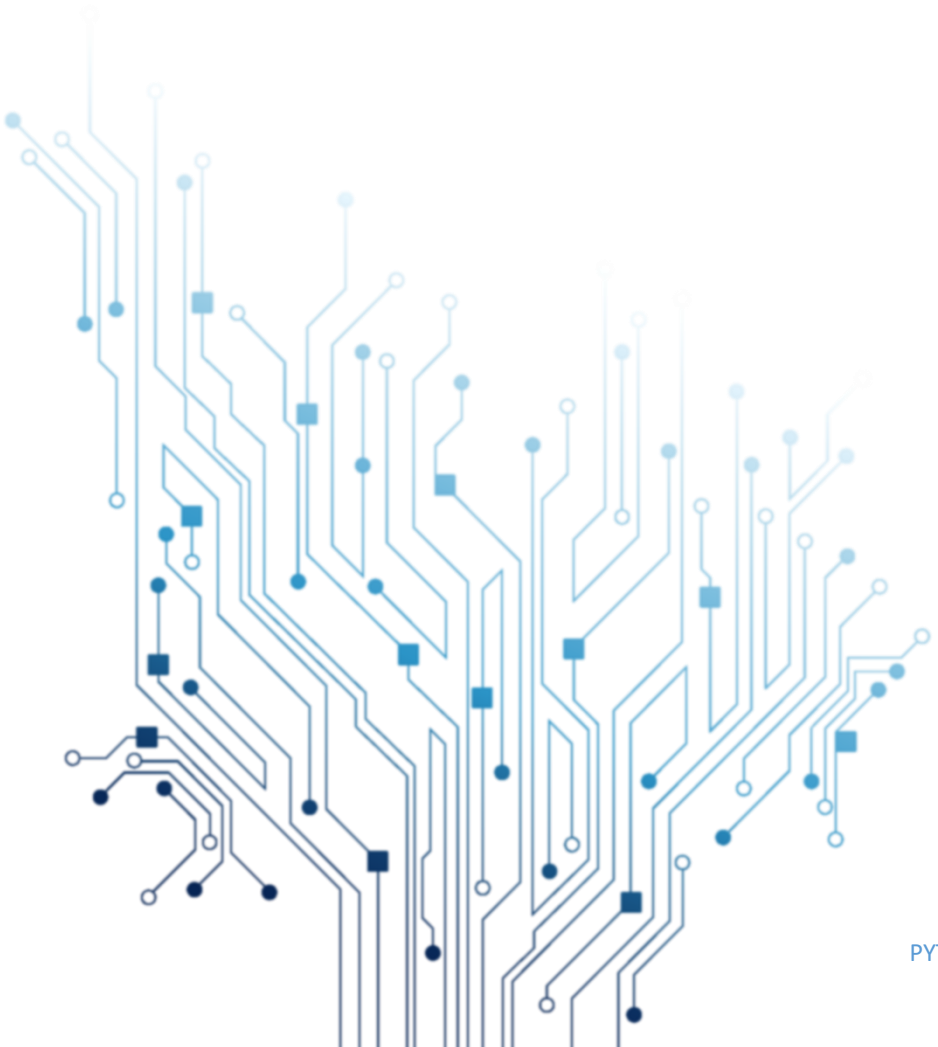
PYTHON

GRAFEN – PILOTO

V. 1.0.0

GRAFEN

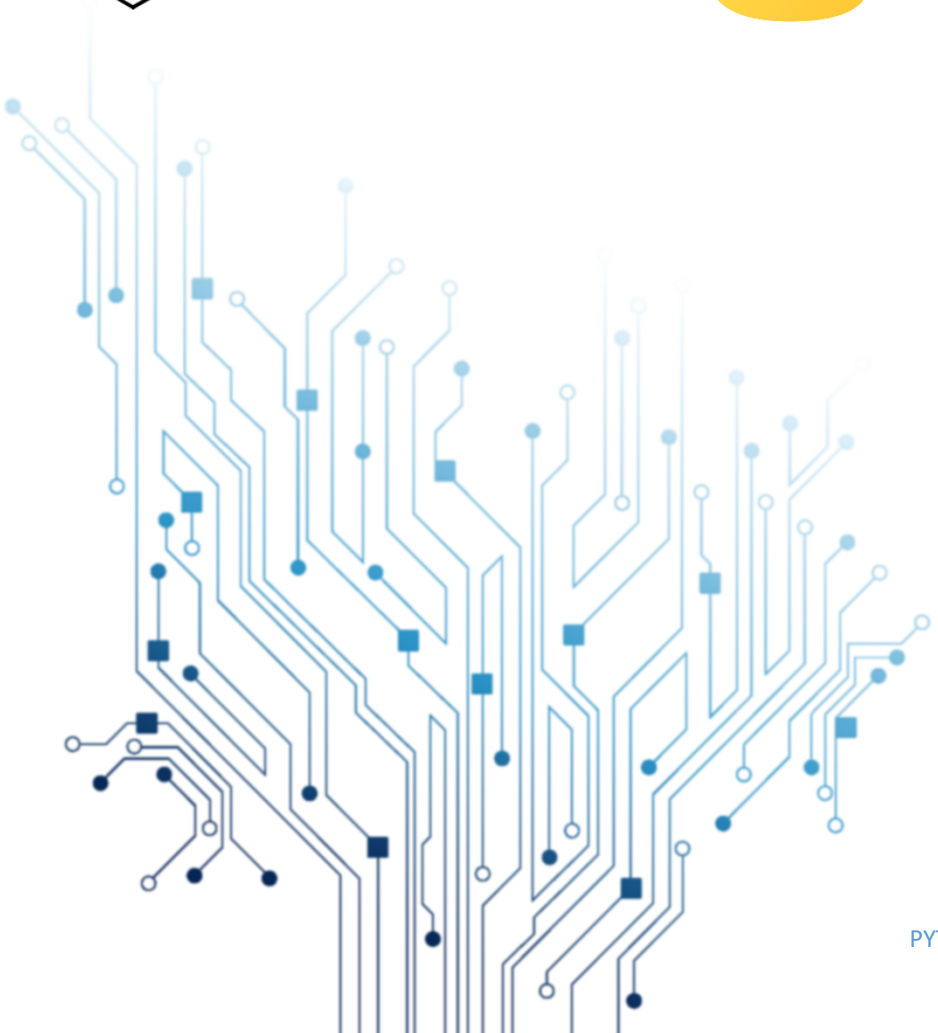
Sumário



Treinamento de Python

Olá pessoal, essa apostila é o conteúdo que conversamos em sala de aula. Não é um material complexo, mas sim um material que precisamos realmente saber, coisas importantes. Mandaremos também links com complementos, métodos mais avançados dos objetos ensinados. O foco aqui é você aprender a usar e fazer. Aprendendo essa base e a colocando em prática, os métodos mais complexos serão simples. Então aproveite e bons estudos.

Ah! Estamos disponíveis para perguntas, aproveitem isso também!



Condições

Utilizamos a palavra reservada IF para realizar condições. Falando em um português no dia a dia, seria como se fosse: “Se eu tenho dez reais, então consigo comprar um jogo”. No Python ficaria assim:

```
dinheiro = 10
if dinheiro == 10:
    print("Comprei o jogo")
```

```
Comprei o jogo
```

Também podemos ter uma condição caso a primeira condição seja falsa, utilizamos a palavra reservada ELSE. Falando em um português no dia a dia, seria como se fosse: “Se eu tenho dez reais, então consigo comprar um jogo, senão eu compro outra coisa”.

```
dinheiro = 8
if dinheiro == 10:
    print("Comprei o jogo")
else:
    print("Vou comprar outra coisa")
```

```
Vou comprar outra coisa
```

Em linguagens de programação, podemos ter ainda várias condições ao mesmo tempo, separamos essas condições pela palavra reservada ELIF. Vejamos:

```
dinheiro = 8
if dinheiro == 10:
    print("Comprei o jogo")
elif dinheiro == 9:
    print("Vou comprar outra coisa por 9 reais")
elif dinheiro == 8:
    print("Vou comprar outra coisa por 8 reais")
elif dinheiro == 7:
    print("Vou comprar outra coisa por 7 reais")
else:
    print("Vou comprar outra coisa mais barata")
```

GRAFEN

Laço de repetição

Quando precisamos percorrer dados, fazemos um loop para que todos os dados sejam lidos. Vamos ver um exemplo:

0	1	2	3	4	5	6	7	8	9

Temos aqui um objeto com dez índices, e cada índice tem um número. Queremos percorrer cada número e somar um. Para isso utilizamos a palavra reservada FOR. Vejamos:

```
lista = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
for numero in lista:
    numero += 1
    print(numero)
```

```
1
2
3
4
5
6
7
8
9
10
```

O FOR simplesmente entra em casa índice e faz a soma, o FOR só termina quando a lista termina. A palavra NUMERO que está após o FOR é um apelido que damos para cada índice da lista, então, um índice é igual a palavra NUMERO, lendo a execução seria: “Para cada NUMERO na lista, eu somo 1”. A palavra NUMERO pode ser substituída por qualquer palavra.

Que tal abrir seu Jupyter Notebook e testar? Altere os valores para fazer alguns testes ☺

Tipo de variáveis

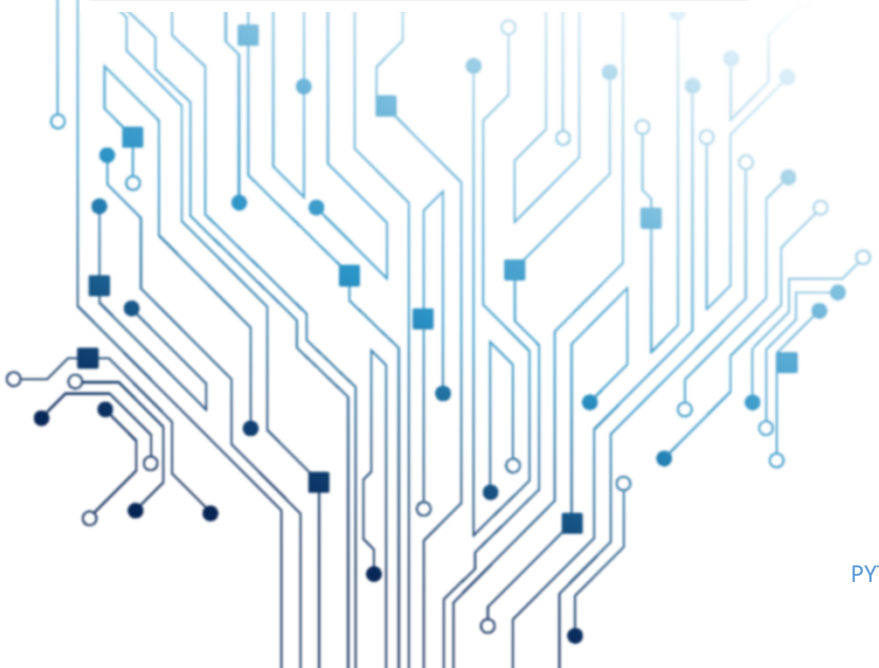
No Python temos alguns tipos de variáveis, tais como:

- Numéricos;
  - ❖ Int (Número)
  - ❖ Float (Número com ponto flutuante)
  - ❖ Long (A partir do python 3 é conhecido como Int também)
  - ❖ Bool (True ou False, que é o mesmo de 1 ou 0)
  - ❖ String (Caracteres)
- Lista;
- Dicionário;
- Tupla;
- Entre outros,

O que queremos neste material é mostrar o que realmente utilizamos mais no dia a dia!

Assim atribuímos valores às variáveis:

```
var_int = 1
var_float = 1.5
var_long = 10000000
var_bool = True
var_string = "Isso é uma String"
var_lista = []
var_dicionario = {}
var_tupla = ()
```



GRAFEN

Explicando melhor Lista e Dicionário

LISTA

Uma lista é um objeto que armazenamos valores ou outros objetos, ela é dividida por índices. O primeiro índice é o zero. Em outas linguagens, uma lista só pode ter um tipo de objeto. Por exemplo, se eu declarar ela como INT, ela aceitará só número. No Python isso já é diferente, podemos colocar qualquer valor na mesma lista já criada, seja número, String ou até mesmo outra lista. Assis declaramos uma lista:

```
#Podemos criar uma lista vazia
lista = []

#Podemos criá-la já colocando valores
lista = [1, 2, 3, 4, 5]
```

