

# Testando o Jupyter Notebook.

## Capítulo 2

### Tomando Nota...

Comentários coloca-se entre 3 aspas duplas """...""" ou com um # no inicio, quando o formato é code. Contudo, quando é Markdown, só basta digitar, utilizando # apenas para titulos.

Quando você coloca esse código: [Testando o Jupyter Notebook.](#), que é html, tornasse possível a alteração da cor e quando ele se encontra após # vira título, onde a quantidade de asteriscos define a hierarquia e o tamanho da letra.

Isso se tratando do formato Markdown

Se quiser executar o código, tecle no sinal de play ao lado da célula, o mesmo também se encontra no menu, acompanhado com o nome Run. Outra maneira de executar o comando é apertando em shift e depois em enter.

Para limpar os resultados aperte em Kernel e logo após em Restart & Clear Output; Se você quiser executar tudo de uma vez aperte em Kernel e logo após em Restart & Run all

Adicionar células = O + no canto esquerdo acrescenta outra célula, ou seja, espaços para código, após a célula selecionada.

Excluir células = Seleccione uma célula e clique na tesoura.

Mover células = Seleccione uma e clique na setinha para cima ou para baixo.

Salvar alterações = Para salvar clique no disco de salvar.

Realizar downloads = Aperta-se em file > download as > clica em um formato

Renomear = Aperta-se o nome que se encontra ao lado do nome jupyter, escreva o nome desejado e aperte enter.

### 2.1 Números e Operações Matemáticas

#### Operações:

In [1]:

```
# Soma  
4 + 4
```

Out[1]:

8

In [2]:

```
# Subtração  
4 - 3
```

Out[2]:

1

In [3]:

```
# Multiplicação  
3 * 3
```

Out[3]: 9

In [4]:  
# Divisão  
3 / 2

Out[4]: 1.5

In [5]:  
# Potência  
4 \*\* 2

Out[5]: 16

In [6]:  
# Módulo (é o resto da divisão)  
10 % 3

Out[6]: 1

## Função Type:

In [7]:  
#Verifica-se o tipo do parâmetro  
type(5)

Out[7]: int

In [8]: type(5.0)

Out[8]: float

In [9]:  
a = 'Eu sou uma string' # O igual sozinho é um simbolo de atribuição  
type(a)

Out[9]: str

## Operações com números float(fracionarios):

In [10]: 3.1 + 6.4

Out[10]: 9.5

In [11]: 4 + 4.0

Out[11]: 8.0

In [12]: 4 + 4

Out[12]: 8

In [13]: # Resultado é um número float(esse é o padrão do resultado da divisão de 2 números i

4 / 2

Out[13]: 2.0

In [14]: # Resultado é um número inteiro(se quiser mudar o padrão adicionasse mais uma barra)  
4 // 2

Out[14]: 2

In [15]: 4 / 3.0

Out[15]: 1.3333333333333333

In [16]: # Ainda retorna um número float porque a divisão foi por um número float  
4 // 3.0

Out[16]: 1.0

## Converter:

In [17]: #Converte o número inteiro em float  
float(9)

Out[17]: 9.0

In [18]: #Converte o número fracionário em inteiro  
int(6.0)

Out[18]: 6

In [19]: #Ele não arredonda ele trunca. Ele está apenas coletando a parte inteira desse valor  
int(6.5)

Out[19]: 6

## Hexadecimal e binário

In [20]: hex(394)

Out[20]: '0x18a'

In [21]: hex(217)

Out[21]: '0xd9'

In [22]: bin(286)

Out[22]: '0b100011110'

In [23]: `bin(390)`

Out[23]: `'0b110000110'`

## Funções abs, round e pow

In [24]: `# Retorna o valor absoluto  
abs(-8)`

Out[24]: `8`

In [25]: `# Retorna o valor absoluto  
abs(8)`

Out[25]: `8`

In [26]: `# Retorna o valor com arredondamento (2 é o valor de casa decimais após a vírgula)  
round(3.14151922,2)`

Out[26]: `3.14`

In [27]: `# Potência (4 elevado a 2)  
pow(4,2)`

Out[27]: `16`

In [28]: `# Potência  
pow(5,3)`

Out[28]: `125`

## 2.2 Variáveis e Operadores

### Tomando nota...

Variáveis são espaços em memoria que armazenam valores. Como, por exemplo, `b = 10`. O sinal de `=` atribui o valor à direita à variável do lado esquerdo.

In [2]: `# Atribuindo o valor 1 à variável var_teste  
var_teste = 1`

A função `print()` em python irá imprimir valores na tela. Por exemplo: `print(10)` imprime na tela o valor 10. Já `print(b)`, imprime na tela o valor da variável `b`, que nesse caso é 10. No caso da variável, também é possível imprimir na tela sem utilizar a função `print`, colocando apenas o nome da variável, sendo só é possível fazer isso se ela foi previamente definida.

In [3]: `# Imprimindo o valor da variável  
print(var_teste)`

1

In [4]: `# Imprimindo o valor da variável  
var_teste`

Out[4]: 1

In [5]: `# Não podemos utilizar uma variável que não foi definida. Veja a mensagem de erro.  
my_var`

```
NameError Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_2288/761553587.py in <module>
      1 # Não podemos utilizar uma variável que não foi definida. Veja a mensagem de
      2 erro.
----> 2 my_var
```

**NameError:** name 'my\_var' is not defined

Você pode sobreescrver a variável com um novo valor sempre que quiser. A variável assumirá um valor novo.

In [7]: `var_teste = 2`

In [8]: `var_teste`

Out[8]: 2

Também pode-se descobrir o tipo da variável utilizando a função type:

In [9]: `type(var_teste)`

Out[9]: int

In [10]: `var_teste = 9.5`

In [11]: `type(var_teste)`

Out[11]: float

O nome da variável não precisa ser uma palavra pode ser apenas uma letra.

