

▼ Tutorial Básico da linguagem Python

Este tutorial apresenta uma introdução a alguns aspectos da linguagem de programação Python. As informações apresentadas aqui não cobrirão todos os detalhes desta linguagem. No entanto, livros inteiros se concentram em Python. O leitor é encorajado a consultar fontes adicionais na linguagem Python. Um tutorial completo pode ser consultado em: <https://docs.python.org/pt-br/3/tutorial/>.

▼ Introdução

A linguagem Python foi desenvolvida em 1991 pelo holandês Guido van Rossum. Python é uma linguagem de propósito geral, visualmente limpa e de sintaxe elegante, que não faz uso excessivo de marcações (ponto ou ponto e vírgula), marcadores (chaves, colchetes ou parênteses) e de palavras especiais (begin/end).

Considerada de fácil aprendizado, compreensão e leitura. É uma linguagem interpretada com tipagem dinâmica (não exige declarações de tipos de dados) que pode ser facilmente transformada para uma aplicação por meio da importação de bibliotecas. É uma linguagem de alto nível, orientada a objetos, mas que permite o usuário programar de forma procedural, se desejar;

Em Python a indentação é fundamental, pois o código não irá funcionar se não estiver devidamente indentado (artifício que indica quais estruturas estão subordinadas sem a necessidade de uso de chaves como acontece em outras linguagens);

Python é um software livre (usuários e colaboradores podem modificar seu código fonte e compartilhar essas novas atualizações, contribuindo para o constante aperfeiçoamento da linguagem).

▼ Versão do Python

```
!python --version
```

```
Python 3.10.12
```

▼ Comando de saída (impressão)

```
print("Hello World!")
```

```
Hello World!
```

▼ Comentários

```
# Comentário de uma única linha (não tem efeito no programa)
print("Hello World!") # Diga alô mundo
```

```
Hello World!
```

▼ Variáveis

Uma variável não pode ser utilizada em uma expressão sem ter sido inicializada. A tipagem é dinâmica.

```
preco = 1000
desconto = 0.05
preco_final = preco - desconto * preco
print(preco_final)
print(type(preco_final))
```

```
950.0
<class 'float'>
```

Também podemos inicializar mais de uma variável:

```
a, b = 10, 20
print('a =', a, 'e b =', b)
```

```
a = 10 e b = 20
```

Podemos iniciar as variáveis com o valor None.

```
x = None
print(x)
print(type(x))
```

```
None
<class 'NoneType'>
```

▼ Operadores aritméticos

```
x = 10
print(type(x))
print(x + 1)  # Adição
print(x - 1)  # Subtração
print(x * 2)  # Multiplicação
print(x ** 2) # Exponenciação
```

```
<class 'int'>
11
9
20
100
```

```
x += 1
print(x)
x *= 2
print(x)
```

```
11
22
```

▼ Booleanos (lógicos)

Python implementa todos os operadores usuais para lógica booleana, mas usa palavras em inglês em vez de símbolos (&&, ||, etc.):

```
t, f = True, False
print(type(t))
```

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

Algumas operações lógicas:

```
print(t and f)
print(t or f)
print(not t)
print(t != f)
```

```
False
True
False
True
```

▼ Strings

```
hello = 'hello'  # Strings podem usar aspas simples
world = "world"  # ou duplas
print(hello, len(hello))
```

```
hello 5
```

```
hw = hello + ' ' + world  # concatenação
print(hw)
```

```
hello world
```

Alguns métodos aplicados em strings:

```
s = "hello"
print(s.capitalize()) # Capitalize
print(s.upper())      # Caixa alta
print(s.lower())      # Caixa baixa
print(s.rjust(7))     # Justifica à direita preenchendo com espaços
print(s.center(7))    # Centraliza preenchendo com espaços
print(s.replace('l', '(ell)')) # Substitui todas as instâncias de uma substring com outra
```

```
Hello
HELLO
hello
  hello
    hello
he(ell)(ell)o
```

Acessando pelo índice:

```
str = 'pyhton'
print(str[0])
print(str[1])
```

```
p
y
```

▼ Cast - conversão de tipos

```
x = '123'
print(x)
print(type(x))
x = float(x)
print(x)
print(type(x))
print(int(x))
```

```
123
<class 'str'>
123.0
<class 'float'>
123
```

▼ Print de texto com variáveis

```
x = 5
y = 2.5
z = 'oi'
print('x é igual a %d, y é igual a %.1f e z é igual a \"%s\"' % (x,y,z))
print('x é igual a {}, y é igual a {} e z é igual a \"{ }\"'.format(x,y,z)) # alternativa (existem outras)
```

```
x é igual a 5, y é igual a 2.5 e z é igual a "oi"
x é igual a 5, y é igual a 2.5 e z é igual a "oi"
```


