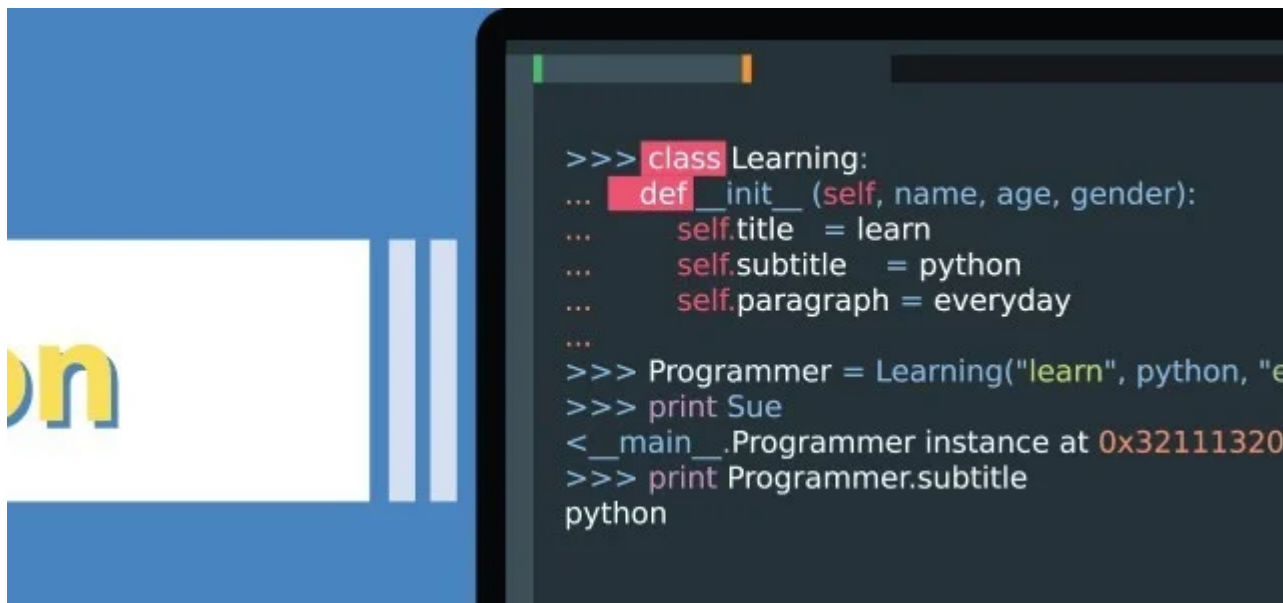


[MATRICULE-SE](#)[PROGRAMAÇÃO _](#)[FRONT-END _](#)[DATA SCIENCE _](#)[INTELIGÊNCIA ARTIFICIAL _](#)[DEVOPS _](#)[UX & DESIGN _](#)[MOBILE _](#)[INOVAÇÃO & GESTÃO _](#)[Artigos > Programação](#)

O que é Python? História, Sintaxe e um Guia para iniciar na Linguagem

**Caroline Carvalho**

Atualizado em 18 de Setembro

[COMPARTILHE](#)

Introdução

Python se tornou uma das **linguagens de programação** mais **populares do mundo** nos últimos anos. Sua versatilidade permite que ele seja utilizado em tudo, desde o aprendizado de máquinas até a construção de sites, automação de tarefas e testes de software.

Além disso, sua **proximidade** com a linguagem **humana** faz com que ele possa ser usado tanto por pessoas que desenvolvem, quanto por quem não está necessariamente inserido neste mercado.

Confira neste artigo:

- [Introdução](#)
- [O que é Python?](#)
- [A História: como tudo começou?](#)
- [Por que o Python é valorizado? Veja por que aprender a programar em Python](#)
- [Primeiros passos e recursos básicos](#)
- [Quais são os tipos de dados em Python?](#)
- [Como mudar o tipo da variável no Python?](#)
- [Estruturas condicionais em Python: if, elif e else](#)
- [Estruturas de repetição Python: for e while](#)
- [Boas Práticas de programação](#)
- [Módulos e bibliotecas em Python](#)
- [O PythonPath](#)
- [PyPI](#)
- [Ambientes virtuais](#)
- [Quais são as Bibliotecas e Frameworks mais utilizados?](#)
- [Empresas que utilizam Python](#)
- [Os melhores livros de tecnologia para ler e se aprofundar](#)
- [Aprenda mais sobre Python gratuitamente](#)
- [Apostilas da Alura — Você profissional em T](#)
- [Conclusão](#)

Nesse artigo vamos apresentar **questões importantes** a respeito da linguagem Python, como o que ele é, como começou, alguns recursos básicos para utilizá-lo, quais as IDEs mais populares entre quem utiliza Python, tipos de dados, bibliotecas e frameworks mais usados, enfim, uma abordagem bem completa. Vamos lá?



Matricule-se na escola de PROGRAMAÇÃO

Junte-se a uma comunidade de **+500 mil** estudantes

- Acesso a **TODOS** os cursos em uma única assinatura
- Novos lançamentos a cada semana
- Desafios práticos

SAIBA MAIS

O que é Python?

Trata-se de uma linguagem de programação de uso geral, o que significa que pode ser usada para criar uma grande **variedade de aplicações** diferentes e não é especializada em nenhum problema determinado.

Essa **versatilidade**, juntamente com sua **facilidade de uso** para iniciantes, tornou-a uma das linguagens mais usadas atualmente.

//

O que é Python? #HipstersPontoTube





Alguns dos **principais pontos** que trazem destaque para ela são:

- O fato de ser uma **linguagem interpretada**, o que significa que ela não precisa passar pelo processo de [compilação](#). No caso do Python, o processo de interpretação é executado dentro de **máquinas virtuais**, nas quais o código passa por uma camada intermediária que irá traduzir os comandos de programa para [código binário](#). Isso acelera bastante a velocidade de desenvolvimento;
- Sua **sintaxe é simples**, fácil de aprender e muito próxima da linguagem falada por nós, por isso podemos dizer que ela se trata de uma **linguagem de alto nível**;
- **Multiparadigma**, pois nos dá a possibilidade de programar em vários paradigmas, tais como:
 - **procedural**, com instruções passadas ao computador na sequência em que devem ser executadas;
 - [funcional](#), paradigma que consiste em programas construídos aplicando e compondo funções;
 - [orientação a objetos](#), o que traz a perspectiva do mundo real para a programação, tornando os programas fáceis de entender por causa desta relação.
- O próprio programa “reconhece” qual **tipo de dado** está sendo utilizado, fazendo com que ele não precise ser previamente declarado. Por isso dizemos que ele possui **semântica dinâmica**.

A História: como tudo começou?

Sobre a [origem do Python](#), ele foi desenvolvido inicialmente no final da **década de 1980** pelo programador holandês Guido van Rossum, e teve sua **primeira versão lançada em 1991** no Centrum Wiskunde & Informatica - CWI (Instituto Nacional de Pesquisa para Matemática e Ciência da Computação), na Holanda.

A linguagem fez grandes avanços para o desenvolvimento de [código aberto](#), utilizando as [PEPs](#) ("Propostas de Enriquecimento do Python") como principal ferramenta para sugestões de melhoramento e discussões da comunidade.

Elas são usadas para descrever mudanças na linguagem ou em suas normas, e são avaliadas pelo público e aceitas ou rejeitadas depois de muitas discussões. Qualquer um pode escrever e enviar uma PEP para avaliação.

//

[Python – Hipsters Ponto Tech #122](#)



Ouvir um pouco de:
Python – Hipsters #122

Por que o Python é valorizado? Veja por que aprender a programar em Python

Utilizando como base a pesquisa "[Python Developers Survey 2021 Results](#)", [realizada pela JetBrains](#), o Python é umas das linguagens que crescem mais rapidamente em relevância e tem sido valorizado pelas empresas que o utilizam por conta das suas características, como **agilidade, praticidade e versatilidade**, sobretudo para projetos que demandam maior complexidade.

