



DEFINIÇÃO

Introdução à linguagem Python e suas características, apresentando as variáveis, os tipos de dados, as expressões, os operadores e as formas de atribuição, de entrada e saída de dados.

PROPÓSITO

Compreender os aspectos básicos de Python visando ao aprendizado da programação nessa linguagem.

OBJETIVOS

MÓDULO 1

Descrever a linguagem Python e suas principais características

MÓDULO 2

Reconhecer o uso de variáveis em Python

MÓDULO 3

Identificar os tipos de dados e as expressões em Python

MÓDULO 4

Identificar as formas de atribuição, de entrada e saída de dados em Python

INTRODUÇÃO

Atualmente, o ensino de disciplinas relacionadas à programação vem crescendo na educação básica e no ensino superior. Nos cursos de graduação voltados à tecnologia da informação, como Engenharia da Computação e Ciência da Computação, é frequente que se tenha contato com mais de uma disciplina focada em programação. Além desses cursos, outras carreiras têm contato com a programação em disciplinas como Introdução à Computação ou similares. Nesses diferentes contextos, várias linguagens de programação são usadas, tanto para o ensino como para o desenvolvimento de projetos.

Dentre as diversas linguagens de programação que existem, **Python** é considerada uma das principais. Por sua simplicidade de aprendizado, ela tem sido utilizada em diversos cursos universitários como a primeira linguagem com que os alunos têm contato ao programar. Atualmente, conta com ampla participação da comunidade, além de ter seu desenvolvimento aberto e gerenciado pela organização sem fins lucrativos Python Software Foundation.



Fonte: Trismegist san/Shutterstock

Recentemente, a **IEEE Computer Society** classificou-a como a linguagem mais indicada para aprender em 2020. Isso se deve à sua eficiência no desenvolvimento de *machine learning*, inteligência artificial, ciência, gestão e análise de dados.

IEEE COMPUTER SOCIETY

Grupo voltado à ciência da computação, engenharia elétrica e eletrônica, apoiando e organizando congressos no mundo todo e divulgando artigos científicos dessas áreas.

MÓDULO 1

- 🕒 Descrever a linguagem Python e suas principais características

CONCEITOS

Python é uma linguagem de programação de alto nível, que permite ao programador utilizar instruções de forma intuitiva, tornando seu aprendizado mais simples do que o aprendizado de uma linguagem de baixo nível.

Nas linguagens de baixo nível, o programador precisa se expressar de forma muito mais próxima do que o dispositivo “entende”, levando naturalmente a um distanciamento da linguagem utilizada para comunicação entre duas pessoas.

A linguagem Python foi lançada pelo holandês **Guido van Rossum** no início dos anos 1990 e desde então tem aumentado sua participação no mundo da programação. Permite uma programação fácil e clara para escalas pequenas e grandes, além de enfatizar a legibilidade eficiente do código, notadamente usando espaços em branco significativos.



Fonte: Wikipedia

📷 Guido van Rossum

CARACTERÍSTICAS DA LINGUAGEM PYTHON

A linguagem Python tem se tornado cada vez mais popular porque, além de permitir um rápido aprendizado inicial, tem características importantes, tais como:



É MULTIPARADIGMA

Apesar de suportar perfeitamente o paradigma de programação estruturada, Python também suporta programação orientada a objetos, tem características do paradigma funcional, com o amplo uso de bibliotecas, assim como permite recursividade e uso de funções anônimas.

É INTERATIVA

Permite que os usuários interajam com o interpretador Python diretamente para escrever os programas, utilizando o prompt interativo. Esse prompt fornece mensagens detalhadas para qualquer tipo de erro ou para qualquer comando específico em execução, suporta testes interativos e depuração de trechos de código.



É HÍBRIDA QUANTO AO MÉTODO DE IMPLEMENTAÇÃO

Python usa uma abordagem mista para explorar as vantagens do interpretador e do compilador. Assim como Java, utiliza o conceito de máquina virtual, permitindo a geração de um código intermediário, mais fácil de ser interpretado, mas que não é vinculado definitivamente a nenhum sistema operacional.

É PORTÁVEL

Tem a capacidade de rodar em uma grande variedade de plataformas de hardware com a mesma interface. Ele roda perfeitamente em quase todos os sistemas operacionais, como Windows, Linux,

UNIX, e Mac OS, sem nenhuma alteração.



É EXTENSÍVEL

Permite que os programadores adicionem ou criem módulos e pacotes de baixo nível / alto nível ao interpretador Python. Esses módulos e pacotes de ferramentas permitem que os desenvolvedores tenham possibilidades amplas de colaboração, contribuindo para a popularidade da linguagem.

SUPOORTA BANCOS DE DADOS

Por ser uma linguagem de programação de uso geral, Python suporta os principais sistemas de bancos de dados. Permite escrever código com integração com MySQL, PostgreSQL, SQLite, ElephantSQL, MongoDB, entre outros.



SUPOORTA INTERFACE COM USUÁRIO

Permite escrever código de tal maneira que uma interface do usuário para um aplicativo possa ser facilmente criada, importando bibliotecas como Tkinter, Flexx, CEF Python, Dabo, Pyforms ou PyGUI wxPython.

PODE SER USADO COMO LINGUAGEM DE SCRIPT

Permite fácil acesso a outros programas, podendo ser compilado para bytecode a fim de criar aplicativos grandes.



PERMITE DESENVOLVIMENTO DE APLICAÇÕES WEB

Devido à escalabilidade já citada, Python oferece uma variedade de opções para o desenvolvimento de aplicativos Web. A biblioteca padrão do Python incorpora muitos protocolos para o desenvolvimento da web, como HTML, XML, JSON, processamento de e-mail, além de fornecer base para FTP, IMAP e outros protocolos da Internet.

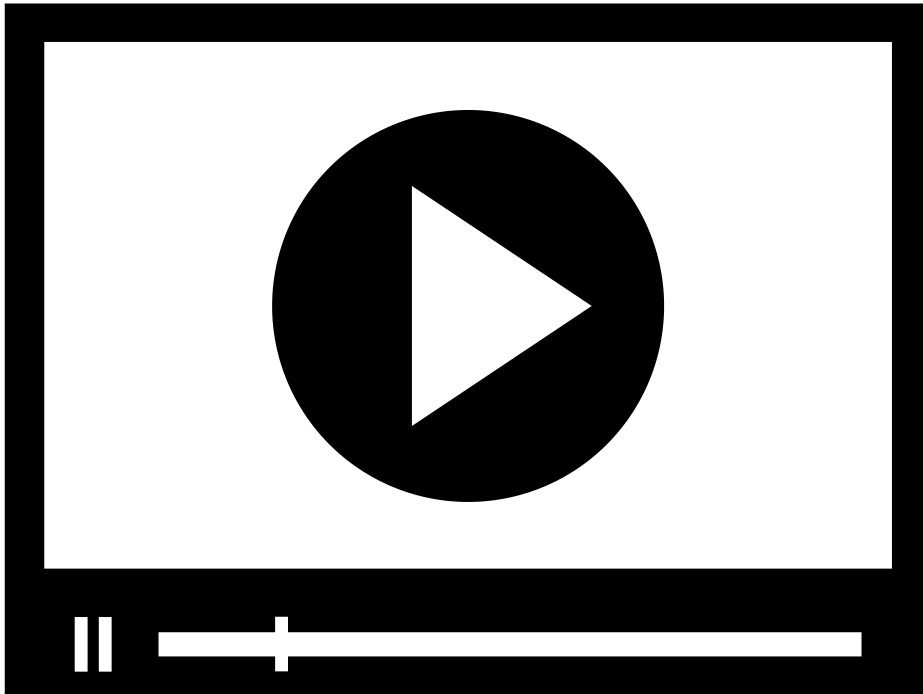
PERMITE CRIAÇÃO DE APLICAÇÕES COMERCIAIS

É desenvolvido sob uma licença de código aberto aprovada pela OSI, tornando-o livremente utilizável e distribuível, mesmo para uso comercial.



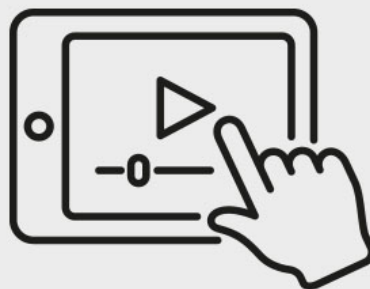
INSTALAÇÃO

A linguagem Python tem seu site oficial, onde é possível encontrar as últimas versões lançadas, notícias e documentação, entre outras informações. A versão que utilizaremos neste tema é a 3.7.7. Por isso, recomendamos que ela seja baixada e instalada em seu computador. Também será necessário baixar e instalar a IDE PyCharm, usando a versão disponível em seu site.



Para conhecer o processo de **instalação do Python**, assista ao vídeo a seguir.

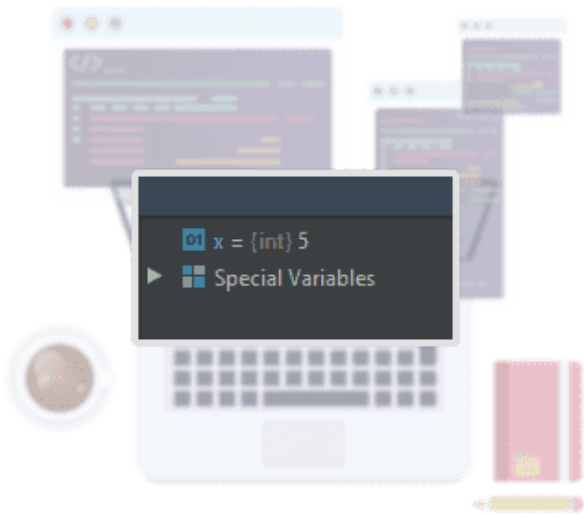
Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



UTILITÁRIOS E MÓDULOS

Apenas como exemplo, na área de Console clique no botão Python Console. No prompt interativo `>>>` que se abrirá, digite `x = 5` e pressione a tecla [ENTER] ou [RETURN] do seu teclado. Observe na figura

2 que, na área Árvore de exibição de variáveis, agora fica disponível a informação que a variável **x** é do tipo **int** e tem o valor 5.



Fonte: o autor (2020)

📷 Figura 2 - Exibição de variáveis

Não se preocupe ainda com o conceito de variável, nem com o seu tipo. Veremos tudo isso com detalhes nos próximos módulos deste tema.

O utilitário **dir** apresenta todos os atributos e métodos disponíveis para determinado tipo de dado. No prompt interativo `>>>`, digite **dir(x)** e pressione a tecla [ENTER] ou [RETURN] do seu teclado.



Fonte: Freepik

No prompt interativo `>>>`, digite **dir(x)** e pressione a tecla [ENTER] ou [RETURN] do seu teclado.



```
Python Console
>>> dir(int)
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__', '__delattr__', '__dir__',
 '__divmod__', '__doc__', '__eq__', '__float__', '__floor__', '__floordiv__', '__format__', '__ge__',
 '__getattr__', '__getnewargs__', '__gt__', '__hash__', '__index__', '__init__',
 '__init_subclass__', '__int__', '__invert__', '__le__', '__lshift__', '__lt__', '__mod__', '__mul__',
 '__neg__', '__new__', '__or__', '__pos__', '__pow__', '__radd__', '__rand__', '__rdivmod__',
 '__reduce__', '__reduce_ex__', '__repr__', '__rfloordiv__', '__rlshift__', '__rmod__', '__rmul__',
 '__ror__', '__round__', '__rpow__', '__rrshift__', '__rshift__', '__rsub__', '__rtruediv__',
 '__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__truediv__',
 '__trunc__', '__xor__', 'as_integer_ratio', 'bit_length', 'conjugate', 'denominator', 'from_bytes',
 'imag', 'numerator', 'real', 'to_bytes']
```

Fonte: o autor (2020)

📷 Figura 3 - Utilitários

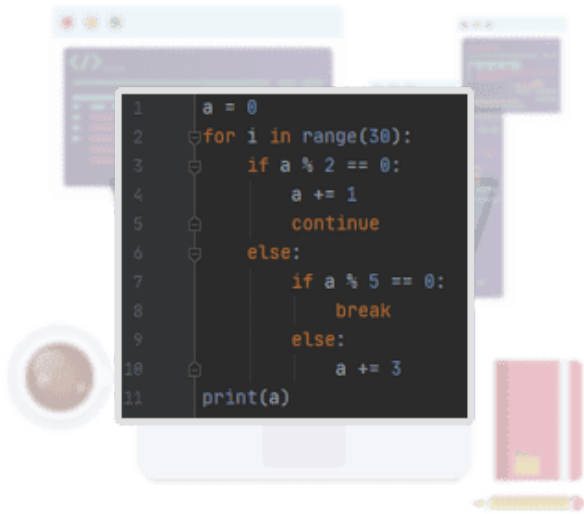
O utilitário **help** apresenta a documentação relativa a determinado tipo de dado. Caso você tenha alguma dúvida sobre o que é possível fazer com determinado tipo, os utilitários **dir** e **help** podem ser úteis.

BLOCOS

Em Python, os blocos são definidos pela **indentação**. Diferente de C e Java, que usam as chaves { e } para delimitar os blocos, em Python todos os blocos são iniciados com o símbolo : (dois pontos) na linha superior e representados pelo acréscimo de 4 (quatro) espaços à esquerda. Sem se preocupar por enquanto com o significado das expressões **for**, **if**, **else** ou **range**, observe a figura 4:

INDENTAÇÃO

Trata-se da digitação dos códigos do programa, afastados por espaço da margem e dispostos hierarquicamente para facilitar a visualização e percepção do programa.



Fonte: o autor (2020)

📷 Figura 4 - Blocos

A partir dessa figura, podemos destacar alguns pontos:

LINHA 1

Está mais à esquerda, assim como as linhas 2 e 11.

LINHA 2

Todas as linhas de 3 a 10 estão dentro do bloco do **for** da linha 2.

LINHA 3

Observe que a linha 3 tem um **if** abrindo um bloco, dentro do qual estão as linhas 4 e 5.

LINHA 6

Por sua vez, a linha 6 tem um **else** abrindo outro bloco, composto pelas linhas de 7 a 10. Os blocos do **if** (linha 3) e do **else** (linha 6) estão no mesmo nível.

LINHA 7

Mostra outro **if** abrindo outro bloco – composto apenas pela linha 8 – que está no mesmo nível do bloco do **else** da linha 9 – composto apenas pela linha 10.

LINHA 11

Como a linha 11 está no mesmo nível da linha 2, ela não faz parte do bloco do **for**.

COMENTÁRIOS

Em Python, os comentários podem ser de **uma linha** ou de **várias linhas**. A tabela 1 mostra as formas de limitar um comentário, além de comparar essas formas em Python e C.

Atenção! Para visualização completa da tabela utilize a rolagem horizontal

	Python	C
Comentários de uma linha	Iniciados com #	Iniciados com //
Comentários com várias linhas	Limitados por """ (três aspas duplas) no início e no fim	Iniciados com /* e encerrados com */

Tabela 1 - Comentários - Fonte: o autor (2020)

 **ATENÇÃO**

É importante lembrar que os comentários não são instruções a serem executadas. Então, você pode escrever de forma simples e objetiva, sem obedecer às regras de sintaxe da linguagem.

BOAS PRÁTICAS DE PROGRAMAÇÃO

Ao começar sua jornada como programador, é importante perceber que existem algumas práticas que não são obrigatórias, mas podem ajudar muito no seu aprendizado. Além disso, podem permitir que você corrija mais rapidamente erros que podem surgir no futuro e tornam seu código mais fácil de ser compreendido por outro programador, favorecendo o trabalho em equipe. Vamos conhecer algumas delas:



Fonte: Porcupen/shutterstock

Uma prática muito importante é utilizar comentários no seu programa, explicando o que aquele trecho resolve.



Fonte: Porcupen/shutterstock

Uma característica marcante da comunidade de desenvolvedores Python é manter uma lista de propostas de melhorias, chamadas PEP (*Python Enhancement Proposals*). Dentre as PEPs, destaca-se a PEP8, que estabelece um guia de estilo de programação.

