

**print("Meu", "nome", "é", "Monty", "Python.", sep="-")**

**Console**

**Meu-nome-é-Monty-Python.**

O argumento sep fornece os seguintes resultados:

Meu-nome-é-Monty-Python.

A função print() agora usa um traço, em vez de um espaço, para separar os argumentos de saída.

Nota: o valor do argumento sep também pode ser uma string vazia. Experimente.

Ambos os argumentos de palavra-chave podem ser misturados em uma chamada, como aqui na janela do editor.

**print("Meu", "nome", "é", sep="\_", end="\*")**

**print("Monty", "Python.", sep="\*", end="\*\n")**

**Console**

**Meu\_nome\_é\*Monty\*Python.\***

O exemplo não faz muito sentido, mas apresenta visivelmente as interações entre end e sep.

Você consegue prever a saída?

Execute o código e veja se ele corresponde às suas previsões.

Agora que você entende a função print(), está pronto para pensar em como armazenar e processar dados em Python.

Sem a print(), você não seria capaz de ver nenhum resultado.

**2.1.12 LAB A função print() e seus argumentos**

**Cenário**

Modifique a primeira linha de código no editor, usando as palavras-chave sep e end, para corresponder à saída esperada. Use as duas funções print() no editor.

Não altere nada na segunda invocação de print().

Saída esperada

Programação\*\*\*Essenciais\*\*\*em...Python

**print("Programação","Essenciais","em", sep="\*\*\*", end="...")**

**print("Python")**

**Console**

**Programação\*\*\*Essenciais\*\*\*em...Python**

2.1.13 LAB Formatação de saída

Cenário

Recomendamos que você brinque com o código que escrevemos para você e faça algumas alterações (talvez até destrutivas). Sinta-se livre para modificar qualquer parte do código, mas há uma condição: aprender com seus erros e tirar suas próprias conclusões.

Tente:

* minimizar o número de invocações de função print() inserindo a sequência \n nas strings;
* aumentar a flecha duas vezes (mas manter as proporções)
* duplique a seta, colocando as duas setas lado a lado; observação: uma string pode ser multiplicada usando o seguinte truque: "string" \* 2 produzirá "stringstring" (falaremos sobre isso em breve)
* remova qualquer uma das aspas e analise com cuidado a resposta do Python; preste atenção onde Python vê um erro - este é o lugar onde o erro realmente existe?
* faça o mesmo com alguns parênteses;
* altere qualquer uma das palavras Print em algo diferente, diferindo apenas no caso (por exemplo, Print) - o que acontece agora?
* substitua algumas aspas por apóstrofos; veja o que acontece com cuidado.

**print("Eu gosto \n de andar de\n bicicleta",end=" \*")**

**print("Eu gosto de", "andar de", "bicicleta", sep=" \* \*")**

**print("Eu não sou exímio ciclista", sep=">")**

**print("Quando eu era pequeno", sep=" \* \*", end="<->")**

**print("\*\*\* \*\*\*")**

**print(" \* \*")**

**print(" \* \*")**

**print(" \*\*\*\*\*")**

**Console**

**Eu gosto**

**de andar de**

**bicicleta \*Eu gosto de \* \*andar de \* \*bicicleta**

**Eu não sou exímio ciclista**

**Quando eu era pequeno<->\*\*\* \*\*\***

**\* \***

**\* \***

**\*\*\*\*\***

**2.1.14 RESUMO DA SEÇÃO**

1. A função print() é uma função interna. Ele imprime/envia uma mensagem especificada para a tela/janela do console.

2. As funções incorporadas, ao contrário das funções definidas pelo usuário, estão sempre disponíveis e não precisam ser importadas. O Python 3.8 vem com 69 funções integradas. Você pode encontrar a lista completa fornecida em ordem alfabética na Python Standard Library.

3. Para chamar uma função (este processo é conhecido como chamada de função ou invocação de função), você precisa usar o nome da função seguido de parênteses. Você pode passar argumentos para uma função colocando-os dentro dos parênteses. Você deve separar os argumentos com uma vírgula, por exemplo, print("Olá,", "world!"). Um print() vazio imprime uma linha vazia na tela.

4. Strings no Python são delimitadas por aspas, por exemplo, "eu sou um barbante" (aspas duplas), ou 'eu sou um barbante, demasiado' (aspas simples).

5. Programas de computador são uma coleção de instruções. Uma instrução é um comando para executar uma tarefa específica quando executado, por exemplo, para imprimir uma determinada mensagem na tela.

6. Em Strings do Python a barra invertida (\) é um caracter especial que anuncia que o próximo caracter terá um significado diferente, por exemplo, \n (o caracter de nova linha) inicia uma nova linha de saída.

7. Argumentos posicionais são aqueles cujo significado é ditado por sua posição, por exemplo, o segundo argumento é gerado após o primeiro, o terceiro é gerado após o segundo, etc.

8. Argumentos de palavras-chave são aqueles cujo significado não é ditado por sua localização, mas por uma palavra especial (palavra-chave) usada para identificá-los.

9. Os parâmetros end e sep podem ser utilizadas para formatar a saída do print(). O parâmetro sep especifica o separador entre os argumentos de saída, por exemplo, print("H", "E", "L", "L", "O", sep="-"), enquanto o parâmetro end especifica o que imprimir no final da impressão.

**2.1.15 TESTE DA SEÇÃO**

**Pergunta 1:** Qual é o resultado do programa a seguir?

print("Meu\nnome\né\nBond.", end=" ")

print("James Bond.")

Verificar

Meu

nome

é

Bond. James Bond.

**Pergunta 2:** Qual é a saída do seguinte programa?

print(sep="&", "peixe", "salgadinhos")

Verificar

File "main.py", line 1

    print(sep="&", "peixe", "salgadinhos")

^

SyntaxError: positional argument follows keyword argument

**Pergunta 3:** Qual das seguintes invocações da função print() causará um SyntaxError?

print('Greg\'s book.')

print("'Greg's book.'")

print('"Greg\'s book."')

print("Greg\'s book.")

print('"Greg's book."')

Verificar

A linha 5 gerará SyntaxError, porque o símbolo ' no Greg's book. string requer um caractere de escape.

**2.2 Seção 2 - Literais Python**

Bem-vindo à seção dois, onde falaremos sobre literais em Python.

**2.2.1 Literais - os dados em si**

Agora que você tem um pouco de conhecimento de alguns dos recursos poderosos oferecidos pela função print(), é hora de aprender sobre alguns novos problemas e um novo termo importante - o literal.

Um **literal** são dados cujos valores são determinados pelo próprio literal.

Como esse é um conceito difícil de entender, um bom exemplo pode ser útil.

Veja o seguinte conjunto de dígitos:

123

Você consegue adivinhar qual valor representa? Claro que você pode - são cento e vinte e três.

Mas e quanto a isso:

c

Isso representa algum valor? Talvez. Pode ser o símbolo da velocidade da luz, por exemplo. Também pode ser a constante de integração. Ou até mesmo o comprimento de uma hipotenusa no sentido de um teorema de Pitágoras. Há muitas possibilidades.

Você não pode escolher o caminho certo sem algum conhecimento adicional.

E esta é a pista: 123 é um literal e c não é.

Você usa literais para codificar dados e colocá-los em seu código. Vamos agora mostrar algumas convenções que você deve seguir ao usar o Python.

Vamos começar com uma experiência simples: dê uma olhada no snippet no editor.

print("2")

print(2)

**Console**

**2**

**2**

A primeira linha parece familiar. A segunda parece estar errada devido à visível falta de aspas.

Tente executá-lo.

Se tudo correr bem, agora você verá duas linhas idênticas.

O que aconteceu? O que isso significa?

Neste exemplo, você encontrará dois tipos diferentes de literais:

* uma string, que você já conhece,
* e um número inteiro, algo completamente novo.

A função print() os apresenta exatamente da mesma maneira ‒ este exemplo é óbvio, pois sua representação legível por humanos também é a mesma. Internamente, na memória do computador, esses dois valores são armazenados de formas completamente diferentes ‒ a string existe apenas como uma string ‒ uma série de letras.

O número é convertido em representação de máquina (um conjunto de bits). A função print() é capaz de mostrar ambos em um formato legível para humanos.

Agora vamos passar algum tempo discutindo literais numéricos e sua vida interna.

**2.2.2 Inteiros**

Você já deve saber um pouco sobre como os computadores realizam cálculos em números. Talvez você já tenha ouvido falar do **sistema binário** e saiba que é o sistema que os computadores usam para armazenar números e que esses computadores podem realizar qualquer operação neles.

Não exploraremos as complexidades dos sistemas numéricos de posição aqui, mas vamos dizer que os números manipulados por computadores modernos são de dois tipos:

* **inteiros**, ou seja, aqueles que são desprovidos da parte fracionária;
* e números de **ponto flutuante** (ou simplesmente **float**), que contêm (ou são capazes de conter) a parte fracionária.

Essa definição não é totalmente precisa, mas é suficiente por enquanto. A distinção é muito importante, e a fronteira entre esses dois tipos de números é muito rigorosa. Esses dois tipos de números diferem significativamente na forma como são armazenados na memória do computador e na faixa de valores aceitáveis.

A característica do valor numérico que determina seu tipo, intervalo e aplicação é chamada de **tipo**.

Se você codificar um literal e o colocar dentro do código Python, a forma do literal determinará a representação (tipo) que o Python usará para armazená-lo na memória.

Por enquanto, vamos deixar os números de ponto flutuante de lado (vamos voltar para eles em breve) e considerar a questão de como o Python reconhece números inteiros.

O processo é quase como se você os escrevesse com um lápis no papel - é simplesmente uma sequência de dígitos que compõe o número. Mas há uma reserva - você não deve inserir caracteres que não sejam dígitos dentro do número.

Tomemos, por exemplo, o número onze milhões cento e onze mil cento e onze. Se você pegasse um lápis na mão agora, escreveria o número assim: 11,111,111, ou assim: 11.111.111, ou até mesmo assim: 11 111 111.

É claro que essa disposição facilita a leitura, especialmente quando o número consiste em muitos dígitos. No entanto, o Python não aceita coisas como essas. **É proibido**. O que o Python permite, porém, é o uso de sublinhados em literais numéricos.\*

Portanto, você pode escrever este número da seguinte forma (**correto**): **11111111** ou desta forma: **11\_111\_111**.

**Observação**: \*O Python 3.6 introduziu sublinhados em literais numéricos, permitindo a colocação de sublinhamentos únicos entre dígitos e especificadores de base para melhorar a legibilidade. Esse recurso não está disponível em versões mais antigas do Python.

E como codificamos números negativos em Python? Como de costume - adicionando um sinal de menos. Você pode escrever: -11111111 ou -11\_111\_111.

Os números positivos não precisam ser precedidos pelo sinal de mais, mas é permitido, se você quiser fazer isso. As seguintes linhas descrevem o mesmo número: +11111111 e 11111111.

**Números octais e hexadecimais**

Há duas convenções adicionais em Python que são desconhecidas para o mundo da matemática. A primeira nos permite usar números em uma representação octal.

Se um número inteiro for precedido por um prefixo **0O** ou **0o** (**zero-o**), ele será tratado como um **valor octal**. Isso significa que o número deve conter dígitos retirados apenas do **intervalo [0..7].**

0o123 é um número octal com um valor (decimal) igual a 83.

A função print() faz a conversão automaticamente. Tente isto:

print(0o123)

**Console**

**83**

A segunda convenção nos permite usar números hexadecimais. Esses números devem ser precedidos pelo prefixo 0x ou **0X (zero-x**).

0x123 é um número hexadecimal com um valor (decimal) igual a 291. A função print() também pode gerenciar esses valores. Tente isto:

print(0x123)

**Console**

**291**

**2.2.3 Floats**

Agora é hora de falar sobre outro tipo, que é projetado para representar e armazenar os números que (como diria um matemático) têm uma fração decimal não vazia.

Eles são os números que têm (ou podem ter) uma parte fracionária após o ponto decimal e, embora essa definição seja muito ruim, certamente é suficiente para o que queremos discutir.

Sempre que usamos um termo como dois e meio ou menos o ponto zero quatro, pensamos em números que o computador considera números de ponto flutuante:

2.5

-0.4

**Nota:** dois e meio parecem normais quando você o escreve em um programa, embora se o seu idioma nativo preferir usar vírgula em vez de um ponto no número, você deve garantir que seu número não contenha nenhuma vírgula.

O Python não aceitará isso, ou (em casos muito raros, mas possíveis) pode interpretar mal suas intenções, pois a própria vírgula tem seu próprio significado reservado em Python.

Se você quiser usar apenas um valor de dois e meio, escreva-o como mostrado acima. Observe mais uma vez: há um ponto entre 2 e 5, não uma vírgula.

Como você provavelmente pode imaginar, o valor de zero ponto quatro poderia ser escrito em Python como:

0.4

Mas não se esqueça desta regra simples - você pode omitir zero quando for o único dígito antes ou depois da vírgula.

Em essência, você pode escrever o valor 0.4 como:

.4

Por exemplo: o valor de 4.0 pode ser escrito como:

4.

Isso não mudará seu tipo nem seu valor.

**Int e. flutua**

O ponto decimal é essencial para reconhecer números de ponto flutuante no Python.

Veja estes dois números:

4

4.0

Você pode pensar que eles são exatamente iguais, mas Python os vê de uma maneira completamente diferente.

* 4 é um número inteiro, enquanto 4.0 é um número de ponto flutuante.
* **O ponto é o que faz um ponto flutuante.**

Por outro lado, não são apenas os pontos que fazem um float. Você também pode usar a letra **e**.

Quando você quiser usar números muito grandes ou muito pequenos, use notação científica.

Veja, por exemplo, a velocidade da luz, expressa em metros por segundo. Escrito diretamente, ficaria assim: **300000000**.

Para evitar escrever tantos zeros, os livros de física usam uma forma abreviada, que você provavelmente já viu: **3 x 108**.

Diz: **Três vezes dez à potência de oito**.

Em Python, o mesmo efeito é obtido de uma maneira ligeiramente diferente - veja:

**3E8**

A letra **E** (você também pode usar a letra minúscula e - vem da palavra expoente) é um registro conciso da frase vezes dez à potência de.

**Nota:**

* o expoente (o valor após o E) tem que ser um número inteiro;
* a base (o valor na frente do E) pode ser um número inteiro ou um valor

flutuante.

**Codificação de flutuantes**

Vamos ver como essa convenção é usada para registrar números muito pequenos (no sentido de seu valor absoluto, que é próximo de zero).

Uma constante física chamada constante de Planck (e indicada como h), de acordo com os livros didáticos, tem o valor de: **6,62607 x 10-34.**

Se você gostaria de usá-lo em um programa, escreva desta forma:

6.62607E-34

**Nota:** o fato de você ter escolhido uma das formas possíveis de codificação de valores flutuantes não significa que o Python a apresentará da mesma forma.

Às vezes, o Python pode escolher uma notação diferente da sua.

Por exemplo, digamos que você decidiu usar o seguinte literal de flutuação:

**0.0000000000000000000001**

Quando você executa esse literal pelo Python:

**print(0.0000000000000000000001)**

Output

esse é o resultado:

Output

**1e-22**

**O Python sempre escolhe a forma mais econômica de apresentação do número, e você deve levar isso em consideração ao criar literais**.

**2.2.4 Strings**

Strings são usadas quando você precisa processar texto (como nomes de todos os tipos, endereços, romances, etc.), não números.

Você já sabe um pouco sobre eles, por exemplo, que **strings precisam de aspas** assim como floats precisam de pontos.

Esta é uma string muito típica: "Eu sou uma string."

No entanto, há um porém. O problema é como codificar uma citação dentro de uma string que já está delimitada por aspas.

Vamos supor que queremos imprimir uma mensagem bem simples dizendo:

**Eu gosto de "Monty Python"**

Como fazemos isso sem gerar um erro? Há duas soluções possíveis.

* A primeira é baseada no conceito que já conhecemos do caractere de escape, que você deve lembrar que é representado pela barra invertida. A barra invertida também pode escapar das aspas. Uma citação precedida por uma barra invertida muda seu significado ‒ não é um delimitador, mas apenas uma citação. Isso funcionará como pretendido:

**print("Eu gosto \"Monty Python\"")**

**Nota:** há duas aspas com escape dentro da string - você consegue ver as duas?

* A segunda solução pode ser um pouco surpreendente. O Python pode usar um apóstrofo em vez de uma aspas. Qualquer um desses caracteres pode delimitar cadeias de caracteres, mas você deve ser coerente.

Se você abrir uma sequência com aspas, será necessário fechá-la com aspas.

Se você iniciar uma sequência de caracteres com um apóstrofo, precisará terminá-la com um apóstrofo.

Este exemplo também funcionará:

**print('Eu gosto "Monty Python"')**

**Nota**: você não precisa de usar escape aqui.

**Codificando strings**

Agora, a próxima pergunta é: como você incorpora um apóstrofo em uma string colocada entre apóstrofos?

Você já deve saber a resposta ou, para ser mais preciso, duas respostas possíveis.

Tente imprimir uma string contendo a seguinte mensagem:

Eu sou Monty Python.

print('I\'m Monty Python.')

print("I'm Monty Python.")

**Console**

**I'm Monty Python.**

**I'm Monty Python.**

Você sabe como fazê-lo? Clique na Check abaixo para ver se você estava certo:

Como você pode ver, a barra invertida é uma ferramenta muito poderosa - ela pode escapar não apenas de aspas, mas também de apóstrofes.

Já mostramos isso, mas queremos enfatizar mais uma vez esse fenômeno: uma string pode estar vazia - pode não conter caracteres.

Uma string vazia ainda permanece uma string:

**''**

**""**

**2.2.5 Valores booleanos**

Para concluir com os literais do Python, há dois adicionais.

Elas não são tão óbvias quanto as anteriores, pois são usadas para representar um valor muito abstrato - a **veracidade** (***True e False***)

Cada vez que você pergunta ao Python se um número é maior que outro, a pergunta resulta na criação de alguns dados específicos - um valor booleano.

O nome vem de George Boole (1815-1864), autor da obra fundamental, The Laws of Thought, que contém a definição de **álgebra booleana** ‒ uma parte da álgebra que faz uso de apenas dois valores distintos: **True e False**, denotados como 1 e 0.

Um programador cria um programa, e o programa faz perguntas. Python executa o programa e fornece as respostas. O programa deve ser capaz de reagir de acordo com as respostas recebidas.

Felizmente, os computadores sabem apenas dois tipos de respostas:

* Sim, isso é verdade;
* Não, isso é falso.

Você nunca receberá uma resposta como: Não sei ou Provavelmente sim, mas não tenho certeza.

Python, então, é um réptil **binário**.

Esses dois valores booleanos têm denotações estritas em Python:

True

False

Você não pode mudar nada. É necessário aceitar esses símbolos como eles são, inclusive distinção entre maiúsculas e minúsculas.

Desafio: Qual será a saída do trecho de código a seguir?

Execute o código no editor para verificar. Você pode explicar o resultado?

*print*(True > False)

*print*(True < False)

**Console**

**True**

**False**

**2.2.6 LAB Literais Python - strings**

Cenário

Escreva um código de uma linha usando a função de print(), bem como os caracteres de nova linha e de escape, para corresponder ao resultado esperado gerado em três linhas.

Saída esperada

"Eu sou"

""aprendizado""

"""Python"""

***print("\"Eu sou\"")***

***print("\"\"Aprendizado\"\"")***

***print("\"\"\"Python\"\"\"")***

**Console**

**"Eu sou"**

**""Aprendizado""**

**"""Python"""**

Ou

***print("\"Eu sou\"\n\"\"aprendizado\"\"\n\"\"\"Python\"\"\"")***

**Console**

**"Eu sou"**

**""aprendizado""**

**"""Python"""**

**2.2.7 RESUMO DA SEÇÃO**

1. **Literais** são notações para representar alguns valores fixos no código. Python tem vários tipos de literais ]

- por exemplo, um literal pode ser um número (literais numéricos, por exemplo, **123**) ou uma string (literais de string, por exemplo, **"Eu sou um literal".).**

2. O **sistema binário** é um sistema de números que emprega 2 como base. Portanto, um número binário é composto de 0s e 1s apenas, por exemplo, 1010 é 10 em decimal.

*Os sistemas de numeração octal e hexadecimal, da mesma forma, empregam 8 e 16 como suas bases respectivamente. O sistema hexadecimal usa os números decimais e seis letras extras.*

3. **Inteiros** (ou simplesmente **ints**) são um dos tipos numéricos compatíveis com o Python. São números escritos sem um componente fracionário, por exemplo, 256 ou -1 (números inteiros negativos).

4. **Ponto flutuante** (ou simplesmente **float**) são outro dos tipos numéricos compatíveis com o Python. São números que contêm (ou são capazes de conter) um componente fracionário, por exemplo, 1.27.

5. Para codificar um apóstrofo ou uma aspas dentro de uma string, você pode usar o caracter de escape, por exemplo, **'I\'m happy.'** ou abrir e fechar a string usando um conjunto oposto de símbolos aos que você deseja codificar, por exemplo, **"I'm happy."** para codificar um apóstrofo e 'Ele disse **"Python", não "typhoon"'** para codificar uma aspas (dupla).

6. **Valores booleanos** são os dois objetos constantes **True** e **False** usados para representar valores de verdade (em contextos numéricos, **1** é **True**, enquanto **0** é **False**.

**Extra**

Há mais um literal especial usado em Python: o literal **None**. Esse literal é um objeto NoneType e é usado para representar a ausência de um valor. Vamos contar mais sobre isso em breve.

**2.2.8 TESTE DA SEÇÃO**

Pergunta 1: Que tipos de literais são os dois exemplos a seguir?

"Olá ", "007"

Verificar

***Respostas: Ambos são Strings (strings literais)***

Pergunta 2: Que tipos de literais são os quatro exemplos a seguir?

"1.5", 2.0, 528, False

Verificar

***Resposta: String, Float, Int e Boolean.***

***Ou***

***O primeiro é uma string, o segundo é um literal numérico (um float), o terceiro é um literal numérico (um inteiro) e o quarto é um literal booleano.***

Pergunta 3: Qual é o valor decimal do seguinte número binário?