Python Fluente – Hipsters Ponto Tech #179

Ouvir um pouco de:

Python Fluente – Hipsters Ponto Tech #179

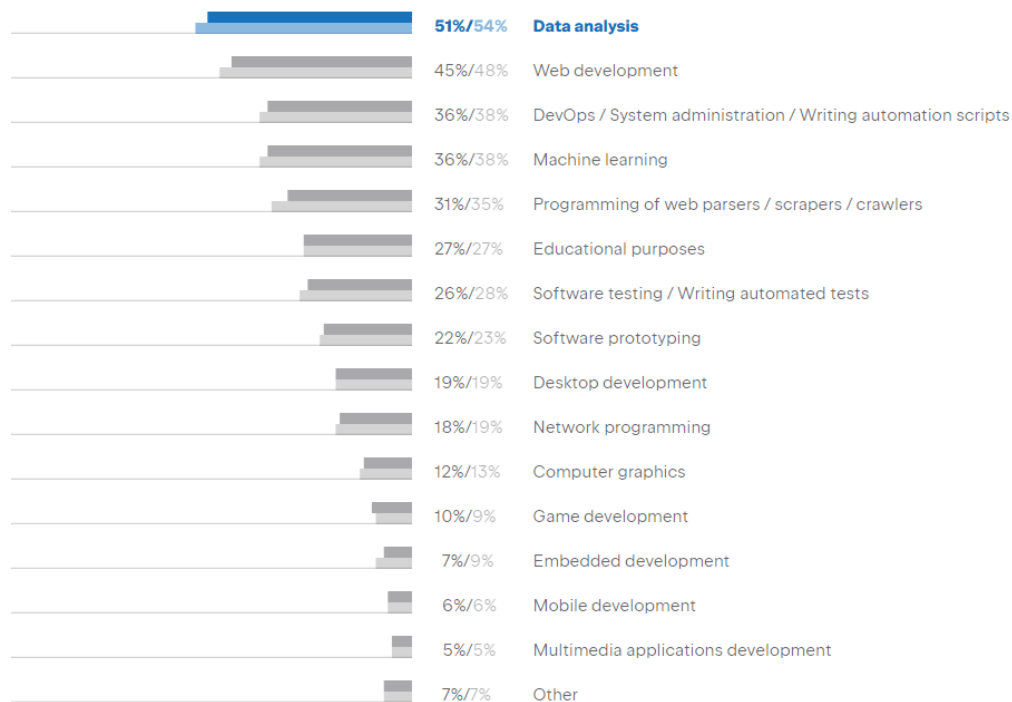
Nas **áreas em que ele é mais utilizado** temos como destaque:

1. A análise de dados;
2. Aprendizado de máquina;
3. Desenvolvimento web; e
4. DevOps.

Uso do Python em 2020 e 2021

100+

- 2021
- 2020



Créditos: Python Developers Survey 2021 Results

7 Motivos para aprender a programar em Python

Podemos apontar como os **principais motivos** para aprender Python:

1. Sintaxe simples;
2. É Multiplataforma e de código aberto;
3. Versatilidade;
4. Comunidade fiel e ativa;
5. Tem sido utilizada por grandes empresas;
6. É a mais popular na Ciência de Dados;
7. E por todos esses motivos está em alta no mercado de trabalho.

Saiba mais: qual a melhor linguagem de programação?

//

*A MELHOR linguagem de programação com Fabio Akita /
#HipstersPontoTube*



Primeiros passos e recursos básicos

//

Primeiros Passos em Data Science: Do Excel e BI ao Python – Hipsters Ponto Tech #134



Ouvir um pouco de:

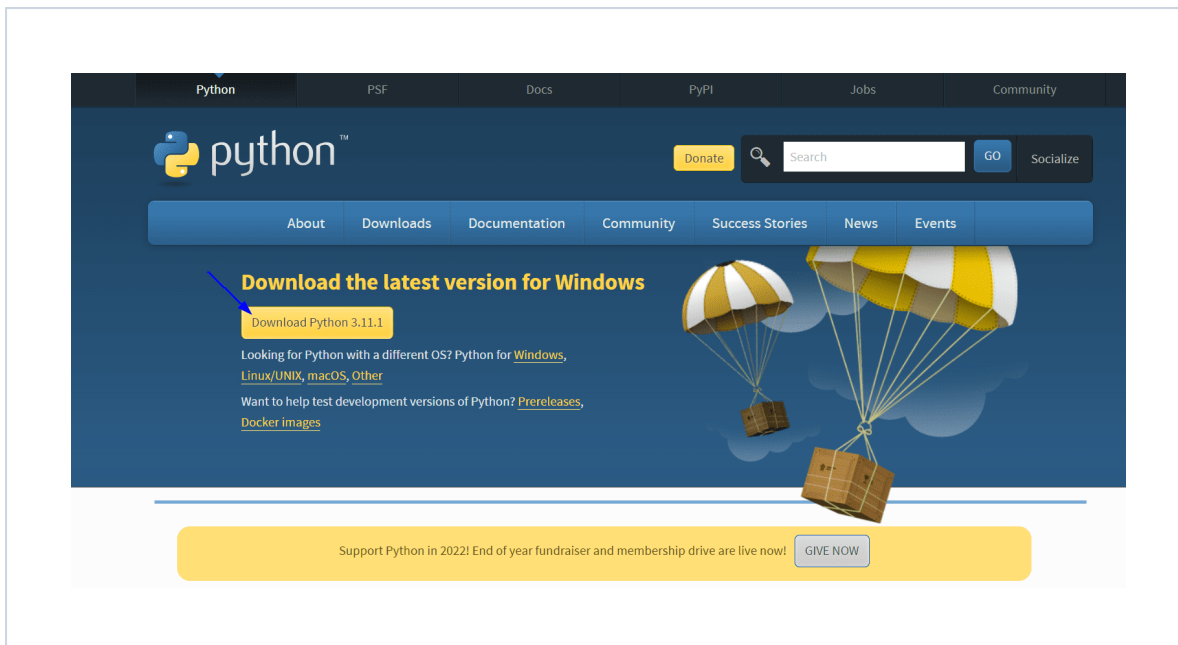
Primeiros Passos em Data Science: Do Excel e BI ao Python – Hipsters #134

Veja mais para seus primeiros passos:

1. Como instalar o Python no Windows;
2. Como instalar o Python no Linux; e
3. IDEs para desenvolvimento.

01) Como instalar o Python no Windows

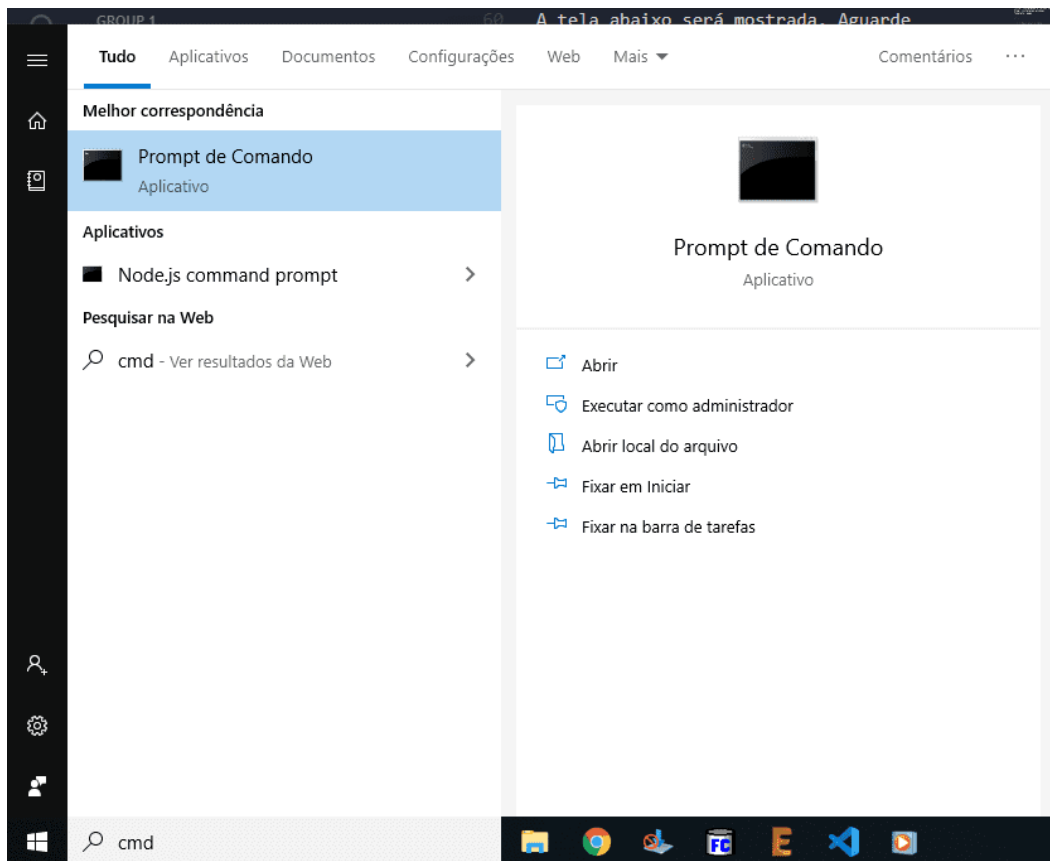
Para instalar o Python no seu sistema operacional **Windows**, você deve baixar o instalador disponível na [página de download oficial do Python](#) e clicar em *download*, como mostrado abaixo.