In [12]: `x = 1`

In [13]: `x`

Out[13]: 1

## Declaração multipla

In [14]: `#Atribuindo valores diferentes para cada variável simultaneamente  
pessoa1, pessoa2, pessoa3 = "Maria", "José", "Tobias"`

In [15]: pessoa1

Out[15]: 'Maria'

In [16]: pessoa2

Out[16]: 'José'

In [17]: pessoa3

Out[17]: 'Tobias'

In [18]: #Atribuindo um mesmo valor para variaveis diferentes simultaneamente  
fruta1 = fruta2 = fruta3 = "Laranja"

In [19]: fruta1

Out[19]: 'Laranja'

In [20]: fruta2

Out[20]: 'Laranja'

In [21]: fruta3

Out[21]: 'Laranja'

In [22]: # Fique atento!!! Python é case-sensitive. Criamos a variável fruta2, mas não a variável Fruta2.

---

```
NameError                                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_2288\1565389715.py in <module>
      1 # Fique atento!!! Python é case-sensitive. Criamos a variável fruta2, mas não
      2 # Letras maiúsculas e minúsculas tem diferença no nome da variável.
      3 Fruta2
----> 3 Fruta2
```

**NameError:** name 'Fruta2' is not defined

## Regras de nomeclatura

Existem algumas regras que devem ser seguidas ao definir nomes de variáveis:

1. Os nomes de variáveis não devem começar com número;
2. Não pode haver espaços no nome; Utilize \_ em vez disso.
3. Não é possível usar qualquer um desses simbolos "", < > \ | / () @ # % ^ & \* ~ - + ! \$
4. Não se pode usar palavras reservadas como nome de variável. Palavras como:

false; class; finally; is; return; none; continue; for; lambda; try; true; def; from; nonlocal; while; and; del; global; not; with; in; as; elif; if; or; yield; assert; else; import; pass; break; raise.

Palavras reservadas são palavras utilizadas pelo interpretador do python para reconhecer que aquilo é um comando python.

In [24]: *# Pode-se usar Letras, números e underline (mas não se pode começar com números)*  
x1 = 50

In [25]: x1

Out[25]: 50

In [27]: *# Mensagem de erro (sintaxe invalida), pois o Python não permite nomes de variáveis*  
1x = 50

```
File "C:\Users\carla\AppData\Local\Temp\ipykernel_2288/2793364712.py", line 2
 1x = 50
 ^
SyntaxError: invalid syntax
```

In [28]: *# Não podemos usar palavras reservadas como nome de variável*  
break = 1

```
File "C:\Users\carla\AppData\Local\Temp\ipykernel_2288/2670791942.py", line 2
  break = 1
 ^
SyntaxError: invalid syntax
```

## Equações com variáveis

Variáveis atribuídas a outras variáveis e ordem dos operadores

In [31]: largura = 2

In [32]: altura = 4

In [39]: *#Pode-se utilizar variaveis em operações matemáticas*  
area = largura \* altura

In [40]: area

Out[40]: 8

In [44]: *#Pode-se utilizar operações matemáticas entre valores numéricos e variaveis*  
perimetro = 2 \* largura + 2 \* altura

In [45]: perimetro

Out[45]: 12

```
In [46]: # A ordem dos operadores é a mesma seguida na Matemática
perimetro = 2 * (largura + 2) * altura
```

```
In [47]: perimetro
```

```
Out[47]: 32
```

Adicionando uma imagem. Tentando, pelo menos....



## Concatenação de variáveis

```
In [51]: nome = "Steve"
```

```
In [52]: sobrenome = "Jobs"
```

```
In [53]: # O sinal de mais nesta equação não representa soma, mas sim uma concatenação, ou seja
fullName = nome + " " + sobrenome
```

```
In [54]: fullName
```

```
Out[54]: 'Steve Jobs'
```

## Operadores

Existem 4 tipos de operadores:

1. Operadores aritméticos;
2. Operadores relacionais;
3. Operadores de atribuição;
4. Operadores lógicos.

Os **operadores aritméticos**, já vimos anteriormente, são utilizados para operações matemáticas.

Foram utilizados no tópico operações com variáveis e no capítulo anterior.

```
In [55]: idade1 = 25
idade2 = 35
```

```
In [56]: # Soma
idade1 + idade2
```

```
Out[56]: 60
```

```
In [57]: # Subtração
idade2 - idade1
```

```
Out[57]: 10
```

In [58]: `# Multiplicação  
idade2 * idade1`

Out[58]: 875

In [59]: `# Divisão  
idade2 / idade1`

Out[59]: 1.4

In [60]: `# Módulo (resto)  
idade2 % idade1`

Out[60]: 10

In [61]: `# Potência  
idade1**2`

Out[61]: 625

Os **operadores relacionais** são utilizados para fazer comparações, entre eles se encontram: == != > < >= <=

In [62]: `Idade_pedro=7  
Idade_joaozinho=14  
Idade_laura=7`

In [63]: `#Igualdade / equivalencia ==  
Idade_pedro==Idade_joaozinho`

Out[63]: False

In [64]: `# Desigualdade / Inequivalencia !=  
Idade_pedro!=Idade_joaozinho`

Out[64]: True

In [65]: `# Maior que >  
Idade_pedro>Idade_joaozinho`

Out[65]: False

In [66]: `# Menor que <  
Idade_pedro<Idade_joaozinho`

Out[66]: True

In [67]: `# Maior que ou igual a >=  
Idade_joaozinho>=Idade_laura>=Idade_pedro`

True

Out[67]:

```
In [68]: # Menor que ou igual a <=
Idade_pedro<=Idade_laura<=Idade_joaozinho
```

Out[68]: True

Os **operadores de atribuição** possibilitam a execução, em uma unica instrução, de duas operações de forma simultânea.

```
In [77]: # Atribuição
z=10
```

```
In [78]: # Soma
# z=z+10
z+=10; z
```

Out[78]: 20

```
In [79]: # Subtração
# z=z-10
z-=10; z
```

Out[79]: 10

```
In [80]: # Multiplicação
# z=z*10
z*=10; z
```

Out[80]: 100

```
In [81]: # Divisão (padrão=float)
# z=z/10
z/=10; z
```

Out[81]: 10.0

```
In [82]: # Módulo (Divide e pega o resto. Como z estava = a 10, então, 10/10=0)
# z=z%10
z%=10; z
```

Out[82]: 0.0

```
In [91]: z=10
```

```
In [92]: # Potência
# z=z**10
z**=10; z
```

Out[92]: 10000000000

```
In [93]: # Divisão inteira(se o z for float o resultado será float)
# z=z//10
z//=10; z
```

```
Out[93]: 1000000000
```

Os **operadores lógicos** podem ser utilizados para checar duas condições (concatena-se 2 condições).

```
In [ ]: # Se ambos os operadores forem true, retorna-se true. (and) = e
# (x and y) é true
```

```
In [ ]: # Se um dos operadores for true, retorna-se true. (or) = ou
# (x or y) é true
```

```
In [ ]: # Usado para reverter o estado da Logica. (not) = não
# Se não for x e y ao mesmo tempo é falso.
# Not(x and y) é falso
```

## 2.3 Strings e Indexação em Python

### Strings

Strings são usados em Python para gravar informações em formato de texto, como nomes por exemplo.