1011

Verificar

***Resposta: 11***

***Porque: (2\*\*0) + (2\*\*1) + (2\*\*3) = 11***

***0 binário tem apenas dois algarismos. A regra de formação é a mesma, cada dígito é o seu valor multiplicado por 2, elevado ao valor da sua posição no número total menos um***

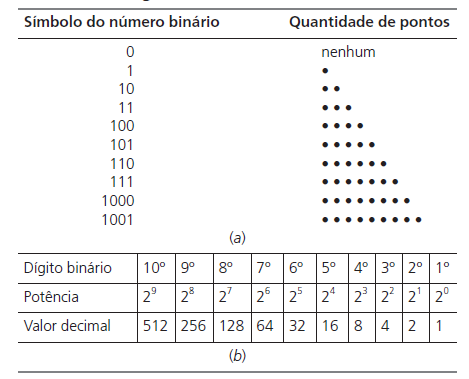
***Posição 3 Posição 2 Posição 1 Posição 0***

***1 0 1 1***

**X 2 ³ x2² x2¹ x2°**

**8** + **0**  +  **2** +  **1** = **11**

**Outras informações sobre finário**



***Sistema Binário par Decimal***

<https://blog.betrybe.com/tecnologia/codigo-binario/#:~:text=O%20número%201%2C%20por%20exemplo%2C%20corresponde%20ao%20binário%2000000001>

**2.3 Seção 3 - Operadores - ferramentas de manipulação de dados**

Bem-vindo à seção três, na qual falaremos sobre operadores Python.

**2.3.1 Python como uma calculadora**

Agora, vamos mostrar a você um lado completamente novo da função print(). Você já sabe que a função é capaz de mostrar os valores dos literais passados a ela por argumentos.

Na verdade, ele pode fazer algo mais. Confira o trecho:

*print*(2+2)

Fazendo um código utilizando calculadores

Você deve ver os resultados conforme operação aplicada. Sinta-se à vontade para experimentar outros operadores.

***print(2+2) # SOMA- ADIÇÃO***

***print(2\*3) # MULTIPLICAÇÃO***

***print(8/4) # DIVSÃO***

***print(2-1) # SUBTRAÇÃO***

***print(2\*10/2+5) # NA SEQUENCIA MULTIPL DIV E SOMA***

***print((2\*50)/(3+7)) #RESOLVENDO ENTRE PARENTESES***

***print(5\*\*2) # EXPOENTE***

**Console**

**4**

**6**

**2.0**

**1**

**15.0**

**10.0**

**25**

Sem levar isso a sério, você acabou de descobrir que o Python pode ser usado como uma calculadora. Não é muito prático, e definitivamente não é de bolso, mas mesmo assim uma calculadora.

Levando isso mais a sério, agora estamos entrando na província de **operadores** e **expressões**.

2.3.2 Operadores básicos

Um operador é um símbolo da linguagem de programação, capaz de operar com os valores.

Por exemplo, assim como na aritmética, o sinal de + (mais) é o operador que é capaz de adicionar dois números, dando o resultado da adição.

Nem todos os operadores Python são tão óbvios quanto o sinal de mais, então vamos ver alguns dos operadores disponíveis no Python, e vamos explicar quais regras regem o uso e como interpretar as operações que realizam.

Vamos começar com os operadores que estão associados às operações aritméticas mais amplamente reconhecidas:

**+**

**-**

**\***

**/**

**//**

**%**

**\*\***

A ordem de aparência não é acidental. Falaremos mais sobre isso depois de passar por todos eles.

Lembre-se: os dados e os operadores, quando conectados, formam expressões. A expressão mais simples é um literal em si.

A ordem de aparência não é acidental. Falaremos mais sobre isso depois de passar por todos eles.

Lembre-se: os dados e os operadores, quando conectados, formam expressões. A expressão mais simples é um literal em si.

**Exponenciação**

Veja o exemplo a seguir no editor:

***print(2 \*\* 3)***

***print(2 \*\* 3.)***

***print(2. \*\* 3)***

***print(2. \*\* 3.)***

**Console**

**8**

**8.0**

**8.0**

**8.0**

**Nota:** Recebemos os asteriscos duplos com espaços em nossos exemplos. Não é obrigatório, mas melhora a legibilidade do código.

Os exemplos mostram uma característica muito importante de praticamente todos os operadores numéricos do Python.

Execute o código e observe cuidadosamente os resultados que produz. Você pode ver alguma regularidade aqui?

**Lembre-se:** É possível formular as seguintes regras com base neste resultado:

* quando os dois \*\* argumentos são inteiros, o resultado também é um número inteiro;
* Quando pelo menos um argumento \*\* é um float, o resultado também é um float.

Esta é uma distinção importante para se lembrar.

**Multiplicação**

Um sinal de \* (asterisco) é um operador de multiplicação.

Execute o código abaixo e verifique se nossa regra de número inteiro vs. flutuante ainda está funcionando.

**print(2 \* 3)**

**print(2 \* 3.)**

**print(2. \* 3)**

**print(2. \* 3.)**

**Console**

**6**

**6.0**

**6.0**

**6.0**

**Divisão**

Um sinal / (barra) é um operador de divisão.

O valor na frente da barra é um dividendo, o valor atrás da barra, um divisor.

Execute o código abaixo e analise os resultados.

**print(6 / 3)**

**print(6 / 3.)**

**print(6. / 3)**

**print(6. / 3.)**

**Console**

**2.0**

**2.0**

**2.0**

**2.0**

Você verá que há uma exceção à regra.

O **resultado produzido pelo operador de divisão é sempre um float**, independentemente de parecer ou não ser um flutuante à primeira vista: 1 / 2, ou se parece com um inteiro puro: 2 / 1.

Isso é um problema? Sim, o limite continua igual. Às vezes acontece que você realmente precisa de uma divisão que forneça um valor inteiro, não um valor flutuante.

Felizmente, o Python pode ajudá-lo com isso.

**Divisão de número inteiro (divisão arredondada)**

Um sinal // (barra dupla) é um operador de divisão inteira. Difere do padrão / operador em dois detalhes:

* seu resultado não possui a parte fracionária ‒ está ausente (para inteiros), ou é sempre igual a zero (para flutuantes); isso significa que os resultados são sempre arredondados;
* ele está de acordo com a regra integer vs. float.

*Execute o exemplo abaixo e veja os resultados:*

***print(6 // 3)***

***print(6 // 3.)***

***print(6. // 3)***

***print(6. // 3.)***

**Console**

**2**

**2.0**

**2.0**

**2.0**

Como você pode ver, a divisão de inteiro por inteiro dá um resultado inteiro. Todos os outros casos produzem floats.

Vamos fazer alguns testes mais avançados.

Observe o trecho a seguir:

**print(6 // 4)**

**print(6. // 4)**

**Console**

**1**

**1.0**

Imagine que usamos / em vez de // - você poderia prever os resultados?

Sim, seria 1.5 em ambos os casos. Isso é claro.

Mas que resultados devemos esperar com a // divisão?

Execute o código e veja por si mesmo.

O que obtemos são dois uns - um inteiro e um flutuador.

O resultado da divisão do número inteiro é sempre arredondado para o valor inteiro mais próximo que é menor que o resultado real (não arredondado).

Isso é muito importante: o arredondamento sempre vai para o número inteiro menor.

Observe o código abaixo e tente prever os resultados mais uma vez:

**print(-6 // 4)**

**print(6. // -4)**

**Console**

**-2**

**-2.0**

Nota: alguns valores são negativos. Obviamente, isso afetará o resultado. Mas como?

O resultado é dois pares negativos. O resultado real (não arredondado) é -1.5 em ambos os casos. No entanto, os resultados são sujeitos a arredondamento. **O arredondamento vai em direção ao valor inteiro menor**, e o valor inteiro menor é -2, portanto: -2 e -2.0.

**Observação** A divisão inteira também pode ser chamada de **divisão de piso.** Você definitivamente vai encontrar esse termo no futuro.

**Restante (módulo)**

O próximo operador é bastante peculiar, porque não tem equivalente entre os operadores aritméticos tradicionais. Sua representação gráfica em Python é o sinal de **% (percentual)**, que pode parecer um pouco confuso.

Tente pensar nisso como uma barra (operador de divisão) acompanhada de dois pequenos círculos engraçados.

**O resultado do operador é o restante após a divisão do número inteiro**.

Em outras palavras, é o valor que falta depois de dividir um valor por outro para produzir um quociente inteiro.

Nota: o operador às vezes é chamado de módulo em outras linguagens de programação.

Dê uma olhada no snippet - tente prever o resultado e execute-o:

**print(14 % 4)**

**Console**

**2**

Como você pode ver, o resultado é dois. É por isso que:

14 // 4 dá 3 → este é o quociente inteiro;

3 \* 4 dá 12 → como resultado da multiplicação de quociente e divisor;

14 - 12 dá **2** → este é o restante.

Este exemplo é um pouco mais complicado:

print(12 % 4.5)

Qual é o resultado?

**3.0** – não 3 mas 3.0.

A regra ainda funciona:

12 // 4.5 dá 2.0,

2.0 \* 4.5 dá 9.0,

12 - 9.0 dá **3.0**.

**Como não dividir (% ou /)**

Como você provavelmente sabe, a divisão por zero não funciona.

Não tente:

* realizar uma divisão por zero;
* realizar uma divisão inteira por zero;
* encontre o resto de uma divisão por zero.

**Adição**

O operador de adição é o sinal de + (mais), que está totalmente de acordo com os padrões matemáticos.

Novamente, dê uma olhada no snippet do programa abaixo:

**print(-4 + 4)**

**print(-4. + 8)**

**Console**

**0**

**4.0**

O resultado não deve ser surpreendente. Execute o código para verificá-lo.

**O operador de subtração, os operadores unários e binários**

O operador de subtração é obviamente o sinal - (menos), embora você deva notar que esse operador também tem outro significado -ele pode alterar o sinal de um número.

Esta é uma grande oportunidade para apresentar uma distinção muito importante entre operadores unários e binários.

Ao subtrair aplicações, o operador menos espera dois argumentos: o esquerdo (**um minuendo em termos aritméticos**) e o direito (**um subtraendo**).

Por esta razão, o operador de subtração é considerado um dos operadores binários, assim como os operadores de adição, multiplicação e divisão.

Mas o operador menos pode ser usado de uma forma diferente (unária) ‒ dê uma olhada na última linha do trecho abaixo:

print(-4 - 4)

print(4. - 8)

print(-1.1)

**Console**

**-8**

**-4.0**

**-1.1**

A propósito: também há um operador + unário. Você pode usá-lo assim:

*print*(+2)

O operador preserva o sinal de seu único argumento - o correto.

Embora essa construção seja sintaticamente correta, usá-la não faz muito sentido e seria difícil encontrar uma boa lógica para isso.

Dê uma olhada no snippet acima - você consegue adivinhar o resultado?

**2.3.3 Operadores e suas prioridades**

Até agora, tratamos cada operador como se não tivesse nenhuma conexão com os outros. Obviamente, uma situação tão simples e ideal é uma raridade na programação real.

Além disso, muitas vezes você encontrará mais de um operador em uma expressão e, em seguida, as coisas não serão mais tão simples.

Considere a seguinte expressão:

2 + 3 \* 5

Você provavelmente se lembra da escola que as multiplicações precedem as adições.

Você certamente se lembra que primeiro deve multiplicar 3 por 5 e, mantendo o 15 na memória, adicione-o a 2, obtendo assim o resultado de 17.

O fenômeno que faz com que alguns operadores ajam antes de outros é conhecido como a **hierarquia de prioridades**.

O Python define com precisão as prioridades de todos os operadores e assume que os operadores de prioridade mais alta executam suas operações antes dos operadores de prioridade mais baixa.

Então, se você sabe que \* tem uma prioridade mais alta que +, o cálculo do resultado final deve ser óbvio.

**Operadores e suas ligações**

A ligação do operador determina a ordem das computações executadas por alguns operadores com igual prioridade, colocados lado a lado em uma expressão.

A maioria dos operadores do Python tem ligação do lado esquerdo, o que significa que o cálculo da expressão é realizado da esquerda para a direita.

Este exemplo simples vai mostrar como ele funciona. Dê uma olhada:

print(9 % 6 % 2)

**Console**

**1**

Há duas maneiras possíveis de avaliar essa expressão:

* da esquerda para a direita: primeiro 9 % 6 dá 3, e então 3 % 2 dá 1;
* da direita para a esquerda: primeiro 6 % 2 dá 0 e depois 9 % 0 causa um erro fatal.

Execute o exemplo e veja o que você obtém.

O resultado deve ser 1. Este operador tem ligação do lado esquerdo. Mas há uma exceção interessante.

Repita o experimento, mas agora com exponenciação.

Use este trecho de código:

**print(2 \*\* 2 \*\* 3)**

**Console**

**256**

Os dois resultados possíveis são:

2 \*\* 2 → 4; 4 \*\* 3 → 64

2 \*\* 3 → 8; 2 \*\* 8 → 256

Execute o código. O que você vê?

O resultado mostra claramente que o operador de exponenciação usa a associação do lado direito.

Isso tem um efeito interessante. Se o operador de exponenciação usar a associação pelo lado direito, você consegue adivinhar a saída do seguinte fragmento?

print(-3 \*\* 2) print(-2 \*\* 3) print(-(3 \*\* 2))

**-9**

**-8**

**-9**

**Lista de prioridades**

Já que você é novo nos operadores Python, não queremos apresentar a lista completa de prioridades do operador agora.

Em vez disso, mostraremos um formulário truncado e o expandiremos de forma consistente ao apresentarmos novos operadores. Veja a tabela abaixo:

|  |  |  |
| --- | --- | --- |
| **Prioridade** | **Operadora** |  |
| 1 | **\*\*** |  |
| 2 | **+, - (observação: operadores unários localizados ao lado direito do operador de potência se vinculam mais fortemente))** | unário |
| 3 | **\*, /, //, %** |  |
| 4 | **+, -** | ⁪binário |

**Nota:** enumeramos os operadores em ordem das prioridades mais altas (1) às mais baixas (4).

Tente trabalhar com a seguinte expressão:

print(2 \* 3 % 5)

Ambos os operadores (\* e %) têm a mesma prioridade, portanto, o resultado pode ser adivinhado apenas quando você sabe a direção da encadernação. O que você acha? Qual é o resultado?

Resposta: 1 (Restas 1 que corresponde a 2 \* 3 = 6 % 5 = 1 e sobre 1)

**Operadores e parênteses**

Claro, você sempre pode usar parênteses, o que pode alterar a ordem natural de um cálculo.

De acordo com as regras aritméticas, **subexpressões entre parênteses são sempre calculadas primeiro.**

Você pode usar quantos parênteses precisar, e eles geralmente são usados para melhorar a legibilidade de uma expressão, mesmo que não alterem a ordem das operações.

Veja um exemplo de expressão com vários parênteses:

Tente calcular o valor impresso no console. Qual é o resultado da função print()?

print((5 \* ((25 % 13) + 100) / (2 \* 13)) // 2)

**Console**

**10.0**

**2.3.4 RESUMO DA SEÇÃO**

**Pontos principais**

1. Uma **expressão** é uma **combinação de valores** (ou variáveis, operadores, chamadas para funções - você aprenderá sobre eles em breve) que resulta em um determinado valor, por exemplo, 1 + 2.

2. **Operadores** são símbolos especiais ou palavras-chave que são capazes de **operar nos valores e realizar operações** (matemáticas), por exemplo, o operador \* multiplica dois valores: x \* y.

3. **Operadores aritméticos** em Python:

+ (adição), - (subtração),

\* (multiplicação),

/ (divisão clássica ‒ sempre retorna um ponto flutuante),

% (módulo ‒ divide o operando esquerdo pelo operando direito e retorna o restante da operação, por exemplo , 5 % 2 = 1),

\*\* (exponenciação ‒ operando esquerdo elevado à potência do operando direito, por exemplo, 2 \*\* 3 = 2 \* 2 \* 2 = 8),

// (floor/divisão inteira ‒ retorna um número resultante da divisão, mas arredondado para o número inteiro mais próximo, por exemplo, 3 // 2.0 = 1.0)

4. Um **operador unário** é um **operador com apenas um operando,** por exemplo, -1 ou +3.

5. Um **operador binário** é um **operador com dois operandos**, por exemplo, 4 + 5 ou 12 % 5.

6. Alguns operadores agem antes de outros - a **hierarquia** de prioridades:

**o operador \*\*** (exponenciação) tem a prioridade mais alta;

então **o unário + e -** (Nota: um operador unário à direita do operador de exponenciação se liga mais fortemente, por exemplo, 4 \*\* -1 é igual a 0.25)

então: \*, /, e %,

e, por fim, a prioridade mais baixa: binário + e -.

7. **Subexpressões entre parênteses** são sempre calculadas primeiro, por exemplo, 15 - 1 \* (5 \* (1 + 2)) = 0.

8. O **operador de exponenciação** usa a associação do lado direito, por exemplo, 2 \*\* 2 \*\* 3 = 256.



**2.3.5 TESTE DA SEÇÃO**

Pergunta 1: Qual é a saída esperada do trecho a seguir?

print((2 \*\* 4), (2 \* 4.), (2 \* 4))

Verificar

**Console**

**16 8.0 8**

Pergunta2: Qual é a saída do trecho a seguir?

print((-2 / 4), (2 / 4), (2 // 4), (-2 // 4))

Verificar

**Console**

**-0.5 0.5 0 -1**

Pergunta 3: Qual é a saída esperada do trecho a seguir?

print((2 % -4), (2 % 4), (2 \*\* 3 \*\* 2))

Verificar

**Console**

**-2 2 512**

**2.4 Seção 4 - Variáveis**

Bem-vindo à seção quatro! Esta parte do curso se concentra nas variáveis - vamos aprender o que são, como usá-las e quais são as regras que as regem. Vamos lá?

**2.4.1 Variáveis - caixas em forma de dados**

Parece bastante óbvio que o Python deva permitir que você codifique literais portando valores de número e texto.

Você já sabe que pode fazer algumas operações aritméticas com esses números: Adicionar, subtrair etc. Você fará isso muitas vezes.

Mas é uma pergunta normal perguntar como armazenar os resultados dessas operações, para usá-los em outras operações e assim por diante.

Como você salva os resultados intermediários e os usa novamente para produzir os subsequentes?

Python irá ajudá-lo com isso. Ele oferece "caixas" especiais (ou "contêineres", como podemos chamá-los) para essa finalidade, e essas caixas são chamadas de **variáveis -** o próprio nome sugere que o conteúdo desses contêineres pode ser variado (quase) de qualquer forma.

O que todas as variáveis Python têm?

* um nome;
* um valor (o conteúdo do contêiner)

Vamos começar com os problemas relacionados ao nome de uma variável.

As variáveis não aparecem em um programa automaticamente. Como desenvolvedor, você deve decidir quantas variáveis e quais usar em seus programas.

Você também deve nomeá-los.

**2.4.2 Nomes de variáveis**

Se quiser dar um nome a uma variável, você deve seguir algumas regras estritas:

* o nome da variável deve ser composto de letras maiúsculas ou minúsculas, dígitos e o caractere \_ (sublinhado)
* o nome da variável deve começar com uma letra;
* o caractere de sublinhado é uma letra;
* as letras maiúsculas e minúsculas são tratadas como diferentes (um pouco diferente do que no mundo real - Alice e ALICE são os mesmos nomes, mas em Python são dois nomes de variáveis diferentes e, consequentemente, duas variáveis diferentes);
* o nome da variável não deve ser nenhuma das palavras reservadas do Python (as palavras-chave - explicaremos mais sobre isso em breve).

Observe que as mesmas restrições se aplicam a nomes de função.

O Python não impõe restrições ao comprimento dos nomes de variáveis, mas isso não significa que um nome de variável longo seja sempre melhor do que um nome curto.

**Aqui estão alguns nomes de variáveis corretos, mas nem sempre convenientes:**

MyVariable

i

l

t34

Exchange\_Rate

counter

days\_to\_christmas

TheNameIsTooLongAndHardlyReadable

\_

**Esses nomes de variáveis também estão corretos:**

Adiós\_Señora

sûr\_la\_mer

Einbahnstraße

переменная.

O Python permite que você use não apenas letras latinas, mas também caracteres específicos de idiomas que usam outros alfabetos.

**E agora, alguns nomes incorretos:**

**10t** (não começa com uma letra)

**!important** (não começa com uma letra)

**exchange rate** (contém um espaço).

Observação

O PEP 8 - Guia de Estilo para Código Python (<https://www.python.org/dev/peps/pep-0008/>) recomenda a seguinte convenção de nomenclatura para variáveis e funções em Python:

* os nomes de variáveis devem estar em letras minúsculas, com palavras separadas por sublinhados para melhorar a legibilidade (por exemplo: var, my\_variable)
* nomes de funções seguem a mesma convenção que nomes de variáveis (por exemplo: fun, my\_function)
* também é possível usar casos mistos (por exemplo, myVariable), mas apenas em contextos onde esse já é o estilo predominante, para manter a compatibilidade com a convenção adotada.

**Palavras-chave**

Dê uma olhada na lista de palavras que desempenham um papel muito especial em todos os programas Python.

**['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']**

Eles são chamados de palavras-chave ou (mais precisamente) palavras-chave reservadas. Eles são reservados porque você não deve usá-los como nomes: nem para suas variáveis, nem funções, nem quaisquer outras entidades nomeadas que você deseja criar.

O significado da palavra reservada é predefinido e não deve ser alterado de forma alguma.

Felizmente, devido ao fato de que o Python faz distinção entre maiúsculas e minúsculas, você pode modificar qualquer uma dessas palavras alterando a letra de qualquer letra, criando assim uma nova palavra, que não é mais reservada.

Por exemplo - você **não pode nomear** sua variável assim:

**import**

Você não deve ter uma variável chamada dessa forma - ela é proibida. Mas você pode fazer isso:

**import**

Essas palavras podem ser um mistério para você agora, mas você aprenderá em breve o significado delas.

