







Tabela de conteúdos

Curso Python Arretado	1.1
Introdução	1.2
Montagem de ambiente	1.3
Primeiro algoritmo	1.4
Variáveis e seus tipos	1.5
Estruturas de Dados Básicas	1.6
Condicionais e Loops	1.7
Funções	1.8
Glossário	1.9
Bibliografia	1.10

- [1. Apresentação](#)

1. Apresentação

Este é o material oficial do curso Python Arretado construído por Geovanne Atanzio (, , ) e Mayara Machado (, , )

Copyright © 2022 Geovanne Atanzio and Mayara Machado all right reserved , powered by GitbookArquivo revisado pela última vez em 28/05/2022

- **1. Introdução**
 - **1.1. História**
 - **1.1.1. Origem do nome**
 - **1.1.2. Ruptura entre versões**
- **2. Aplicação da Linguagem**
 - **2.1. Desenvolvimento Web**
 - **2.2. Computação Científica**
 - **2.3. Inteligência Artificial e Aprendizado de Máquina**

1. Introdução

[Python](#) é uma linguagem de programação poderosa e divertida que tem crescido devido sua compatibilidade — roda na maioria dos Sistemas Operacionais (SOs) — e capacidade de auxiliar outras linguagens. Com ela, é possível fazer diversas coisas como: análise de dados, inteligência artificial e construção de sistemas *web*, *mobile* e *desktop*.

A simplicidade do Python encoraja a programação modularizada. Dessa forma, códigos escritos para um programa podem ser reutilizados em outros. Por conta disso, a linguagem provê uma vasta coleção de módulos que podem ser utilizados como base ou complemento para novos projetos.

1.1. História

Python foi criada em 1989 por Guido Van Rossum, no [Centro de Matemática Stichting \(CWI\)](#) na Holanda, com o objetivo de criar uma linguagem de programação que tivesse sintaxe semelhante a linguagem de programação ABC e fosse capaz de interagir com o SO [Amoeba](#). Nessa época, a aceitação de Python foi tão boa que em 1991 Guido lançou sua criação na Internet.

1.1.1. Origem do nome

Muitas pessoas costumam pensar que o nome Python tem origem em um tipo de cobra, já que o sua logo mostra a imagem de uma cobra azul e outra amarela. No entanto, o nome da língua vem do gosto de seu criador pelos humoristas britânicos [Monty Python](#).

1.1.2. Ruptura entre versões

Em dezembro de 2008 foi lançado a versão 3.0 do Python, mais conhecida como Python3. Nela, houve uma quebra de compatibilidade com o Python2 — família de versões anterior da linguagem. Essa ruptura foi feita para corrigir falhas que foram descobertas e reestruturar a linguagem.

2. Aplicação da Linguagem

Pela sua versatilidade, boa curva de aprendizado e vastidão na variedade de bibliotecas, que turbinam a linguagem tornando-a pronta para resolver os mais diversos problemas, que o Python é uma das linguagens mais utilizadas no mercado. Crescendo cada vez mais e com uma grande comunidade, a linguagem Python é uma escolha muito utilizada no mercado os mais diversos projetos.

2.1. Desenvolvimento Web

Python oferece diversas opções de ferramentas para poder desenvolver dos mais simples aos mais robustos projetos que serão executados na web. Nas aplicações da internet nós temos a segmentação entre Frontend e Backend onde as aplicações Python podem ser utilizados como tecnologia na estrutura presente no Backend de uma aplicação. Responsáveis pelo processamento da ação executada na internet, as aplicações que utilizam Python possui disponível diversas bibliotecas que apresentam suporte para os protocolos de internet e outras funcionalidades mais específicas de programação. Empresas como a Globo, Google, Amazon, Mercado Livre utilizam Python em suas stacks.

2.2. Computação Científica

Contando com bibliotecas bem robustas e mantidas por uma grande comunidade ativa, Python conta com bibliotecas diversas para realizar cálculos matemáticos que podem ser utilizados por cientistas, pesquisadores e etc. Alguns dos pacotes mais conhecidos são o SciPy e Numpy, que contém implementação de diversos pacotes para realizar cálculos matemáticos e plotagem de gráficos, assim como o Matplotlib. Também possui bibliotecas que possibilitam programassão paralela e visualização de dados.

2.3. Inteligência Artificial e Aprendizado de Máquina

Python hoje é uma das linguagens mais utilizadas para implementação e processamento de algoritmos de inteligência artificial e aprendizado de máquina. Ele é hoje uma linguagem que possui diversas bibliotecas com uma boa maturidade de desenvolvimento, contendo um amplo acervo de modelos matemáticos e de inteligência artificial que tornam o desenvolvimento de modelos de aprendizado de máquinas mais seguros e estáveis. Algumas das bibliotecas mais famosas são a Scikit-Learn, SciPy, TensorFlow e Keras.

Outra aplicação da linguagem é a sua integração com ferramentas open source para processamento de Big Data. Libs como PySpark e Faust.

Copyright © 2022 Geovanne Atanazio and Mayara Machado all right reserved , powered by GitbookArquivo revisado pela última vez em 28/05/2022

- **1. Montagem de ambiente**
 - **1.1. Python em ambientes locais**
 - **1.1.1. Python no GNU/Linux e no macOS**
 - **1.1.2. Python no Windows**
 - **1.1.3. Interface de desenvolvimento**
 - **1.2. Python em ambientes online**

1. Montagem de ambiente

Para começar a trabalhar com Python é preciso operar uma máquina que entenda a linguagem. Atualmente, é possível utilizar Python em ambientes locais e *online*.

1.1. Python em ambientes locais

O Python está disponível para os três principais SOs feitos para computadores pessoais — sendo eles GNU/Linux, Windows e macOS. Caso opte por aprender Python em algum deles, será preciso fazer configurações que facilitam o trabalho com a linguagem. Esse processo é conhecido como montagem de ambiente.

A seguir será explicado, de forma breve, como fazer a montagem de ambiente Python em cada um dos SOs citados.

1.1.1. Python no GNU/Linux e no macOS

No GNU/Linux e no macOS, o Python já vem instalado. Porém, em alguns casos, a versão instalada é o Python2 que vem sendo substituída gradativamente pelo Python3. Para saber qual versão está instalada, basta acessar o terminal do SO e executar o comando:

```
python --version
```

Se o Python3 já estiver instalado, já é possível executá-lo. Entretanto, no caso de somente o Python2 estiver instalado, será preciso baixar o Python3 por meio do [site oficial](#) e instalá-lo.

1.1.2. Python no Windows

No SO Windows, para que um usuário possa utilizar o Python, é preciso efetuar sua instalação — ao contrário do GNU/Linux e do macOS, no Windows nenhuma versão do Python vem instalada. Entretanto, existe a possibilidade de alguém já ter feito a instalação do Python no computador. Para verificar basta acessar o terminal do Windows e executar o comando:

```
python --version
```

Se o Python3 já estiver instalado, é possível executá-lo. Caso contrário, será preciso baixar o Python3 por meio do [site oficial](#) e instalá-lo — o mesmo vale se for identificado a instalação somente do Python2.

