**Quais são os tipos de dados em Python?**

Como você já sabe, 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

1. Ponto Flutuante ou Decimal (float);

Exemplo: 1.1

1. Tipo Complexo (complex);

Exemplo: 8j

1. String (str);

Exemplo: hello

1. Boolean (bool);

Exemplo: true / false

1. List (list);

Exemplo: ['Mônica', 'Ana', 'Bruno', 'Alice']

1. Tuple;

Exemplo: (90, 79, 54, 32, 21)

1. 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, teremos 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, teremos 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 profissão, 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](https://www.alura.com.br/artigos/listas-no-python) 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](https://www.alura.com.br/artigos/listas-no-python) 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](https://www.alura.com.br/artigos/conhecendo-as-tuplas-no-python) 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”](https://www.alura.com.br/artigos/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)

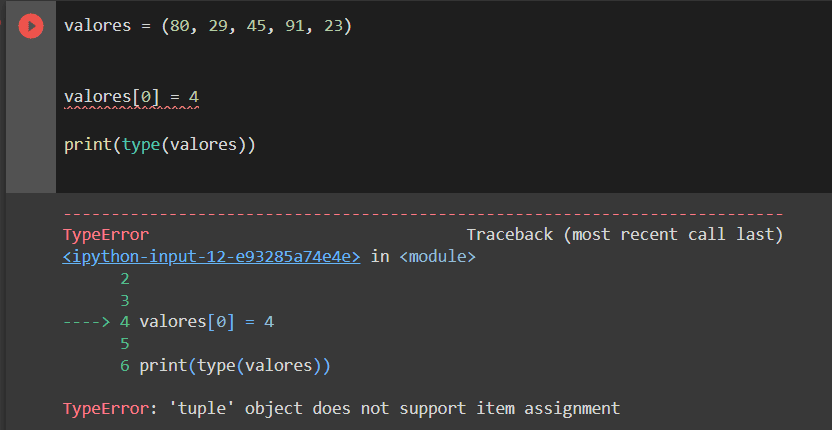
print(type(valores)

print(type(pontos))

<class ‘tuple’>

<class ‘tuple’>

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



**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.

Vejamos a seguir, 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 e 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”.

**Estrutura Match**

A partir do Python 3.10, podemos utilizar **a estrutura match**, que é uma maneira **mais organizada e legível para lidar com múltiplas condições**. Ela nos permite combinar padrões e executar o bloco de código correspondente ao primeiro padrão que for verdadeiro.

media = 6

match media:

case media if media < 5:

print("Você foi reprovado")

case media if 5 <= media < 7:

print("Você fará a recuperação")

case media if media >= 7:

print("Você foi aprovado")

O trecho primeiro verifica a media e avalia cada caso para determinar qual bloco de código executar, da seguinte forma:

* O primeiro case verifica se a média é menor que 5 e, se isso for verdadeiro, imprime "Você foi reprovado".
* O segundo case verifica se a média está entre 5 e 7 (inclusive) e, se isso for verdadeiro, imprime "Você fará a recuperação".
* O terceiro case verifica se a média é maior ou igual a 7 e, se isso for verdadeiro, imprime "Você foi aprovado".

Se nenhuma das condições for satisfeita, o match não executa nenhum bloco de código, mas podemos adicionar um case como padrão para lidar com casos que não prevemos.

Embora a estrutura match em Python não seja exatamente uma estrutura condicional no sentido tradicional como if, elif, e else, ela serve para controlar o fluxo do programa com base em diferentes condições e padrões

**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 em Python**

Com a constante evolução da tecnologia 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](https://www.youtube.com/watch?v=GUanHEGlje4)

**Módulos e bibliotecas em Python**

Você deve já deve 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 casos 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 suas funcionalidades 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](https://www.alura.com.br/artigos/trabalhando-com-arquivos-e-diretorios-no-python).

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?](https://www.alura.com.br/artigos/lidando-com-datas-e-horarios-no-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 e usuárias 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”](https://www.alura.com.br/artigos/como-publicar-seu-codigo-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** caso seja necessária a resolução de algum problema em particular, ou ainda, se o software foi desenvolvido utilizando-se de uma versão obsoleta da interface da biblioteca.

Isso significa que talvez não seja possível uma instalação Python preencher 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 seja executada.

