

NÃO PODE FALTAR

▲  
Imprimir

# A LINGUAGEM PYTHON

Vanessa Cadan Scheffer

## PRIMEIROS PASSOS EM PYTHON

Vamos conhecer o poder dessa linguagem de programação.

0

Ver anotações



Fonte: Shutterstock.

**Deseja ouvir este material?**

Áudio disponível no material digital.

```
n [1]:print("hello world!")
```

```
hello world!
```

Diz a lenda dos programadores que, se você não imprimir o “hello world” quando começar a aprender uma linguagem de programação, você não conseguirá aprender nada da linguagem em questão (<https://cienciacomputacao.com.br>). Mito ou verdade? Por via das dúvidas, essa foi nossa primeira linha de comando, de muitas que aprenderemos nessa disciplina. Como você pode ver, um comando de

sintaxe "limpa", sem ponto e vírgula, sem colchetes, sem importação de bibliotecas é a essência da linguagem de programação Python, que foi criada para ter comandos e estruturas simples, de fácil leitura e compreensão.

Python foi criado no início dos anos 1990 por Guido van Rossum no Stichting Mathematisch Centrum (CWI), na Holanda, como sucessor de uma linguagem chamada ABC. Guido é o principal autor do Python, embora haja muitas contribuições de outros pesquisadores ([Python Reference Manual, 2020](#)). Guido passou por outras instituições, como a CNRI, em 1995, na Virgínia (Corporação para Iniciativas Nacionais de Pesquisa) e na BeOpen, em 2000, onde formou a BeOpen PythonLabs. Em outubro do mesmo ano, a equipe do PythonLabs mudou-se para a Digital Creations (agora Zope Corporation). Em 2001, a Python Software Foundation (PSF) foi formada, uma organização sem fins lucrativos criada especificamente para possuir a propriedade intelectual relacionada ao Python.

0  
Ver anotações

## POR QUE USAR PYTHON?

Segundo o guia de desenvolvimento para iniciantes Python ([Beginners Guide Overview](#)), Python é uma linguagem de programação orientada a objetos, clara e poderosa, comparável a Perl, Ruby, Scheme ou Java. Podemos destacar algumas características que tornam essa linguagem notável:

- Utiliza uma sintaxe elegante, facilitando a leitura dos programas que você escreve.
- É uma linguagem fácil de usar, o que torna o Python ideal para o desenvolvimento de protótipos e outras tarefas de programação ad-hoc, sem comprometer a manutenção.
- Vem com uma grande biblioteca padrão que suporta muitas tarefas comuns de programação, como se conectar a servidores da Web, pesquisar texto com expressões regulares, ler e modificar arquivos.

- Possui inúmeras bibliotecas que estendem seu poder de atuação.
- É uma linguagem interpretada, ou seja, uma vez escrito o código, este não precisa ser convertido em linguagem de máquina por um processo de compilação.
- Permite atribuição múltipla. Podemos atribuir valores a mais de uma variável em uma única instrução. Por exemplo, `a, b = 2, 3`.

o  
Ver anotações

Uma das grandes características da linguagem é sua sintaxe. Uma das principais ideias de Guido é que o código é lido com muito mais frequência do que está escrito ([PEP 8 - Style Guide for Python Code](#)). Tal aspecto é tão relevante, que um código que segue as regras do idioma python é chamado de “pythonic code”. Essas regras são definidas pelo PEP 8 (Python Enhancement Proposal) e dizem respeito a formatação, indentação, parâmetros em funções e tudo mais que possa estar relacionado à sintaxe do código.

Python tem se mostrado uma linguagem muito poderosa e está sendo amplamente adotada por profissionais na área de dados.