**2.4.3 Como criar variáveis**

**O que você pode colocar dentro de uma variável? Qualquer coisa.**

Você pode usar uma variável para armazenar qualquer valor de qualquer um dos tipos já apresentados, e muitos outros que ainda não mostramos.

O valor de uma variável é o que você coloca nela. Pode variar com a frequência desejada ou desejada. Pode ser um inteiro um momento, e um momento depois, e, se tornar uma string.

Vamos falar agora de duas coisas importantes - como as variáveis são criadas e como colocar valores dentro delas (ou melhor, como dar ou passar valores para elas).

**Lembre-se**

**Uma variável passa a existir como resultado da atribuição de um valor a ela.** Ao contrário de outros idiomas, você não precisa declará-lo de nenhuma maneira especial.

Se você atribuir qualquer valor a uma variável inexistente, a variável será criada automaticamente. Você não precisa fazer mais nada.

A criação (ou seja, sua **sintaxe**) é extremamente simples: basta usar o nome da variável desejada, depois o sinal de igual (=) e o valor que deseja colocar na variável.

**Como comentar em Python várias linhas?** Em Python o "comentários em várias linhas" é iniciado com e fechado com a sequência de **3 aspas dupla '"""'**.

Dê uma olhada no snippet no editor:

var = 1

print(var)

**Console**

**1**

Consiste em duas instruções simples:

O primeiro deles cria uma variável chamada var e atribui um literal com um valor inteiro igual a 1.

O segundo imprime o valor da variável recém-criada no console.

Como você pode ver, print() tem outro lado – ele também pode manipular variáveis. Você sabe qual será a saída do snippet? Execute o código para verificar.

**2.4.4 Como usar uma variável**

Você tem permissão para usar quantas declarações de variáveis forem necessárias para atingir seu objetivo, assim:

**var = 1**

**var2 = 2**

**var3 = (var + var2)**

**print(var)**

**print(var2)**

**print(var3)**

**Console**

**1**

**2**

**3**

**var = 1**

**account\_balance = 1000.0**

**client\_name = 'John Doe'**

**print(var, account\_balance, client\_name)**

**print(var)**

**Console**

**1 1000.0 John Doe**

**1**

No entanto, você não pode usar uma variável que não existe (em outras palavras, uma variável que não recebeu um valor).

Este exemplo **causará um erro:**

**var = 1**

**print(Var) # Mandamos imprimir uma variável Var com maiusculo (náo existe)**

**Console**

**Traceback (most recent call last):**

**File "main.py", line 2, in <module>**

**print(Var)**

**NameError: name 'Var' is not defined**

Você sabe por quê? Tentamos usar uma variável chamada Var, que não tem nenhum valor (nota: var e Var são entidades diferentes e não têm nada em comum no que diz respeito ao Python).

**Lembre-se**

Você pode usar a função print() e combinar texto e variáveis usando o operador + para gerar sequências e variáveis. Por exemplo:

var = "3.8.5"

*print*("Versão Python: " + var)

Versão Python: 3.8.5

**2.4.5 Como atribuir um novo valor a uma variável já existente**

Como você atribui um novo valor a uma variável que já existe? Da mesma forma. Você só precisa usar o sinal de igual.

O sinal de igual é, na verdade, um operador de atribuição. Embora isso possa parecer estranho, o operador tem uma sintaxe simples e uma interpretação clara.

Ela atribui o valor do argumento da direita para a esquerda, enquanto o argumento da direita pode ser uma expressão arbitrariamente complexa envolvendo literais, operadores e variáveis já definidas.

Veja os códigos abaixo:

***var = 1***

***print(var)***

***var = var + 1***

***print(var)***

**Console**

**1**

**2**

O código envia duas linhas para o console:

* A primeira linha do trecho cria uma nova variável chamada var e atribui 1 a ela.
* A instrução diz: atribua um valor de 1 a uma variável denominada var.
* Podemos dizer mais curto: atribua 1 ao var.
* Alguns preferem ler uma declaração como: var se torna 1.
* A terceira linha atribui a mesma variável com o novo valor retirado da própria variável, somada com 1. Vendo um registro como esse, um matemático provavelmente protestaria - nenhum valor pode ser igual a si mesmo mais um. Isso é uma contradição. Mas Python trata o sinal = não como igual a, mas como atribuir um valor a.

Então, como você lê esse registro no programa?

Pegue o valor atual da variável var, adicione 1 e armazene o resultado na variável var.

Com efeito, o valor da variável var tem sido incrementado por um, o que não tem nada a ver com a comparação da variável com qualquer valor.

Outro exemplo a seguir:

**var = 2**

**print(var)**

**var = (var + 1) \*(var +5)**

**print(var)**

**Console**

**2**

**21**

Você sabe qual será a saída do snippet a seguir?

**var = 100**

**var = 200 + 300**

***print*(var)**

**500** ‒ por que?

* Bem, primeiro, o var variável é criada e recebe o valor 100.
* Em seguida, a mesma variável recebe um novo valor: o resultado da adição de 200 a 300, que é 500.

**2.4.6 Solução de problemas matemáticos simples**

Agora você deve ser capaz de construir um programa curto que resolva problemas matemáticos simples, como o teorema de Pitágoras:

**O quadrado da hipotenusa** é **igual à soma dos quadrados dos outros dois lados**.

O código a seguir avalia o comprimento da hipotenusa (ou seja, o lado mais longo de um triângulo retângular, o oposto do ângulo reto) usando o teorema de Pitágoras:

a = 3.0

b = 4.0

c = (a \*\* 2 + b \*\* 2) \*\* 0.5

*print*("c =", c)

Nota: precisamos fazer uso do operador \*\* para avaliar a raiz quadrada como:

√ (x)  = x(½)

e

c = √ a2 + b2

Você consegue adivinhar a saída do código?

**a = 3.0**

**b = 4.0**

**c = (a \*\* 2 + b \*\* 2) \*\* 0.5**

**print("c =", c)**

**Console**

**c = 5.0**

**2.4.7 LAB Variáveis**

Cenário

Aqui está uma breve história:

Era uma vez em Appleland, John tinha três maçãs, Maria cinco maçãs e Adam tinha seis maças. Todos ficaram muito felizes e viveram por muito tempo. Fim da história.

**Sua tarefa:**

* crie as variáveis: john, mary e adam;
* atribuir valores às variáveis. Os valores devem ser iguais aos números de fruto possuído por John, Mary e Adam, respectivamente;
* tendo armazenado os números nas variáveis, imprimindo as variáveis em uma linha e separando cada uma delas com uma vírgula;
* Agora, crie uma nova variável chamada total\_apples igual à adição das três variáveis anteriores.
* imprima o valor armazenado em total\_apples no console;

**john = 3**

**mary = 5**

**adam = 6**

**print(john, mary, adam, sep=',')**

**total\_apples = john + mary + adam**

**print(total\_apples)**

**Console**

**3,5,6**

**14**

* Agora experimente com seu código: crie novas variáveis, atribua valores diferentes a elas e execute várias operações aritméticas nelas (por exemplo, +, -, \*, /, //, etc.). Tente imprimir uma sequência de caracteres e um número inteiro juntos em uma linha, por exemplo, "Número total de maças:" e total\_apples.

**city = "Appleland"**

**John = 3**

**Maria = 5**

**Adam = 6**

**total\_apples = (John + Maria + Adam)**

**print(John, Maria, Adam)**

**print(total\_apples)**

**#aqui somando o total de maçãs nova variável**

**dist\_apples\_igual = (total\_apples) / 3**

**print(dist\_apples\_igual)**

**# Usando a concatenação com virugula**

**print("John => " , John , " Apples")**

**print("Maria => " , Maria , " Apples")**

**print("Adam => " , Adam , " Apples")**

**print("- - - - - - - - - - - - - - - -")**

**print ("Eles moram em => " , city)**

**print(" O total de Apples que possuem é => " , total\_apples)**

**print(" Divindo as Apples igualmente, cada um receberá => " , dist\_apples\_igual)**

**Console**

**3 5 6**

**14**

**4.666666666666667**

**John => 3 Apples**

**Maria => 5 Apples**

**Adam => 6 Apples**

**- - - - - - - - - - - - - - - -**

**Eles moram em => Appleland**

**O total de Apples que possuem é => 14**

**Divindo as Apples igualmente, cada um receberá => 4.666666666666667**

**2.4.8 Operadores atalhos**

É hora do próximo conjunto de operadores que facilita a vida do desenvolvedor. Frequentemente, queremos usar uma e a mesma variável nos lados direito e esquerdo do **operador =.**

Por exemplo, se precisamos calcular uma série de valores sucessivos de potências de 2, podemos usar uma parte como esta:

**x = x \* 2**

Você pode usar uma expressão como essa se não conseguir adormecer e estiver tentando lidar com ela usando alguns métodos bons e antiquados:

sheep = sheep + 1

O Python oferece uma maneira reduzida de escrever operações como essas, que podem ser codificadas da seguinte forma:

x \*= 2

sheep += 1

Vamos tentar apresentar uma descrição geral para essas operações. Se op for um operador de dois argumentos (esta é uma condição muito importante) e o operador for usado no seguinte contexto...:

variable = variable op expression

... então pode ser simplificado e exibido da seguinte forma:

variable op= expression

Veja os exemplos abaixo. Certifique-se de entender todos eles.

|  |  |
| --- | --- |
| **Expressão** | **Operador de atalho** |
| **i = i + 2 \* j** | **i += 2 \* j** |
| **var = var / 2** | **var /= 2** |
| **rem = rem % 10** | **rem %= 10** |
| **j = j - (i + var + rem)** | **j -= (i + var + rem)** |
| **x = x \*\* 2** | **x \*\*= 2** |

**2.4.9 LAB Variáveis ‒ um simples conversor**

Cenário

Milhas e quilômetros são unidades de comprimento ou distância.

Lembrando que **1 milha é igual a aproximadamente 1.61 quilômetros**, complete o programa no editor para que converta:

* milhas em quilômetros;
* quilômetros em milhas.

Não altere nada no código atual. Escreva seu código nos locais indicados por ###. Teste seu programa com os dados que fornecemos no código-fonte.

Preste atenção especial ao que está acontecendo na função de print(). Analise como fornecemos multiplos argumentos para a função e como produzimos os dados esperados.

Observe que alguns dos argumentos dentro da função print() são strings (por exemplo, "milhas é", enquanto outros são variáveis (por exemplo, miles).

Dica

Há mais uma coisa interessante acontecendo lá. Você pode ver outra função dentro da função print()? É a função round(). Seu trabalho é arredondar o resultado de saída para o número de casas decimais especificadas entre parênteses e retornar um float (dentro da função **round()** você pode encontrar o nome da variável, uma vírgula e o número de casas decimais que pretendemos por). Falaremos sobre funções muito em breve, então não se preocupe que tudo pode não estar totalmente claro ainda. Só queremos despertar a sua curiosidade.

Depois de concluir o laboratório, abra o Sandbox e experimente um pouco mais. Tente escrever conversores diferentes, por exemplo, um conversor de USD para EUR, um conversor de temperatura, etc. - deixe sua imaginação voar! Tente gerar os resultados combinando strings e variáveis. Tente usar e experimentar a função round() para arredondar seus resultados para uma, duas ou três casas decimais. Confira o que acontece se você não fornecer qualquer número de dígitos. Lembre-se de testar seus programas.

Experimente, tire conclusões e aprenda. Seja curioso.

Saída esperada

7.38 milhas é 11.88 quilômetros

12.25 quilômetros é 7.61 milhas

Desafio completar as áreas com ###

kilometers = 12.25

miles = 7.38

miles\_to\_kilometers = ###

kilometers\_to\_miles = ###

print(miles, "milhas é", round(miles\_to\_kilometers, 2), "quilômetros")

print(kilometers, "quilômetros é", round(kilometers\_to\_miles, 2), "milhas")

EXECUÇÃO

**kilometers = 12.25**

**miles = 7.38**

**# CONSIDERAR: 1 milha é igual a aproximadamente 1.61 quilômetros**

**# 1 - Converter miles em km**

**miles\_to\_kilometers = miles \* 1.61**

**# 2 - Converter km em miles**

**kilometers\_to\_miles = kilometers / 1.61**

**print(miles, "milhas é", round(miles\_to\_kilometers, 2), "quilômetros")**

**print(kilometers, "quilômetros é", round(kilometers\_to\_miles, 2), "milhas")**

**Console**

**7.38 milhas é 11.88 quilômetros**

**12.25 quilômetros é 7.61 milhas**

**2.4.10 LAB Operadores e Expressões**

Cenário

Dê uma olhada no código no editor: ele lê um valor float, coloca-o em uma variável chamada x e imprime o valor de uma variável chamada y. Sua tarefa é completar o código para avaliar a seguinte expressão:

3x³ - 2x² + 3x - 1

O resultado deve ser atribuído a y.

Lembre-se de que a notação algébricas clássica gosta de omitir o operador de multiplicação - você precisa usá-la explicitamente. Observe como mudamos o tipo de dados para garantir que x seja do tipo float.

Mantenha seu código limpo e legível e teste-o usando os dados que fornecemos, atribuindo-o sempre à variável x (codificando-o). Não desanime por qualquer falha inicial. Seja persistente e inquisitivo.

**Exemplo de entrada**

x = 0

x = 1

x = -1

**Exemplo de saída**

y = -1.0

y = 3.0

y = -9.0

x = 0# Codifique seus dados de teste aqui.

x = float(x)

y = (3 \* x \*\*3) - (2 \* x \*\* 2) + (3 \* x) - 1# Escreva seu código aqui.

print("y =", y)

x = 1# Codifique seus dados de teste aqui.

y = (3 \* x \*\*3) - (2 \* x \*\* 2) + (3 \* x) - 1# Escreva seu código aqui.

print("y =", y)

x = -1# Codifique seus dados de teste aqui.

y = (3 \* x \*\*3) - (2 \* x \*\* 2) + (3 \* x) - 1# Escreva seu código aqui.

print("y =", y)

**Console**

**y = -1.0**

**y = 3**

**y = -9**

**x = 0 # Codifique seus dados de teste aqui.**

**x = float(x)**

**y = 3 \* x \*\*3 - 2 \* x \*\* 2 + 3 \* x – 1 # Escreva seu código aqui.**

**print("y =", y)**

**x = 1 # Codifique seus dados de teste aqui.**

**x = float(x)**

**y = 3 \* x \*\*3 - 2 \* x \*\* 2 + 3 \* x – 1 # Escreva seu código aqui.**

**print("y =", y)**

**x = -1# Codifique seus dados de teste aqui.**

**x = float(x)**

**y = 3 \* x \*\*3 - 2 \* x \*\* 2 + 3 \* x – 1 # Escreva seu código aqui.**

**print("y =", y)**

**Console**

**y = -1.0**

**y = 3.0**

**y = -9.0**

**2.4.11 RESUMO DA SEÇÃO**

1. Uma variável é um local nomeado reservado para armazenar valores na memória. Uma variável é criada ou inicializada automaticamente quando você atribui um valor a ela pela primeira vez. (2.1.4.1)
2. Cada variável deve ter um nome exclusivo ‒ um identificador. Um nome de identificador legal deve ser uma sequência não vazia de caracteres, deve começar com o sublinhado (\_) ou uma letra e não pode ser uma palavra-chave do Python. O primeiro caractere pode ser seguido por sublinhados, letras e dígitos. Os identificadores em Python diferenciam maiúsculas de minúsculas.
3. Python é uma linguagem de tipagem dinâmica, o que significa que você não precisa declarar variáveis nela. (2.1.4.3) Para atribuir valores a variáveis, você pode usar um operador de atribuição simples na forma do sinal de igual (=), ou seja, var = 1.
4. Você também pode usar operadores de atribuição compostos (operadores de atalho) para modificar valores atribuídos a variáveis, por exemplo: var += 1 ou var /= 5 \* 2.
5. Você pode atribuir novos valores a variáveis já existentes usando o operador de atribuição ou um dos operadores compostos, por exemplo:

var = 2

print(var)

var = 3

print(var)

var += 1

print(var)

CONSOLE

2

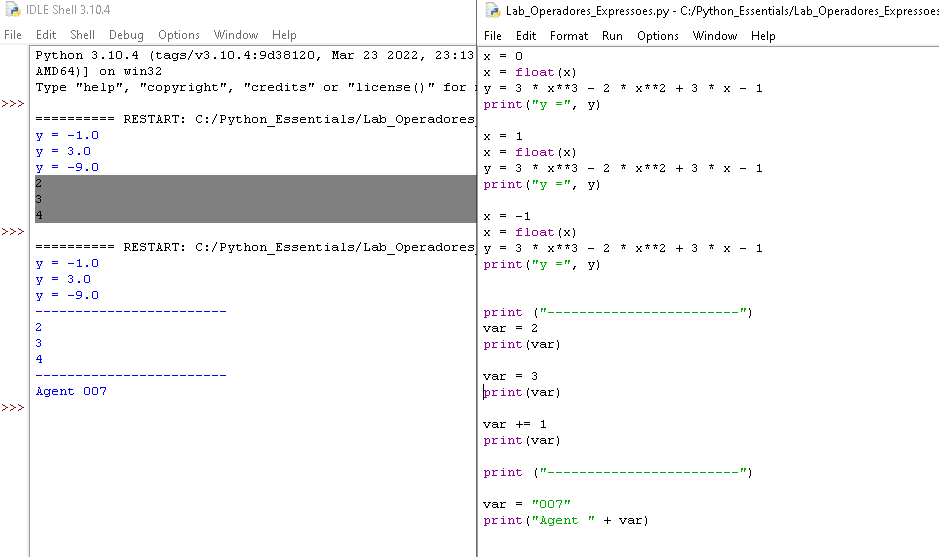
3

4

Você pode combinar texto e variáveis usando o operador + e usar a função print() para gerar sequências e variáveis, por exemplo:

var = "007"

print("Agent " + var)



**2.4.12 TESTE DA SEÇÃO**

Pergunta 1: Qual é a saída do código a seguir?

var = 2

var = 3

print(var)

* 3
* 2
* 5

Pergunta de múltipla escolha

Pergunta 2: Quais dos seguintes nomes de variáveis são ilegais no Python? (Selecione três respostas)

* my\_var
* m
* 101
* averylongVariablename
* m101
* m 101
* del
* Del

Pergunta 3: Qual é a saída do trecho a seguir?

a = '1'

b = "1"

print(a + b)

* 1
* 11
* 2
* Erro

Pergunta 4: Qual é a saída do trecho a seguir?

a = 6

b = 3

a /= 2 \* b

print(a)

* 1.0
* 1
* 9
* 6
* 6.0

**2.5 Seção 5 - Comentários**

Bem-vindo à seção cinco, na qual compartilharemos alguns comentários sobre comentários. Você aprenderá aqui por que é importante documentar seu código e por que você deve deixar comentários. Você também aprenderá como fazer isso e quando os comentários forem considerados uma boa prática. Vamos lá!

**2.5.1 Comentários - por que, quando e como?**

Você pode querer colocar algumas palavras dirigidas não ao Python, mas aos humanos, geralmente para explicar a outros leitores do código como funcionam os truques usados no código ou o significado das variáveis e, eventualmente, para manter armazenado informações sobre quem é o autor e quando o programa foi escrito.

Uma observação inserida no programa, que é omitida em tempo de execução, é chamada de comentário.

Como você deixa esse tipo de comentário no código-fonte? Ela precisa ser feita de forma que não force o Python a interpretá-la como parte do código.

Sempre que o Python encontra um comentário em seu programa, o comentário é completamente transparente para ele ‒ do ponto de vista do Python, esse é apenas um espaço (independentemente do tamanho do comentário real).

Em Python, um comentário é um pedaço de texto que começa com um sinal # (hash) e se estende até o final da linha.

Se você deseja um comentário que se estenda por várias linhas, deve colocar um hash na frente de todas elas. Assim como aqui:

# Esse programa calcula a hipotenusa c.

# a e b são os tamanhos dos lados.

a = 3.0

b = 4.0

c = (a \*\* 2 + b \*\* 2) \*\* 0.5 # Nós usamos \*\* ao invés de raiz quadrada.

print("c =", c)

Desenvolvedores bons e responsáveis descrevem cada parte importante do código, por exemplo, explicando o papel das variáveis. Embora deva ser declarado que a melhor maneira de comentar variáveis é nomeá-las de maneira inequívoca.

Por exemplo, se uma variável específica for projetada para armazenar uma área de algum quadrado único, o nomesquare\_area será obviamente melhor do que aunt\_jane.

Dizemos que o primeiro nome está auto comentado.

**2.5.2 Marcando fragmentos de código**

Os comentários podem ser úteis sob outra perspectiva - você pode usá-los para marcar um código que não é necessário no momento por qualquer motivo. Veja o exemplo abaixo, se você descomentar a linha destacada, isso afetará a saída do código:

# Esse é um programa de teste.

x = 1

y = 2

# y = y + x

print(x + y)

Isso geralmente é feito durante o teste de um programa, a fim de isolar o local onde um erro pode estar oculto.

**Dica**

Se desejar comentar ou descomentar rapidamente várias linhas de código, selecione a(s) linha(s) que deseja modificar e use o seguinte atalho de teclado: **CTRL+ /** (Windows) ou CMD + / (Mac OS). É um truque muito útil, não é? Agora experimente o código no editor.

# uncomment\_me = 1

# uncomment\_me\_too = 3

# uncomment\_me\_also = 5

print(uncomment\_me, uncomment\_me\_too, uncomment\_me\_also, sep="\n")

**2.5.3 LAB Comentários**

Cenário

O código no editor contém comentários. Tente melhorá-lo: adicione ou remova comentários onde achar apropriado (sim, às vezes, remover um comentário pode tornar o código mais legível) e altere os nomes de variáveis onde você achar que isso melhorará a compreensão do código.

***Observação***

Os comentários são muito importantes. Eles são usados não apenas para tornar seus programas mais fáceis de entender, mas também para desabilitar as partes do código que não são necessárias no momento (por exemplo, quando você precisa testar apenas algumas partes do seu código e ignorar outras). Bons programadores descrevem cada parte importante do código e dão nomes auto-comentados às variáveis, pois às vezes é simplesmente muito melhor deixar as informações no código.