1.1.3. Interface de desenvolvimento

Com a instalação do Python finalizada, o próximo passo é instalar uma ferramenta que auxilie e facilite a escrita de código, popularmente conhecida como *Integrated Development Environment (IDE)* ou, em português, Ambiente de Desenvolvimento Integrado. Atualmente existem muitas *IDEs*, aqui indicaremos o [Visual Studio Code](#), que está disponível para GNU/Linux, Windows e macOS. Sinta-se a vontade para escolher a ferramenta que mais te agrade.

1.2. Python em ambientes online

Também é possível utilizar o Python em ambientes *online*. Na internet existem diferentes ferramentas desse tipo, aqui indicaremos o [Replit](#). Sinta-se a vontade para escolher a ferramenta que mais te agrade.

Copyright © 2022 Geovanne Atanazio and Mayara Machado all right reserved , powered by GitbookArquivo revisado pela última vez em 27/05/2022

- **1. Primeiro algoritmo**
 - **1.1. Saída de dados**
 - **1.2. Entrada de dados**

1. Primeiro algoritmo

Com o ambiente de desenvolvimento pronto já é possível começar a programar. Nessa seção será ensinado como fazer um algoritmo simples em Python utilizando as operações de entrada e saída de dados — conhecidas popularmente pela gíria I/O, que em inglês significa *Input/Output* ou, em português, *Entrada/Saída*.

⚠ Observação:

Para começar a escrever algoritmos é preciso criar um arquivo com a extensão da linguagem que se deseja trabalhar. Arquivos com a extensão `.py` indicam que os códigos escritos nele são da linguagem Python. Com o arquivo criado, é só utilizar o *IDE* para abri-lo e começar a codificar. Caso esteja utilizando um ambiente de desenvolvimento *online*, geralmente, o processo é ainda mais simples, visto que o arquivo `.py` é criado automaticamente.

1.1. Saída de dados

Exibir dados com Python é uma tarefa simples. Nesse primeiro exercício, o objetivo é fazer o *output* da frase "Olá Mundo!". Ela é popularmente apresentada para novos programadores quando estão dando os primeiros passos no mundo da programação.

Para conseguir exibir a mensagem, é preciso utilizar o método `print()`. Dentro do parêntese, coloca-se o que deseja exibir no terminal. Assim, para conseguir exibir a frase "Olá Mundo!", basta escrever :

```
print("Olá Mundo!")  
# output = Olá Mundo!
```

Quando o arquivo for executando o algoritmo escrito será interpretado e a mensagem "Olá Mundo!" aparecerá no terminal.

1.2. Entrada de dados

Fazendo uma pequena alteração no código é possível fazer o *output* de uma mensagem escrita no teclado ao invés de um texto previamente definido. Para isso basta utilizar o método `input()` e guardar os dados inseridos. Em seguida, é só exibir os dados com o método `print()`, da seguinte forma:

```
dados = input("Insira algo aqui: ")  
print(dados)
```

Ao executar um arquivo com o código a cima, o terminal exibirá a mensagem "Insira algo aqui: " e irá aguardar a entrada de uma mensagem. Por isso, para continuar a execução do código é preciso digitar a mensagem e apertar a tecla `Enter` . Em seguida a variável `dados` irá guardar a mensagem escrita. Por fim, a variável `dados` é passada para o método `print()` , que faz o *output* da mensagem.

Copyright © 2022 Geovanne Atanazio and Mayara Machado all right reserved , powered by GitbookArquivo revisado pela última vez em 27/05/2022

- **1. Variáveis e seus tipos**
 - **1.1. Tipos numéricos**
 - **1.1.1. Inteiro (int)**
 - **1.1.2. Ponto flutuante ou decimal (float)**
 - **1.1.3. Complexo (complex)**
 - **1.2. String (str)**
 - **1.3. Boolean (bool)**

1. Variáveis e seus tipos

Quando se trata de linguagens de programação, é comum identificar o tipo de dado que uma variável receberá. No entanto, em Python, isso não é necessário. Por ser 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 — é fácil banalizar a importância de conhecer os tipos de variáveis.

Porém, compreender esse assunto é muito importante para quem desenvolve código. Isso porque, durante uma execução, o tipo da variável pode ser alterado e entender esse comportamento pode ser o diferencial para conseguir realizar a construção de um algoritmo.

Durante esta seção será utilizado o método `type()` do Python. Com ele, é possível saber o tipo de uma variável ou de um dado com precisão.

1.1. Tipos numéricos

Os tipos de dados usados para números se dividem em três conjuntos: inteiro, ponto flutuante e complexos.

1.1.1. Inteiro (int)

Esse tipo representa o conjunto dos números inteiros, ou seja, são os números positivos e negativos que não apresentam parte decimal. A seguir, será apresentado um algoritmo que insere um valor inteiro dentro de uma variável e exibe o seu tipo, que no caso é um `int`, veja:

```
numero = 7
print(type(numero))
# output = <class 'int'>
```

⚠ Observação:

Os números inteiros podem ser representados também nos formatos binário, hexadecimal e octal. Abaixo há exemplos dessas representações de forma respectiva:

- 0b10000, 0b11111111
- 0x01, 0x10, 0xFF
- 0o1, 0o20, 0o377

1.1.2. Ponto flutuante ou decimal (float)

Representação de números reais, ou seja, os que contêm casas decimais. A seguir, será apresentado um algoritmo que insere um valor decimal dentro de uma variável e exibe o seu tipo, que no caso é um

`float`, veja:

```
altura = 7.1
print(type(altura))
# output = <class 'float'>
```

1.1.3. Complexo (complex)

Os números complexos são mais utilizados na engenharia e pesquisa. No Python, a parte imaginária do número recebe a letra `j` — podendo ser maiúscula ou minúscula. A seguir, será apresentado um algoritmo que insere um numero complexo dentro de uma variável e exibe o seu tipo, que no caso é um `complex`, veja:

```
complexo = 7j
print(type(complexo))
# output = <class 'complex'>
```

1.2. String (str)

Trata-se de um conjunto de caracteres dispostos numa determinada ordem, geralmente utilizada para representar palavras, frases ou textos. No Python, para indicar que o código escrito é uma string, é preciso colocar aspas simples (`'`) ou aspas duplas (`"`) tanto no inicio, quanto no fim da cadeia de caracteres. A seguir, será apresentado um algoritmo que insere uma *string* dentro de uma variável e exibe o seu tipo, que no caso é um `str`, veja:

```
texto = 'Olá mundo!'
print(type(texto))
# output = <class 'str'>
```

1.3. Boolean (bool)

É um tipo de dado lógico que pode assumir apenas dois valores: falso ou verdadeiro — `False` ou `True` em Python. A seguir, será apresentado um algoritmo que insere um *boolean* dentro de uma variável e exibe o seu tipo, que no caso é um `bool`, veja:

```
mentira = True
print(type(mentira))
# output = <class 'bool'>
```

Na lógica computacional, considera-se o `False` como 0 e o `True` como 1. Por isso, no Python, um *boolean* é um subtipo inteiro onde 1 é equivalente a `True` e 0 equivale a `False`.