O interpretador Python 3 utiliza unicode por padrão, o que torna possível usar nomes de variáveis com acento e até outros caracteres especiais, porém não é uma boa prática. Tal flexibilidade mostra porque a linguagem tem sido adotada por pessoas que não são da área de exatas, pois a leitura está mais próxima da interpretação humana do que da máquina

## ONDE PROGRAMAR?

Agora que já conhecemos um pouco dessa importante linguagem e já sabemos que existe até um “código” de escrita determinado pelo PEP 8, podemos nos perguntar: mas onde eu escrevo os códigos em Python e vejo os resultados? A implementação de códigos em Python pode ser feita tanto em ferramentas instaladas no seu computador quanto em ambientes em nuvem. Nesse universo de possibilidades, o

primeiro ponto que você precisa entender é que, para obter os resultados de um código, precisamos ter um interpretador Python. Vamos começar analisando as opções que podem ser instaladas.

0

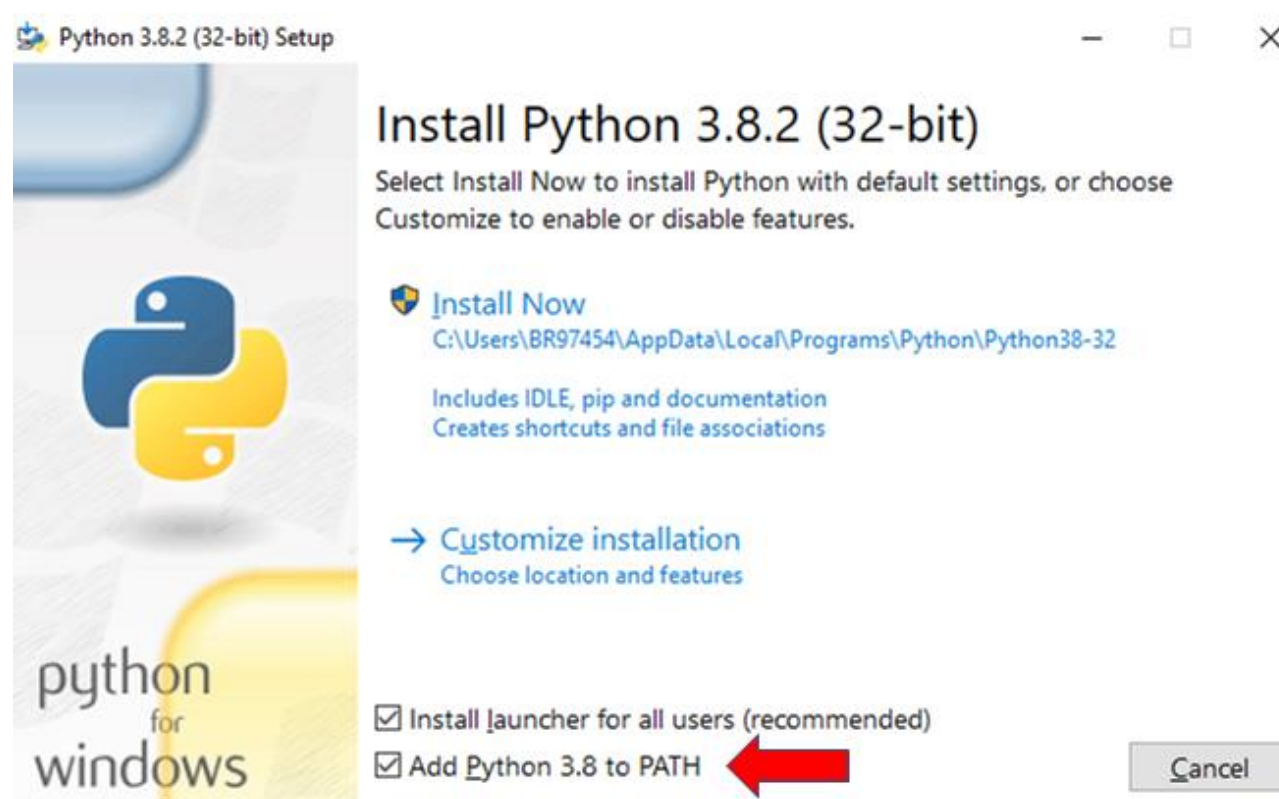
Ver anotações

## INSTALAÇÃO DO INTERPRETADOR PYTHON

Na página oficial da linguagem (<https://www.python.org/downloads/>), podemos encontrar as várias versões do interpretador Python puro (sem bibliotecas de terceiros), disponíveis para diversos sistemas operacionais. Para instalar, basta escolher o sistema operacional, fazer o download da versão mais atual do interpretador 3.X e, após concluído o download, clique no arquivo executável que foi descarregado do repositório oficial.