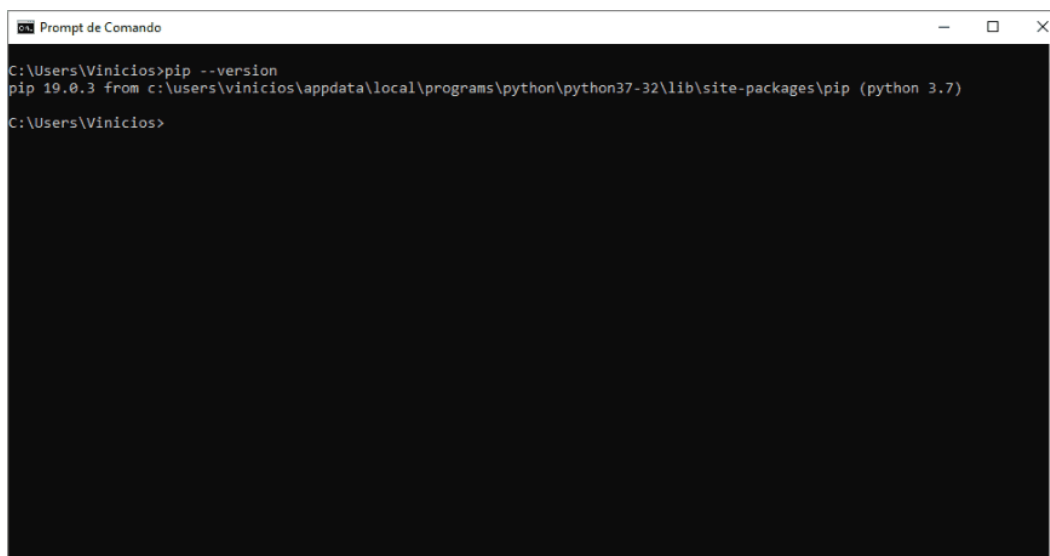
Faça o download do **instalador executável** do Windows e **clique duas vezes** nele para iniciar o assistente de instalação.

O **processo** de instalação é bem simples:

1. **Marque** a opção "Add Python to PATH";
2. **Clique** em "Install Now";
3. Após a instalação, basta **clique** no botão "Close";
4. Para verificar se a instalação do Python foi bem-sucedida, **pesquise** no **Menu Iniciar** por "cmd" e **clique** duas vezes para abri-lo;

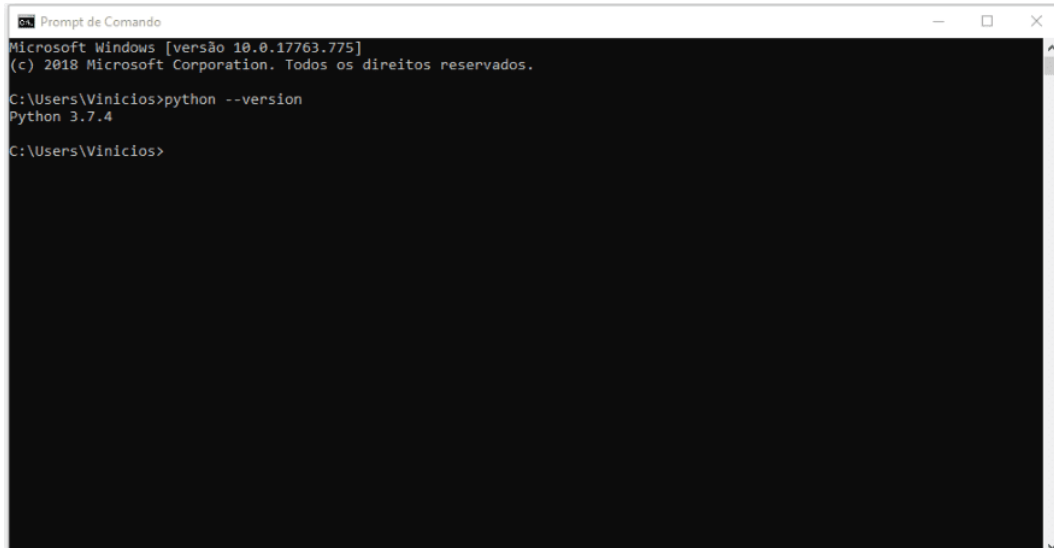


5. **Digite** o comando `python --version` para se assegurar da versão do Python que está instalada.



6. Agora **digite** `pip --version`. Este comando retornará a versão do pip instalada em sua máquina. O pip é o gerenciador de pacotes. Com

ele, você poderá adicionar novas funcionalidades ao Python.



```
Prompt de Comando
Microsoft Windows [versão 10.0.17763.775]
(c) 2018 Microsoft Corporation. Todos os direitos reservados.

C:\Users\Vinicios>python --version
Python 3.7.4

C:\Users\Vinicios>
```

02) Como instalar o Python no Linux

1. Os sistemas operacionais baseados no **Debian** já possuem o Python3 pré-instalado. Verifique se esse é o caso do seu sistema executando o seguinte comando no terminal:

```
python3 -V
```

//

É importante que a letra V esteja maiúscula!