Copyright © 2022 Geovanne Atanazio and Mayara Machado all right reserved , powered by GitbookArquivo revisado pela última vez em 27/05/2022

- **1. Estrutura de Dados Básicas**
 - **1.1. Listas**
 - **1.1.1. Métodos de listas:**
 - **1.2. Tuplas**
 - **1.3. Sets**
 - **1.4. Dicionários**

1. Estrutura de Dados Básicas

Estrutura de dados fazem o agrupamento de dados e são úteis para resolver problemas em diversas situações, no Python nós iremos ter algumas implementações delas. As principais estruturas são as Listas, Tuplas, Sets e Dicionários e nesta seção veremos as principais características de cada uma.

1.1. Listas

No Python, uma lista é um tipo de dado que armazena uma sequência de diferentes tipos de dados, seus valores são definidos separados por vírgula, **(,)**, e podem ser mutáveis, ou sejam, uma vez definidas seus valores podem ser alterados. Seus valores são acessados através de um índice que é representado por um valor inteiro iniciando de 0.

```
lista1 = [1,2,3,4,5]
lista2 = ["P", "Y", "T", "H", "O", "N"]
lista3 = ["PY", 2022]
```

Podemos construir listas a partir de strings, tuplas, sets e dicionários a partir da função `list()`.

```
string = "python"
print(list(string))

## output: ['p', 'y', 't', 'h', 'o', 'n']
```

1.1.1. Métodos de listas:

Listas apresentam uma vasta quantidade de métodos, dentre eles destaque:

- `append()`: adiciona novos valores ao final da lista

```
>>> frutas = ['laranja', 'maçã', 'pera', 'banana', 'kiwi', 'maçã', 'banana']

>>> frutas.append('uva')
>>> frutas
['banana', 'maçã', 'kiwi', 'banana', 'pera', 'maçã', 'laranja', 'uva']
```

- `extend()`: estende os valores de uma lista adicionando todos os valores de outro objeto iterável, por exemplo, lista A pode ser estendida com os valores uma lista B.

```
>>> frutas = ['laranja', 'maçã', 'pera', 'banana', 'kiwi', 'maçã', 'banana']
>>> frutas.extend(["abacaxi", "melão"])
['maçã', 'maçã', 'banana', 'banana', 'uva', 'kiwi', 'laranja', 'pera', 'abacaxi', 'melão']
```

- `count()`: conta a quantidade de ocorrências de um dado item na lista. ``python

```
frutas = ['laranja', 'maçã', 'pera', 'banana', 'kiwi',
          'maçã', 'banana']

frutas.count('maçã') 2 frutas.count('tangerine') 0
```

- `sort()`: ordena a lista de forma crescente.

```
``python
>>> frutas = ['laranja', 'maçã', 'pera', 'banana', 'kiwi', 'maçã', 'banana']

>>> frutas.sort()
>>> frutas
['maçã', 'maçã', 'banana', 'banana', 'uva', 'kiwi', 'laranja', 'pera']
```

- `reverse()`: reverte as posições dos valores na lista, o primeiro valor se tornará o último e o último se tornará o primeiro.

```
>>> frutas = ['laranja', 'maçã', 'pera', 'banana', 'kiwi', 'maçã', 'banana']

>>> frutas.reverse()
>>> frutas
['banana', 'maçã', 'kiwi', 'banana', 'pera', 'maçã', 'laranja']
```

Existem outros métodos mais que são bastante interessantes, como atividade complementar recomendo essa [leitura](#).

1.2. Tuplas

As tuplas são estrutura de dados muito similar às Listas, são tipos de dados que armazenam múltiplos tipos de dados, separados por uma vírgula. Sua principal diferença é que tuplas representam objetos ordenados e imutáveis, ou seja, uma vez definidos seus valores não podem ser alterados.

```

frutas = ('maçã', 'manga', 'banana')
print(frutas)
# output: ('maçã', 'manga', 'banana')

# ou

frutas_2 = 'maçã', 'manga', 'banana'
print(frutas_2)
# output: ('maçã', 'manga', 'banana')

frutas_3 = ('abacaxi',)
print(frutas_3)
# output: ('abacaxi',)

```

As tuplas podem ser definidas com ou sem parentêsis. Para definir tuplas com valores únicos nós utilizando uma virgula após o valor da tupla, importante ressaltar que se não definirmos com a virgula após um valor única de uma tupla, o Python não irá considerar a variável como uma tupla.

```

eh_tupla = ('maçã')
print(type(eh_tupla)) # OUTPUT: <class 'str'>

eh_tupla_2 = 'maçã',
print(type(eh_tupla_2)) # OUTPUT: <class 'tuple'>

nao_eh_tupla = ('maçã',)
print(type(nao_eh_tupla)) # OUTPUT: <class 'tuple'>

```

Por ser um tipo de dados imutável, não temos muitos métodos que interagem com a tupla como é o caso das listas, porém podemos acessar as tuplas através de `index` e calcular o tamanho da tupla com o método `len()`.

```

frutas = ('maçã', 'manga', 'banana')

print(len(frutas))
# output: 3

print(frutas[1])
# output: "manga"

```

1.3. Sets

Na matemática os Sets são os conjuntos de objetos distintos que forma um grupo, e é exatamente isso que o Sets representa no Python. Temos algumas definições que são básicas quanto aos conjuntos, eles precisam:

1. Não conter valores duplicados;
2. Não possui ordem nos seus elementos;
3. Os seus elementos são imutáveis porém o Set como um todo é mutável.

Vamos ver exemplos que demonstram essas características:

```
conjunto = {0, 2, 4, 6}
print(conjunto)
# output = {0, 2, 4, 6}

conjunto_2 = {"zero", 2, "quatro", 6.0}
print(conjunto_2)
# output = {"zero", 2, "quatro", 6.0}

conjunto_3 = {0, 2, 2, 4, 4}
print(conjunto_3)
# output = {0, 2, 4}

conjunto_4 = set([1, 1, 2, 3])
print(conjunto_4)
# output = {1, 2, 3}
```

Como podemos ver, existem duas formas de se definir Sets, eles podem ser definidos com o uso de chaves ({}) e vírgulas (,) ou passando um iterável (lista, tupla, dicionário) para o método `set()`. Outra coisa também que podemos observar é que mesmo que passemos valores duplicados no momento de criação de um Set, este não ficará com o valor pois apenas aceita valores não duplicados.