No processo de instalação do interpretador Python, é preciso marcar a opção Add Python 3.X to PATH (Figura 1.1). Ao marcar essa opção, a variável de ambiente Path é configurada para entender o comando “python”. Caso não seja marcada a opção, você precisará configurar manualmente depois.

Figura 1.1 | Processo de instalação do interpretador

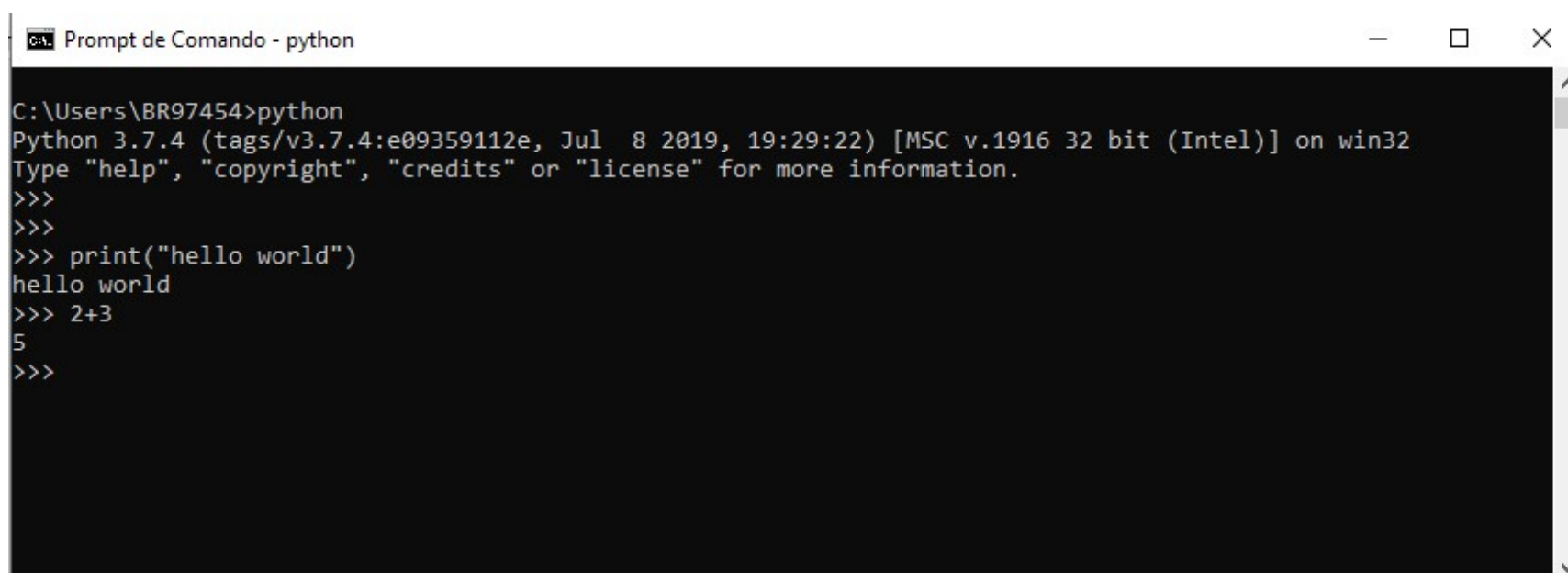


Fonte: captura de tela do instalador do Python.

