



# FUNDAMENTOS DE BIG DATA E DATA ANALYTICS COM PYTHON



# AULA 01

# O que é Python?



Python é uma linguagem de alto nível, de código aberto, tipagem dinâmica e forte;  
Foi criada por Guido Van Rossum, em 1991.  
O nome é inspirado na série britânica Monty Python.



# Quais suas aplicações?



- Aplicação web, desktop e mobile;
- Cálculos científicos;
- Computação gráfica;
- Automação de sistema;
- Mineração de dados;
- Big data;
- Machine Learning;
- Processamento de textos;
- Tratamento e reconhecimento de imagens;
- Animações 3D.

# Instalando o Python



Nesta seção vamos te mostrar como instalar o pacote **Anaconda** que já contém o **Jupyter**, mas tem várias outras bibliotecas e recursos já inseridos para que você não tenha que ficar instalando separadamente depois.

Agora vamos a como Instalar Jupyter Notebook, primeiro basta ir ao no Google e buscar por Anaconda e entrar neste [link](#), ou neste endereço <https://www.anaconda.com/>.

# Instalando o Python



## Anaconda Installers



Windows

**Python 3.10**

↓ 64-Bit Graphical Installer (786 MB)



Mac

**Python 3.10**

↓ 64-Bit Graphical Installer (599 MB)

↓ 64-Bit Command Line Installer (601 MB)

↓ 64-Bit (M1) Graphical Installer (564 MB)

↓ 64-Bit (M1) Command Line Installer (565 MB)



Linux

**Python 3.10**

↓ 64-Bit (x86) Installer (860 MB)

↓ 64-Bit (Power8 and Power9) Installer (434 MB)

↓ 64-Bit (AWS Graviton2 / ARM64) Installer (618 MB)

↓ 64-bit (Linux on IBM Z & LinuxONE) Installer (360 MB)

# Variáveis

Variáveis são espaços na memória que utilizamos para armazenar valores. Em outras palavras, variável é o nome que damos a um valor ou expressão. Em Python, usamos o sinal de igual, "=", para atribuir um valor à uma variável.

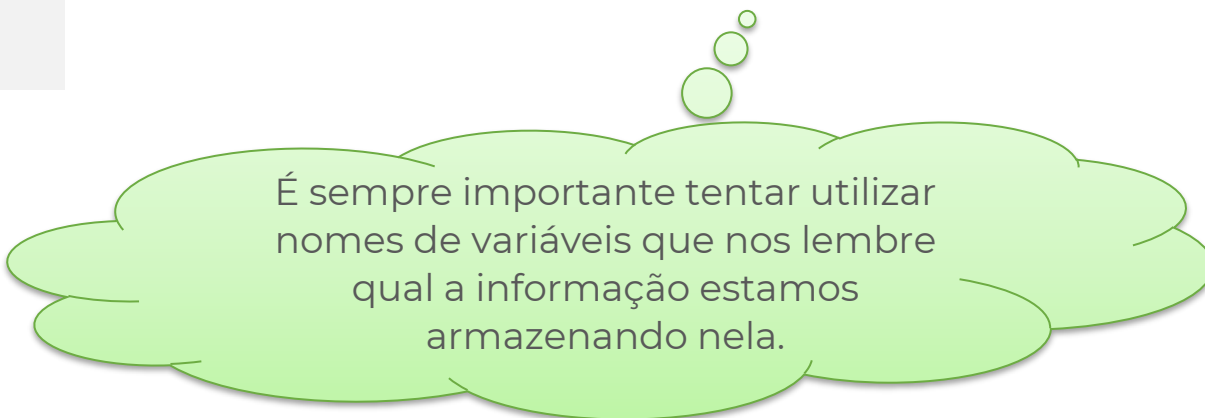
```
A = 10
```

```
B = "Banana"
```

```
C = 0.5
```

```
fruta = "Banana"
```

```
xpto = "Banana"
```



É sempre importante tentar utilizar nomes de variáveis que nos lembre qual a informação estamos armazenando nela.

# Variáveis – Regras



Em Python, existem algumas regras para declararmos variáveis:

- Podem ser usadas letras e algarismos;
- Nunca devem começar com um algarismo;
- Não podem utilizar palavras-chaves naturais;
- No Python, por exemplo: If, While, For etc;
- Um nome de variável deve começar com uma letra ou o caractere de sublinhado.