Para adicionar novos valores a um Set já definido podemos fazer da seguinte forma:

```
conjunto = {0, 2, 4, 6}
print(conjunto)
# output = {0, 2, 4, 6}

conjunto.add(8) # adiciona um elemento por vez
print(conjunto)
# output = {0, 2, 4, 6, 8}

conjunto.update([10, 12]) # adiciona múltiplos elementos por vez
print(conjunto)
# output = {0, 2, 4, 6, 8, 10, 12}
```

Para remover valores de um Set existem algumas possibilidades diferentes que podem ser utilizadas de acordo com o cenário que você tenha no seu algoritmo, vamos de exemplos:

```

conjunto = {0, 2, 4, 6}
conjunto.remove(6) # se não encontrar o elemento o código lançará um erro.
print(conjunto)
# output = {0, 2, 4}

conjunto.discard(4) # se não encontrar o elemento nada acontecerá ao Set.
print(conjunto)
# output = {0, 2}

conjunto.discard(4)
print(conjunto)
# output = {0, 2}

conjunto.clear() # remove todos os elementos de um Set.
print(conjunto)
# output = set()

```

1.4. Dicionários

Você pode pensar nesse tipo de dado como exatamente um dicionário onde teremos uma chave e para cada chave teremos um valor. Os valores das chaves em um dicionário são sempre únicas, e elas também são **case-sensitive**, já os seus valores podem ou não serem únicos. As chaves são valores imutáveis em um dicionário, uma vez definidos só podem ser excluídos e nunca alterados, seus valores no entanto podem sofrer modificações.

Vamos de exemplos sobre como criar o seu dicionário em Python:

```

dict_exemplo = {'nome': 'Maria', 'idade': 22}
print(dict_exemplo)
# output = {'nome': 'Maria', 'idade': 22}

empty_dict = {}
print(empty_dict)
# output = {}

frutas = {1: 'maçã', 2: 'banana', 3: 'cereja'}
print(frutas)
# output = {1: 'maçã', 2: 'banana', 3: 'cereja'}

dicionarios_misto = {1: 'vermelho', 'nome': 'Jose'}
print(dicionarios_misto)
# output = {1: 'vermelho', 'nome': 'Jose'}

```

Para acessar valores de dicionários nós utilizamos das suas chaves.

```
dict_exemplo = {'nome': 'Maria', 'idade': 22}

print(dict_exemplo["nome"]) # lança exceção se essa chave não existir
# output = Maria

print(dict_exemplo.get("nome")) # retorna None ou valor default caso não encontre
# output = Maria

print(dict_exemplo.get("inexistente")) # j
# output = None

print(dict_exemplo.get("inexistente", "Não tem valor")) # j
# output = Não tem valor
```

Para atualizar um dicionário ou para adicionar novos valores nós podemos fazer a partir das chaves

```
dict_exemplo = {}
print(dict_exemplo)
# output = {}

dict_exemplo["nome"] = "Maria"
print(dict_exemplo)
# output = {"nome": "Maria"}

dict_exemplo["idade"] = 22
print(dict_exemplo)
# output = {'nome': 'Maria', 'idade': 22}

dict_exemplo["nome"] = "Jose"
print(dict_exemplo)
# output = {'nome': 'Jose', 'idade': 22}

dict_exemplo["idade"] = "25"
print(dict_exemplo)
# output = {'nome': 'Jose', 'idade': "25"}
```

E para deletar uma chave de um dicionário nós utilizamos a cláusula `del`.

```
meu_dicionario = {1: 'Jose', 2: 'Maçã', 3: 'Maria', 4: 'Banana, Cereja, Kiwi'}
del meu_dicionario[4]
print(meu_dicionario)

# output = {1: 'Jose', 2: 'Maçã', 3: 'Maria'}
```

Copyright © 2022 Geovanne Atanazio and Mayara Machado all right reserved , powered by GitbookArquivo revisado pela última vez em 28/05/2022

- **1. Controle de fluxos: Condicionais e Loops**
 - **1.1. Condicionais**
 - **1.1.1. If**
 - **1.1.2. If Else**
 - **1.1.3. If Elif Else**
 - **1.2. Loops**
 - **1.2.1. For loops**
 - **1.2.2. While loops**
 - **1.3. Break, Pass e Continue**
 - **1.3.1. Break**
 - **1.4. Continue**
 - **1.4.1. Pass**

1. Controle de fluxos: Condicionais e Loops

Podemos dizer que todo algoritmo Python consistirá em um conjunto de definições de operações que devem ser executadas pelo algoritmo. Por exemplo, podemos ter um algoritmo cujo o intuito é de realizar a soma de dois valores, desse modo nosso algoritmo estará definindo uma operação de adição do número A e do número B. Um algoritmo pode ser composto por diversos tipos de definição de operações, com número, palavras e etc. Mas da forma como estamos fazendo até o momento não nos permite controlar ou definir nenhum fluxo para a execução das nossas operações.

Vamos imaginar com um exemplo da vida real, quando encontramos com um amigo e vamos cumprimentá-lo nós falamos *Bom dia!* se estiver de dia, *Boa tarde!* se estiver de tarde e *Boa noite!* se estiver a noite. Ou seja, definimos qual a operação estamos realizando com base em uma condição externa a nossa operação, no caso do nosso exemplo definimos com base no horário do dia. Algo similar acontece com nossos algoritmos, podemos construir diferentes fluxos com base em condições definidas no nosso código. Vamos entender mais sobre como podemos definir o controle de fluxos com Python.

1.1. Condicionais

O tipo de fluxo condicional é composto pelo uso de um dos três tipos de termos da linguagem:

- ```
if (condição):
 (ação)
```



- ```
if (condição):
    (ação 1)
else:
    (ação 2)
```

- ```
if (condição):
 (ação 1)
elif (condição):
 (ação 2)
else:
 (ação 3)
```

### 1.1.1. If

O `if` é a estrutura mais simples de condição e representa um **se**.  
Veamos o exemplo abaixo:

```
turno = "tarde"

if turno == "manhã":
 print("Bom dia!")

if turno == "tarde":
 print("Boa Tarde!")

if turno == "noite":
 print("Boa noite!")
```

Temos no exemplo 3 definições de operação que verificam **SE** a variável `turno` tem um dos valores comparados. O algoritmo então irá executar a primeira verificação da condicional e identificar se a variável possui o valor `"manhã"` e essa verificação irá retornar o valor `False` pois a variável `turno` não possui o valor `"manhã"` e por isso não irá executar o trecho de código `print("Bom dia!")`. A seguir o algoritmo irá verificar se a variável `turno` é igual ao valor `"tarde"` e essa operação irá retornar `True`, com isso o algoritmo irá entrar nesse novo fluxo marcado pela condicional e imprimir na console o resultado da operação `print("Boa Tarde!")`. Por fim, o algoritmo irá verificar se a variável `turno` é igual ao valor `"noite"` e essa verificação irá retornar `False`, encerrando o algoritmo.