É bom usar nomes de variáveis legíveis e, às vezes, é melhor dividir seu código em partes nomeadas (por exemplo, funções). Em algumas situações, é uma boa ideia escrever as etapas dos cálculos de forma mais clara.

Só mais uma coisa: pode acontecer de um comentário conter uma informação errada ou incorreta - você nunca deve fazer isso de propósito!

esse programa calcula o número de segundos em um dado número de horas

# esse programa foi escrito a dois dias atrás

a = 2 # número de horas

seconds = 3600 # número de segundos em uma hora

print("Horas: ", a) #imprimindo o número de horas

print("Seconds in Hours: ", a \* seconds) # imprimindo o número de segundos em uma dada hora

#aqui também devería escrever "Adeus", mas o programador não teve tempo de escrever código

print("Obrigado pela sua atenção. Adeus e até a próxima!")

#esse é o final do programa que computa o número de segundos em 3 horas

a = 3

print("Seconds in ", a , "Hours =>", a\* seconds)

**Console**

**Horas: 2**

**Seconds in Hours: 7200**

**Obrigado pela sua atenção. Adeus e até a próxima!**

**Seconds in 3 Hours => 10800**

**2.5.4 RESUMO DA SEÇÃO**

Os comentários podem ser usados para deixar informações adicionais no código. Eles são omitidos em tempo de execução. As informações deixadas no código-fonte são endereçadas aos leitores humanos. Em Python, um comentário é um texto que começa com #. O comentário se estende até o final da linha.

Se você deseja colocar um comentário que abrange várias linhas, você precisa colocar # na frente de todas. Além disso, você pode usar um comentário para marcar um pedaço de código que não é necessário no momento (veja a última linha do snippet abaixo), por exemplo:

**# Este programa imprime**

**# uma introdução à tela.**

**print("Olá!") # Chamando a função print()**

**# print("Em Python.")**

Sempre que possível e justificado, você deve fornecer nomes auto comentados a variáveis, por exemplo, se você estiver usando duas variáveis para armazenar o length e width de algo, podem ter uma escolha melhor que o myvar1 e o myvar2.

É importante usar comentários para facilitar a compreensão dos programas e usar nomes de variáveis legíveis e significativos no código. No entanto, é igualmente importante não usar nomes de variáveis que são confusos ou deixar comentários que contêm informações incorretas ou incorretas!

Os comentários podem ser importantes quando você está lendo seu próprio código depois de algum tempo (confie em nós, os desenvolvedores esquecem o que seu próprio código faz) e quando outros estão lendo seu código (eles podem ajudá-los a entender o que seus programas fazem e como eles fazem isso mais rapidamente).

**2.5.5 TESTE DA SEÇÃO**

Pergunta 1: Qual é a saída do trecho a seguir?

# print("String #1")

print("String #2")

Verificar

String #2

Pergunta 2: O que acontecerá quando você executar o código a seguir?

# Isso é

uma multilinha

Comente. #

print("Olá!")

Verificar

SyntaxError: invalid syntax

**2.6 Seção 6 - Interação com o usuário**

Nesta seção, você aprenderá a falar com um computador: você vai se familiarizar com a função input (), realizar conversões de tipo e aprender a usar operadores de string.

**2.6.1 A função input()**

Vamos agora apresentar uma função completamente nova, que parece ser um reflexo da boa e velha função print().

Por quê? Bem, print() envia dados para o console.

A nova função obtém dados dela.

print() não tem resultado utilizável. O significado da nova função é retornar um resultado muito útil.

A **função** é chamada de **input().** O nome da função diz tudo.

A função input() é capaz de ler os dados inseridos pelo usuário e retornar os mesmos dados para o programa em execução.

O programa pode manipular os dados, tornando o código verdadeiramente interativo.

Praticamente todos os programas lêem e processam dados. Um programa que não recebe a entrada de um usuário é um programa surdo.

Dê uma olhada no nosso exemplo:

print("Conta-me qualquer coisa...")

anything = input()

print("Hum...", anything, "... Realmente?")

Ele mostra um caso muito simples de uso da função input().

**Nota:**

* O programa solicita que o usuário insira alguns dados do console (provavelmente usando um teclado, embora também seja possível inserir dados usando voz ou imagem);
* a função input() é invocada sem argumentos (essa é a maneira mais simples de usar a função); a função mudará o console para o **modo de entrada**; você verá um cursor piscando e poderá inserir algumas teclas, finalizando pressionando a tecla Enter; todos os dados inseridos serão enviados ao seu programa através do resultado da função;
* **nota**: você precisa atribuir o resultado a uma variável; isso é crucial - perder esta etapa fará com que os dados inseridos sejam perdidos;
* em seguida, usamos a função print() para exibir os dados que obtemos, com algumas observações adicionais.
* Execute o código e deixe a função mostrar o que ela pode fazer por você.

print("Conta-me qualquer coisa...")

anything = input()

print("Hum...", anything, "... Realmente?")

**Console**

**Conta-me qualquer coisa...**

**eu corria muito quando era jovem**

**Hum... eu corria muito quando era jovem ... Realmente?**

**2.6.2 A função input() com um argumento**

A função input() pode fazer outra coisa: pode solicitar ao usuário sem qualquer ajuda de print().

Modificamos um pouco nosso exemplo, veja o código:

anything = input("Conta-me qualquer coisa...")

print("Hum...", anything, "...Realmente?")

**Nota:**

* a função input() é invocada com um argumento ‒ é uma string contendo uma mensagem;
* a mensagem será exibida no console antes que o usuário tenha a oportunidade de digitar qualquer coisa;
* input() fará seu trabalho.

Essa variante da invocação **input()** simplifica o código e o torna mais claro.

**2.6.3 O resultado da função input()**

Já dissemos isso, mas deve ser afirmado de forma inequívoca mais uma vez: o resultado da função input() é uma string.

Uma string contendo todos os caracteres que o usuário insere no teclado. Não é um inteiro ou um float.

Isso significa que você não deve usá-lo como argumento de nenhuma operação aritmética, por exemplo, você não pode usar esses dados para elevá-los ao quadrado, dividi-los por qualquer coisa ou dividir qualquer coisa por eles.

**anything = input("Digite um número: ")**

**something = anything \*\* 2.0**

**print(anything, "elevado a 2 é", something)**

**2.6.4 A função input() - operações proibidas**

Veja o código no editor. Execute-o, digite qualquer número e pressione Enter.

anything = input("Digite um número: ")

something = anything \*\* 2.0

print(anything, "elevado a 2 é", something)

**Console**

**Digite um número: 10**

**Traceback (most recent call last):**

**File "main.py", line 4, in**

**something = anything \*\* 2.0**

**TypeError: unsupported operand type(s) for \*\* or pow(): 'str' and 'float'**

O que acontece? O Python deveria ter lhe dado a seguinte saída:

Traceback (most recent call last):

File ".main.py", line 4, in

something = anything \*\* 2.0

TypeError: unsupported operand type(s) for \*\* or pow(): 'str' and 'float'

A última linha da frase explica tudo - você tentou aplicar o operador \*\* a 'str' (string) acompanhado de 'float'. ***Isso é proibido.***

Isso deve ser óbvio - você pode prever o valor de "ser ou não ser" elevado à potência de 2? Nós não podemos. Python também não.

Já caímos em um impasse? Há uma solução para esse problema? Claro que sim.

**2.6.5 Conversão de tipo (tipo de conversões)**

O Python oferece duas funções simples para especificar um tipo de dados e resolver esse problema: aqui estão eles: int() e float().

Seus nomes são de comentários automáticos:

a **função int()** usa um argumento (por exemplo, uma string: int(string)) e tenta convertê-lo em um número inteiro; se falhar, o programa inteiro também falhará (há uma solução para essa situação, mas mostraremos isso um pouco mais tarde);

a **função float()** usa um argumento (por exemplo, uma string: float(string)) e tenta convertê-la em um flutuante (o resto é o mesmo).

Isso é muito simples e muito eficaz. Além disso, você pode chamar qualquer uma das funções passando os resultados de input() diretamente para elas. Não há necessidade de usar nenhuma variável como armazenamento intermediário.

Implementamos a ideia no editor. Dê uma olhada no código.

Você pode imaginar como a string inserida pelo usuário flui de input() para print()?

Tente executar o código modificado. Não se esqueça de inserir um número válido.

Verifique alguns valores diferentes, pequeno e grande, negativo e positivo. Zero também é uma boa entrada.

anything = float(input("Digite um número: "))

something = anything \*\* 2.0

print(anything, "elevado a 2 é", something)

**Console**

**Digite um número: 10**

**10.0 elevado a 2 é 100.0**

**2.6.6 Mais informações sobre input() e conversão de tipos**

Ter uma equipe composta por três input()-int()-float() abre muitas novas possibilidades.

Você poderá escrever programas completos, aceitando dados na forma de números, processando-os e exibindo os resultados.

Obviamente, esses programas serão muito primitivos e não muito utilizáveis, pois não podem tomar decisões e, portanto, não são capazes de reagir de forma diferente a situações diferentes.

Isso não é realmente um problema; mostraremos como superar isso em breve.

Nosso próximo exemplo se refere ao programa anterior para encontrar o comprimento de uma hipotenusa. Vamos executá-lo e torná-lo capaz de ler os comprimentos das pernas do console.

Confira a janela do editor - é isso que parece agora:

**leg\_a = float(input("Insira o comprimento da primeira perna: "))**

**leg\_b = float(input("Insira o comprimento da segunda perna: "))**

**hypo = (leg\_a\*\*2 + leg\_b\*\*2) \*\* .5**

**print("O comprimento da hipotenusa é", hypo)**

**Console**

**Insira o comprimento da primeira perna: 10**

**Insira o comprimento da segunda perna: 12**

**O comprimento da hipotenusa é 15.620499351813308**

O programa solicita ao usuário o comprimento de ambas as pernas, avalia a hipotenusa e imprime o resultado. Execute-o e tente inserir alguns valores negativos.

O programa, infelizmente, não reage a esse erro óbvio. Vamos ignorar essa fraqueza por enquanto. Voltaremos em breve.

Observe que, no programa que você pode ver no editor, a variável hypo é usada para apenas um objetivo - para salvar o valor calculado entre a execução da linha de código adjacente.

Como a função print() aceita uma expressão como seu argumento, você pode remover a variável do código.

Bem assim:

**leg\_a = float(input("Insira o comprimento da primeira perna: "))**

**leg\_b = float(input("Insira o comprimento da segunda perna: "))**

**print("O comprimento da hipotenusa é", (leg\_a\*\*2 + leg\_b\*\*2) \*\* .5)**

**Console**

**Insira o comprimento da primeira perna: -10**

**Insira o comprimento da segunda perna: 20**

**O comprimento da hipotenusa é 22.360679774997898**

**2.6.7 Operadores de string**

É hora de voltar para esses dois operadores aritméticos: + e \*.

Queremos mostrar que eles têm uma segunda função. Eles são capazes de fazer algo mais do que apenas adicionar e multiplicar.

Já vimos eles em ação, onde seus argumentos são números (flutuantes ou números inteiros, não importa).

Agora, vamos mostrar que eles podem lidar com cadeias de caracteres, embora de uma forma muito específica.

O sinal de + (mais), quando aplicado a duas **cadeias de caracteres**, torna-se um operador de concatenação:

string + string

Ela simplesmente concatena (cola) duas sequências de caracteres em uma. Obviamente, como seu irmão aritmético, ele pode ser usado mais de uma vez em uma expressão e, nesse contexto, se comporta de acordo com a ligação do lado esquerdo.

Em contraste com seu irmão aritmético, o operador de concatenação não é comutativo, ou seja, "ab" + "ba" não é o mesmo que "ba" + "ab".

Não se esqueça - se você quiser que o **sinal de + seja um concatenador, não um somador, certifique-se de que ambos os argumentos sejam cadeias**.

Você não pode misturar tipos aqui.

Este programa simples mostra o sinal + em seu segundo uso:

**fnam = input("Posso ter seu primeiro nome, por favor? ")**

**lnam = input("Posso ter seu sobrenome, por favor? ")**

**print("Obrigado!.")**

**print("\nSeu nome é " + fnam + " " + lnam + ".")**

**Console**

**Posso ter seu primeiro nome, por favor? Jose**

**Posso ter seu sobrenome, por favor? da Silva**

**Obrigado!.**

**Seu nome é Jose da Silva.**

**Nota:** usar + para concatenar strings permite que você construa a saída de uma maneira mais precisa do que com uma função print() pura, mesmo se enriquecida com os argumentos de palavra-chave end= e sep=.

Execute o código e veja se a saída corresponde às suas previsões.

**Replicação**

O sinal \* (asterisco), quando aplicado a uma string e um número (ou um número e uma string, pois permanece comutativo nessa posição) se torna um operador de replicação:

string \* number

number \* string

Ele replica a string o mesmo número de vezes especificado pelo número.

Por exemplo:

* **"James" \* 3 gera "JamesJamesJames"**
* **3 \* "an" gera "ananan"**
* **5 \* "2" (or "2" \* 5) gera "22222" (não10!)**

**Lembre-se**

Um número menor ou igual a zero produz uma string vazia.

Este simples programa "desenha" um retângulo, fazendo uso de um antigo operador (+) em um novo papel:

**print("+" + 10 \* "-" + "+")**

**print(("|" + " " \* 10 + "|\n") \* 5, end="")**

**print("+" + 10 \* "-" + "+")**

**Console**

**+----------+**

**| |**

**| |**

**| |**

**| |**

**| |**

**+----------+**

**print("+" + 20 \* "-" + "+")**

**print(("|" + " " \* 20 + "|\n") \* 10, end="")**

**print("+" + 20 \* "-" + "+")**

**Console**

**+--------------------+**

**| |**

**| |**

**| |**

**| |**

**| |**

**| |**

**| |**

**| |**

**| |**

**| |**

**+--------------------+**

Observe a maneira como usamos os parênteses na segunda linha do código.

Tente praticar para criar outras formas ou sua própria arte!

**2.6.8 Conversões de tipo mais uma vez concluídas**

**str()**

Você já sabe como usar as funções int() e float() para converter uma string em um número.

Esse tipo de conversão não é uma via de mão única. Você também pode converter um número em uma string, o que é muito mais fácil e seguro - esse tipo de operação é sempre possível.

Uma função capaz de fazer isso é chamada str():

**str(number)**

Para ser honesto, ele pode fazer muito mais do que apenas transformar números em cadeias, mas isso pode esperar mais tarde.

**O triângulo de ângulo direito novamente**

Aqui está nosso programa de "ângulo reto" novamente:

**leg\_a = float(input("Insira o comprimento da primeira perna: "))**

**leg\_b = float(input("Insira o comprimento da segunda perna: "))**

**print("O comprimento da hipotenusa é " + str((leg\_a\*\*2 + leg\_b\*\*2) \*\* .5))**

**Console**

**Insira o comprimento da primeira perna: 10**

**Insira o comprimento da segunda perna: 20**

**O comprimento da hipotenusa é 22.360679774997898**

Nós modificamos um pouco para mostrar como a função str() funciona. Graças a isso, podemos passar o resultado inteiro para a função print() como uma string, esquecendo as vírgulas.

Você fez alguns avanços importantes no caminho para a programação em Python.

Você já conhece os tipos de dados básicos e um conjunto de operadores fundamentais. Você sabe como organizar a saída e como obter dados do usuário. Essas são as bases muito fortes do Módulo 3. Mas antes de prosseguirmos para o próximo módulo, vamos fazer alguns laboratórios e recapitular tudo o que você aprendeu nesta seção.

**2.6.9 LAB Simples entradas e saídas**

Cenário

Sua tarefa é completar o código para avaliar os resultados de quatro operações aritméticas básicas.

Os resultados devem ser impressos no console.

Talvez você não consiga proteger o código de um usuário que deseja dividir por zero. Tudo bem, não se preocupe com isso por enquanto.

Teste seu código ‒ ele produz os resultados esperados?

Não mostraremos nenhum dado de teste - isso seria muito simples.

# entre com um valor float para a variável a aqui

# entre com um valor float para a variável b aqui

# imprima o resultado da adição aqui

# imprima o resultado da subtração aqui

# imprima o resultado da multiplicação aqui

# imprima o resultado da divisão aqui

print("\nIsso é tudo, pessoal!")

Solução: Utilizado os valores das variáveis em float e no print(utilizamos o str para operações). Nos exemplos abaixo são: **virgula , e no outro o sinal de positivo +, fazendo a concatenação**

**valor\_a = float("10")# entre com um valor float para a variável a aqui**

**valor\_b = float("5")# entre com um valor float para a variável b aqui**

**print("A soma dos valores de a e b são => " , str(valor\_a + valor\_b)) # imprima o resultado da adição aqui**

**print(" A subtração dos valores de a e b é => " , str(valor\_a - valor\_b))# imprima o resultado da subtração aqui**

**print("A multiplicação dos valores de a e b é => " , str(valor\_a \* valor\_b))# imprima o resultado da multiplicação aqui**

**print("A divisão dos valores de a e b é =>" , str(valor\_a / valor\_b))# imprima o resultado da divisão aqui**

**print("\nIsso é tudo, pessoal!")**

**valor\_a = float("10")# entre com um valor float para a variável a aqui**

**valor\_b = float("5")# entre com um valor float para a variável b aqui**

**print("A soma dos valores de a e b são => " + str(valor\_a + valor\_b)) # imprima o resultado da adição aqui**

**print(" A subtração dos valores de a e b é => " + str(valor\_a - valor\_b))# imprima o resultado da subtração aqui**

**print("A multiplicação dos valores de a e b é => " + str(valor\_a \* valor\_b))# imprima o resultado da multiplicação aqui**

**print("A divisão dos valores de a e b é => " + str(valor\_a / valor\_b))# imprima o resultado da divisão aqui**

**print("\nIsso é tudo, pessoal!")**

**Console**

**A soma dos valores de a e b são => 15.0**

**A subtração dos valores de a e b é => 5.0**

**A multiplicação dos valores de a e b é => 50.0**

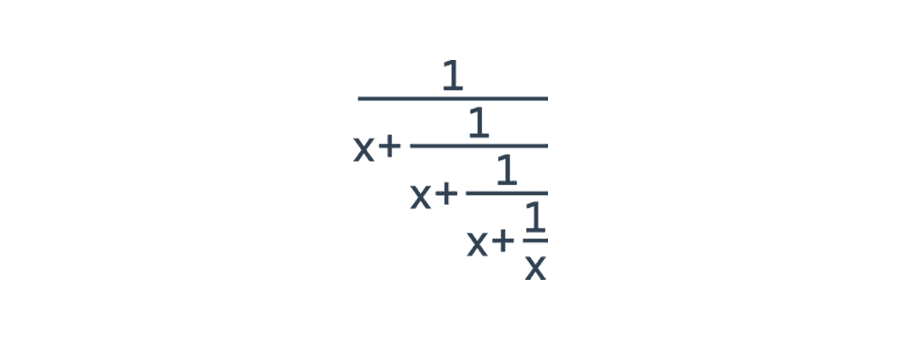
**A divisão dos valores de a e b é => 2.0**

**Isso é tudo, pessoal!**

**2.6.10 LAB Operadores e Expressões**

Cenário

Sua tarefa é completar o código para avaliar a seguinte expressão:



O resultado deve ser atribuído a y. Tenha cuidado - observe os operadores e mantenha suas prioridades em mente. Não hesite em usar quantos parênteses forem necessários.

Você pode usar variáveis adicionais para encurtar a expressão (mas não é necessário). Teste seu código com cuidado.

**Dados de teste**

Exemplo de entrada:

1

Saída prevista:

y = 0.6000000000000001

Output

Exemplo de entrada:

10

Saída prevista:

y = 0.09901951266867294

Output

Exemplo de entrada:

100

Saída Prevista:

y = 0.009999000199950014

Output

Exemplo de entrada:

-5

Saída prevista:

y = -0.19258202567760344

Output

**X** QUASE ACERTEI (ERRADO O MEU RACIOCÍNIO FOI O DEBAIXO)

**x = float(input("Digite o valor 1 para x: "))**

**y = (1 /(x + 1 /(x+1 / (x+1))))# Escreva seu código aqui.**

**print("y =", y)**

**x = float(input("Digite o valor 10 para x: "))**

**y = (1 /(x + 1 /(x+1 / (x+1))))# Escreva seu código aqui.**

**print("y =", y)**

**x = float(input("Digite o valor 100 para x: "))**

**y = (1 /(x + 1 /(x+1 / (x+1))))# Escreva seu código aqui.**

**print("y =", y)**

**x = float(input("Digite o valor -5 para x: "))**

**y = (1 /(x + 1 /(x+1 / (x+1))))# Escreva seu código aqui.**

**print("y =", y)**

**Console**

***Digite o valor 1 para x: 1***

***y = 0.6000000000000001***

***Digite o valor 10 para x: 10***

***y = 0.09901873327386262***

***Digite o valor 100 para x: 100***

***y = 0.009999000198970305***

***Digite o valor -5 para x: -5***

***Digite o valor 1 para x: y = -0.19266055045871558***

**O CORRETO**

**x = float(input("Digite o valor 1 para x: "))**

**y = 1./(x + 1./(x + 1./(x + 1./x))) # Escreva seu código aqui.**

**print("y =", y)**

**x = float(input("Digite o valor 10 para x: "))**

**y = 1./(x + 1./(x + 1./(x + 1./x)))**

**print("y =", y)**

**x = float(input("Digite o valor 100 para x: "))**

**y = 1./(x + 1./(x + 1./(x + 1./x)))**

**print("y =", y)**

**x = float(input("Digite o valor -5 para x: "))**

**y = 1./(x + 1./(x + 1./(x + 1./x)))**

**print("y =", y)**

**Console**

***Digite o valor 1 para x: 1***

***y = 0.6000000000000001***

***Digite o valor 10 para x: 10***

***y = 0.09901951266867294***

***Digite o valor 100 para x: 100***

***y = 0.009999000199950014***

***Digite o valor -5 para x: -5***

***y = -0.19258202567760344***

**2.6.11 LAB Operadores e Expressões – 2**

Cenário

Sua tarefa é preparar um código simples capaz de avaliar a hora de término de um período, dado como um número de minutos (pode ser arbitrariamente grande). A hora de início é fornecida como um par de horas (0..23) e minutos (0..59). O resultado deve ser impresso no console.

