1. Instalação do Python no Windows

Site para download: https://www.python.org/downloads/ (baixar o python 3 na versão mais atual).

- Abra o instalador:
  - Na primeira tela é importante marcar o *checkbox **Add python 3.*\ to PATH***;
  - Na próxima tela seleciona instalação customizada;
  - Na tela seguinte todos os *checkbox* podem permanecer marcados;
  - Na próxima tela selecione o diretório de instalação;
  - Agora é só clicar *Install*.

1.1 Verificando a instalação do Python

- Abra o terminal, pode ser o **Prompt de Comando** ou o **Power Shell** e digite:
  - python -V

2. Criando um ambiente virtual e instalando pacotes utilizando o pip

Link da documentação: https://packaging.python.org/guides/installing-using-pip-and-virtual-environments/.

A partir do python 3.3 o modulo nativo mais usado para a criação e gerenciamento de ambientes virtuais é o **venv**. Através dos ambientes virtuais é possível separar a instalação de pacotes para diferentes projetos.

2.1. Criando um ambiente virtual no Ubuntu

- Abra o terminal e intale os pacotes python3-pip e python3-venv:
  - **sudo apt install python3-pip python3-venv**
- Crie um diretório para o projeto, entre no diretório e digite o seguinte comando:
  - **python3 -m venv env** (é muito comum utilizar **python -m venv .venv**, dessa maneira colocando o ambiente virtual na pasta oculta **.venv**)

2.2 Ativando e desativando o ambiente virtual no Windows

- No diretório do projeto digite o comando:
  - **source .env/bin/activate**
- Para sair do ambiente virtual é só digitar

- **deactivate**

2.3 Instalando o jupyterlab no ambiente virtual

Link para o site do Jupyter: https://jupyter.org/install

- O **jupyterlab** é um pacote que fornece acesso a uma interface com mais utilitários para usar o **jupyter notebook**.
- Primeiro ative o ambiente virtual
  - **source .env/bin/activate**
- Dentro do ambiente virtual é só instalar o pacote utilizando o gerenciador de pacotes **pip** executando o comando abaixo:
  - **python -m pip install jupyterlab**
- Ainda dentro do ambiente, para abrir o **jupyter notebook** digite o comando:
  - **jupyter lab**
- O notebook vai abrir no navegador automaticamente como um serviço local.
- Para se desconectar do notebook, no terminar pressione as teclas **Ctrl + c**.

3. Primeiro programa em Python

Para imprimir um mensagem na tela basta utilizar a função print(). Segue um exemplo abaixo:

In [1]:
```
print("mentoria de python")
```

```
mentoria de python
```

3.1 Tipagem e Variáveis no Python

3.1.1 A função type() pode ser usada para verificar o tipo de variáveis e valores:

In [2]:
```
print(type(1))
print(type(1.5))
print(type("mentoria de python"))
```

```
<class 'int'>
<class 'float'>
<class 'str'>
```

3.1.2 Para atribuir valores a variáveis em Python basta utilizar o sinal de igualdade (=):

In [3]:
```
num_1 = 1
num_2 = 1.5
string = "mentoria de python"
```

3.1.3 Exibindo os valores das variáveis com a função **print()**

In [4]:
```
print(num_1)
print(num_2)
print(string)
```

```
1
1.5
mentoria de python
```

3.1.4 Verificando o tipo das variáveis com a função **type()**

In [5]:
```
print(type(num_1))
print(type(num_2))
print(type(string))
```

```
<class 'int'>
<class 'float'>
<class 'str'>
```

3.1.5 Operações Aritméticas

- + : soma
- - : subtração
- * : multiplicação
- / : divisão
- // : divisão inteira
- % : módulo
- ** : exponenciação

In [6]:
```
# declarando as variáveis 'a' e 'b' com valores '3' e '2' respectivamente
a = 3
b = 2

# atribudindo o resultado das operações com as variáveis 'a' e 'b' a outras v
soma = a + b
sub = a - b
mult = a * b
div = a / b
div_int = a // b
mod = a % b
exp = a ** b

"""
Ao utilizar o caractere 'f' antes das aspas na função print(),
tudo que estiver entre {} (valores ou variáveis) será formatado
de acordo com a sentença dada como entrada para a função print().
"""
print(f"SOMA ({a} + {b}): {soma}")
print(f"SUBTRAÇÃO ({a} - {b}): {sub}")
print(f"MULTIPLICAÇÃO ({a} * {b}): {mult}")
print(f"DIVISÃO ({a} / {b}): {div}")
print(f"DIVISÃO INTEIRA ({a} // {b}): {div_int}")
print(f"MÓDULO ({a} % {b}): {mod}")
print(f"EXPONENCIAÇÃO ({a} ** {b}): {exp}")
```

```
SOMA (3 + 2): 5
SUBTRAÇÃO (3 - 2): 1
MULTIPLICAÇÃO (3 * 2): 6
```

```
DIVISÃO (3 / 2): 1.5
DIVISÃO INTEIRA (3 // 2): 1
MÓDULO (3 % 2): 1
EXPONENCIAÇÃO (3 ** 2): 9
```

4. Listas, Tuplas e Dicionários

Link para documentação: **https://docs.python.org/3.9/library/stdtypes.html#sequence-types-list-tuple-range**

4.1 Listas:

Uma lista é uma sequência de valores indexados a partir de 0 (por exemplo, uma lista de tamanho 5, possui índices de 0 à 4).