Agora que já vimos os principais conceitos deste módulo, é hora de testar seus conhecimentos.

VERIFICANDO O APRENDIZADO

1. ANALISE AS AFIRMATIVAS A SEGUIR:

I. PYTHON É UMA LINGUAGEM LIVRE DE ALTO NÍVEL, ORIENTADA A OBJETOS E DE DIFÍCIL LEITURA, POIS NÃO PERMITE INDENTAÇÃO DE LINHAS DE CÓDIGO.

II. PYTHON SUPORTA A MAIORIA DAS TÉCNICAS DA PROGRAMAÇÃO ORIENTADA A OBJETOS.

III. A LINGUAGEM PYTHON E SEU INTERPRETADOR ESTÃO DISPONÍVEIS PARA AS MAIS DIVERSAS PLATAFORMAS.

SÃO CORRETAS:

- A) Somente II.
- B) Somente III.
- C) Somente II e III.
- D) Somente I e II.

2. (2018/IF-MT/INFORMÁTICA) SOBRE A LINGUAGEM PYTHON, É INCORRETO AFIRMAR QUE:

- A) Suporta os paradigmas: imperativo, orientado a objetos e funcional.
- B) Utiliza indentação para delimitar início e fim de blocos.
- C) A linguagem Python é distribuída sob licença que proíbe sua incorporação em produtos proprietários.
- D) Python é um software de código aberto.

GABARITO

1. Analise as afirmativas a seguir:

I. Python é uma linguagem livre de alto nível, orientada a objetos e de difícil leitura, pois não permite indentação de linhas de código.

II. Python suporta a maioria das técnicas da programação orientada a objetos.

III. A linguagem Python e seu interpretador estão disponíveis para as mais diversas plataformas.

São corretas:

A alternativa "**C**" está correta.

A linguagem Python é de fácil leitura, inclusive pela **indentação**. Isso torna a afirmativa I falsa. As afirmativas II e III são verdadeiras.

2. (2018/IF-MT/Informática) Sobre a linguagem Python, é incorreto afirmar que:

A alternativa "**C**" está correta.

A linguagem Python é desenvolvida sob uma licença de código aberto aprovada pela OSI, tornando-a livremente utilizável e distribuível, mesmo para uso comercial.

MÓDULO 2

- ⦿ Reconhecer o uso de variáveis em Python

CONCEITOS

As variáveis são abstrações para endereços de memória que permitem que os programas fiquem mais fáceis de codificar, entender e depurar. Podemos entender que ao nomear uma variável com o identificador **x**, determinado espaço em memória passará a ter esse apelido. Em outras palavras, será possível acessar esse espaço de memória sabendo o seu apelido e, conseqüentemente, recuperar o valor guardado nele, que no nosso exemplo é **10**.

Uma analogia possível com o mundo real é com aquelas caixas de correio que ficam em frente às casas.



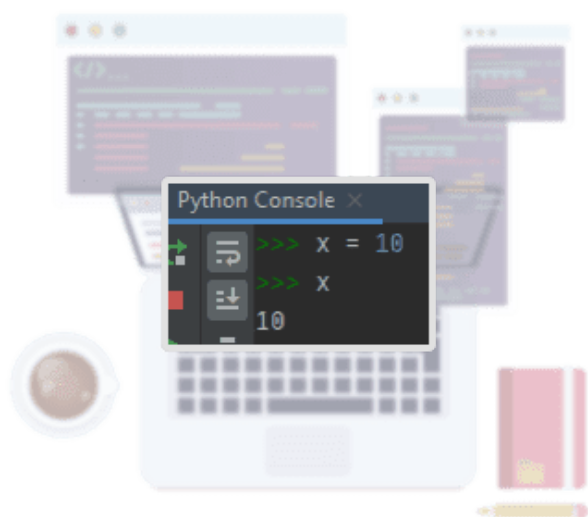
Fonte: Fabio Balbi/Shutterstock

Em Python, o operador de atribuição é o `=` (símbolo de igual). A instrução `x = 10` atribui o valor 10 à variável `x`.



Fonte: Freepik

No prompt interativo `>>>`, digite `x = 10` e pressione a tecla [ENTER] ou [RETURN] do seu teclado. Em seguida, digite `x` e pressione a tecla [ENTER] ou [RETURN].



Fonte: o autor (2020)

📷 Figura 5 - Primeira variável

📢 ATENÇÃO

Observe na figura 5 que o retorno do Python Console foi 10.

Se, posteriormente, você utilizar novamente o prompt interativo `>>>` para digitar `x = 20` e pressionar a tecla [ENTER], você alterará o valor da variável `x`. Ou seja, você estará mudando o valor armazenado naquele espaço de memória, mas sem alterar seu apelido. Observe a figura 6:



Fonte: o autor (2020)

📷 Figura 6 - Atualização de variável

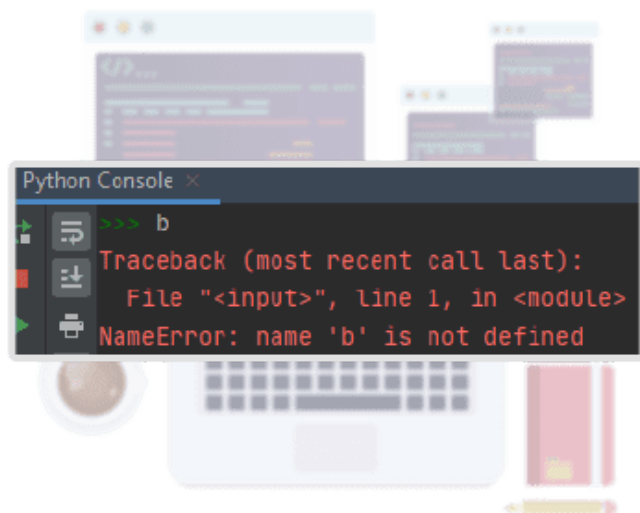
📢 ATENÇÃO

Diferentemente de outras linguagens, como C ou Java, não é necessário declarar uma variável antes de utilizá-la em Python. Basta atribuir um valor inicial à variável e utilizá-la dali em diante. Embora não seja necessário declarar uma variável para utilizá-la, não é possível utilizar uma variável que não tenha recebido alguma atribuição de valor.



Fonte: Freepik

No prompt interativo, digite **b** e pressione a tecla [ENTER] ou [RETURN].



Fonte: o autor (2020)

📷 Figura 7 - Variável não definida

Veja na figura 7 que a mensagem de erro informa que o nome **b** não foi definido. Ou seja, não é possível determinar o valor atribuído a esse nome.

IDENTIFICADORES DE VARIÁVEIS

Os identificadores das variáveis podem ser compostos por letras, o **underline** () e, com exceção do primeiro caractere, números de 0 a 9. Veja os exemplos:

minhaVariavel, **_variavel**, **salario1** e **salario1_2** são válidos.

1variavel e **salario-1** não são válidos.

minhaVariavel e **minhavariavel** são identificadores de duas variáveis distintas.

Mesmo que seja um identificador permitido, nem sempre um identificador é bom para uma variável. Tente utilizar nomes que ajudem a entender o significado da variável para ganhar tempo quando for entender o código posteriormente.

★ EXEMPLO

salario é um nome de variável melhor que **s**.

Algumas palavras são consideradas reservadas e não podem ser usadas como identificadores de variáveis em Python. São elas: **and**, **as**, **assert**, **break**, **class**, **continue**, **def**, **del**, **elif**, **else**, **except**, **exec**, **finally**, **for**, **from**, **global**, **if**, **import**, **in**, **is**, **lambda**, **not**, **or**, **pass**, **print**, **raise**, **return**, **try**, **while**, **with** e **yield**.

AMARRAÇÕES

Chamamos de amarração (*binding*) a associação entre entidades de programação. Veja alguns exemplos:

Variável amarrada a um valor

Operador amarrado a um símbolo

Identificador amarrado a um tipo

O tempo em que a amarração ocorre é chamado de **tempo de amarração**. Cada linguagem pode ter os seguintes tempos de amarração:

TEMPO DE PROJETO DA LINGUAGEM

Os símbolos são amarrados ao operador, como * (multiplicação), ou à definição das palavras reservadas.

TEMPO DE IMPLEMENTAÇÃO

Ocorre em geral nos compiladores, como a definição de faixa de valores para determinado tipo.

TEMPO DE COMPILAÇÃO

Associação da variável ao seu tipo. Lembre-se de que Python associa a variável ao tipo, como foi explicado na seção anterior.

TEMPO DE LIGAÇÃO

A ligação de vários módulos compilados previamente, como a chamada a uma função de um módulo importado. Em C, utilizamos a diretiva **#include** para termos permissão de utilizar as funções de determinada biblioteca. Em Python, utilizamos o **import** para isto.

TEMPO DE CARGA

Quando o programa é carregado. Por exemplo: endereços de memória relativos são substituídos por endereços absolutos.

TEMPO DE EXECUÇÃO

Associação de valores a variáveis que dependam de entradas do usuário, por exemplo. A variável é vinculada ao valor apenas durante a execução do programa.

O momento em que ocorre a ligação pode ser classificado como cedo (*early binding*) ou tardio (*late binding*). Quanto mais cedo ocorre a ligação, maior a eficiência de execução do programa, porém menor a flexibilidade das estruturas disponibilizadas.

AMARRAÇÃO DE TIPO

As amarrações de tipo vinculam a variável ao tipo do dado. Elas podem ser:

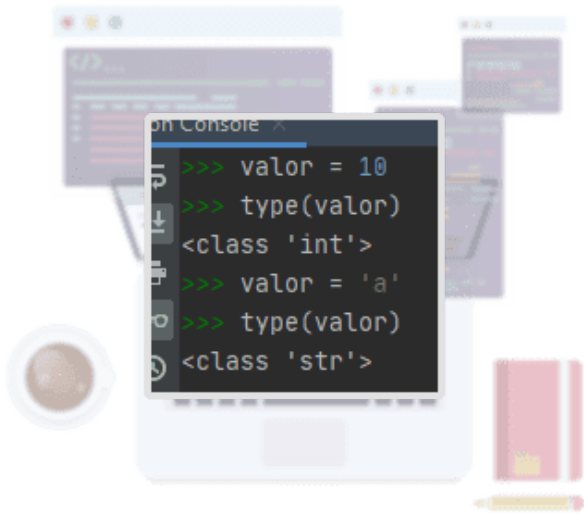
ESTÁTICAS

Ocorrem antes da execução e permanecem inalteradas. Em C, declaramos **int a**.

DINÂMICAS

Ocorrem durante a execução e podem ser alteradas. É o caso do Python.

Veja a figura 8:



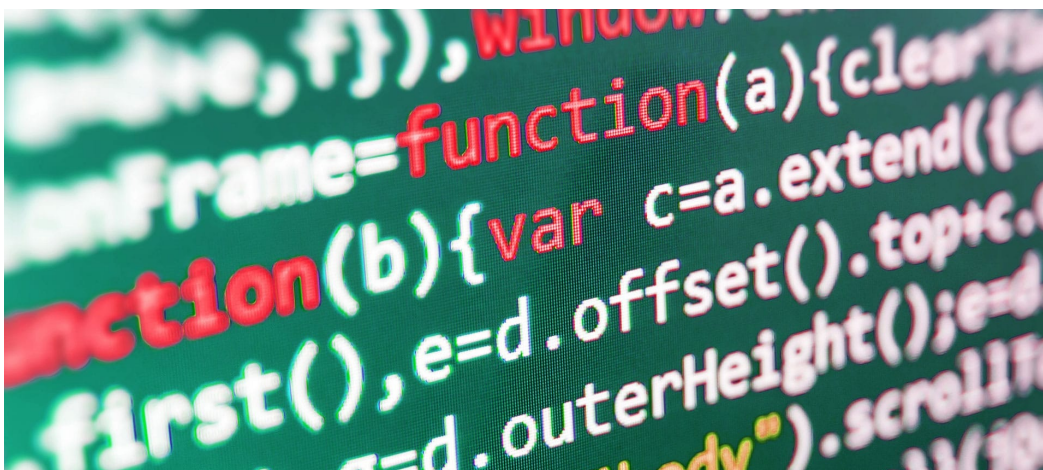
Fonte: o autor (2020)

📷 Figura 8 - Tipagem dinâmica

Perceba que a chamada **type** (parâmetro) retorna o tipo do parâmetro informado entre parênteses. Observe que a variável **valor** recebeu 10 e, com isso, ficou vinculada ao tipo **int**. Porém, ao receber o valor 'a', passou a estar vinculada ao tipo **str** (string).

ESCOPO DE VISIBILIDADE

O escopo define em quais partes do programa uma variável é visível. Cada nome de variável em Python tem seu escopo e fora desse escopo o nome não existe, gerando um erro quando se tenta referenciar esse nome. Quanto ao escopo, chamamos as **variáveis de globais ou locais**.



Fonte: BEST-BACKGROUNDS/Shutterstock

VARIÁVEIS GLOBAIS

Todos os nomes atribuídos no prompt interativo ou em um módulo fora de qualquer função são considerados como de escopo global. Por exemplo, ao executar a instrução da figura 9, a variável **x** é uma variável global.



Fonte: o autor (2020)

📷 Figura 9 - Variável global

VARIÁVEIS LOCAIS

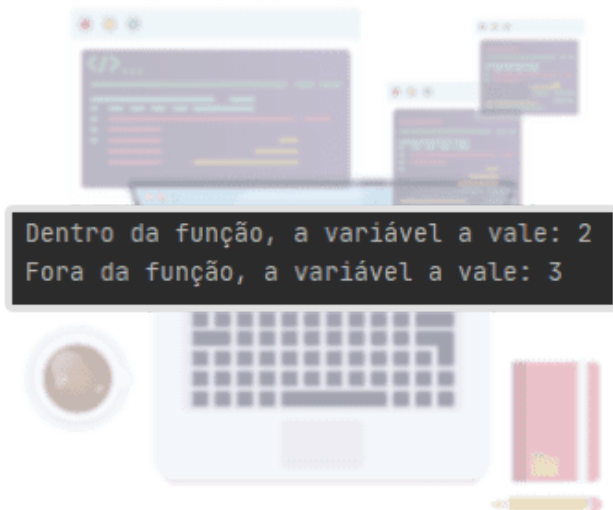
Para exemplificar o uso de variáveis com escopo local, vamos utilizar uma função definida pelo desenvolvedor. Não se preocupe com esse tipo de função por enquanto, você aprenderá mais detalhes posteriormente. Observe a figura 10:



Fonte: o autor (2020)

📷 Figura 10 - Variáveis locais - código-fonte

As linhas 2, 3 e 4 compõem o bloco interno à função chamada **multiplicador()**. Embora as variáveis das linhas 2 e 7 tenham o mesmo nome, elas são abstrações a endereços de memória diferentes. Dentro da função **multiplicador()**, a chamada ao nome **a** recupera o valor 2. Fora da função **multiplicador()**, a chamada ao nome **a** recupera o valor 3. Veja a execução na figura 11:



Fonte: o autor (2020)

📷 Figura 11 - Variáveis locais - execução

Agora, observe a função **multiplicador()** com uma pequena alteração, em que retiramos a inicialização da variável **a** dentro da função.



