

# Python Fundamentals

## Módulo 1

Clodonil Trigo (@clodonil).

# Python Fundamentals

“ Python é uma linguagem de programação de alto nível, `interpretada`, imperativa, orientada a objetos, de `tipagem dinâmica` e `forte`. ”

Criado por Guido van Rossum em 1989.

# Filosofia da Linguagem

- Bonito é melhor que feio
- Explícito é melhor que implícito;
- Simples é melhor que complexo;
- Linear é melhor que aninhado;
- Se a implementação é difícil de explicar, a ideia é ruim;
- Se a implementação é fácil de explicar, talvez a ideia seja boa;

# Ambiente de Desenvolvimento

- **Python 3**
  - VS Code (Microsoft)
  - TextEdit
  - Notepad++
  - Vim
  - PyCharm

# Segregação de Ambientes

“ Nunca se deve confiar em bibliotecas pré-instaladas, sempre deve ser declarado através de um manifesto todas as dependências envolvidas na aplicação, conforme descrito [12factor](#). ”

- VirtualEnv
- Máquinas Virtuais

# VirtualEnv

## Criado um VirtualEnv

```
$ python -m venv workspace_projeto
```

## Ativando o VirtualEnv

```
$ source workspace_projeto/bin/active
```

## Desativando o VirtualEnv

```
$ source workspace_projeto/bin/deactive
```

# Sintaxe do Python

Python tem uma sintaxe simples e direta. Porém, um ponto de dificuldade aos iniciantes, são os blocos de códigos.

# Bloco de Código

Blocos de códigos em outras linguagens são marcados por { e }, como por exemplo em C:

```
if (x > 5){  
printf("maior");  
}else {  
printf("menor");  
}
```



# Bloco de Código

Ruby utilizam o `do` e `end` para limitar o bloco de código:

```
if x > 5  
  echo "Maior"  
else  
  echo "Menor"  
end
```

# Bloco de código

Em Python bloco de código é realizado por espaçamento. Se os espaçamentos não forem realizados corretamente, o código não vai funcionar, ou seja o código fica completamente indentado.

```
if x > 5:  
    print("Maior")  
else  
    print("Menor")
```

# Variável

Os nomes das variáveis é conforme a vontade do programador, com nomes longos, contendo letras e números.

No entanto, elas devem necessariamente começar com 'letras minúsculas'.

and	def	exec	if	not	return
assert	del	finally	import	or	try
break	elif	for	in	pass	while
class	else	from	is	print	yield
continue	except	global	lambda	raise	

# Variável (exemplos)

```
dados=10
```

Atribuição também pode ser realizado em cadeia, conforme o exemplo:

```
dados=x=filho=10
```

Outra forma de realizar atribuição em cadeia:

```
nome, idade, tel = 'jose', 10, '56945-2342'
```

# Estrutura de Dados

- String;
- Number (Integer e Float);
- List;
- Tuple;
- Dictionary;
- Boolean

# String

Exemplo:

```
texto="As vezes você tem que se levantar sozinho e seguir em frente"
```

Para declarar múltiplas linhas utilize 3 aspas simples.

Exemplo:

```
texto='''  
Tudo e possível.  
O impossível apenas demora mais  
'''
```

# Manipulando String

Exemplo:

```
texto="As vezes você tem que se levantar sozinho e seguir em frente"  
# Recuperar o caractere da posição 5  
print(texto[6])    # Retorno "e"  
  
# Recuperar do inicio da string ate posição 10  
print(texto[0:10]) # Retorno "As vezes v"  
  
# Recuperar da posição 10 até a posição 15  
print(texto[10:15]) # retorno "ocê t"
```

# Numbers

```
a = 5
print(a, "do tipo", type(a))

a = 2.0
print(a, "do tipo", type(a))

a = 1+2j
print(a, "número complexo?", isinstance(1+2j, complex))
```

“ No exemplo utilizamos a função `type()` para mostrar o tipo da classe da variável. Também usamos a função `isinstance()` para comparar o tipo da variável.

”



# List

Lista é Python é uma sequência de item, equivale aos arrays de outras linguagens.

```
lista=[10,60,'jose','45234234',50.9]  
# Mostrando a lista  
print("Conteúdo da lista", lista)
```

Estrutura da Lista:

0	1	2	3	4
10	60	'jose'	'45234234'	50.9

# List

Acessando elementos da lista

0	1	2	3	4	5	6	7
10	30	50	90	100	1	5	19

```
lista=[10,30,50,90,100,1,5,19]

# lista[2] = 50
print("lista[2] = ", lista[2])

# lista[0:3] = [10, 30, 50]
print("lista[0:3] = ", lista[0:3])

# lista[5:] = [1, 5, 19]
```

# List

Alterando elementos da lista

```
a = [1, 2, 3]
```

```
a[2]=4
```

```
# [1, 2, 4]
```

```
print(a)
```

# Tuple

As tuplas são uma sequência de itens, semelhante a uma lista. A diferença que as tuplas são imutáveis.

```
t = (10, 40, 'jose', 'maria', 6.5)
print(t[1])
```