Se quisermos ver o valor armazenado em algum índice da lista, basta passar para a lista qual índice deseja visualizar (entre colchetes). No exemplo abaixo queremos ver o número 1 e o número 5. Lembrando que lista se inicia no índice ZERO, para visualizar o número 1 precisamos passar o índice ZERO. E para visualizar o número 5, precisamos passar o índice 4:

```
print(lista[0])
```

```
1
```

```
print(lista[4])
```

```
5
```

Se quisermos adicionar valores à lista, basta usar o método chamado APPEND. Abaixo adicionaremos o valor 6 na lista já criada. Vejamos:

```
lista.append(6)
```

```
print(lista)
```

```
[1, 2, 3, 4, 5, 6]
```

Se quisermos substituir algum valor na lista, basta escolher qual índice queremos substituir e passar o valor novo. Abaixo iremos substituir o número 3 por 30. Vejamos:

```
lista[2] = 30
```

```
print(lista)
```

```
[1, 2, 30, 4, 5, 6]
```

Simples! Não? :D

DICIONÁRIO

O dicionário é bem parecido com uma lista, porém as informações são armazenadas diferente. Na lista colocamos valores por índice, no dicionário é armazenado CHAVE e VALOR. Isso quer dizer que ao invés de procuramos algo no dicionário por índice, procuraremos por CHAVE. A declaração da variável Dicionário também muda. Vejamos:

```
#Podemos criar um dicionário vazio
dicionario = {}

#Podemos criar um dicionário com valores
dicionario = {"Nome":"William", "Idade":26, "Estado":"Sao Paulo"}
```

No exemplo acima criamos um dicionário com 3 chave e valor. As chaves Nome, Idade e Estado. E seus respectivos valores, William, 26 e São Paulo. Sempre precisamos colocar chave e valor juntos separados por dois pontos. E separamos cada chave-valor, por vírgula. Podemos colocar quantas chaves-valor quisermos.

Caso queiramos ver algum valor nesse dicionário, ao invés de colocar o número como na lista, colocaremos a chave que relaciona o valor que queremos, por exemplo, abaixo queremos saber qual o nome que está armazenado nesse dicionário:

```
print(dicionario["Nome"])
```

```
William
```

Para substituir um valor de qualquer chave, é bem parecido com a lista. Basta colocarmos a chave que desejamos trocar o valor e passar o valor novo. Vejamos:

```
dicionario["Nome"] = "Nemesis"
```

```
print(dicionario)
```

```
{'Idade': 26, 'Estado': 'Sao Paulo', 'Nome': 'Nemesis'}
```

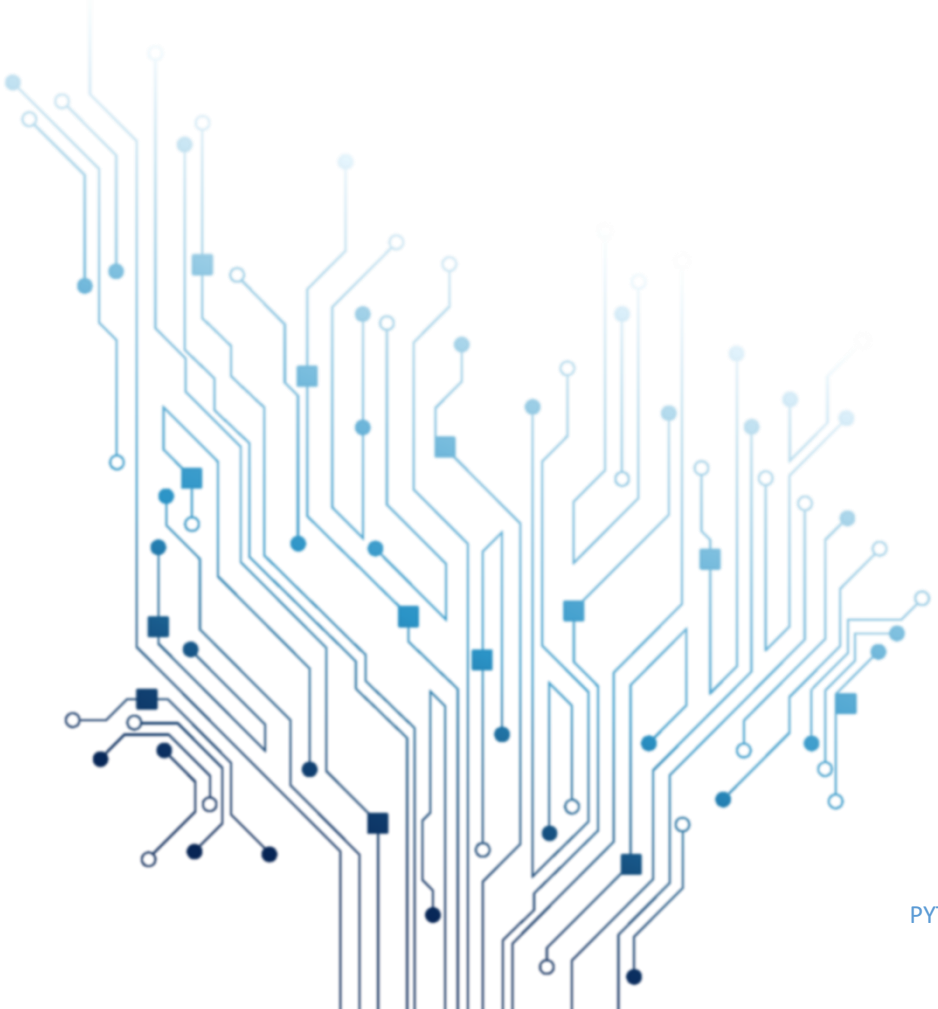
Podemos ver nesse exemplo que as chaves não estão na mesma ordem onde criamos. Isso acontece porque dicionário não se importa onde as chaves estão, pois o que importa é o nome da chave e não a ordem em que está. Contrário da lista que os valores nunca trocarão de ordem, pois a referência está por índice.