Ao concluir a instalação do interpretador, se você marcou a opção “Add Python 3.X to PATH”, já podemos começar a programar através do modo iterativo. Para isso, abra o prompt de comando do Windows e digite “python” (Figura 1.2). Após aberto o modo iterativo, já podemos digitar comandos python e, ao apertar “enter”, o comando imediatamente é interpretado e o resultado é exibido.

Ver anotações

Figura 1.2 | Modo de programação iterativo



```
C:\Users\BR97454>python
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
>>>
>>> print("hello world")
hello world
>>> 2+3
5
>>>
```

Fonte: captura de tela do prompt de comando.

Caso não queira navegar até o prompt de comando, você também pode utilizar o aplicativo IDLE, já pré-instalado no Windows.

## IDE'S PYCHARM E VISUAL STUDIO CODE

Utilizar o modo iterativo é muito simples e recomendado para testes rápidos, porém essa forma de implementação não é aplicável quando precisamos criar um código que será executado mais de uma vez. Isso acontece porque no modo iterativo, ao fechar o prompt, todo o código é perdido.



Para implementação de soluções (e não somente testes rápidos), nós programadores usamos uma IDE, (Integrated Development Environment) que traduzindo significa Ambiente de Desenvolvimento Integrado. Esse tipo de software possui uma série de ferramentas que auxiliam o desenvolvimento, como integração com sistemas de versionamento de código, refatoração de código, debug, etc. No mercado, duas IDE's disputam a preferência dos desenvolvedores Python, o PyCharm (<https://www.jetbrains.com/pycharm/>) e o Visual Studio Code (VSCode) (<https://visualstudio.microsoft.com/pt-br/>).

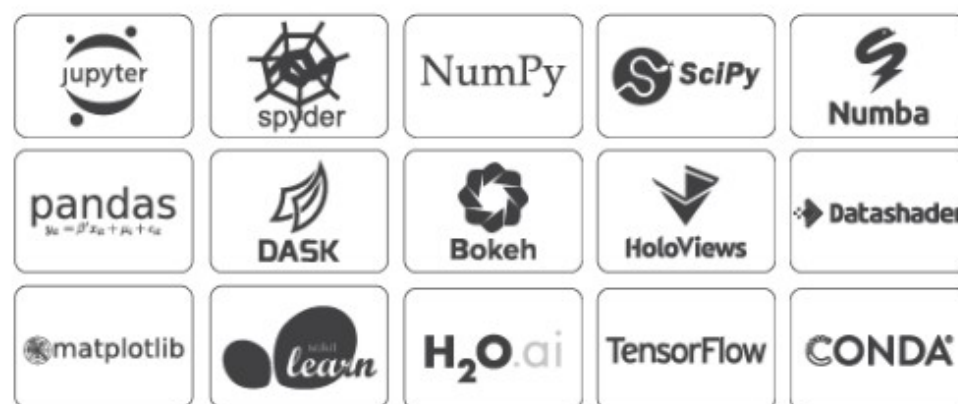
Ver anotações

O PyCharm é oferecido em duas opções: Professional e Community, sendo a primeira paga e a segunda gratuita. Portanto, caso opte por essa IDE, você deverá fazer o download da versão Community no site (<https://www.jetbrains.com/pycharm/download/>). Já o VSCode é gratuito e pode ser encontrado no site (<https://visualstudio.microsoft.com/pt-br/>).

## PROJETO PYTHON ANACONDA

Para quem está começando a aprender Python, uma opção de ferramenta de trabalho é o projeto Python Anaconda (<https://www.anaconda.com/distribution/>). O projeto Anaconda consiste na união de diversas ferramentas Python, compostas por bibliotecas e IDE's (Figura 1.3).

Figura 1.3 | Ferramentas disponíveis no projeto Anaconda



Fonte: captura de tela de Anaconda (2020).

Fazendo a instalação do projeto Anaconda, você passa a ter tanto o interpretador Python quanto várias bibliotecas, além de duas interfaces de desenvolvimento: a IDE spyder e o projeto Jupyter. A IDE spyder, é similar ao PyCharm e ao VSCode, ou seja, é um ambiente de desenvolvimento com várias funcionalidades integradas.