# Palavras chaves reservadas



Para saber quais são as palavras chaves reservadas, digite:

```
import keywords  
print(keywords.kwlist)
```

A screenshot of an Anaconda Prompt terminal window. The title bar reads "Anaconda Prompt (anaconda3) - python". The terminal shows the following commands and output:

```
(base) C:\Users\dell>python  
Python 3.9.13 (main, Aug 25 2022, 23:51:50) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32  
Type "help", "copyright", "credits" or "license" for more information.  
>>> import keyword  
>>> print(keyword.kwlist)  
['False', 'None', 'True', '__peg_parser__', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def',  
, 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal',  
'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']  
>>>
```

# Variáveis - Tipos



Na programação, compreender os tipos de dados é um conceito importante. Variáveis podem armazenar dados de diferentes tipos, e cada tipo permite fazer coisas diferentes.

O Python tem os seguintes tipos de dados integrados por padrão, nestas categorias:

- **Texto:** str;
- **Numéricos:** int, float, complex;
- **Sequência:** list, tuple, range;
- **Mapeamento:** dict;
- **Conjunto:** set, frozenset;
- **Booleano:** bool;
- **Binário:** bytes, bytearray, memoryview.

# Tipo inteiro (int)



O tipo inteiro é um tipo composto por caracteres numéricos (algarismos) inteiros.

É um tipo usado para um número que pode ser escrito sem um componente decimal, podendo ter ou não sinal, isto é: ser positivo ou negativo.

Por exemplo, 21, 4, 0, e -2048 são números inteiros, enquanto 9.75, 1/2, 1.5 não são.

## Entrada

```
idade = 18  
ano = 2002  
  
print(type(idade))  
print(type(ano))
```

## Saída

```
<class 'int'>  
<class 'int'>
```

# Ponto Flutuante (float)



É um tipo composto por caracteres numéricos (algarismo) decimais.

O famoso ponto flutuante é um tipo usado para números racionais (números que podem ser representados por uma fração) informalmente conhecido como “número quebrado”.

## Entrada

```
altura = 1.80  
peso = 73.55  
  
print(type(peso))  
print(type(altura))
```

## Saída

```
<class 'float'>  
<class 'float'>
```

# Complexo (complex)



Tipo de dado usado para representar números complexos (isso mesmo, aquilo que provavelmente estudou no terceiro ano do ensino médio).

Esse tipo normalmente é usado em cálculos geométricos e científicos.

Um tipo complexo contem duas partes: a parte real e a parte imaginária, sendo que a parte imaginária contem um  $j$  no sufixo.

# Complexo (complex)



A função `complex(real[, imag])` do Python possibilita a criação de números imaginários passando como argumento: `real`, que é a parte Real do número complexo e o argumento opcional `imag`, representando a parte imaginária do número complexo.

## Entrada

```
a = 5+2j
b = 20+6j

print(type(a))
print(type(b))
print(complex(2, 5))
```

## Saída

```
<class 'complex'>
<class 'complex'>

(2+5j)
```

# String (str)



É um conjunto de caracteres dispostos numa determinada ordem, geralmente utilizada para representar palavras, frases ou textos.

## Entrada

```
nome = 'Guilherme'  
profissao = 'Engenheiro de Software'  
  
print(type(profissao))  
print(type(nome))
```

## Saída

```
<class 'str'>  
<class 'str'>
```

# Boolean (bool)



Tipo de dado lógico que pode assumir apenas dois valores: falso ou verdadeiro (False ou True em Python).

Na lógica computacional, podem ser considerados como 0 ou 1.

## Entrada

```
fim_de_semana = True
feriado = False

print(type(fim_de_semana))
print(type(feriado))
```

## Saída

```
<class 'bool'>
<class 'bool'>
```



# Listas (list)



Tipo de dado muito importante e que é muito utilizado no dia a dia do desenvolvedor Python!

Listas agrupam um conjunto de elementos variados, podendo conter: inteiros, floats, strings, outras listas e outros tipos.

