código.py

GALÁXIA 2

Lógica de programação

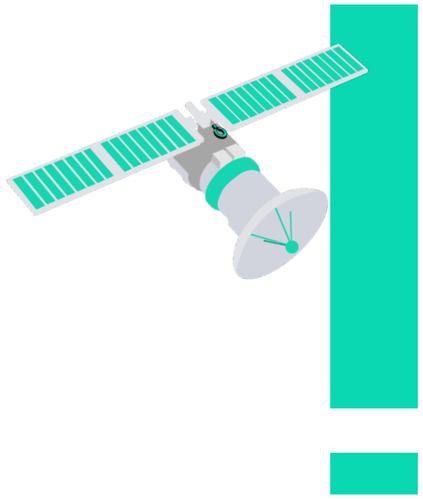
Introdução

Seja bem-vindo à galáxia 2, esse módulo abordará a lógica de programação. Assuntos como tipos de dados, operações matemáticas e laços de repetição são alguns dos assuntos que compõem esse módulo. O objetivo deste PDF é servir como uma caixa de ferramentas, onde poderá tirar dúvidas e consultar durante as aulas do curso.

Mundo 1

1. O que é um algoritmo?

Apesar do receio quanto ao nome, algoritmos são simples de entender. Você pode até não os identificar no dia a dia, mas eles são mais comuns do que parecem. O simples fato de toda manhã ao acordar, olhar a janela e pegar um guarda-chuva, caso esteja chovendo, já é um algoritmo de decisão no seu dia a dia.



A tradução para o Python também é bem simples. Vamos imaginar o exemplo acima, mas agora dentro da linguagem de programação:

Exemplo:

```
olhar_a_janela = True
esta_chovendo = True

if olhar_a_janela == True:
    if esta_chovendo == True:
        print('Pegar guarda-chuva')
    else:
        print('Não pegar guarda-chuva')
```

Resposta:

```
>> Pegar guarda-chuva
```

Claro que esse algoritmo é simples, porém a ideia é você entender que essas tomadas de decisões são parte fundamental do código. É importante ressaltar que tudo dentro da computação se resume ao sistema binário, composto por 0 ou 1. Textos, scripts, filmes e redes sociais são apenas algumas das infinitas interpretações que o computador faz dos sistemas binários.

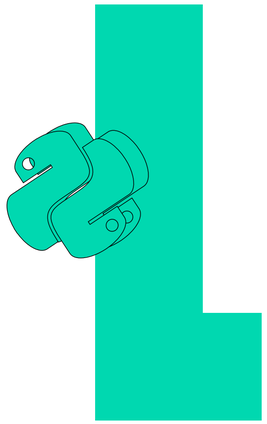
2. Como criar um algoritmo?

O primeiro passo para se criar um algoritmo é formular uma hipótese, que nada mais é, que um conjunto de regras que não foram validadas nem testadas. A nossa pode ser, por exemplo, **toda vez que chove no Rio de Janeiro a bolsa de valores do Brasil cai**. A justificativa pode ser a mais banal do mundo: dias chuvosos são tristes.

O segundo passo é criar um código que analise a previsão do tempo no Rio de Janeiro e, toda vez que estiver chovendo, esse código venderá bolsa de valores.

Além disso, seu código deve levar em consideração todas as possibilidades:

- E se estiver nublado?
- E se chover e parar?



Tudo isso deve ser levado em consideração na hora da construção do algoritmo.

O terceiro passo é testar se a hipótese estava certa ou não. Verificar o retorno dos dias chuvosos, quantos dias confirmaram a sua hipótese e quantos não confirmaram. No final, com os dados em mãos, a conclusão poderá ser tomada.

Esse exemplo tentou mostrar que o Python será apenas uma das ferramentas para auxiliar suas decisões. No final de tudo, quem decide como o código será construído e estruturado é uma pessoa.

Mundo 2

1. Função print()

A função print é a mais básica do Python, mas também a mais utilizada. Ela exibe mensagens, ou seja, ela imprime uma mensagem no terminal.

Exemplo:

```
print("Hello world")
```

Resposta:

```
>> Hello world
```

2. Atribuição de objetos

No Python é comum atribuir valores às variáveis. Esses valores podem ser números ou strings (palavras), booleanos, etc.

Exemplo:

```
#Números
a = 2
b = 4

#String
codigo_py = "o melhor curso de python brasil"

#Podemos printar o que nós programamos

print(codigo_py)
print(a)
print(b)
```

Resposta:

```
>>  o melhor curso
de python brasil
>>  2
>>  4
```

3. Quebra de linha

É importante a utilização de ferramentas, que deixem a visualização do seu código mais agradável. Existem algumas formas de fazer isso e uma delas é a quebra de linha. Caso tenha um texto muito grande, ou queira imprimir uma matriz, a quebra de linha é fundamental para a visualização do seu código, e até mesmo para organização também.

Imagine que você queira imprimir uma matriz 3 x 3.

Exemplo:

```
print(" [1,2,3] [1,2,3] [1,2,3]")
```

Resposta:

```
>> [1,2,3] [1,2,3]
[1,2,3]
```

Ao tentar quebrar a linha da forma comum com a tecla “enter”, verá que o python não reconhece esse método, ele retornará um erro. Para essa função teremos dois modos de fazer. O primeiro método é utilizando o “ \n “ e o segundo método é utilizando o ‘’’ ‘’’.

Exemplo:

```
print(" [1,2,3]
      [1,2,3]
      [1,2,3]")
```

Resposta:

```
>>  File "c:\DEV\codigos_curso-master\codigo.py", line
3
    print(" [1, 2, 3]
          ^
    SyntaxError: unterminated string literal (detected
at line 3)
```

3.1. Pelo método \n

Exemplo:

```
print(" [1, 2, 3]\n [1, 2, 3]\n [1, 2, 3]")
```

Resposta:

```
>>   [1, 2, 3]
     [1, 2, 3]
     [1, 2, 3]
```

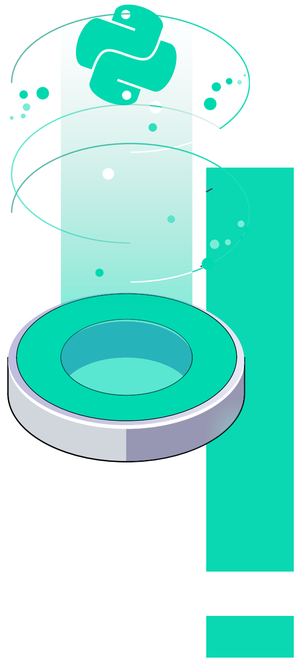
3.2. Pelo método do ”’ ”’

Exemplo:

```
print(''' [1, 2, 3]
    [1, 2, 3]
    [1, 2, 3] ''')
```

Resposta:

```
>>   [1, 2, 3]
     [1, 2, 3]
     [1, 2, 3]
```



3.3. A diferença entre atribuição e igualdade no Python

Diferentemente da equação matemática que utiliza o símbolo “=” com significado de “igual a”, O python utiliza o “=” como forma de atribuir valor a um objeto.

No exemplo acima, o Python está lendo e atribuindo o “2” a variável “a”. Caso queira fazer uma igualdade (uma comparação), o Python exige que utilize “==” e ele retornará um valor.

No código abaixo está sendo atribuído ao objeto “verificando_uma_condicao” a igualdade “b == 4”. Como já tinha sido atribuído o “4” ao “b” o Python retornará “True”.

Exemplo:

```
b = 4

verificando_uma_condicao = b == 4

print(verificando_uma_condicao)
```

Resposta:

```
>> True
```

Isso se dá pois o valor “4” foi atribuído ao objeto “b”, e depois foi feita uma igualdade**(conferência)** onde o python retornaria um valor, seja ele True ou False.

Exemplo:

```
b = 4

verificando_uma_condicao = b == 7

print(verificando_uma_condicao)
```

Resposta:

```
>> False
```

Mundo 3

1. Comentários

Existem dois jeitos de comentar dentro do código. Pode ser através da **#** ou ‘’’ ‘’’ (exemplo abaixo). Entretanto, o comentário é apenas para alguns casos e, no geral, quanto menos melhor.

Exemplo:

```
'''
comentar
assim
'''
```

Exemplo:

```
#comentar assim
```

A boa prática de programação ressalta a importância de saber programar bem, ao ponto de ser simples e eficaz. Saber escolher o nome das variáveis é uma característica subestimada e isso pode te ajudar muito no seu dia a dia.

Um exemplo simples de como a escolha do nome das variáveis influencia no entendimento está logo abaixo. Ao ter que explicar o nome das variáveis por meio de comentários, podemos dizer que você "falhou como programador".

Exemplo ruim:

```
#as variáveis a e b representam quantas vezes o palmeiras e o flamengo ganharam
# no campeonato brasileiro, respectivamente.

a = 2
b = 4
```

Exemplo bom:

```
vitorias_palmeiras = 2
vitorias_flamengo = 4
```

Mundo 4

1. Tipos de dados

Pode ser que em algum momento apareça uma necessidade de transformar dados, seja uma verificação, um arredondamento ou até mesmo uma concatenação. Nesse módulo entraremos a fundo nos tipos de dados, quais são e como manipulá-los.

1.1. Tipo inteiro(int)

O tipo inteiro(int) representa números inteiros positivos e negativos, que não apresentam a parte decimal. Exemplo de números inteiros (-23455, -1234, -1000 -4 , -7 , 0 , 3, 28, 165, 1283).

1.2. Tipo ponto flutuante ou decimal(float)

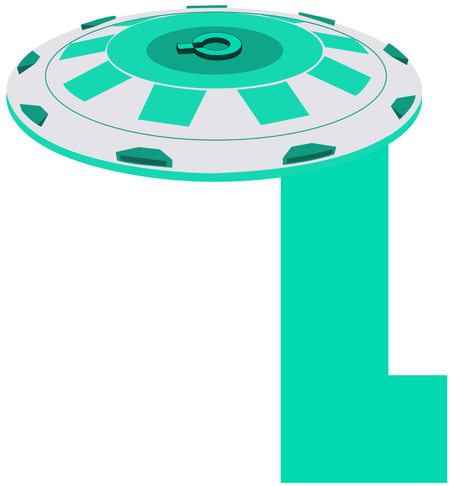
O tipo ponto flutuante(float) representa números reais positivos e negativos, que apresentam a parte decimal. Exemplo de números reais (-23455.65 , -1234.7654 , -1000.358 , -4.325 , -7.10 , 0.154 , 3.97 , 28.66 , 165.9 , 1283.7). Se atentar ao fato de que o Python utiliza o ponto como separador decimal, ao invés da vírgula, como é comum no Brasil.

1.3. Tipo string(str)

O tipo string(str) representa a junção de caracteres, que são utilizados com intuito de representar palavras, textos ou até mesmo frases. Uma string é representada entre aspas duplas ou aspas simples.

1.4. Tipo boolean(bool)

O tipo boolean(bool) representa um tipo de dado lógico, que pode representar apenas True ou False.



2. Como verificar o tipo de um objeto?

Função type()

Essa função retornará o tipo da variável passada como parâmetro.

Exemplo:

```
inteiro = 6
float = 6.6
string = "6"
boolean = True

print('O Tipo do inteiro é =>', type(inteiro))
print('O Tipo do float é =>', type(float))
print('O Tipo da string é =>', type(string))
print('O Tipo do boolean é =>', type(boolean))
```

Resposta:

```
>> O Tipo do inteiro é => <class 'int'>
>> O Tipo do float é => <class 'float'>
>> O Tipo da string é => <class 'str'>
>> O Tipo do boolean é => <class 'bool'>
```

3. Como converter um tipo para o outro?

Existem algumas funções que irão ajudar na hora de converter os tipos de dados, esses são alguns exemplos:

3.1. Função int()

A função **int()** converte qualquer número em um número **int**.

Exemplo:

```
String = "5" #string
inteiro = int(5)
inteiro2 = int(6.5)
flutuante3 = float(String)

print("Tipo inteiro 1 => ", type(inteiro))
print("Inteiro 1 => ",inteiro)
print("Tipo inteiro 2 => ",type(inteiro2))
print("Inteiro 2 => ",inteiro2)
print("Tipo inteiro 3 => ",type(flutuante3))
print("Inteiro 3 => ",flutuante3)
```

Resposta:

```
>> Tipo inteiro 1 =>  <class 'int'>
>> Inteiro 1 =>  5
>> Tipo inteiro 2 =>  <class 'int'>
>> Inteiro 2 =>  6
>> Tipo inteiro 3 =>  <class 'int'>
>> Inteiro 3 =>  5
```

3.2. Função float()

A função **float()** converte qualquer número em um número **float**.

Exemplo:

```
String = "5" #string
flutuante1 = float(6.5)
flutuante2 = float(6)
flutuante3 = float(String)

print("flutuante 1 => ",flutuante1)
print("Tipo flutuante 1 => ", type(flutuante1))
print("flutuante 2 => ",flutuante2)
print("Tipo flutuante 2 => ", type(flutuante2))
print("flutuante 3 => ",flutuante3)
print("Tipo flutuante 3 => ", type(flutuante3))
```

Resposta:

```
flutuante 1 =>  6.5
Tipo flutuante 1 =>  <class 'float'>
flutuante 2 =>  6.0
Tipo flutuante 2 =>  <class 'float'>
flutuante 3 =>  5.0
Tipo flutuante 3 =>  <class 'float'>
```

3.3. Função str()

A função **str()** converte qualquer objeto para string.

Exemplo:

```
flutuante = 6.3
inteiro = 6
String = "codigo.py"
String2 = "curso de python"
String3 = str(flutuante)
String4 = str(inteiro)

print("flutuante => ",flutuante)
print("Tipo flutuante => ", type(flutuante))
print("inteiro => ",inteiro)
print("Tipo inteiro => ", type(inteiro))
print("String => ",String)
print("Tipo String => ", type(String))
print("String 2 => ",String2)
print("Tipo String 2 => ", type(String2))
print("String 3 => ",String3)
print("Tipo String 3 => ", type(String3))
print("String 4 => ",String4)
print("Tipo String 4 => ", type(String4))
```

Resposta:

```
>> flutuante =>  6.3
>> Tipo flutuante =>  <class 'float'>
>> inteiro =>  6
>> Tipo inteiro =>  <class 'int'>
>> String =>  codigo.py
>> Tipo String =>  <class 'str'>
>> String 2 =>  curso de python
>> Tipo String 2 =>  <class 'str'>
>> String 3 =>  6.3
>> Tipo String 3 =>  <class 'str'>
>> String 4 =>  6
>> Tipo String 4 =>  <class 'str'>
```

3.4. Função bool()

Sua função **bool()** retorna True ou False. É muito utilizado para conferência de resultados.

Exemplo:

```
boleano = True
boleano2 = False
a = 10
b = 20
c = 20

print("Boleano => ",boleano)
print("Tipo Boleano => ", type(boleano))
print("Boleano 2 => ",boleano2)
print("Tipo Boleano 2 => ", type(boleano2))
print("Verificação se a é igual a b =>", a == b)
print("Verificação se a é igual a c =>", c == b)
```

Resposta:

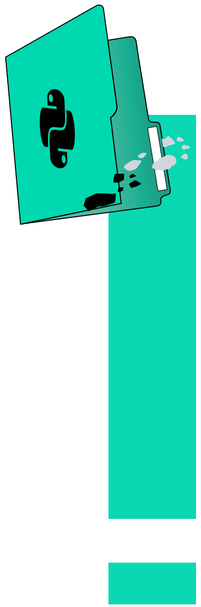
```
>> Boleano =>  True
>> Tipo Boleano =>  <class 'bool'>
>> Boleano 2 =>  False
>> Tipo Boleano 2 =>  <class 'bool'>
>> Verificação se a é igual a b => False
>>  Verificação se a é igual a c => True
```

4. Como criar strings variáveis?

String variáveis podem ser criadas por meio de "f functions". Sua utilidade é concatenar uma variável e uma string, tornando o texto dentro do Python variável. Faremos a concatenação por meio de " { } ", o que significa que entre os colchetes poderemos colocar a variável de escolha.

Exemplo:

```python
empresa = "Weg" #podemos colocar um input de usuário aqui
string_variavel1 = f"A melhor empresa do Brasil é a {empresa}"
empresa2 = 444
empresa3 = 333
string_variavel2 = f"A melhor empresa do Brasil é a {empresa3 + empresa2} Partners"

print(string_variavel1)
print("O tipo da string variável 1 =>", type(string_variavel1))
print(string_variavel2)
print("O tipo da string variável 2 =>", type(string_variavel2))
```



Resposta:

```
>> A melhor empresa do Brasil é a Weg
>> O tipo da string variável 1 => <class 'str'>
>> A melhor empresa do Brasil é a 777 Partners
>> O tipo da string variável 2 => <class 'str'>
```

Mundo 5

1. Como colocar dados de fora do programa?

1.1. função input()

Essa função possibilita a interação com o programa, ou seja, é possível adicionar informações de fora. Ao armazenar o valor de entrada à uma variável, poderá interagir com esse valor. É importante notar que a função input sempre retornará uma string. Por isso, é importante que os tipos de objetos sejam especificados, como ensinado no mundo 4.

Exemplo:

```
numero_digitado = input('Digite um número inteiro: ')
print(f" Número digitado é => {numero_digitado} e seu tipo é => {type(numero_digitado)}")
```

Ao executar programa, aparecerá no terminal a seguinte mensagem:

Resposta 1:

```
>>   Digite um número inteiro:[]
```

Resposta 2:

```
>>   Número digitado é => 6 e seu tipo é => <class 'str'>
```

Caso necessite de uma variável de tipo inteiro, precisará colocar em prática os conhecimentos do mundo 4.

Exemplo:

```
numero_digitado = int(input('Digite um número inteiro: '))

numero_final = numero_digitado + 2

print(f" Número digitado é => {numero_digitado} e seu tipo é => {type(numero_digitado)}")
print(f" Número final é => {numero_final} e seu tipo é => {type(numero_final)}")
```

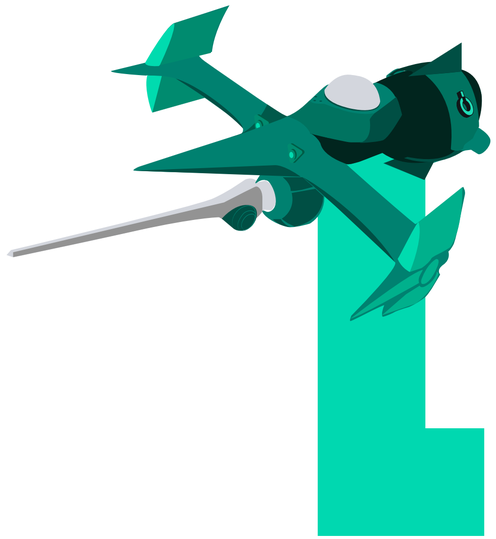
Resposta:

```
>>   Digite um número inteiro:[]
```

Ao digitar o número 6, a seguinte mensagem aparecerá:

Resposta:

```
>>   Número digitado é => 6 e seu tipo é
=> <class 'int'>
>>   Número final é => 8 e seu tipo é =>
<class 'int'>
```



Mundo 6

1. Operadores aritméticos

A tabela a seguir representa os principais operadores aritméticos do python. Vale ressaltar que não existe uma operação específica para tirar a raiz de um número, mas sabemos que a exponenciação é o inverso da raiz. Nesse caso precisaremos fazer a operação inversa, para encontrar um resultado.