Fonte: o autor (2020)

📷 Figura 12 - Variáveis locais - código-fonte 2

Na linha 6, ao se chamar a função **multiplicador()**, a variável **a** será procurada. Como não existe uma variável **a** no bloco interno da função, ela é procurada como variável global. Uma vez encontrada, o valor recuperado é 3. Ao executar esse exemplo, você verá:



Fonte: o autor (2020)

📷 Figura 13 - Variáveis locais – execução 2

Usamos este exemplo para mostrar que o interpretador Python pode procurar o mesmo nome de variável em diferentes escopos. A ordem utilizada para a procura é:

1

A chamada da função delimitadora

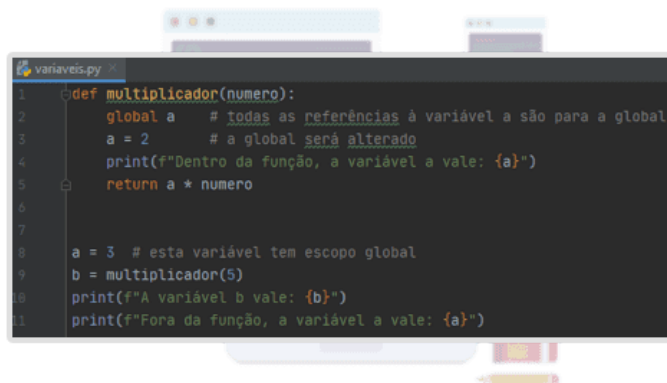
2

Variáveis globais

3

O módulo builtins

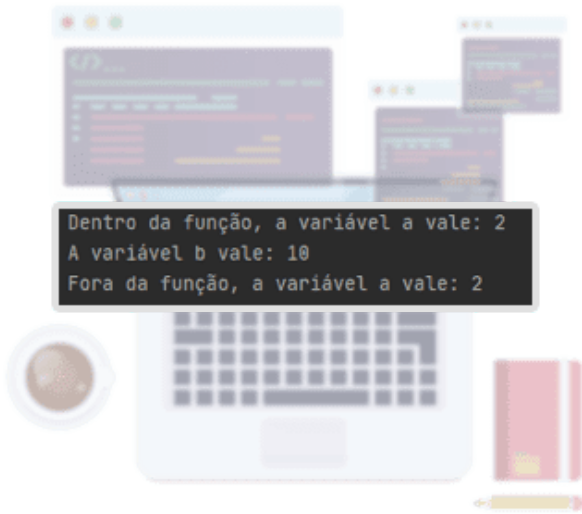
Perceba que, se a variável **a** é inicializada na função **multiplicador()**, qualquer chamada a esse nome dentro da função resultará na referência a essa variável local. Mas seria possível alterar a variável **a** global com uma instrução dentro da função **multiplicador()**? Sim, utilizando-se a palavra reservada **global**. Veja como isso poderia ser feito na figura 14 e o seu resultado na figura 15:



```
1 def multiplicador(numero):
2     global a # todas as referências à variável a são para a global
3     a = 2 # a global será alterado
4     print(f"Dentro da função, a variável a vale: {a}")
5     return a * numero
6
7
8 a = 3 # esta variável tem escopo global
9 b = multiplicador(5)
10 print(f"A variável b vale: {b}")
11 print(f"Fora da função, a variável a vale: {a}")
```

Fonte: o autor (2020)

📷 Figura 14 - Variáveis locais - código-fonte 3



Fonte: o autor (2020)

📷 Figura 15 - Variáveis locais - execução 3

TIPOS DE ESCOPO

Os tipos de escopo são:

ESTÁTICO

O escopo é baseado na descrição textual do programa e as amarrações são feitas em tempo de compilação. É o caso de C, C++ e Java, por exemplo.

DINÂMICO

O escopo é baseado na sequência de chamada dos módulos (ou funções). Por isso, as amarrações são feitas em tempo de execução. É o caso do Python.

O fato de Python ser de escopo dinâmico traz alguns problemas, como a perda de eficiência – uma vez que os tipos precisam ser verificados em tempo de execução – e a redução na legibilidade – porque é difícil determinar a sequência exata de todas as chamadas de função.



Fonte: BEST-BACKGROUNDS/Shutterstock

TEMPO DE VIDA

Embora escopo e tempo de vida tenham uma relação próxima, eles são conceitos diferentes. Observe:

Escopo é um conceito textual



Tempo de vida é um conceito temporal

As variáveis globais têm o tempo de vida que é o de execução do programa, ao passo que as variáveis locais somente existem no intervalo de duração da função ou do bloco a que se limitam.

CONSTANTES

Em Python, não existe o conceito de constante. Se você precisar de uma constante ao longo de sua jornada como programador, atribua o valor a uma variável e tome cuidado para não mudar esse valor.

💡 DICA

Uma dica é iniciar o nome dessa variável com **c_** ou utilizar todas as letras maiúsculas, o que vai diferenciar essa variável das outras. Por exemplo, é possível utilizar a expressão **c_PI = 3.141592** para armazenar o valor de PI e agilizar o cálculo de área e perímetro de um círculo, ou utilizar a expressão

PRECISION = 0.001 para armazenar a precisão a ser utilizada em qualquer cálculo matemático no seu programa.

É importante ficar atento ao uso correto das variáveis, especialmente observando as questões de escopo e visibilidade, para evitar que algum cálculo seja realizado corretamente, mas com resultado diferente do esperado por você ao programar.



Fonte: metamorworks/Shutterstock



No vídeo a seguir, assista a uma demonstração das **variáveis estáticas e dinâmicas**.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



VERIFICANDO O APRENDIZADO

1. (2017/IF-CE/TÉCNICO DE LABORATÓRIO/INFORMÁTICA) CONSIDERE O TRECHO DO PROGRAMA PYTHON ABAIXO:

```
DEF FUNC():
```

```
    X = 1
```

```
    PRINT(X)
```

```
X = 10
```

```
FUNC()
```

```
PRINT(X)
```

OS VALORES IMPRESSOS, AO SE EXECUTAR O PROGRAMA, SÃO, RESPECTIVAMENTE:

A) 1 e 1.

B) 10.

C) 1 e 10.

D) 10 e 10.

2. (MS CONCURSOS/2016/CRECI 1º REGIÃO (RJ)/ANALISTA DE TI) QUAL ALTERNATIVA REPRESENTA A DECLARAÇÃO DE UMA VARIÁVEL NA LINGUAGEM DE PROGRAMAÇÃO PYTHON?

A) var valor = 3;

B) boolean inicio = falso;

C) texto = 'texto de exemplo'

D) int i = 1;

GABARITO

1. (2017/IF-CE/Técnico de Laboratório/Informática) Considere o trecho do programa Python abaixo:

```
def func():
```

```
    x = 1
```

```
    print(x)
```

```
x = 10
```

```
func()
```

```
print(x)
```

Os valores impressos, ao se executar o programa, são, respectivamente:

A alternativa **"C "** está correta.

A variável **x** da linha 2 é local da função **func()**, sendo visível para a chamada **print()** da linha 3. Por sua vez, a variável **x** da linha 5 é global, sendo visível para a chamada **print()** da linha 7.

2. (MS CONCURSOS/2016/Creci 1º Região (RJ)/Analista de TI) Qual alternativa representa a declaração de uma variável na linguagem de programação Python?

A alternativa **"C "** está correta.

Lembre-se de que, em Python, as variáveis não são declaradas com o tipo vinculado. Assim, basta atribuir um valor inicial à variável para que ela possa ser usada. Isso ocorre com a variável **texto**, que recebe o valor inicial "texto de exemplo".

MÓDULO 3

⦿ Identificar os tipos de dados e as expressões em Python

CONCEITOS

Neste módulo, você será apresentado aos tipos padrão incorporados ao interpretador Python. Os principais tipos internos são **numéricos**, **sequenciais** e **dicionários**. Classes, instâncias e exceções também são tipos padrão, mas não entraremos em detalhes neste tema. Para ter nosso primeiro contato com expressões em Python, use o prompt interativo `>>>`.



Fonte: Freepik

Digite a expressão algébrica **5 + 8** e pressione a tecla [ENTER]. Observe o resultado na figura 16:



Fonte: o autor (2020)

📷 Figura 16 - Expressão no console

O Python Console permite que você calcule expressões algébricas como uma calculadora, além de executar instruções básicas em Python.

TIPOS NUMÉRICOS

Existem três tipos numéricos distintos em Python: inteiros, números de ponto flutuante e números complexos. Além disso, os booleanos são um subtipo dos inteiros.

O TIPO INT

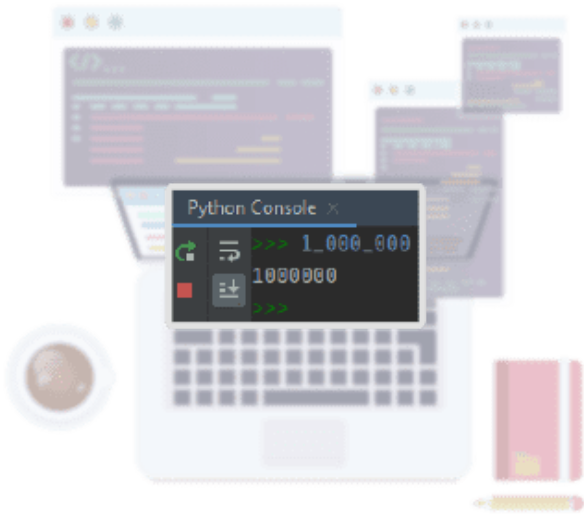
É o tipo usado para manipular números inteiros. Fazendo uma analogia com a Matemática, o tipo **int** é usado para elementos do conjunto dos inteiros (**Z**).

Diferentemente de outras linguagens, como C ou Java, a linguagem Python não limita o tamanho de uma variável de qualquer tipo, logo, não existe um valor inteiro máximo definido. O limite depende da quantidade de memória disponível no computador. De volta ao Python Console, veja algumas coisas interessantes.



Fonte: Freepik

Digite **1_000_000** e pressione a tecla [ENTER].



Fonte: o autor (2020)

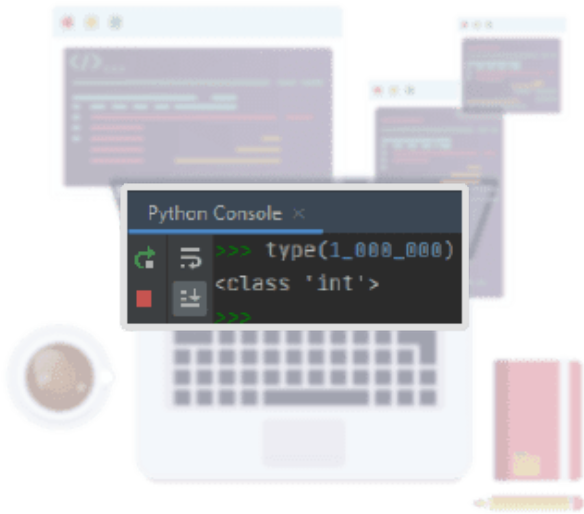
📷 Figura 17 - Inteiro com underline

Python permite que você utilize o **underline** () como separador de milhar. Isso ajuda a visualizar números com muitos dígitos. Para encerrarmos este primeiro contato com o tipo **int**, verifique que o valor do exemplo anterior é inteiro.



Fonte: Freepik

Digite **type(1_000_000)** e pressione a tecla [ENTER].



Fonte: o autor (2020)

📷 Figura 18 - Inteiro com underline 2

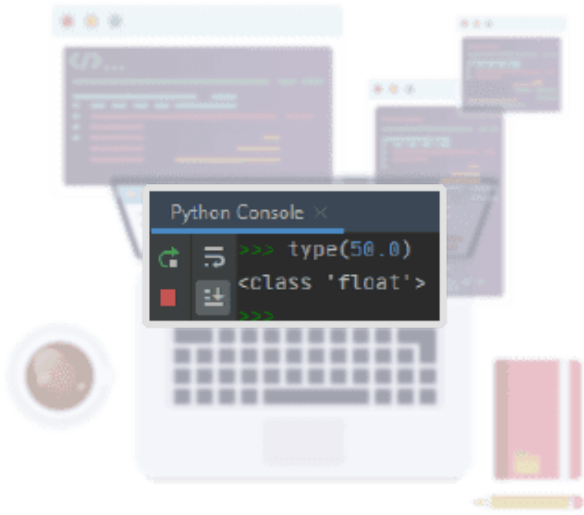
O TIPO FLOAT

É o tipo usado para manipular números com parte inteira e parte decimal, chamados de números de ponto flutuante. Fazendo uma analogia com a Matemática, o tipo **float** é usado para elementos do conjunto dos reais (**R**).



Fonte: Freepik

Para diferenciar um número real de um inteiro, é possível utilizar a parte decimal zerada. No prompt interativo >>>, digite **type(50.0)** e pressione a tecla [ENTER].



Fonte: o autor (2020)

📷 Figura 19 - Real com parte decimal zerada

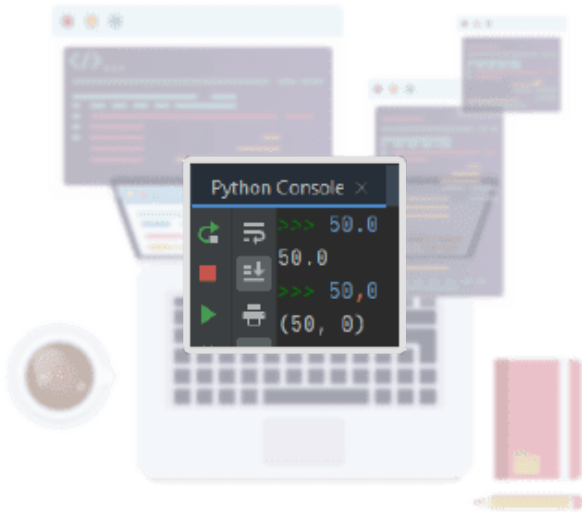
📢 ATENÇÃO

Vale ressaltar que devemos **usar o ponto** para separar a parte inteira da parte decimal, **e não a vírgula**.



Fonte: Freepik

No prompt, digite **50.0** (com ponto) e pressione [ENTER]. Em seguida, digite **50,0** (com vírgula) e pressione a tecla [ENTER].

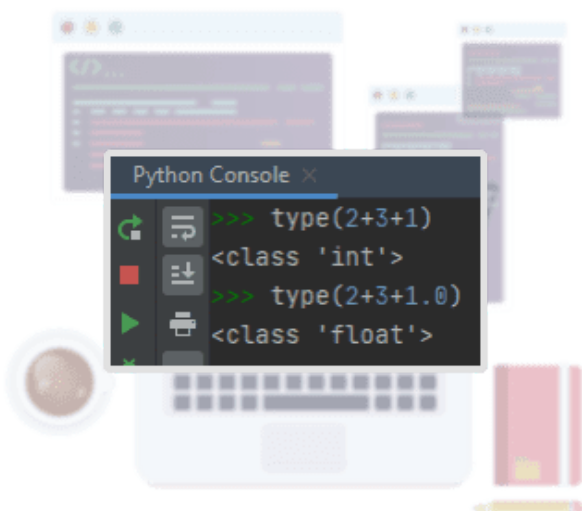


Fonte: o autor (2020)

📷 Figura 20 - Separador de número real

Ao usar a vírgula como separador em Python, o que ocorre, na verdade, é a criação de uma tupla de dois elementos, e não o tipo *float*. Você verá mais detalhes sobre tuplas em um momento posterior.

Embora os tipos **int** e **float** sejam semelhantes, por tratarem de números, eles têm propriedades um pouco diferentes. Em expressões algébricas, sempre que somamos, subtraímos ou multiplicamos apenas elementos do tipo **int**, o resultado é **int**. Porém, basta um operando do tipo **float** para que o resultado seja **float**. Observe a figura 21:

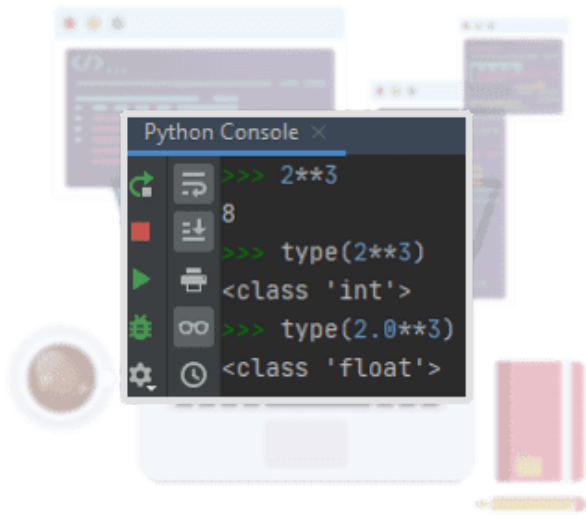


Fonte: o autor (2020)

📷 Figura 21 - Expressões com tipos diferentes

Vamos analisar a exponenciação. Para realizar essa operação matemática, utilizamos o operador (**).

Veja a figura 22:



Fonte: o autor (2020)

📷 Figura 22 - Exponenciação

Veja que basta que a base seja **float** para que o resultado também o seja.