Strings em Python são na verdade uma sequência de caracteres, o que significa basicamente, que Python mantem o controle de cada elemento de sequência.

Python entende a string "Olá" como sendo uma sequencia de letras em uma ordem especifica. Isso significa que você será capaz de usar a indexação para obter um caracter específico, (como a primeira letra ou a ultima letra).

Strings é uma sequencia **imutável** de caracteres ou apenas um caracter.

### Criando uma String

Para criar uma string em Python você pode usar aspas simples ou duplas. Desta forma você pode imprimir direto na saída, sem precisar da função print(). Por exemplo:

```
In [94]: # Uma única palavra
'Oi'
```

```
Out[94]: 'Oi'
```

```
In [95]: # Uma frase
'Criando uma string em Python'
```

```
Out[95]: 'Criando uma string em Python'
```

```
In [96]: # Podemos usar aspas duplas
"Podemos usar aspas duplas ou simples para strings em Python"
```

```
Out[96]: 'Podemos usar aspas duplas ou simples para strings em Python'
```

```
In [97]: # Você pode combinar aspas duplas e simples
        "Testando strings em 'Python'"
```

```
Out[97]: "Testando strings em 'Python'"
```

## Imprimindo uma string

```
In [1]: print ('Testando Strings em Python')
```

```
Testando Strings em Python
```

```
In [2]: #Imprime as strings separadas por uma quebra de Linha # O \n É a tecla enter do computador
        print ('Testando \nStrings \nem \nPython')
```

```
Testando
Strings
em
Python
```

```
In [3]: # Pula uma Linha
        print ('\n')
```

## Indexando strings

Já sabemos que strings são uma sequencia. Isso significa que Python pode usar indices para chamar partes da sequencia. Vamos aprender como funciona.

Em Python, usamos colchetes [] para representar o indice de um objeto. Em Phyton, a indexação começa por 0. No R começa-se por 1.

```
In [4]: # Atribuindo uma string
        s = 'Data Science Academy'
```

```
In [5]: print(s)
```

```
Data Science Academy
```

```
In [6]: # Primeiro elemento da string.
        s[0]
```

```
Out[6]: 'D'
```

```
In [7]: s[1]
```

```
Out[7]: 'a'
```

```
In [8]: s[4]
```

Out[8]:

Podemos usar um : para executar um slicing(fatiamento) que faz a leitura de tudo até um ponto designado. Por exemplo:

In [10]:

```
# Retorna todos os elementos da string, começando pela posição (Lembre-se que Python
# até o fim da string.
s[1:]
```

Out[10]:

'ata Science Academy'

In [11]:

```
# A string original permanece inalterada #Imutabilidade
s
```

Out[11]:

'Data Science Academy'

In [12]:

```
# Retorna tudo até a posição 3 # <3 #É exclusivo (e não inclusivo), pois não inclui
s[:3]
```

Out[12]:

'Dat'

In [13]:

```
#Retorna tudo após a posição informada e inclui o valor.
s[-3:]
```

Out[13]:

'emy'

In [14]:

```
# Retorna tudo
s[:]
```

Out[14]:

'Data Science Academy'

In [15]:

```
# Nós também podemos usar a indexação negativa e ler de trás para frente.
s[-1]
```

Out[15]:

'y'

In [16]:

```
# Retornar tudo, exceto a última letra
s[:-1]
```

Out[16]:

'Data Science Academ'

Nós também podemos usar a notação de índice e fatiar a string em pedaços específicos (o padrão é 1). Por exemplo, podemos usar dois pontos duas vezes em uma linha e, em seguida, um número que especifica a frequência para retornar elementos. Por exemplo:

In [17]:

```
#Retorna todos os caracteres saltando uma posição(no caso tudo)
s[::-1]
```

Out[17]:

'Data Science Academy'

In [18]:

```
#Retorna todos os caracteres saltando duas posições
```

```
s[::-2]
```

```
Out[18]: 'Dt cec cdm'
```

```
In [19]: # Retorna todos os caracteres saltando uma posição de tras para frente
s[::-1]
```

```
Out[19]: 'ymedacA ecneicS ataD'
```

```
In [20]: # Retorna todos os caracteres satando duas posições de tras para frente
s[::-2]
```

```
Out[20]: 'yeaAeniSaa'
```

## Propriedades de strings

É importante ressaltar que as strings têm uma importante propriedade conhecida como imutabilidade. Isso significa que uma vez criada uma string, os elementos dentro dela não podem ser substituídos ou alterados.

```
In [21]: s
```

```
Out[21]: 'Data Science Academy'
```

```
In [22]: # Alterando um caracter #Propriedade da imutabilidade # Não é possivel atribuir um item dentro de uma string
s[0] = 'x'
```

---

```
TypeError Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_12696\4172503925.py in <module>
      1 # Alterando um caracter #Propriedade da imutabilidade # Não é possivel atribuir um item dentro de uma string
----> 2 s[0] = 'x'
```

**TypeError:** 'str' object does not support item assignment

```
In [23]:
```

```
# Concatenando strings (não muda s, apenas une neste resultado)
```

```
s + ' é a melhor maneira de estar preparado para o mercado de trabalho em Ciência de
```

```
Out[23]: 'Data Science Academy é a melhor maneira de estar preparado para o mercado de trabalho em Ciência de Dados!'
```

```
In [24]:
```

```
# Pode-se fazer uma concatenação e gravar na string(na propria variavel), ou seja, v
s = s + ' é a melhor maneira de estar preparado para o mercado de trabalho em Ciênci
```

```
In [25]:
```

```
print(s)
```

Data Science Academy é a melhor maneira de estar preparado para o mercado de trabalho em Ciência de Dados!

```
In [26]:
```

```
# Podemos usar o símbolo de multiplicação para criar repetição!
letra = 'w'
```

In [27]: `letra * 3`

Out[27]: `'www'`

## Funções Built-in de Strings

Python é uma linguagem orientada a objeto, sendo assim, as estruturas dos dados possuem atributos e métodos(rotinas associadas as propriedades), tanto os atributos como os métodos são acessados usando `(.)`.