Ver anotações

Um dos grandes diferenciais do projeto Anaconda é ter o Jupyter Notebook (<https://jupyter.org/>) integrado na instalação.

O Jupyter Notebook é um ambiente de computação interativa, que permite aos usuários criar documentos de notebook que incluem: código ativo, gráficos, texto narrativo, equações, imagens e vídeo. Esses documentos fornecem um registro completo e independente de uma computação (um código) que pode ser convertida em vários formatos e compartilhada com outras pessoas usando email, Dropbox, sistemas de controle de versão (como git / GitHub). Uma das grandes vantagens do Jupyter Notebook é funcionar em um navegador de internet. No endereço (<https://jupyter.org/try>), você pode experimentar a ferramenta, sem precisar de instalação. Porém, instalando o projeto Python Anaconda, essa ferramenta já é instalada e você pode acessar fazendo a busca por Jupyter Notebook nos aplicativos do seu sistema operacional.

Em um Jupyter Notebook, o programador pode escrever trechos de códigos em células e, assim que executa a célula, o trecho é interpretado e o resultado aparece logo abaixo, da mesma forma como fizemos no primeiro comando `print("hello world")`. No canal do Caio Dallaqua, você encontra um vídeo (<https://www.youtube.com/watch?v=m0FbNlhNyQ8>) de 2 minutos, com uma rápida introdução ao uso dessa ferramenta.

## GOOGLE COLABORATORY (COLAB)

Colaboratory, ou "Colab", é um produto da Pesquisa do Google (<https://research.google.com/colaboratory/faq.html>). O Colab permite que qualquer pessoa escreva e execute código Python através do navegador. Além disso, é especialmente adequado para aprendizado de máquina, análise de dados e educação. Mais tecnicamente, o Colab é um serviço de notebook Jupyter hospedado que não requer configuração para ser usado, além de fornecer acesso gratuito a recursos de computação, incluindo GPUs.

Ver anotações

O Colab é baseado no projeto de código aberto Jupyter. Portanto, o Colab permite que você use e compartilhe os notebooks Jupyter com outras pessoas sem precisar baixar, instalar ou executar nada. Ao criar um Jupyter Notebook no Colab, você pode compartilhar o link com pessoas específicas, ou então, compartilhar o trabalho no GitHub.

Para começar a usar o Colab, basta acessar o endereço <https://colab.research.google.com/notebooks/> ou [https://colab.research.google.com/notebooks/intro.ipynb?utm\\_source=scs-index](https://colab.research.google.com/notebooks/intro.ipynb?utm_source=scs-index) e desfrutar das vantagens dessa ferramenta.

***Devido a facilidade e vantagens do Colab, indicamos essa ferramenta como principal meio de trabalho para essa disciplina.***

Faça o link do seu Colab com seu Google drive, copie os códigos que serão apresentados, altere-os, explore!



# MÃO NA MASSA!

Agora que já conhecemos um pouco do poder dessa linguagem de programação e já sabemos onde programar, chegou a hora de colocar a mão na massa e começarmos a codificar.

Ver anotações

## VARIÁVEIS E TIPOS BÁSICOS DE DADOS EM PYTHON

Variáveis são espaços alocados na memória RAM para guardar valores temporariamente. Em Python, esses espaços não precisam ser tipados, ou seja, a

```
In [2]: x = 10
nome = 'aluno'
nota = 8.75
 fez_inscricao = True
```

Para saber o tipo de dado que uma variável guarda, podemos imprimir seu tipo usando a função `type()` , veja como:

```
In [3]: print(type(x))
print(type(nome))
print(type(nota))
print(type(fez_inscricao))

<class 'int'>
<class 'str'>
<class 'float'>
<class 'bool'>
```

A célula de entrada 3 (In [3]) têm os comandos para imprimir na tela, os tipos das quatro variáveis que criamos anteriormente. Veja que foram impressas quatro diferentes classes de tipos de dados. A variável "x" é do tipo inteira (int). A variável "nome" é do tipo string (str). A variável "nota" é do tipo ponto flutuante (float). A variável "fez\_inscricao" é do tipo booleana (bool).