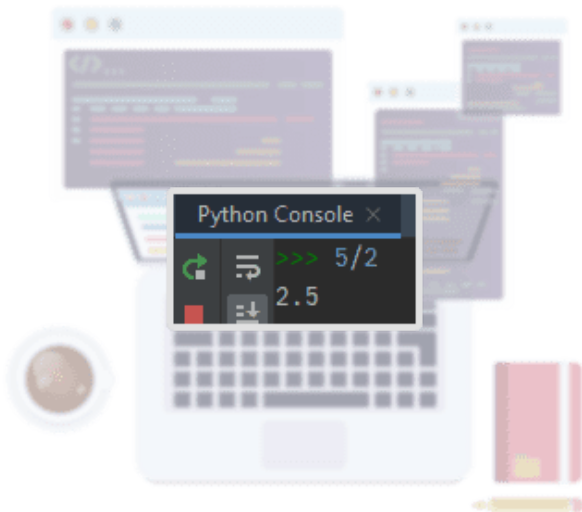
📢 ATENÇÃO

Diferentemente de outras linguagens, como C, a divisão de dois números inteiros não necessariamente tem resultado inteiro.



Fonte: Freepik

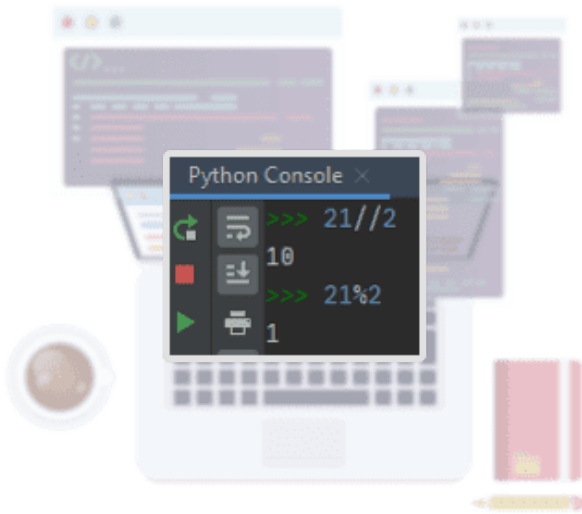
Digite **5/2** e pressione [ENTER].



Fonte: o autor (2020)

📷 Figura 23 - Divisão de inteiros

Para obter o quociente inteiro e o resto, quando dois inteiros são divididos, é necessário utilizar os operadores `//` e `%`, respectivamente. Ao dividir 21 por 2, temos quociente 10 e resto 1. Observe a figura 24:



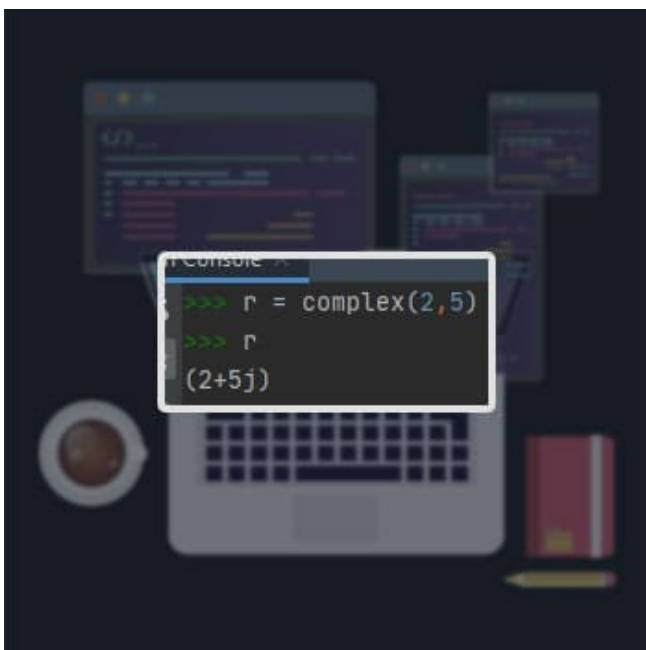
Fonte: o autor (2020)

📷 Figura 24 - Divisão inteira e resto

O TIPO COMPLEX

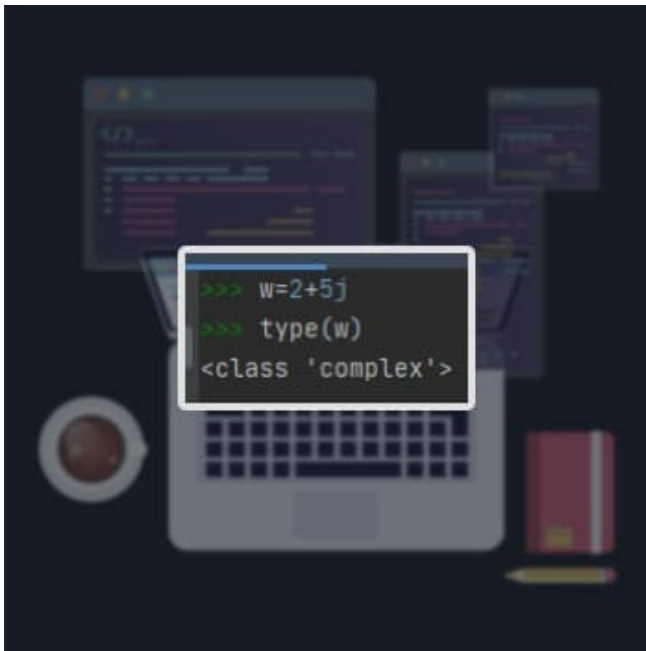
É o tipo utilizado para manipular números complexos, na forma **$x + yj$** , sendo **x** a parte real e **y** a parte imaginária do complexo.

Veja dois exemplos de variáveis do tipo **complex** nas figuras 25 e 26, em que a parte real é 2 e a parte imaginária é 5:



Fonte: o autor (2020)

📷 Figura 25 - complex 1



Fonte: o autor (2020)

📷 Figura 26 - complex 2

A chamada **r.conjugate()** retorna o conjugado do número complexo **r**, em que a parte real é mantida e a parte imaginária tem o seu sinal trocado.

O TIPO BOOL

Uma expressão algébrica, como vimos nos exemplos dos tipos **int** e **float**, é avaliada como um número, seja desses tipos ou de outro tipo numérico admitido em Python. Porém, utilizar expressões não algébricas também é bastante comum. E uma boa notícia é que Python pode avaliar expressões desse tipo também. Essa é uma diferença entre Python e outras linguagens, como C, por exemplo, em que não existe o tipo **bool**.



Fonte: Freepik

No prompt interativo `>>>`, digite a expressão `2 < 3` e pressione [ENTER]. Observe o resultado na figura 27:



Fonte: o autor (2020)

📷 Figura 27 – bool 1

Repare que o resultado dessa expressão não é um número, mas sim a palavra **True**. Caso você colocasse a expressão `2 > 3`, o resultado seria **False**, como pode ver na figura 28.



Fonte: o autor (2020)

📷 Figura 28 - bool 2

⊕ SAIBA MAIS

As expressões que você viu nos dois exemplos são chamadas de **expressões booleanas**. Trata-se de expressões que podem ser avaliadas com um dos dois valores booleanos: **True** ou **False**. Assim, em Python, existe o tipo **bool**, utilizado para permitir o tratamento de expressões como essas.

Agora, vamos ver o operador **not**, que é um operador unário, ou seja, só precisa de um operando. Esse operador inverte o valor booleano, ou seja, se o valor original for **True**, **not(valor)** terá o valor **False**. E vice-versa.



Fonte: Freepik

No prompt interativo `>>>`, digite a expressão **`not(2 < 3)`** e pressione [ENTER].



Fonte: o autor (2020)

📷 Figura 29 - bool 3

É possível também escrever expressões booleanas compostas, utilizando conectivos como **E OU**. Vamos ver mais detalhes sobre essas expressões ainda neste módulo.

OPERADORES NUMÉRICOS

OPERADORES MATEMÁTICOS

Os operadores matemáticos são muito semelhantes àqueles que vimos ao longo de nossa jornada como estudantes, aprendendo Álgebra e Aritmética na escola. Existem algumas pequenas diferenças, como a divisão (que pode ser a usual ou a divisão inteira). Mas é possível identificar operações que fizemos ao longo de toda nossa vida. A tabela 2 lista os operadores de expressão aritmética disponíveis em Python.

Atenção! Para visualização completa da tabela utilize a rolagem horizontal

Operação matemática	Símbolo usado	Exemplo	
		Equação	Resultado
Soma	+	2.5 + 1.3	3.8
Subtração	-	2.5 - 1.3	1.2
Multiplicação	*	2.5 * 1.3	3.25
Divisão	/	2.5/1.3	1.923076923076923
Divisão inteira	//	9/2	4
Resto na divisão inteira	%	9%2	1
Valor absoluto	abs(parâmetro)	abs(-2.5)	2.5
Exponenciação	**	2**4	16

Tabela 2 - Operadores matemáticos - Fonte: o autor (2020)

Além das operações algébricas, é possível realizar operações de comparação. Os operadores de comparação têm como resultado um valor booleano (**True** ou **False**). Observe a tabela 3:

Atenção! Para visualização completa da tabela utilize a rolagem horizontal

--	--

Símbolo usado	Descrição
<	Menor que
<=	Menor ou igual a
>	Maior que
>=	Maior ou igual a
==	Igual
!=	Não igual

Tabela 3 - Operadores de comparação - Fonte: o autor (2020)

ATENÇÃO

Cabe observar que o operador utilizado para comparar se dois valores são iguais é o **==**, ou seja, duplo sinal de igual. Tome cuidado para não confundir com o operador de atribuição, que é representado pelo sinal de igual apenas uma vez (=).

Existe outra lista de operadores que executam operações matemáticas, mas, além disso, atualizam o valor da variável utilizada. Eles são chamados de operadores compostos e serão detalhados no módulo 4. Para mais funções matemáticas, você pode utilizar os módulos matemáticos `math` e `fractions`.

OPERADORES BOOLEANOS

As expressões booleanas são aquelas que podem ter como resultado um dos valores booleanos: `True` ou `False`. É comum utilizarmos os operadores de comparação em expressões booleanas, mas não só eles.

Assim como é possível escrever expressões algébricas complexas concatenando diversas expressões menores, podemos escrever expressões booleanas grandes, com os operadores **and**, **or** e **not**. Observe o comportamento dos operadores booleanos nas tabelas 4, 5 e 6.

Atenção! Para visualização completa da tabela utilize a rolagem horizontal

p	not (p)
True	False
False	True

Tabela 4 - Operador not

p	q	p and q
True	True	True
True	False	False
False	True	False
False	False	False

Tabela 5 - Operador and

p	q	p or q
True	True	True
True	False	True
False	True	True
False	False	False

Tabela 6 - Operador or

```

1  #!/usr/bin/env python
2  import sys
3  import os
4  import simpleknn
5  from bigfile import BigFile
6
7  if __name__ == "__main__":
8      trainCollection = 'toydata'
9      nimages = 2
10     feature = 'f1'
11     dim = 3

```

Fonte: kikujungboy CC/Shutterstock

TIPOS SEQUENCIAIS

Existem três tipos sequenciais básicos em Python: **listas**, **tuplas** e **objetos range**. Além dos tipos básicos citados, existe um tipo especial criado para tratamento de dados textuais: o tipo **str** (string).

Assim como em C ou Java, a indexação dos itens é iniciada com 0 e cada item tem o seu índice incrementado uma unidade em relação ao item anterior. Porém, Python também permite a indexação com valores negativos. O valor -1 é o índice do último item, e cada item anterior é decrementado de uma unidade em relação ao sucessor. Observe a tabela 7:

Atenção! Para visualização completa da tabela utilize a rolagem horizontal

índice	0	1	2	3	4
s	t	e	s	t	e
índice negativo	-5	-4	-3	-2	-1

Tabela 7 - Índices em tipos sequenciais - Fonte: o autor (2020)

STRINGS

Em uma variável do tipo str, é possível armazenar letras, números, espaços, pontuação e diversos símbolos. Diferentemente da linguagem C, não existe o tipo char. Cada caractere em Python é uma

string. Para delimitar uma string, podemos utilizar:

ASPAS SIMPLES

```
'uma string'
```

ASPAS DUPLAS

```
"uma string"
```

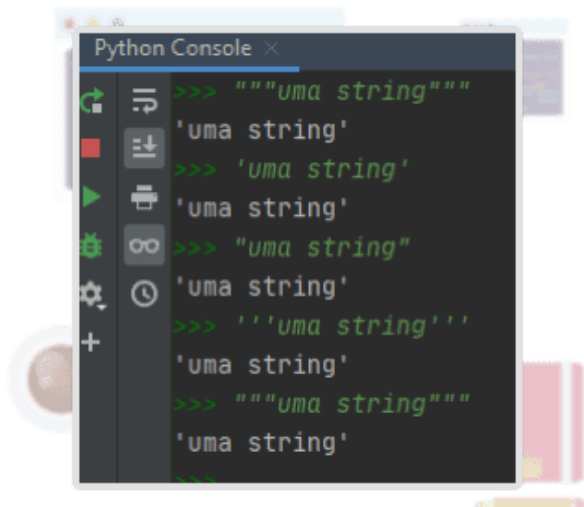
ASPAS SIMPLES TRIPLAS

```
"""uma string"""
```

ASPAS DUPLAS TRIPLAS

```
"""uma string"""
```

A Figura 30 ilustra um exemplo de delimitadores de strings:



Existem alguns métodos interessantes para tratar strings em Python. Entre eles, ressaltamos:

UPPER

Transforma todas as letras em maiúsculas.

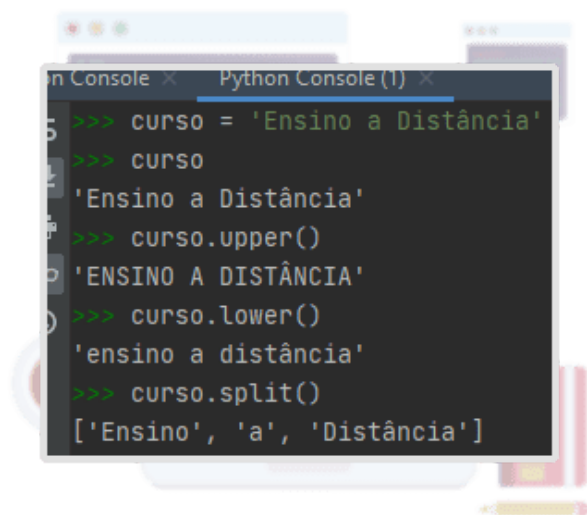
LOWER

Transforma todas as letras em minúsculas.

SPLIT

Quebra a string em substrings.

Veja os exemplos a seguir:



```
Python Console (1) x
>>> curso = 'Ensino a Distância'
>>> curso
'Ensino a Distância'
>>> curso.upper()
'ENSINO A DISTÂNCIA'
>>> curso.lower()
'ensino a distância'
>>> curso.split()
['Ensino', 'a', 'Distância']
```

Fonte: o autor (2020)

A lista gerada com o método **split()** tem três elementos, porque a string original tinha três palavras.

LISTAS

Listas são sequências mutáveis, normalmente usadas para armazenar coleções de itens homogêneos.

Uma lista pode ser criada de algumas maneiras, tais como:

[]

Usando um par de colchetes para denotar uma lista vazia.

[A], [A, B, C]

Usando colchetes, separando os itens por vírgulas.

[X FOR X IN ITERABLE]

Usando a compreensão de lista.

LIST() OU LIST(ITERABLE)

Usando o construtor do tipo list.

SAIBA MAIS

iterable pode ser uma sequência, um container que **suporte iteração ou um objeto iterador**. Por exemplo, **list('abc')** retorna ['a', 'b', 'c'] e **list((1, 2, 3))** retorna [1, 2, 3]. Se nenhum argumento for passado, o construtor cria uma lista vazia: [].

SUORTE ITERAÇÃO OU UM OBJETO ITERADOR

Um iterador é um objeto que contém um número contável de valores. Ele pode ser iterado, o que significa que podemos percorrer todos os valores.