Exemplo:

- **Matematicamente**

$$\sqrt{4} = 4^{1/2} = 2$$

- **Python**

$$\sqrt{4} = 4 ** (1/2) = 2$$

Exemplo 2 :

- **Matematicamente**

$$\sqrt[3]{4} = 4^{1/3} = 1,587$$

- **Python**

$\sqrt[3]{4}$ = 4 ** (1/3) = 1,587

Operadores	Descrição	Exemplos
+	Realiza soma	2 + 3 = 5
-	Realiza subtração	2 - 3 = -1
*	Realiza multiplicação	2 * 3 = 6
/	Realiza divisão	2 / 3 = 0,667
//	Realiza divisão inteira	2 // 3 = 0
%	Realiza resto da divisão	1 % 3 = 2
**	Realiza a exponenciação	2 ** 3 = 8

O python também tem uma ordem de prioridade de execução, e segue a mesma convenção utilizada na matemática. A ordem de execução dentro do Python é a seguinte:

1. Parênteses
2. Exponenciação
3. Multiplicação e Divisão
4. Soma e subtração

É importante observar que, no Python, não utilizamos colchetes nem chaves para representar outras prioridades. Dentro da linguagem, usamos outros parênteses para definir prioridades dentro de outras prioridades.

Exemplo:

```
valor = ((2+3) + 25) + 25 ** 2
print(f"O valor da operação matemática é {valor}")
```

Resposta:

```
>> O valor da operação matemática é 655
```

Mundo 7

1. Operadores matemáticos prática

Utilizando os conhecimentos dos mundos anteriores, colocaremos em prática os conceitos aprendidos e faremos um programa que fará as principais operações matemáticas. Note que esse programa servirá para qualquer número escolhido e, para isso, o input será parte fundamental do código, dado que ele será a porta de entrada dos dados externos.

No código é importante explicitar as variáveis, “numero” e “numero2”, como um número inteiro, por meio da função int() . Caso isso não esteja explícito no código, ele considerará que as variáveis são strings, como visto no mundo 5.

Nesse exemplo, numero = 5 e numero2 = 5.

Exemplo:

```
numero = int(input('Digite um número inteiro qualquer: '))
numero2  = int(input('Digite um número inteiro qualquer: '))

soma = numero + numero2
multiplicar = numero * numero2
dividir = numero / numero2
elevar = numero ** numero2
raiz = numero ** (1/2)
raiz2 = numero2 ** (1/2)
dividir_inteiro = numero //numero2
resto = numero % numero2

print(f''' A soma dos dois números é {soma}\n
A multiplicação é {multiplicar}\n
A divisão é {dividir}\n
O primeiro número elevado ao segundo chega ao resultado de {elevar}\n
A raiz quadrada do 1º número é {raiz} e do segundo número é {raiz2}\n
A divisão inteira é {dividir_inteiro} e o resto é {resto}''')
```

Resposta:

```
>>  A soma dos dois números é 10
      A multiplicação é 25
      A divisão é 1.0
      O primeiro número elevado ao segundo chega ao resultado de 3125
      A raiz quadrada do 1° número é 2.23606797749979 e do segundo número
é 2.23606797749979
      A divisão inteira é 1 e o resto é 0
```

2. Multiplicação de string

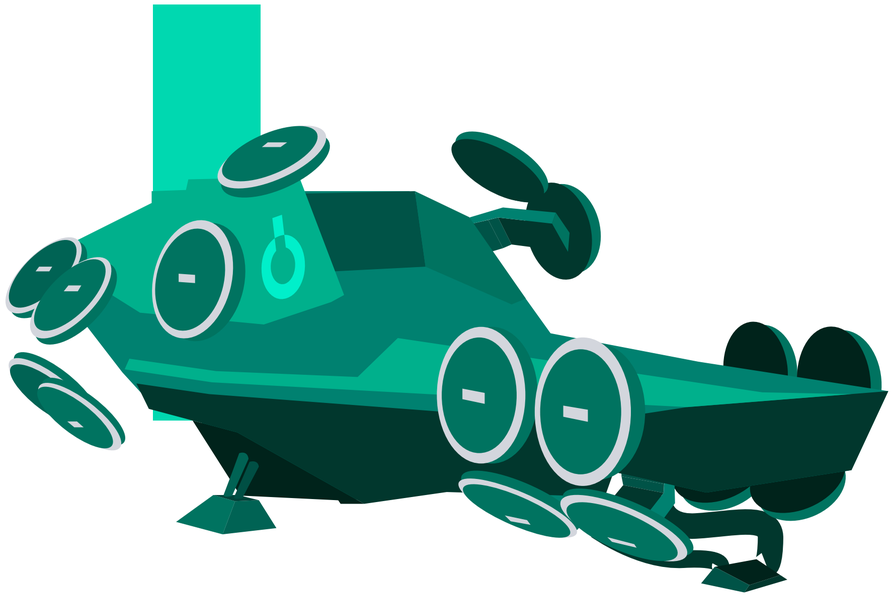
Também há a possibilidade de multiplicar strings, mas nesse caso ele multiplicará a quantidade de vezes que a string aparece.

Exemplo:

```
string = 'python '
string_multiplicada = string * 20
print(string_multiplicada)
```

Resposta:

```
>>  python python python python python python python python python python
python python python python python python python python python python
```



Mundo 8

1. Módulos e bibliotecas

Pense em uma caixa de ferramentas com utensílios de obra. Cada tarefa feita na construção demandará uma ferramenta específica, mas que facilita o trabalho do construtor. É dessa forma que devemos enxergar as bibliotecas, que, por definição, são um conjunto de módulos e funções criados por outros desenvolvedores, com o intuito de facilitar a programação. Existem milhares de bibliotecas, e cada uma delas trabalha de uma forma diferente. As mais importantes e que abordaremos neste mundo são pandas, numpy, matplotlib, random.

- Numpy é uma biblioteca muito utilizada para operações matemáticas, tratamentos de dados, estatísticas descritivas, gerações de filtragens e etc...

- Pandas é uma biblioteca que fornece estruturas e ferramentas para análise de dados, por meio de listas e dataframes.

- Matplotlib é uma biblioteca utilizada para visualização de dados e plotagem gráfica.

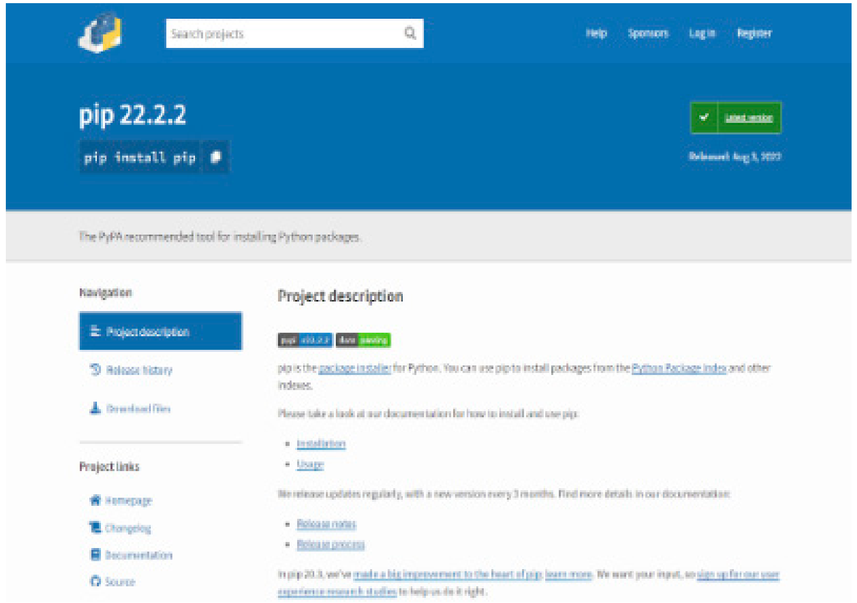
- Random é uma biblioteca utilizada para gerar números aleatórios.

2. Instalação de bibliotecas

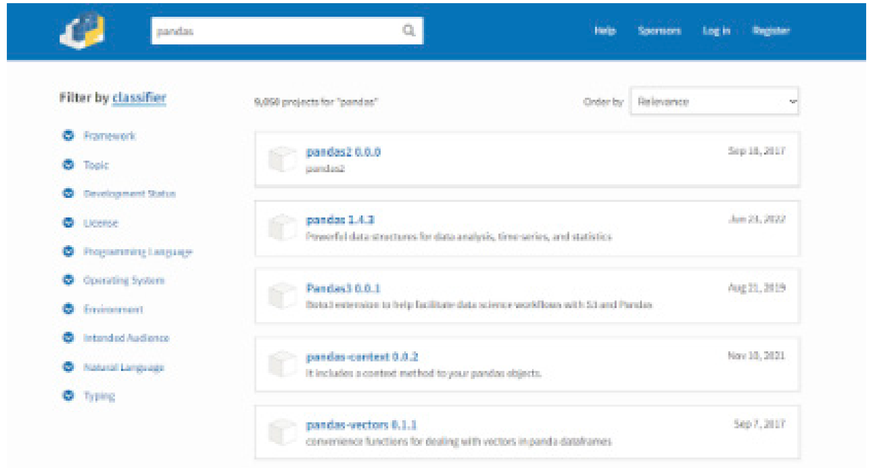
A instalação de bibliotecas no python é feita por meio do gerenciador de pacotes “pip”. Com ele que instalamos, removemos e atualizamos os pacotes dentro do python. Existe um site do pip onde podemos consultar uma palavras específicas ou até uma palavra chave.

Vamos supor que a gente não tem ideia de qual comando utilizar para baixar as bibliotecas numpy, pandas, matplotlib, random. Então acesse o site do gerenciador de pacotes pip por meio deste link (https://pypi.org/project/pip/).

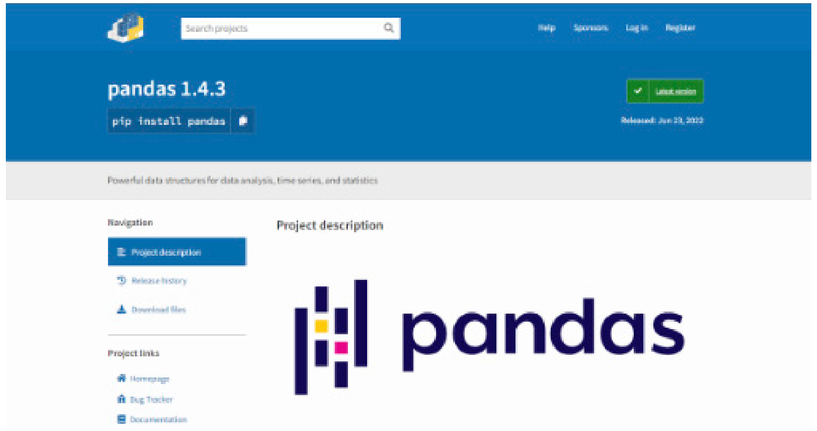
O primeiro passo é acessar o site do pip e note que o pip já mostra o código de instalação do próprio pacote na página principal do site.

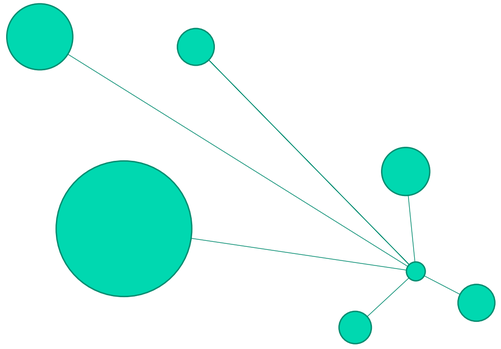


O segundo passo é procurar na barra de pesquisa pelo pacote que deseja. No nosso caso, o pacote que estamos procurando é o pandas. Nesse caso aparecerá algumas opções e, nesse caso, a melhor opção é a segunda ,"pandas 1.4.3" , pela descrição do pacote.



Ao clicar em "pandas 1.4.3", abrirá uma página principal com as instruções de instalação, utilização e breves descrições sobre o pacote clicado. Depois de verificado, basta fazer a instalação por meio do terminal com os seguintes comandos "pip install pandas". Esse mesmo passo a passo pode ser feito para as outras bibliotecas.





Mundo 9

1. Módulos random

Como visto anteriormente, o módulo random gera números aleatórios. Veremos agora algumas funções deste módulo.

1.1. Função random()

A função random() gera números decimais (float) entre 0 e 1, totalmente aleatórios. Essa função não recebe parâmetro.

Exemplo:

```
import random

valor_entre_0_ou_1 = random.random()

print(f"O número aleatório gerado foi => {valor_entre_0_ou_1}")
print(f"O tipo do número aleatório gerado foi => {type(valor_entre_0_ou_1)}")
```

Resposta:

```
>> O número aleatório gerado foi => 0.7317250673052144
>> O tipo do número aleatório gerado foi => <class 'float'>
```

1.2. Função uniform()

A função uniform() gera números aleatórios decimais (float) e é considerada uma distribuição retangular, onde todos os números têm a mesma probabilidade de ocorrer. Essa função recebe como parâmetro o intervalo escolhido, no exemplo abaixo o intervalo foi de 10 até 100.

Exemplo:

```python
import random

valor_decimal_de_igual_probabilidade_entre_valores = random.uniform(10, 100)
print(f"O número aleatório gerado foi => {valor_decimal_de_igual_probabilidade_entre_valores}")
print(f"O tipo do número aleatório gerado foi => {type(valor_decimal_de_igual_probabilidade_entre_valores)}")
```

Resposta:

```
>> O número aleatório gerado foi => 50.35965327731095
>> O tipo do número aleatório gerado foi => <class 'float'>
```

1.3. Função gauss()

A função gauss() gera números aleatórios decimais (float) de acordo com a distribuição gaussiana, onde números perto da média têm maior probabilidade de ocorrência. Essa função recebe como parâmetro a média e o desvio padrão. No exemplo abaixo os valores escolhidos foram 10 para média e 30 para o desvio padrão.

Exemplo:

```python
import random

valor_dist_normal = random.gauss(10, 30)
print(f"O número aleatório gerado foi => {valor_dist_normal}")
print(f"O tipo do  número aleatório gerado foi => {valor_dist_normal}")
```

Resposta:

```
>>  O número aleatório gerado foi => 14.934131739313187
>>  O tipo do  número aleatório gerado foi => <class 'float'>
```

1.4. Função choice()

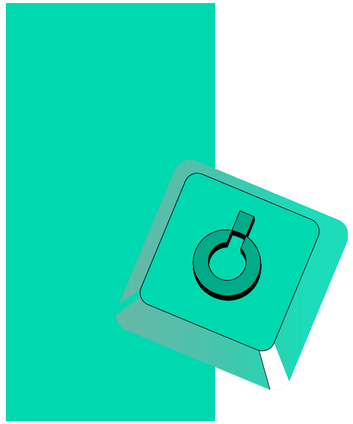
A função choice() escolhe com igual probabilidade, dentro do intervalo de uma lista, um item dentre as opções possíveis. Recebe como parâmetro uma lista e, no exemplo abaixo, a lista escolhida foi a 'tickers'.

Exemplo:

```python
from random import choice

tickers = ['WEGE3', 'PCAR3', 'LREN3']
empresa_escolhida = choice(tickers)

print(f"Escolheu dentro do intervalo 'tickers' a e empresa => {empresa_escolhida}")
```

Resposta:

```
>>  Escolheu dentro do intervalo 'tickers' a e empresa => PCAR3
```



Mundo 10

1. O que são coleções

Coleções dentro do python permitem armazenar infinitos itens dentro de uma variável, ou seja, será apenas um container onde poderá armazenar o tipo de informação que quiser. Essa informação pode ser números, letras, palavras, objetos, dataframes, listas, tuplas, dicionários e etc.. Calma, veremos todos esses itens ainda neste curso.

2. Tipos de coleções

Nesse mundo abordaremos 4 tipos de coleções, listas, tuplas, sets e dicionários. Entraremos a fundo em todas elas, para que servem e como utilizá-las.

2.1. Tuplas

As tuplas são parecidas com as listas (veremos a frente) e, por se tratar de um tipo de coleção, servem para guardar elementos. Dentro de uma mesma tupla podem existir elementos de diferentes características, números, strings e até mesmo outras tuplas.

Elas podem ser ordenadas, acessadas pelo index e utilizadas como forma de iteração de loop. Mas diferentemente das listas, elas são inalteráveis, ou seja, uma vez criadas elas viram objetos imutáveis no código. Para acessar o item de uma tupla utiliza-se colchetes. É importante ressaltar que o index sempre começa em 0, então caso você queira acessar o primeiro item de uma tupla: tupla[0]. Os itens da tupla são separados por vírgulas.

Exemplo:

```
tuple1 = ("codigo.py", 1)
tuple2 = ("Melhor curso", 2, tuple1)
tuple3 = ()

print(f"Essa é a tuple 1 => {tuple1}")
print(f"Esse é o tipo da tuple 1 => {type(tuple1)}")
print(f"Esse é a posição 0 da tuple 1 => {tuple1[0]}")

print(f"Essa é a tuple 2=> {tuple2}")
print(f"Esse é o tipo da tuple 2 => {type(tuple2)}")
print(f"Esse é a posição 1 da tuple 2 => {tuple2[1]}")

print(f"Essa é a tuple 3=> {tuple3}")
print(f"Esse é o tipo da tuple 3 => {type(tuple3)}")
```

Resposta:

```
>> Essa é a tuple 1 => ('codigo.py', 1)
>> Esse é o tipo da tuple 1 => <class 'tuple'>
>> Esse é a posição 0 da tuple 1 => codigo.py

>> Essa é a tuple 2=> ('Melhor curso', 2, ('codigo.py', 1))
>> Esse é o tipo da tuple 2 => <class 'tuple'>
>> Esse é a posição 1 da tuple 2 => 2

>> Essa é a tuple 3=> ()
>> Esse é o tipo da tuple 3 => <class 'tuple'>
```

2.2. Listas

As listas são parecidas com as tuplas, por se tratar de um tipo de coleção, servem para guardar elementos. Dentro de uma mesma lista podem existir elementos de diferentes características, números, strings, tuplas e até mesmo outras listas.

Elas podem ser ordenadas, acessadas pelo index e utilizadas como forma de interação de loop. Mas diferentemente das tuplas, elas podem ser alteradas. É comum adicionar, remover e atualizar dados dentro de uma lista e, dessa forma, as listas são mais utilizadas do que as tuplas. Nós podemos utiliza-lás para manipulação de dados e criação de dataframes (veremos a frente). Para acessar o item de uma lista utilizam-se colchetes e, é importante ressaltar mais uma vez, que o index sempre começa em 0, então caso você queira acessar o primeiro item de uma lista: lista[0]. Os itens da lista são separados por vírgulas.