Para inserirmos mais chaves no dicionário basta colocar o novo nome da chave e inserir um valor. Abaixo estamos criando a nova chave chamada ALTURA e colocando um valor de 1.98. Vejamos:

```
dicionario["Altura"] = 1.98
```

```
print(dicionario)
```

```
{'Idade': 26, 'Altura': 1.98, 'Estado': 'Sao Paulo', 'Nome': 'Nemesis'}
```



Classes

Classes são criadores de objetos, como se fosse um molde que cria coisas. Nas classes colocamos as ações que um objeto pode realizar e as características deste objeto. Na programação as ações são chamadas de MÉTODOS e as características de PROPRIEDADES ou ATRIBUTOS. Utilizamos classes também para separar o código por funcionalidades, um modo de deixar o código mais organizado e de fácil manutenção. Abaixo mostro como se cria uma Classe:

```
class Carro:

    def __init__(self, marca, ano):
        self.Marca = marca
        self.ano = ano
```

No exemplo foi criado uma classe chamada Carro. Só que para que essa classe funcione precisamos adicionar o método chamado __init__. Esse método é obrigatório, pois o nome __init__ é uma palavra reservada do python. Esse método inicializa uma classe. Toda vez que declaramos uma classe (farei mais a baixo) a primeira coisa que o python faz é executar esse método __init__. Dentro dele colocamos dois atributos ou propriedades, marca e ano. SELF também é uma palavra reservada do python, e resumidamente quer dizer que, nesse exemplo, Marca e Ano são propriedades exclusivas de cada objeto Carro. Ou seja, se eu criar vários objetos carro, cada um vai ter uma marca e ano, e apesar de serem criados pela mesma classe, cada um vai ter ser valor marca e ano. Note também que na mesma linha onde está escrito __init__ tem o self, marca e ano. Os valores que estão dentro dos parênteses são parâmetro que o método está esperando, ou seja, quando eu for criar um carro, vou ter que passar esses parâmetros, caso contrário dará erro. Vamos ver um exemplo:

```
bmw_x1 = Carro("BMW", "2019")
```

No exemplo estamos criando um Carro que será armazenado na variável bmw_x1. Para que pudéssemos cria-lo, tivemos que passar a marca e o ano na criação. Isso acontece porque lá no método __init__ ele espera esses parâmetros. Não precisamos passar o self, pois o python passa ele escondido pra gente.