Os métodos estão sob a forma:

1. `objeto.atributo`
2. `objeto.metodo()`
3. `objeto.metodo(parâmetros)`

Os parâmetros são argumentos extras, que podemos passar para o metodo.

In [28]: `s`

Out[28]: `'Data Science Academy é a melhor maneira de estar preparado para o mercado de trabalho em Ciência de Dados!'`

In [29]:

```
# Tecle s coloque um ponto e aperte na tecla tab.  
# Mostra-se uma lista de metodos e atributos que estão disponíveis para esse objeto.  
# Cada objeto em python dependendo do tipo de objeto, vai ter um conjunto de atributos  
# Se for um metodo adicione ()
```

In [30]:

```
# Uper case # Converte para maiusculo.  
s.upper()
```

Out[30]:

`'DATA SCIENCE ACADEMY É A MELHOR MANEIRA DE ESTAR PREPARADO PARA O MERCADO DE TRABALHO EM CIÊNCIA DE DADOS!'`

In [31]:

```
# Lower case # Converte para minusculo.  
s.lower()
```

Out[31]:

`'data science academy é a melhor maneira de estar preparado para o mercado de trabalho em ciência de dados!'`

In [32]:

```
# Quando não se passa nenhum parametro você faz a divisão por espaços em branco que  
s.split()
```

Out[32]:

```
['Data',  
 'Science',  
 'Academy',  
 'é',  
 'a',  
 'melhor',  
 'maneira',  
 'de',  
 'estar',  
 'preparado',  
 'para',  
 'o',  
 'mercado',  
 'de',  
 ',']
```

```
'trabalho',
'em',
'Ciéncia',
'de',
'Dados!']
```

In [33]:

```
# Dividir uma string por um elemento específico
s.split('y')
```

Out[33]:

```
['Data Science Academ',
 ' é a melhor maneira de estar preparado para o mercado de trabalho em Ciéncia de Da
dos!']
```

## Funções String

In [34]:

```
s = 'seja bem vindo ao universo de python'
```

In [35]:

```
# Transforma o primeiro caracter em maiusculo.
s.capitalize()
```

Out[35]:

```
'Seja bem vindo ao universo de python'
```

In [36]:

```
# Conta quantas vezes o caracter que se encontra entre ('') apareceu
s.count('a')
```

Out[36]:

```
2
```

In [37]:

```
# Mostra onde(Em qual posição) o caracter descrito entre ('') apareceu
s.find('p')
```

Out[37]:

```
30
```

In [38]:

```
s.center(20, 'z')
```

Out[38]:

```
'seja bem vindo ao universo de python'
```

In [39]:

```
s.isalnum()
```

Out[39]:

```
False
```

In [40]:

```
s.isalpha()
```

Out[40]:

```
False
```

In [41]:

```
# Verifica-se se a string tem caracter minusculo
s.islower()
```

Out[41]:

```
True
```

In [42]:

```
# Verifica se existe apenas espaço na string
s.isspace()
```

Out[42]: False

In [43]: `# Verifica se a string termina com a letra decrita entre ('')`  
`s.endswith('o')`

Out[43]: False

In [44]: `s.partition('!')`

Out[44]: ('seja bem vindo ao universo de python', '', '')

## Comparando Strings

In [45]: `# A string Python é igual a string R?`  
`print("Python" == "R")`

False

In [46]: `# A string Python é igual a string Python?`  
`print("Python" == "Python")`

True

## 2.4 Estruturas de dados - Listas

### Tomando nota...

Quando discutimos Strings introduzimos o conceito de uma sequencia em Python. As listas podem ser consideradas a versão geral de uma sequencia em Python. Ao contrario de Strings **as listas são mutaveis**, ou seja, os elementos dentro de uma lista podem ser alterados.

As listas são construidas com o uso de colchetes [] e vírgulas separando cada elemento da lista.

```
Lista = [item1, item2, ...,
         itemz]
```

Se você estiver familiarizado com outra linguagem de programação, você pode traçar parametros entre matrizes em outras linguagens e listas em Python. Listas em Python, no entanto, tendem a ser mais flexiveis do que as matrizes em outras linguagens por dois bons motivos:

1. As listas não tem tamanho fixo( o que significa que não precisamos especificar quão grande uma lista será)
2. Listas não tem restrições de tipo fixo.

Uma grande caracteristica de estruturas de Phyton é que elas suportam alinhamento. Isto singnifica que podemos usar estruturas de dados dentro de estruturas de dados, ou seja, podemos criar uma lista dentro de outra lista, isto se chama lista alinhada.

In [1]: `# Criando uma Lista (Lista com um unico elemento, uma única string)`  
`listadomercado = ["ovos, farinha, leite, maças"]`

In [2]:

```
# Imprimindo a Lista
print(listadomercado)
```

```
['ovos, farinha, leite, maças']
```

In [3]:

```
# Criando outra Lista (Lista com 4 elementos)
listadomercado2 = ["ovos", "farinha", "leite", "maças"]
```

In [4]:

```
# Imprimindo a Lista
print(listadomercado2)
```

```
['ovos', 'farinha', 'leite', 'maças']
```

Diferença entre as duas listas:

In [6]:

```
listadomercado[0]
```

```
'ovos, farinha, leite, maças'
```

In [7]:

```
listadomercado2[0]
```

```
'ovos'
```

In [10]:

```
print("\n")
```

In [11]:

```
# Criando Lista (uma Lista não tem restrições de tipo fixo)
lista3 = [12, 100, "Universidade"]
```

In [12]:

```
# Imprimindo
print(lista3)
```

```
[12, 100, 'Universidade']
```

In [15]:

```
# Atribuindo cada valor da Lista a uma variável. Utilizando indexação.
item1 = lista3[0]
item2 = lista3[1]
item3 = lista3[2]
```

In [16]:

```
# Imprimindo as variáveis de uma só vez.
print(item1, item2, item3)
```

```
12 100 Universidade
```