Por exemplo, se um evento começa às 12:17 e dura 59 minutos, termina às 13:16.

Não se preocupe com imperfeições no código - tudo bem se ele aceitar um tempo inválido - o mais importante é que o código produz resultados válidos para dados de entrada válidos.

Teste seu código com cuidado. Dica: usar o operador % pode ser a chave para o sucesso.

Dados de teste

**Exemplo de entrada:**

12

*17*

*59*

**Saída prevista:**

13:16

**Output**

**Exemplo de entrada:**

23

58

642

**Saída prevista:**

10:40

**Output**

**Exemplo de entrada:**

0

1

2939

**Saída prevista:**

1:0

**Output**

**hour = int(input("Hora de início (horas): "))**

**mins = int(input("Hora de início (minutos): "))**

**dura = int(input("Duração do evento (minutos): "))**

**mins = mins + dura # encontre um total de todos os minutos**

**hour = hour + mins // 60 # encontre um número de horas escondido em minutos e atualize a hora**

**mins = mins % 60 # minutos corretos para cair no intervalo (0..59)**

**hour = hour % 24 # horas corretas para cair no intervalo (0..23)**

**print(hour, ":", mins, sep='')**

**Console**

**Hora de início (horas): 12**

**Hora de início (minutos): 17**

**Duração do evento (minutos): 59**

**13:16**

Teste do aluno:

**hour = int(input("Hora de início (horas): "))**

**mins = int(input("Hora de início (minutos): "))**

**dura = int(input("Duração do evento (minutos): "))**

**mins = mins + dura # encontre um total de todos os minutos**

**hour = hour + mins // 60 # encontre um número de horas escondido em minutos e atualize a hora**

**mins = mins % 60 # minutos corretos para cair no intervalo (0..59)**

**hour = hour % 24 # horas corretas para cair no intervalo (0..23)**

**print(hour, ":", mins, sep='')**

**seconds = int(dura \* 60) # criando variável Calculando os segundos de duração**

**print("Duração do evento=> " , dura)**

**print("Teste utilizando seconds separado=>", seconds)**

**#calculando os segundos de duração diretamente no print sem a variável seconds**

**print("Tempo duração em segundos calculando no print=> ", (dura \* 60))**

**Console**

**Hora de início (horas): 12**

**Hora de início (minutos): 17**

**Duração do evento (minutos): 59**

**13:16**

**Duração do evento=> 59**

**Teste utilizando seconds separado=> 3540**

**Tempo duração em segundos calculando no print=> 3540**

**2.6.12 RESUMO DA SEÇÃO**

1. A função print() envia dados para o console, enquanto a função input() obtém dados do console.

2. A função input() vem com um parâmetro opcional: a string de prompt. Ele permite que você escreva uma mensagem antes da entrada do usuário, por exemplo:

**name = input("Digite seu nome: ")**

**print("Olá, " + name + ". Prazer em conhecê-lo!")**

3. Quando a função input() é chamada, o fluxo do programa é interrompido, o símbolo de prompt fica piscando (ele solicita que o usuário tome medidas quando o console for alternado para o modo de entrada) até que o usuário tenha inserido uma entrada e/ou pressionado o Enter chave.

**Observação**

Você pode testar a funcionalidade da função input() em seu escopo completo localmente em sua máquina. Por motivos de otimização de recursos, limitamos o tempo máximo de execução do programa no Edube a alguns segundos. Vá para Sandbox, copie e cole o snippet acima, execute o programa e não faça nada. Apenas espere alguns segundos para ver o que acontece. Seu programa deve ser interrompido automaticamente após um breve momento. Agora abra o IDLE e execute o mesmo programa. Você consegue ver a diferença?

**Dica:** o recurso mencionado acima da função input() pode ser usado para solicitar que o usuário encerre um programa. Observe o código abaixo:

**name = input("Digite seu nome: ")**

**print("Olá, " + name + ". Prazer em conhecê-lo!")**

**print("\nPressione Enter para finalizar o programa.")**

**input()**

**print("O FIM.")**

4. O resultado da função input() é uma string. Você pode adicionar strings umas às outras usando o operador de concatenação (+). Confira este código:

**num\_1 = input("Digite o primeiro número: ") # Digite 12**

**num\_2 = input("Digite o segundo número: ") # Digite 21**

**print(num\_1 + num\_2) # o programa retorna 1221**

**Você também pode multiplicar cadeias (\* - replicação), por exemplo:**

**my\_input = input("Enter something: ") # Exemplo de entrada: Olá**

**print(my\_input \* 3) # Saída esperada: OláOláOlá**

**Enter something: Olá**

**OláOláOlá**

**2.6.13 TESTE DA SEÇÃO**

Pergunta 1: Qual é a saída do trecho a seguir?

x = int(input("Digite um número: ")) # O usuário digita 2

print(x \* "5")

**Digite um número: 2**

**55**

Pergunta 2: Qual é a saída esperada do trecho a seguir?

x = input("Digite um número: ") # O usuário digita 2

print(type(x))

<class 'str'>

**2.7 Conclusão do Módulo 2 - TESTE DO MÓDULO**

**TESTE DO MÓDULO 2**

Muito bem! Você chegou ao final do Módulo 2 e concluiu um grande marco na sua formação em programação em Python. Aqui está um breve resumo das áreas de tópicos abordadas no Módulo 2:

* como escrever e executar programas Python simples;
* o que são literais, operadores e expressões do Python;
* o que são variáveis e quais são as regras que as regem;
* como executar operações básicas de entrada e saída.

Agora você está pronto para fazer o teste do módulo, o que ajudará a avaliar o que você aprendeu até agora.

O teste a seguir tem como base o que você acabou de aprender. Há vinte perguntas no total e você precisa marcar pelo menos 70% para passar.

Boa sorte!

**Pergunta 1**

Questão de múltipla escolha

O dígrafo \n força a função print() a:

* duplicar o caractere ao lado do dígrafo
* quebrar a linha de saída
* produz exatamente dois caracteres: \ e n
* interromper a execução

**Pergunta 2**

Questão de múltipla escolha

O significado do parâmetro de palavra-chave é determinado por:

* sua posição na lista de argumentos
* sua conexão com variáveis existentes
* seu valor
* o nome do argumento especificado junto com seu valor

SOBRE PALAVRA-CHAVE: <http://www.estruturas.ufpr.br/disciplinas/pos-graduacao/introducao-a-computacao-cientifica-com-python/introducao-python/1-4-funcoes/>

Incompleta Pergunta 3Pergunta 3

Questão de múltipla escolha

O valor vinte ponto doze vezes dez elevado à potência de oito deve ser escrito como:

* 20.12\*10^8
* 20E12.8
* 20.12E8
* 20.12E8.0

**Pergunta 4**

Questão de múltipla escolha

O prefixo 0o significa que o número após ele é indicado como:

* ⁪binário
* hexadecimal
* Octal
* Decimal

Pergunta 5

Questão de múltipla escolha

O operador \*\*:

* não existe
* realiza multiplicação duplicada
* realiza multiplicação de ponto flutuante
* executa exponenciação

Pergunta 6

Questão de múltipla escolha

O resultado da seguinte divisão:

1 / 1

* é igual a 1
* é igual a 1.0
* não pode ser avaliado
* não pode ser previsto

Pergunta 7

Questão de múltipla escolha

Qual das seguintes afirmações são verdadeiras? (Selecione duas respostas)

* O argumento correto do operador % não pode ser zero.
* O resultado do operador / é sempre um valor inteiro.
* A adição precede a multiplicação.
* O operador \*\* usa ligação do lado direito.

Pergunta 8

Questão de múltipla escolha

A associação do lado esquerdo determina que o resultado da seguinte expressão:

1 // 2 \* 3

É igual a

* 4.5
* 0.16666666666666666
* 0,0
* 0

Pergunta 9

Questão de múltipla escolha

Quais dos seguintes nomes de variáveis são ilegais? (Selecione duas respostas)

* True
* true
* TRUE
* and

Pergunta 10

Questão de múltipla escolha

A função print() pode gerar valores de:

* qualquer número de argumentos (excluindo zero)
* não mais de cinco argumentos
* apenas um argumento
* qualquer número de argumentos (incluindo zero)

Pergunta 11

Questão de múltipla escolha

Qual é a saída do seguinte snippet?

x = 1

y = 2

z = x

x = y

y = z

print(x, y)

* 2 2
* 2 1
* 1 2
* 1 1

Pergunta 12

Questão de múltipla escolha

Qual é a saída do seguinte trecho se o usuário digitar duas linhas contendo 2 e 4 respectivamente?

x = input()

y = input()

print(x + y)

* 24
* 2
* 6
* 4

Pergunta 13

Questão de múltipla escolha

Qual é a saída do seguinte trecho se o usuário digitar duas linhas contendo 2 e 4 respectivamente?

x = int(input())

y = int(input())

x = x // y

y = y // x

print(y)

* o código causará um erro em tempo de execução
* 2.0
* 8.0
* 4,0

Pergunta 14

Questão de múltipla escolha

Qual é a saída do seguinte trecho se o usuário digitar duas linhas contendo 2 e 4 respectivamente?

x = int(input())

y = int(input())

x = x / y

y = y / x

print(y)

* 4,0
* 8.0
* 2.0
* o código causará um erro em tempo de execução

Pergunta 15

Questão de múltipla escolha

Qual é a saída do seguinte snippet se o usuário digitar duas linhas contendo 11 e 4 respectivamente?

x = int(input())

y = int(input())

x = x % y

x = x % y

y = y % x

print(y)

* 1
* 2
* 3
* 4

Pergunta 16

Questão de múltipla escolha

Qual é a saída do seguinte snippet se o usuário digitar duas linhas contendo 3 e 6 respectivamente?

x = input()

y = int(input())

print(x \* y)

* 666
* 333333
* 18
* 36

Pergunta 17

Questão de múltipla escolha

Qual é a saída do seguinte snippet?

z = y = x = 1

print(x, y, z, sep='\*')

* 1 1 1
* 1\*1\*1
* x y z
* x\*y\*z

Pergunta 18

Questão de múltipla escolha

Qual é a saída do seguinte snippet?

y = 2 + 3 \* 5.

print(Y)

* 25.
* 17,0
* o snippet causará um erro de execução
* 17

**Pergunta 19**

Questão de múltipla escolha

Qual é a saída do seguinte snippet?

x = 1 / 2 + 3 // 3 + 4 \*\* 2

print(x)

* 8,5
* 8
* 17,5
* 17

**Pergunta 20**

Questão de múltipla escolha

Qual é a saída do seguinte trecho se o usuário digitar duas linhas contendo 2 e 4 respectivamente?

x = int(input())

y = int(input())

print(x + y)

* 24
* 4
* 6
* 2

Clique em 'Enviar' se estiver pronto para receber a pontuação do teste.

**3.1 Seção 1 - Tomada de decisões em Python**

Bem-vindo ao módulo três! Na primeira seção, aprenderemos sobre declarações condicional e como usá-las para tomar decisões em Python.

**3.1.1 Perguntas e respostas**

Um programador cria um programa e o **programa faz perguntas**.

Um computador executa o programa e **fornece as respostas**. O programa deve ser capaz de reagir de acordo com as respostas recebidas.

Felizmente, os computadores sabem apenas dois tipos de respostas:

* Sim, isso é verdade.
* não, isso é falso.

Você nunca receberá uma resposta como Deixe-me pensar ...,, Não sei, ou provavelmente sim, mas não tenho certeza.

Para fazer perguntas, o **Python usa um conjunto de operadores muito especiais**. Vamos examiná-los um após o outro, ilustrando seus efeitos em alguns exemplos simples.

**3.1.2 Comparação: operador de igualdade**

Pergunta: os dois valores são iguais?

Para fazer essa pergunta, você usa o operador == (igual igual).

Não se esqueça dessa importante distinção:

* = é um operador de atribuição, por exemplo, a = b **atribui** a com o valor de b;
* == A pergunta é: ***esses valores são iguais?*** então a == b c**ompara** a e b.

É um operador binário com ligação do lado esquerdo. Ela precisa de dois argumentos e verifica se são iguais.

**3.1.3 Exercícios**

Agora, vamos fazer algumas perguntas. Tente adivinhar as respostas.

Pregunta #1: Qual é o resultado da comparação a seguir?

2 == 2

**True** - claro, 2 é igual a 2. Python responderá True (lembre-se deste par de literais predefinidos, True e False - eles também são palavras-chave do Python).

Pregunta #2: Qual é o resultado da comparação a seguir?

2 == 2.

Esta questão não é tão fácil quanto a primeira. Felizmente, o Python é capaz de converter o valor inteiro em seu equivalente real e, consequentemente, a resposta é **True.**

Pregunta #3: Qual é o resultado da comparação a seguir?

1 == 2

Isso deve ser fácil. A resposta será (ou melhor, sempre é) False.

**3.1.4 Operadores**

**Igualdade: o operador de igualdade (==)**

O operador == (igual a) compara os valores de dois operandos. Se forem iguais, o resultado da comparação é True. Se eles não forem iguais, o resultado da comparação é False.

Veja a comparação de igualdade abaixo - qual é o resultado dessa operação?

var == 0

Observe que não podemos encontrar a resposta se não soubermos qual valor está armazenado no momento na variável var.

Se a variável tiver sido alterada várias vezes durante a execução do programa ou o valor inicial for inserido no console, a resposta para essa pergunta pode ser dada apenas pelo Python e apenas no tempo de execução.

Agora imagine um programador que sofre de insônia e precisa contar ovelhas preto e branco separadamente, desde que haja exatamente o dobro da ovelha negra que a branca.

A pergunta será a seguinte:

**black\_sheep == 2 \* white\_sheep**

Devido à baixa prioridade do operador ==, a pergunta deve ser tratada como equivalente a esta:

**black\_sheep == (2 \* white\_sheep)**

Então, vamos praticar sua compreensão do operador == agora - você consegue adivinhar a saída do código abaixo?

**var = 0 # Atribuindo 0 a var**

**print(var == 0)**

**var = 1 # Atribuindo 1 a var**

**print(var == 0)**

**Console**

**True**

**False**

**Desigualdade: o operador de desigualdade (!=)**

O operador != (Não é igual a) também compara os valores de dois operandos. Aqui está a diferença: se eles são iguais, o resultado da comparação é False. Se eles não forem iguais, o resultado da comparação é True.

Agora, dê uma olhada na comparação de desigualdade abaixo. Você consegue adivinhar o resultado dessa operação?

var = 0 # Atribuindo 0 a var

print(var != 0)

var = 1 # Atribuindo 1 a var

print(var != 0)

False

True

Execute o código e verifique se você está certo.

**Operadores de comparação: maior que**

Você também pode fazer uma pergunta comparativa usando **o >** (maior que).

Se quiser saber se há mais ovelhas negras do que brancas, escreva-as da seguinte forma:

black\_sheep > white\_sheep # Maior que

True confirma isso; False nega.

O código seria assim, medindo o o cumprimento e tamanho de uma string em seus caracteres:

**print (str("black\_sheep" > "white\_sheep"))**

**False**

**Operadores de comparação: maior ou igual a**

O operador maior que tem outra variante especial, não estrita, mas é denotado de forma diferente em notação aritmética clássica: **>=** (maior que ou igual a).

Há dois sinais subsequentes, não um.

Ambos os operadores (estrito e não-estrito), bem como os outros dois discutidos na próxima seção, ***são operadores binários com ligação do lado esquerdo***, e sua prioridade é maior do que a mostrada por **==** e **!=**.

Se quisermos descobrir se temos ou não que usar um chapéu quente, fazemos a seguinte pergunta:

**Comparação de operadores: menor que e menor ou igual que**

Como você provavelmente já adivinhou, os operadores usados neste caso são: o < (menor que) operador e seu irmão não rígido: **<=** (menor que ou igual a).

Veja este exemplo simples:

**current\_velocity\_mph < 85 # Menor que**

**current\_velocity\_mph <= 85 # Menos que ou igual a**

Vamos verificar se há o risco de ser multado pela polícia de trânsito (a primeira pergunta é rigorosa, a segunda não).

**3.1.5 Utilização das respostas**

O que você pode fazer com a resposta (ou seja, o resultado de uma operação de comparação) que você obtém do computador?

Há pelo menos duas possibilidades: primeiro, você pode memorizá-la (armazená-la em uma variável) e usá-la posteriormente. Como você faz isso? Bem, você usa uma variável arbitrária como esta:

**answer = number\_of\_lions >= number\_of\_lionesses**

O conteúdo da variável indicará a resposta para a pergunta.

Executando o código para teste:

answer = str("number\_of\_lions" >= "number\_of\_lionesses")

print (answer) # Maior ou igual a

True

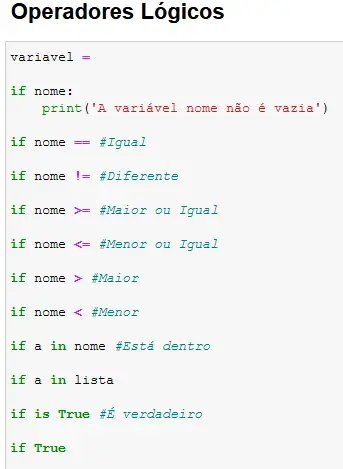
A segunda possibilidade é mais conveniente e muito mais comum: você pode usar a resposta para **tomar uma decisão sobre o futuro do programa**.

Você precisa de uma instrução especial para esse fim, e nós a discutiremos em breve.

Agora precisamos atualizar nossa **tabela de prioridades** e colocar todos os novos operadores nela. Agora, ele tem a seguinte aparência:

**TABELA DE PRIORIDADES**

|  |  |  |
| --- | --- | --- |
| **Prioridade** | **Operadora** |  |
| 1 | **+, -** | unário |
| 2 | **\*\*** |
| 3 | **\*, /, //, %** |
| 4 | **+, -** | ⁪binário |
| 5 | **<, <=, >, >=** |
| 6 | **==, !=** |



**3.1.6 LAB Variáveis ‒ Perguntas e respostas**

Cenário

Usando um dos operadores de comparação em Python, escreva um programa simples de duas linhas que aceita o parâmetro n como entrada, que é um inteiro, e imprime False se n for menor 100, e True se n for maior ou igual a 100.

Não crie blocos if (falaremos sobre eles em breve). Teste seu código usando os dados que fornecemos para você.

**Dados de teste**

**Exemplo de entrada:**

55

**Saída prevista:**

False

**Output**

**Exemplo de entrada:**

99

**Saída prevista:**

False

**Output**

**Exemplo de entrada:**

100

**Saída prevista:**

True

**Output**

**Exemplo de entrada:**

101

**Saída prevista:**

True

**Output**

**Exemplo de entrada:**

-5

**Saída prevista:**

False

**Output**

**Exemplo de entrada:**

+123

**Saída prevista:**

True

**Output**

imprime False se n for menor 100, e True se n for maior ou igual a 100

**n = int(input("Insira um dos números definidos :"))**

**# imprime False se n for menor 100, e True se n for maior ou igual a 100**

**print (n >= 100)**

**Console**

**Insira um dos números definidos :55**

**False**

**Insira um dos números definidos :99**

**False**

**Insira um dos números definidos :100**

**True**

**Insira um dos números definidos :101**

**True**

**Insira um dos números definidos :-5**

**False**

**Insira um dos números definidos :+123**

**True**

**3.1.7 Condições e execução condicional**

Você já sabe fazer perguntas em Python, mas ainda não sabe como fazer uso racional das respostas. Você precisa ter um mecanismo que permita que você faça algo se uma condição for atendida, e não faça se não for.

**É como na vida real:** você faz certas coisas ou não quando uma condição específica é atendida ou não, por exemplo, você pode passear se o tempo estiver bom ou ficar em casa se estiver chuvoso e frio.

Para tomar essas decisões, o Python oferece uma instrução especial. Devido à sua natureza e aplicação, é chamada de **instrução condicional** (ou declaração condicional).

Existem várias variantes. Vamos começar pelo mais simples, aumentando a dificuldade lentamente.

A primeira forma de uma declaração condicional, que você pode ver abaixo é escrita de forma muito informal, mas figurativa:

if true\_or\_not:

do\_this\_if\_true

Essa declaração condicional consiste nos seguintes elementos, estritamente necessários, somente nesta e nesta ordem:

* a palavra-chave **if**;
* **um** ou mais **espaços em branco**;
* uma expressão (uma pergunta ou uma resposta) cujo valor será interpretado apenas em termos de **True** (quando seu valor for diferente de zero) e **False** (quando for igual a zero);
* **dois pontos** seguidos por u**ma nova linha**;
* uma instrução recuada ou um conjunto de instruções (pelo menos uma instrução é absolutamente necessária); o recuo pode ser obtido de duas maneiras: inserindo um número específico de espaços (a recomendação é **usar quatro espaços de recuo**) ou usando o caractere de tabulação; Nota: se houver mais de uma instrução na peça recuada, a indentação deve ser a mesma em todas as linhas; mesmo que você use as guias misturadas com os espaços, é importante tornar todos os recuos exatamente iguais - o Python 3 não permite a mistura de espaços e guias para o recuo.

**Como essa declaração funciona?**

* Se a expressão **true\_or\_not** representar a verdade (ou seja, seu valor não for igual a zero), as instruções recuadas serão executadas;
* se a expressão **true\_or\_not** não representa a verdade (ou seja, seu valor é igual a zero), as instruções recuadas serão omitidas (ignoradas) e a próxima instrução executada será a seguinte ao nível de recuo original.

Na vida real, muitas vezes expressamos um desejo:

***se o tempo estiver bom, vamos dar uma volta***

***então, vamos almoçar***

Como você pode ver, almoçar não **é uma** **atividade condicional** e não depende do clima.