Exemplo:

```
tuple1 = ("codigo.py", 1)
lista1 = ["Melhor curso", 2, tuple1]
lista2 = ["Primeiro lançamento", lista1]
lista3 = list()

print(f"Essa é a lista 1 => {lista1}")
print(f"Esse é o tipo da lista 1 => {type(lista1)}")
print(f"Esse é a posição 0 da lista 1 => {lista1[0]}")

print(f"Essa é a lista 2=> {lista2}")
print(f"Esse é o tipo da lista 2 => {type(lista2)}")
print(f"Esse é a posição 1 da lista 2 => {lista2[1]}")

print(f"Essa é a lista 3=> {lista3}")
print(f"Esse é o tipo da lista 3 => {type(lista3)}")
```

Resposta:

```
>> Essa é a lista 1 => [‘Melhor curso’, 2, (‘codigo.py’, 1)]
>> Esse é o tipo da lista 1 => <class ‘list’>
>> Esse é a posição 0 da lista 1 => Melhor curso

>> Essa é a lista 2=> [‘Primeiro lançamento’, [‘Melhor curso’, 2, (‘codi-
go.py’, 1)]]
>> Esse é o tipo da lista 2 => <class ‘list’>
>> Esse é a posição 1 da lista 2 => [‘Melhor curso’, 2, (‘codigo.py’, 1)]

>> Essa é a lista 3=> []
>> Esse é o tipo da lista 3 => <class ‘list’>
```

2.3. Sets

Assim como os dicionários (veremos a frente), os sets também são criados a partir de chaves e servem para guardar elementos, mas possuem características um pouco diferentes. Os itens do set não podem ser ordenados, não podem ser editados e não podem ter duplicatas.

Essa última característica se torna muito útil, quando existe uma lista enorme com muitos itens que não podem se repetir. O objetivo é jogar essa lista para o tipo set, e depois esse set para o tipo lista, para que os itens repetidos sumam. Os sets são a melhor ferramenta, quando se trata de união e interseção de vetores.

Exemplo1:

```
#retirando elementos repetidos
lista = [1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 5, 6, 7, 7, 8, 8, 8, 8, 8, 8, 9, 9, 9]
print(f"Essa é a lista => {lista}")
print(f"Esse é o tipo da lista {type(lista)}")

set = set(lista)
print(f"Essa é a lista depois do set => {set}")
print(f"Esse é o tipo depois do set => {type(set)}")

lista_nova = list(set)
print(f"Essa é a lista_nova => {lista_nova}")
print(f"Esse é o tipo da lista_nova => {type(lista_nova)}")
```

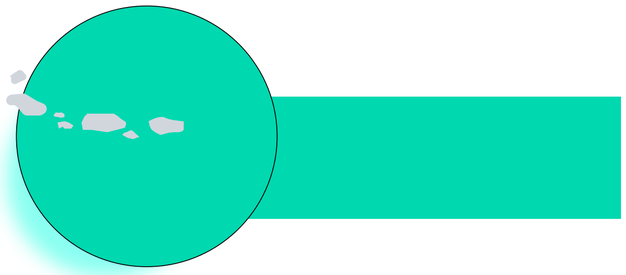
Resposta1:

```
>> Essa é a lista => [1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 5, 6, 7, 7, 8, 8, 8,
8, 8, 8, 9, 9, 9]
>> Esse é o tipo da lista <class 'list'>

>> Essa é a lista depois do set => {1, 2, 3, 4, 5, 6, 7, 8, 9}
>> Esse é o tipo depois do set => <class 'set'>

>> Essa é a lista_nova => [1, 2, 3, 4, 5, 6, 7, 8, 9]
>> Esse é o tipo da lista_nova => <class 'list'>
```

Abaixo pode-se observar que um set pode receber números, strings, listas, dicionários e outros sets também, sendo que os três últimos precisam ser entregues em formato de string. Sua estrutura é sempre entre chaves, pode ter um formato mais simples ou mais complexo (com sets dentro de sets). Uma de suas principais características é que não dá para acessar pelo index, pois eles não possuem posições fixas.

Exemplo2:

```python
#retirando elementos repetidos
lista = ["VZA", "VBOX", "codigo.py"]
dicionario = {"Curso": "Codigo.py"}
set1 = {"Brenno", "Sullivan"}
set2 = {"Sabe o melhor curso??", "Sabe a melhor empresa?", str(lista), str(dicionario)}


print(f"Essa é a lista => {lista}")
print(f"Esse é o tipo da lista => {type(lista)}")

print(f"Esse é o dicionário => {dicionario}")
print(f"Esse é o tipo do dicionário => {type(dicionario)}")

print(f"Esse é o set1 => {set1}")
print(f"Esse é o tipo do set1 => {type(set1)}")

print(f"Esse é o set2 => {set2}")
print(f"Esse é o tipo do set2 => {type(set2)}")
```



Resposta2:

```
>> Essa é a lista => ['VZA', 'VBOX', 'codigo.py']
>> Esse é o tipo da lista => <class 'list'>

>> Esse é o dicionário => {'Curso': 'Codigo.py'}
>> Esse é o tipo do dicionário => <class 'dict'>

>> Esse é o set1 => {'Sullivan', 'Brenno'}
>> Esse é o tipo do set1 => <class 'set'>

>> Esse é o set2 => {'Sabe o melhor curso??', 'Sabe a melhor empresa?',
"{'Curso': 'Codigo.py'}", "['VZA', 'VBOX', 'codigo.py']"}
>> Esse é o tipo do set2 => <class 'set'>
```

2.4. Dicionários

A estrutura do dicionário e dos sets são muito parecidas, porém o dicionário é mais utilizado por ter algumas funcionalidades a mais. É uma peça fundamental do python e, devido a sua estrutura, pode ser utilizado em vários contextos. Alguns deles são transformar os dados em tabelas e transformar os dados em JSON (abordaremos já). Existem também bancos de dados (NoSQL) que utilizam esse formato para armazenamento de dados, como se fosse uma enorme lista contendo várias outras listas organizadas. Por se tratar de um formato simples, esse tipo de banco de dados costuma consumir menos memória e ter mais escalabilidade.

Além disso, eles funcionam com um sistema de chaves primárias ou identificadores, assim como um banco de dados, e cada item é constituído de duas entradas: chave e valor, tudo separado por 2 pontos. Criando um dicionário de uma empresa ficaria desse jeito:

```
empresa = {"nome": "Edufinance", "sede": "Niterói"}
```

Repare que estamos atribuindo à variável "empresa" um "banco de dados" próprio. Em algumas ocasiões será mais interessante utilizar o dicionário, por isso é importante que entenda bem essa parte.

Lembrando que qualquer string dentro do python deve ser acompanhado de aspas, e os nomes dos itens não são diferentes. A característica dos itens, que vem depois dos dois pontos podem ser qualquer coisa: booleano, inteiro, número decimal, fora que eles também podem se repetir ao longo do dicionário.

Entretanto, as chaves, que vem antes dos dois pontos, devem ser únicas e identificáveis, embora também possam ser de qualquer categoria. É por isso que dicionários podem servir como banco de dados: essas chaves únicas garantem a integridade e a especificidade do dado.

É importante reforçar também, que os dicionários podem receber qualquer tipo de dado, strings, números, tuplas, listas e dicionários.

Exemplo1:

```
lista = ["Melhor cidade", "Niteroi"]
tupla = ("Melhor curso", "codigo.py")
set = {"Melhor empresa", "edufinance"}
dicionario1 = {"Melhor professor": "Brenno Sullivan"}
dicionario2 = {"Instagram": "@brenno.edufinance"}

formulario = dict()
formulario = {"Avaliação 1": lista, "Avaliação 2": tupla,
              "Avaliação 3": set, "Avaliação 4": dicionario1, "Avaliação 5": dicionario2}

print(formulario)
```

Resposta1:

```
>> {'Avaliação 1': ['Melhor cidade', 'Niteroi'], 'Avaliação 2': ('Me-
lhor curso', 'codigo.py'), 'Avaliação 3': {'Melhor empresa', 'edufinance'},
'Avaliação 4': {'Melhor professor': 'Brenno Sullivan'}, 'Avaliação 5':
{'Instagram': '@brenno.edufinance'}}
```

Mundo 11

1. Manipulação de listas

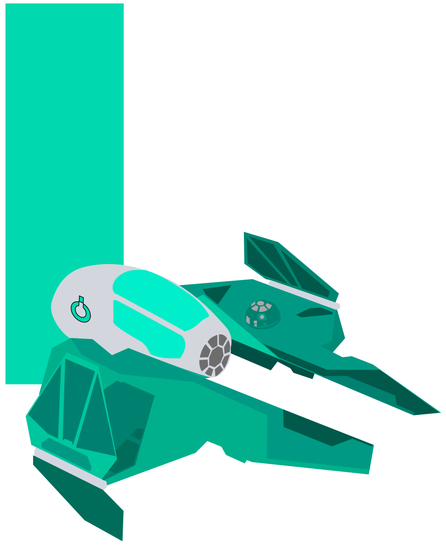
1.1. Criando listas

Exemplo:

```
lista_base = ['Weg', 'Raia drogasil', 'Vale', 'Lojas Renner']
lista_tipos_diferentes = list(['flamengo', 8, 'campeonatos brasileiros'])

print(f"Essa é a lista_base => {lista_base}")
print(f"Essa é a lista_tipos_diferentes {lista_tipos_diferentes}")
```

Resposta:

```
>>  Essa é a lista_base => ['Weg', 'Raia drogasil', 'Vale', 'Lojas Ren-
ner']
>>  Essa é a lista_tipos_diferentes => ['flamengo', 8, 'campeonatos brasi-
leiros']
```



1.2. Percorrendo lista

No exemplo abaixo, é feita a criação de 3 listas dentro de uma única lista. Essa lista, que agrega outras, pode ser considerada uma matriz 3x3. Uma das funções que irão te auxiliar muito, que veremos a frente, é a função len(). Ela nos retorna o tamanho da lista.

Exemplo:

```
lista_dentro_de_lista = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]  # matriz 3 x 3
print(lista_dentro_de_lista)
print(f"O tamanho da lista é => {len(lista_dentro_de_lista)}") #
```

Resposta:

```
>>  [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>> O tamanho da lista é => 3
```

Podemos usar uma estrutura de repetição (veremos a frente) para mexer apenas na parte visual, por meio da função print(). No nosso exemplo, utilizaremos a estrutura de repetição “for”.

1.2.1. Percorrendo a lista_dentro_de_lista

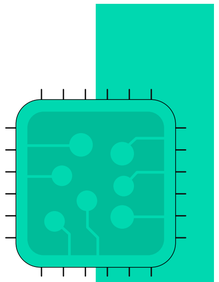
Pense que o “for” significa “para cada” e o “in” significa “dentro”. Então o código abaixo poderia ser lido como “para cada lista dentro da “lista_dentro_de_lista”. Como a nossa lista tem tamanho 3, ela percorrerá as 3 posições uma de cada vez. Isso será retomado mais à frente, mas é importante saber que no lugar de “lista”, você poderia nomear qualquer coisa, isso é apenas um parâmetro que recebe um nome qualquer.

Exemplo:

```
lista_dentro_de_lista = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]  # matriz 3 x 3

for lista in lista_dentro_de_lista:
    print(lista)
```

Resposta:

```
>> [1, 2, 3]
   [4, 5, 6]
   [7, 8, 9]
```



1.2.2. Melhorando a visualização

Apesar da visualização já ter melhorado, dá para ficar melhor. Desta vez, utilizaremos uma estrutura de repetição dentro de outra estrutura de repetição. Por se tratar de listas dentro de uma lista, conseguimos acessar todos os itens de cada lista e o objetivo aqui é imprimir na tela cada número de cada lista. Vamos fazer isso para as três listas e, ao final de cada item imprimido, adicionaremos um espaço “ “, para melhorar a visualização. Então ficaria assim:

Exemplo:

```
lista_dentro_de_lista = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]  # matriz 3 x 3

for lista in lista_dentro_de_lista:
    for item in lista:
        print(item, end=' ')
    print()
```

Resposta:

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

1.3. Adicionando itens

1.3.1. Função append()

Essa função recebe como parâmetro o elemento escolhido e o adiciona ao final da lista.

Exemplo:

```
lista_sinais = ['compra tudo','vende um pouco','vende sua casa para comprar']
print(f"A lista antes da função append => {lista_sinais}")

lista_sinais.append("compra devagar")
print(f"A lista depois da função append => {lista_sinais}")
```

Resposta:

```
>> A lista antes da função append => [‘compra tudo’, ‘vende um pouco’,
‘vende sua casa para comprar’]
>> A lista depois da função append => [‘compra tudo’, ‘vende um pouco’,
‘vende sua casa para comprar’, ‘compra devagar’]
```

1.3.2. Função insert()

Essa função recebe como parâmetro a posição que deseja inserir e o elemento escolhido. Com isso, adiciona o item na posição desejada e vale relembrar que toda contagem dentro do python começa em 0.

Exemplo:

```
lista_sinais = ['compra tudo','vende um pouco','vende sua casa para comprar']
print(f"A lista antes da função append => {lista_sinais}")

lista_sinais.insert(2, "compra até acabar o caixa")
print(f"A lista depois da função append => {lista_sinais}")
```

Resposta:

```
>> A lista antes da função append => [‘compra tudo’, ‘vende um pouco’,
‘vende sua casa para comprar’]
>> A lista depois da função append => [‘compra tudo’, ‘vende um pouco’,
‘compra até acabar o caixa’, ‘vende sua casa para comprar’]
```

1.4. Selecionando itens

1.4.1. Selecionando um único elemento na lista

A seleção de elementos de uma lista é feita através dos colchetes, que recebem como parâmetro a posição desejada. Ao digitar a posição [0] em uma lista, estamos nos referindo à primeira posição. Enquanto a posição [-1] está sendo referida como a última posição.

Exemplo:

```
lista_sinais = ['compra tudo','vende um pouco','vende sua casa para comprar','compra devagar',
'vende rápido ou vai quebrar','compra mais']

print(lista_sinais[0])
print(lista_sinais[1])
print(lista_sinais[-1])
```

Resposta:

```
>> compra tudo
>> vende um pouco
>> compra mais
```

Há também a possibilidade de acessar itens de uma lista que está dentro de outra lista. Nesse caso, vamos associar a posição da lista desejada dentro da lista, e depois associar à posição do elemento dentro da lista acessada.

Exemplo:

```
lista_dentro_de_lista = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

print(f"A lista desejada é => {lista_dentro_de_lista[0]}")
print(f"O primeiro item da lista desejada é => {lista_dentro_de_lista[0][0]}")
```

Resposta:

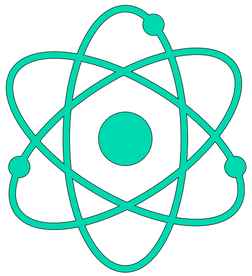
```
>> A lista desejada é => [1, 2, 3]
>> O primeiro item da lista desejada é => 1
```

1.4.2. Selecionando múltiplos elementos de uma lista

Para selecionar vários itens de uma vez, utilizaremos o seguinte formato: lista[x : y]. Onde x é a posição inicial que você deseja e y posição final, sendo que a posição y não está incluída. Fazendo um parâmetro com a matemática, é como se a seleção fosse aberta no final: [1, 2, 3[. Caso deseje selecionar até a posição 3, você deve referenciar a posição como uma a mais, ou seja, nesse caso posição 4.

Exemplo:

```
lista_sinais = ["1º", "2º", "3º", "4º", "5º", "6º", "7º"]

print(f" Da primeira até a segunda posição => {lista_sinais[0:2]}")
print(f" Da primeira posição até o limite => {lista_sinais[1:]}")
print(f" Do limite inicial até a terceira posição => {lista_sinais[:3]}")
print(f" Das duas últimas posições até o final => {lista_sinais[:-2]}")
```



Resposta:

```
>> Das duas ultimas posições até o final => ['1º', '2º', '3º', '4º', '5º']
```

1.5. Alterando itens

Alterar um item de uma lista é muito parecido com selecionar um elemento, a diferença é que você vai igualar essa posição ao objeto que deseja.

Exemplo:

```
lista_sinais = ["1º", "2º", "3º", "4º", "5º", "6º", "7º"]
lista_sinais[0] = 'vende'

print(lista_sinais)
```

Resposta:

```
>> ['vende', '2º', '3º', '4º', '5º', '6º', '7º']
```

1.6. Removendo itens da lista

1.6.1. Função remove()

Essa fórmula recebe como parâmetro o elemento escolhido para excluir e, caso esse elemento não esteja no intervalo da lista, aparecerá um erro dizendo que esse item não está na lista.

Exemplo:

```
lista_sinais = ["1°", "2°", "3°", "4°", "5°", "6°", "7°"]
lista_sinais.remove("1°")

print(lista_sinais)
```

Resposta:

```
>> ['2°', '3°', '4°', '5°', '6°', '7°']
```

1.6.2. Função del

Essa fórmula recebe como parâmetro a posição do item e a lista, que deseja que seja excluído.

Exemplo:

```
lista_sinais = ["1°", "2°", "3°", "4°", "5°", "6°", "7°"]
del lista_sinais[0]

print(lista_sinais)
```

Resposta:

```
>> ['2°', '3°', '4°', '5°', '6°', '7°']
```

1.7. Verificando itens na lista

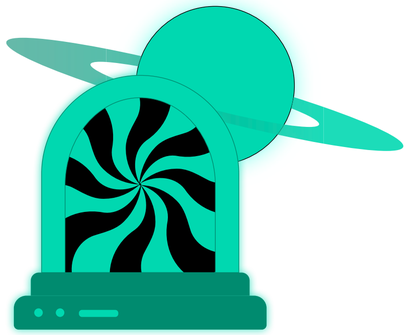
In e not in são verificações que indicam se um item está, ou não, em uma lista. Se estiver na lista, essa verificação retornará True, se não estiver, retornará False. Essa condição recebe como parâmetro o item que está sendo procurado procurado e a lista verificada.

Exemplo:

```
lista_sinais = ["1°", "2°", "3°", "4°", "5°", "6°", "7°"]

verificando = "1°" in lista_sinais
verificando2 = "1°" not in lista_sinais

print(verificando, verificando2)
```

Resposta:

```
>> True False
```



1.8. Juntando listas

Para juntar duas listas é só somar uma com a outra. Diferentemente de vetores, os valores da lista não serão modificados.

Exemplo:

```
lista1 = ["1°", "2°", "3°"]
lista2 = ["4°", "5°", "6°", "7°"]

juntando_listas = lista1 + lista2

print(juntando_listas)
```

Resposta:

```
>> ['1°', '2°', '3°', '4°', '5°', '6°', '7°']
```

1.9. Ordenando listas

1.9.1. Função sorted() - Ordenando lista por ordem crescente

A função sorted() recebe como parâmetro a lista que deseja ordenar.

Exemplo:

```
lista_desordenada = [56, 45, 1, 75, 23]
lista_ordenada_crescente = sorted(lista_desordenada)

print(lista_ordenada_crescente)
```

Resposta:

```
>> [1, 23, 45, 56, 75]
```

1.9.2. Função sorted() - Ordenando lista por ordem decrescente

A função sorted(lista, reverse=True) recebe como parâmetro a lista que deseja ordenar e o parâmetro fixo reverse=True.