**Tuple é mais rápido que uma lista**

# Dictionary

Dicionário são estrutura de dados com um par conhecido como `chave/valor`.

```
login = {"user": "jose", "password": "okri"}  
  
print("usuario", login['user'])
```

Hash:

user	password
jose	okri

# Set

Set é uma coleção não ordenada de itens exclusivos (não pode ter itens repetidos).

```
posicao_chegada={3,4,1,7,8}  
print(posicao_chegada)
```

# Frozenset

**Frozen set** são conjuntos similares ao set. A diferença que frozenset são imutáveis, portanto não podem ser modificados.

```
lista = ('a', 'e', 'i', 'o', 'u')  
  
fSet = frozenset(lista)  
print('O frozen set é:', fSet)
```

# Um outro exemplo:

```
# random dictionary
person = {"name": "John", "age": 23, "sex": "male"}

fSet = frozenset(person)
print('The frozen set is:', fSet)
```



# Boolean

O tipo de dados Boolean em Python são criados pela classe `bool` que logicamente aceita 2 valores constantes `True` e `False`.

```
print(type(True))
```

Os booleanos também podem ser apresentados por inteiros sendo:

- 1 para `True`;
- 0 para `False`;

# Conversão entre tipos

Descrição	Função	Exemplo
Converte para intero.	int(x)	num=int(a)
Converte para float.	float(x)	num = float(a)
Converte para String	str(x)	letra = str(num)
Converte para uma tuple	tuple(s)	t_lista = tuple(lista)
Descrição	Função	Exemplo

# Operadores

Igualmente as outras linguagens de programação, as operações básicas de matemáticas são realizadas pelos operadores **+**, **-**, **/** e **\***.

Descrição	Operador
adição	+
subtração	-
multiplicação	/
divisão	*

# Exemplo

A utilização é bem simples.

```
soma = 10 + 5
```

Python também segue as precedências definidas pela Matemática, porém sempre recomendamos utilizar parentes `( )` para separar os operadores, para tornar mais legível.

Por exemplo:

```
result = 2+2*2    # Resultado 6  
#dessa forma fica mais legível  
result = (2+2)*2  # Resultado 8
```

# Operadores de Atribuição

Operadores que fazem operação e atribuição.

Descrição	Operador
soma e atribui	<code>+=</code>
subtrai e atribui	<code>-=</code>
dividi e atribui	<code>*=</code>
multiplica e atribui	<code>/=</code>

# Exemplo:

```
a = 10  
a = a + 10  
a += 10
```

Além dos operadores básicos, também temos os operadores para exponenciação, extração do módulo da divisão, parte inteira de uma divisão.

Descrição	Operador
exponenciação	**
Descrição	Operador

# Exemplo

A sintaxe para utilização desses operadores.

```
result = 2 ** 2 # resultado 4  
result = 5 // 2 # resultado 2  
result = 5 % 2 # resultado 1
```

Esses operadores podem ser utilizados com outras estruturas de dados tais como string e list.

Um exemplo de utilização com string:

```
result = "-" * 50  
result = "Ola" + "Mundo"
```

# Operadores com Lista:

```
lista = [1,2,3] * 10  
todos_numeros = [2,4,6,8] + [1,3,5,7]
```

Além dos operadores matemáticos, também temos os operadores lógicos que retornam `True` e `False`.

Descrição	Operador
Maior que	>
Menor que	<
Descrição	Operador



# Operadores Lógicos (Exemplos)

A utilização dos operadores lógicos também é bem simples.

```
a=10 > 5
b=100 == 200
if 200 > 100:
    print("maior")
if a or b:
    print("Tudo certo")
if a and b:
    print("Tudo errado")
x = not(a and b)
```

# Operadores Contido e Identidade

Os operadores `contido` e teste de `identidade`.

Descrição	Operador
Contido em	<code>in</code>
Identidade	<code>is</code>
Criar funções	<code>lambda</code>

# Operador Contido (IN)

Operador `in` para verificar se um valor está em uma lista.

```
x = 10 in [3,4,5,10]  
y = 50 in [3,4,5,10]
```

x => True

y => False

# Identidade (IS)

Operador `is` é para verificar a identidade de um objeto. Verificar se realmente é o mesmo objeto.

```
a=10
b=a
k = a is b
print(id(a))
print(id(b))
a=[10,2,3]
b=[10,2,3]
k = a is b
```

# Lambda

O operador lambda é bastante útil, com ele podemos fazer pequenas funções.

```
produto = lambda x,y: x*y  
result = produto(6,4)  
print(result)
```

# Controle de Fluxo

O controle de fluxo de dados no Python podem ser realizados utilizando desvio no fluxo de código, ou através de sistema de repetição.