A solução para esse 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**.

Para aprender mais sobre esses ambientes, [leia o artigo Python: Venv e Poetry para criar ambientes virtuais](https://www.alura.com.br/artigos/ven-poetry-no-python).

**Python e Inteligência Artificial**

Python é uma das linguagens de programação mais versáteis e poderosas para trabalhar com inteligência artificial (IA).

Uma das maiores vantagens de usar Python para IA é a disponibilidade de bibliotecas que facilitam a construção de modelos de aprendizado de máquina e deep learning. Algumas das bibliotecas mais populares incluem:

* Scikit-learn: ideal para tarefas de aprendizado de máquina tradicionais, como classificação, regressão e clustering. É fácil de usar e possui uma vasta gama de algoritmos.
* [TensorFlow](https://www.alura.com.br/curso-online-tensorflow-implementando-perceptron-multiplas-camadas) e [Keras](https://www.alura.com.br/curso-online-keras-primeiros-passos-redes-neurais): utilizadas principalmente para deep learning. TensorFlow, desenvolvido pelo Google, é uma biblioteca para construir e treinar redes neurais.

Keras oferece uma interface de alto nível que facilita o desenvolvimento de modelos complexos. A biblioteca pode ser executada em conjunto com o Tensorflow.

* [PyTorch](https://www.alura.com.br/formacao-deep-learning-pytorch): desenvolvido pelo Facebook, é outra biblioteca popular para deep learning, especialmente em pesquisa acadêmica. É conhecido pela sua flexibilidade e facilidade de uso.

Além de criar seus próprios modelos, você também pode utilizar APIs de empresas especializadas em IA para acessar recursos avançados sem precisar construir tudo do zero. Um exemplo notável é a [OpenAI](https://openai.com/pt-BR/), que oferece uma API para IA generativa.

A API da OpenAI permite integrar capacidades avançadas de processamento de linguagem natural (NLP) em suas aplicações. Com a API, é possível realizar tarefas como geração de texto, tradução, resumo e muito mais, utilizando modelos treinados por especialistas em IA.

**Quais são as Bibliotecas e Frameworks mais utilizados?**

Algumas das bibliotecas e frameworks mais utilizados em Python são:

* **NumPy:** para computação numérica, permite a manipulação de arrays multidimensionais e funções matemáticas de alto desempenho.
* **Pandas:** para manipulação e análise de dados, fornece estruturas de dados flexíveis e eficientes para trabalhar com dados tabulares.
* **Matplotlib:** para criação de gráficos e visualizações de dados em Python.
* **TensorFlow:** framework de aprendizado de máquina de código aberto para desenvolver e treinar modelos de aprendizado profundo.
* **Django:** framework web de alto nível que facilita o desenvolvimento rápido e seguro de aplicativos web.
* **Flask:** framework web leve e flexível para a criação de aplicativos web em Python.
* **SciPy:** biblioteca para computação científica que fornece funcionalidades para otimização, interpolação, processamento de sinais, álgebra linear e muito mais.
* **Scikit-learn:** para aprendizado de máquina, fornece algoritmos de classificação, regressão, clustering e pré-processamento de dados.
* **BeautifulSoup:** para extração de dados de páginas web, permitindo a raspagem de dados de forma fácil e eficiente.
* **Requests:** para fazer requisições HTTP de maneira simples e fácil, muito útil para acessar APIs e baixar conteúdo da web.
* **Pillow:** para processamento de imagens, oferece funcionalidades para abrir, manipular e salvar muitos formatos de arquivos de imagem.
* **Asyncio:** para programação assíncrona, permite escrever código concorrente usando a estrutura de async/await, facilitando a execução de operações de I/O de forma eficiente.
* **Tkinter:** biblioteca padrão do Python para a criação de interfaces gráficas (GUI), permitindo o desenvolvimento de aplicativos desktop com uma variedade de widgets.