Elas são definidas utilizando-se colchetes para delimitar a lista e vírgulas para separar os elementos, veja alguns exemplo abaixo:

## Entrada

```
alunos = ['Amanda', 'Ana', 'Bruno', 'João']  
notas = [10, 8.5, 7.8, 8.0]
```

```
print(type(alunos))  
print(type(notas))
```

## Saída

```
<class 'list'>  
<class 'list'>
```

# Listas (list) – Acessando itens



Os elementos de uma lista podem ser acessados através dos índices:

```
lista_compra[0]
```

```
lista2 = lista_coisas[0:3]
```

É possível, inclusive, criar uma lista de listas!

```
Lista_geral = [lista_compra, lista_coisas, lista2]
```

# Listas (list) – Acessando itens



```
x = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
zero = x[0] # igual a 0, as listas são indexadas a partir de 0
```

```
one = x[1] # igual a 1
```

```
nine = x[-1] # igual a 9, 'Pythonic' para o último elemento
```

```
eight = x[-2] # igual a 8, 'Pythonic' para o penúltimo elemento
```

# Lista (list) – Adicionando itens



Para adicionar elementos à uma lista já existente, usamos o comando `append()`:

```
lista_compra.append('batata')
```

# Lista (list) – Conversão para string



Já para criar uma string com todos os elementos de uma lista, podemos utilizar o comando `join()`:

```
supermercado = 'e'.join('lista_compra')
```

# Lista (list) – string para lista



Ainda com strings, podemos criar uma lista a partir de uma string com o comando `split()`:

```
feira = 'batata, cebola, tomate, pepino'
lista_feira = feira.split(',')
```

# Tuplas (tuple)



Assim como Lista, Tupla é um tipo que agrupa um conjunto de elementos.

Porém sua forma de definição é diferente: utilizamos parênteses e também separado por vírgula.

A diferença para Lista é que Tuplas são imutáveis, ou seja, após sua definição, Tuplas não podem ser modificadas.

## Entrada

```
valores = (90, 79, 54, 32, 21)
pontos = (100, 94.05, 86.8, 62)

print(type(valores))
print(type(pontos))
```

## Saída

```
<class 'tuple'>
<class 'tuple'>
```

# Tuplas (tuple) – Outros exemplos



```
my_list = [1, 2]

my_tuple = (1, 2)

other_tuple = 3, 4

my_list [1] = 3 # my_list é [1, 3]

try:

    my_tuple[1] = 3

except TypeError:

    print("cannot modif a tuple")
```



# Tuplas (tuple) – Outros exemplos



As tuplas são uma forma eficaz de usar funções para retornar múltiplos valores:

```
def sum_and_product (x, y):  
  
    return (x+y), (x * y)  
  
sp = sum_and_product (2, 3) #sp é (5, 6)  
  
s, p = sum_and_product (5, 10) #s é 15, p é 50
```

# Dicionários (dict)



Dict é um tipo de dado muito flexível do Python.

Eles são utilizados para agrupar elementos através da estrutura de **chave** e **valor**, onde a chave é o primeiro elemento **seguido por dois pontos** e pelo valor.

Vamos ver alguns exemplos:

## Entrada

```
altura = {'Amanda': 1.65, 'Ana': 1.60, 'João': 1.70}
peso = {'Amanda': 60, 'Ana': 58, 'João': 68}

print(type(altura))
print(type(peso))
```

## Saída

```
<class 'dict'>
<class 'dict'>
```

# Dicionários (dict) – Outros exemplos



```
cadastro = {  
  
    'nome': "Tânia",  
  
    'idade': 35,  
  
    'fruta': "uva",  
  
    'cor': "roxa",  
  
    'musica': "pop"  
  
}
```

# Dicionários (dict) – Outros exemplos



Podemos criar um dicionário a partir de uma lista de tuplas usando a função `dict()`.

```
tupla_1 = ('nome', 'Tânia')  
  
tupla_2 = ('idade', 35)  
  
tupla_3 = ('fruta', 'uva')  
  
tupla_4 = ('cor', 'roxa')  
  
tupla_5 = ('musica', 'pop')  
  
lista_tuplas = [tupla_1, tupla_2, tupla_3, tupla_4, tupla_5]  
  
cadastro = dict (lista_tuplas)
```

# Dicionários (dict) – Editando informações



Podemos editar ou adicionar um valor em um dicionário:

```
## editando um valor
```

```
cadastro ['fruta'] = 'laranja'
```

Inserindo uma nova chave-valor:

```
cadastro ['sobrenome'] = 'Silva'
```

# Dicionários (dict) – Excluindo um elemento

Podemos editar ou adicionar um valor em um dicionário:

```
del cadastro['musica']
```

# Dicionários (dict) – Atualizando dados



Podemos atualizar mais de um valor usando a função update():

```
cadastro.update({'nome': "Ana", 'idade': 27, 'fruta': 'abacaxi'})
```