Sabendo quais condições influenciam nosso comportamento e assumindo que temos as funções sem parâmetros go\_for\_a\_walk() e have\_lunch(), podemos escrever o seguinte trecho:

if the\_weather\_is\_good:

go\_for\_a\_walk()

have\_lunch()

**Execução condicional: o comando if**

Se um desenvolvedor Python sem sono adormecer quando contar 120 ovelhas, e o procedimento de indução do sono puder ser implementado como uma função especial chamada **sleep\_and\_dream(),** todo o código assumirá a seguinte forma:

if sheep\_counter >= 120: # Avaliar uma expressão de teste

sleep\_and\_dream() # Execute se a expressão de teste for verdadeira

Você pode lê-lo como: se **sheep\_counter** for maior ou igual a **120**, adormeça e sonhe (ou seja, execute a função **sleep\_and\_dream**).

Dissemos que as instruções **executadas condicionalmente devem ser recuadas**. Isso cria uma estrutura muito legível, demonstrando claramente todos os caminhos de execução possíveis no código.

Veja o seguinte código:

**if sheep\_counter >= 120:**

**make\_a\_bed()**

**take\_a\_shower()**

**sleep\_and\_dream()**

**feed\_the\_sheepdogs()**

Como você pode ver, arrumar a cama, tomar um banho e adormecer e sonhar são **executados condicionalmente** - quando o **sheep\_counter** atinge o limite desejado.

Alimentar os cães pastores, no entanto, é **sempre feito** (ou seja, a função **feed\_the\_sheepdogs()** não é recuada e não pertence ao bloco **if**, o que significa que é sempre executado.)

Agora, vamos discutir outra variante da declaração condicional, que também permite que você execute uma ação adicional quando a condição não for atendida.

**Execução condicional: o comando if-else**

Começamos com uma frase simples que diz: ***Se o tempo estiver bom, vamos dar um passeio***.

**Nota:** não há uma palavra sobre o que acontecerá se o tempo estiver ruim. Sabemos apenas que não vamos ao ar livre, mas o que poderíamos fazer, em vez disso, é desconhecido. Podemos também querer planejar algo em caso de mau tempo.

Podemos dizer, por exemplo: ***se o tempo estiver bom, vamos dar um passeio, caso contrário, vamos a um teatro***.

Agora sabemos o que faremos se as **condições forem atendidas**, e o que faremos se não **tudo seguir nosso caminho**. Em outras palavras, temos um "Plano B".

O Python nos permite expressar esses planos alternativos. Isso é feito com uma segunda forma um pouco mais complexa da instrução condicional, a instrução **if-else**:

**if true\_or\_false\_condition:**

**perform\_if\_condition\_true**

**else:**

**perform\_if\_condition\_false**

Assim, há uma nova palavra: **else** - esta é uma **palavra-chave**.

A parte do código que começa com **else** diz o que fazer se a condição especificada para o **if** não for atendida (observe os dois pontos após a palavra).

A execução se-senão é a seguinte:

* se a condição for avaliada como **True** (seu valor não é igual a zero), a instrução **perform\_if\_condition\_true** é executada e a instrução condicional chega ao fim;
* se a condição for avaliada como **False** (é igual a zero), a instrução **perform\_if\_condition\_false** é executada e a instrução condicional chega ao fim.

**O comando if-else: mais execução condicional**

A estrutura condicional **if** é uma palavra inglesa que na tradução para o português é “**se”,** ou seja uma condicional. Já **else** também vem do inglês e em português é **“Caso contrário”** ou **“oposto”** ou seja algo para ser feito caso contrário a uma condição que é verificada pelo “se” (if).

Ao usar esta forma de declaração condicional, podemos descrever nossos planos da seguinte forma:

**if the\_weather\_is\_good:** # Se o tempo estiver bom

**go\_for\_a\_walk()** # vamos dar um passeio

**else:** # Caso contrário

**go\_to\_a\_theater()** # vamos ao cinema

**have\_lunch()** # de qualquer forma almoçaremos depois do passeio ou depois de ir ao cinema

Se o tempo estiver bom, vamos dar um passeio. Caso contrário, vamos ao cinema. Não importa se o tempo estiver bom ou ruim, almoçaremos depois (depois da caminhada ou depois de ir ao cinema).

Tudo o que dissemos sobre a indentação funciona da mesma maneira dentro do ramo else:

if the\_weather\_is\_good:

go\_for\_a\_walk()

have\_fun()

else:

go\_to\_a\_theater()

enjoy\_the\_movie()

have\_lunch()

**Comandos if-else aninhados**

Agora vamos discutir dois casos especiais da declaração condicional.

Primeiro, considere o caso em que a instrução colocada após o **if** é outra if.

Leia o que planejamos para este domingo. Se o tempo estiver bom, vamos dar uma volta. Se encontrarmos um bom restaurante, almoçaremos lá. Caso contrário, vamos comer um sanduíche. Se o tempo estiver ruim, vamos ao teatro. Se não houver ingressos, faremos compras no shopping mais próximo.

Vamos escrever o mesmo em Python. Considere com cuidado o código aqui:

**if the\_weather\_is\_good:**

**if nice\_restaurant\_is\_found:**

**have\_lunch()**

**else:**

**eat\_a\_sandwich()**

**else:**

**if tickets\_are\_available:**

**go\_to\_the\_theater()**

**else:**

**go\_shopping()**

Aqui estão dois pontos importantes:

* esse uso da instrução if é conhecido como aninhamento; lembre-se de que todo else se refere ao if que está no mesmo nível de indentação; você precisa saber disso para determinar como os ifs e elses se combinam;
* considere como o recuo melhora a legibilidade e torna o código mais fácil de entender e rastrear.

**O comando elif**

*O segundo caso especial* apresenta outra nova palavra-chave Python: **elif**. Como você provavelmente suspeita, é uma forma mais **else if** senão.

**elif** é usado para ***verificar mais de uma condição e parar quando a primeira declaração verdadeira é encontrada***.

Nosso próximo exemplo é semelhante ao aninhamento, mas as semelhanças são muito pequenas. Novamente, vamos mudar nossos planos e expressá-los da seguinte forma: Se o tempo estiver bom, vamos dar uma volta, caso contrário, se conseguirmos ingressos, vamos ao teatro, caso contrário, se houver mesas gratuitas no restaurante, vamos almoçar; se tudo mais falhar, ficaremos em casa e jogaremos xadrez.

Você já reparou quantas vezes usamos essa palavra? Este é o estágio em que a palavra-chave **elif** desempenha seu papel.

Vamos escrever o mesmo cenário usando Python:

**if the\_weather\_is\_good:**

**go\_for\_a\_walk()**

**elif tickets\_are\_available:**

**go\_to\_the\_theater()**

**elif table\_is\_available:**

**go\_for\_lunch()**

**else:**

**play\_chess\_at\_home()**

A maneira de montar instruções **if-elif-else** subsequentes é ***chamada de cascata***.

Observe novamente como o recuo melhora a legibilidade do código.

Alguma atenção adicional deve ser prestada neste caso:

* você ***não deve usar else sem preceder if***;
* **else é sempre o último ramo da cascata**, independentemente de você ter usado elif ou não;
* ***else é uma parte opcional da cascata e pode ser omitida***;
* se houver ***uma else na cascata, apenas uma de todas as ramificações será executada****;*
* se ***não houver a ramificação else, é possível que nenhuma das outras ramificações sejam executadas***;

Isso pode parecer um pouco confuso, mas espero que alguns exemplos simples ajudem a esclarecer melhor.

**ESTRUTURAS CONDICIONAIS NO PYTHON: COMO UTILIZÁ-LAS?**

## Estrutura condicional Python if-else

Uitilizações de identação e estruturas: if, else, if-else, elif, if com for,

<https://panda.ime.usp.br/cc110/static/cc110/05-if.html>

**Execução condicional e alternativas: if, if-else e if-elif-else**

## Tópicos e Objetivos

Ao final dessa aula você deverá saber:

* Usar o comando de execução condicional if;
* Usar e identificar situações onde a utilização do comando if-else é mais adequada.
* Usar e identificar situações onde a utilização do comando if-elif-else é mais adequada.
* Usar o operador de divisão inteira //
* Usar o operador de resto de divisão %

## Introdução

O comando if (que significa se em português) permite que uma parte do programa seja executada apenas quando uma condição for verdadeira. A sintaxe do comando ìf é a seguinte:

**if** condição\_do\_if:

*# bloco executado se a condição for verdadeira*

comando\_1

comando\_2

...

comando\_m *# último comando do bloco*

comando\_após\_if

Apenas caso a condição\_do\_if for verdadeira, o bloco contendo os comandos comando\_1 a comando\_m é executado. Caso contrário, esse bloco não é executado e a execução do programa continua com o comando comando\_após\_if.

Dessa forma, o if permite resolver problemas do tipo:

idade = int(input("Digite sua idade: "))

if idade >= 18:

print("Parabéns, voce ja tem idade para tirar a sua carteira.")

print("fim.")

(Exemplo\_de\_uso\_do\_comando\_if)

**Como funciona esse trecho?**

Após ler a idade (linha 1), o programa testa se a idade é maior ou igual a 18 na linha 3. Se a condição for verdadeira, o programa imprime a mensagem de parabéns (linha 4) e depois imprime “fim”. Caso contrário, o programa pula o bloco dentro do if e imprime “fim” direto.

### Comando if-else

Em várias ocasiões é necessário executar blocos de forma alternativa. Nesses casos, podemos utilizar o comando if-else (que significa se-senão em português), cuja sintaxe é a seguinte:

**if** condição:

*# bloco contendo comandos a serem executados*

dentro\_do\_if\_1

dentro\_do\_if\_2

...

dentro\_do\_if\_m

**else**:

dentro\_do\_else\_1

dentro\_do\_else\_2

...

dentro\_do\_else\_n

comando\_apos\_if

Apenas caso a condição for verdadeira, o bloco contendo os comandos comando\_1 a comando\_n é executado. Caso contrário, esse bloco não é executado e a execução do programa continua com o comando comando\_apos\_if.

Esse comando nos ajuda a resolver problemas como, por exemplo, determinar se um número é par ou ímpar:

**n = int(input("Digite um numero: "))**

**if n % 2 == 0: # se n é múltiplo de 2**

**print(n, "é par")**

**if n % 2 != 0: # se n não é múltiplo de 2**

**print(n , "é impar")**

**print("fim.")**

(Exemplo\_ruim\_de\_uso\_do\_if)

**Como funciona esse trecho?**

Após n receber um número, o programa testa, na linha 3, se n é múltiplo de 2 usando o operador aritmético %. Esse operador determina o resto da divisão de n por 2. Caso a condição seja verdadeira (= True), o programa imprime que o número é par. Na linha 5 o programa recalcula o resto da divisão de n por 2 e verifica se o resto é diferente de zero, ou seja, se o número não é múltiplo de 2 e, nesse caso, imprime que o número é ímpar.

Observe que a condição dos ifs nas linhas 3 e 5 foram executadas, mesmo quando na linha 3 o número já foi identificado como par, o teste da linha 5 é realizado.

Embora essa solução produza resultados corretos, o uso do comando if-else torna o programa bem mais elegante e seguro:

**n = int(input("Digite um numero: "))**

**if n % 2 == 0: # se n é múltiplo de 2 então é par**

**print(n, "é par")**

**else: # senão o número é ímpar, não precisa fazer outro teste!**

**print(n, "é impar")**

**print("fim.")**

(Exemplo\_de\_uso\_do\_if-else)

**Como funciona esse trecho?**

Esse trecho usando if-else é muito semelhante ao anterior usando dois ifs. A grande diferença agora é que na linha 5 temos um else. Nesse caso, a condição do if é testada apenas uma vez e, caso verdadeira, o programa imprime que o número é par e, caso contrário, o programa vai direto para o bloco dentro do else, imprimindo que o número é ímpar, sem precisar perder tempo calculando outro teste.

Essa construção é mais elegante pois evita a computação de “n não é múltiplo de 2” e torna o programa mais fácil de ler, corrigir e modificar.

Um if-else é utilizado quando temos apenas duas alterativas. Quando o número de alternativas é maior, podemos aninhar comandos if-else. Por exemplo, considere o problema de ler a nota de um aluno para verificar se ele está reprovado, está de recuperação ou foi aprovado. Suponha que as notas são números inteiros entre 0 e 100. Um aluno está reprovado se sua nota e menor que 30, está de recuperação se sua nota é um inteiro entre 30 e 49 e está aprovado se sua nota é pelo menos 50. Uma solução aninhando comandos if-else seria:

**nota = int(input("Digite uma nota: "))**

**if nota < 30:**

**print("Reprovado")**

**else:**

**if nota < 50:**

**print("Recuperacao")**

**else:**

**print("Aprovado")**

**print("fim.")**

(Exemplo\_de\_uso\_do\_if-else\_aninhados)

**Como funciona esse trecho?**

Observe com atenção a tabulação desse programa, que mostra um if-else na mesma coluna da linha 1 e, dentro do primeiro else, vemos um outro comando if-else. Por esse deslocamento sabemos que o segundo if só será testado caso a condição do primeiro if (da linha 3) seja falsa.

Vamos simular esse trecho usando 3 testes: nota 2 (que deve resultar em reprovado), nota 4 (que deve resultar em recuperação) e nota 7 que deve resultar em aprovado.

Quando a nota é 2, a condição do primeiro if é verdadeira e assim o programa imprime “Reprovado”. Quando a condição do if é verdadeira o else correspondente não é executado, e o programa termina imprimindo “fim”.

Quando a nota é 4, a condição do primeiro if é falsa e o bloco dentro do else correspondente (que está na mesma coluna desse if) é executado, ou seja, o segundo if-else é executado. Para isso a condição do if na linha 6 é testada, resultando em verdadeiro. O programa imprime então “Recuperação”, o else correspondente não é executado e o programa termina com “fim”.

Finalmente quando a nota é 7, a condição dos dois ifs é falsa e, nesse caso, o bloco dentro do segundo else (linha 9) é executado. Assim o programa imprime “Aprovado” e em seguida imprime “fim”.

Observe que esse mesmo trecho de código poderia ser escrito apenas com ifs da seguinte forma:

**nota = int(input("Digite uma nota: "))**

**if nota < 30:**

**print("Reprovado")**

**if nota >= 30:**

**if nota < 50:**

**print("Recuperacao")**

**if nota >= 50:**

**print("Aprovado")**

**print("fim.")**

(Não\_utilize\_if\_para\_substituir\_if-else)

**Dica de estilo**

Embora seja possível utilizar apenas ifs, o uso de if-else é bem mais elegante pois torna o código mais simples, fácil de entender, de encontrar erros e de corrigi-los. Por esses motivos, sempre que possível, prefira usar if-else ao invés de apenas ifs.

### Comando if-elif-else

Para simplificar ainda mais o código de programas com if-else aninhados, o Python oferece o comando if-elif-else.

Usando esse comando poderíamos escrever o programa para saber a condição de um aluno da seguinte forma:

Se digitarmos: 90

**nota = int(input("Digite uma nota: "))**

**if nota < 30:**

**print("Reprovado")**

**elif nota < 50:**

**print("Recuperacao")**

**else:**

**print("Aprovado")**

**print("fim.")**

Aprovado  
fim.

ActiveCode (Exemplo\_de\_uso\_do\_if-elif-else)

O elif portanto é apenas uma contração do else if que torna mais claro o tratamento das várias alternativas, encadeando as condições. Blocos de elif podem ser repetido várias vezes. Suponha por exemplo que gostaríamos de conhecer os alunos aprovados com louvor, ou seja, com nota superior a 90. Nesse caso, o código seria o seguinte.

Se digitarmos uma nota 60 por exemplo:

**nota = int(input("Digite uma nota: "))**

**if nota < 30:**

**print("Reprovado")**

**elif nota < 50:**

**print("Recuperacao")**

**elif nota < 90:**

**print("Aprovado")**

**else:**

**print("Aprovado com louvor!")**

**print("fim.")**

Aprovado  
fim.

(Outro\_exemplo\_de\_uso\_do\_if-elif-else)

A sintaxe do comando if-elif-else é a seguinte:

**if** condição:

*# bloco contendo comandos a serem executados*

dentro\_do\_if\_1

dentro\_do\_if\_2

...

dentro\_do\_if\_p

**elif** condição elif1:

dentro\_do\_elif1\_1

dentro\_do\_elif1\_2

...

dentro\_do\_elif1\_m1

**elif** condição elif2:

dentro\_do\_elif2\_1

dentro\_do\_elif2\_2

...

dentro\_do\_elif2\_m2

*# outros possíveis blocos de elif, cada um com a sua condição*

**elif** condição elifk:

dentro\_do\_elifk\_1

dentro\_do\_elifk\_2

...

dentro\_do\_elifk\_mk

**else**:

dentro\_do\_else\_1

dentro\_do\_else\_2

...

dentro\_do\_else\_q

comando\_apos\_if

Ou seja, o if-elif-else pode ter um ou mais blocos com elif, cada um com a sua condição específica. Cada bloco (condição) é testado um de cada vez, até que uma condição seja satisfeita e apenas os comandos dentro desse bloco são executados. Quando nenhuma condição é satisfeita, os comandos dentro do else são executados. Assim, **não** é necessário que haja um else no final.

Faça experimentos com o trecho de código a seguir.

**n = 25 # altere esse valor e simule esse trecho novamente**

**if n < 0:**

**print(n, "< 0")**

**elif n == 0:**

**print(n, "== 0")**

**elif n < 10:**

**print("0 < ", n,"< 10")**

**elif n < 20:**

**print("10 <=", n, "< 20")**

**elif n < 30:**

**print("20 <=", n, "< 30")**

**elif n < 50:**

**print("30 <=", n, "< 50")**

**else:**

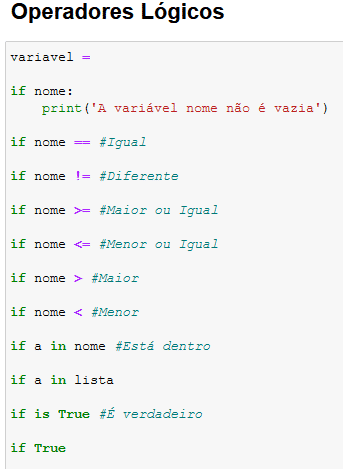
**print("50 <=", n)**

**print("Termino do teste.")**

20 <= 25 < 30  
Termino do teste.

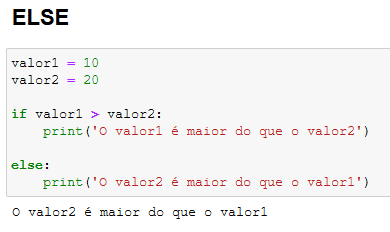
(condicao\_em\_cadeia)

**EXEMPLOS DE OPERADORES**

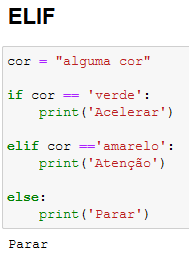


Operadores Lógicos no Python

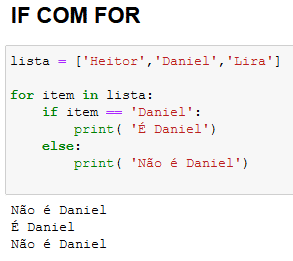
| **Nível** | **Categoria** | **Operadores** |
| --- | --- | --- |
| 7(alto) | exponenciação | \*\* |
| 6 | multiplicação | \*,/,//,% |
| 5 | adição | +,- |
| 4 | relacional | ==,!=,<=,>=,>,< |
| 3 | lógico | not |
| 2 | lógico | and |
| 1(baixo) | lógico | or |



Estrutura ELSE



Estrutura ELIF



Estrutura IF com FOR

**num = int(input("Digite um número: "))**

**while num != 0:**

**quadrado = num \* num**

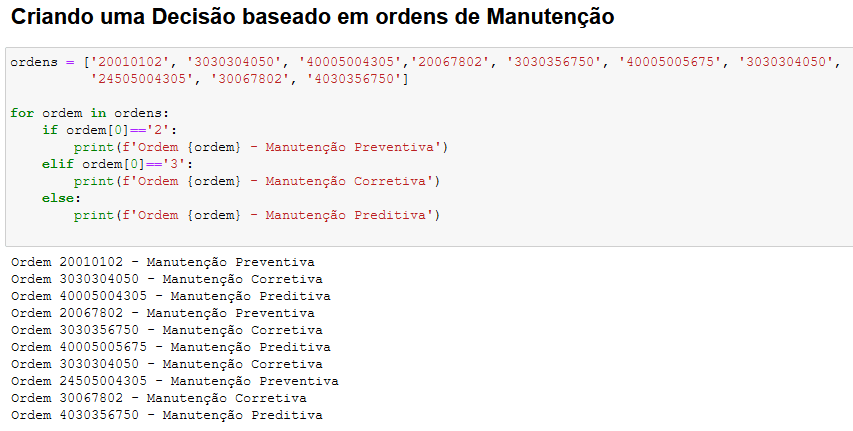
**print(num, "ao quadrado é", quadrado)**

**num = int(input("Digite um número: "))**

**print("Cálculo dos quadrados de uma sequência de números.")**

**print("A sequência termina com um 0 (zero).\n")**

10 ao quadrado é 100  
4 ao quadrado é 16  
2 ao quadrado é 4  
5 ao quadrado é 25

Junção das estruturas par um exemplo prático

**3.1.8 Análise de amostras de código**

Agora vamos mostrar alguns programas simples, mas completos. Não vamos explicar detalhadamente, porque consideramos os comentários (e os nomes das variáveis) dentro do código como guias suficientes.

Todos os programas resolvem o mesmo problema: eles encontram o maior de vários números e o imprimem.

Exemplo 1:

Vamos começar com o caso mais simples: como identificar o maior de dois números:

**#Ler dois números**

**number1 = int(input("Digite o primeiro número: "))**

**number2 = int(input("Digite o segundo número: "))**

**# Escolha o número maior**

**if number1 > number2:**

**larger\_number = number1**

**else:**

**larger\_number = number2**

**# Imprimir o resultado**

**print("O maior número é:", larger\_number)**

**Console**

**Digite o primeiro número: 5**

**Digite o segundo número: 10**

**O maior número é: 10**

O trecho acima deve ser claro: ele lê dois valores inteiros, os compara e descobre qual é o maior.