In [7]:
```
# declarando uma lista
lista_1 = [2, 3, 4, 5, 6]
print(f"acessando o primeiro elemento da lista_1: {lista_1[0]}")
```

```
acessando o primeiro elemento da lista_1: 2
```

In [8]:
```
# Pode-se usar também operações de sequências (slices)
# Note que ao fazer um slice do índice 1 ao 3, são retornados os valores dos
# O slice, na realidade, acessa os valores a partir do índice inicial até o u
print(f"slice de 1 até 3: {lista_1[1:4]}")
```

```
slice de 1 até 3: [3, 4, 5]
```

In [9]:
```
# Existe ainda a possibilidade de acessar os valores de uma lista a partir do
print(f"último elemento: {lista_1[-1]}")
print(f"penúltimo elemento: {lista_1[-2]}")
```

```
último elemento: 6
penúltimo elemento: 5
```

In [10]:
```
# Uma lista pode ser inicializada vazia
lista_2 = [] # ou lista_2 = list()

# Inserindo valores na lista_2
lista_2.append("a")
lista_2.append("b")
print(f"valores da lista_2: {lista_2}")
```

```
valores da lista_2: ['a', 'b']
```

In [11]:
```
# As listas podem possuir tipos diferentes de dados
# Ainda, é possível concatenar listas (aqui, a lista_2 é concatenada ao final
lista_1.extend(lista_2)
print(f"lista_1 concatenada com a lista_2: {lista_1}")
```

```
lista_1 concatenada com a lista_2: [2, 3, 4, 5, 6, 'a', 'b']
```

In [12]:
```
# Removendo itens de uma lista
lista_1.remove("a")
lista_1.remove("b")
print(f"lista_1: {lista_1}")
```

```
lista_1: [2, 3, 4, 5, 6]
```

In [13]:
```
# Atribuindo a concatenação de duas listas à uma terceira lista
# Note que o operador '+' pode ser usado para concatenar listas
lista_3 = lista_1 + lista_2
print(f"lista_3: {lista_3}")
```

```
lista_3: [2, 3, 4, 5, 6, 'a', 'b']
```

In [14]:
```
# Também é possível alterar o valor de um elemento de uma lista
lista_3[-1] = "z" # mudando o último elemento da lista_3 de 'b' para 'z'
print(f"lista_3: {lista_3}")
```

```
lista_3: [2, 3, 4, 5, 6, 'a', 'z']
```

4.2 Tuplas

Tuplas são sequências imutáveis, ou seja, não é permitido mudar os valores dos itens de uma tupla.

In [15]:
```
# Declarando uma tupla
dias = ("domingo", "segunda", "terça", "quarta", "quinta")
print(f"dias: {dias}")
```

```
dias: ('domingo', 'segunda', 'terça', 'quarta', 'quinta')
```

In [16]:
```
# Elementos podem ser adicionados a uma tupla
dias = dias + ("sexta", "sábado")
print(f"dias: {dias}")
```

```
dias: ('domingo', 'segunda', 'terça', 'quarta', 'quinta', 'sexta', 'sábado')
```

In [17]:
```
# O acesso aos valores das tuplas é parecido com o acesso aos valores de list
print(f"dia 1: {dias[0]}")
print(f"dias 2 ao 7: {dias[1:]}")
```

```
dia 1: domingo
dias 2 ao 7: ('segunda', 'terça', 'quarta', 'quinta', 'sexta', 'sábado')
```

4.3 Dicionários

Os dicionários são estruturas de dados em que os valores podem ser acessados através de chaves. O exemplo abaixo é um dicionário cuja a chave é o nome da pessoa e o valor é o peso da pessoa.

In [18]:
```
# Inicializando um dicionário
peso = {
    "Alex": 80.0,
    "Aline": 58.5,
    "Marcelo": 60.0,
    "Maria": 60.0,
}
print(f"Dicionário: {peso}")
```

```
Dicionário: {'Alex': 80.0, 'Aline': 58.5, 'Marcelo': 60.0, 'Maria': 60.0}
```

In [19]:
```
# Adicionando items ao dicionário
peso["Ulisses"] = 75.9
print(f"Dicionário: {peso}")
```

```
Dicionário: {'Alex': 80.0, 'Aline': 58.5, 'Marcelo': 60.0, 'Maria': 60.0, 'Ul
isses': 75.9}
```

In [20]:
```
# É possível acessar apenas as chaves de um dicionário
print(f"Chaves do dicionário: {peso.keys()}")
```

```
Chaves do dicionário: dict_keys(['Alex', 'Aline', 'Marcelo', 'Maria', 'Ulisse
s'])
```

In [21]:
```
# Assim como apenas os valores
print(f"Valores do dicionário: {peso.values()}")
```

```
Valores do dicionário: dict_values([80.0, 58.5, 60.0, 60.0, 75.9])
```

In [22]:
```
# Exemplo de iteração nas chaves. O exemplo abaixo retorna a média dos pesos.
# A função len() retorna o tamanho de um sequência (Nota: a mesma pode ser us
num_items = len(peso)
peso_total = 0
for key in peso.keys():
    peso_total += peso[key]
peso_médio = peso_total / num_items
print(f"Peso médio: {peso_médio}")
```

```
Peso médio: 66.88
```