Então como podemos observar, o algoritmo executa todas as verificações de condicional e executa somente aqueles que se enquadram no que a expressão de condicional está informando. Enquanto isso pode ser útil em alguns casos, no caso do nosso exemplo não é, pois se identificamos que o turno já equivale a tarde não faz sentido verificar se o turno também é noite. Por isso podemos ter outras estruturas de condicional.

### 1.1.2. If Else

Podemos reformular nosso exemplo para dizer `Boa tarde!` toda vez que for tarde e quando for outro turno o algoritmo deverá dizer outro tipo de cumprimento. Na estrutura If Else nós teremos a relação Se- Senão, onde se a condição não for satisfeita a segunda ação sempre será executada, mas se a condição for verdadeira essa ação será a única a ser executada.

```
turno = "tarde"

if turno == "tarde":
 print("Boa Tarde!")
else:
 print("Olá!")
```

Dessa forma, o algoritmo irá verificar se o turno é igual a `tarde` e se não for ele irá executar o `olá!`. Mas podemos ver que esse código não está completo com o cenário que tínhamos previamente, então podemos reformular ele de uma forma diferente utilizando a estrutura If-Else.

```
turno = "tarde"

if turno == "manhã":
 print("Bom dia!")
else:
 if turno == "tarde":
 print("Boa Tarde!")
 else:
 print("Boa noite!")
```

Nesse novo formato nós teremos uma estrutura condicional mais aninhada, onde o else somente será executado se a condicional do primeiro if `turno == "manhã"` for falsa, e dentro do else nós temos outro if-else. No nosso exemplo teremos a seguinte sequência de passos:

1. Verifica se o turno é igual a manhã e retorna falso, entrando na ação definida no else.
2. Dentro do else ele cai em uma nova condicional, onde a verificação de se turno é igual a tarde o que retorna verdadeiro. Nesse ponto o algoritmo irá imprimir no console `Boa Tarde!` e encerrar, como a condição do SE foi completada não é necessário executar a ação do SENÃO.

Ainda que agora nosso código funcione um pouco melhor, encerrando o algoritmo quando a condição é verdadeiramente satisfeita, essa ainda não é a estrutura que melhor representa o nosso algoritmo. Vamos seguir para o próximo exemplo.

### 1.1.3. If Elif Else

Como podemos ter um cenário onde apenas queremos executar um tipo de cumprimento e ele é de acordo com o valor atribuído para variável `turno`, podemos reformular o algoritmo para verificar três condicionais. Dessa forma, seria mais eficiente identificar se é manhã, tarde ou noite para saber qual o valor deve ser impresso no console. E supondo que o valor de turno seja por exemplo `"madrugada"`, um turno não mapeado no nosso fluxo, podemos retornar o valor padrão de `"Olá!"`.

```
turno = "tarde"

if turno == "manhã":
 print("Bom dia!")

elif turno == "tarde":
 print("Boa Tarde!")

elif turno == "noite":
 print("Boa noite!")

else:
 print("Olá!")
```

Com essa modificação podemos aproveitar o melhor do que o controle de fluxos do Python nos oferece, podemos misturar expressões e combinar a estrutura If Elif e Else para definir melhor o fluxo que o nosso algoritmo deve seguir.

## 1.2. Loops

No geral, loops representam estruturas que ficam executando repetidamente até uma condição ser completada. Dessa forma, loops são compostos pela sua inicialização, definição da condição e incrementação. Podemos combinar o uso de loops com as condicionais e assim montar fluxos que executam repetidas operações e seguem determinadas condições para a sua execução. Vamos observar os exemplos:

### 1.2.1. For loops

A estrutura do For loop é composta pela definição `for (contador) in (incremento): (ação)`, que representa algo do tipo "por X interações faça a ação".

```
limite_iteracoes = 5

for iteracao in range(0, limite_iteracoes):
 print(iteracao)

output:
0
1
2
3
4
```

Nesse exemplo, para fazer o incremento do valor da nossa iteração utilizamos a função `range(inicio, fim)` que realiza o somatório de unidade em unidade do valor inicial, no nosso caso 0, até o valor final, no nosso caso 5. Importante ressaltar que o `range` não compreende o valor definido no final, se quiséssemos incluir no nosso `print` o número 5 então devemos definir assim: `range(0, 6)`. Também podemos definir de quanto em quanto deve ser a iteração, por exemplo:

```
for iteracao in range(0, 10, 3):
 print(iteracao)

output:
0
3
6
9
```

No caso do exemplo queremos que seja impresso no console os valores que estejam no intervalo entre 0 e 10 começando de 0 e que seja acrescido de 3.

Também podemos combinar a execução do `for` loop com condicionais, como podemos ver no exemplo abaixo.

```
Encontros de Segunda, Terça, Quarta, Quinta e sexta.
turnos_encontro = ["manhã", "noite", "manhã", "tarde", "madrugada"]
for iteracao in turnos_encontro:
 if turno == "manhã":
 print("Bom dia!")

 elif turno == "tarde":
 print("Boa Tarde!")

 elif turno == "noite":
 print("Boa noite!")

 else:
 print("Olá!")

output:
Bom dia!
Boa noite!
Bom dia!
Boa Tarde!
Olá!
```

### 1.2.2. While loops

O conceito para o While loop é o mesmo do For mas a sua implementação é complementamente diferente. Nesse Loop nós teremos uma definição do tipo "enquanto condição faça", vamos dar uma olhada no exemplo.

```
valor = 0
limite = 10
while valor < limite:
 print(valor)
 valor = valor + 2

output
0
2
4
6
8
```

Na estrutura do While nós iremos ter a definição de uma condicional `valor < limite` para determinar enquanto o loop ainda deve acontecer e em qual condição o loop deve ter sua execução terminada. Aliado a condição também precisamos definir previamente um incremento que será utilizado pelo loop, no caso do exemplo `valor`, e esse incremento deve ser acrescido de alguma forma durante a ação.

A condição não precisa necessariamente ser numérica, podemos por exemplo ter apenas uma variável booleana, por exemplo:



```

turnos_encontro = ["manhã", "noite", "madrugada", "manhã", "tarde"]
eh_madrugada = False
contador = 0
while not eh_madrugada:
 if turnos_encontro[contador] == "madrugada":
 eh_madrugada = True
 else:
 print("Olá!")
 contador += 1

output
Olá!
Olá!

```

Nesse exemplo nós vamos executar a impressão de "Olá!" no console até que a variável `eh_madrugada` passe a ser verdadeira. Nesse exemplo vemos algumas coisas novas também, como por exemplo o uso de condicionais juntos com o While Loop, o acesso a valores de uma lista e uma nova forma de fazer soma com a mesma variável. Mas essa ainda não é a melhor forma de resolver esse problema, pois se nunca tiver um encontro na madrugada teremos esse loop executando eternamente? Logo mais esse algoritmo irá lançar um erro no console, vejamos no próximo tópico como podemos melhora-lo

## 1.3. Break, Pass e Continue

Temos também algumas palavras que podemos incrementar os nossos loops, como por exemplo.