TUPLAS

Tuplas são sequências imutáveis, tipicamente usadas para armazenar coleções de itens heterogêneos. Elas são aplicadas também quando é necessário utilizar uma sequência imutável de dados homogêneos. Uma tupla pode ser criada de algumas maneiras, tais como:

()

Usando um par de parênteses para denotar uma tupla vazia.

A, B, C OU (A, B, C)

Separando os itens por vírgulas.

TUPLE() OU TUPLE(ITERABLE)

Usando o construtor do tipo **tuple**.

Novamente, iterable pode ser uma sequência, um container que suporte iteração ou um objeto iterador. Por exemplo, **tuple('abc')** retorna ('a', 'b', 'c') e **tuple([1, 2, 3])** retorna (1, 2, 3). Se nenhum argumento for passado, o construtor cria uma tupla vazia: **()**.

⊕ ATENÇÃO

Note que o uso das vírgulas é o que gera a tupla, e não o uso de parênteses. Os parênteses são opcionais, exceto no caso em que queremos gerar uma tupla vazia.

RANGE

O tipo **range** representa uma sequência imutável de números e frequentemente é usado em *loops* de um número específico de vezes, como o **for**.

Ele pode ser chamado de maneira simples, apenas com um argumento. Nesse caso, a sequência começará em 0 e será incrementada de uma unidade até o limite do parâmetro passado (exclusive). Por exemplo, **range(3)** cria a sequência (0, 1, 2).

Para que a sequência não comece em 0, podemos informar o início e o fim como parâmetros, lembrando que o parâmetro fim não entra na lista (exclusive o fim). O padrão é incrementar cada termo em uma unidade. Ou seja, a chamada **range(2, 7)** cria a sequência (2, 3, 4, 5, 6).

⊕ SAIBA MAIS

Também é possível criar sequências mais complexas, indicando os parâmetros de início, fim e passo, nessa ordem. O passo é o valor que será incrementado de um termo para o próximo. Por exemplo, **range(2, 9, 3)** cria a sequência (2, 5, 8).

```
13     $(".function-filters-outer").toggleClass("show-filters-window")
14   }}, $(document).on("click.filter_log_click", ".filter-option-log", function() {
15     var a = $(this),
16         b = a.data(),
17         c = function.state.filters[b.filter_name],
18         d = function.state.options[b.filter_name],
19         f = $(".ub_filter_name_" + b.filter_name);
20     if (!("price_range" === b.filter_name)) c.splice(c.indexOf(b.value + ""), 1);
21     else if ("range_min" === b.option_name) c.filter_price[0] = d.range_min.value;
22     filter_price: [c.filter_price[0], null]
23   });
24   else if ("range_max" === b.option_name) c.filter_price[1] = d.range_max.value;
25   filter_price: [null, c.filter_price[1]]
26   });
27   else {
28     var g = c.separate_checkboxes.findIndex(i => i.name === b.option_name);
29     if (-1 < g) {
30       var h = c.separate_checkboxes.findIndex(i => i.name === b.option_name);
```

Fonte: BEST-BACKGROUNDS/Shutterstock

OPERADORES SEQUENCIAIS COMUNS

Os operadores sequenciais permitem a manipulação dos tipos sequenciais, inclusive as strings. Vale ressaltar a sobrecarga dos operadores **+** e *****, que realizam operações diferentes quando os operandos são numéricos ou sequenciais.

★ EXEMPLO

O operador **==** verifica se as strings dos dois lados são iguais. Porém, os operadores **<** e **>** comparam as strings usando a ordem do dicionário.

A tabela a seguir traz um pequeno conjunto dos operadores disponíveis em Python para manipulação de sequências. Lembre-se de que você pode utilizar o utilitário **help** no Python Console para verificar a lista completa. Para isso, basta digitar **help(str)** e pressionar [ENTER] no teclado.

Atenção! Para visualização completa da tabela utilize a rolagem horizontal

Uso	Resultado
<code>x in s</code>	True se x for um subconjunto de s
<code>x not in s</code>	False se x for um subconjunto de s
<code>s + t</code>	Concatenação de s e t
<code>n*s</code>	Concatenação de n cópias de s
<code>s[i]</code>	Caractere de índice i em s
<code>len(s)</code>	Comprimento de s
<code>min(s)</code>	Menor item de s
<code>max(s)</code>	Maior item de s

Tabela 8 – Operadores sequenciais

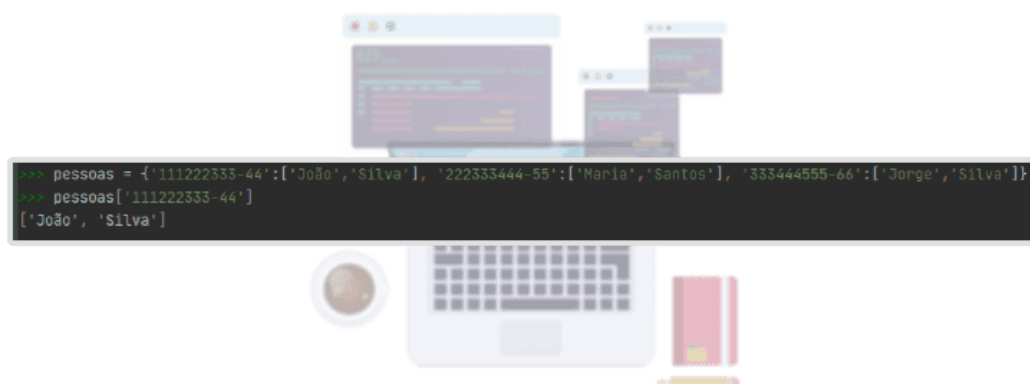
DICIONÁRIOS

Os dicionários permitem que itens de uma sequência recebam índices definidos pelo usuário. Um dicionário contém pares de (chave, valor). O formato geral de um objeto dicionário é:

```
{<chave 1>:<valor 1>, <chave 2>:<valor 2>, ..., <chave i>:<valor i>}
```

★ EXEMPLO

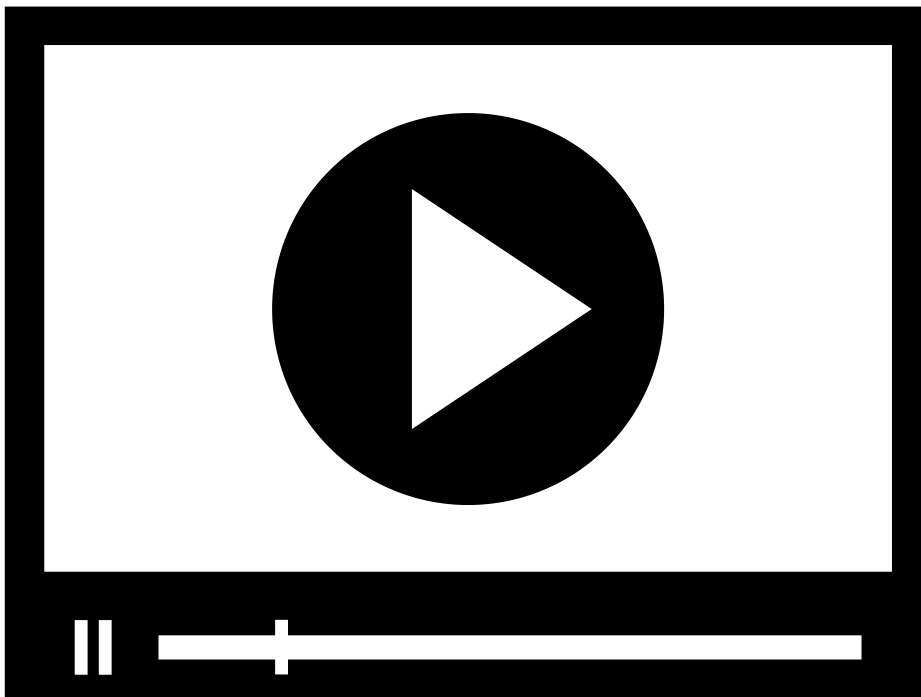
Poderíamos criar um dicionário em que cada pessoa fosse representada pelo seu CPF, com nome e sobrenome. Para isso, teríamos:



Fonte: o autor (2020)

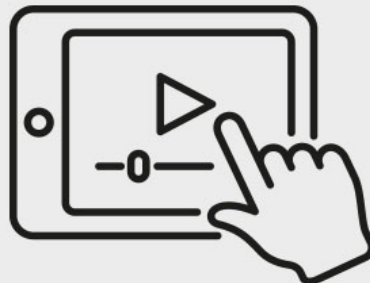
📷 Figura 32 - Dicionários - Fonte: o autor (2020)

Na figura 32, o dicionário tem 3 entradas. Observe como foi possível recuperar nome e sobrenome de uma entrada, baseado na chave informada '111222333-44'.



Para ver uma demonstração no Python dos **tipos sequenciais** e dos **dicionários**, assista ao vídeo a seguir.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



PRECEDÊNCIA DE OPERADORES

Ao escrever uma expressão algébrica, o programador pode utilizar a precedência de operadores existente em Python (implícita) ou explicitar a ordem em que ele deseja que a expressão seja avaliada.

★ EXEMPLO

Por exemplo, a expressão **$3 + 2 * 5$** tem como resultado 25 ou 13? Aprendemos no ensino fundamental que as operações de produto e divisão têm precedência sobre as operações de soma e subtração. Ou seja, um produto será realizado antes de uma soma, na mesma expressão. Assim, a expressão acima

tem como resultado 13. Isso ocorre sempre que não forem explicitadas outras relações de precedência com o uso de parênteses. Caso o programador quisesse forçar que a soma ocorresse primeiro, ele deveria escrever assim: $(3 + 2) * 5$.

Sempre que o programador quiser forçar a ocorrência de uma operação antes de outras, ele pode utilizar os parênteses para aumentar a prioridade sobre ela. A tabela a seguir traz as relações de precedência entre os operadores, com as linhas mais altas tendo prioridade sobre as linhas mais baixas. Ou seja, elas ocorrem primeiro. Dentro da mesma linha, a precedência é da esquerda para a direita.

Atenção! Para visualização completa da tabela utilize a rolagem horizontal

Operador	Descrição
[expressões ...]	Definição de lista
x[], x[índice : índice]	Operador de indexação
**	Exponenciação
+x, -x	Sinal de positivo e negativo
*, /, //, %	Produto, divisão, divisão inteira, resto
+, -	Soma, subtração
in, not in, <, <=, >, >=, <>, !=, ==	Comparações, inclusive a ocorrência em listas
not x	Booleano NOT (não)
and	Booleano AND (e)
or	Booleano OR (ou)

Tabela 9 - Precedência de operadores

★ ATENÇÃO

É importante ficar atento ao uso correto dos operadores, respeitando a precedência entre eles, para evitar que algum cálculo seja realizado corretamente, mas com resultado diferente do esperado por você ao programar.

CONVERSÕES DE TIPOS

Quando temos tipos diferentes envolvidos na mesma expressão, o Python converte implicitamente cada operando para o tipo mais abrangente envolvido na expressão. Estamos usando a palavra abrangente, mas poderíamos falar que existem tipos que englobam (ou contêm) outros.

★ EXEMPLO

Um número do tipo **int** pode ser visto como um **float** com a parte decimal nula. Porém, o inverso não é verdade. Ou seja, o conjunto dos inteiros (**int**) é um subconjunto do conjunto dos reais (**float**). Assim, a expressão **5 + 0.68** – que envolve um **int** e um **float** – tem como resultado **5.68**. O inteiro 5 é convertido pelo Python para o número de ponto flutuante 5.0 antes que a soma (de dois valores **float**) seja realmente efetuada.

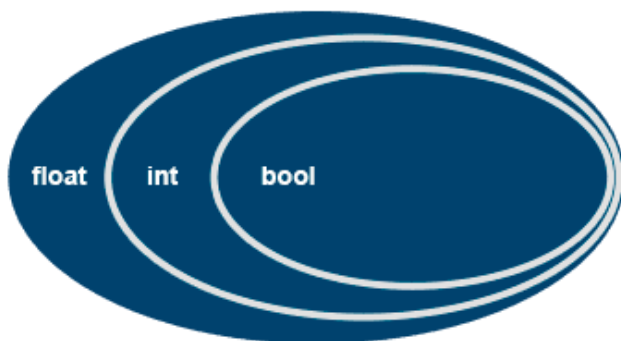
Uma conversão implícita não intuitiva é a dos valores booleanos **True** e **False** em inteiros, respectivamente, 1 e 0. Veja os exemplos a seguir:



Fonte: o autor (2020)

📷 Figura 33 - Conversão de tipos

Com isso, podemos perceber a seguinte relação entre os tipos **bool**, **int** e **float**:



Fonte: o autor (2020)

📷 Figura 34 - Relação entre os tipos bool, int e float

Além das conversões implícitas, o programador também pode usar as conversões explícitas, quando ele força que o valor seja tratado como de determinado tipo. Para isso, é necessário usar o construtor do tipo desejado, com o valor passado como parâmetro (entre parênteses). Veja o exemplo a seguir:



Fonte: o autor (2020)

📷 Figura 35 - Conversão explícita

O **int** 2 pode ser tratado naturalmente como o **float** 2.0, basta acrescentar a parte decimal nula. Porém, ao tentar tratar um **float** como **int**, ocorre a remoção da parte decimal.

★ ATENÇÃO

Fique atento, porque não é uma aproximação para o inteiro mais próximo, e sim o truncamento.

Agora que você já viu os principais tipos de dados suportados em Python, vamos exercitar e verificar o aprendizado.

VERIFICANDO O APRENDIZADO

1. CONSIDERE A EXPRESSÃO A SEGUIR: $2 + 3 - 4 ** 2 + 5 / 2 - 5 // 2$

ASSINALE A OPÇÃO COM O VALOR CORRETO DESSA EXPRESSÃO EM PYTHON.

A) -10.5

B) -1

C) 1.5

D) 2

2. (ADAPTADA DE COMPERVE/2019/UFRN/ENGENHARIA DA COMPUTAÇÃO) PYTHON É UMA LINGUAGEM INTERPRETADA MUITO UTILIZADA. NÃO REQUER TIPAGEM DE VARIÁVEIS E SUA SINTAXE INDENTADA FAVORECE A ORGANIZAÇÃO DO CÓDIGO. UMA DAS SUAS FUNCIONALIDADES MAIS PODEROSAS SÃO AS LISTAS. CONSIDERE O CÓDIGO EM PYTHON DO QUADRO ABAIXO:

```
1 a = ['UF'] + ['RN']
2 len(a)
3 b = ['4']*4
4 len(b)
```

A SAÍDA CORRETA CORRESPONDENTE ÀS LINHAS 2 E 4 DO CÓDIGO É:

A) 2 e 4.

B) 4 e 16.

C) 2 e 16.

D) 4 e 4.

GABARITO

1. Considere a expressão a seguir: $2 + 3 - 4 ** 2 + 5 / 2 - 5 // 2$

Assinale a opção com o valor correto dessa expressão em Python.

A alternativa "A" está correta.

Lembre-se que o operador `**` tem precedência maior do que os operadores `/` e `//`, os quais, por sua vez, têm precedência sobre `+` e `-`. Ou seja, primeiro será efetuada a exponenciação ($4**2$), depois as divisões, comum ($5/2$) e inteira ($5//2$), para posteriormente serem efetuadas as somas e subtrações.

2. (Adaptada de COMPERVE/2019/UFRN/Engenharia da Computação) Python é uma linguagem interpretada muito utilizada. Não requer tipagem de variáveis e sua sintaxe indentada favorece a

organização do código. Uma das suas funcionalidades mais poderosas são as listas. Considere o código em Python do quadro abaixo:

```
1 a = ['UF'] + ['RN']
2 len(a)
3 b = ['4']*4
4 len(b)
```

A saída correta correspondente às linhas 2 e 4 do código é:

A alternativa "A " está correta.