Vamos criar métodos agora;

Para criar um método usamos a palavra reservada DEF. Vamos criar um método para classe carro. Vejamos abaixo:

```
class Carro:

    def __init__(self, marca, ano):
        self.Marca = marca
        self.ano = ano

    def acelerar(self):
        print("Acelerando o carro")

    def freiar(self):
        print("Freiando o carro")
```

Criamos dois métodos, acelerar e freiar. Se criarmos outro objeto carro agora, conseguiremos chamar esses métodos, Vamos fazer:

```
camaro = Carro("chevrolet", "2019")
```

```
camaro.acelerar()
```

```
Acelerando o carro
```

```
camaro.freiar()
```

```
Freiando o carro
```

Criamos um carro numa variável chamada mercedes, e logo após executamos os métodos que a classe Carro tem. E o resultado é o print que colocamos lá dentro da Classe. Podemos ver os atributos do carro também que declaramos na criação do objeto:

```
camaro.Marca
```

```
'chevrolet'
```

```
camaro.ano
```

```
'2019'
```

IO – Input & Output

No python conseguimos criar arquivos e manipulá-los também. Conseguimos fazer tudo isso com um objeto chamado OPEN, que já faz parte do python. Assim usamos esse objeto:

```
variavel = open(<arquivo>, <modo>)
```

Simples assim. No parâmetro <arquivo> precisamos passar o diretório onde está o arquivo ou onde o arquivo será criado. No parâmetro <modo> temos que colocar o modo que queremos, aqui estão eles pré-determinados:

```
r = abrir ou criar um arquivo para somente leitura
w = abrir ou criar um arquivo somente para escrita
a = adicionar novas linhas em um arquivo que ja possui linhas
r+ = Abrir ou criar um arquivo no modo leitura e escrita
```

Vamos fazer exemplos utilizando esses modos:

```
#Utilizando o modo de leitura <r>
arquivo = open("C:\\Users\\boliveih\\Documents\\teste\\file.txt", "r")
```

O arquivo deverá existir no diretório para que possa ser lido. Caso contrário dará erro. Temos os métodos que podemos usar no arquivo criado. Aqui estão eles:

```
arquivo.read()
```

```
'testando o modo r\nlendo o arquivo'
```

```
arquivo.readline()
```

```
'lendo o arquivo'
```

```
arquivo.readlines()
```

```
['testando o modo r\n', 'lendo o arquivo']
```

O método READ nos retorna todo o conteúdo do arquivo.

O método READLINE retorna linha por linha do arquivo, para visualizar a próxima linha devemos executar de novo o método READLINE.

O método READLINES nos retorna todas as linhas em formato de lista. Cada linha no arquivo vira um índice na lista.

```
#Utilizando o modo de escrita <w>
arquivo = open("C:\\Users\\boliveih\\Documents\\teste\\file.txt", "w")
```

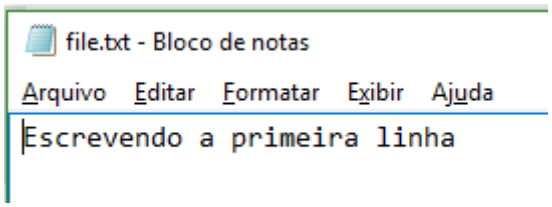
ATENÇÃO neste ponto. Quando executar este comando com o modo <W> todos os dados do arquivo serão apagados. Utilize o modo <W> somente para criar arquivos do zero.

Após criar o arquivo com o modo <W>, vamos colocar linhas dentro dele:

```
arquivo.write("Escrevendo a primeira linha")
```

Se você verificar o arquivo criado, não terá nada dentro dele ainda. Isso acontece porque estamos criando esta linha na memória ainda. Só será escrito no arquivo quando dermos o comando CLOSE. Vejamos:

```
arquivo.close()
```



Agora sim. É importante sempre utilizar o comando CLOSE, independentemente do modo que esteja usando.

```
#Utilizando o modo append <a>
arquivo = open("C:\\Users\\boliveih\\Documents\\teste\\file.txt", "a")
```

```
arquivo.write("\nEscrevendo a segunda linha")
```

```
arquivo.close()
```

Aqui estamos inserindo uma linha nova no arquivo, sem apagar o que já havia no arquivo.

Utilizamos o \n para quebrarmos uma linha.

GRAFEN

Contato

WILLIAM HELENO

Linkedin: in/williamheleno/

DOUGLAS LEAL

Linkedin: Douglas Leal

BRUNNO RAMOS

Linkedin: in/brunnolramos/

grafen.contact@gmail.com