Exemplo:

```
lista_desordenada = [56, 45, 1, 75, 23]
lista_ordenada_decrescente = sorted(lista_desordenada, reverse=True)

print(lista_ordenada_decrescente)
```

Resposta:

```
>> [75, 56, 45, 23, 1]
```

2. Informações da lista

2.1. Função len()

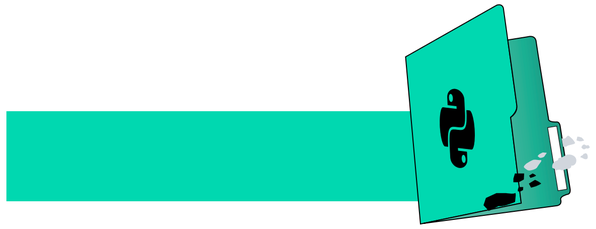
Essa função recebe como parâmetro a lista, e retorna o tamanho dela.

Exemplo:

```
lista_desordenada = [56, 45, 1, 75, 23]

print(f"O tamanho da lista é => {len(lista_desordenada)}")
```

Resposta:

```
>> O tamanho da lista é => 5
```



2.2. Função max()

Essa função recebe como parâmetro a lista, e retorna o maior valor dela.

Exemplo:

```
lista_desordenada = [56, 45, 1, 75, 23]

print(f"O maior valor da lista é => {max(lista_desordenada)}")
```

Resposta:

```
>> O maior valor da lista é => 75
```

2.3. Função min()

Essa função recebe como parâmetro a lista, e retorna o menor valor dela.

Exemplo:

```
lista_desordenada = [56, 45, 1, 75, 23]

print(f"O menor valor da lista é => {min(lista_desordenada)}")
```

Resposta:

```
>> O maior valor da lista é => 1
```

2.4. Função sum()

Essa função recebe como parâmetro a lista, e retorna a soma de todos os itens.

Exemplo:

```
lista_desordenada = [56, 45, 1, 75, 23]

print(f"A soma dos itens da lista é => {sum(lista_desordenada)}")
```

Resposta:

```
>> A soma dos itens da lista é => 200
```

3. Manipulação de tuplas

3.1. Função count()

Essa função recebe como parâmetro o valor que deseja e retorna a quantidade de vezes que o valor aparece dentro da tupla.

Exemplo:

```
tupla = (2, 3, 5, 2, 2)
valor_desejado = 2

print(f"A quantidade de {valor_desejado} dentro da tupla é =>{tupla.count(valor_desejado)}")
```

Resposta:

```
>> A quantidade de 2 dentro da tupla é => 3
```

3.2. Função index()

Essa função recebe como parâmetro o valor e retorna qual posição que o valor aparece na tupla.

Exemplo:

```
tupla = (2, 3, 5, 2, 2)
valor_desejado = 2

print(f'''A  primeira posição que o valor 2 aparece é
=> {tupla.index(valor_desejado)}''')
```

Resposta:

```
>> A primeira posição que o valor 2 aparece é => 0
```

3.3. Função len()

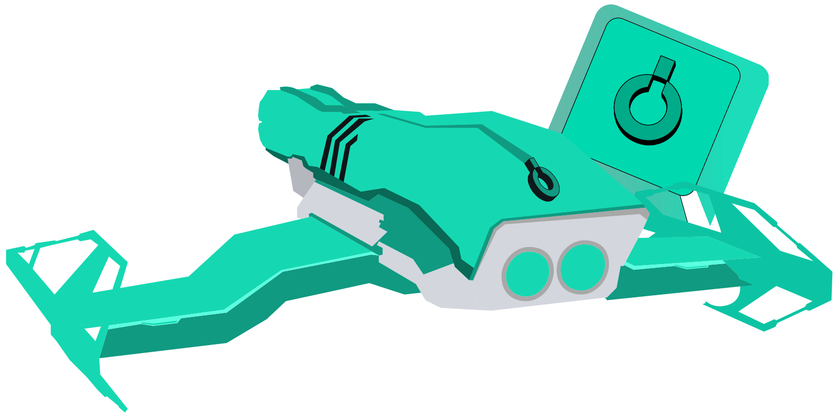
Essa função recebe como parâmetro a tupla e retorna o tamanho dela.

Exemplo:

```
tupla = (2, 3, 5, 2, 2)

print(f"O tamanho da tupla é => {len(tupla)}")
```

Resposta:

```
>> O tamanho da tupla é => 5
```



Mundo 12

1. Manipulação de sets

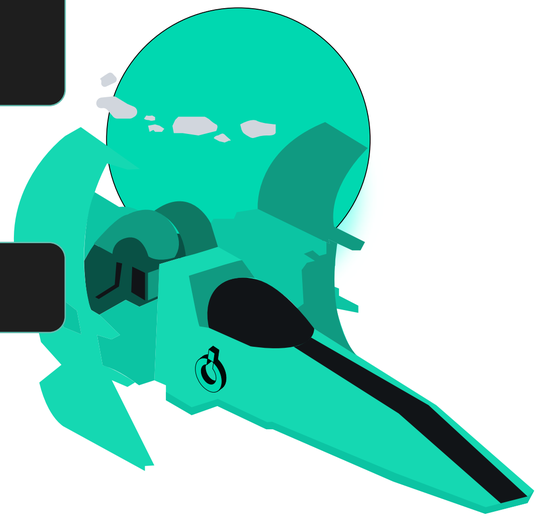
1.1. Criando sets

Exemplo:

```
set_teste = {'weg', 'renner', 'c&a', 'SLC agricola'}
set_numero = {1, 4, 5, 8}

print(f"Esse é o set_teste => {set_teste}")
print(f"Esse é o set_numero => {set_numero}")
```

Resposta:

```
>>  Esse é o set_teste => {'SLC agricola', 'renner', 'c&a', 'weg'}
>>  Esse é o set_numero => {8, 1, 4, 5}
```



1.2. Removendo duplicatas do set

Vale relembrar que os sets, por padrão, não aceitam valores duplicados, então quando uma lista com duplicatas é transformada em set, o python elimina todos os valores duplicados. Depois é só retornar ao tipo que deseja, no caso abaixo é a lista.

Exemplo:

```
lista_duplicada = ['renner', 'renner', 'weg', 'weg', 'weg', 'renner', 'renner', 'petrobras']
lista_unica = list(set(lista_duplicada)) #você pode converter listas em sets ou tuplas sem problemas

print(f"Lista sem duplicatas => {lista_unica}")
```

Resposta:

```
>>  Lista sem duplicatas => ['petrobras', 'renner', 'weg']}
```

1.3. Adicionando itens ao set

1.3.1. Função add()

Essa função recebe como parâmetro o valor que deseja adicionar e retorna o set com o item adicionado ao fim.

Exemplo:

```
set_numero = {1, 4, 5, 8}
set_numero.add(10)

print(f" Set com o valor adicionado => {set_numero}")
```

Resposta:

```
>>  Set com o valor adicionado => {1, 4, 5, 8, 10}
```

1.3.2. Função update()

Essa função recebe como parâmetro os valores que deseja adicionar e retorna o set com os itens adicionados ao fim.

Exemplo:

```
set_numero = {1, 4, 5, 8}
set_numero.update([20, 25, 60])

print(f" Set com os valores adicionados => {set_numero}")
```

Resposta:

```
>>  Set com os valores adicionados => {1, 4, 5, 8, 20, 25, 60}
```

1.4. Removendo itens do set

1.4.1. Função discard()

Essa função recebe como parâmetro o valor que deseja excluir e retorna o set sem o valor. Essa função vai deletar todas as ocorrências do valor escolhido dentro do set.

Exemplo:

```
set_numero = {1, 4, 5, 8}
set_numero.discard(1)

print(f" Set com os valores removidos => {set_numero}")
```

Resposta:

```
>>  Set com os valores adicionados => {8, 4, 5}
```

2. Operações matemáticas com sets

2.1. União

Por definição matemática, a união de dois conjuntos é o conjunto de elementos que aparecem pelo menos uma vez. Traduzindo de uma forma mais simples, são todos os números que aparecem nos conjuntos, sem valores duplos.

2.1.1 Função union()

Essa função recebe como parâmetro os dois sets no qual deseja fazer a união e retorna o set com o valor da união dos dois.

Exemplo:

```
meu_set = {1, 2, 3, 5, 8, 9}
meu_set_2 = {1, 4, 5, 6, 8, 11, 12}
# União
print(meu_set.union(meu_set_2))
```

Resposta:

```
>>  A união dos dois conjuntos é => {1, 2, 3, 4, 5, 6, 8, 9, 11, 12}
```

2.1.2. set | set2

Esse sinal recebe como parâmetro os dois sets, cada um de um lado e retorna o set com valores da união entre eles.

Exemplo:

```
meu_set = {1, 2, 3, 5, 8, 9}
meu_set_2 = {1, 4, 5, 6, 8, 11, 12}
# União
print(f" A união dos dois conjuntos é => {meu_set | meu_set_2}")
```

Resposta:

```
>>  A união dos dois conjuntos é => {1, 2, 3, 4, 5, 6, 8, 9, 11, 12}
```

2.2. Interseção

Por definição matemática, a interseção de dois conjuntos é o conjunto de elementos que simultaneamente aparece entre os dois, ou seja, é o conjunto de dados que aparece nos dois conjuntos, sem valores duplos.

2.1.1. Função intersection()

Essa função recebe como parâmetro os dois sets no qual deseja fazer a interseção e retorna o set com os valores da interseção dos dois.

Exemplo:

```
meu_set = {1, 2, 3, 5, 8, 9}
meu_set_2 = {1, 4, 5, 6, 8, 11, 12}
# Interseção
print(f" A interseção dos dois conjuntos é => {meu_set.intersection(meu_set_2)}")
```

Resposta:

```
>>  A interseção dos dois conjuntos é => {8, 1, 5}
```

2.1.2. set & set2

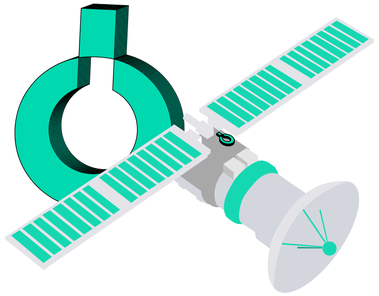
Esse sinal recebe como parâmetro os dois sets, cada um de um lado e retorna o set com os valores da interseção entre eles.

Exemplo:

```
meu_set = {1, 2, 3, 5, 8, 9}
meu_set_2 = {1, 4, 5, 6, 8, 11, 12}
# Interseção
print(f" A interseção dos dois conjuntos é => {meu_set & meu_set_2}")
```

Resposta:

```
>>  A interseção dos dois conjuntos é => {8, 1, 5}
```



2.3. Diferença

Por definição matemática, a diferença de dois conjuntos é o conjunto de elementos que pertence ao primeiro conjunto e não pertence ao segundo conjunto.

2.3.1. Função difference()

Essa função recebe como parâmetro os dois sets, no qual deseja fazer a diferença e retorna o set com os valores que pertencem ao primeiro conjunto, mas não pertencem ao segundo.

Exemplo:

```
meu_set = {1, 2, 3, 5, 8, 9}
meu_set_2 = {1, 4, 5, 6, 8, 11, 12}
# Diferença
print(f" A interseção dos dois conjuntos é => {meu_set.difference(meu_set_2)}")
```

Resposta:

```
>>  A interseção dos dois conjuntos é => {8, 1, 5}
```

2.3.2. set - set2

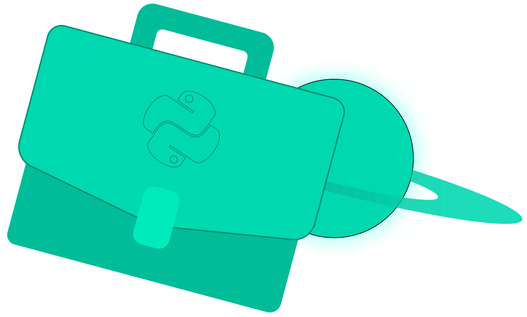
Esse sinal recebe como parâmetro os dois sets, cada um de um lado e retorna o set com os valores da diferença entre eles.

Exemplo:

```
meu_set = {1, 2, 3, 5, 8, 9}
meu_set_2 = {1, 4, 5, 6, 8, 11, 12}
# Interseção
print(f" A interseção dos dois conjuntos é => {meu_set - meu_set_2}")
```

Resposta:

```
>>  A interseção dos dois conjuntos é => {9, 2, 3}
```



Mundo 13

1. Manipulação de dicionários

Dicionários são conjuntos de dados que possuem uma chave e um valor. São facilmente manipuláveis, sua sintaxe básica é {"chave" : valor}. Esse valor pode ser um número, uma string, uma tupla, uma lista e um dicionário.

1.1. Criando dicionários

Exemplo:

```
dicionario_vazio = {}
dicionario_povoado = dict({'nome': 'brenno', 'sobrenome': 'sullivan'})
dicionario_numerico = {1: 'brenno', 2: 'lucas', 3: 'leandro'} #pódio
dicionario_com_listas = {'empresas_novo_mercado':['Weg', 'Renner', 'Vale'],
                         'empresas_em_outros_segmentos': ['Petrobras', 'Alpargatas']}
dicionario_empresas_na_carteira = {True: ['Pão de Açucar', 'Weg'],
                                   False: ['Lojas Renner', 'Guararapes']}

print(f"Esse é o dicionario_vazio que contém => {dicionario_vazio}")
print(f"Esse é o dicionario_povoado que contém => {dicionario_povoado}")
print(f"Esse é o dicionario_com_listas que contém => {dicionario_com_listas}")
print(f"Esse é o dicionario_empresas_na_carteira que contém => {dicionario_empresas_na_carteira}")
```

Resposta:

```
>>  Esse é o dicionario_vazio que contém => {}
>>  Esse é o dicionario_povoado que contém => {'nome': 'bren-
no', 'sobrenome': 'sullivan', 'sobrenome': 'sullivan'}'}
egmentos': ['Petrobras', 'Alpargatas']} que contém => {'empresas_novo_mer-
cado': ['Weg', 'Renner', 'Vale'], 'empresas_em_outros_segmento
>>  Esse é o dicionario_com_listas que contém => {'empresas_novo_mercado':
['Weg', 'Renner', 'Vale'], 'empresas_em_outros_segmentos': ['Petrobras',
'Alpargatas']}        e contém => {True: ['Pão de Açucar', 'Weg'], False:
['Lojas Renner', 'Guararapes']}
>>  Esse é o dicionario_empresas_na_carteira que contém => {True: ['Pão de
Açucar', 'Weg'], False: ['Lojas Renner', 'Guararapes']
```

1.2. Pegando itens dentro do dicionário

Existem algumas formas de pegar informações dentro de um dicionário, tudo depende da forma que este está armazenado.

1.2.1. Função get()

Essa função recebe como parâmetro a chave especificada pelo nome e retorna ao valor correspondente.

Exemplo:

```
dicionario_numerico = {"Professor": 'Brenno', 'Empresa': 'Edufinance'}

print(f"O valor correspondente a chave Empresa é => {dicionario_numerico.get('Empresa')}")
```

Resposta:

```
>>  O valor correspondente a chave Empresa é => edufinance
```

1.2.2. Pegando itens a partir da seleção

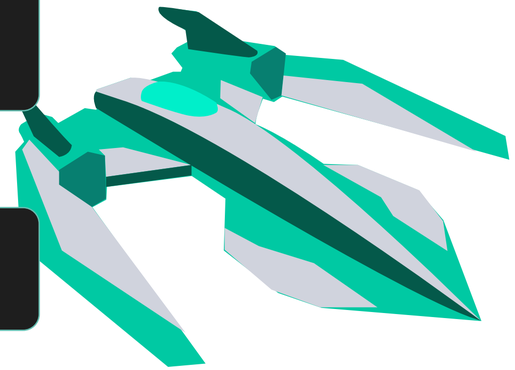
Esse método recebe como parâmetro o nome da chave e o valor desejado entre colchetes. É importante lembrar que toda string é representada pelo python entre aspas.

Exemplo:

```
dicionario_numerico = {"Professor": 'Brenno', 'Empresa': 'Edufinance'}

print(f"O valor correspondente a chave Empresa é => {dicionario_numerico['empresas_novo_mercado']}")
```

Resposta:

```
>>  O valor correspondente a chave Empresa é => edufinance
```



1.2.3. Utilizado índices juntos com seleção

Esse método recebe como parâmetro o nome da chave entre colchetes, e a posição na lista. Esse método acessa uma lista que está dentro de um dicionário

Exemplo:

```
dicionario_numerico = {"Professor": 'Brenno', 'Empresa': 'Edufinance','Produtos':
['Codigo.Py','VBOX','VZA']}

print(f'''O valor correspondente ao primeiro elemento da lista dentro do dicionário é =>
{dicionario_numerico['Produtos'][0]}''')
```

Resposta:

```
>>  O valor correspondente ao primeiro elemento da lista => Codigo.Py
```

1.3. Adicionando itens dentro do dicionário

1.3.1. Função update()

Essa função recebe como parâmetro a chave e o valor do objeto que deseja adicionar, podendo ser um ou mais objetos.

Exemplo:

```
dicionario_vazio = {}
valor = {'Galaxia': 2, 'Mundo': 13}
dicionario_vazio.update(valor)

print(f"O valor do dicionário após a atualização é  => {dicionario_vazio}")
```

Resposta:

```
>>  O valor do dicionário após a atualização é  => {'Galaxia': 2,
'Mundo': 13}
```

1.3.2. Atribuindo uma variável a uma chave

Esse método recebe atribui a uma chave um valor escolhido. Este método retornará um novo objeto ao dicionário existente

Exemplo:

```
dicionario_vazio = {}
dicionario_vazio['instagram'] = '@brenno.edufinance'

print(f"O valor do dicionário após a atualização é  => {dicionario_vazio}")
```

Resposta:

```
>>  O valor do dicionário após a atualização é  => {'instagram': '@brenno.
edufinance'}
```

1.4. Removendo itens dentro do dicionário

1.4.1. Função pop()

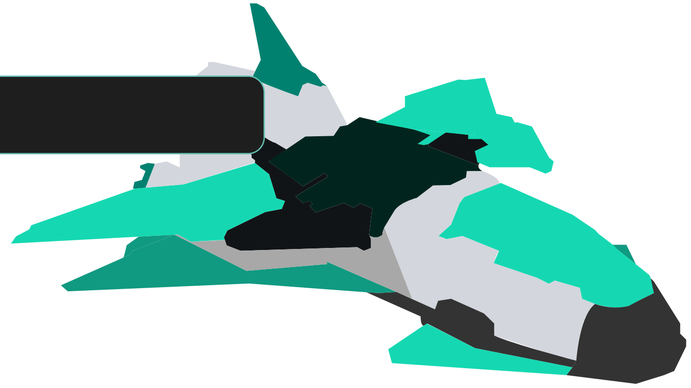
Essa função recebe como parâmetro a chave que deseja excluir.

Exemplo:

```
dicionario_vazio = {'instagram': '@brenno.edufinance'}
dicionario_vazio.pop('instagram')

print(f"O valor do dicionário após a atualização é  => {dicionario_vazio}")
```

Resposta:

```
>>  {}
```



2. Aplicabilidade dicionários

2.1. Utilização da biblioteca pandas

A biblioteca pandas é muito utilizada para tratamentos de dados. Ela tem diversas funções que se integram com dicionários. Uma delas é a transformação de um dicionário para um dataframe, facilitando muito na hora do tratamento e manipulação dos dados.