## Atualizando um item da lista

In [17]:

```
# Imprimindo um item da Lista
listadomercado2[2]
```

```
'leite'
```

```
In [18]: # Atualizando um item da Lista (provando a mutabilidade)
listadomercado2[2] = "chocolate"
```

```
In [19]: # Imprimindo Lista alterada
listadomercado2
```

```
Out[19]: ['ovos', 'farinha', 'chocolate', 'maças']
```

## Deletando um item da lista

```
In [20]: # Out of index. Não é possível deletar um item que não existe na lista (o indice com
del listadomercado2[4]
```

```
-----  
IndexError Traceback (most recent call last)  
~\AppData\Local\Temp\ipykernel_1908\1216354677.py in <module>  
    1 # Out of index. Não é possível deletar um item que não existe na lista (o indice começa de 0)  
----> 2 del listadomercado2[4]
```

```
IndexError: list assignment index out of range
```

```
In [21]: # Deletando um item específico da lista
del listadomercado2[3]
```

```
In [22]: # Imprimindo o item com a Lista alterada
listadomercado2
```

```
Out[22]: ['ovos', 'farinha', 'chocolate']
```

## Listas de listas (Listas aninhadas)

Listas de listas são matrizes em Python

```
In [23]: # Criando uma Lista de Listas(forma de representar uma matriz), onde cada elemento é
listas = [[1,2,3], [10,15,14], [10.1,8.7,2.3]]
```

```
In [24]: # Imprimindo a Lista
listas
```

```
Out[24]: [[1, 2, 3], [10, 15, 14], [10.1, 8.7, 2.3]]
```

```
In [25]: # Atribuindo um item da Lista a uma variável (como é uma Lista aninhada, um item da
a = listas[0]
```

```
In [26]: a
```

```
Out[26]: [1, 2, 3]
```

```
In [27]: # Atribuindo um elemento da variável a para a variavel b
```

```
b = a[0]
```

```
In [28]: b
```

```
Out[28]: 1
```

```
In [29]: # Atribuindo o item 1(a Lista 2) da Lista a variavel list1.  
list1 = listas[1]
```

```
In [30]: list1
```

```
Out[30]: [10, 15, 14]
```

```
In [31]: # Atribuindo o elemento 1 da variável list1 a variavel valor_1_0.  
valor_1_0 = list1[0]
```

```
In [32]: valor_1_0
```

```
Out[32]: 10
```

```
In [33]: # Atribuindo o elemento 3 da variável list1 a variavel valor_1_2.  
valor_1_2 = list1[2]
```

```
In [34]: valor_1_2
```

```
Out[34]: 14
```

```
In [35]: # Atribuindo o item 3(a Lista 3) da Lista a variavel list2.  
list2 = listas[2]
```

```
In [36]: list2
```

```
Out[36]: [10.1, 8.7, 2.3]
```

```
In [37]: # Atribuindo o elemento 1 da variável list2 a variavel valor_2_0.  
valor_2_0 = list2[0]
```

```
In [38]: valor_2_0
```

```
Out[38]: 10.1
```

## Operações com listas

```
In [39]: # Criando uma Lista aninhada (Lista de Listas, ou seja, uma matriz)  
listas = [[1,2,3], [10,15,14], [10.1,8.7,2.3]]
```

In [40]: listas

Out[40]: [[1, 2, 3], [10, 15, 14], [10.1, 8.7, 2.3]]

In [41]: *# Atribuindo à variável a, o primeiro valor da primeira lista  
# Símbolo de fatiamento dois colchetes com índices.*  
a = listas[0][0]

In [42]: a

Out[42]: 1

In [43]: *# Atribuindo à variável b, o terceiro valor da segunda lista*  
b = listas[1][2]

In [44]: b

Out[44]: 14

In [45]: *#Pode-se somar um valor ao elemento adquirido da lista e atribuir o resultado a uma*  
c = listas[0][2] + 10

In [46]: c

Out[46]: 13

In [47]: d = 10; d

Out[47]: 10

In [48]: *#Também é possível multiplicar o elemento adquirido da lista a uma variável e atribuir o resultado a uma*  
e = d \* listas[2][0]

In [49]: e

Out[49]: 101.0

## Concatenando listas

In [50]: lista\_s1 = [34, 32, 56]

In [51]: lista\_s1

Out[51]: [34, 32, 56]

In [52]: lista\_s2 = [21, 90, 51]

In [53]: lista\_s2

Out[53]: [21, 90, 51]

In [54]: # Concatenando Listas  
lista\_total = lista\_s1 + lista\_s2

In [55]: lista\_total

Out[55]: [34, 32, 56, 21, 90, 51]

## Operador in

In [56]: # Criando uma lista  
lista\_teste\_op = [100, 2, -5, 3.4]

In [57]: # Verificando se o valor 10 pertence a lista  
print(10 in lista\_teste\_op)

False

In [58]: # O print é só para imprimir  
10 in lista\_teste\_op

Out[58]: False

In [59]: # Verificando se o valor 100 pertence a lista  
print(100 in lista\_teste\_op)

True

## Funções Built-in

In [60]: # Função len() retorna o comprimento da lista  
len(lista\_teste\_op)

Out[60]: 4

In [61]: # Função max() retorna o valor máximo da lista  
max(lista\_teste\_op)

Out[61]: 100

In [62]: # Função min() retorna o valor mínimo da lista  
min(lista\_teste\_op)