Exemplo 2:

Agora, vamos mostrar um fato intrigante. Python tem um recurso interessante - veja o código abaixo:

#Ler dois números

number1 = int(input("Digite o primeiro número: "))

number2 = int(input("Digite o segundo número: "))

# Escolha o número maior

if number1 > number2: larger\_number = number1

else: larger\_number = number2

# Imprimir o resultado

print("O maior número é:", larger\_number)

**Console**

**Digite o primeiro número: 10**

**Digite o segundo número: 5**

**O maior número é: 10**

Nota: se qualquer uma das ramificações if-elif-else contiver apenas uma instrução, você poderá codificá-la de forma mais abrangente (não é necessário criar uma linha recuada após a palavra-chave, mas apenas continuar a linha após os dois pontos) .

Esse estilo, no entanto, pode ser enganoso, e não vamos usá-lo em nossos programas futuros, mas definitivamente vale a pena saber se você deseja ler e entender os programas de outras pessoas.

Não há outras diferenças no código.

Exemplo 3:

É hora de complicar o código - vamos encontrar o maior de três números. Será que vai ampliar o código? Um pouco.

Supomos que o primeiro valor seja o maior. Em seguida, verificamos essa hipótese com os dois valores restantes.

Veja o código abaixo:

# Leia três números

number1 = int(input("Digite o primeiro número: "))

number2 = int(input("Digite o segundo número: "))

number3 = int(input("Digite o terceiro número: "))

# Assumimos temporariamente que o primeiro número

# é o maior deles.

# Em breve verificaremos isso.

largest\_number = number1

# Nós verificamos se o segundo número é maior que o maior\_número atual

# e atualize o maior\_número, se necessário.

if number2 > largest\_number:

largest\_number = number2

# Nós verificamos se o terceiro número é maior que o maior\_número atual

# e atualize o maior\_número, se necessário.

if number3 > largest\_number:

largest\_number = number3

# Imprimir o resultado

print("O maior número é:", largest\_number)

**Console**

**Digite o primeiro número: 1**

**Digite o segundo número: 5**

**Digite o terceiro número: 10**

**O maior número é: 10**

Esse método é significativamente mais simples do que tentar encontrar o maior número de uma só vez, ao comparar todos os pares possíveis de números (ou seja, primeiro com o segundo, segundo com o terceiro, terceiro com o primeiro). Tente recriar o código por conta própria.

**# LEIA QUATRO NUMEROS**

**numero1 = int(input("Digite o primeiro número: "))**

**numero2 = int(input("Digite o segundo número: "))**

**numero3 = int(input("Digite o terceiro número: "))**

**numero4 = int(input("Digite o terceiro número: "))**

**#AQUI DEFINIMOS QUE O NUMERO 1 É O MAIOR NUMERO**

**maior\_numero = numero1**

**#AQUI VEMOS DE É MAIOR QUE O MAIOR\_NUMERO, SE FOR E VAI SEGUINDO O CÓDIGO**

**# USANDO O MAIOR ATUALIZADO**

**if numero2 > maior\_numero:**

**maior\_numero = numero2**

**#IDEM AO DE CIMA**

**if numero3 > maior\_numero:**

**maior\_numero = numero3**

**#IDEM AO DE CIMA**

**if numero4 > maior\_numero:**

**maior\_numero = numero4**

**# Imprimir o resultado**

**print("O maior número finalmente é:", maior\_numero)**

**Console**

**Digite o primeiro número: 10**

**Digite o segundo número: 11**

**Digite o terceiro número: 5**

**Digite o terceiro número: 2**

**O maior número finalmente é: 11**

# LEIA QUATRO NUMEROS

numero1 = int(input("Digite o primeiro número: "))

numero2 = int(input("Digite o segundo número: "))

numero3 = int(input("Digite o terceiro número: "))

numero4 = int(input("Digite o terceiro número: "))

print("--------Resultados-----------")

#AQUI DEFINIMOS QUE O NUMERO 1 É O MAIOR NUMERO

maior\_numero = numero1

print("\nResultado do primeiro numero :", numero1)

#AQUI VEMOS DE É MAIOR QUE O MAIOR\_NUMERO, SE FOR E VAI SEGUINDO O CÓDIGO

# USANDO O MAIOR ATUALIZADO

if (numero2 + maior\_numero) > maior\_numero:

maior\_numero = numero2

print("Resultado do segundo maior numero :", maior\_numero)

#IDEM AO DE CIMA

if (numero3 - numero2) > maior\_numero:

maior\_numero = numero3

print("Resultado do terceiro maior numero :", maior\_numero)

#IDEM AO DE CIMA

if (numero4 + numero3 - maior\_numero) > maior\_numero:

maior\_numero = numero4

print("Resultado do quarto maior número :", maior\_numero)

print("--------FINALMENTE CONCLUÍMOS-----------")

print("\n Que o maior número finalmente é:", maior\_numero)

**Console**

**Digite o primeiro número: 3**

**Digite o segundo número: 10**

**Digite o terceiro número: 1**

**Digite o terceiro número: 15**

**--------Resultados-----------**

**Resultado do primeiro numero : 3**

**Resultado do segundo maior numero : 10**

**Resultado do terceiro maior numero : 10**

**Resultado do quarto maior número : 10**

**--------FINALMENTE CONCLUÍMOS-----------**

**Que o maior número finalmente é: 10**

**3.1.9 Pseudocódigo e introdução aos loops**

Agora você deve ser capaz de escrever um programa que encontre o maior de quatro, cinco, seis ou até mesmo dez números.

Você já conhece o esquema, então estender o tamanho do problema não será particularmente complexo.

Mas o que acontece se pedirmos que você escreva um programa que encontre o maior de duzentos números? Você consegue imaginar o código?

Você precisará de duzentas variáveis. Se duzentas variáveis não forem ruins o suficiente, tente imaginar a busca pelo maior de um milhão de números.

Imagine um código que contenha 199 instruções condicionais e duzentas invocações da função input(). Felizmente, você não precisa lidar com isso. Há uma abordagem mais simples.

Ignoraremos os requisitos da sintaxe do Python por enquanto e tentaremos analisar o problema sem pensar na programação real. Em outras palavras, tentaremos escrever o **algoritmo** e, quando estivermos felizes com ele, vamos implementá-lo.

Nesse caso, usaremos um tipo de notação que não é uma linguagem de programação real (não pode ser compilada nem executada), mas é formalizada, concisa e legível. Chama-se pseudocódigo.

Vejamos nosso pseudocódigo abaixo:

**largest\_number = -999999999**

**number = int(input())**

**if number == -1:**

**print(largest\_number)**

**exit()**

**if number > largest\_number:**

**largest\_number = number**

**# Ir para a linha 02**

O que está acontecendo

Em primeiro lugar, podemos simplificar o programa se, no início do código, atribuirmos a variável largest\_number a um valor menor do que qualquer um dos números inseridos Usaremos -999999999 para essa finalidade.

Em segundo lugar, assumimos que nosso algoritmo não saberá antecipadamente quantos números serão entregues ao programa. Esperamos que o usuário insira quantos números desejar - o algoritmo funcionará bem com cento e mil. Como fazemos isso?

Nós fazemos um acordo com o usuário: quando o valor -1 for inserido, será um sinal de que não há mais dados e que o programa deve encerrar seu trabalho.

Caso contrário, se o valor inserido não for igual a -1, o programa lerá outro número e assim por diante.

O truque se baseia na suposição de que qualquer parte do código pode ser executada mais de uma vez - exatamente, quantas vezes for necessário.

Executar uma determinada parte do código mais de uma vez é chamado de loop. O significado desse termo é provavelmente óbvio para você.

As linhas 02 a 08 formam um loop. Vamos passar por elas quantas vezes forem necessárias para revisar todos os valores inseridos.

Você pode usar uma estrutura semelhante em um programa escrito em Python? Sim, você pode.

**Informações adicionais**

O Python geralmente vem com muitas funções internas que farão o trabalho para você. Por exemplo, para encontrar o maior número de todos, você pode usar uma função interna do Python chamada **max().** Você pode usá-lo com vários argumentos. Analise o código abaixo: Utilizando o max (aqui veremos o maiornúmero inserido)

*# Leia três números.*

*number1 = int(input("Digite o primeiro número: "))*

*number2 = int(input("Digite o segundo número: "))*

*number3 = int(input("Digite o terceiro número: "))*

*# Verifique qual dos números é o maior*

*# e passe-o para a variável de número\_maior.*

*largest\_number = max(number1, number2, number3)*

*# Imprimir o resultado.*

*print("O maior número é:", largest\_number)*

**Console**

**Digite o primeiro número: 5**

**Digite o segundo número: 25**

**Digite o terceiro número: 15**

**O maior número é: 25**

Da mesma forma, você pode usar a função min() para retornar o número mais baixo. Você pode recriar o código acima e testá-lo na Sandbox.

Utilizando o min (aqui veremos o menor número inserido)

# Leia três números.

number1 = int(input("Digite o primeiro número: "))

number2 = int(input("Digite o segundo número: "))

number3 = int(input("Digite o terceiro número: "))

# Verifique qual dos números é o maior

# e passe-o para a variável de número\_maior.

largest\_number = min(number1, number2, number3)

# Imprimir o resultado.

print("O maior número é:", largest\_number)

**Console**

**Digite o primeiro número: 10**

**Digite o segundo número: 15**

**Digite o terceiro número: 17**

**O maior número é: 10**

Falaremos sobre essas (e muitas outras) funções em breve. Por enquanto, nosso foco será a execução condicional e loops para que você obtenha mais confiança na programação e ensine as habilidades que permitirão que você compreenda e aplique os dois conceitos em seu código. Então, por enquanto, não estamos pegando nenhum atalho.

**3.1.10 LAB Operadores de comparação e execução condicional**

Cenário

**Spathiphyllum**, mais conhecido como um lírio da paz ou planta de vela branca, é um dos mais populares plantas de interior que filtra as toxinas nocivas do ar. Alguns dos efeitos tóxicos que ele neutraliza incluem o benzeno, o formaldeído e a amônia.

Imagine que seu programa de computador adora essas fábricas. Sempre que recebe uma entrada na forma da palavra Spathiphyllum, involuntariamente grita para o console a seguinte string: "Spathiphyllum é a melhor fábrica de todos os tempos!"



**(**[**https://upload.wikimedia.org/wikipedia/commons/b/bd/Spathiphyllum\_cochlearispathum\_RTBG.jpg**](https://upload.wikimedia.org/wikipedia/commons/b/bd/Spathiphyllum_cochlearispathum_RTBG.jpg)**)**

Escreva um programa que utilize o conceito de execução condicional, use uma string como entrada e:

* imprime a frase "Sim - Spathiphyllum é a melhor
* fábrica de todos os tempos!" para a tela se a sequência inserida for "Spathiphyllum" (maiúscula)
* imprime "Não, eu quero um grande Spathiphyllum!" se a sequência inserida for "spathiphyllum" (letra minúscula)
* imprime "Spathiphyllum! Not[input]!", Caso contrário. Nota: [input] é a string usada como entrada.

Teste seu código usando os dados que fornecemos para você. E compre um Spathiphyllum também!

DADOS:

**Exemplo de entrada:**

spathiphyllum

**Saída prevista:**

No, I want a big Spathiphyllum!

**Output**

**Exemplo de entrada:**

pelargonium

**Saída prevista:**

Spathiphyllum! Not pelargonium!

**Output**

**Exemplo de entrada:**

Spathiphyllum

**Saída prevista:**

Yes - Spathiphyllum *is* the best plant ever!

Código RESULTADO CORRETO (DA PLATAFORMA)

**name = input("Insira o nome da flor: ")**

**if name == "Spathiphyllum":**

**print("Sim - Spathiphyllum é a melhor planta de todos os tempos!")**

**elif name == "spathiphyllum":**

**print("Não, eu quero uma grande Spathiphyllum!")**

**else:**

**print("Spathiphyllum! Não", name + "!")**

Console

Insira o nome da flor: ROSEIRA

Spathiphyllum! Não ROSEIRA!

Insira o nome da flor: Spathiphyllum

Sim - Spathiphyllum é a melhor planta de todos os tempos!

Insira o nome da flor: SPATHIPHYLLUM

Spathiphyllum! Não SPATHIPHYLLUM!

FEITO POR MIM:

# Insira o nome SPATHIPHYLLUM - em letras maiúsculas ou outra condição

lirio = "SPATHIPHYLLUM"

lirio2 = "Spathiphyllum"

uma\_frase = (input("Digite a palavra SPATHIPHYLLUM -letras masculas :"))

frase\_1 = ("Sim, " + lirio + "é a melhor fábrica de todos os tempos!")

frase\_2 = ("Não, eu quero um grande "+ lirio2)

frase\_3 = (lirio2 + "! Not " + uma\_frase +"!")

# Aqui verificamos se uma\_frase é igual a lirio maíusculo

# Se for imprime frasse\_a acrecentando o lirio

if uma\_frase == lirio:

print (frase\_1)

# Aqui verificamos se uma\_frase é igual a lirio2 minusculo

# Se for imprimirá a frase\_2 pedindo inserir lirio com maiusculo

if uma\_frase == lirio2:

print (frase\_2)

# Aqui se nenhum dos casos anteriores for previsto, verificamos se é dirente

# Se for diferente de lirio1 e lirio2, imprimirá a frase\_3 e o imput uma\_frase

if uma\_frase != lirio and uma\_frase != lirio2 :

print (frase\_3)

# Se a palavra digitada for a certa o código informa para concluir - fiim

if uma\_frase == lirio:

print("Concluído o código - Fim")

**Console**

**Digite a palavra SPATHIPHYLLUM -letras masculas :Roseira**

**Spathiphyllum! Not Roseira!**

**Digite a palavra SPATHIPHYLLUM -letras masculas :Spathiphyllum**

**Não, eu quero um grande Spathiphyllum**

**Digite a palavra SPATHIPHYLLUM -letras masculas :SPATHIPHYLLUM**

**Sim, SPATHIPHYLLUMé a melhor fábrica de todos os tempos!**

**Concluído o código – Fim**

**3.1.11 LAB Essenciais da declaração if-else**

Cenário

Era uma vez uma terra - uma terra de leite e mel, habitada por pessoas felizes e prósperas. As pessoas pagavam impostos, é claro - a felicidade tinha limites. O imposto mais importante, chamado de imposto de renda pessoal (PIT) tinha que ser pago uma vez por ano e foi avaliado usando a seguinte regra:

* se a renda do cidadão não era superior a 85.528 talões, o imposto era igual a 18% da renda, menos 556 thaler e 2 centavos (isso era o que eles chamavam de isenção de imposto)
* se a receita fosse superior a esse valor, o imposto seria igual a 14.839 talões e 2 centavos, mais 32% do excedente em mais de 85.528 thaler.

Sua tarefa é escrever uma calculadora de impostos.

* Ela deve aceitar um valor de ponto flutuante: a receita.
* Em seguida, ele deve imprimir o imposto calculado, arredondado para inteiro. Há uma função chamada round() que fará o arredondamento para você - você a encontrará no código do esqueleto no editor.

Nota: esse país feliz nunca devolveu dinheiro para seus cidadãos. Se o imposto calculado for menor que zero, isso significaria apenas nenhum imposto (o imposto foi igual a zero). Leve isso em consideração durante os cálculos.

Observe o código no editor: ele só lê um valor de entrada e gera um resultado, então você precisa concluí-lo com alguns cálculos inteligentes.

Teste seu código usando os dados que fornecemos.

## Dados de teste

**Exemplo de entrada:**

10000

**Saída prevista:**

The tax *is*: 1244.0 thalers

**Output**

**Exemplo de entrada:**

100000

**Saída prevista:**

The tax *is*: 19470.0 thalers

**Output**

**Exemplo de entrada:**

1000

**Saída prevista:**

The tax *is*: 0.0 thalers

**Output**

**Exemplo de entrada:**

-100

**Saída prevista:**

The tax *is*: 0.0 thalers

Feito por mim de forma simples

**income = float(input("Entre com os rendimentos anuais : "))**

**tax = float()**

**if income < 85528:**

**tax = income \* 0.18 - 556.02**

**if income > 85528:**

**tax = 14839.02 + (income - 85528) / 100 \* 32**

**tax = round(tax, 0)**

**if tax <= 0:**

**tax = 0**

**print("-> A taxa é:", tax, "thalers")**

**Console**

**Entre com os rendimentos anuais : 10000**

**-> A taxa é: 1244.0 thalers**

**Entre com os rendimentos anuais : 100000**

**-> A taxa é: 19470.0 thalers**

**Entre com os rendimentos anuais : 1000**

**-> A taxa é: 0 thalers**

**Entre com os rendimentos anuais : -100**

**-> A taxa é: 0 thalers**

EXERCICIO ELABORADO POR IRAÊ CÉSAR BRANDÃO

**# income corresponde a renda (esse valor será inserido pelo input) e calc**

**income = float(input("Entre com os rendimentos anuais : "))**

**"""**

**Ela deve aceitar um valor de ponto flutuante: a receita. Transformamos**

**o valor da taxa em float com valor indefinido**

**"""**

**tax = float()**

**"""**

**Aqui definimos quando a renda for menor que 85528**

**se a renda do cidadão não era superior a 85.528 talões,**

**o imposto era igual a 18% da renda, menos 556 taller e 2 centavos**

**(isso era o que eles chamavam de isenção de imposto)**

**"""**

**if income < 85528:**

**tax = income \* 0.18 - 556.02**

**"""**

**se a receita fosse superior a esse valor, o imposto seria igual**

**a 14.839 talões e 2 centavos, mais 32% do excedente em mais de**

**85.528 taller. definimos a tax + (a diferença da receita) e encontramos os 32%**

**Sua tarefa é escrever uma calculadora de impostos.**

**"""**

**if income > 85528:**

**tax = 14839.02 + (income - 85528) / 100 \* 32**

**"""**

**Aqui utilizamos a função round() para fazer o arredondamento**

**Se o imposto calculado for menor que zero, nenhum imposto (o imposto foi igual a zero).**

**"""**

**tax = round(tax, 0)**

**if tax <= 0:**

**tax = 0**

**"""**

**Aqui imprimimos o valor do imposto (tax)**

**"""**

**print ("------ Resultado do calculo da taxa/imposto-----")**

**print ("-> O valor do rendimento foi de :" , income , "thalers")**

**print ("-> O percentual de imposto cobrado foi de :", tax \* 100 / income , "%")**

**print("-> A taxa é:", tax, "thalers")**

**print(" ---------- Fim-----------")**

**Console**

**Entre com os rendimentos anuais : 10000**

**------ Resultado do calculo da taxa/imposto-----**

**-> O valor do rendimento foi de : 10000.0 thalers**

**-> O percentual de imposto cobrado foi de : 12.44 %**

**-> A taxa é: 1244.0 thalers**

**---------- Fim-----------**

**Entre com os rendimentos anuais : 100000**

**------ Resultado do calculo da taxa/imposto-----**

**-> O valor do rendimento foi de : 100000.0 thalers**

**-> O percentual de imposto cobrado foi de : 19.47 %**

**-> A taxa é: 19470.0 thalers**

**---------- Fim-----------**

**Entre com os rendimentos anuais : 1000**

**------ Resultado do calculo da taxa/imposto-----**

**-> O valor do rendimento foi de : 1000.0 thalers**

**-> O percentual de imposto cobrado foi de : 0.0 %**

**-> A taxa é: 0 thalers**

**---------- Fim-----------**

**Entre com os rendimentos anuais : -100**

**------ Resultado do calculo da taxa/imposto-----**

**-> O valor do rendimento foi de : -100.0 thalers**

**-> O percentual de imposto cobrado foi de : -0.0 %**

**-> A taxa é: 0 thalers**

**---------- Fim-----------**

**3.1.12 LAB Essenciais da declaração if-elif-else**

Cenário

Como você certamente sabe, devido a algumas razões astronômicas, os anos podem ser bissextos ou comuns. Os primeiros têm 366 dias, enquanto os segundos têm 365 dias.

Desde a introdução do calendário gregoriano (em 1582), a regra a seguir é usada para determinar o tipo de ano:

* se o número do ano não é divisível por quatro, é um ano comum;
* caso contrário, se o número do ano não for divisível por 100, será um ano bissexto;
* caso contrário, se o número do ano não for divisível por 400, é um ano comum ;
* caso contrário, é um ano bissexto .

Observe o código no editor: ele só lê o número de um ano e precisa ser concluído com as instruções de implementação do teste que acabamos de descrever.

O código deve gerar uma das duas mensagens possíveis, que são ano bissexto ou ano comum, dependendo do valor inserido.

Seria bom verificar se o ano inserido cai na era gregoriana e emitir um aviso caso contrário: não está dentro do período do calendário gregoriano.

Dica: use os operadores != E %. Teste seu código usando os dados que fornecemos.

Dados de teste:



year = int(input("Digite um ano: "))

if year < 1582:

print("Não dentro do período do calendário gregoriano")

else:

# Escreve o bloco se-então aqui

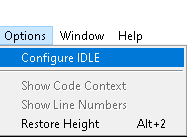
**MUDANDO A COR DE FUNDO DO IDLE DO PYTHON**

Ao abrir o IDLE, verá que o fundo é branco, e para torná-lo DARK, ou sesja, personalizando o fundo de sua tela, vou te orientar como chegar lá.

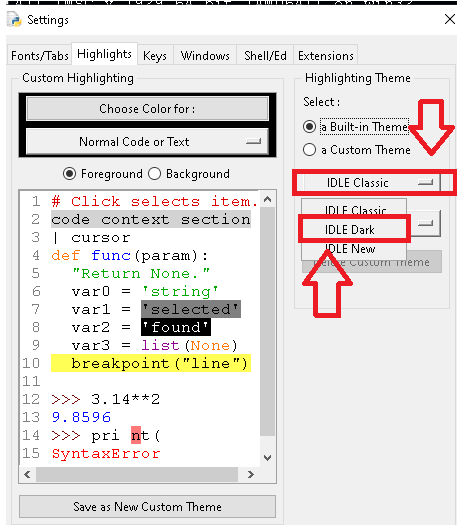
Valos lá aprender?