- if
- for
- while

# Condição **IF**

No desvio de fluxo, utilizamos o comando **IF**. A sintaxe é a seguinte:

```
if (expressão):  
    pass  
elif (expressão):  
    pass  
else:  
    pass
```

# Condição **IF** (Exemplo)

Vamos fazer uma entrada de dados via **input**, transformar em inteiro e realizar as comparações.

```
x = int(input("Please enter an integer: "))

if x < 0:
    print('Valor negativo')
elif x == 0:
    print('Zero')
elif x > 0 and x < 10:
    print('Maior que Zero e menor que 10')
else:
    print('Maior que 10')
```



# Repetição **For**

Além do desvio de fluxo, podemos repetir um pedaço de código utilizando o **FOR**.

```
words = ['gato', 'cachorro', 'coelho']  
for w in words:  
    print(w, len(w))
```

“ Caso não tenha uma lista de `coisas`, uma nova lista de números pode ser criada utilizando a função `range`. Por exemplo, se você precise gerar uma lista de números de 0 até 5 `range(5)` ”

# Gerador de Lista de Números

```
for i in range(5):  
    print(i)
```

A sintaxe do `range` é a seguinte:

```
range(inicio, fim, pulo)
```

# Mais exemplo de Range

Exemplos:

```
range(5, 10)  
5, 6, 7, 8, 9
```

```
range(0, 10, 3)  
0, 3, 6, 9
```

```
range(-10, -100, -30)  
-10, -40, -70
```

# Repetição `while`

Como forma de loop também temos o `while`. Com o `while` não possui uma lista, portanto não é possível saber a quantidade de vezes que o fluxo do programa vai ficar no `while`.

O fluxo vai ficar no `while` até a condição for `False`.

Sintáxe:

```
while condição:  
    pass  
else:  
    pass
```

# Repetição **while** (Exemplo)

Vamos repetir um bloco de código enquanto o valor de **x** seja menor que **10**.

```
x= 0

while x < 10:
    print('Numero', x)
    x += 1

else:
    print('Finalizado')
```

# Repetição **while** (Exemplo)

Em muitos casos isso não é possível determinar a quantidade de vezes que o **While** vai ser executado.

```
x = 'inicio'
lista = []
while x.lower() != 'fim':
    x = input("Digite o próximo nome:")
    lista.append(x)

print(lista)
```

## `break`, `continue` e `else`

- A instrução `break`, como em C, é interna para as funções `for` ou `while` e quando invocado finaliza de forma bruta o loop.
- A instrução `continue` finaliza aquela parte do bloco de código e volta para o início do loop.
- As instruções de loop podem ter uma cláusula `else`; ele é executado quando o loop termina por esgotamento da lista (`for`) ou quando a condição se torna `falsa` (`while`), mas não quando o loop é finalizado por uma instrução `break`.



# break, continue e else (Exemplo)

```
nomes=['maria','jose','carlos','eduardo']

for nome in nomes:
    if nome == 'jose':
        continue
    elif nome == 'eduardo':
        break
    print(nome)

else:
    print("Finalizado naturalmente")
```

# Outras funções

- print

O `print` como já vimos em vários exemplos, é utilizado para imprimir (mostrar) algo na tela.

- split

A função `split` utiliza um padrão para 'quebrar' uma string e gerar uma lista.

# Exemplos de print()

Vamos aprender com esses exemplos:

```
# declarar variavel  
nome, idade, salario = "jose", 30, 100.00
```

```
# imprimindo texto  
print("Inicio do programa")
```

```
# imprimindo apenas variável  
print(nome)
```

```
# Imprime string com variável  
print("Usuario:", nome)
```

# Continuando...

```
# Imprime string com variável
print("O usuario:" + nome)

# Usando o format
print("O usuario {0} tem {1:d} idade".format(nome, idade))

# Usando format com float
print("salario {0:0.2f} l".format(salario))

# alinha a direita com 20 espaços em branco
print("{0:>20}".format(nome))
```

# Continuando...

```
# alinha a direita com 20 símbolos -  
print("{0:->20}".format(nome))
```

```
# alinha ao centro usando 10 espaços em branco a esquerda e 10 a direita  
print("{0:^20}".format(nome))
```

```
# imprime só as primeiras cinco letras  
print("{0:.3}".format(nome))
```

```
#Imprime em hexadecimal  
print("{0:x}".format(23))
```

# Split (Exemplo)

Como exemplo, vamos utilizar as seguinte strings:

```
texto = "jose:30anos:rua de baixo:sp"
```

Temos um delimitador na variável que é o ":".

```
lista = texto.split(":") # ['jose', '30anos', 'rua de baixo', 'sp']  
print(lista[0])         # imprimir o nome
```

“ Qualquer caractere pode ser o delimitador.

”

**==That's all!==**

**Let's create beautiful slides  
with ==Marp== + ==Gaia== theme!**

“ In Greek mythology, **Gaia** also spelled **Gaea**, was the personification of the Earth and one of the Greek primordial deities.

<small>-- [Gaia \(mythology\) - Wikipedia, the free encyclopedia](#)</small>

”





`<!-- $theme: gaia -->` of Marp

<https://yhatt.github.io/marp>