Em Python, tudo é objeto! Por isso os tipos de dados aparecem com a palavra "class", que significa classe. Teste você mesmo os códigos no emulador a seguir e aproveite para explorar outras variáveis!



Ver anotações 0

Agora que já sabemos como criar variáveis, vamos aprimorar nosso hello world. Vamos solicitar que o usuário digite um nome, para que possamos fazer uma saudação personalizada. A função `input()` faz a leitura de um valor digitado. Veja como usar:

```
n [4]: nome = input("Digite um nome: ")
      print(nome)
```

```
Digite um nome: João
João
```

Veja que dentro da função `input()`, colocamos a mensagem que irá aparecer na tela, o que o usuário digitar será capturado e guardado dentro da variável "nome". Na linha em seguida, imprimos o que o usuário digitou. Mas como podemos combinar a frase "hello world" com o nome digitado? Como vamos imprimir variáveis e textos juntos? Como podemos imprimir: "Olá João! Bem-vindo à disciplina de programação. Parabéns pelo seu primeiro hello world"?

Temos uma variedade de formas de imprimir texto e variável em Python. Vejamos algumas: podemos usar formatadores de caracteres (igual em C), podemos usar a função `format()` e podemos criar uma string formatada. Vejamos cada uma delas:

```
n [5]:# Modo 1 - usando formatadores de caracteres (igual na linguagem C) para imprimir
variável e texto
print("Olá %s, bem vindo a disciplina de programação. Parabéns pelo seu primeiro
hello world" % (nome))
```

Olá João, bem vindo a disciplina de programação. Parabéns pelo seu primeiro hello world

```
n [6]:# Modo 2 - usando a função format() para imprimir variável e texto
print("Olá {}, bem vindo a disciplina de programação. Parabéns pelo seu primeiro
hello world".format(nome))
```

Ver anotações

Olá João, bem vindo a disciplina de programação. Parabéns pelo seu primeiro hello world

```
n [7]:# Modo 3 - usando strings formatadas
print(f"Olá {nome}, bem vindo a disciplina de programação. Parabéns pelo seu
primeiro hello world")
```

Olá João, bem vindo a disciplina de programação. Parabéns pelo seu primeiro hello world

Em primeiro lugar, como vocês podem observar, usamos o hash # para criar comentários de uma linha. Lembrando que em qualquer linguagem de programação, os comentários não são nem interpretados, nem compilados.

Os três modos usados obtiveram o mesmo resultado e você poderia adotar o que achar mais conveniente. Entretanto, lembra que existe um código para sintaxe do "pythonic code"? Pois bem, a PEP 498 (<https://www.python.org/dev/peps/pep-0498/>) fala sobre a criação de strings com interpolação (mistura de variáveis com texto". Nessa PEP, o autor destaca o uso do "modo 3" como a melhor opção chamando-a de "f-strings". Portanto, em nossos códigos, iremos adotar essa sintaxe para a criação de strings com interpolação.

As strings formatadas com "f-strings" só podem ser compiladas com o interpretador Python na versão 3.6. Se tentar usar essa sintaxe em um interpretador em versões anteriores, será dado erro de sintaxe.

O uso do interpretador na versão 2 é fortemente desencorajado pela equipe Python, pois em 2020 ele será descontinuado. Para saber mais, leia a seção Update do endereço: <https://www.python.org/dev/peps/pep-0373/>

# OPERAÇÕES MATEMÁTICAS EM PYTHON

O Quadro 1.1 apresenta um resumo das operações matemáticas suportadas por Python. Com exceção das funções `abs()` e `pow()` e da notação de potência `**`, as outras operações e sintaxe são similares a diversas linguagens de programação.