### 1.3.1. Break

Podemos utilizar o Break para quebrar um loop de acordo com uma condicional.

```

turnos_encontro = ["manhã", "noite", "madrugada", "manhã", "tarde"]
contador = 0
while contador < len(turnos_encontro):
 if turnos_encontro[contador] == "madrugada":
 break
 else:
 print("Olá!")
 contador += 1

output
Olá!
Olá!

```

Reescrevendo o nosso algoritmo para fazer um loop com o número de iterações iguais a quantidade de valores definidos na lista

`turnos_encontro` . Observe que removemos a variável booleana `eh_madrugada` e substituímos a ação da condicional com uma cláusula **Break** que irá terminar a execução do loop quando a condição for

satisfeita. Outra forma de corrigir esse código foi de limitar o número de execução, uma vez que os valores da nossa lista de encontros não é infinita.

## 1.4. Continue

O continue é uma cláusula oposta ao break e quando aplicada sinaliza que o processamento deve continuar o loop pulando a iteração que caiu no caso do `continue`. Vamos olhar um exemplo:

```
turnos_encontro = ["manhã", "noite", "madrugada", "manhã", "tarde"]
for turno in turnos_encontro:
 if turno == "madrugada":
 continue

 print("Olá!")

output
Olá!
Olá!
Olá!
Olá!
```

### 1.4.1. Pass

O Pass é uma cláusula que significa nenhuma execução, isso mesmo, ele significa que não queremos definir nenhuma ação em específica, vamos ver sua estrutura:

```
for item in range(0, 10, 2):
 if item % 3 == 0:
 print(item)
 else:
 pass

output
0
6
```

Nesse exemplo nós utilizamos um for loop para iterar entre os múltiplos de 2 dentro do intervalo 0 e 10 e mandamos o algoritmo imprimir somente aqueles que também são divisíveis por 3. Para todos os outros números simplesmente passamos, nenhuma ação era necessária, e é assim que utilizamos o Pass.

Copyright © 2022 Geovanne Atanazio and Mayara Machado all right reserved , powered by GitbookArquivo revisado pela última vez em 28/05/2022

- **1. Funções**
  - **1.1. Definindo Funções**
  - **1.2. Funções com Argumentos**

# 1. Funções

Funções são blocos de códigos que executam um processamento em específico e retornam valores. São uma ótima forma de evitar duplicação de código e promover a reutilização de processamento em múltiplos lugares dos seus algoritmos.

As funções vão tornar o seu código mais legível, fácil de programar, de testar e manter. Um exemplo de uma função que já aprendemos e utilizamos bastante ao longo desse tutorial é a `print()`. No Python nós vamos ter funções chamadas de *built-in*, que significam função que são definidas pela própria linguagem e por isso são disponibilizadas por padrão em qualquer lugar que você execute um código em Python. Você também vai poder definir suas próprias funções, o que chamaremos de funções *user-defined*.

## 1.1. Definindo Funções

Para definir uma função é muito simples, todas as funções em Python iniciam com a nomenclatura `def` seguida por um nome para a função, parênteses e dois pontos (:).

```
def dizer_ola():
 """ Imprime no console uma saudação. """
 print("Hello World!")

dizer_ola()
output = Hello World!
```

Para definir os nomes das suas funções lembre-se das seguintes regras:

1. Evitar o uso de acentos
2. As letras devem estar minúsculas e as palavras devem estar separadas por underscores.

Além disso, opcionalmente as funções podem ou não retornar valores. Para retornar valores em uma função é utilizado a cláusula `return` (expressão) .

## 1.2. Funções com Argumentos

Para invocar uma função basta definir o seu nome juntamente com os parênteses e argumentos, se necessário. Algumas funções definem argumentos como necessidade para fazer a sua execução, como é o

caso do exemplo `multiplicar(x, y)` que esperar receber dois argumentos para realizar a multiplicação.

```
def multiplicar(x, y):
 """ Multiplica os valores x vezes y. """
 return x * y

resultado_multiplicacao = multiplicar(2, 3)
print(resultado_multiplicacao)
output = 6

print(multiplicar(2, 3))
output = 6

print(multiplicar(10, 2))
output = 20
```

Algumas vezes as funções que recebem argumentos podem necessitar de valores padrões. Nesse caso é garantido que a função sempre terá um valor para o argumento da função. Por exemplo:

```
def dizer_ola(nome="World"):
 """ Imprime no console uma saudação. """
 return f"Hello {nome}!"

print(dizer_ola())
output = Hello World!

print(dizer_ola("Maria"))
output = Hello Maria!

print(dizer_ola(nome="Jose"))
output = Hello Jose!
```

Alternativamente, também podemos usar as variáveis mágicas `*args` e `**kwargs` na definição de funções. Essas variáveis permitem que no momento da chamada da função nós possamos passar um número variável de argumentos. Ou seja, quando não temos uma definição exata de quais serão os argumentos que estamos passando para uma função, esses atributos nos permite flexibilizar a definição das funções.

Enquanto o `*args` significa argumentos que não são nomeados, ou seja, não acompanhem palavras chaves, os `**kwargs` seguem uma estrutura de múltiplos argumentos que seguem o padrão **chave=valor**.

```
def soma(*args):
 total = 0
 for num in args:
 total += num
 return total

print(soma(2,3,4,6)) # 15
print(soma(2,3,4,6,5,5,1,2)) # 32
print(soma(2,3)) # 5
```

```
def paises(**kwargs):
 print(kwargs)
 for pais, sigla in kwargs.items():
 print(f"A sigla do País {pais} é {sigla}.")

paises(Brasil="BR", Portugal="PT")
output:
{"Brasil": "BR", "Portugal": "PT"}
A sigla do País Brasil é BR.
A sigla do País Portugal é PT.
```

Copyright © 2022 Geovanne Atanazio and Mayara Machado all right reserved , powered by GitbookArquivo revisado pela última vez em 28/05/2022

- 1. Palavras-chaves

# 1. Palavras-chaves

Esta seção apresenta um glossário com as palavras-chaves presentes neste material. As explicações contidas aqui tem o objetivo de elucidar, rapidamente, a compreensão do leitor perante as informações trazidas.