Out[62]: -5

In [63]:

```
# Criando uma Lista
listadomercado2 = ["ovos", "farinha", "leite", "maças"]
```

```
In [64]: # Adicionando um item no final da lista
listadomercado2.append("carne")
```

```
In [65]: listadomercado2
```

```
Out[65]: ['ovos', 'farinha', 'leite', 'maças', 'carne']
```

```
In [66]: listadomercado2.append("carne")
```

```
In [67]: listadomercado2
```

```
Out[67]: ['ovos', 'farinha', 'leite', 'maças', 'carne', 'carne']
```

```
In [68]: listadomercado2.count("carne")
```

```
Out[68]: 2
```

```
In [69]: # Criando uma Lista vazia
a = []
```

```
In [70]: print(a)
```

```
[]
```

```
In [71]: # Tipo da Lista vazia
type(a)
```

```
Out[71]: list
```

```
In [72]: #Adicionando valores a Lista vazia
a.append(10)
```

```
In [73]: a
```

```
Out[73]: [10]
```

```
In [74]: a.append(50); a
```

```
Out[74]: [10, 50]
```

```
In [75]: old_list = [1,2,5,10]
```

```
In [76]: new_list = []
```

```
In [77]: # Copiando os itens de uma Lista para outra
# Para cada item da Lista old_list faça um append para a Lista new_list. (Preencher
for item in old_list:
    new_list.append(item)
```

```
In [78]: new_list
```

```
Out[78]: [1, 2, 5, 10]
```

```
In [81]: c = [20,30]
```

```
In [82]: c.append(60)
```

```
In [83]: c.append(70)
```

```
In [84]: c
```

```
Out[84]: [20, 30, 60, 70]
```

```
In [85]: c.count(20)
```

```
Out[85]: 1
```

```
In [136...]: #list.extend(iterable)-> Adiciona no fim todos os elementos do argumento iterable para
cidades = ['Recife', 'Manaus', 'Salvador']
cidades.extend(['Fortaleza', 'Palmas'])
print (cidades)
```

```
['Recife', 'Manaus', 'Salvador', 'Fortaleza', 'Palmas']
```

```
In [137...]: # Informa o indice do item
# Lembrando que em Python o indice começa em 0
cidades.index('Salvador')
```

```
Out[137...]: 2
```

```
In [138...]: # Este elemento não consta na lista
cidades.index('Rio de Janeiro')
```

```
-----
ValueError                                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_1908\279291114.py in <module>
      1 # Este elemento não consta na lista
----> 2 cidades.index('Rio de Janeiro')
```

```
ValueError: 'Rio de Janeiro' is not in list
```

```
In [139...]: cidades
```

```
['Recife', 'Manaus', 'Salvador', 'Fortaleza', 'Palmas']
```

Out[139...]

In [140...]

```
# Inseri um elemento na Lista em uma posição específica.  
# Possui dois parâmetros o primeiro indica a posição e o segundo o elemento.  
cidades.insert(2, 110)
```

In [141...]

cidades

Out[141...]

['Recife', 'Manaus', 110, 'Salvador', 'Fortaleza', 'Palmas']

In [142...]

```
# Remove um item da Lista  
cidades.remove(110)
```

In [143...]

cidades

Out[143...]

['Recife', 'Manaus', 'Salvador', 'Fortaleza', 'Palmas']

In [144...]

```
#Remove um item em uma dada posição na Lista.  
cidades.pop(2)
```

Out[144...]

'Salvador'

In [145...]

cidades

Out[145...]

['Recife', 'Manaus', 'Fortaleza', 'Palmas']

In [146...]

```
#Remove todos os itens de uma Lista  
cidades.clear()
```

In [147...]

cidades

Out[147...]

[]

In [148...]

sabor=["morango", "baunilha", "chocolate"]

In [149...]

```
# Reverte a Lista (de tras para frente)  
sabor.reverse()
```

In [150...]

```
# Imprime a Lista  
sabor
```

Out[150...]

['chocolate', 'baunilha', 'morango']

In [151...]

x = [3, 2, 4, 1]

In [152...]

x

Out[152... [3, 2, 4, 1]

In [153... # Ordena a Lista (ela pode estar toda embaralhada, mas fica em ordem)  
x.sort()

In [154... x

Out[154... [1, 2, 3, 4]

## 2.4 Estruturas de dados - Dicionários

Até aqui falamos bastante sobre sequencias em Python, mas agora vamos mudar um pouco o foco e aprender sobre mapeamentos em Python. Se você estiver familiarizado com outras linguagens de programação, pode imaginar os dicionarios como tabelas de hash (hash tables).

Os dicionarios são construidos com o uso de chaves {} e virgulas separando cada elemento do dicionario.

```
dict =  
{k1:v1,K2:v2,...,Kn:vn}
```

Obs.: O que define o tipo de estrutura em Python é:

1. Listas-> []
2. Dicionários-> {}
3. Tuplas-> ()

Então, o que são mapeamentos? Mapeamentos são uma coleção de objetos que são armazenados por uma chave, ao contrario de uma sequencia de objetos armazenados em uma posição relativa.

Um dicionário em python consiste de uma chave e em seguida, um valor associado, ou seja, dicionários são mapeamentos de chaves e valores. Esse valor pode ser quase qualquer objeto python.

```
{chave1:valor1,chave2:valor2,...,chaven:valorn}
```

Obs.: Se parece com banco de dados não relacionais.

Tanto a chave como o valor quem define é o programador. Ele vai escolher o que ele quer como chave e como valor de acordo com o seu proposito. Ex: Chaves(strings)Valor(numero e inteiro). No dicionário não existe restrição de tipo.

## Dicionários

In [5]: # Isso é uma Lista  
estudantes\_lst = ["Mateus", 24, "Fernanda", 22, "Tamires", 26, "Cristiano", 25]

In [6]: estudantes\_lst

```
Out[6]: ['Mateus', 24, 'Fernanda', 22, 'Tamires', 26, 'Cristiano', 25]
```

Tanto a chave como o valor quem define é o programador. O programador é que escolhe o que ele quer como chave e como valor, neste caso, a chave é uma string e o valor uma idade.

```
In [7]: # Isso é um dicionário
estudantes_dict = {"Mateus":24, "Fernanda":22, "Tamires":26, "Cristiano":25}
```

```
In [8]: estudantes_dict
```

```
Out[8]: {'Mateus': 24, 'Fernanda': 22, 'Tamires': 26, 'Cristiano': 25}
```

Pode-se usar as chaves dos dicionários como índices, imprimindo desta maneira o seu par, ou seja, o valor correspondente a chave. Utiliza-se [] porque está representando um índice, de acordo com a indexação.

```
In [9]: # Quer retornar o valor do indice onde a chave é Mateus
estudantes_dict["Mateus"]
```

```
Out[9]: 24
```