Esse exemplo abaixo é um dos exemplos de como utilizar essa biblioteca ao seu favor, mas ainda abordaremos esse módulo exclusivamente, o exemplo serve para você entender um pouco do poder do pandas.

Exemplo:

```
import pandas as pd
# tabela de cotações
dicionario_cotacoes = {'WEGE3': [50.10, 49.09, 30.06],
                       'PCAR3': [20.05, 19.06, 21.39]}
tabela_cotacoes = pd.DataFrame(data=dicionario_cotacoes,
index=pd.date_range(start="2022-07-05", end="2022-07-07"))

print(tabela_cotacoes)
```

Resposta:

```
>>             WEGE3  PCAR3
2022-07-05  50.10  20.05
2022-07-06  49.09  19.06
2022-07-07  30.06  21.39
```

Mundo 14

1. Manipulação de strings

Nesse mundo aprofundaremos os conhecimentos de manipulação de strings. A utilização da biblioteca "re" será de grande importância, principalmente se tratando de validação de dados.

1.1. Tamanho do Texto

1.1. Função len()

Essa função recebe como parâmetro o texto, e retorna o tamanho desse texto, ou seja, o número de caracteres e os espaços em branco.

Exemplo:

```
texto = 'esse é o curso código.py'
print(len(texto))
```

Resposta:

```
>>  O tamanho desse texto é => 24
```

1.2. Concatenando textos

1.2.1. Operador de soma (+)

Esse método utiliza o +, que na matemática tem função de soma. No texto não é muito diferente, ele adiciona uma string a outra.

Exemplo:

```
texto = 'esse é o curso código.py'
print(texto + " e eu sou um ótimo aluno!")
```

Resposta:

```
>>  esse é o curso código.py e eu sou um ótimo aluno!
```

1.3. Substituindo textos

1.3.1. Função replace()

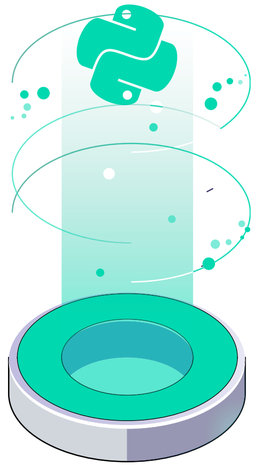
Essa função recebe como parâmetro o texto a ser substituído e o texto que substituirá.

Exemplo:

```
texto = 'esse é o curso código.py'
texto_novo = texto.replace("código.py", "Código.py 2.0")

print(f"O texto novo agora é => {texto_novo}")
```

Resposta:

```
>>  O texto novo agora é => esse é o curso Código.py 2.0
```



1.4. Contando texto

1.4.1. Função count()

Essa função recebe como parâmetro o texto a ser contado e retorna o número de vezes que ele aparece.

Exemplo:

```
texto = 'Esse é o curso código.py, o melhor curso do Brasil'
print(f"A quantidade de vezes que a palavra curso aparece é => {texto.count('curso')}")
```

Resposta:

```
>>  A quantidade de vezes que a palavra curso aparece é => 2
```

1.5. Contando textos que começam ou terminam com textos específicos

1.5.1. Função startswith()

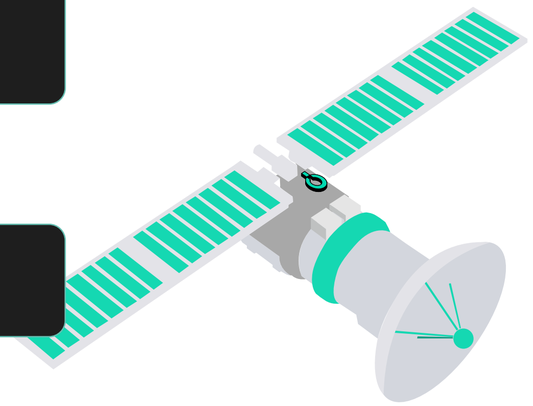
Essa função recebe como parâmetro um texto e verifica se a string a ser verificada começa ou não com o texto. Ele retorna um booleano (True ou False).

Exemplo:

```
texto = 'esse é o curso código.py'
texto2 = 'curso mais badalado do Brasil'

print(f'O texto começa com "curso" => {texto.startswith("curso")}')
print(f'O texto 2 começa com "curso" => {texto2.startswith("curso")}')
```

Resposta:

```
>>  O texto começa com "curso" => False
>>  O texto 2 começa com "curso" => True
```



1.5.2. Função endswith()

Essa função recebe como parâmetro um texto e verifica se a string a ser verificada termina ou não com o texto. Ele retorna um booleano (True ou False).

Exemplo:

```
texto = 'código.py esse é o curso'
texto2 = 'curso mais badalado do Brasil'

print(f'O texto termina com "curso" => {texto.endswith("curso")}')
print(f'O texto 2 termina com "curso" => {texto2.endswith("curso")}')
```

Resposta:

```
>>  O texto termina com "curso" => True
>>  O texto 2 termina com "curso" => False
```

1.6. Selecionando caracteres

1.6.1. String[x] - Selecionando caracteres únicos

Esse método recebe como parâmetro a posição do caractere que deseja selecionar. Vale relembrar alguma das posições:

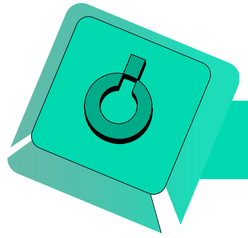
[-1] - Pega a última posição
[0] - Pega o primeiro valor

Exemplo:

```
texto = 'código.py esse é o curso'

print(f'Pegando o primeiro caracter da string => {texto[0]}')
print(f'Pegando o ultimo caracter da string => {texto[-1]}')
```

Resposta:

```
>>  Pegando o primeiro caractere da string => c
>>  Pegando o último caractere da string => o
```



1.6.2. String[x : y] - Selecionando intervalo de caracteres

Esse método recebe como parâmetro as posições iniciais e finais do intervalo que deseja selecionar. Vale relembrar que a última posição não está inclusa, deve sempre somar + 1.

Exemplo:

```
texto = 'código.py esse é o curso'

print(f'Pegando o intervalo da string => {texto[0:4]}')
print(f'Pegando o intervalo da string => {texto[3:8]}')
```

Resposta:

```
>>  Pegando o intervalo da string => códi
>>  Pegando o intervalo da string => igo.p
```

1.7. Modificando caracteres

1.7.1. Função upper()

Transforma todos os caracteres da string em letras maiúsculas. Não recebe nenhum parâmetro.

1.7.2. Função lower()

Transforma todos os caracteres da string em letras minúsculas. Não recebe nenhum parâmetro.

1.7.3. Função title()

Transforma os primeiros caracteres de cada palavra em letras maiúsculas. Não recebe nenhum parâmetro.

Exemplo:

```
texto = 'código.py esse é o curso'

print(f"Texto com tudo maiusculo => {texto.upper()}")
print(f"Texto com tudo minusculo => {texto.lower()}")
print(f"Texto só com as primeiras letras maiusculas => {texto.title()}")
```

Resposta:

```
>>  Texto com tudo maiusculo => CÓDIGO.PY ESSE É O CURSO
>>  Texto com tudo minusculo => código.py esse é o curso
>>  Texto só com as primeiras letras maiusculas => Código.py Esse É O Curso
```

1.8. Regex

1.8.1. Entendendo a estrutura do regex

O regex tende a ser uma ferramenta de validação. É muito poderosa, principalmente se tratando de validação e manipulação de textos. Veremos algumas das funcionalidades dela. Além das validações, ela também faz buscas e substituições de texto. Vamos considerar os seguintes textos:

1° - brennosullivan22081999@gmail.com

2° - lucasvgm@hotmail.com

3° - 123leandro456@yahoo.com

Primeira regra do regex: Tudo digitado entre colchetes no regex equivale a um caractere. Por exemplo, se digitarmos a letra "b", o regex retornará:

1° Passo = [b]

1° - brennosullivan22081999@gmail.com

2° - lucasvgm@hotmail.com

3° - 123leandro456@yahoo.com

Caso a gente queira digitar um ou mais caracteres, ele retorna todos os caracteres presentes dentro do colchete.

2° Passo = [bog]

1° - brennosullivan22081999@gmail.com

2° - lucasvgm@hotmail.com

3° - 123leandro456@yahoo.com

Não seria diferente se a gente trabalhasse com números.

3° Passo = [18]

1° - brennosullivan22081999@gmail.com

2° - lucasvgm@hotmail.com

3° - 123leandro456@yahoo.com

No regex também conseguimos trabalhar com intervalos, e eles são separados por "-". Se a gente quisesse um intervalo de "A" até" Z", ele responderia com todas as letras do alfabeto.

4° Passo = [a-z]

1° - brennosullivan22081999@gmail.com

2° - lucasvgm@hotmail.com

3° - 123leandro456@yahoo.com

Mais uma vez, com números não será diferente. Se a gente precisar de um intervalo de "1" até "5", basta especificar pro regex dentro dos colchetes.

5° Passo = [1-5]

1° - brennosullivan22081999@gmail.com

2° - lucasvgm@hotmail.com

3° - 123leandro456@yahoo.com

Podemos misturar strings com números. Se a gente quisesse que o regex retornasse um intervalo de “a” até “c” e de “3” até” 6”, ficaria dessa maneira:

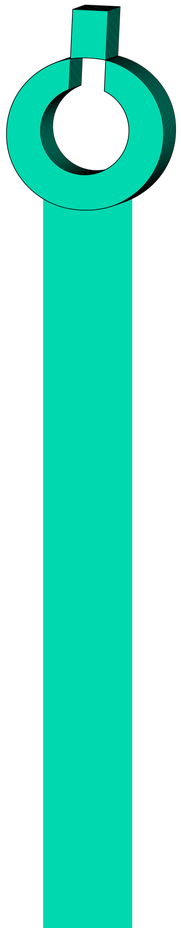
6° Passo = [a-c3-6]

1° - brennosullivan22081999@gmail.com

2° - lucasvgm@hotmail.com

3° - 123leandro456@yahoo.com

Pense que agora, a gente precisa de caracteres especiais. É só seguir a mesma lógica, colocar tudo entre colchetes.



7° Passo = [a-c3-6@+.\]

1° - brennosullivan22081999@gmail.com

2° - lucasvgm@hotmail.com

3° - 123leandro456@yahoo.com

Agora que já entendeu a lógica por trás do regex, vamos direto para o exercício, para entrarmos a fundo nesse assunto tão temido entre os programadores.

1.8.2. Entendendo a estrutura do exercício

O primeiro passo de todos é importar o módulo que utiliza o regex:

```
```

A ideia do código é ser uma validação de dados para emails. Então precisaremos de uma amostra para poder verificar se nosso código está funcionando.

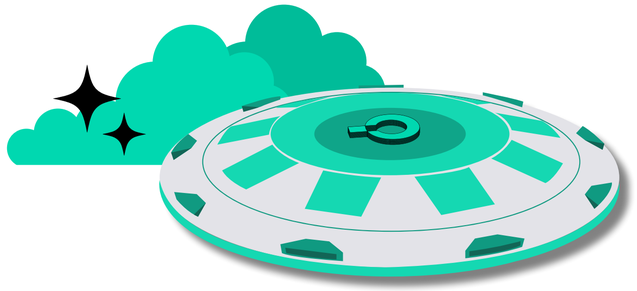
```
bloco_texto = ''' brenno  brennolima@gmail.com  lucas  lucasvgm@hotmail.com
leandro  leandropaulo@yahoo.com.br'''
```

A parte a seguir é a que dá mais insegurança para as pessoas, mas vamos detalhar passo a passo para poder descomplicar esse assunto tão temido. Essa parte a seguir é onde é feita a validação de um email.

```
padrao = r'[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}'
```

A gente sabe que um email é dividido, geralmente, em 4 partes. E que segue basicamente esse padrão:

1° parte - identificação do email, cada pessoa escolhe a sua
2° parte - @
3° parte - gerenciador do seu email (gmail, hotmail, yahoo e etc...)
4° parte - Domínio de topo (.com)
5° parte - Domínio do seu país (.br, .us, .fr e etc...) - Não é obrigatório



Agora traduzindo do regex:

1° parte - `[A-Z0-9._%+-]+` - Essa primeira parte valida todos os números de "0" até "9", valida todas as letras de "a" até "z", além de validar caracteres especiais (._%+-). No final há esse "+" que acompanha a primeira parte, está informando ao regex que a sequência acabou e que não há limitação de caracteres. Essa primeira parte é referente a identificação dos emails.

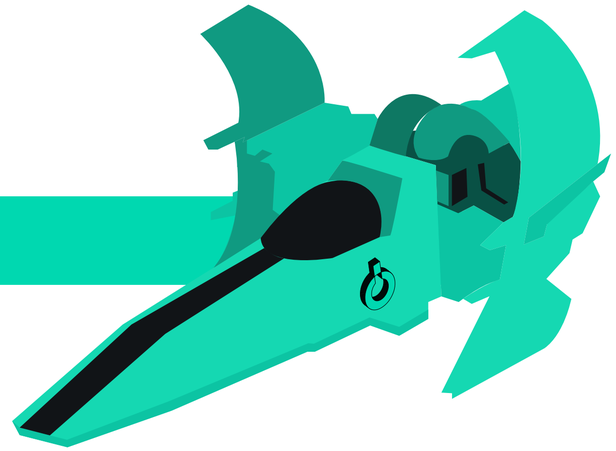
2° parte - `@` - Essa segunda parte nos mostra que, após essa sequência de caracteres da 1° parte., terá um "@" com certeza, se não tiver um "@" não pode ser validado.

3° parte - `[A-Z0-9.-]+` - Essa 3° parte é a do gerenciador do email. Como gerenciador varia muito, temos que validar todos os números de "0" até "9", todas as letras de "a" até "z" e validar caracteres especiais (.-). No final esse "+" que acompanha, está informando ao regex que a sequência acabou e que não há limitação de caracteres.

4° parte - `\.[A-Z]{2,4}` - Essa 4° parte é a do domínio de topo, primeiramente teremos que validar o ponto. A barra " \. " antes do ponto, passa ao regex que esse ponto é um caractere que está incluso, pode ser substituído por [.]. Depois temos que validar o intervalo de [A-Z], como visto anteriormente. Por fim, passaremos ao regex que o intervalo de [A-Z] escolhido, terá um tamanho de 2 até 4 (.com , .gov. , .id).

5° parte - É importante notar que nem todos os emails possuem essa 5° parte, por isso não é obrigatório passar. O regex é apenas um filtro, para validar um email só o regex não será suficiente.

A ideia aqui não é fazer você virar um mestre no regex, mas sim te apresentar a ferramenta, te fazer entender a estrutura e te mostrar o que ela pode fazer.



1.8.3. Finalizando o exercício

Após ter finalizado toda lógica da estrutura dos dados que deseja, deve compilar por meio da fórmula " re.compilar " e passar também o parâmetro " flags=re.IGNORECASE " que informa ao regex que não faz diferença se a letra for maiúscula ou minúscula (pois o email não diferencia). Depois, é só utilizar o " .findall() " para poder encontrar os textos que passou como padrão.

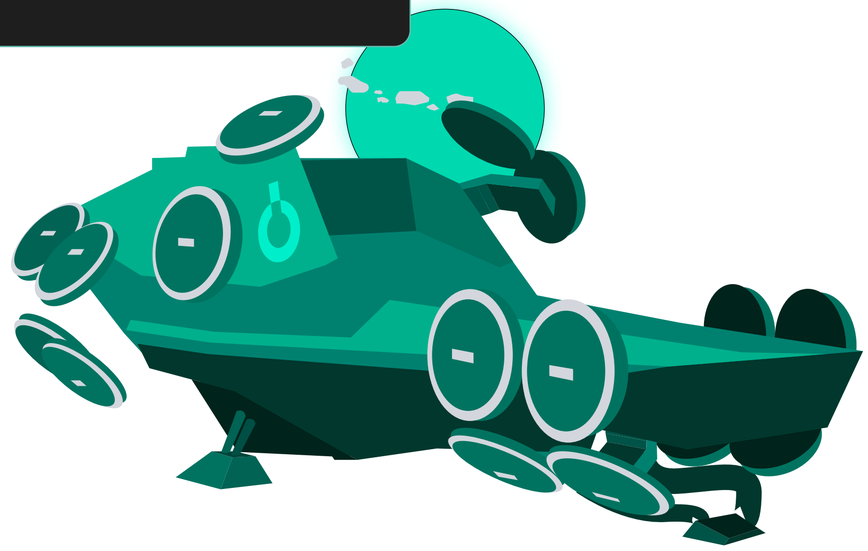
O python possui muitas bibliotecas, e cada biblioteca possui sua linguagem específica. Fique tranquilo que você não gravará todas, até porque a cada dia uma biblioteca surge no python e outra deixa de ser utilizada. Então, é importante que entenda o que cada uma faz e seu potencial, ao invés de tentar decorar cada função.

Exemplo:

```
import re

bloco_texto = ''' brenno  brennolima@gmail.com  lucas  lucasvgm@hotmail.com
leandro  leandropaulo@yahoo.com.br'''

padrao = r'[A-Z0-9._%+-]+@[A-Z0-9.-]+[.][A-Z]{2,4}'
expressao_regular = re.compile(padrao, flags=re.IGNORECASE)

print(expressao_regular.findall(bloco_texto))
```

Resposta:

```
>>  ['brennolima@gmail.com', 'lucasvgm@hotmail.com',
'leandropaulo@yahoo.com.br']
```



Mundo 15

1. Introdução à condicionais

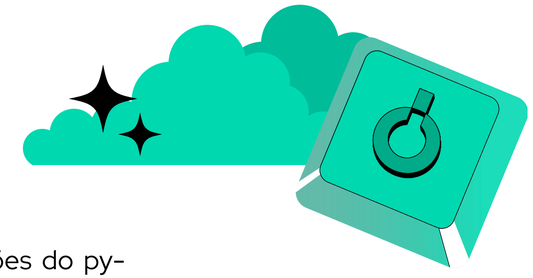
Assim como na programação, diariamente aparecem situações onde você precisa decidir suas ações a partir de condições. Vamos ver um exemplo prático. Vamos supor que você tenha acordado atrasado e precisa decidir se vai para o trabalho sem tomar café da manhã, ou se toma café da manhã e chega atrasado no trabalho. Você mal acordou e já está repleto de decisões que precisa tomar, e não para por aí, porque você precisa escolher qual fruta comer, se vai beber água ou não, qual ônibus pegar, se vai pagar no dinheiro ou no cartão. Então diariamente somos sobrecarregados de situações onde precisamos tomar decisões a partir de certos acontecimentos. Na programação não é diferente, porque um bom programador avalia todas as situações e prepara seu código para quaisquer eventuais erros.

2. Estruturas condicionais

2.1. If

A tradução dessa estrutura é " se " e é assim que podemos ler essa estrutura. Utilizando o exemplo prático, se eu acordar atrasado, eu tomarei café da manhã e chegarei atrasado ao trabalho. Lembrando que essa estrutura é condicional, então ela recebe como parâmetro sempre uma comparação. Vamos relembrar algumas dessas comparações no python agora:

== ➡ Significado de igual
!= ➡ Significado de diferente
y < x ➡ Significado de x maior que y
y > x ➡ Significado de x menor que y



Agora que relembramos algumas das principais comparações do python, vamos formular um pequeno código tentando traduzir para a linguagem python nosso exemplo acima.