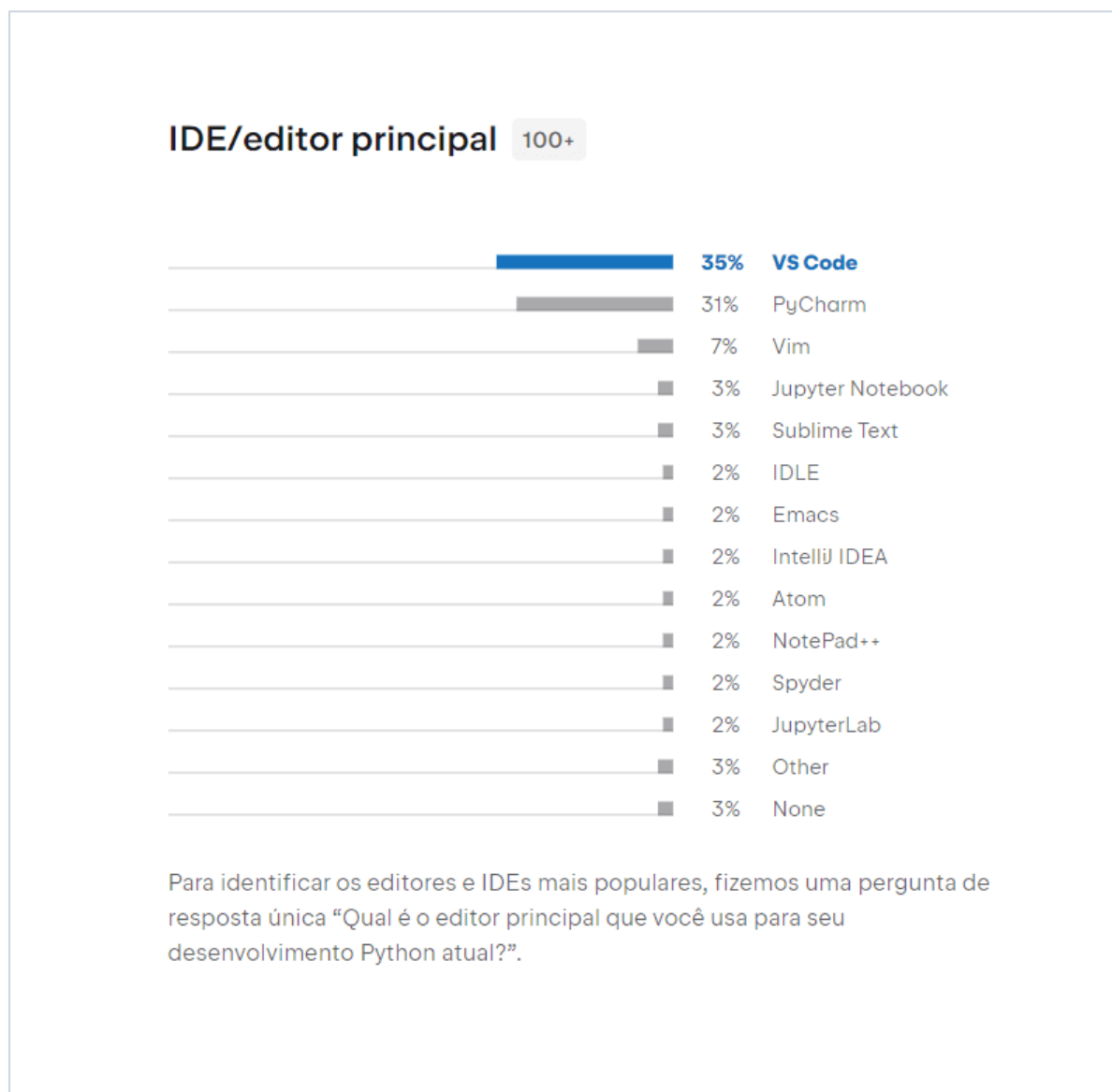
Assim será retornada a **versão** do Python3 instalada.

2. Se você ainda não tiver ele instalado, **digite** os seguintes **comandos no terminal**:

```
sudo apt-get update  
sudo apt-get install python3
```

03) IDEs para desenvolvimento

Ainda segundo a pesquisa “**Python Developers Survey 2021 Results**” esses são os [ambientes de desenvolvimento](#) mais utilizados para desenvolver em Python:

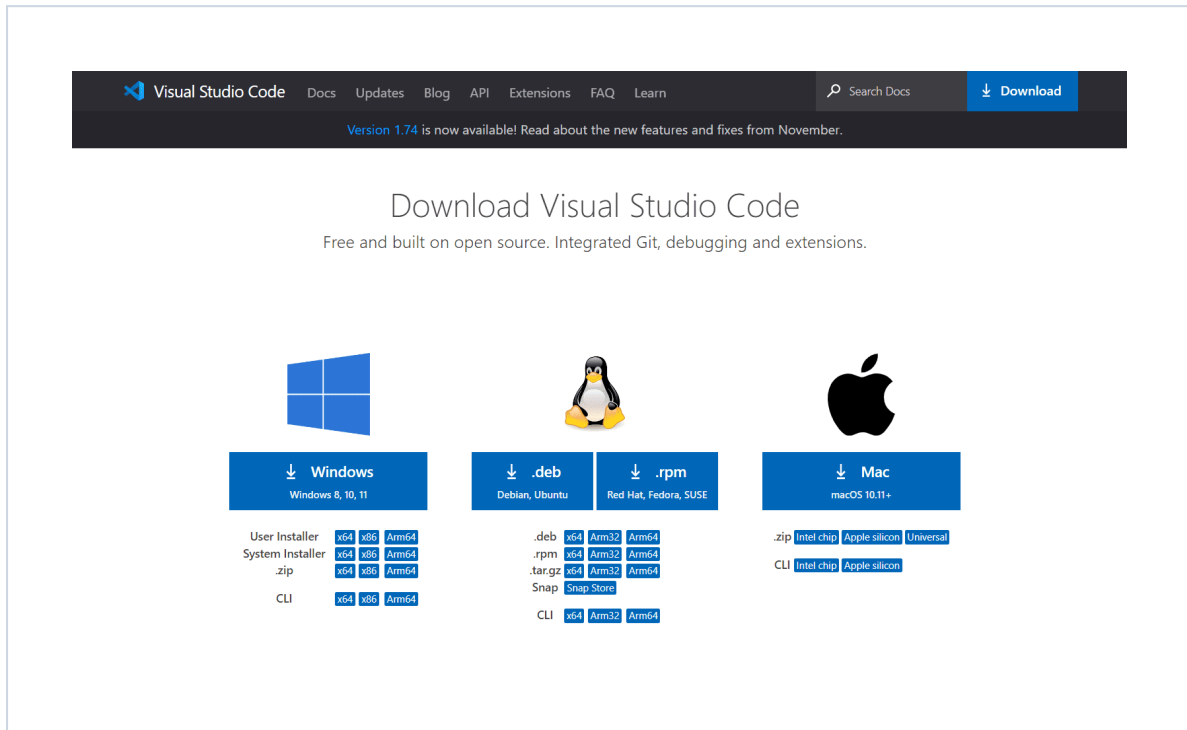


Créditos: *Python Developers Survey 2021 Results*

Fica bem evidente que as opções mais utilizadas no mercado por pessoas que trabalham com desenvolvimento web são o **Visual Studio Code** e **Pycharm**.

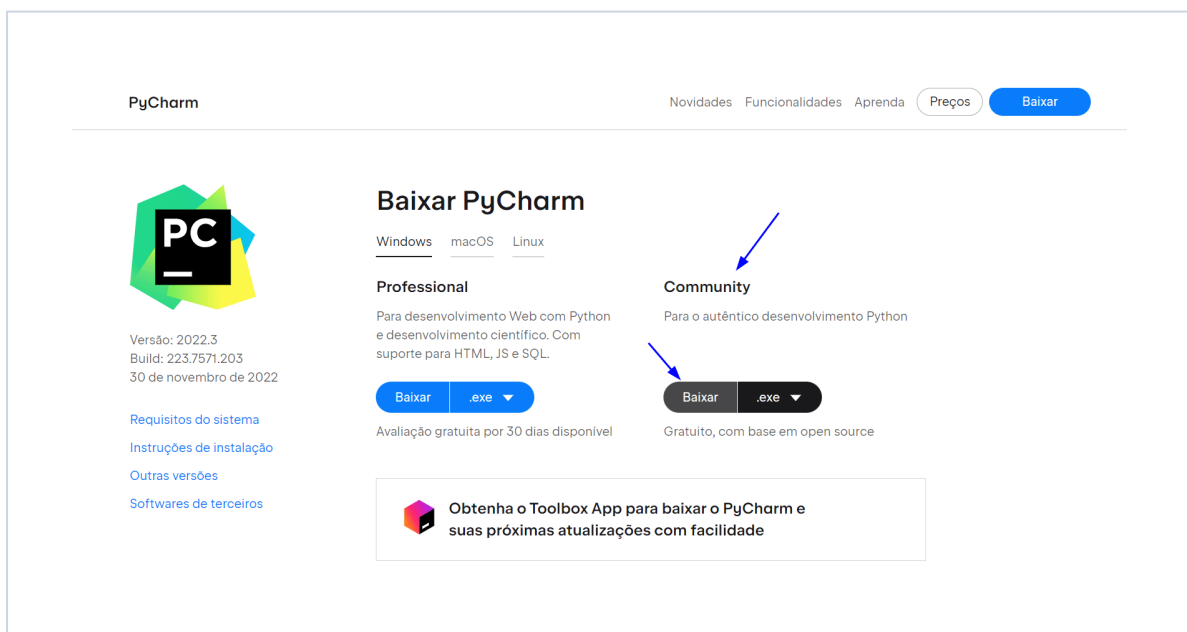
3.1) VS Code

Então, que tal **instalar o VS Code**? Para isso basta fazer o [download do pacote de instalação disponível no site oficial](#), que será semelhante à imagem a seguir:



3.2) PyCharm

Para instalar a outra IDE, nós também utilizamos o pacote de instalação disponível no [site oficial do PyCharm](#). Mantenha a atenção para realizar o download da versão **Community**, pois ela está disponível gratuitamente:



Quais são os tipos de dados em Python?