Quando um índice não consta no dicionário, ao atribuir um valor o Python cria a chave no dicionário, que neste caso foi Pedro. Desta maneira, adiciona-se itens no dicionário(pares de chaves e valores)

```
In [14]: #Adicionando uma nova chave
estudantes_dict["Pedro"] = 23
```

```
In [15]: estudantes_dict["Pedro"]
```

```
Out[15]: 23
```

```
In [16]: estudantes_dict
```

```
Out[16]: {'Mateus': 24, 'Fernanda': 22, 'Tamires': 26, 'Cristiano': 25, 'Pedro': 23}
```

```
In [17]: # Pode-se mudar o valor em uma das chaves
estudantes_dict["Tamires"] = 62
```

```
In [18]: estudantes_dict
```

```
Out[18]: {'Mateus': 24, 'Fernanda': 22, 'Tamires': 62, 'Cristiano': 25, 'Pedro': 23}
```

```
In [19]: #Pode-se Limpar o dicionário utilizando o metodo clear(). Lembrando que para a utili
estudantes_dict.clear()
```

```
In [20]: estudantes_dict
```

```
Out[20]: {}
```

In [21]: *# Pode-se deletar o dicionário utilizando o comando del*  
**del** estudantes\_dict

In [22]: *# Se tentar imprimir vai dizer que esse dicionário não existe mais.*  
estudantes\_dict

```
NameError Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_788/724380893.py in <module>
      1 # Se tentar imprimir vai dizer que esse dicionário não existe mais.
----> 2 estudantes_dict

NameError: name 'estudantes_dict' is not defined
```

In [23]: **print**('\\n')

In [24]: estudantes = {"Mateus":24, "Fernanda":22, "Tamires":26, "Cristiano":25}

In [25]: estudantes

Out[25]: {'Mateus': 24, 'Fernanda': 22, 'Tamires': 26, 'Cristiano': 25}

In [26]: *# Verifica-se o comprimento do dicionário*  
**len**(estudantes)

Out[26]: 4

In [27]: *# Extraí apenas as chaves do dicionário*  
estudantes.keys()

Out[27]: dict\_keys(['Mateus', 'Fernanda', 'Tamires', 'Cristiano'])

In [28]: *# Extraí apenas os valores do dicionário*  
estudantes.values()

Out[28]: dict\_values([24, 22, 26, 25])

In [29]: *# Extraí as combinações(pares) do dicionário*  
estudantes.items()

Out[29]: dict\_items([('Mateus', 24), ('Fernanda', 22), ('Tamires', 26), ('Cristiano', 25)])

In [30]: estudantes2 = {"Maria":27, "Erika":28, "Milton":26}

In [31]: estudantes2

Out[31]: {'Maria': 27, 'Erika': 28, 'Milton': 26}

```
In [32]: # Quer atualizar o dicionário estudantes com o conteúdo de estudantes2  
# Como uma concatenação, junta-se(une-se) 2 dicionários  
estudantes.update(estudantes2)
```

```
In [33]: estudantes
```

```
Out[33]: {'Mateus': 24,  
          'Fernanda': 22,  
          'Tamires': 26,  
          'Cristiano': 25,  
          'Maria': 27,  
          'Erika': 28,  
          'Milton': 26}
```

```
In [34]: print('\n')
```

```
In [35]: # Pode-se criar um dicionário vazio.  
dic1 = {}
```

```
In [36]: dic1
```

```
Out[36]: {}
```

```
In [37]: # Adiciona-se elementos no dicionário, quando se atribui um valor a uma chave que não existe  
dic1["key_one"] = 2
```

```
In [38]: print(dic1)
```

```
{'key_one': 2}
```

```
In [39]: # Adicionou novamente outra chave atribuindo um valor, só que neste caso a chave possui uma vírgula no nome  
dic1[10] = 5
```

```
In [40]: dic1
```

```
Out[40]: {'key_one': 2, 10: 5}
```

```
In [41]: # Não há restrições de tipo  
dic1[8.2] = "Python"
```

```
In [42]: dic1
```

```
Out[42]: {'key_one': 2, 10: 5, 8.2: 'Python'}
```

```
In [43]: dic1["teste"] = 5
```

```
In [44]: dic1
```

```
Out[44]: {'key_one': 2, 10: 5, 8.2: 'Python', 'teste': 5}
```

```
In [45]: print('\n')
```

```
In [46]: dict1 = {}
```

```
In [47]: dict1
```

```
Out[47]: {}
```

```
In [48]: dict1["teste"] = 10
```

```
In [49]: dict1["key"] = "teste"
```

```
In [50]: # Atenção, pois chave e valor podem ser iguais, mas representam coisas diferentes.  
dict1
```

```
Out[50]: {'teste': 10, 'key': 'teste'}
```

```
In [51]: print('\n')
```

```
In [52]: dict2 = {}
```

```
In [53]: dict2["key1"] = "Big Data"
```

```
In [54]: dict2["key2"] = 10
```

```
In [55]: dict2["key3"] = 5.6
```

```
In [56]: dict2
```

```
Out[56]: {'key1': 'Big Data', 'key2': 10, 'key3': 5.6}
```

```
In [57]: #Pode-se extrair os valores de cada uma das chaves e atribuir a uma variável.  
a = dict2["key1"]
```

```
In [58]: b = dict2["key2"]
```

```
In [59]: c = dict2["key3"]
```

In [60]: a, b, c

Out[60]: ('Big Data', 10, 5.6)

In [61]: print('\n')

In [62]: # Dicionário de Listas (os valores das chaves podem ser Listas)  
dict3 = {'key1':1230,'key2':[22,453,73.4],'key3':['leite','maça','batata']}

In [63]: dict3

Out[63]: {'key1': 1230, 'key2': [22, 453, 73.4], 'key3': ['leite', 'maça', 'batata']}

In [64]: # Imprime o valor da chave 2 que é uma Lista  
dict3['key2']

Out[64]: [22, 453, 73.4]

In [65]: # Acessando um item da Lista, dentro do dicionário  
# Extrai o primeiro item da Lista da chave 3, que se encontra no dicionario, depois  
dict3['key3'][0].upper()

Out[65]: 'LEITE'

In [66]: # Operações com itens da Lista, dentro do dicionário  
# Subtraiu o primeiro item da Lista que se encontra na segunda chave do dicionario p  
var1 = dict3['key2'][0] - 2

In [67]: var1

Out[67]: 20

In [68]: # Duas operações no mesmo comando, para atualizar um item dentro da Lista  
# Utilizando um Operador de atribuição, fazendo 2 operações em uma, faz a subtração  
# Para isso é necessário colocar o símbolo de slice(fatiamento) que é [], fatia o ob  
dict3['key2'][0] -= 2