Lembra que eu disse que um bom programador deve prever possíveis erros? O python difere maiúsculo e minúsculo, por isso no input é importante colocar a função .upper(). Para transformar a variável "atraso" em caps lock prevendo um possível erro de alguém colocar letra minúscula.

Exemplo:

```
print('\nBom dia, hoje é um belo dia... ')
print('Acabei de acordar, preciso saber se estou atrasado para o trabalho. \n')

atraso = input("Estou atrasado para o trabalho? Digite S/N: ").upper()
if atraso == 'S':
    print("Estou atrasado... Hoje não tem café da manhã")
```

Resposta:

```
>>  Bom dia, hoje é um belo dia...
>>  Acabei de acordar, preciso saber se estou atrasado para o trabalho.

>>  Estou atrasado para o trabalho? Digite S/N: S
```

Resposta:

```
>>  Estou atrasado... Hoje não tem café da manhã
```

É importante mencionar que a partir de agora, as indentações se tornarão mais comuns. Indentação é esse espaço, que parece um parágrafo, que significa que esse texto faz parte da estrutura. No caso acima, o print está fazendo parte da estrutura if. No caso do if, caso a condição esteja correta, irá executar tudo que estiver na indentação correta.

2.2. Elif

A segunda estrutura condicional é utilizada quando a primeira condição não for realizada. Então, utilizando o exemplo acima, caso a resposta não satisfaça a primeira condição, o código verificará se a segunda condição é válida. Podemos repetir a estrutura elif quantas vezes quisermos.

Exemplo:

```
print('\nBom dia, hoje é um belo dia... ')
print('Acabei de acordar, preciso saber se estou atrasado para o trabalho. \n')

atraso = input("Estou atrasado para o trabalho? Digite S/N: ").upper()
if atraso == 'S':
    print("Estou atrasado... Hoje não tem café da manhã")
elif atraso == 'N':
    print("Não estou atrasado... Hoje tem café da manhã")
```

Resposta:

```
>>  Bom dia, hoje é um belo dia...
>>  Acabei de acordar, preciso saber se estou atrasado para o
trabalho.

>>  Estou atrasado para o trabalho? Digite S/N: N
```

Resposta:

```
>>  Não estou atrasado... Hoje tem café da manhã
```

2.3. Else

Por último, é importante mencionar a estrutura “else”. Ela pode ser considerada como “o que sobrou”. Caso nenhuma das condições seja satisfeita, o código percorrerá até chegar no “else”. Depois do else, não há mais nenhuma condição a seguir.

Eu disse anteriormente que é importante tentar prever possíveis erros. Já tratamos um erro, que é o da letra maiúscula. Agora o próximo erro a ser prevenido é a pessoa escrever qualquer outra coisa que não seja “s” e “n”. Já que o programa só reconhece o “s” e o “n” não faz sentido reconhecer nenhuma outra letra, palavra ou número.

Então podemos traduzir para o python a seguinte conclusão: se a resposta não for “s” e não for “n” repita até uma das letras ser escolhida.

Exemplo:

```
print('\nBom dia, hoje é um belo dia... ')
print('Acabei de acordar, preciso saber se estou atrasado para o trabalho. \n')

atraso = input("Estou atrasado para o trabalho? Digite S/N: ").upper()
if atraso == 'S':
    print("Estou atrasado... Hoje não tem café da manhã")
elif atraso == 'N':
    print("Não estou atrasado... Hoje tem café da manhã")
else:
    atraso = input("Resposta inválida, digite S ou N para seguir: ")
```

Resposta:

```
>>  Bom dia, hoje é um belo dia...
>>  Acabei de acordar, preciso saber se estou atrasado para o trabalho.

>>  Estou atrasado para o trabalho? Digite S/N: Abacaxi
```

Resposta:

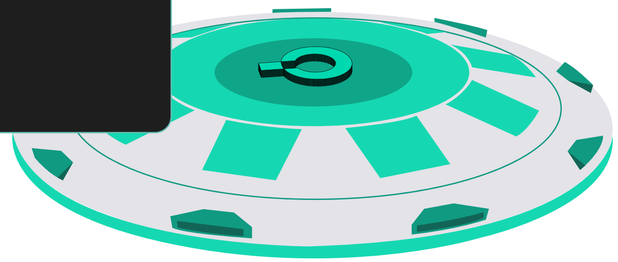
```
>>  Resposta inválida, digite S ou N para seguir:
```

3. Mais de 3 condições

As condições não têm limite de quantidade. Podem ter 3, 4 ou até 90 condições, tudo depende da sua necessidade. O comando elif pode ser utilizado mais de uma vez. Além disso, uma condição pode estar dentro de outra condição.

Exemplo:

```
print('\nBom dia, hoje é um belo dia... ')
print('Acabei de acordar, preciso saber se estou atrasado para o trabalho. \n')

atraso = input("Estou atrasado para o trabalho? Digite S/N: ").upper()
if atraso == 'S':
    cafe = input("Estou atrasado para o trabalho, tomar café mesmo assim? Digite S/N: ").upper()
    if cafe == 'S':
        print('Chegarei muito atrasado, mas alimentado')
    elif cafe == 'N':
        print('Tenho que correr....')
elif atraso == 'N':
    print("Não estou atrasado... Hoje tem café da manhã")
elif atraso == 'Não sei':
    print("Volte a dormir")
else:
    atraso = input("Resposta invalida, digite S ou N para seguir: ")
```



Resposta:

```
>>  Estou atrasado para o trabalho? Digite S/N: S
>>  Estou atrasado para o trabalho, tomar café mesmo assim? Digite S/N: S
```

Resposta:

```
>>  Chegarei muito atrasado, mas alimentado
```

4. Dupla condição

Além de poder colocar infinitas condições dentro de outras condições, no Python conseguimos colocar duas condições ao mesmo tempo.

4.1. And

Traduzido do inglês a palavra “and” tem significado de “e”. Pode ser utilizado quando precisamos satisfazer duas condições ao mesmo tempo.

Exemplo:

```
melhor_professor = "Brenno Sullivan"
melhor_curso = "Codigo.py"

if melhor_professor == "Brenno Sullivan" and melhor_curso == "Codigo.py":
    print("Assistindo as melhores aulas")
```

Resposta:

```
>>  Assistindo as melhores aulas
```

Exemplo2:

```
melhor_professor = "Vinicius Viana"
melhor_curso = "Codigo.py"

if melhor_professor == "Brenno Sullivan" and melhor_curso == "Codigo.py":
    print("Assistindo as melhores aulas")
else:
    print("O professor até é bom, mas não é lendário")
```

Resposta2:

```
>>  O professor até é bom, mas não é lendario
```

4.1. Or

Traduzido do inglês a palavra “or” tem significado de “ou”. Utilizada quando precisamos satisfazer uma condição ou outra.

Exemplo:

```
melhor_professor = "Vinicius Viana"
melhor_curso = "Curso_qualquer.py"

if melhor_professor == "Brenno Sullivan" or melhor_curso == "Codigo.py":
    print("Assistindo as melhores aulas")
else:
    print("Até é bom... mas podia ser melhor...")
```

Resposta:

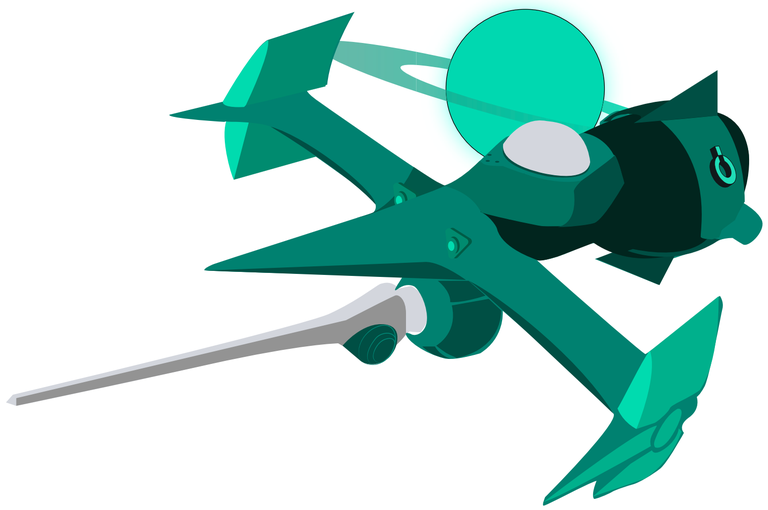
```
>>  Até é bom... mas podia ser melhor…
```

Exemplo2:

```
melhor_professor = "Brenno Sullivan"
melhor_curso = "Curso_qualquer.py"

if melhor_professor == "Brenno Sullivan" or melhor_curso == "Codigo.py":
    print("Assistindo as melhores aulas")
else:
    print("Até é bom... mas podia ser melhor...")
```

Resposta2:

```
>>  Assistindo as melhores aulas
```



Mundo 16

1. Parte prática condicionais

Nesse mundo entraremos a fundo sobre condicional. No exemplo abaixo faremos um programa que armazena a empresa desejada e imprime as informações de cada uma. Utilizaremos o dicionário como um micro banco de dados.

Criaremos um banco de dados com as seguintes informações: ticker, lucro, segmento de listagem e ROE.

```
#banco de dados

weg = {"Ticker": "WEGE3", "Lucro": 1700, 'Segmento de listagem': "Novo mercado", 'ROE': 0.20}
vale = {"Ticker": "VALE3", "Lucro": 21700, 'Segmento de listagem': "Novo mercado", 'ROE': 0.19}
petrobras = {"Ticker": ["PETR3", "PETR4"], "Lucro": 6700, 'Segmento de listagem': "Nível 2", 'ROE': 0.15}
tranqueira_fc = {"Ticker": "TRNQ3", "Lucro": -20000, 'Segmento de listagem': "Tradicional", 'ROE': 0.02}
```

Vamos selecionar uma empresa dentre as que estão no banco de dados. É importante mencionar que utilizamos a função .lower() como tratamento de dados. Sabemos que existe diferenciação de maiúsculas e minúsculas no python, e cada pessoa que digitar as informações pode colocar de um jeito diferente.

```
empresa_escolhida = str(input("Escolha uma empresa entre Weg, Petrobras, Vale ou Tranqueira: ")).lower()
escolha = True
```

No começo do código já definimos escolha = True. Ao escolher "tranqueira" como a empresa, o programa pergunta se temos certeza da escolha e, caso a gente desista da escolha, ele mudará para escolha = False. Com isso, seremos direcionados para "Melhor não saber mesmo".

```
if empresa_escolhida == "tranqueira":
    usuario_escolha = str(input('''Tem certeza que deseja saber sobre a Tranqueira?
    Digite 'S' para sim e 'N' para não ''')).lower()
    if usuario_escolha == "n":
        escolha = False
    else:
        pass
elif empresa_escolhida == "tranqueira" and escolha == False:

    print("Melhor não saber mesmo!")
```

Caso a gente digite que quer prosseguir, ele irá redirecionar para a condicional que está pegando informações do dicionário.

Nessa condicional, ele formata o ROE em porcentagem com nenhuma casa decimal.

```
elif empresa_escolhida == "tranqueira" and escolha != False:

    porcentagem_roe = "{:.0%}".format(tranqueira_fc['ROE'])

    print(f'''O lucro da Tranqueira é R${tranqueira_fc['Lucro']}, seu código de negociação é
    {tranqueira_fc['Ticker']}, e o ROE da empresa é de {porcentagem_roe}''')
```

Exemplo completo:

```
weg = {"Ticker": "WEGE3", "Lucro": 1700, 'Segmento de listagem': "Novo mercado", 'ROE': 0.20}
vale = {"Ticker": "VALE3", "Lucro": 21700, 'Segmento de listagem': "Novo mercado", 'ROE': 0.19}
petrobras = {"Ticker": ["PETR3", "PETR4"], "Lucro": 6700, 'Segmento de listagem': "Nível 2", 'ROE': 0.15}
tranqueira_fc = {"Ticker": "TRNQ3", "Lucro": -20000, 'Segmento de listagem': "Tradicional", 'ROE': 0.02}

empresa_escolhida = str(input("Escolha uma empresa entre Weg, Petrobras, Vale ou Tranqueira: ")).lower()
escolha = True

if empresa_escolhida == "tranqueira":
    usuario_escolha = str(input("Tem certeza que deseja saber sobre a Tranqueira? Digite 'S' para sim e 'N' para não ")).lower()
    if usuario_escolha == "n":
        escolha = False
    else:
        pass

if empresa_escolhida == "weg":
    porcentagem_roe = "{:.0%}".format(weg['ROE'])
    print(f"O lucro da Weg é R${weg['Lucro']}, seu código de negociação é {weg['Ticker']}, e o ROE da empresa é de {porcentagem_roe}")
elif empresa_escolhida == "vale":
    porcentagem_roe = "{:.0%}".format(vale['ROE'])
    print(f"O lucro da Vale é R${vale['Lucro']}, seu código de negociação é {vale['Ticker']}, e o ROE da empresa é de {porcentagem_roe}")
elif empresa_escolhida == "petrobras":
    porcentagem_roe = "{:.0%}".format(petrobras['ROE'])
    print(f"O lucro da Petrobras é R${petrobras['Lucro']}, seu código de negociação é {petrobras['Ticker']}, e o ROE da empresa é de {porcentagem_roe}")
elif empresa_escolhida == "tranqueira" and escolha != False:
    porcentagem_roe = "{:.0%}".format(tranqueira_fc['ROE'])
    print(f"O lucro da Tranqueira é R${tranqueira_fc['Lucro']}, seu código de negociação é {tranqueira_fc['Ticker']}, e o ROE da empresa é de {porcentagem_roe}")
elif empresa_escolhida == "tranqueira" and escolha == False:
    print("Melhor não saber mesmo!")
else:
    print('Por favor, digite uma empresa válida!')
```

Mundo 17

1. Introdução a estruturas de repetição

Estrutura de repetição são recursos do python que executam um código repetidamente até que uma certa condição seja satisfeita. Essa condição pode ser tamanho de uma lista a ser percorrida ou um valor a ser atingido. Seja como for, é uma estrutura muito utilizada que possibilita a compactação do código.

2. Estruturas de repetição

2.1. Estrutura for

A estrutura for é utilizada com uma variável é um item a ser percorrido. Sua estrutura obedece a seguinte ordem: for variável in item, onde a variável pode assumir qualquer nome que você quiser colocar. Isso ocorre pois ela apenas representa o item que está sendo iterado naquele momento, logo, você pode nomear essa variável de abacaxi, péricles ou bolsonaro, literalmente tanto faz.

Mas pela boa prática de programação é importante que o nome seja compatível com seu programa. O item é o que vai ser percorrido, pode ser uma lista, um dicionário ou poderá ser definido antes.

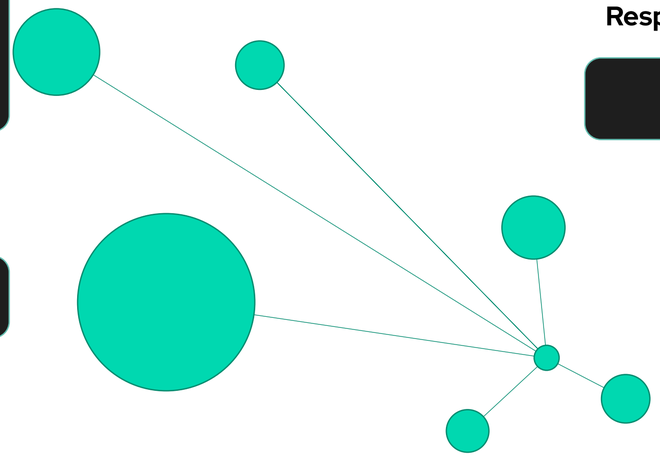
No exemplo a seguir, estamos percorrendo uma lista com 10 itens e dando print nela. O objetivo é para que você possa entender o funcionamento da estrutura. Note também, que a variável escolhida não interferiu em nada, ela é apenas para ser referenciada depois.

Exemplo:

```
lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for numero in lista:
    print(numero, end=" ")
print()
for bolsonaro in lista:
    print(bolsonaro, end=" ")
```

Resposta:

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



Agora para dificultar um pouquinho vamos misturar estruturas de repetição com estruturas condicionais. Vamos colocar um if dentro de um for. Ele fará com que o código só retorne números maiores que 5. É muito comum dentro do python que tenha estruturas condicionais dentro de loop's, por isso é importante que você entenda bem.

Exemplo:

```
lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for numero in lista:
    if numero > 5:
        print(numero, end=" ")
```

Resposta:

```
>>  6 7 8 9 10
```

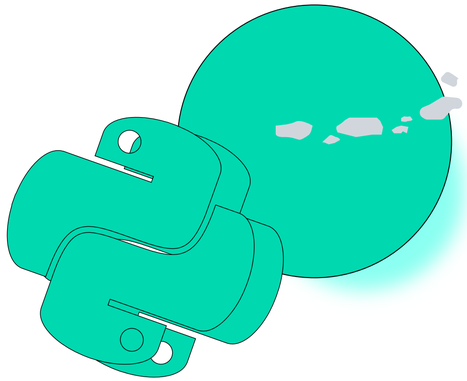
2.2. Estrutura while

Traduzindo para o português, o while significa enquanto e essa estrutura recebe como parâmetro a condição a ser satisfeita. Ou seja, ele fará o loop enquanto a condição não for satisfeita. Nesse tipo de estrutura você deve tomar bastante cuidado, pois caso esqueça de referenciar a condição da forma certa, ele fará o loop infinitamente.

No caso abaixo, a condição é ser menor que 5 e, toda vez que o programa faz um looping, ele soma à variável cont e printa o cont daquele looping. Ele faz isso até que a condição seja satisfeita.

Exemplo:

```
cont = 0
while cont < 5:
    print(cont)
    cont = cont + 1
```



Resposta:

```
>>  0
    1
    2
    3
    4
```

3. Quebrando o looping

3.1. Break

Essa ferramenta é responsável por encerrar um loop.

Exemplo:

```
objeto = True
cont = 0
while objeto == True:
    cont = cont + 1
    print(cont)
    if cont == 3:
        break
```

Resposta:

```
>> 1
   2
   3
```

3.2. Continue

Essa ferramenta é responsável por interromper um loop e dar continuidade para próxima iteração do loop.

Exemplo:

```
lista = [1, 2, 3, 4, 5, 6, 7, 8]
for numero in lista:
    if 3 < numero < 8:
        continue
    print(numero)
```

Resposta:

```
>> 1
   2
   3
```

Mundo 18

1. Estrutura de repetição - prática

1.1. Loop simples em uma lista

Nesse exemplo utilizaremos o módulo time. Neste módulo, a função sleep é utilizada para dar pausa de alguns segundos no programa. Ela recebe como parâmetro o número de segundos que você quer que o pause aconteça.