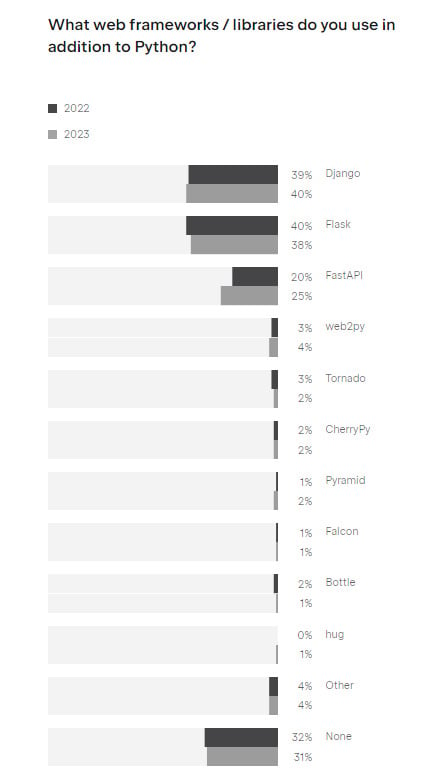
Essas são apenas algumas das bibliotecas e frameworks mais populares em Python, mas existem muitas outras disponíveis para diferentes fins e necessidades.

É importante lembrar que um framework Python é um conjunto de pacotes e módulos que tornam a programação mais fácil e rápida. Para saber ainda mais sobre o assunto, [leia o artigo sobre frameworks](https://www.alura.com.br/artigos/framework-o-que-e-pra-que-serve-essa-ferramenta)

Eles fornecem uma série de ferramentas necessárias na hora de desenvolver uma aplicação.

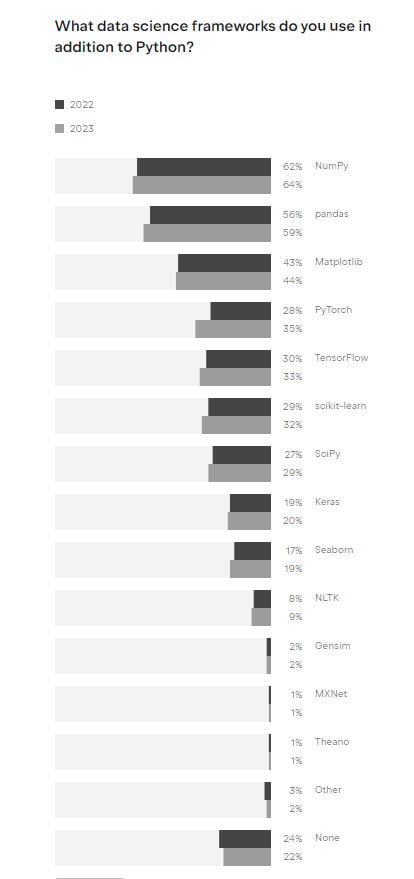
A pesquisa “Python Developers Survey 2023” nos traz os seguintes resultados, reforçando os frameworks e bibliotecas mais utilizadas:

“Flask, Django e FastAPI ainda são os 3 principais frameworks web Python. O FastAPI tem visto um uso crescente nos últimos anos, subindo de 14% em 2021 para 25% em 2023. Este framework é renomado por seu foco em alto desempenho e ergonomia para desenvolvedores, aproveitando os recursos modernos do Python, como type hinting e suporte assíncrono embutido. Ele é especificamente projetado para a construção de APIs com Python, oferecendo uma gama de recursos e benefícios adicionais para aumentar a eficiência do desenvolvimento.”; e



***Créditos:****Python Developers Survey 2023*

“NumPy, pandas e Matplotlib continuam sendo os frameworks mais populares para tarefas de ciência de dados. PyTorch melhorou sua posição em 7 pontos percentuais desde o ano passado, provavelmente como resultado da crescente popularidade do deep learning.”



***Créditos:****Python Developers Survey 2023*

**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](https://www.alura.com.br/empresas/artigos/automacao-de-processos), 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](https://www.alura.com.br/artigos/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](https://www.alura.com.br/artigos/quais-as-diferencas-entre-python-2-e-python-3) e anunciou o gerenciamento do maior projeto de desenvolvimento web de [Django](https://www.alura.com.br/artigos/django-instalacao-configuracao-e-escrevendo-seu-primeiro-app) 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*](https://www.alura.com.br/artigos/machine-learning) e [Big Data](https://www.alura.com.br/artigos/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 back-end.

**05) Facebook**

É a terceira linguagem mais usada entre quem desenvolve o Facebook, responsável por publicar diversos projetos de [código aberto](https://www.alura.com.br/artigos/open-source-uma-breve-introducao) 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.