| Termos e siglas          | Definição e significado                                                                                                                                                                                                             |
|--------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Linguagem de Programação | Linguagem formal que, através de uma série de instruções, permite que um programador escreva um conjunto de ordens, para criar programas que controlam o comportamento físico e lógico de uma máquina.                              |
| Sistema Operacional (SO) | É o conjunto de programas que fazem a interface do usuário e seus programas com o computador.                                                                                                                                       |
| Terminal                 | Também conhecido como "console" ou simplesmente "terminal virtual", é um programa que executa um <i>shell</i> — programa responsável por interpretar as instruções enviadas pelo usuário e seus programas para o SO.                |
| Extensão de Arquivos     | São os caracteres separados do nome do arquivo por um <code>.</code> , sendo úteis para localizar ou indicar como utilizar o arquivos.                                                                                              |
| Método                   | bloco de código que contém uma série de instruções.                                                                                                                                                                                 |
| Algoritmo                | Qualquer sequência finita de passos que levam à execução de uma certa tarefa ou à resolução de um problema.                                                                                                                         |
| Variável                 | Representação do espaço na memória do computador destinado a um dado, sendo alterado durante a execução do algoritmo.                                                                                                               |
| Interpretador            | Faz a conversão, em tempo real, do código escrito pelo programador para linguagem de máquina, formato compreendido pelo computador que está executando o algoritmo.                                                                 |
| Aprendizado de Máquina   | É uma tecnologia onde os computadores tem a capacidade de aprender de acordo com as respostas esperadas por meio associações de diferentes dados, os quais podem ser imagens, números e tudo que essa tecnologia possa identificar. |
| Big Data                 | É um conjunto de dados maior e mais complexo, esses conjuntos de dados são tão volumosos que o software tradicional de processamento de dados simplesmente não consegue gerenciá-los.                                               |
| Case-sensitive           | Faz distinção de palavras escritas em maiúsculas e minúsculas.                                                                                                                                                                      |

Copyright © 2022 Geovanne Atanazio and Mayara Machado all right reserved , powered by GitbookArquivo revisado pela última vez em 28/05/2022



- **1. Referências bibliográficas**
  - **1.1. Introdução**
  - **1.2. História**
    - **1.2.1. Origem do nome**
    - **1.2.2. Ruptura entre versões**
    - **1.2.3. Aplicação da Linguagem**
  - **1.3. Montagem de ambiente**
    - **1.3.1. Python em ambientes locais**
    - **1.3.2. Python em ambientes online**
  - **1.4. Primeiro algoritmo**
    - **1.4.1. Saída de dados**
    - **1.4.2. Entrada de dados**
  - **1.5. Tipos de variáveis**
    - **1.5.1. Tipos numéricos**
    - **1.5.2. String (str)**
    - **1.5.3. Boolean (bool)**
    - **1.5.4. Estrutura de dados Básica**
    - **1.5.5. Condicionais e Loops**
    - **1.5.6. Funções**

## 1. Referências bibliográficas

Esta seção apresenta o conjunto das fontes efetivamente utilizadas na construção deste manual. Cada referência dessa bibliografia está organizada pelo tópico que a mesma ajudou a construir — para facilitar uma busca mais aprofundada do conteúdo.

### 1.1. Introdução

- Python para quem está começando. **Python Brasil**. Disponível em: <https://python.org.br/introducao/>. Acesso em: 27 de mar. 2022.
- Python e orientação a objetos. **Caelum**. Disponível em: <https://www.caelum.com.br/apostila/apostila-python-orientacao-a-objetos.pdf>. Acesso em: 27 de mar. 2022.
- Abrindo seu apetite — documentação Python 3.10.4. **Python**. Disponível em: <https://docs.python.org/pt-br/3/tutorial/appetite.html>. Acesso em: 27 de mar. 2022.
- O que é uma linguagem de programação e quais os tipos existem?. **Rockcontent**. Disponível em: <https://rockcontent.com/br/blog/linguagem-de-programacao/>. Acesso em: 27 de mar. 2022.

### 1.2. História

- História do Python. **Python Brasil**. Disponível em: <https://wiki.python.org.br/HistoriaDoPython>. Acesso em: 27 de mar.

2022.

- Python: como surgiu a linguagem e o seu cenário atual. **Vulpi**. Disponível em: <https://blog.vulpi.com.br/python-como-surgiu/#:~:text=Python%20foi%20criado%20no%20final,com%20o%20sistema%20operacional%20Amoeba..> Acesso em: 27 de mar. 2022.
- Python e orientação a objetos. **Caelum**. Disponível em: <https://www.caelum.com.br/apostila/apostila-python-orientacao-a-objetos.pdf>. Acesso em: 27 de mar. 2022.

### 1.2.1. Origem do nome

- Python: A origem do nome. **Alura**. Disponível em: <https://www.alura.com.br/artigos/python-origem-do-nome>. Acesso em: 27 de mar. 2022.
- Python: como surgiu a linguagem e o seu cenário atual. **Vulpi**. Disponível em: <https://blog.vulpi.com.br/python-como-surgiu/#:~:text=Python%20foi%20criado%20no%20final,com%20o%20sistema%20operacional%20Amoeba..> Acesso em: 27 de mar. 2022.

### 1.2.2. Ruptura entre versões

- Python: como surgiu a linguagem e o seu cenário atual. **Vulpi**. Disponível em: <https://blog.vulpi.com.br/python-como-surgiu/#:~:text=Python%20foi%20criado%20no%20final,com%20o%20sistema%20operacional%20Amoeba..> Acesso em: 27 de mar. 2022.

### 1.2.3. Aplicação da Linguagem

- Qual é a linguagem de programação mais usada no mundo atualmente?. **CanalTech**. Disponível em: <https://canaltech.com.br/mercado/qual-e-a-linguagem-de-programacao-mais-usada-no-mundo-atualmente-200857/>. Acesso em: 25 de mar. 2022.
- Introdução ao Aprendizado de Máquina. **IBM**. Disponível em: <https://www.ibm.com/br-pt/analytics/machine-learning>. Acesso em: 25 de mar. 2022.
- O que é Big Data?. **Oracle**. Disponível em: <https://www.oracle.com/br/big-data/what-is-big-data/>. Acesso em: 27 de mar. 2022.

## 1.3. Montagem de ambiente

- Curso Python #03 - Instalando o Python3 e o IDLE. **Curso em Vídeo**. Disponível em: <https://youtu.be/VuKvR1J2LQE>. Acesso em:

19 de mai. 2022.

### 1.3.1. Python em ambientes locais

- Curso Python #03 - Instalando o Python3 e o IDLE. **Curso em Vídeo**. Disponível em: <https://youtu.be/VuKvR1J2LQE>. Acesso em: 19 de mai. 2022.
- Download Python. **Python**. Disponível em: <https://www.python.org/downloads/>. Acesso em: 19 de mai. 2022.

### Python no GNU/Linux e no macOS

- Curso Python #03 - Instalando o Python3 e o IDLE. **Curso em Vídeo**. Disponível em: <https://youtu.be/VuKvR1J2LQE>. Acesso em: 19 de mai. 2022.
- Download Python. **Python**. Disponível em: <https://www.python.org/downloads/>. Acesso em: 19 de mai. 2022.