Relembrando, dissemos anteriormente que Python é uma linguagem **dinamicamente tipada**, o que significa que não é necessário declarar o tipo de variável ou fazer casting (mudar o tipo de variável), pois o Interpretador se encarrega disso para nós!

Isso significa também que, se por algum motivo precisarmos **alterar o tipo de variável durante a execução do programa**, é possível fazer essa mudança.

Os **tipos de dados padrão** do Python são:

1. Inteiro (int);
 - Exemplo: 1
2. Ponto Flutuante ou Decimal (float);
 - Exemplo: 1.1
3. Tipo Complexo (complex);
 - Exemplo: 8j
4. String (str);
 - Exemplo: hello
5. Boolean (bool);
 - Exemplo: true / false
6. List (list);
 - Exemplo: ['Mônica', 'Ana', 'Bruno', 'Alice']
7. Tuple;
 - Exemplo: (90, 79, 54, 32, 21)

8. Dictionary (dic);

- Exemplo: {'Camila': 1.65, 'Larissa': 1.60, 'Guilherme': 1.70}

01) Tipo Inteiro (int)

É um tipo usado para um **número** que pode ser escrito **sem um componente decimal**, podendo ser **positivo ou negativo**. No código abaixo, por exemplo, vemos as variáveis, `idade` e `ano`, com os valores `20` e `2010` atribuídos a elas, se pedirmos que o programa imprima o tipo das variáveis, vamos ter como retorno que elas são do tipo inteiro:

```
idade = 20
ano = 2010

print(type(idade))
print(type(ano))

<class 'int'>
<class 'int'>
```

02) Ponto Flutuante ou Decimal (float)

É um tipo composto por **números decimais**. O **float** é usado para números racionais (que podem ser representados por uma fração), informalmente conhecidos como “números quebrados”.

No código abaixo, por exemplo, vemos as variáveis `altura` e `peso` com os valores `1.73` e `78.500` atribuídos a elas. Se pedirmos que o programa imprima o tipo das variáveis, vamos ter como retorno que elas são do tipo float:


```
altura = 1.73
peso = 78.500

print(type(peso))
print(type(altura))

<class 'float'>
<class 'float'>
```

03) Complexo (complex)

Tipo de dado usado para representar **números complexos**, normalmente em **cálculos geométricos e científicos**.

Um tipo complexo contém **duas partes**: a parte **real** e a parte **imaginária**, sendo que a imaginária contém um `j` no sufixo.

Com a função `complex()` podemos **converter** reais em números complexos. Ela traz dois argumentos, sendo o primeiro deles um número real, correspondente à parte real do número complexo, e o outro, um argumento opcional, que representa a parte imaginária. Por exemplo: `z = complex(x,y)`.

No exemplo abaixo, vemos as variáveis, `a` e `b`, com os valores `10+16j` e `3+80j` atribuídos a elas. Se pedirmos que o programa imprima o tipo das variáveis, vamos ter como retorno que elas são do tipo `complex`:

```
a = 10+16j
b = 3+80j

print(type(a))
print(type(b))

<class 'complex'>
<class 'complex'>
```

04) String (str)

É um **conjunto de caracteres** geralmente utilizados para representar palavras, frases ou textos.

Temos como exemplo as variáveis `nome` e `profissao`, com os dados `Guilherme` e `Engenheiro de Software` atribuídos a elas. Pedindo para o programa imprimir o tipo dessas variáveis, teremos como retorno que elas são strings:

```
nome = 'Guilherme'
profissao = 'Engenheiro de Software'

print(type(profissao))
print(type(nome))

<class 'str'>
<class 'str'>
```

05) Boolean (bool)

Tipo de dado **lógico** que pode representar apenas **dois** valores: **falso ou verdadeiro** (False ou True, em Python).

Na lógica computacional, podem ser considerados como **0** ou **1**.

Como exemplo, temos as variáveis: `sexta_feira` e `feriado`, com os dados `True` e `False` atribuídos a elas. Se pedirmos que o programa imprima o tipo das variáveis, vamos ter como retorno que elas são do tipo boolean:

```
sexta_feira = True
feriado = False

print(type(sexta_feira))
print(type(feriado))
```

```
<class 'bool'>
<class 'bool'>
```

06) Listas (list)

As [listas](#) agrupam um **conjunto de elementos variados**, podendo conter: inteiros, floats, strings, outras listas e outros tipos.

Elas são definidas utilizando-se **colchetes** para delimitar a lista e **vírgulas** para separar os elementos. Já existe um [artigo sobre listas em Python: operações básicas](#) na plataforma, caso você queira se aprofundar no tema.

No código abaixo, por exemplo, vemos as variáveis `alunos` e `notas`, com os dados: `'Mônica', 'Ana', 'Bruno', 'Alice'` e `10, 8.5, 7.8, 8.0` atribuídos a elas. Pedindo para o programa imprimir o tipo das variáveis, vamos ter como retorno que elas são do tipo lista:

```
alunos = ['Mônica', 'Ana', 'Bruno', 'Alice']
notas = [10, 8.5, 7.8, 8.0]

print(type(alunos))
print(type(notas))

<class 'list'>
<class 'list'>
```

07) Tuplas (tuple)

Assim como as listas, a [tupla](#) é um tipo que **agrupa um conjunto de elementos**. Porém, sua forma de definição é diferente, já que utilizamos **parênteses** e as informações são separadas por **vírgula**.

Mas a **principal diferença** das listas é que as **tuplas são imutáveis**, ou seja, após sua definição, não podem ser modificadas. Para saber mais a respeito desse tipo, [confira o artigo "Conhecendo as tuplas no Python"](#).

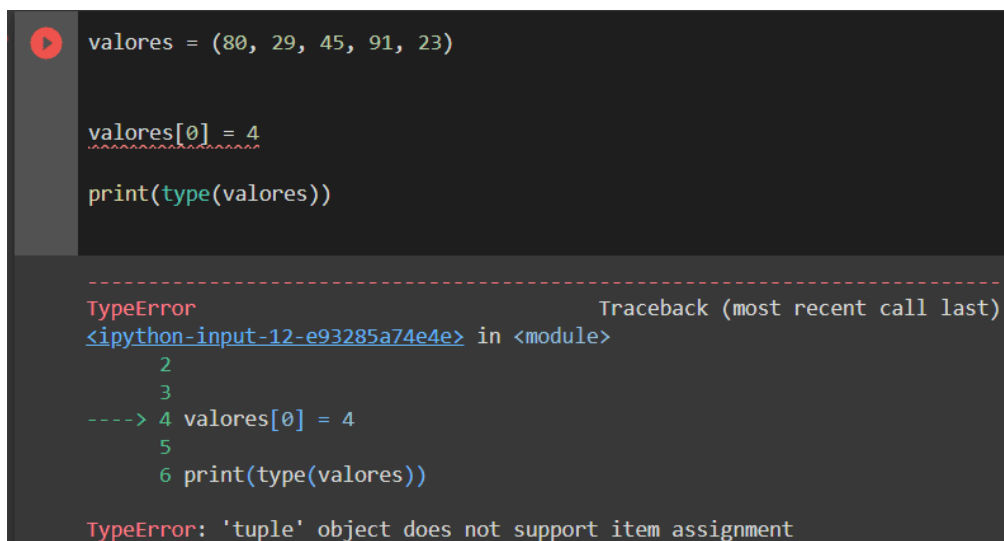
No exemplo abaixo, há as variáveis `valores` e `pontos`, com conjuntos de dados atribuídos a elas. Pedindo para o programa imprimir o tipo das variáveis, temos como retorno que elas são do tipo `tuple`.

```
valores = (80, 29, 45, 91, 23)
pontos = (10, 29.05, 66.8, 72)

print(type(valores))
print(type(pontos))

<class 'tuple'>
<class 'tuple'>
```

Se **tentarmos modificar** uma tupla, receberemos a seguinte **mensagem de erro**:



```
valores = (80, 29, 45, 91, 23)

valores[0] = 4
print(type(valores))

-----
TypeError                                Traceback (most recent call last)
<ipython-input-12-e93285a74e4e> in <module>
      2
      3
----> 4 valores[0] = 4
      5
      6 print(type(valores))

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

08) Dicionários (dict)

Os dicionários são normalmente utilizados para **agrupar elementos através da estrutura de chave e valor**, de forma que a chave é o primeiro elemento, seguido por dois pontos e pelo valor.

Vemos abaixo um exemplo de seu uso, por meio de variáveis de altura e idade, com dados de pessoas como primeiro elemento e os respectivos dados referentes às alturas e idades como valores.