Ver anotações

Quadro 1.1 | Operações matemáticas em Python

Operação	Resultado
$x + y$	soma de $x$ e $y$
$x - y$	Diferença de $x$ e $y$
$x * y$	Produto de $x$ e $y$
$x / y$	Quociente de $x$ e $y$
$x // y$	Parte inteira do quociente de $x$ e $y$
$x \% y$	Resto de $x / y$
<code>abs(x)</code>	Valor absoluto de $x$
<code>pow(x, y)</code>	$x$ elevado a $y$
$x ** y$	$x$ elevado a $y$

Ver anotações 0

Fonte: adaptada de (<https://docs.python.org/3/library/stdtypes.html>)

Com relação às operações matemáticas, seja na programação, seja na resolução analítica, o mais importante é lembrar a ordem de precedências das operações.

- 1. Primeiro resolvem-se os parênteses, do mais interno para o mais externo.
- 2. Exponenciação.
- 3. Multiplicação e divisão.
- 4. Soma e subtração.

```
n [8]:# Qual o resultado armazenado na variável operacao_1: 25 ou 17?
operacao_1 = 2 + 3 * 5

# Qual o resultado armazenado na variável operacao_2: 25 ou 17?
operacao_2 = (2 + 3) * 5

# Qual o resultado armazenado na variável operacao_3: 4 ou 1?
operacao_3 = 4 / 2 ** 2

# Qual o resultado armazenado na variável operacao_4: 1 ou 5?
operacao_4 = 13 % 3 + 4

print(f"Resultado em operacao_1 = {operacao_1}")
print(f"Resultado em operacao_2 = {operacao_2}")
print(f"Resultado em operacao_3 = {operacao_3}")
print(f"Resultado em operacao_4 = {operacao_4}")
```

0  
Ver anotações

```
Resultado em operacao_1 = 17
Resultado em operacao_2 = 25
Resultado em operacao_3 = 1.0
Resultado em operacao_4 = 5
```



Agora vamos juntar tudo o que aprendemos até o momento. Uma equação do segundo grau possui a fórmula:  $y = a*x**2 + b*x + c$ , onde  $a$ ,  $b$ ,  $c$  são constantes. O valor de  $y$  (resultado) depende do valor de  $x$ , ou seja,  $x$  é a variável independente e  $y$  a dependente. Considerando os valores  $a = 2$ ,  $b = 0.5$  e  $c = 1$ , vamos solicitar para o usuário um valor de  $x$  e retornar o valor de  $y$  correspondente ao  $x$  que ele digitou. Veja como deve ser implementado:



```
n [9]:a = 2
      b = 0.5
      c = 1
      x = input("Digite o valor de x: ")

      y = a * x ** 2 + b * x + c

      print(f"O resultado de y para x = {x} é {y}.")
```

0  
Ver anotações

Digite o valor de x: 3

-----  
-----

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

<ipython-input-9-42060e5b5536> in <module>

4 x = input("Digite o valor de x: ")

5

----> 6 y = a \* x \*\* 2 + b \* x + c

7

8 print(f"O resultado de y para x = {x} é {y}.")

**TypeError:** unsupported operand type(s) for \*\* or pow(): 'str' and 'int'

Erro? E agora? Calma, erros fazem parte da vida do desenvolvedor. Vamos ler a mensagem de erro. Primeiro o erro foi do tipo "TypeError", isso quer dizer que alguma variável não está com o tipo adequado para a situação. A mensagem nos diz que não é permitida a operação de potenciação (\*\*) entre um tipo string e inteiro. Portanto, o erro é porque estamos tentando fazer uma operação matemática entre string e um tipo numérico. Então, dentre as variáveis deve ter uma com o tipo errado. Vamos usar a função type() para verificar o tipo das variáveis usadas.