### Python no Windows

- Curso Python #03 - Instalando o Python3 e o IDLE. **Curso em Vídeo**. Disponível em: <https://youtu.be/VuKvR1J2LQE>. Acesso em: 19 de mai. 2022.
- Download Python. **Python**. Disponível em: <https://www.python.org/downloads/>. Acesso em: 19 de mai. 2022.

### Interface de desenvolvimento

- Visual Studio Code. **Visual Studio Code**. Disponível em: <https://code.visualstudio.com/>. Acesso em: 19 de mai. 2022.

### 1.3.2. Python em ambientes online

- Replit. **Replit**. Disponível em: <https://replit.com/languages/python3>. Acesso em: 19 de mai. 2022.

## 1.4. Primeiro algoritmo

- Fundamentos de programação. **Programação Em C++: Algoritmos, Estruturas De Dados E Objetos**. Disponível em: <https://statics-submarino.b2w.io/sherlock/books/firstChapter/6718890.pdf>. Acesso em: 19 de mai. 2022.
- Gerenciamento de arquivos. **Manual GNU/Linux**. Disponível em: [https://geovanneatanazio.github.io/manual\\_linux/gerenciamento\\_de\\_arquivos.html](https://geovanneatanazio.github.io/manual_linux/gerenciamento_de_arquivos.html). Acesso em: 20 de mai. 2022.

### 1.4.1. Saída de dados

- O shell e o terminal. **Manual GNU/Linux**. Disponível em: [https://geovanneatanazio.github.io/manual\\_linux/o\\_shell\\_e\\_o\\_terminal.html](https://geovanneatanazio.github.io/manual_linux/o_shell_e_o_terminal.html). Acesso em: 20 de mai. 2022.
- Entrada e Saída. **O tutorial de Python**. Disponível em: <https://docs.python.org/pt-br/3/tutorial/inputoutput.html>. Acesso em: 20 de mai. 2022.

### 1.4.2. Entrada de dados

- Entrada e Saída. **O tutorial de Python**. Disponível em: <https://docs.python.org/pt-br/3/tutorial/inputoutput.html>. Acesso em: 20 de mai. 2022.
- O que são variáveis e para que elas servem na programação?. **People**. Disponível em: <https://www.people.com.br/noticias/informatica/o-que-sao-variaveis-e-para-que-elas-servem-na-programacao>. Acesso em: 20 de mai. 2022.

## 1.5. Tipos de variáveis

- Tipos de variáveis disponíveis no Python. **Python Academy**. Disponível em: <https://pythonacademy.com.br/blog/tipos-de-variaveis-no-python>. Acesso em: 24 de mai. 2022.
- Métodos de tradução: interpretador x compilador. **GeekHunter**. Disponível em: <https://blog.geekhunter.com.br/metodos-de-traducao-compiladores-ou-interpretores/>. Acesso em: 24 de mai. 2022.
- Métodos de tradução: interpretador x compilador. **iMasters**. Disponível em: <https://imasters.com.br/desenvolvimento/metodos-de-traducao-interpreteador-x-compilador>. Acesso em: 24 de mai. 2022.
- Linguagem da Máquina. **TechLib**. Disponível em: [https://techlib.wiki/definition/machine\\_language.html](https://techlib.wiki/definition/machine_language.html). Acesso em: 24 de mai. 2022.

### 1.5.1. Tipos numéricos

- Tipos de variáveis disponíveis no Python. **Python Academy**. Disponível em: <https://pythonacademy.com.br/blog/tipos-de-variaveis-no-python>. Acesso em: 24 de mai. 2022.
- Tipos de dados em Python: Numéricos. **DevMedia**. Disponível em: <https://www.devmedia.com.br/tipos-de-dados-em-python-numericos/40652>. Acesso em: 24 de mai. 2022.

### Inteiro ( `int` )

- Tipos de variáveis disponíveis no Python. **Python Academy**. Disponível em: <https://pythonacademy.com.br/blog/tipos-de-variaveis-no-python>.

[variaveis-no-python](#). Acesso em: 24 de mai. 2022.

- Tipos de dados em Python: Numéricos. **DevMedia**. Disponível em: <https://www.devmedia.com.br/tipos-de-dados-em-python-numericos/40652>. Acesso em: 24 de mai. 2022.

## Ponto flutuante ou decimal ( `float` )

- Tipos de variáveis disponíveis no Python. **Python Academy**. Disponível em: <https://pythonacademy.com.br/blog/tipos-de-variaveis-no-python>. Acesso em: 24 de mai. 2022.
- Tipos de dados em Python: Numéricos. **DevMedia**. Disponível em: <https://www.devmedia.com.br/tipos-de-dados-em-python-numericos/40652>. Acesso em: 24 de mai. 2022.

## Complexo ( `complex` )

- Tipos de variáveis disponíveis no Python. **Python Academy**. Disponível em: <https://pythonacademy.com.br/blog/tipos-de-variaveis-no-python>. Acesso em: 24 de mai. 2022.
- Tipos de dados em Python: Numéricos. **DevMedia**. Disponível em: <https://www.devmedia.com.br/tipos-de-dados-em-python-numericos/40652>. Acesso em: 24 de mai. 2022.

### 1.5.2. String ( `str` )

- Tipos de variáveis disponíveis no Python. **Python Academy**. Disponível em: <https://pythonacademy.com.br/blog/tipos-de-variaveis-no-python>. Acesso em: 24 de mai. 2022.
- Tipos de dados em Python: Numéricos. **DevMedia**. Disponível em: <https://www.devmedia.com.br/tipos-de-dados-em-python-numericos/40652>. Acesso em: 24 de mai. 2022.

### 1.5.3. Boolean ( `bool` )

- Tipos de variáveis disponíveis no Python. **Python Academy**. Disponível em: <https://pythonacademy.com.br/blog/tipos-de-variaveis-no-python>. Acesso em: 24 de mai. 2022.
- Tipos de dados em Python: Numéricos. **DevMedia**. Disponível em: <https://www.devmedia.com.br/tipos-de-dados-em-python-numericos/40652>. Acesso em: 24 de mai. 2022.

### 1.5.4. Estrutura de dados Básica

- Data Structures. **Python.org**. Disponível em: <https://docs.python.org/3/tutorial/datastructures.html>. Acesso em 25 de mar. 2022.

### 1.5.5. Condicionais e Loops

- More Control Flow Tools. **Python.org**. Disponível em: <https://docs.python.org/3/tutorial/controlflow.html>. Acesso em 25 de mar. 2022.
- Control Flow Statements. **OReilly**. Disponível em: <https://www.oreilly.com/library/view/python-in-a/0596001886/ch04s09.html>. Acesso em 25 de mar. 2022.

### 1.5.6. Funções

- Defining Your Own Python Function. **Real Python**. Disponível em: <https://realpython.com/defining-your-own-python-function/>. Acesso em 25 de mar. 2022.

Copyright © 2022 Geovanne Atanazio and Mayara Machado all right reserved , powered by GitbookArquivo revisado pela última vez em 28/05/2022