```
altura={'Sandra':1.65, 'Elizabeth': 1.60, 'Roberto': 1.70}
idade = {'Sandra':35, 'Elizabeth':58, 'Roberto':68}"

print(type(altura)
print(type(idade))

<class 'dict'>
<class 'dict'>
```

Como mudar o tipo da variável no Python?

Em alguns cenários pode ser necessário mudar o tipo de uma variável e no Python isso é muito fácil, justamente por se tratar de uma linguagem dinamicamente tipada.

Vamos ver alguns exemplos de como trocar os tipos de variáveis.

Float para String

```
#Antes de converter
altura=1.55
print(type(altura)

#Fazendo a conversão
altura = str(altura)"

#Depois da conversão
```

```
print(type(altura))
print(altura)

<class 'float'>
<class 'str'>
1.55
```

Inteiro para Float

```
#Antes de converter
idade=18
print(type(idade))

#Fazendo a conversão
idade = float(idade)

#Depois da conversão
print(type(idade))
print(idade)

<class 'intt'>
<class 'float'>
18.0
```

Estruturas condicionais em Python: if, elif e else

Estruturas condicionais são artifícios utilizados para **controlar o fluxo de execução de programas**, assim podemos determinar qual **bloco de código** será executado a partir de uma determinada condição. Na linguagem Python, podemos fazer isso utilizando as estruturas `if`, `elif` e `else`, como veremos a seguir.

If

Utilizamos o **if** em um programa quando pretendemos **verificar se uma ação é verdadeira** e executar o bloco de código contido em seu escopo. Fazemos isso da seguinte forma:

```
media = 7
if media > 6.9:
    print("Você foi aprovado")
```

Neste exemplo, nosso programa irá executar e retornar a mensagem: "Você foi aprovado", apenas se a variável `media` possuir um valor maior que `6.9`. Caso o valor seja menor do que isso, ele simplesmente irá ignorar o comando "print" e finalizará logo em seguida.

Else

Se usarmos o **if/else em conjunto**, faremos com que **uma das ações presentes no código sejam executadas**, pois se determinada condição (`if`) for verdadeira, o programa executará uma ação. Caso não seja verdadeira, executaremos o `else`, ou seja, outra ação. Desse modo, testamos se uma condição é verdadeira.

Neste exemplo abaixo, nosso programa irá executar e retornar a mensagem: "Você foi reprovado", apenas se a variável `media` possuir um valor menor que `4.9`. Caso contrário, será printado: "Você foi aprovado".

```
media = 5
if media < 4.9:
    print("Você foi reprovado")
else:
    print("Você foi aprovado")
```


If...elif...else

Utilizamos **if/elif/else** quando precisamos verificar mais de uma condição. Imagine que precisamos verificar se um aluno foi reprovado ou se ele ficou de recuperação. Teríamos algo parecido com o seguinte cenário:

```
media = 6
if media < 5
    print("Você foi reprovado")
elif media > 5
    media < 7
    print("Você fará a recuperação")
else:
    print("Você foi aprovado")
```

Primeiro, precisamos verificar se a média do aluno é menor que 5. Em caso positivo, vamos imprimir a mensagem: "Você foi reprovado". Caso essa condição seja falsa, precisamos verificar se ele foi aprovado ou se fará a recuperação. Para isso, o `elif` irá testar se a média está entre os valores 5 e 7. Se sim, vamos imprimir a mensagem: "Você fará a recuperação". Se a condição do `if` e do `elif` forem falsas, o código contido no `else` será executado e imprimirá a mensagem: "Você foi aprovado".

Estruturas de repetição

Python: for e while

As estruturas de repetição são utilizadas quando queremos que um **bloco de código seja executado várias vezes**. Em Python podemos fazer isso de duas formas: utilizando `for` ou `while`.

Como usar For no Python?

Utilizamos o `for` quando queremos **iterar** sobre um bloco de código por um **determinado número de vezes**.

```
for i in range(1,10):  
    print(i)
```

Saída:

```
1  
2  
3  
4  
5  
6  
7  
8  
9
```

Como usar o While no Python?

A outra forma de repetir a execução de um trecho de código até que uma condição seja satisfeita é utilizar o `while`. Ou seja, é necessário que uma **expressão booleana dada seja verdadeira**. A partir do momento que ela se tornar falsa, o `while` para.

```
gastos=0  
valor_gasto=0  
while gastos < 1000  
    valor_gasto = int(input("Digite o valor gasto"))  
    gastos = gastos + valor_gasto  
print(gastos)
```

```
Digite o valor do novo gasto 20  
Digite o valor do novo gasto 500
```

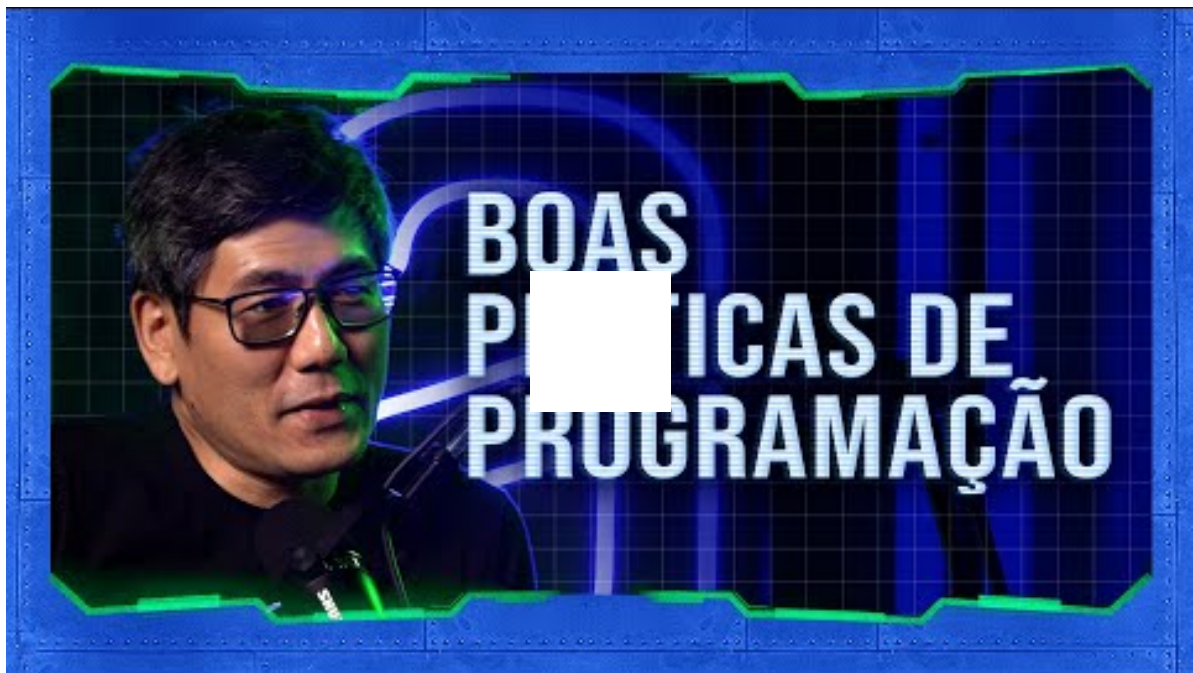
```
Digite o valor do novo gasto 480  
1000
```

Boas Práticas de programação

A tecnologia continua a evoluir e sempre surgem novas formas de escrever um **código**. Por isso, **aplicar boas práticas de programação é fundamental** para aprimorar o ambiente de trabalho, ter qualidade no momento de programar, manter o código legível, organizado e funcional.

//

Como desenvolver boas práticas de programação? com Fabio Akita / #HipstersPontoTube



Módulos e bibliotecas em Python

Você deve já ter visto menções aos pacotes, módulos e bibliotecas de Python. Mas afinal, o que são eles e como funcionam? Vamos entender melhor o comportamento e a utilidade de cada um a seguir.

Vejamos uma definição de módulo de acordo com a documentação da linguagem:

//

*“Um **módulo** é um **arquivo Python** contendo **definições e instruções**.”*

O nome do arquivo é o módulo com o sufixo `.py` adicionado. Dentro de um módulo, o nome do módulo (como uma string) está disponível na variável `global __name__`.

O **conjunto desses módulos** (arquivos) pode ser chamado de **pacote ou biblioteca**. Já os pacotes, ainda segundo a documentação:

//

“São uma maneira de estruturar namespaces para módulos Python utilizando a sintaxe de ‘nomes pontuados’ (dotted names).”

Um **namespace** é uma **coleção de nomes simbólicos**, atualmente definidos junto com informações sobre o objeto ao qual cada nome faz referência. Você pode pensar em um namespace como um dicionário no qual as chaves são os nomes dos objetos e os valores são os próprios objetos. Cada par chave-valor mapeia um nome para seu objeto correspondente.

Para ilustrar melhor, vamos observar o seguinte exemplo:

- Criaremos um **módulo que representa um círculo**. Para isso, geramos um arquivo chamado `‘circulo.py’` com o seguinte conteúdo:

```
Pi=3.14159

def area(raio):
    return Pi*(raio **2)

def comprimento_circunferencia(raio):
    return 2*Pi*raio: return 2*Pi*raio”]()
```

- Para usar o módulo acima, podemos **importar** seus dados para outro arquivo da seguinte forma:

```
import circulo

print(circulo.Pi)
print(circulo.area(5))
print(circulo.comprimento_circunferencia(5))
```

- Dessa forma, teremos o retorno:

```
3.14159
78.53975
31.4159
```

Caso quiséssemos utilizar apenas a função `area()` poderíamos importar apenas ela por meio do comando `from circulo import area`. Isso é útil em caso de módulos muito grandes e quando queremos usar apenas uma ou outra função deles.

Com base nisso, precisamos entender que a biblioteca padrão Python é um conjunto de módulos disponíveis para que você possa importá-los e usar as funcionalidades deles quando quiser.

O PythonPath

Até agora, tratamos apenas de casos em que os módulos importados estavam no **mesmo nível de diretório** daqueles que os importavam. Em **outros casos**, precisamos entender **como o interpretador Python busca por módulos** em outros [diretórios](#).

Quando uma instrução “**import**” é executada, o interpretador primeiramente irá verificar se o módulo requerido está no diretório atual. Se estiver, vai

importá-lo como vimos no exemplo. Caso contrário, a busca se estenderá ao **PYTHONPATH**.

O PYTHONPATH é uma **lista de diretórios** na qual o interpretador Python irá buscar por módulos para importação.

Um dos módulos mais utilizados é o **datetime**. Você pode aprender mais sobre o módulo datetime no artigo [Python datetime: Como faço para definir data e hora em Python?](#)

PyPI

O Python Package Index, abreviado como PyPI é o **repositório de software** oficial de terceiros para Python.

De acordo com a documentação oficial:

//

“PyPI é o Índice de Pacotes padrão para a comunidade Python. Está aberto a todos os desenvolvedores Python para consumir e distribuir suas distribuições.”

Alguns gerenciadores de pacotes, incluindo o `pip`, usam o PyPI como a fonte padrão para os pacotes e suas dependências, e mais de 113.000 pacotes Python podem ser acessados por meio do PyPI.

O PyPI, primariamente, hospeda pacotes Python na forma de **arquivos pré-compilados** e funciona como um **índice**. Assim, ele permite que os usuários pesquisem e criem pacotes por meio de palavras-chave ou por filtros que ficam disponíveis com licença de software livre.

Para saber mais sobre o PyPI [você pode consultar o artigo “Como publicar seu código Python no PyPI”](#).

Ambientes virtuais

Normalmente, aplicações em Python usam pacotes e módulos que não vêm como parte da instalação padrão. Algumas aplicações podem precisar de uma **versão específica de uma biblioteca**, porque ela requer que algum problema em particular tenha sido consertado, ou foi escrita utilizando-se de uma versão obsoleta da interface da biblioteca.

Isso significa que talvez não seja possível que uma instalação Python preencha todos os requisitos necessários. Se uma aplicação “A” necessita da versão 1.0 de um módulo particular, mas a aplicação “B” necessita da versão 2.0, os **requisitos entrarão em conflito** e instalar qualquer uma das duas versões fará com que uma das aplicações não execute.

A solução para este problema é criar um ambiente virtual, uma **árvore de diretórios** que contenha uma **instalação Python** para uma versão particular do Python, além de uma série de **pacotes adicionais**.

Nesse artigo: você pode ver detalhadamente [como criar e configurar um ambiente virtual em Python](#).

Quais são as Bibliotecas e Frameworks mais utilizados?

Um [framework](#) Python é um conjunto de pacotes e módulos que tornam a programação mais **fácil e rápida**. Eles fornecem uma série de ferramentas necessárias na hora de desenvolver uma aplicação.

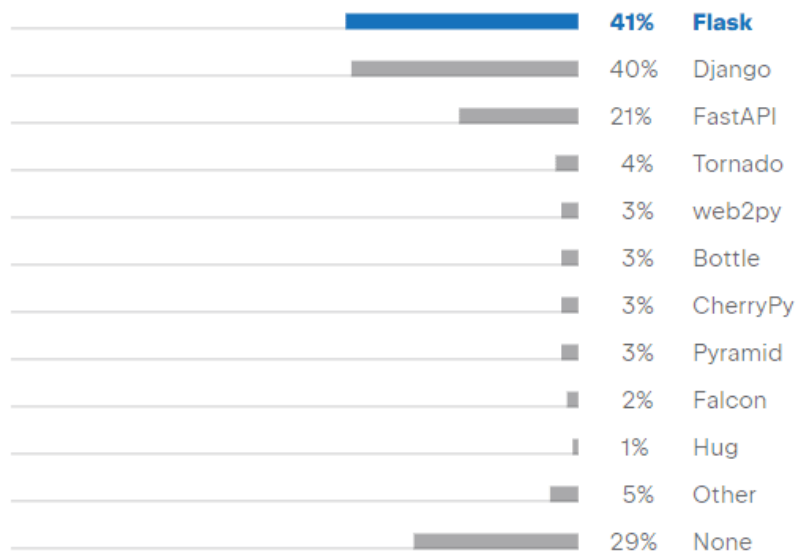
A pesquisa “Python Developers Survey 2021 Results” nos traz os seguintes resultados a respeito dos **frameworks e bibliotecas mais utilizadas**:

//

“Flask, Django e FastAPI ainda são os 3 principais frameworks web Python. O FastAPI, lançado inicialmente no final de 2018, apresenta o crescimento mais rápido, tendo crescido 9 pontos percentuais em relação ao ano anterior. Ao mesmo tempo, em comparação com 2020, a parcela de usuários do Flask diminuiu 5 pontos percentuais.”

Frameworks and Libraries

Web frameworks 100+



Créditos: Python Developers Survey 2021 Results

Empresas que utilizam Python

Podemos destacar algumas das **gigantes de tecnologia** que adotaram Python como sua linguagem queridinha, e a utilizam em áreas como: análise de dados, big data, desenvolvimentos de jogos, inteligência artificial, computação gráfica, automação, dentre outras. Confira a seguir, algumas dessas empresas:

01) Google

Uma das maiores fãs de Python no mundo possui seu próprio jargão, que diz: “Python where we can, C++ where we must” (em tradução livre: **Python onde possível, C++ onde obrigatório**).

Python é uma das linguagens de programação mais populares dentro da companhia, junto com Java e C++.

02) Instagram

Utiliza o Python como principal linguagem de programação desde 2016. Em 2017, realizou uma grande mudança para o [Python 3](#) e anunciou o gerenciamento do maior projeto de desenvolvimento web de [Django](#) escrito inteiramente em Python.

03) Amazon

As sugestões que aparecem na plataforma são resultados de análises de hábitos de compras e padrões de busca, realizadas tanto com Python — [machine learning](#) e [Big Data](#) — quanto pelo Hadoop, banco de dados da Amazon.



04) Spotify

O Spotify **patrocina** conferências e eventos dedicados à linguagem, além de **alimentar as bibliotecas com documentação**, utilizá-la para análise de dados, e também em seus serviços de backend.

05) Facebook

É a terceira linguagem mais usada entre quem desenvolve o Facebook, responsável por publicar diversos projetos de [código aberto](#) usando Python, contribuindo para diversas bibliotecas.

06) Globo

Por último, mas não menos importante, a Globo, que realiza o gerenciamento dos conteúdos de seus portais utilizando Python, que por ser uma linguagem muito dinâmica, proporciona um grande **benefício para tratar do volume altíssimo de informações diárias** que a empresa recebe.