Basta seguir os passos:

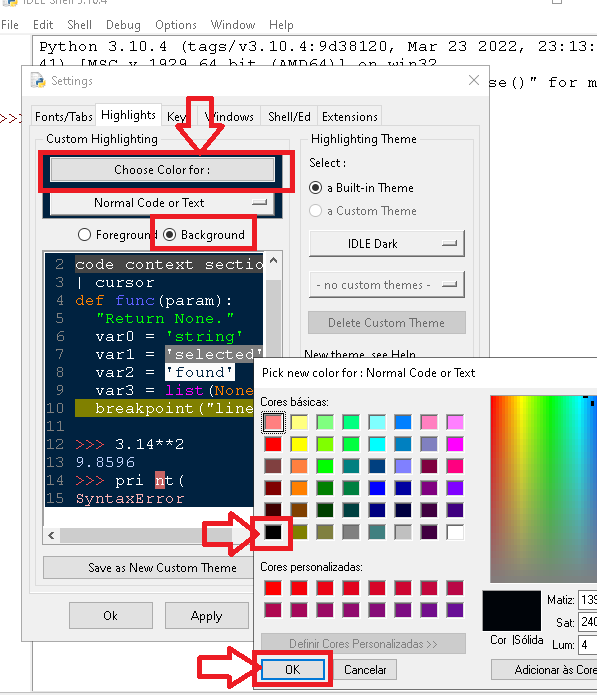
* 1. Após aberto o IDLE, ir ao menu OPTIONS da janela principal e escolher ao menu a opção CONFIGURE IDLE:



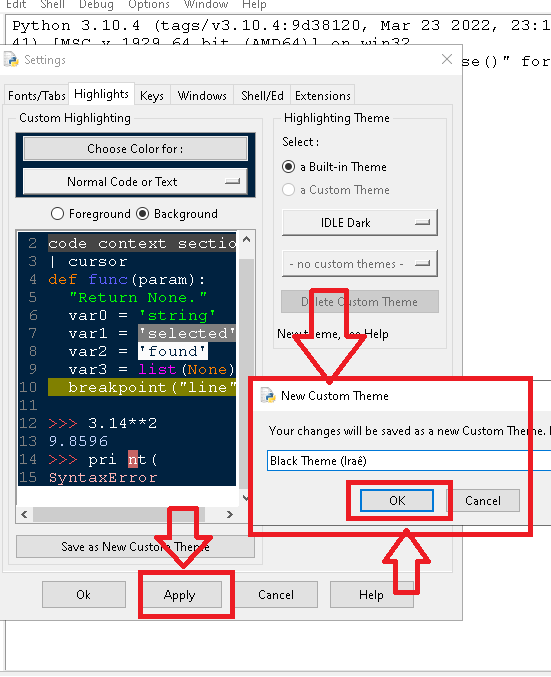
* 1. Feito isso abrirá a janela de configuração. Basta ir em: IDLE CLASSIC e escolher a opção IDLE DARK:



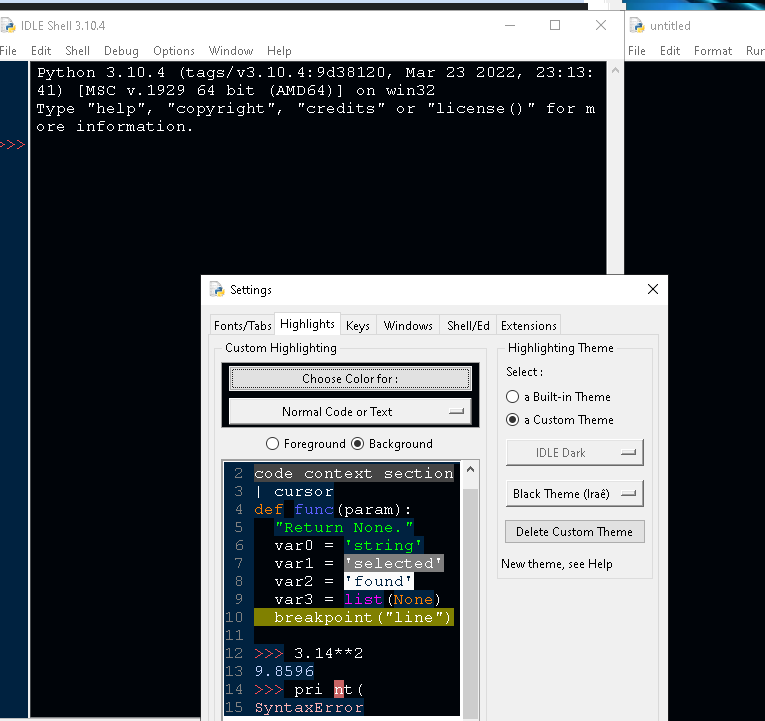
* 1. Após isso, basta escolher a opção no centro e marcar a opção BACKGROUND (tela de fundo), escolher a opção CHOOSE COLOR FOR: e abrirá a paleta de cores conforme exemplo abaixo, aonde escolhera a cor PRETA e basta clicar em OK:



* 1. Após, de um nome a sua PERSONALIZAÇÃO do THEME (baixo coloquei o nome de Black Theme). Clicar em OK e depois em APPLY (Para aplicar o tema de cor preta conforme sua escolha.



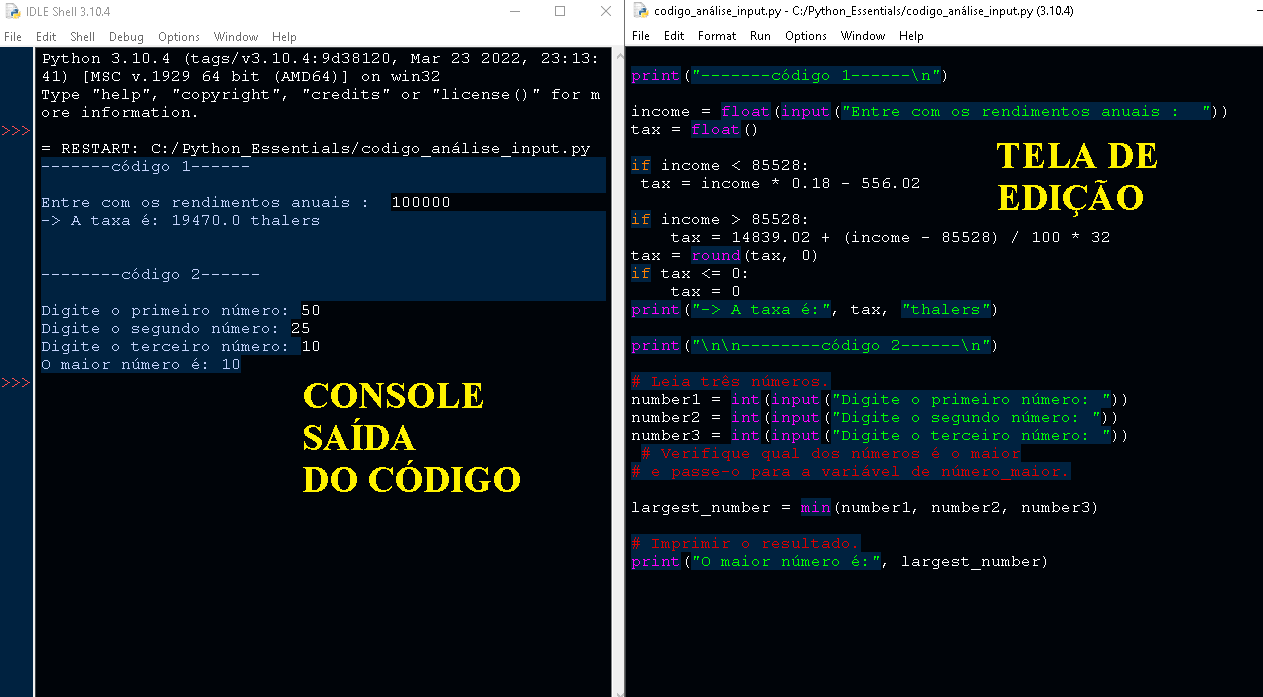
* 1. Agora o FUNDO do seu IDLE estará BLACK (preto) conforme sua escolha. A cor de fundo pode ser qualquer uma que escolher, porém, deverá escolher cores que descansem sua vista e seja mais agradável para a realização de suas tarefas de edição de códigos, visto que passará bastante tempo editando códigos.



* 1. Agora que concluiu a configuração, basta utilizar.

“BONS ESTUDOS e realize ótimos projetos”.

Sucesso!!!



**year = int(input("Digite um ano: "))**

**if year < 1582:**

**print("Não dentro do período do calendário gregoriano")**

**else:**

**if year % 4 != 0:**

**print("ano comum")**

**elif year % 100 != 0:**

**print("Ano bissexto")**

**elif year % 400 != 0:**

**print("ano comum")**

**else:**

**print("Ano bissexto")**

**3.1.13 RESUMO DA SEÇÃO**

1. Os operadores de comparação (também conhecidos como relacionais) são usados para comparar valores. A tabela abaixo ilustra como os operadores de comparação funcionam, supondo que x = 0, y = 1, e z = 0:

|  |  |  |
| --- | --- | --- |
| **Operador** | **Descrição** | **Exemplo** |
| **==** | retorna True se os valores dos operandos forem iguais e False caso contrário | **x == y # False x == z # True** |
| **!=** | retorna True se os valores dos operandos não forem iguais e False caso contrário | **x != y # True x != z # False** |
| **>** | True se o valor do operando esquerdo for maior que o valor do operando direito e False caso contrário | **x > y # False y > z # True** |
| **<** | True se o valor do operando esquerdo for menor que o valor do operando direito e False caso contrário | **x < y # True y < z # False** |
| **>=** | True se o valor do operando esquerdo for maior ou igual ao valor do operando da direita e False caso contrário | **x >= y # False x >= z # True y >= z # True** |
| **<=** | True se o valor do operando esquerdo for menor ou igual ao valor do operando à direita e False caso contrário | **x <= y # True x <= z # True y <= z # False** |

2. Quando você deseja executar algum código apenas se uma determinada condição for atendida, você pode usar uma declaração condicional:

uma única declaração if, por exemplo:

x = 10

if x == 10: # condição

print("x é igual a 10") # Executado se a condição for True.

* uma série de declarações if, por exemplo:

x = 10

if x > 5: # condição um

print("x é maior que 5") # Executado se a condição um for True.

if x < 10: # condição dois

print("x é menor que 10") # Executado se a condição dois for True.

if x == 10: # condição três

print("x é igual a 10") # Executado se a condição três for True.

Cada declaração if é testada separadamente.

* uma declaração if-else, por exemplo:

x = 10

if x < 10: # condição

print("x é menor que 10") # Executado se a condição for True.

else:

print("x é maior ou igual a 10") # Executado se a condição for False.

* uma série de instruções if seguidas de um else, por exemplo:

x = 10

if x > 5: # condição um

print("x é maior que 5") # Executado se a condição um for True.

if x < 10: # condição dois

print("x é menor que 10") # Executado se a condição dois for True.

if x == 10: # condição três

print("x é igual a 10") # Executado se a condição três for True.

Cada um if é testado separadamente. O corpo do else é executado se o último if for False.

* A declaração if-elif-else, por exemplo:

x = 10

if x == 10: # True

print("x == 10")

if x > 15: # False

print("x > 15")

elif x > 10: # False

print("x > 10")

elif x > 5: # True

print("x > 5")

else:

print("senão não será executado")

Se a condição para if for False, o programa verifica as condições dos blocos elif subsequentes - o primeiro bloco elif que é True é executado. Se todas as condições forem False, o bloco else será executado.

* Instruções condicional aninhadas, por exemplo:

x = 10

if x > 5: # True

if x == 6: # False

print("aninhado: x == 6")

elif x == 10: # True

print("aninhado: x == 10")

else:

print("aninhado: else")

else:

print("else")

**OPERADORES ARITMÉTICOS E LÓGICOS EM PYTHON**

<https://pythonacademy.com.br/blog/operadores-aritmeticos-e-logicos-em-python#:~:text=Python%20nos%20disponibiliza%20tr%C3%AAs%20tipos,o%20or%20e%20o%20not%20>.

lve,salve Pythonista!

Nesse post vamos falar de Operadores em Python: Operadores Lógicos, Operadores Aritméticos, Operadores de Associação, Operadores de Comparação, Operadores de Identidade e Operadores de Atribuição!

Wow, é muito Operador! :heart\_eyes:

Se você já programa em Python, já se deparou com esses operadores em algum momento: is, is not, ==, in, not in, and, or ou %=.

Nesse post, você vai dominar TODOS esses Operadores e aprender como utilizá-los em seus programas Python!

Está pronto, então vamos aprender sobre Operadores!

**Vá Direto ao Assunto…**

* Operadores em Python
* Operadores Aritméticos

Operadores de Comparação

* Operadores de Atribuição
* Operadores Lógicos
* Operadores de Identidade
* Operadores de Associação

**Operadores em Python**

Os Operadores em Python possibilitam que o desenvolvedor consiga transcrever a lógica para código.

Python disponibiliza uma série desses operadores para os desenvolvedores e é muito importante dominá-los se você quiser se tornar um verdadeiro Pythonista!

Veremos todos em detalhes agora, começando pelos Operadores Aritméticos!

**Operadores Aritméticos**

Esses operadores são utilizados para criarmos expressões matemáticas comuns, como soma, subtração, multiplicação e divisão.

Veja quais estão disponíveis no Python:

**Operador Nome Função**

+ Adição Realiza a soma de ambos operandos.

- Subtração Realiza a subtração de ambos operandos.

\* Multiplicação Realiza a multiplicação de ambos operandos.

/ Divisão Realiza a Divisão de ambos operandos.

// Divisão inteira Realiza a divisão entre operandos e a parte decimal de ambos operandos.

% Módulo Retorna o resto da divisão de ambos operandos.

\*\* Exponenciação Retorna o resultado da elevação da potência pelo outro.

Veja agora a utilização de cada operador aritmético mencionado acima:

**quatro = 4**

**dois = 2**

**soma = quatro + dois**

**print(soma) # Resultado: 6**

**subtracao = quatro - dois**

**print(subtracao) # Resultado: 2**

**multiplicacao = quatro \* dois**

**print(multiplicacao) # Resultado: 8**

**divisao = quatro / dois**

**print(divisao) # Resultado: 2.0**

**divisao\_interna = quatro // dois**

**print(divisao\_interna) # Resultado: 2**

**modulo = quatro % dois**

**print(modulo) # Resultado: 0**

**exponenciacao = quatro \*\* dois**

**print(exponenciacao) # Resultado: 16**

**Console**

**6**

**2**

**8**

**2.0**

**2**

**0**

**16**

**Operadores de Comparação**

Como o nome já diz, esses Operadores são usados para comparar dois valores:

**Operador Nome Função**

**==** Igual a Verifica se um valor é igual ao outro

**!=**  Diferente de Verifica se um valor é diferente ao outro

**>**  Maior que Verifica se um valor é maior que outro

**>=**  Maior ou igual Verifica se um valor é maior ou igual ao outro

**<** Menor que Verifica se um valor é menor que outro

**<=** Menor ou igual Verifica se um valor é menor ou igual ao outro

Vamos ver exemplos da utilização de cada operador de comparação mencionado acima.

Para facilitar o entendimento, todas as operações estão retornando um valor igual a True, para que você entenda como cada condição é aceita:

**var = 5**

**if var == 5:**

**print('Os valores são iguais')**

**if var != 7:**

**print('O valor não é igual a 7')**

**if var > 2:**

**print('O valor da variável é maior de 2')**

**if var >= 5:**

**print('O valor da variável é maior ou igual a 5')**

**if var < 7:**

**print('O valor da variável é menor que 7')**

**if var <= 5:**

**print('O valor da variável é menor ou igual a 5')**

**Resultado do código acima:**

Os valores são iguais

O valor não é igual a 5

O valor da variável é maior de 5

O valor da variável é maior ou igual a 5

O valor da variável é menor que 7

O valor da variável é menor ou igual a 5

**Operadores de Atribuição**

Esse Operadores são utilizados no momento da atribuição de valores à variáveis e controlam como a atribuição será realizada.

Veja quais Operadores de Atribuição estão disponíveis em Python:

**Operador Equivalente a**

**=**  x = 1

**+=**  x = x + 1

**-=** x = x - 1

**\*=** x = x \* 1

**/=** x = x / 1

**%=** x = x % 1

Exemplo da utilização de cada operador de atribuição mencionado acima:

Operador +=:

**numero = 5**

**numero += 7**

**print(numero) # Resultado será 10**

Operador -=:

**numero = 5**

**numero -= 3**

**print(numero) # Resultado será 2**

Operador \*=:

**numero = 5**

**numero \*= 2**

**print(numero) # Resultado será 10**

Operador /=:

**numero = 5**

**numero /= 4**

**print(numero) # Resultado será 1.25**

Operador %=:

numero = 5

numero %= 2

print(numero) # Resultado será 1

Obs: O operador % é chamado módulo e nada mais é que o resto da divisão. No exemplo acima: 5 dividido por 2 dá 2 de resultado e sobra 1. Por isso numero %= 2 será 1!

**Operadores Lógicos**

Esses Operadores nos possibilitam construir um tipo de teste muito útil e muito utilizado em qualquer programa Python: os testes lógicos.

Python nos disponibiliza três tipos de Operadores Lógicos: o and, o or e o not.

Vamos ver mais sobre eles agora!

**Operador Definição**

**and Retorna True se ambas as afirmações forem verdadeiras**

**or Retorna True se uma das afirmações for verdadeira**

**not retorna Falso se o resultado for verdadeiro**

**Exemplo da utilização de cada um:**

**num1 = 7**

**num2 = 4**

**# Exemplo and**

**if num1 > 3 and num2 < 8:**

**print("As Duas condições são verdadeiras")**

**# Exemplo or**

**if num1 > 4 or num2 <= 8:**

**print("Uma ou duas das condições são verdadeiras")**

**# Exemplo not**

**if not (num1 < 30 and num2 < 8):**

**print('Inverte o resultado da condição entre os parânteses')**

**Console**

**As Duas condições são verdadeiras**

**Uma ou duas das condições são verdadeiras**

**Operadores de Identidade**

Estes Operadores são utilizados para comparar objetos, verificando se os objetos testados referenciam o mesmo objeto (is) ou não (is not).

**Operador Definição**

**is Retorna True se ambas as variáveis são o mesmo objeto**

**is not Retorna True se ambas as variáveis não forem o mesmo objeto**

Agora vamos aos exemplos de como utilizar cada operador de identidade mencionado acima:

**Exemplo do operador is:**

**lista = [1, 2, 3]**

**outra\_lista = [1, 2, 3]**

**recebe\_lista = lista**

**# Recebe True, pois são o mesmo objeto**

**print(f"São o mesmo objeto? {lista is recebe\_lista}")**

**# Retorna False, pois são objetos diferentes**

**print(f"São o mesmo objeto? {lista is outra\_lista}")**

**Resultado do código acima:**

**São o mesmo objeto? True**

**São o mesmo objeto? False**

**Exemplo do operador is not:**

**tupla = (1, 2, 3)**

**outra\_tupla = (1, 2, 3)**

**print(f"Os objetos são diferentes? {outra\_tupla is tupla}")**

**Resultado do código acima:**

**Os objetos são diferentes? True**

Muitas vezes programadores Python ficam na dúvida em quando utilizar o operador de igualdade == ou o operador de identidade is.

Mas agora que você já conhece os dois sabe que o operador == verifica os valores testados, enquanto o operador is testa a referência dos valores testados!

**Operadores de Associação**

Por último, temos os Operadores de Associação.

Eles servem para verificar se determinado objeto está associado ou pertence a determinada estrutura de dados.

**Operador Função**

**in Retorna True caso o valor seja encontrado na sequência**

**not in Retorna True caso o valor não seja encontrado na sequência**

Exemplo da utilização de cada operador de associação mencionado acima:

**lista = ["Python", 'Academy', "Operadores", 'Condições']**

**# Verifica se existe a string dentro da lista**

**print('Python' in lista) # Saída: True**

**# Verifica se não existe a string dentro da lista**

**print('SQL' not in lista) # Saída: True**

**Console**

**True**

**True**

<https://pythonacademy.com.br/jornada-python/>

**USO DO FOR**

**precos = {**

**"Celular": 1500,**

**"computador": 5000,**

**"tablet": 2500,**

**}**

**for item in precos:**

**print(item)**

Celular

computador

tablet

precos = {

"Celular": 1500,

"computador": 5000,

"tablet": 2500,

}

#Aqui imprimirá os ítens

for item in precos:

print(item)

#Aqui imprimirá os preços

**for item in precos:**

**print(precos[item])**

Celular

computador

tablet

1500

5000

2500

**#Transformar um número em moeda**

**numero = 2000**

**numero = f"R${numero:,.2f}"**

**print(numero)**

R$2,000.00

# Conferindo metas dentro valores

vendas = 6000

if vendas >= 4500:

print("Parabéns, você bateu a 3ª meta")

if vendas >= 3000:

print("Parabéns, você bateu a 2ª meta")

if vendas >= 1000:

print("Parabéns, você bateu a 1ª meta")

Parabéns, você bateu a 3ª meta

Parabéns, você bateu a 2ª meta

Parabéns, você bateu a 1ª meta

**lista = ["ipnone", "ipad", "notebook"]**

**nova\_lista = lista.append("airpod")**

**print(lista)**

**print(nova\_lista)**

['ipnone', 'ipad', 'notebook', 'airpod']

None

lista =["café", "açucar", "colher"]

print(lista)

lista.append("garrafa\_termica")

nova\_lista = lista

print(lista)

print(nova\_lista)print(nova\_lista)

['café', 'açucar', 'colher']

['café', 'açucar', 'colher', 'garrafa\_termica']

['café', 'açucar', 'colher', 'garrafa\_termica']