Exemplo:

```
import time

lista_empresas = ['Weg', 'Vale', 'Petrobras']
for empresa in lista_empresas:
    print(empresa)
    time.sleep(1)
```

Resposta:

```
>> Weg
   Vale
   Petrobras
```

1.2. Loop em um range

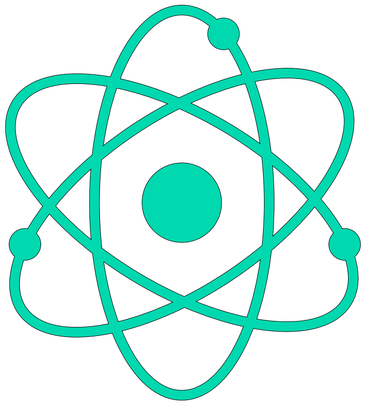
A estrutura de range() recebe como parâmetros o início, o final e a frequência do intervalo (começo, final, frequência). A frequência padrão por definição é 1 e no caso abaixo será percorrido um loop no intervalo 0 - 1. O motivo disso é que o último número do intervalo não está incluso em uma seleção no Python.

Exemplo:

```
lista_empresas = ['Weg', 'Vale', 'Petrobras']
for i in range(0, 2):
    print(lista_empresas[i])
```

Resposta:

```
>>  Weg
    Vale
```



1.2.1. Por que utilizar um range?

É importante saber que, ao utilizar a estrutura “for”, muitas das vezes será necessário utilizar o número da repetição para associação de objetos pelo index. Faremos isso logo abaixo e daremos um ponta pé inicial no módulo pandas.

1.2.1.1. Função append()

Como já citado neste pdf, essa função recebe como parâmetro o elemento escolhido e o adiciona ao final da lista.

1.2.1.2. Função concat()

A função concat() é utilizada para concatenar (juntar) séries ou dataframes.

1.2.1.3. Função pd.DataFrame()

A função DataFrame() é a mais utilizada de todo módulo pandas, pois o objetivo dela é criar dataframes. Ela pode receber alguns parâmetros, mas nesse exercício a função irá receber os dados que serão inseridos dentro do dataframe e o index desses dados. Isso irá retornar uma tabela com os dados apresentados e relacionados com o index.

1.2.2. Exemplo utilização de um range()

No exemplo abaixo é disponibilizado uma lista de empresas e suas respectivas cotações. O objetivo desse exercício é utilizar essas listas para criar um dataframe (uma tabela).

1° Passo - O primeiro passo é importar os módulos que iremos utilizar que, neste caso, será apenas o módulo do pandas. Podemos também definir por qual nome chamaremos os pacotes importados, nesse caso decidimos importar esse módulo como "pd" ao invés de pandas, que seria o tradicional.

```
import pandas as pd
```

2° Passo - Criaremos 3 listas, uma com os nomes das empresas, outra com as cotações e a última vazia. A lista vazia irá armazenar os dataframes que criaremos.

```
lista_empresas = ['Weg', 'Vale', 'Petrobras']
lista_cotacoes = [20, 30, 45]
lista_dicionarios = []
```

3° Passo - Utilizando a estrutura de repetição "for", criaremos um dicionário para cada uma das três posições da lista, contendo o nome e a cotação delas. No exercício abaixo, a posição delas está representada pelo "i" que é cada intervalo do range que será percorrido.

Logo após, utilizaremos a função pd.DataFrame para criar um dataframe para cada empresa, contendo o nome e a cotação dela. O parâmetro index , que pode ser interpretado neste caso como o número da linha, será passado associando com a posição percorrida no intervalo (range(0,3)).

Por fim, adicionamos na lista vazia cada dataframe criado. Ao fim do loop, teremos uma lista com 3 dataframes. (cada dataframe representará uma linha).

```
for i in range(0, 3):
    dicionario = {'Nome empresa': lista_empresas[i], 'Cotação': lista_cotacoes[i]}
    dataframe = pd.DataFrame(dicionario, index=[i])
    lista_dicionarios.append(dataframe)
```

4° Passo - Depois de armazenado os dataframes em cada uma das posições da lista, utiliza-se a função pd.concat() para juntar os dataframes e passar como resposta um único dataframe completo.

Exemplo:

```
# 1° Passo
import pandas as pd

# 2° Passo
lista_empresas = ['Weg', 'Vale', 'Petrobras']
lista_cotacoes = [20, 30, 45]
lista_dicionarios = []

# 3° Passo
for i in range(0, 3):
    dicionario = {'Nome empresa': lista_empresas[i], 'Cotação': lista_cotacoes[i]}
    dataframe = pd.DataFrame(dicionario, index=[i])
    lista_dicionarios.append(dataframe)

# 4° Passo
tabela = pd.concat(lista_dicionarios)
print(tabela)
```

Resposta:

```
>>      Nome empresa  Cotação
      0          Weg       20
      1         Vale       30
      2    Petrobras       45
```

1.3. Função enumerate()

Essa função é utilizada com tudo que é iterável dentro do python, como listas, tuplas, etc. Quando essa função é utilizada com o for, ela retornará o objeto que deseja e a posição do item dentro do objeto percorrido.

Repetindo o exercício acima, ao invés de pré-definir um intervalo igual no exercício acima (range(0,3)), desta vez utilizaremos a função enumerate() pois ela fará com que a estrutura de repetição reconheça a posição do item que está percorrendo e essa posição poderá ser utilizada para referenciar uma outra lista.

Exemplo:

```python
# 1º Passo
import pandas as pd

# 2º Passo
lista_empresas = ['Weg', 'Vale', 'Petrobras']
lista_cotacoes = [20, 30, 45]
lista_dicionarios = []

# 3º Passo
for i,empresas in enumerate(lista_empresas):
    dicionario = {'Nome empresa': empresas, 'Cotação': lista_cotacoes[i]}
    dataframe = pd.DataFrame(dicionario, index=[i])
    print(type(dataframe))
    lista_dicionarios.append(dataframe)

# 4º Passo
tabela = pd.concat(lista_dicionarios)
print(tabela)
```

Resposta:

```
>>      Nome empresa  Cotação
        0          Weg       20
        1         Vale       30
        2    Petrobras       45
```

1.4. While

O significado de “while” no portugues é “enquanto”. Então, sempre que tentar entender o que aquela condição está significando, substitua “while” por “enquanto”.

1.4.1. Exercício resolvido 1

Nesse exercício, o objetivo é fazer um loop infinito que retorna quanto tempo o programa está rodando. Esse programa rodará enquanto while == True e nele não há nada que faça mudar essa condição, então dizemos que isso é um loop infinito.

Exemplo:

```python
import time
import datetime
import pytz

tz = pytz.timezone('America/Sao_Paulo')
tempo = datetime.datetime.now(tz)
while True:
    tempo_rodando = datetime.datetime.now(tz) - tempo
    print(f'O programa está rodando há {tempo_rodando} segundos')
    time.sleep(5)
```

Resposta:

```
>>   O programa está rodando há 0:00:00 segundos
     O programa está rodando há 0:00:05.004215 segundos
     O programa está rodando há 0:00:10.016573 segundos
     O programa está rodando há 0:00:15.031725 segundos
     O programa está rodando há 0:00:20.043052 segundos
     O programa está rodando há 0:00:25.045919 segundos
     O programa está rodando há 0:00:30.059093 segundos
     O programa está rodando há 0:00:35.063057 segundos
```

1.4.2. Exercício resolvido 2

Ao menos que queira fazer um loop infinito, sempre que for trabalhar com “while” deve colocar uma condição que, depois de atendida, interrompa o loop.

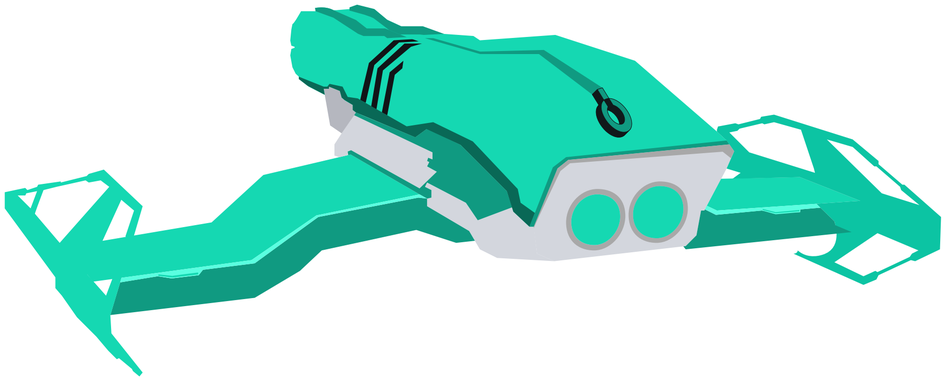
No caso abaixo, o while está percorrendo a lista_cotacoes e, se o valor da lista for maior do que 35, o loop será interrompido.

Exemplo:

```
contador = 0
lista_cotacoes = [20, 30, 45]

while lista_cotacoes[contador] < 35:
    print(lista_cotacoes[contador])
    contador = contador + 1
```

Resposta:

```
>>  20
    30
```



Mundo 19

1. Função

Apesar de já termos visto algumas funções ao longo do curso (print, len, max e etc...), essas são funções nativas do Python. Veremos agora como fazer a criação das nossas próprias funções. No módulo de orientação a objeto, iremos criar muitas funções e utilizá-las para mudar o estado dos objetos dentro do código, mas falaremos à frente sobre isso. No momento você só precisa entender que as funções são uma forma de você evitar repetições e otimizar seu código.

1.1. Estrutura das funções

Como as estruturas de repetição, as funções também utilizam a indentação de forma obrigatória. Caso não lembre, a indentação é aquele espaço à esquerda do código que parece com um parágrafo. Para criar uma função você deve respeitar a seguinte estrutura:

def nome_da_função()

código

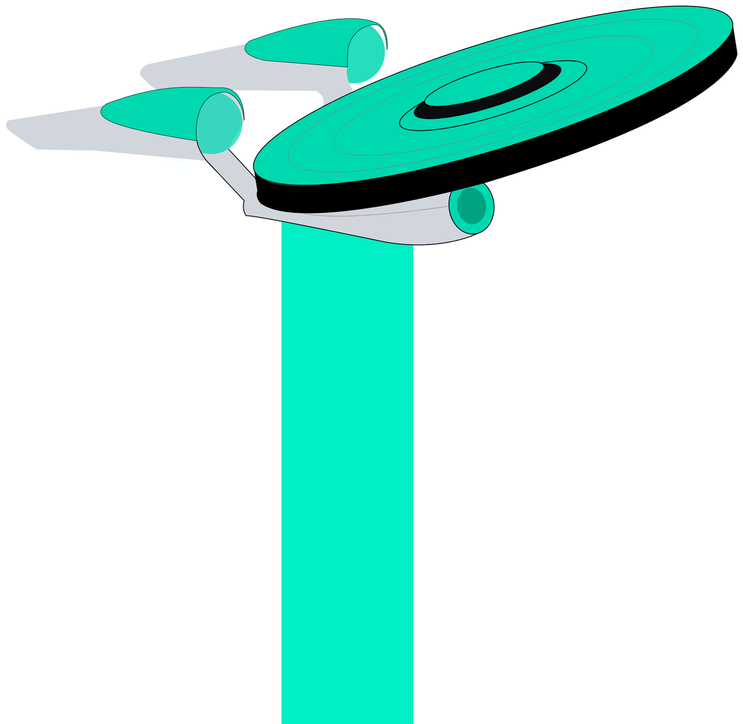
Não é obrigatório preencher o parênteses com argumentos, porém você criá-los como parâmetros na função. Caso queira retornar um valor quando executar uma função, basta colocar o comando return e a variável de retorno desejada. Para executar a função, basta escrever o nome da função com parênteses no final e passar os argumentos, caso seja necessário.

Exemplo:

```
soma_teste = 0
print("A variável soma_teste é igual =>", soma_teste)

def soma_dois_numeros(a, b):
    soma = a + b
    return soma

soma_teste = soma_dois_numeros(2, 3)
print("A variável soma_teste é igual =>", soma_teste)
```

Resposta:

```
>>  A variável soma_teste é igual => 0
    A variável soma_teste é igual => 5
```



Mundo 20

1. Função - prática

1.1. Função que soma o número 2 ao número escolhido

O objetivo deste exercício é fazer uma função que sempre some o número 2 ao número adicionado pelo usuário. Dentro dos parentes nós definimos 2 argumentos: o primeiro é o número a ser somado e o segundo que é a soma executada. Por padrão, definimos que a soma é igual a 2. Caso o usuário queira somar outro número, basta passar como argumento.

Quando formos chamar a função, precisamos colocar como parâmetro o argumento que ainda não foi definido. Se não informarmos ao python o valor do argumento que ficou pendente, o Python retornará um erro de argumentos.

Exemplo errado:

```
numero_digitado = int(input('Digite um número inteiro: '))

def soma_numeros(numero, soma = 2):
    numero_somado = numero + soma
    return numero_somado

novo_numero = soma_numeros()
print(novo_numero)
```

Resposta:

```
>>  soma_numeros() missing 1 required positional argument: 'numero'
```

Exemplo correto:

```
numero_digitado = int(input('Digite um número inteiro: '))

def soma_numeros(numero, soma = 2):
    numero_somado = numero + soma
    return numero_somado

novo_numero = soma_numeros(numero = numero_digitado)
print(novo_numero)
```

Resposta:

```
>>  Digite um número inteiro: 5
>>  A soma é => 7
```

1.2. Função que calcula a rentabilidade

O objetivo deste exercício é fazer uma função que calcule a rentabilidade de até 2 casas decimais, de uma ação. A função recebe como parâmetro duas variáveis que serão escolhidas pelo usuário.

Exemplo correto:

```
def calcula_rentabilidade(valor_inicial, valor_final):
    rentabilidade = valor_final/valor_inicial - 1
    rentabilidade_em_porcentagem = "{:.2%}".format(rentabilidade)
    return rentabilidade_em_porcentagem

valor_inicial = float(input('Digite a cotação inicial: '))
valor_final = float(input('Digite a cotação final: '))
rentabilidade = calcula_rentabilidade(valor_inicial=valor_inicial, valor_final=valor_final)

print("A rentabilidade foi de =>",rentabilidade)
```

Resposta:

```
>>  Digite a cotação inicial: 5
>>  Digite a cotação final: 5.9
>>  A rentabilidade foi de => 18.00%
```

2. Função lambda

2.1. Função lambda - Introdução

A função lambda é uma função que não é definida, popularmente chamada de função anônima. Resumindo, é uma função simplificada. Vamos supor que você precise de uma função que retorne quanto de imposto você precisa pagar. Do modo tradicional, você faria desse jeito:

Exemplo:

```
def imposto(salario):
    salario = salario * 0.3
    return print(salario)
salario = int(input("Digite o valor do imposto? "))
imposto(salario)
```

Resposta:

```
>>  Digite o valor do salário: 5000
>>  O valor a ser pago é => 1500
```

Com a função lambda, 4 linhas de códigos podem virar algumas palavras. Isso é um exemplo simples, o poder da função lambda pode ir muito além. A principal aplicação das lambdas é utilizar essa função como parâmetro de outra função. Veremos nos exemplos a seguir.

Exemplo:

```
salario = int(input("Digite o valor do salário: "))
def imposto(x): return salario * 0.3

print("O valor a ser pago é =>", imposto(salario))
```

Resposta:

```
>>  Digite o valor do salário: 5000
>>  O valor a ser pago é => 1500
```

2.2 Função lambda - Prática

2.2.1. Função map()

A função map() aplica uma função a todos os elementos de uma coleção (lista,tupla, etc). Podemos utilizar uma função lambda para otimizar a forma que escrevemos nosso código e passar a função de imposto para ser aplicada pelo map através de uma lambda. Dessa forma, nós não precisamos definir nossa função previamente e economizamos muitas linhas de código para escrever uma função simples.

2.2.2. Exercício

Exemplo sem lambda:

```
def imposto(salario):
    salario = salario * 0.3
    return salario

precos = [10, 20, 30, 40]
descontos1 = list(map(imposto, precos))
print(f"Pela função normal: {descontos1}")
```

Resposta:

```
>>  Pela função normal:  [3.0, 6.0, 9.0, 12.0]
```

Exemplo:

```
precos = [10, 20, 30, 40]
descontos2 = list(map(lambda x: x * 0.3, precos))
print(f"Pela função lambda: {descontos2}")
```

Resposta:

```
>>  Pela função lambda:  [3.0, 6.0, 9.0, 12.0]
```

Mundo 21

1. Tratamento de erros e exceções

Uma das formas de prever bugs é tentar prever todas as possibilidades de erros que podem ocorrer. Por isso, falaremos de tratamento de erros. É importante entender o porquê de o programa não ter rodado corretamente e tentar contornar da melhor forma possível. O Python possui algumas ferramentas que ajudam nisso, como os blocos de tratamento de erros Try, Except, Else e Finally.

Essas ferramentas são parecidas com as estruturas condicionais if, elif e else.

1.1. Try

O seu uso na estrutura é obrigatório e deve ser utilizado junto com o Except. Na tradução literal para o português, o significado seria “tente”. A grosso modo, estamos mandando nosso código tentar fazer tal coisa.

1.2. Except

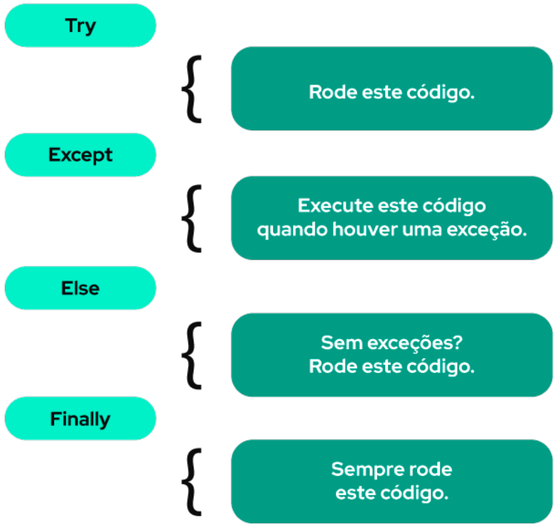
Também é obrigatório e deve ser utilizado junto com o try. Na tradução literal para o portugues, o significado seria “exceto”. O except identifica o erro, possibilita imprimir o erro na tela e realizar alguma ação caso o erro aconteça. Isso é muito útil, pois caso o programa entre em alguma exceção, ou erro, não é necessário fazer o programa parar. Uma das possibilidades é simplesmente ignorar o erro (péssima prática de programção 😂). Podemos usar o except genérico (para qualquer erro) ou para erros específicos. Alguns desses erros específicos são o `ZeroDivisionError` e o `ValueError`, que são, respectivamente, erro de divisão por zero e erro de tipo.

1.3. Else

Não é obrigatório. Podemos utilizar o else para executar um código quando o programa não apresenta nenhum erro.

1.4. Finally

Não é obrigatório. Ele é utilizado para executar um código que roda independente se o programa deu errado ou não. Uma mensagem de "volte sempre" em um sistema que interage com o usuário, por exemplo.