Os melhores livros de tecnologia para ler e se aprofundar

//

Os MELHORES livros de tecnologia para ler em Programação com Roberta Arcoverde / #HipstersPontoTube



Aprenda mais sobre Python gratuitamente

01) Formação: Python para Data Science

Acesse gratuitamente as primeiras aulas da **Formação Python para Data Science**, feita pela [Escola de Dados](#) da Alura e continue aprendendo sobre temas como:

1. [Python para Data Science](#)
2. [Python para Data Science: linguagem e Numpy](#)
3. [Python para Data Science: Funções, Pacotes e Pandas](#)
4. [Python Pandas: tratando e analisando dados](#)
5. [Data Visualization: explorando com Seaborn](#)
6. [Data Science: análise de series temporais](#)
7. [Corretor Ortográfico em Python: aplicando técnicas de NLP](#)

//

[Como aprender melhor? Com Diogo Pires / #HipstersPontoTube](#)



02) Formação: Aprenda a programar em Python com Orientação a Objetos

Acesse gratuitamente as primeiras aulas da **Formação Aprenda a programar em Python com Orientação a Objetos**, feita pela [Escola de Programação](#) da Alura e continue aprendendo sobre temas como:

1. [Python: começando com a linguagem](#)
2. [Python: avançando na linguagem](#)
3. [Python: entendendo a Orientação a Objetos](#)
4. [Python: avançando na orientação a objetos](#)
5. [String em Python: extraindo informações de uma URL](#)
6. [Python Collections parte 1: listas e tuplas](#)
7. [Python Collections parte 2: conjuntos e dicionários](#)
8. [Python Brasil: validação de dados no padrão nacional](#)
9. [Python e TDD: explorando testes unitários](#)

Apostilas da Alura — Você profissional em T

Apostila de Python e Orientação a Objetos da Alura

Mergulhe na [Apostila de Python e Orientação a Objetos](#) e aprenda **tópicos importantes** que envolvem essa linguagem de programação, como:

1. Como aprender Python;
2. O que é Python;
3. Variáveis e tipos embutidos;
4. Introdução ao Pycharm;

5. Estrutura de dados;
6. Funções;
7. Arquivos;
8. Orientação a Objetos;
9. Modificadores de acesso e métodos de classe;
10. Pycharm e Orientação a objetos;
11. Herança e Polimorfismo;
12. Herança Múltipla e Interfaces;
13. Exceções e Erros;
14. Collections;
15. Apêndice - Python2 ou Python3?;
16. Apêndice - Instalação.

Baixe ela completa e as demais apostilas da coleção da Alura em: [Apostilas da Alura - Conteúdo livre para o seu aprendizado.](#)

Conclusão

Neste artigo você aprendeu mais sobre Python entendendo um pouco do porquê ele é considerado uma linguagem versátil, dinâmica e de fácil aprendizagem, além de ter uma noção de sua história.

Você também viu alguns conceitos básicos da linguagem, quais são os ambientes de desenvolvimento mais utilizados por pessoas desenvolvedoras em Python, suas principais aplicações dentro da área de desenvolvimento e algumas das empresas que mais utilizam Python (e em quais processos a linguagem é utilizada por elas).

Para aprender mais e continuar mergulhando nesse universo gigante que a linguagem nos permite explorar, você pode navegar pelos seguinte conteúdos:

- Podcast - [HipstersPontoTube | O que é o Python](#)
- Artigo - [Python: a origem do nome](#)
- Artigo - [Python - uma introdução a linguagem](#)
- Artigo - [Trabalhando com arquivos e diretórios no python](#)
- Artigo - [O que é Compilação e qual o papel dos Compiladores?](#)

- Artigo - [Programação funcional: O que é?](#)
- Artigo - [Programação orientada a objetos e programação estruturada](#)
- Artigo - [O que são as tipagens estática e dinâmica em programação](#)
- Artigo - [O que é Python?](#)



Caroline Carvalho

Analista de qualidade de software, estudante de Engenharia de Computação e de Análise e Desenvolvimento de Sistemas. Faço parte do Scuba Team, exploradora do universo Python , e também gosto muito de DevOps. Nas horas vagas gosto de falar sobre cultura geek e café.

[Artigo Anterior](#)

Node.JS: o que é, como funciona esse ambiente de execução JavaScript e um Guia para iniciar

[Próximo Artigo](#)

JVM: conhecendo o processo de execução de código

Leia também:

- [Listas em Python: operações básicas](#)
- [Python datetime: Como faço para definir data e hora em Python?](#)
- [Python: O que significa if __name__ == '__main__'?](#)
- [Python: Qual a diferença entre == e is?](#)
- [Python: Trabalhando com dicionários](#)
- [Python: a origem do nome](#)
- [Como criar APIs em Python usando FastAPI](#)
- [Como remover linhas e colunas no Pandas](#)
- [Matplotlib uma biblioteca Python para gerar gráficos interessantes](#)

- [Python: A diferença das funções input\(\) e raw_input\(\)](#)
- [Google Colab: o que é, tutorial de como usar e criar códigos](#)
- [Jupyter Notebook: Exemplos de Códigos e Como Usar](#)

Veja outros artigos sobre
[Programação](#)

Quer mergulhar em tecnologia e aprendizagem?

Receba a newsletter que o nosso CEO escreve pessoalmente, com insights do mercado de trabalho, ciência e desenvolvimento de software

Escreva seu email

ME INSCREVA

Nossas redes e apps



Institucional

[Sobre nós](#)[Trabalhe conosco](#)[Para Empresas](#)[Para Escolas](#)[Política de Privacidade](#)[Compromisso de Integridade](#)[Termos de Uso](#)[Status](#)

A Alura

[Formações](#)[Como Funciona](#)[Todos os cursos](#)[Depoimentos](#)[Instrutores\(as\)](#)[Dev em <T>](#)[Luri by ChatGPT](#)

Conteúdos

[Alura Cases](#)[Imersões](#)[Artigos](#)[Podcasts](#)[Artigos de educação corporativa](#)

Fale Conosco

[Email e telefone](#)[Perguntas frequentes](#)

Novidades e Lançamentos

CURSOS

Cursos de Programação

Lógica | Python | PHP | Java | .NET | Node JS | C | Computação | Jogos | IoT

Cursos de Front-end

HTML, CSS | React | Angular | JavaScript | jQuery

Cursos de Data Science

Ciência de dados | BI | SQL e Banco de Dados | Excel | Machine Learning | NoSQL | Estatística

Cursos de Inteligência Artificial

IA para Programação | IA para Dados

Cursos de DevOps

AWS | Azure | Docker | Segurança | IaC | Linux

Cursos de UX & Design

Usabilidade e UX | Vídeo e Motion | 3D

Cursos de Mobile

React Native | Flutter | iOS e Swift | Android, Kotlin | Jogos

Cursos de Inovação & Gestão

Métodos Ágeis | Softskills | Liderança e Gestão | Startups | Vendas