In [69]: dict3

Out[69]: {'key1': 1230, 'key2': [20, 453, 73.4], 'key3': ['leite', 'maça', 'batata']}

## Criando dicionários aninhados

In [ ]: # Criando dicionários aninhados (um dicionário dentro de outro dicionário)  
# Em vez do valor na chave é outro dicionário.  
dict\_aninhado = {'key1':{'key2\_aninhada':{'key3\_aninhada':'Dict aninhado em Python'}}

```
In [ ]: dict_aninhado
In [ ]: dict_aninhado['key1']['key2_aninhada']['key3_aninhada']
In [74]: nomedodicionario = {'key1_a0': 2, 'key2_a0': {'key1_a1': 1, 'key2_a1': {'key1_a2': 4
In [75]: nomedodicionario['key1_a0']
Out[75]: 2
In [76]: nomedodicionario['key2_a0']
Out[76]: {'key1_a1': 1, 'key2_a1': {'key1_a2': 4, 'key2_a2': 5}}
In [81]: nomedodicionario ['key2_a0'] ['key1_a1']
Out[81]: 1
In [82]: nomedodicionario ['key2_a0'] ['key2_a1']
Out[82]: {'key1_a2': 4, 'key2_a2': 5}
In [83]: nomedodicionario ['key2_a0'] ['key2_a1'] ['key1_a2']
Out[83]: 4
In [85]: nomedodicionario ['key2_a0'] ['key2_a1'] ['key2_a2']
Out[85]: 5
```

## 2.4 Estruturas de dados - Tuplas

Em python tuplas são muito semelhantes as listas, no entanto, ao contrario de listas, **tuplas são imutáveis**, o que significa que não podem ser alteradas, como os dias da semana ou datas em um calendario.

As tuplas são construidas com o uso de parenteses () e virgulas separando cada elemento da tupla.

Tupla = (item1,item2,item3,...,itemz)

Tuplas não são utilizadas com frequencia, como listas por exemplo, mas são usadas quando é necessário a imutabilidade. Se em seu programa você precisa ter certeza que os dados não sofreram mudança, então tupla pode ser a sua solução. Ela fornece uma fonte conveniente de integriade de dados.

Observações.:

1. Não existe restrição para tipos de dados;
2. Como ela é imutável, não é possível adicionar/deletar/atribuir um item na tupla;
3. É possível usar indexação para retornar um único índice da tupla, assim como usar slicing;
4. Dentre os métodos que podem ser usados estão: len(); **index()**; **del\_**. Com isso, pode-se dizer que é possível, verificar o comprimento da tupla, verificar o índice de um dos elementos da tupla e deletar a tupla.

Como alterar uma tupla, seria necessário deletá-la e criá-la novamente, só para apagar um único elemento? Não. Pode-se converter uma tupla para uma lista, fazer as alterações necessárias e depois converter novamente para tupla. Desta maneira contorna-se a limitação da tupla.

## Tuplas

In [86]:

```
# Criando uma tupla
tupla1 = ("Geografia", 23, "Elefantes")
```

In [87]:

```
# Imprimindo a tupla
tupla1
```

Out[87]:

```
('Geografia', 23, 'Elefantes')
```

In [88]:

```
# Tuplas não suportam append()
tupla1.append("Chocolate")
```

---

```
AttributeError                                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_788/3030862692.py in <module>
      1 # Tuplas não suportam append()
----> 2 tupla1.append("Chocolate")
```

```
AttributeError: 'tuple' object has no attribute 'append'
```

In [89]:

```
# Tuplas não suportam delete de um item específico
del tupla1["Gatos"]
```

---

```
TypeError                                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_788/3590979409.py in <module>
      1 # Tuplas não suportam delete de um item específico
----> 2 del tupla1["Gatos"]
```

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

In [92]:

```
# Tuplas podem ter um único item
tupla1 = ("Chocolate")
```

In [93]:

```
tupla1
```

Out[93]:

```
'Chocolate'
```

In [94]:

```
tupla1 = ("Geografia", 23, "Elefantes")
```

In [95]: #Pode-se usar indexação, para extrair um unico item da tupla.  
tupla1[0]

Out[95]: 'Geografia'

In [96]: # Verificando o comprimento da tupla  
len(tupla1)

Out[96]: 3

In [97]: # Slicing, da mesma forma que se faz com listas  
# Retorna todos os elementos a partir do elemento 1 do indice  
tupla1[1:]

Out[97]: (23, 'Elefantes')

In [98]: # Verifica qual o indice de um dos elementos da tupla.  
tupla1.index('Elefantes')

Out[98]: 2

In [99]: # Tuplas não suportam atribuição de item, pois não é possível alterar os itens dentro  
tupla1[1] = 21

---

**TypeError** Traceback (most recent call last)  
~\AppData\Local\Temp\ipykernel\_788/1373090751.py in <module>  
 1 # Tuplas não suportam atribuição de item  
----> 2 tupla1[1] = 21

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

In [100...]: # Deletando a tupla  
~~del~~ tupla1

In [101...]: tupla1

---

**NameError** Traceback (most recent call last)  
~\AppData\Local\Temp\ipykernel\_788/3141490790.py in <module>  
----> 1 tupla1

**NameError**: name 'tupla1' is not defined

In [102...]: # Criando uma tupla  
t2 = ('A', 'B', 'C')

In [103...]: t2

Out[103...]: ('A', 'B', 'C')

In [104...]: # Tuplas não suportam atribuição de item

```
t2[0] = 'D'
```

```
-----  
TypeError                                                 Traceback (most recent call last)  
~\AppData\Local\Temp\ipykernel_788/3449531607.py in <module>  
      1 # Tuplas não suportam atribuição de item  
----> 2 t2[0] = 'D'
```

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

In [105...]

```
# Usando a função list() para converter uma tupla para lista, atribuindo o resultado  
lista_t2 = list(t2)
```

In [106...]

```
lista_t2
```

Out[106...]

```
['A', 'B', 'C']
```

In [107...]

```
lista_t2.append('D')
```

In [108...]

```
# Usando a função tuple() para converter uma lista para tupla  
t2 = tuple(lista_t2)
```

In [109...]

```
t2
```

Out[109...]

```
('A', 'B', 'C', 'D')
```