Ver anotações

```
n [10] print(type(a))
      print(type(b))
      print(type(c))
      print(type(x))
```

```
<class 'int'>
<class 'float'>
<class 'int'>
<class 'str'>
```

Olhando o resultado da entrada 10 (In [10]), vemos que o tipo da variável x é string (str), isso acontece porque ao usar a função input(), ela retorna uma string, independente do que o usuário digitou, sempre será string. Portanto, para corrigir o erro, precisamos converter o resultado da entrada em um tipo numérico. Como não sabemos se o usuário vai digitar um número inteiro ou decimal, vamos fazer a conversão usando a função float(). Veja como fica o código:

```
n [11] a = 2
      b = 0.5
      c = 1
      x = input("Digite o valor de x: ")

      x = float(x) # aqui fazemos a conversão da string para o tipo numérico

      y = a * x ** 2 + b * x + c

      print(f"O resultado de y para x = {x} é {y}.")
```

Digite o valor de x: 3  
O resultado de y para x = 3.0 é 20.5.

Aprendemos bastante até aqui, agora é hora de colocar em prática. Não deixe de

explorar os links apresentados, quanto mais você buscar informações, mais fluente ficará na linguagem de programação.

## REFERÊNCIAS E LINKS ÚTEIS

0

A ORIGEM do “Hello World”. **Ciência da Computação**, 2015. Disponível em: <https://cienciacomputacao.com.br/curiosidade/a-origem-do-hello-world/>. Acesso em: 25 maios 2020.

Ver anotações

ANACONDA Individual Edition: The World's Most Popular Python/R Data Science Platform. The World's Most Popular Python/R Data Science Platform. Disponível em: <https://www.anaconda.com/distribution/>. Acesso em: 10 abr. 2020.

DALLAQUA, C. **Curso Python 3**: aula 1 - notebook jupyter. 1 vídeo ( 2,54 min.), 2016. Disponível em: <https://www.youtube.com/watch?v=m0FbNIhNyQ8>. Acesso em: 10 abr. 2020.

VISUAL Studio Code. Disponível em: <https://visualstudio.microsoft.com/pt-br/>. Acesso em: 10 abr. 2020.

FARIA, F. A. **Lógica de Programação**: aula 01 - introdução. São José dos Campos: Unifesp, 2016. 33 slides, color. Disponível em: <https://www.ic.unicamp.br/~ffaria/lp2s2016/class01/lp-aula01.pdf>. Acesso em: 10 abr. 2020.

GOOGLE. **Colaboratory**: Frequently Asked Questions. Disponível em: <https://research.google.com/colaboratory/faq.html>. Acesso em: 10 abr. 2020.

PYCHARM: The Python IDE for Professional Developers. Disponível em: <https://www.jetbrains.com/pycharm/>. Acesso em: 10 abr. 2020.

PYTHON Beginners Guide. Última atualização em 19 set. 2019, por Chris M. Disponível em: <https://wiki.python.org/moin/BeginnersGuide/Overview>. Acesso em: 10 abr. 2020.

PYTHON SOFTWARE FOUNDATION (org.). **History of the software**. Disponível em: <https://docs.python.org/3.0/license.html>. Acesso em: 10 abr. 2020.

PYTHON SOFTWARE FOUNDATION. **Numeric Types**: int, float, complex. Disponível em: <https://docs.python.org/3/library/stdtypes.html>. Acesso em: 10 abr. 2020.

PYTHON SOFTWARE FOUNDATION. **PEP 373**: Python 2.7 Release Schedule. Disponível em: <https://www.python.org/dev/peps/pep-0373/>. Acesso em: 10 abr. 2020.

ROSSUM, G. V.; WARSAW, B.; COGHLAN, N. **PEP 8**: Style Guide for Python Code. Disponível em: <https://www.python.org/dev/peps/pep-0008/#introduction>. Acesso em: 10 abr. 2020.

THE JUPYTER Notebook. Disponível em: <https://jupyter.org/>. Acesso em: 10 abr. 2020.

VISUAL Studio Code. Disponível em: <https://visualstudio.microsoft.com/pt-br/>. Acesso em: 10 abr. 2020.