O operador + realiza operações de soma para tipos numéricos e concatenação para tipos sequenciais. Assim, a variável **a** na linha 1 passa a ser composta dos itens 'UF' e 'RN'. Assim, a chamada **len(a)** retorna o tamanho 2, número de elementos de **a**. De forma semelhante, o operador * realiza operações de multiplicação para tipos numéricos e concatenação de cópias para tipos sequenciais. Assim, a variável **b** na linha 3 passa a ser a lista ['4', '4', '4', '4']. E a chamada **len(b)** retorna o tamanho 4, número de elementos de **b**.

MÓDULO 4

🕒 Identificar as formas de atribuição, de entrada e saída de dados em Python

CONCEITOS

Já vimos, de maneira básica, como podemos atribuir valor a uma variável, no módulo 2. Vamos agora conhecer outras formas de atribuição.



Fonte: BEST-BACKGROUNDS/Shutterstock

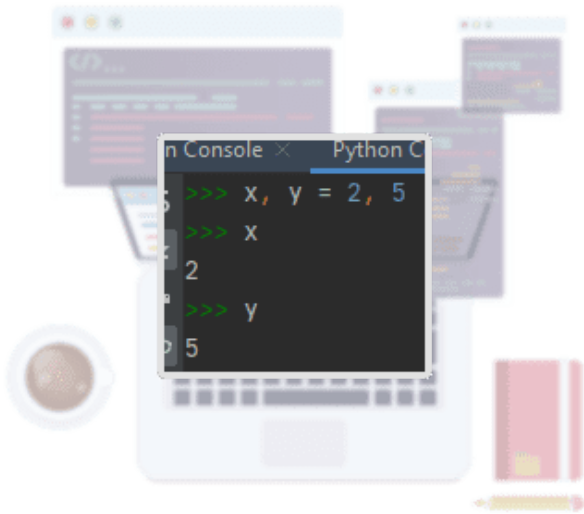
SENTENÇAS DE ATRIBUIÇÃO

ATRIBUIÇÃO SIMPLES

Chamamos de atribuição simples a forma que já utilizamos neste tema, com uma expressão parecida com **x = 10**. Nessa atribuição, a variável **x** recebe o valor 10.

ATRIBUIÇÃO MÚLTIPLA

Python também permite a atribuição múltipla, ou seja, mais de uma variável receber atribuição na mesma linha. Veja o exemplo na figura 36:



Fonte: o autor (2020)

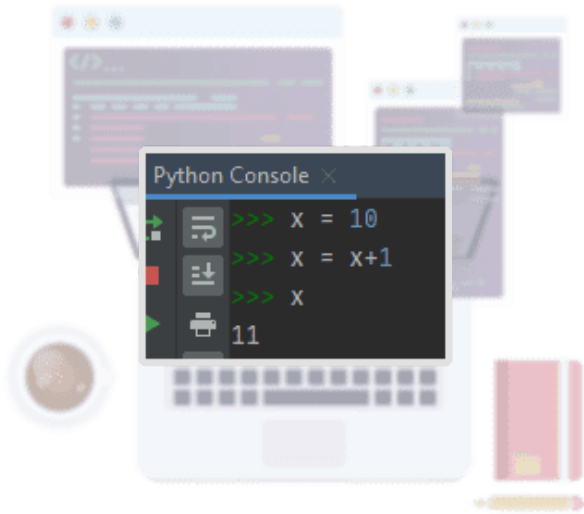
📷 Figura 36 - Atribuição múltipla

★ ATENÇÃO

Observe que as variáveis **x** e **y** receberam atribuição na mesma instrução, com a variável **x** armazenando o valor 2, e a variável **y** armazenando o valor 5.

OPERADORES DE ATRIBUIÇÃO COMPOSTOS

Os operadores de atribuição compostos executam operações matemáticas e atualizam o valor da variável utilizada. Por exemplo, veja a figura 37:



Fonte: o autor (2020)

📷 Figura 37 - Atribuição e atualização de variável

A variável **x** inicialmente recebeu o valor 10. Em seguida, a instrução **x = x + 1**, que causa estranheza quando lembramos da matemática aprendida ao longo da vida, é muito comum quando estamos programando. Essa instrução significa “acrescente uma unidade ao valor de x e guarde este resultado na própria variável **x**”. Como **x** valia 10, o resultado do lado direito do operador (=) é 11. Esse resultado é, então, armazenado na própria variável **x**.

Essa operação de acrescentar determinado valor a uma variável e armazenar o resultado na própria variável poderia ser feita com o operador **+=** (mais igual). Veja a figura 38:



Fonte: o autor (2020)

📷 Figura 38 - Operador mais igual

Na tabela 10, estão os operadores compostos disponíveis em Python. Considere a variável **x**, com o valor inicial 10, para verificar os resultados.

Atenção! Para visualização completa da tabela utilize a rolagem horizontal

Nome	Símbolo usado	Exemplo	
		Instrução	Resultado
Mais igual	<code>+=</code>	<code>x += 2</code>	x passa a valer 12
Menos igual	<code>-=</code>	<code>x -= 2</code>	x passa a valer 8
Vezes igual	<code>*=</code>	<code>x *= 2</code>	x passa a valer 20
Dividido igual	<code>/=</code>	<code>x /= 2</code>	x passa a valer 5
Módulo igual	<code>%=</code>	<code>x %= 3</code>	x passa a valer 1

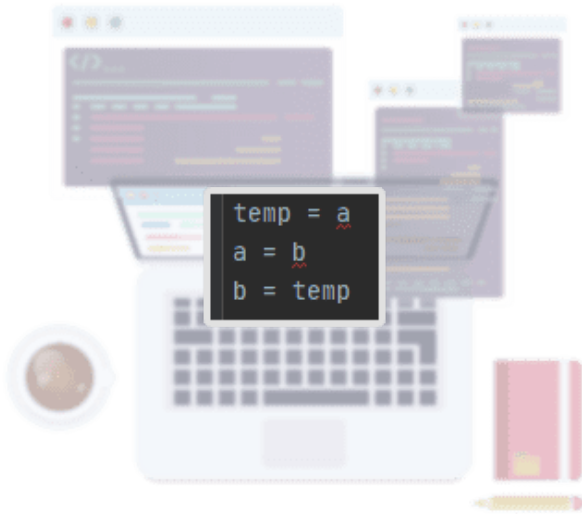
Tabela 10 - Operadores compostos

★ ATENÇÃO

Diferente de C, em Python não é possível incrementar ou decrementar uma variável com um operador unário, como o `++` ou `--`.

TROCA DE VARIÁVEIS

Um dos problemas iniciais que envolvem atribuição de valores a variáveis é a troca entre duas delas. Suponha que as variáveis **a** e **b** armazenem, respectivamente, os valores **1** e **2**. Caso quiséssemos inverter os valores em linguagens como C ou Java, seria necessário usar uma variável auxiliar, com uma sequência de instruções exibida na figura a seguir:



Fonte: o autor (2020)

📷 Figura 39 - Troca de variáveis 1

Em Python, é possível fazer essa troca de uma maneira muito mais fácil. Veja o uso da atribuição múltipla, nesse caso, na figura a seguir:



Fonte: o autor (2020)

📷 Figura 40 - Troca de variáveis 2

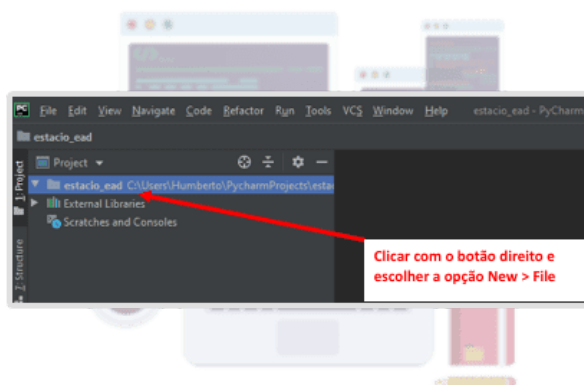
O PRIMEIRO PROGRAMA EM PYTHON

Para escrever um programa em Python, será essencial utilizar as formas de **saída de dados** para exibir ao usuário mensagens e resultados de operações. Caso você deseje que o usuário informe algum dado para que seu programa processe, será necessário utilizar as formas de **entrada de dados**.



Fonte: Freepik

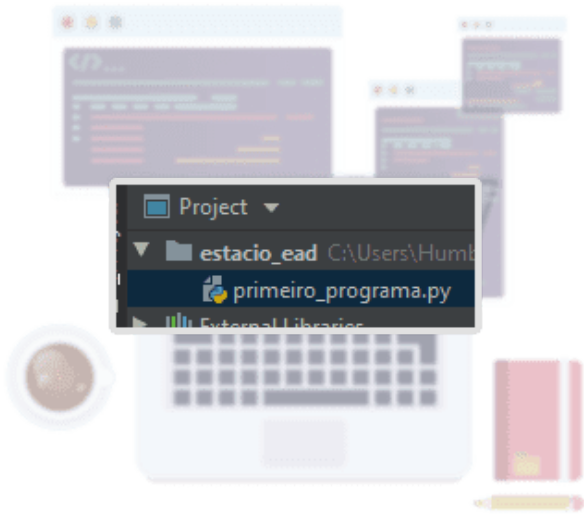
Para criar seu primeiro programa, clique com o botão direito do mouse no nome do projeto, na guia de navegação do lado esquerdo. Em seguida, escolha a opção New > File.



Fonte: o autor (2020)

📷 Figura 41 - Novo arquivo no PyCharm

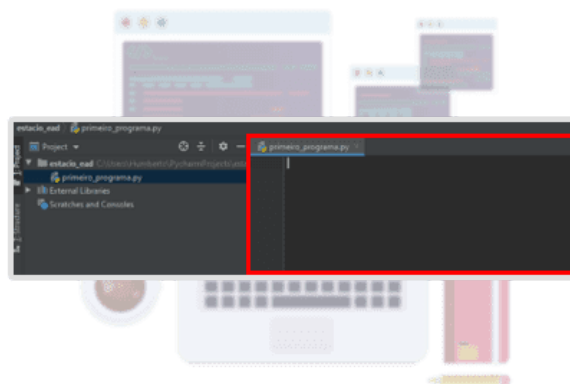
Atribua um nome ao seu arquivo. Neste primeiro exemplo, vamos chamar de *primeiro_programa.py*



Fonte: o autor (2020)

📷 Figura 42 - primeiro_programa.py

Ao nomear o arquivo, será aberta uma nova aba, do lado direito, com o espaço para que você efetivamente digite as instruções.



Fonte: o autor (2020)

📷 Figura 43 - Aba de codificação

SAÍDA DE DADOS COM A FUNÇÃO PRINT()

A função **print()** em Python atua de forma semelhante à **printf()** em C. Para um programador iniciante, as maiores diferenças entre elas são:

Duas chamadas da **print()** em Python são impressas na tela em linhas diferentes, sem a necessidade do uso do caractere 'n' para pular a linha, como ocorre na **printf()** em C.

Uma chamada da **print()** em Python permite a impressão de valores de variáveis sem a indicação do formato, como ocorre na **printf()** em C, quando precisamos escrever %c, %d ou %f, por exemplo.



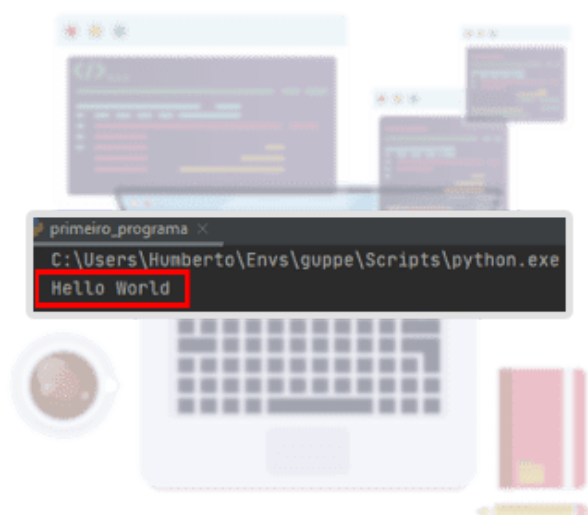
Fonte: Freepik

Para escrever seu Hello World em Python, digite a seguinte linha, exatamente como está escrita:

```
print("Hello World")
```

Em seguida, clique com o botão direito do mouse sobre o nome do programa e escolha a opção **Run** ‘**primeiro_programa**’. Também é possível executar com a combinação de teclas CTRL+Shift+ F10.

Após executar, observe o console na figura 44:



Fonte: o autor (2020)

📷 Figura 44 - Execução do Hello World em Python

Veja que foi impresso no console exatamente o que colocamos entre aspas, ao chamar a função **print()**. Essa é a primeira forma de saída de dados: usar a função **print()** com uma string sendo passada como parâmetro (entre os parênteses). É importante perceber que a função **print()**, além de imprimir a string, também salta o cursor para a próxima linha.

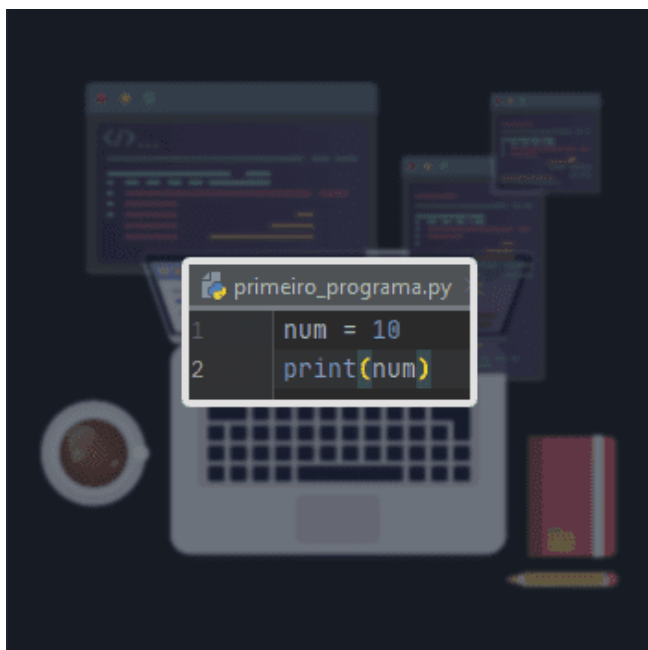
Como você deve ter percebido, o que a função **print()** recebeu entre parênteses foi uma string. Ou seja, poderíamos ter passado para ela uma string definida anteriormente, como no exemplo a seguir:



Fonte: o autor (2020)

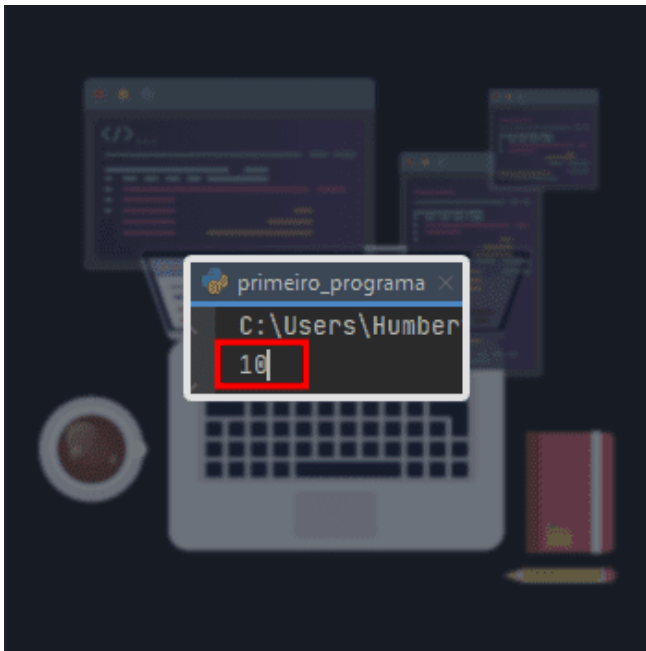
📷 Figura 45 - Hello World com string

Também poderíamos ter passado como parâmetro uma variável definida anteriormente. A função **print()** vai trabalhar com o valor dessa variável. Observe as figuras 46 e 47:



Fonte: o autor (2020)