2. Tratamento de erros e exceções - Prática

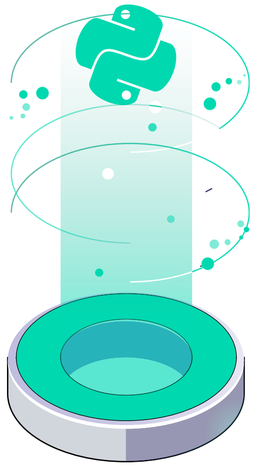
No caso abaixo foi feita uma calculadora que efetua a operação divisão e uma boa prática de programação é tratar todos os possíveis erros. O primeiro erro que tratamos no código é o de divisão por zero, pois sabemos que na matemática não existe divisão zero. Dessa forma, caso isso aconteça ele retornará a mensagem *"Não é possível dividir um número por 0!"*. No segundo caso, trataremos de um erro de valor, quando o usuário coloca um valor que não é um número, como uma string por exemplo. Caso isso aconteça, o programa retornará *"Ocorreu um erro com o tipo de dados digitado."*. Caso o código não apresente nenhum desses erros, o programa apresentará o resultado e a seguinte mensagem: *'A divisão entre o 1° e o 2° número é {divisao}'*. Independente do resultado, o código sempre retornará a mensagem "*Obrigado por participar!*".

Exemplo:

```
try:
    numero1 = float(input("Digite um número: "))
    numero2 = float(input("Digite outro número: "))
    divisao = numero1/numero2

except ZeroDivisionError:
    print("Não é possível dividir um número por 0!")
except ValueError:
    print("Ocorreu um erro com o tipo de dados digitado.")
else:
    print(f'A divisão entre o 1º e o 2º número é {divisao}')
finally:
    print("Obrigado por participar!")
```

Resposta:

```
>>  Digite um número: 3
>>  Digite outro número: 7
>>  A divisão entre o 1° e o 2° número é 0.42857142857142855
>>  Obrigado por participar!
```



Mundo 22

1. Ferramentas avançadas: List Comprehension, Map e Filter - Como melhorar a utilização de listas?

O objetivo desse mundo é a otimização do código e o aprendizado de ferramentas avançadas. Quanto mais efetivo for o código, melhor. Dizemos que um código é efetivo quando ele tem poucas linhas e muitas funcionalidades. A seguir verá um exemplo de otimização utilizando uma List Comprehension.

1.1. Exemplo 1 - List Comprehension

O objetivo do código abaixo é criar uma nova lista apenas com as ações ordinárias. Ações ordinárias são aquelas que possuem o 3 no ticker.

Exemplo não otimizado:

```
lista_codigos_negociacao = ['WEGE3', 'PETR4', 'PETR3', 'PCAR3', 'ALPA4']

nova_lista_codigos = []
for ticker in lista_codigos_negociacao:
    if "3" in ticker:
        nova_lista_codigos.append(ticker.lower())

print(" A lista apenas com as ordinárias é: ",nova_lista_codigos)
```

Resposta:

```
>>  A lista apenas com as ordinárias é:  ['wege3', 'petr3', 'pcar3']
```

Utilizando o List Comprehension poderemos fazer essa mesma coisa, mas agora com apenas duas linhas de código. List Comprehension é uma forma eficiente de criar uma nova lista a partir de uma antiga. Dizemos que essa estrutura é uma fábrica de lista. Ela possui uma estrutura que acompanha a seguinte lógica **ação -> for loop -> condição (caso necessário)**.

Repare que os códigos fazem a mesma função, só que o de baixo está apenas mais "enxugado". Dentro da lista, ele está pegando todos os códigos da *lista_codigos_negociacao* e conferindo se ele possui 3 no código. Caso a resposta seja positiva, ele irá adicionar à nova lista transformando todos em letra minúscula.

Exemplo otimizado:

```
lista_codigos_negociacao = ['WEGE3', 'PETR4', 'PETR3', 'PCAR3', 'ALPA4']
nova_lista_codigos = [ticker for ticker in lista_codigos_negociacao if "3" in ticker]
print(" A lista apenas com as ordinárias é: ",nova_lista_codigos)
```

Resposta:

```
>>  A lista apenas com as ordinárias é:  ['WEGE3', 'PETR3', 'PCAR3']
```

1.2. Exemplo 2 - Filter + funções lambda

O caso abaixo filtra da lista *lista_codigos_negociacao* os valores que estão na lista_restrita, por meio da função filter. Isso irá retornar uma nova lista sem os valores filtrados. Juntando essa ferramenta com a função lambda, é possível.

1.2.1. Função filter()

A função filter recebe como parâmetro uma função pré-definida pelo usuário e um iterável (lista, set, tupla etc). No caso, ela filtrará no iterável os parâmetros que você colocar na função. No caso abaixo, a função está recebendo os valores que não estão na *lista_restrita*, que são "WEG3" e "PCAR3". Então a função filter retornará qualquer outro valor da lista *lista_codigos_negociacao*, a função filter retorna um objeto, por isso é importante transformá-la em uma lista depois.

A função lambda percorre todos os elementos de uma lista executando a função pré-definida, no exemplo abaixo ela pode ser traduzida como: se o ticker da *lista_codigos_negociacao* não estiver na *lista_restrita*. Ele será retornado.

Exemplo de utilização filter + lambda:

```
lista_codigos_negociacao = ['WEGE3', 'PETR4', 'PETR3', 'PCAR3', 'ALPA4']
lista_restrita = ['WEGE3', 'PCAR3']
nova_lista_codigos = list(
    filter(lambda ticker: ticker not in lista_restrita, lista_codigos_negociacao))
print(" A nova lista é: ",nova_lista_codigos)
```

Resposta:

```
>>  A nova lista é:  ['PETR4', 'PETR3', 'ALPA4']
```

1.3. Função lambda + map()

O objetivo deste exercício é transformar todos os itens da lista em strings.

1.3.1. Função map()

A função map() aplica uma função a todos os elementos de uma coleção (lista,tupla, etc).

Podemos utilizar uma função lambda em conjunto com uma função map(). No caso abaixo ele está percorrendo a lista, e a cada item da lista ele transforma o tipo desse item em string pela função str().

Exemplo otimizado:

```
lista_numeros = [9, 8, 3, 4]
lista_string_numeros = list(map(lambda x: str(x), lista_numeros))

print(lista_string_numeros)
```

Resposta:

```
>>  ['9', '8', '3', '4']
```

Mundo 23

1. Módulo datetime

1.1. Importando módulos

Além de importar os módulos completos, você também pode importar os submódulos deste módulo. Desta forma, você não é obrigado a importar o módulo completo. Lembre-se: menos é mais. Observe o exemplo abaixo, neste mundo importaremos algumas funções do módulo datetime e aprofundaremos nas principais.

Exemplo:

```
from datetime import timedelta
from datetime import date
from datetime import datetime
from dateutil.relativedelta import relativedelta
import pytz
```

1.2. Função date()

Essa função recebe como parâmetro o ano, o mês e o dia, respectivamente, nesta ordem. Retorna uma data no ano como um tipo datetime.date.

Exemplo:

```
from datetime import date

data_sem_horas = date(2022, 8, 31) #ano, mes, dia
print("A data é => ",data_sem_horas)
print("E seu tipo é => ",type(data_sem_horas))
```

Resposta:

```
A data é =>  2022-08-31
E seu tipo é =>  <class 'datetime.date'>
```

1.3. Função datetime()

Essa função recebe como parâmetro o ano, o mês, o dia, as horas, os minutos e os segundos, respectivamente, nesta ordem. Retorna a data e a hora do ano como um tipo datetime.datetime.

Exemplo:

```
from datetime import datetime

data_completa = datetime(2022, 8, 31, 20, 30, 56) #ano, mes, dia, horas, minuto, segundo
print("A data é => ",data_completa)
print("E seu tipo é => ",type(data_completa))
```

Resposta:

```
A data é =>  2022-08-31 20:30:56
E seu tipo é =>  <class 'datetime.datetime'>
```

1.4. Função strptime()

Essa função recebe como parâmetro uma data no formato de string e a formatação da data. Retorna essa data no formato datetime.datetime e na formatação pré definida, a formatação é apenas para o Python entender como está escrita a String e a validar como data.

Exemplo:

```
from datetime import datetime

data_texto1 = "18/04/1987"
data_texto2 = "2022-08-22"

data_python1 = datetime.strptime(data_texto1, '%d/%m/%Y')
data_python2 = datetime.strptime(data_texto2, '%Y-%m-%d')

print("Essa aqui é a data 2 => ",data_python2)
print("Essa aqui é a data 1 => ",data_python1)
print("Essa aqui é o tipo 2 => ",type(data_python2))
print("Essa aqui é o tipo 1 => ",type(data_python1))
```

Resposta:

```
>>  Essa aqui é a data 2 =>  2022-08-22 00:00:00
>>  Essa aqui é a data 1 =>  1987-04-18 00:00:00
>>  Essa aqui é o tipo 2 =>  <class 'datetime.datetime'>
>>  Essa aqui é o tipo 1 =>  <class 'datetime.datetime'>
```

1.5. Função date()

Essa função recebe como parâmetro uma data no formato datetime.datetime. Retorna essa data como tipo datetime.date.

Exemplo:

```
from datetime import datetime
from datetime import date

data_texto1 = "18/04/1987"
data_python1 = datetime.strptime(data_texto1, '%d/%m/%Y')
print("Data antes da função date() => ",data_python1)
print("Tipo da data antes da função date() =>", type(data_python1))

data_python = data_python1.date()
print("Data depois da função date() => ",data_python)
print("Tipo da data antes da função date() =>", type(data_python))
```

Resposta:

```
>>  Data antes da função date() =>  1987-04-18 00:00:00
>>  Tipo da data antes da função date() => <class 'datetime.datetime'>
>>  Data depois da função date() =>  1987-04-18
>>  Tipo da data antes da função date() => <class 'datetime.date'>
```

1.6. Extraindo informações das datas.

1.6.1. Atributo year

Esse atributo é um parâmetro da data no formato datetime.date ou datetime.datetime. Retorna o ano desta data como um inteiro.

1.6.2. Atributo month

Esse atributo é um parâmetro da data no formato datetime.date ou datetime.datetime. Retorna o mês desta data como um inteiro.

1.6.3. Atributo day

Esse atributo é um parâmetro da data no formato datetime.date ou datetime.datetime. Retorna o dia desta data como um inteiro.

1.6.4. Atributo hour

Esse atributo é um parâmetro da data no formato datetime.date ou datetime.datetime. Retorna a hora desta data como um inteiro.

1.6.5. Atributo minute

Esse atributo é um parâmetro da data no formato datetime.date ou datetime.datetime. Retorna o minuto desta data como um inteiro.

Exemplo:

```python
from datetime import date

data = date(2021, 3, 28)
print("A data é => ", data)
print(f"O ano da data é {data.year} e seu tipo é {type(data.year)}")
print(f"O ano da data é {data.month} e seu tipo é {type(data.month)}")
print(f"O ano da data é {data.day} e seu tipo é {type(data.day)}")
```

Resposta:

```
>>  A data é =>  2021-03-28
>>  O ano da data é 2021 e seu tipo é <class 'int'>
>>  O ano da data é 3 e seu tipo é <class 'int'>
>>  O ano da data é 28 e seu tipo é <class 'int'>
```

1.7. Função datetime.now()

Essa função recebe como parâmetro apenas o fuso horário e retorna o exato momento que a função roda no código. Por padrão, ela irá passar o fuso horário local como argumento. Caso você rode o código no seu computador, irá funcionar normalmente, mas quando for rodar um código na nuvem da AWS, isso será um problema caso você precise do horário do Brasil "agora".

No código abaixo, estamos importando do módulo "datetime" a classe "datetime" que contém a função "now".

Exemplo:

```
import datetime
import pytz

tz = pytz.timezone('America/Sao_Paulo')
date = datetime.datetime.now(tz)
print(date)
```

Resposta:

```
>>   2022-08-15 12:54:35.321670-03:00
```

2. Criando timedeltas

2.1. Função timedelta()

É uma função do módulo datetime. Essa função representa as diferenças no tempo, que podem variar em minutos, horas ou dias (Dia é a unidade máxima). Essa função recebe como parâmetro o intervalo de tempo desejado e a sua unidade em minutos, horas ou dias.

Exemplo:

```
from datetime import timedelta
from dateutil.relativedelta import relativedelta
from datetime import datetime

timedelta_teste = timedelta(days = 1) #dia é a maior escala
amanha = datetime.now() + timedelta_teste
print(" Intervalo de tempo => ",timedelta_teste)
print(" Dia de hoje + Intervalo de tempo => ",amanha)
```

Resposta:

```
>>   1 day, 0:00:00
>>   2022-08-19 15:55:38.472909
```

2.2. Função relativedelta()

Apesar de ser muito parecido com a função timedelta(), a função relativedelta() não é uma função do módulo datetime. Mas assim como a sua irmã, ela também adiciona ou retira um certo intervalo de tempo de uma data, podendo ser em minutos, horas, dias, meses e anos, sendo esse último a escala máxima.

Exemplo:

```
from dateutil.relativedelta import relativedelta
from datetime import date

final_desse_ano = date(2022, 12, 31)
final_ano_que_vem =  final_desse_ano + relativedelta(years = 1)
print("A data inicial é => ", final_desse_ano)
print("O intervalo de tempo é => ", relativedelta(years=1))
print(" A data final com o intervalo de tempo adicionado é => ",final_ano_que_vem)
```

Resposta:

```
>>   A data inicial é =>  2022-12-31
>>   O intervalo de tempo é =>  relativedelta(years=+1)
>>    A data final com o intervalo de tempo adicionado é =>  2023-12-31
```

3. Módulo pytz

É um módulo que é muito utilizado em conjunto com o módulo datetime. Ele lida com os fusos horários dos países. Como qualquer módulo, para instalá-lo no Python basta digitar no terminal “pip install pytz”.

3.1. Função all_timezones()

Essa função retorna uma lista com todos os lugares que contém os fusos horários. Ao percorrer essa lista com a estrutura for e printar na tela todos os fuso horários, conseguirá ter uma visualização melhor de todos os fuso horários possíveis. O que melhor se enquadra ao escritório da Edufinance é “America/Sao_Paulo”.

Exemplo:

```
from time import time
import pytz
for tz in pytz.all_timezones:
    print(tz)
```

Resposta:

```
>>      Asia/Phnom_Penh
        Asia/Pontianak
        Asia/Pyongyang
        Asia/Qatar
          .
          .
       America/New_York
        America/Sao_Paulo
          .
          .
          .
```

Exemplo2:

```
import pytz
from datetime import datetime

tz_sp = pytz.timezone('America/Sao_Paulo')
tz_ny = pytz.timezone('America/New_York')
data_agora_sp = datetime.now(tz_sp)
data_agora_ny = datetime.now(tz_ny)

print("Horário em São Paulo => ", data_agora_sp)
print("Horário em Nova York => ", data_agora_ny)
```

Resposta2:

```
>>  Horário em São Paulo =>  2022-08-18 16:11:48.902076-03:00
>>  Horário em Nova York =>  2022-08-18 15:11:48.902093-04:00
```

4. Formatação de datas (gráficos, leituras, etc)

O método strptime() cria uma data completa a partir de uma string mas esse método, por si só, não identifica o padrão da data. Dessa forma, você deve passar como parâmetro a string e o formato da data da string. Abaixo são apenas alguns exemplos, você deve se adequar ao seu caso em específico por isso segue a tabela abaixo com todos os exemplos :

Exemplo:

```
from datetime import datetime

data_texto = "18/04/1987"
data_texto2 = "18-04-87"
data_texto3 = "jan-01"
data_python = datetime.strptime(data_texto, '%d/%m/%Y')
data_python2 = datetime.strptime(data_texto2, '%d-%m-%y')
data_python3 = datetime.strptime(data_texto3, '%b-%y')

print("Data completa => ",data_python)
print("Data completa => ",data_python2)
print("Data completa => ",data_python3)
```

Resposta:

```
>>  1987-04-18 00:00:00
>>  1987-04-18 00:00:00
>>  2001-01-01 00:00:00
```

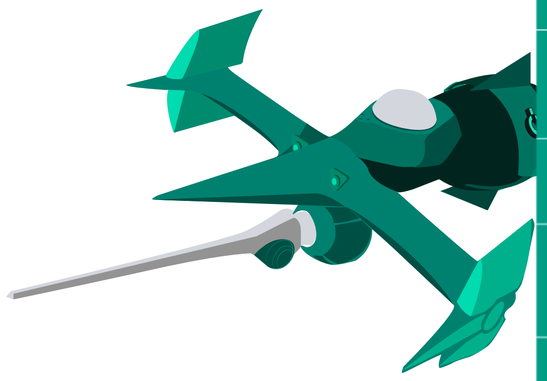


Tabela com todas as formatações de datas.

Diretiva	Significado	Exemplo
%a	Dias da semana como nomes abreviados da localidade.	Sun, Mon, ..., Sat (Dias da semana em inglês).
%A	Dia da semana como nome completo da localidade.	Sunday, Monday, ..., Saturday (Dias da semana completos em inglês).
%w	Dia da semana como um número decimal, onde 0 é domingo e 6 é sábado.	0, 1, ..., 6
%d	Dia do mês como um número decimal com zeros à esquerda.	01, 02, ..., 31
%b	Mês com o nome da localidade abreviado.	Jan, Feb, ..., Dec (Meses do ano abreviados em inglês)
%B	Mês como nome completo da localidade.	January, February, ..., December (Meses do ano completos em inglês)
%m	Mês como um número decimal com zeros à esquerda.	01, 02, ..., 12
%y	Ano sem século como um número decimal com zeros à esquerda.	00, 01, ..., 99
%Y	Ano com século como um número decimal.	0001, 0002, ..., 2013, 2014, ..., 9998, 9999

%H	Hora (relógio de 24 horas) como um número decimal com zeros à esquerda.	00, 01, ..., 23
%I	Hora (relógio de 12 horas) como um número decimal com zeros à esquerda.	01, 02, ..., 12
%p	Equivalente da localidade a AM ou PM.	AM, PM (Referência aos dois períodos do dia em inglês)
%M	Minutos como um número decimal, com zeros à esquerda.	00, 01, ..., 59
%S	Segundos como um número decimal, com zeros a esquerda.	00, 01, ..., 59
%f	Microssegundos como um número decimal, com arredondamento de 6 casas decimais.	000000, 000001, ..., 999999
%z	Diferença UTC no formato ±HHMM[SS[.ff ff]] (string vazia se o objeto é ingênuo).	(vazio), +0000, -0400, +1030, +063415, -030712.345216
%Z	Nome do fuso horário (string vazia se o objeto é ingênuo).	(vazio), UTC, GMT
%j	Dia do ano como um número decimal, com zeros à esquerda.	001, 002, ..., 366
%U	Número da semana do ano (domingo como o primeiro dia da semana) como um número decimal preenchido com zeros. Todos os dias de um novo ano que antecede o primeiro domingo são considerados na semana 0.	00, 01, ..., 53
%W	Número da semana do ano (segunda-feira como o primeiro dia da semana) como um número decimal preenchido com zeros. Todos os dias de um novo ano que antecede a primeira segunda-feira são considerado na semana 0.	00, 01, ..., 53
%c	Representação de data e hora apropriada da localidade.	Tue Aug 16 21:30:00 1988 (Dia da semana e o mês em inglês)
%x	Representação de data apropriada de localidade.	08/16/88
%X	Representação de hora apropriada da localidade.	21:30:00
%%	Um caractere literal '%'.	%