📷 Figura 46 - Print de variável

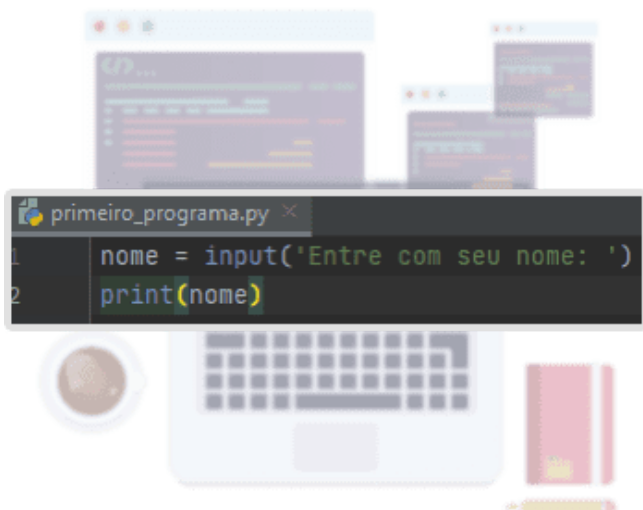


Fonte: o autor (2020)

📷 Figura 47 - Execução do print com variável

ENTRADA DE DADOS COM A FUNÇÃO INPUT()

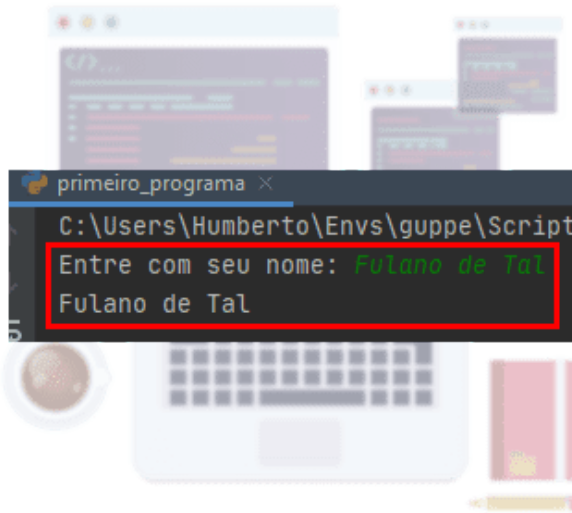
Quando o programador quiser que o usuário entre com algum valor, ele deverá exibir na tela o seu pedido. Em C, é necessário utilizar a função **printf()** para escrever a solicitação ao usuário e a função **scanf()** para receber a entrada e armazenar em uma variável. Em Python, é possível utilizar a função **input()**. Ela tanto exibe na tela o pedido, como permite que o valor informado pelo usuário seja armazenado em uma variável do seu programa. Analise a figura 48:



Fonte: o autor (2020)

📷 Figura 48 - A função input()

A linha 1 fará com que a frase **Entre com seu nome:** seja exibida no console, mas a execução do programa fica travada até que o usuário aperte [ENTER] no teclado. Tudo o que foi digitado até o [ENTER] vai ser armazenado na variável **nome**. A linha 2 fará a exibição do conteúdo da variável **nome**. Veja o resultado no console, com o usuário tendo digitado **Fulano de Tal**.



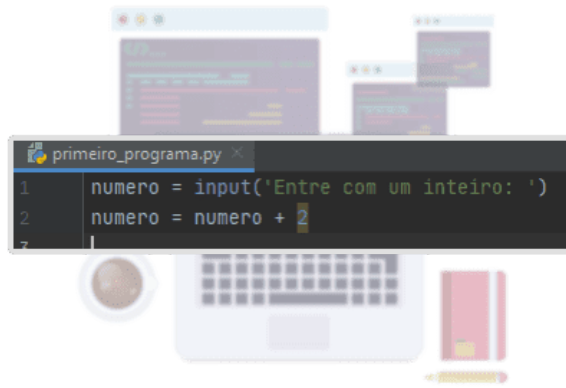
Fonte: o autor (2020)

📷 Figura 49 - Execução da entrada de dados

★ ATENÇÃO

É importantíssimo perceber que a função **input()** trata tudo o que for digitado pelo usuário como uma string, armazenando na variável designada pelo programador para isso. Mesmo que o usuário entre com apenas uma letra ou um número, isso será armazenado como uma string na variável.

Vamos analisar o exemplo a seguir:



Fonte: o autor (2020)

📷 Figura 50 - A função input() e operação numérica

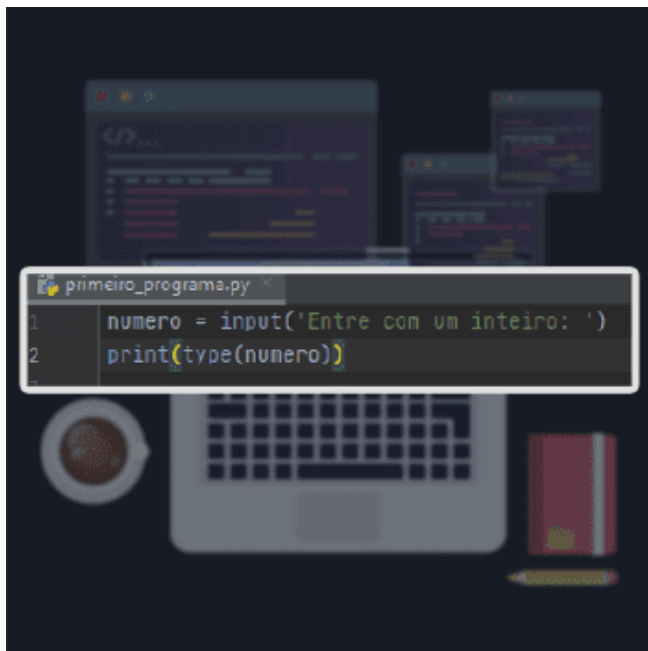
Veja o console quando o programa é executado:



Fonte: o autor (2020)

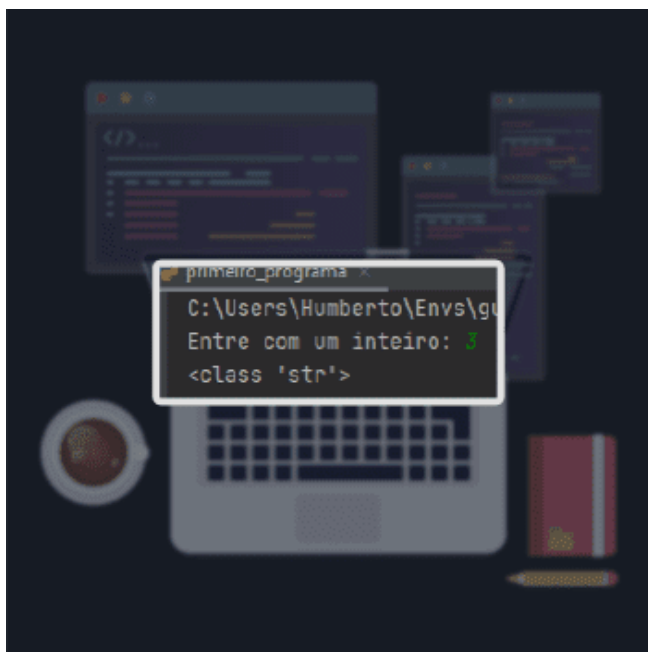
📷 Figura 51 - Erro com a função input()

O usuário digitou 3 e [ENTER]. Mesmo sendo um valor, a variável **numero** trata como a string '3'. Isso impede que seja realizada a operação de soma com o inteiro 2, por exemplo. Poderíamos também usar a instrução **print(type(numero))** na linha 2 para confirmar. Veja:



Fonte: o autor (2020)

📷 Figura 52

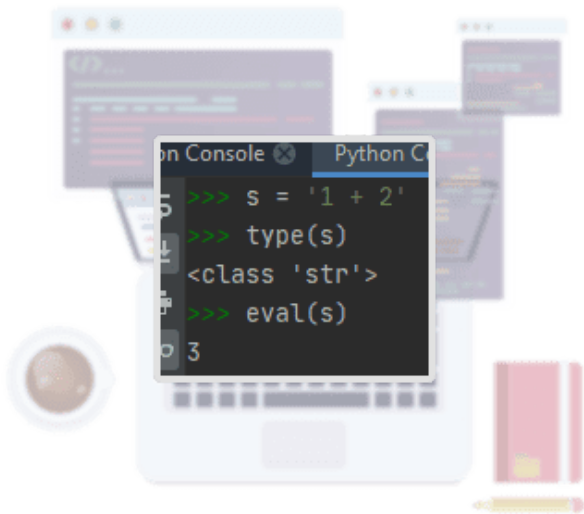


Fonte: o autor (2020)

📷 Figura 53

A FUNÇÃO EVAL()

A função **eval()** recebe uma string, mas trata como um valor numérico. Veja o exemplo a seguir:

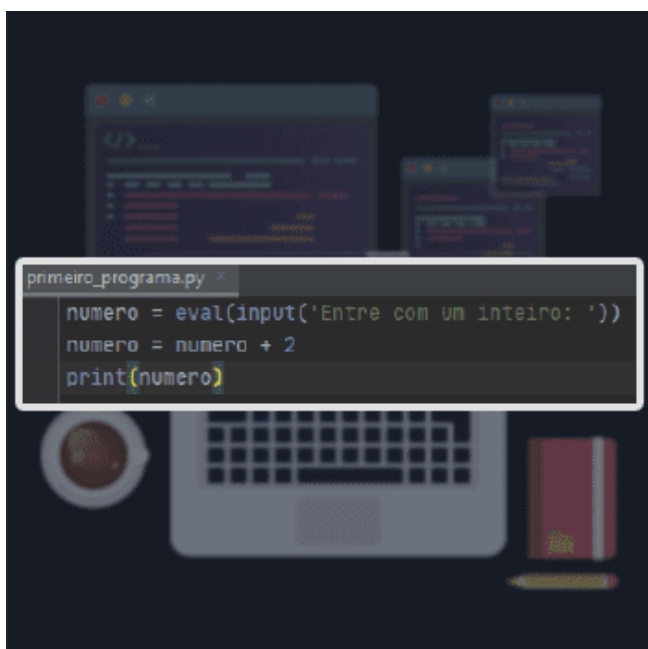


Fonte: o autor (2020)

📷 Figura 54

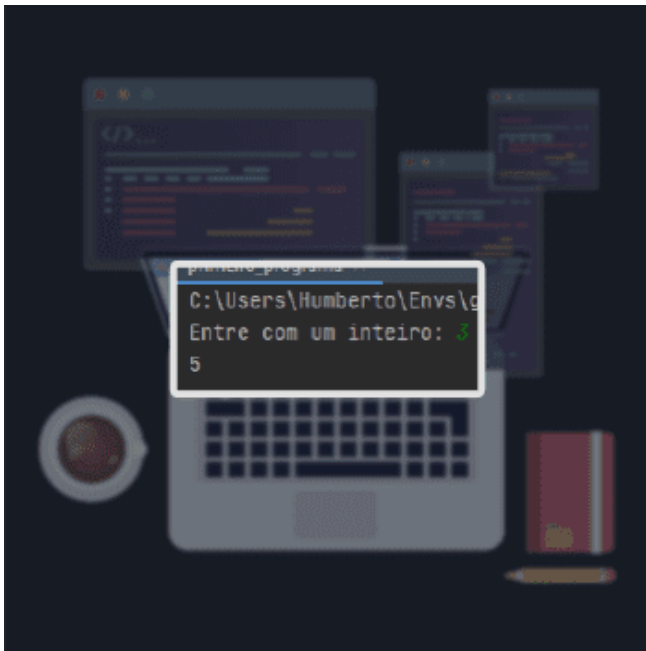
Mesmo tendo recebido a string '1+2' como parâmetro, a função **eval()** efetuou a soma de 1 com 2. Observe que confirmamos que s é uma string com a instrução `type(s)`.

Para tratar a entrada do usuário como um número e, com isso, realizar operações algébricas, por exemplo, é necessário utilizar a função **eval()** em conjunto com a **input()**. Veja o próximo exemplo:



Fonte: o autor (2020)

📷 Figura 55



Fonte: o autor (2020)

📷 Figura 56

MÃO NA MASSA

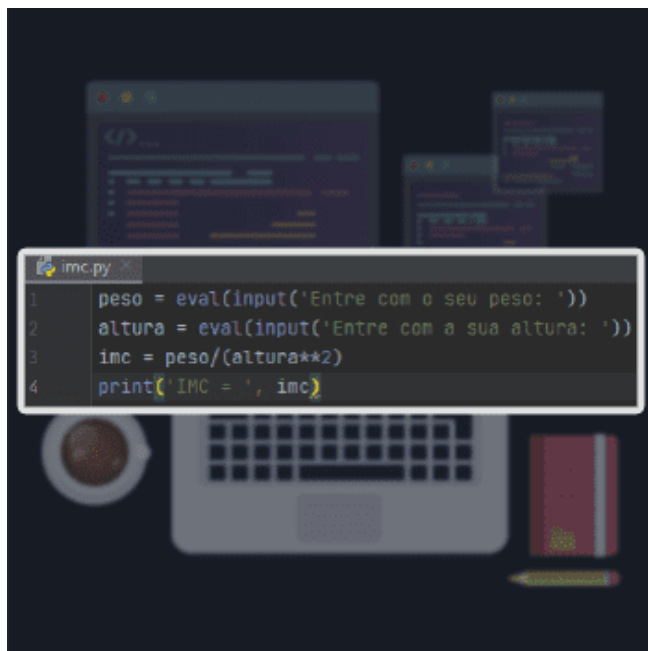
COMO EXERCÍCIO PRÁTICO, TENDE ESCREVER UM PROGRAMA PARA CALCULAR E INFORMAR O IMC (ÍNDICE DE MASSA CORPÓREA) DO USUÁRIO, QUE DEVERÁ FORNECER SEU PESO E SUA ALTURA.

LEMBRE-SE QUE O IMC É CALCULADO PELA

FÓRMULA: $IMC = \frac{peso}{altura^2}$

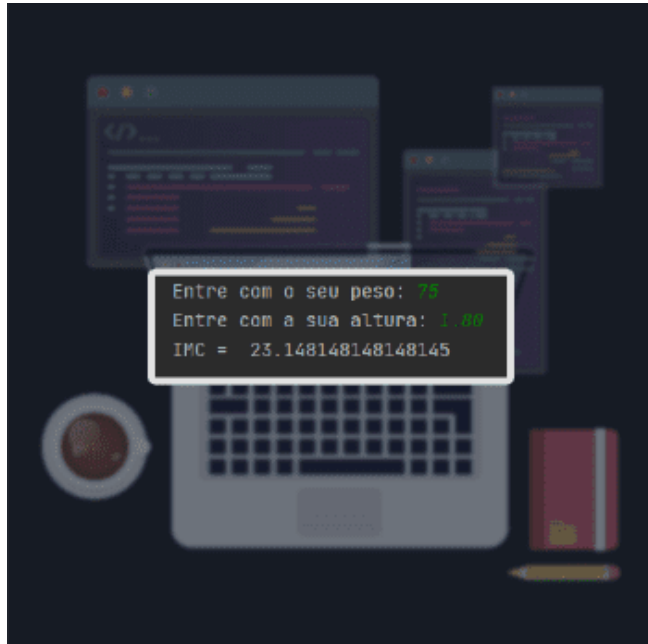
RESPOSTA

Uma solução simples é a da figura 57:



Fonte: o autor (2020)

📷 Figura 57



Fonte: o autor (2020)

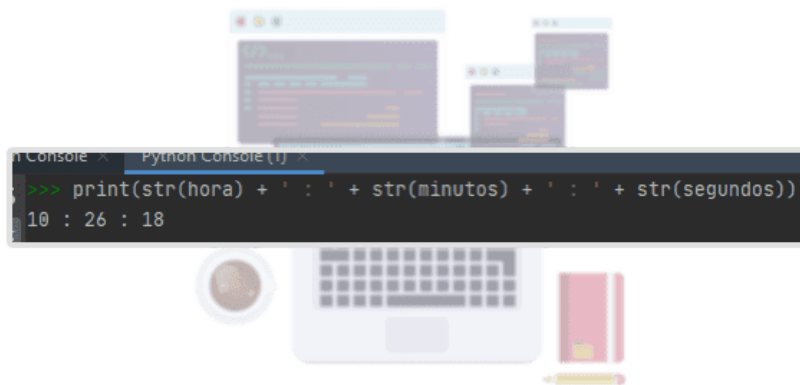
📷 Figura 58

SAÍDA FORMATADA DE DADOS

Quando desejamos que a saída siga determinado padrão – por exemplo, de hora ou de data – existem algumas possibilidades para usar a função **print()**. É sempre possível utilizar a concatenação de strings, com o operador +, para montar a frase como quisermos. Suponha que tenhamos as seguintes variáveis:

hora = 10 minutos = 26 segundos = 18

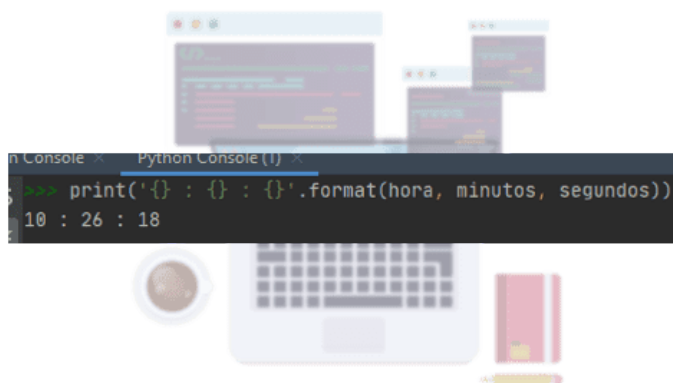
Poderíamos chamar a função **print()** com o separador : da seguinte forma:



Fonte: o autor (2020)

📷 Figura 59

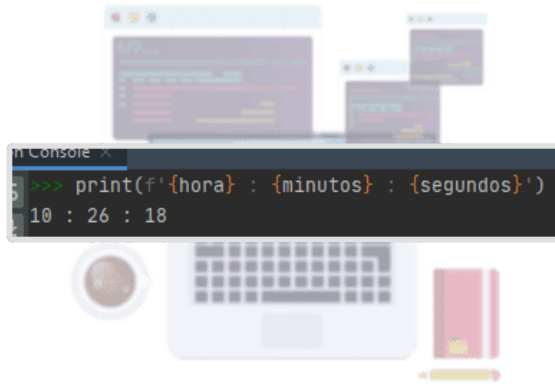
Porém, existe outra possibilidade, usando o método **format()**. Ele permite que a chamada à função **print()** fique muito parecida com as chamadas à função **printf()** em C, com passagem de parâmetros a serem colocados em ordem na string. Com o método **format()**, podemos montar a string com as chaves {} indicando onde entrarão valores, passados como parâmetros separados por vírgulas.



Fonte: o autor (2020)

📷 Figura 60

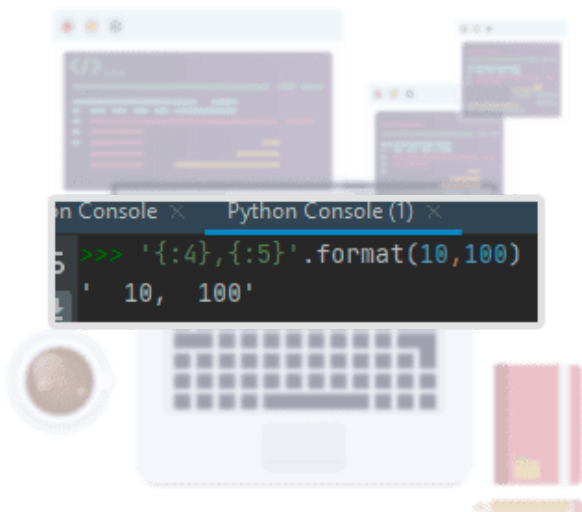
É importante observar que a quantidade de chaves precisa ser igual à quantidade de variáveis passadas como parâmetros no método **format()**. Seria muito bom se não precisássemos nos preocupar com essa correspondência para evitar erros bobos. E isso é possível! Para tornar a saída formatada ainda mais intuitiva, basta utilizar a letra 'f' no início dos parâmetros da função **print()** e colocar cada variável dentro das chaves na posição em que deve ser impressa. Veja como fica ainda mais fácil entender:



Fonte: o autor (2020)

📷 Figura 61

Também é possível especificar a largura de campo para exibir um inteiro. Se a largura não for especificada, ela será determinada pela quantidade de dígitos do valor a ser impresso. Veja a figura 62:



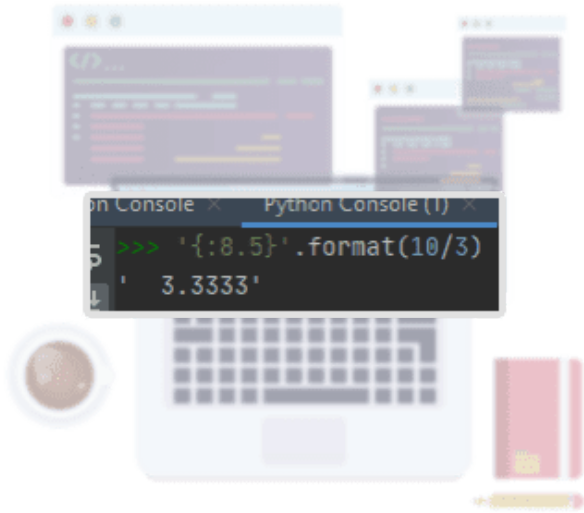
Fonte: o autor (2020)

📷 Figura 62

Observe que os valores 10 e 100 foram impressos com espaços em branco à esquerda. Isso ocorreu porque definimos que a primeira variável deveria ser impressa com 4 espaços com `{:4}` (2 foram ocupados e 2 ficaram em branco), e que a segunda variável deveria ser impressa com 5 espaços com `{:5}` (3 foram ocupados e 2 ficaram em branco).

Também é válido perceber que o padrão é alinhar os valores à direita do espaço reservado para a impressão da variável.

O método **format()** também pode ser usado para imprimir valores de ponto flutuante com a precisão definida. Veja a figura 63:



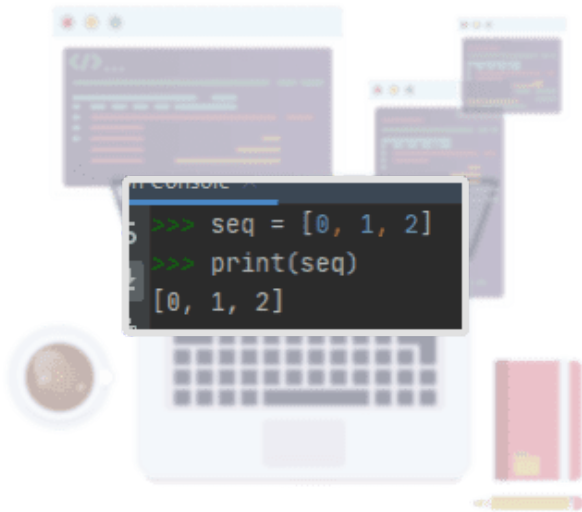
Fonte: o autor (2020)

📷 Figura 63

Ao usar **{:8.5}**, estamos determinando que a impressão será com 8 espaços, mas apenas 5 serão utilizados.

IMPRESSÃO DE SEQUÊNCIAS

Python também permite a impressão de sequências com mais possibilidades que C, incluindo as strings. Para imprimir um vetor em C, por exemplo, precisamos chamar a **printf()** item a item. Em Python, basta chamar a função **print()** passando como parâmetro a sequência. Veja a figura 64:



Fonte: o autor (2020)

📷 Figura 64

Para imprimir uma substring, por exemplo, basta utilizar os colchetes para indicar o intervalo de índices que deve ser impresso. Vale lembrar que o primeiro caractere da string é indexado com 0. Veja a figura 65:



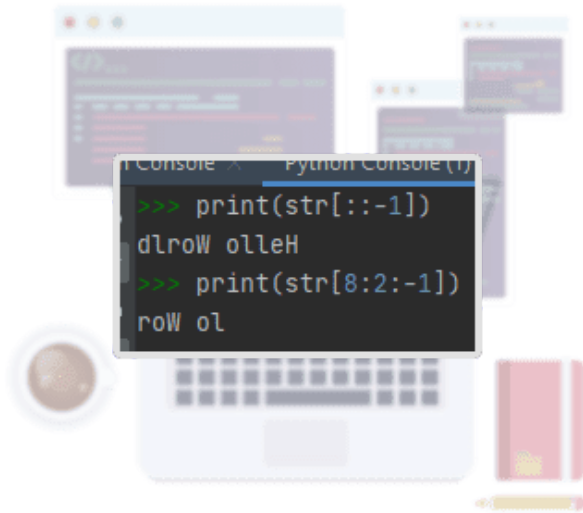
Fonte: o autor (2020)

📷 Figura 65

★ **ATENÇÃO**

Usar **[0:4]** provoca a impressão dos índices 0, 1, 2 e 3, mas não do índice 4. Analogamente, usar **[2:8]** provoca a impressão dos índices de 2 a 7, mas não do 8.

Também é possível imprimir a string como lida da direita para a esquerda. Para isso, deve-se utilizar **[: :-1]**. Esse valor -1 indica que a leitura dos caracteres será feita no sentido oposto ao tradicional. Observe a figura 66:

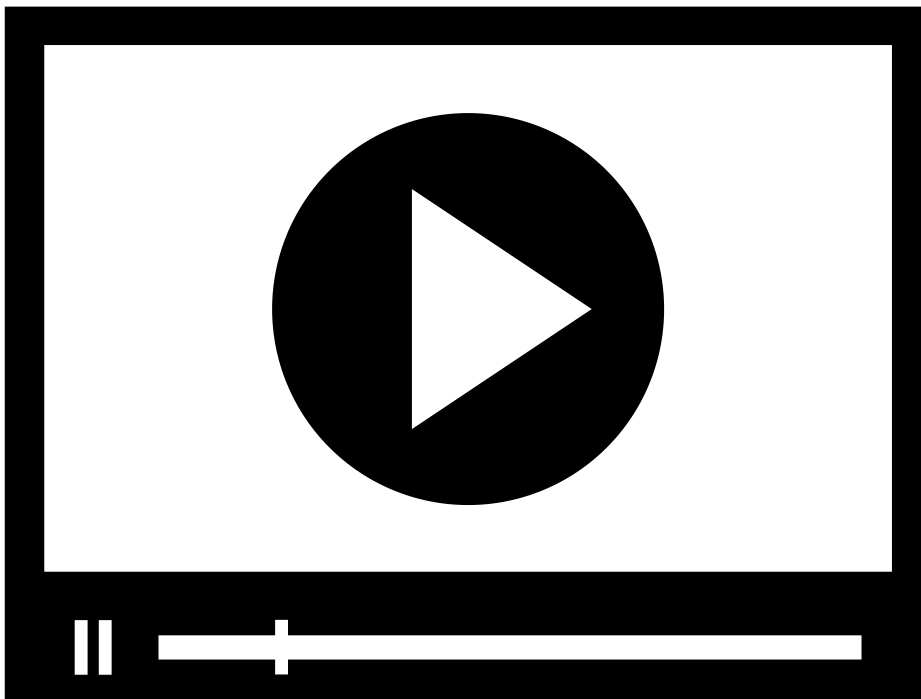


Fonte: o autor (2020)

📷 Figura 66

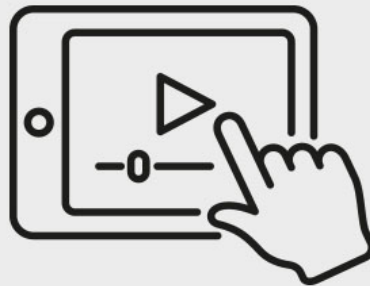
★ ATENÇÃO

Fique atento quando utilizar o intervalo na impressão no sentido inverso, porque os limites do intervalo devem respeitar esse sentido.



Para ver uma demonstração dos procedimentos de **entrada e saída no Python**, assista ao vídeo a seguir.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



VERIFICANDO O APRENDIZADO

1. (2015/PGE-RO/TÉCNICO DA PROCURADORIA/TECNOLOGIA DA INFORMAÇÃO) NA LINGUAGEM PYTHON, UM COMANDO COMO

A=INPUT("XXX")

PROVOCA:

A) A associação à variável “a” de uma função denominada “XXX” que pertence à biblioteca “input”.

- B)** A criação de uma lista de valores denominada “a” cujo elemento inicial é a string “XXX”.
- C)** A leitura de um valor do arquivo de entrada correntemente designado de acordo com um formato expresso pela string “XXX”.
- D)** Um prompt no dispositivo de saída e a leitura de um valor que é armazenado na variável “a”.

2. (2015/TJ-BA/ANALISTA JUDICIÁRIO/TECNOLOGIA DA INFORMAÇÃO/REAPLICAÇÃO) ANALISE O TRECHO DE PROGRAMA PYTHON APRESENTADO A SEGUIR.

```
L = [1,2,3,4,5,6,7,8]  
PRINT L[::-1]
```

AO SER EXECUTADO, O RESULTADO EXIBIDO É:

- A)** [1, 2, 3, 4, 5, 6, 7, 8]
- B)** [8]
- C)** []
- D)** [8, 7, 6, 5, 4, 3, 2, 1]

GABARITO

1. (2015/PGE-RO/Técnico da Procuradoria/Tecnologia da Informação) Na linguagem Python, um comando como

```
a=input("XXX")
```

provoca:

A alternativa **"D "** está correta.

A função **input()** tanto exhibe na tela a string “XXX”, como permite que o valor informado pelo usuário seja armazenado na variável **a**.

2. (2015/TJ-BA/Analista Judiciário/Tecnologia da Informação/Reaplicação) Analise o trecho de programa Python apresentado a seguir.

```
L = [1,2,3,4,5,6,7,8]
```

```
print L[::-1]
```

Ao ser executado, o resultado exibido é:

A alternativa "**D**" está correta.

A impressão da sequência L com a chamada L[::-1] é feita percorrendo toda a sequência L, em sentido inverso.

CONCLUSÃO

CONSIDERAÇÕES FINAIS

Neste tema, você conheceu as principais características da linguagem Python e alguns conceitos relativos ao uso de variáveis, bem como aspectos concernentes à vinculação, como tempo e ambientes. Além disso, viu o conceito de escopo de visibilidade, tanto estático como dinâmico.

Nos módulos, foi feita uma referência a tipos de dados, como **int**, **float** ou **string**. Você estudou ainda as formas de atribuição, além de ter escrito seu primeiro programa em Python, utilizando o que aprendeu e, também, as formas de entrada e saída de dados que a linguagem oferece.

O uso correto desses conceitos é essencial na sua jornada de formação como programador.

Recomendamos que fique atento aos detalhes e procure sempre programar de forma organizada. Isso vai evitar erros bobos e tornar sua experiência mais agradável.

Para ouvir um *podcast* sobre o assunto, acesse a versão online deste conteúdo.



REFERÊNCIAS

BELANI, G. **Programming Languages You Should Learn in 2020**. Consultado em meio eletrônico em: 26 mai. 2020.

PERKOVIC, L. **Introdução à computação usando Python**: um foco no desenvolvimento de aplicações. Rio de Janeiro: LTC, 2016.

PYCHARM. **The Python IDE for Professional Developers**. Consultado em meio eletrônico em: 15 jun. 2020.

PYTHON. **PEP 0 – Index of Python Enhancement Proposals (PEPs)**. Consultado em meio eletrônico em: 2 jun. 2020.

PYTHON. **Fractions – Rational Number**. Consultado em meio eletrônico em: 16 jun. 2020.

PYTHON. **Math – Mathematical Functions**. Consultado em meio eletrônico em: 16 jun. 2020.

EXPLORE+

Se você quiser ter mais exercícios para treinar e desafios mais complexos, recomendamos a visita ao site da **Python Brasil**.

Como vimos, dentre as PEPs, destaca-se a **PEP8**, que estabelece um guia de estilo de programação. Sugerimos que você pesquise mais sobre isso.

Para mais funções matemáticas, você pode utilizar os **módulos matemáticos math e fractions**.

CONTEUDISTA

Humberto Henriques de Arruda

 **CURRÍCULO LATTES**