▼ Leitura de dados (entradas)

```
]x = int(input("Entre com um valor inteiro: "))
print(x)
y = float(input("Entre com um valor real: "))
print(y)
s = input("Digite um texto: ") # os dados de entrada serão strings por padrão
print(s)
print(type(s))
```

```
Entre com um valor inteiro: 7
7
Entre com um valor real: 3.14
3.14
Digite um texto: olá mundo
olá mundo
<class 'str'>
```

▼ Estruturas de controle de seleção (if-else)

```
expressao = 0
resp = -2
if expressao == True:
    print("true")
else:
    print("false")
if resp > 5:
    print(resp)
    print("positivo") #observe que o bloco é definido pela indentação
elif resp < 5:
    print(resp)
    print("negativo") #observe que o bloco é definido pela indentação
else:
    print(resp)
    print("nulo") #observe que o bloco é definido pela indentação
```




```
false
-2
negativo
```

Estruturas de controle de repetição

▼ For


```
soma = 0
N = 5
for contador in range(5): # A função range() retorna uma série numérica no intervalo enviado como argumento
    x = int(input("Entre com um valor inteiro: "))
    soma = soma + x
# imprima o resultado
print ("O valor da soma dos", N, "numeros é", soma)
```



```
Entre com um valor inteiro: 4
Entre com um valor inteiro: 2
Entre com um valor inteiro: 6
Entre com um valor inteiro: 3
Entre com um valor inteiro: 1
O valor da soma dos 5 numeros é 16
```

▼ While

```
soma = 0
contador = 1
N = 5
# repete N vezes o trecho de programa
while contador <= N:
    x = int(input("Entre com um valor inteiro: "))
    soma = soma + x
    contador = contador + 1
# imprima o resultado
print ("O valor da soma dos", N, "numeros é", soma)
```



```
Entre com um valor inteiro: 8
Entre com um valor inteiro: 2
Entre com um valor inteiro: 10
Entre com um valor inteiro: 4
Entre com um valor inteiro: 3
O valor da soma dos 5 numeros é 27
```

▼ Containers

Python inclui diversos tipos de "containers": listas, dicionários, conjuntos e tuplas.

▼ Listas

Uma lista é equivalente a um vetor (array), mas seu tamanho pode ser modificado e pode conter elementos de diferentes tipos.

```
list = [3, 1, 'Márcio'] # Cria uma lista
print(list)
print(list[0])
print(list[-1]) # Índices negativos contam a partir do final da lista
print(type(list))
```

```
[3, 1, 'Márcio']
3
Márcio
<class 'list'>
```

Adicionando um elemento no final da lista:

```
list.append('Leandro')
print(list)
```

```
[3, 1, 'Márcio', 'Leandro']
```

Removendo o último elemento:

```
x = list.pop() # Remove and return the last element of the list
print(list)
```

```
[3, 1, 'Márcio']
```

▼ Slicing

```
nums = [0, 1, 2, 3, 4] # cria uma lista
print(nums) # Imprime "[0, 1, 2, 3, 4]"
print(nums[2:4]) # Pega um slice do índice 2 a 4 (exclusive)
print(nums[2:]) # Pega um slice do índice 2 até o fim
print(nums[:2]) # Pega um slice do começo até o índice 2 (exclusive)
print(nums[:]) # Pega a lista toda"
print(nums[:-1]) # Índices podem ser negativos (exclue o último elemento)
print(nums[:-1]) # Índices podem ser negativos (exclue os dois últimos)
nums[2:4] = [8, 9] # Atribui a nova sublista
print(nums) # Prints "[0, 1, 8, 9, 4]"
```

```
[0, 1, 2, 3, 4]
[2, 3]
[2, 3, 4]
[0, 1]
[0, 1, 2, 3, 4]
[0, 1, 2, 3]
[0, 1, 2, 3]
[0, 1, 8, 9, 4]
```

▼ Loops com uma lista

```
animais = ['cão', 'gato', 'macaco']
for animal in animais:
    print(animal)
```

```
cão
gato
macaco
```

▼ Dicionários

Um dicionário armazena pares (chave, valor):

```
dic = {'gato': 'felino', 'cão': 'amigo'} # cria um dicionário
print(dic['gato']) # Pega uma entrada a partir da chave
print('cão' in dic) # Verifica se o dicionário tem uma determinada chave
print(type(dic))
```

```
felino
True
<class 'dict'>
```

```
dic['peixe'] = 'molhado' # Determina uma nova entrada no dicionário
print(dic['peixe'])
print(dic)
```

```
molhado
{'gato': 'felino', 'cão': 'amigo', 'peixe': 'molhado'}
```

```
del dic['peixe'] # Remove um elemento do dicionário
print(dic)
```

```
{'gato': 'felino', 'cão': 'amigo'}
```

✓ Sets (conjuntos)

Um conjunto (set) é uma coleção não indexada de elementos distintos.

```
animais = {'cão', 'gato'}
print('gato' in animais) # verifica se um elemento está no conjunto
print('peixe' in animais)
```

```
True
False
```

```
animais.add('peixe') # Adiciona um elemento
print(animais)
print(len(animais))
```

```
{'cão', 'gato', 'peixe'}
3
```

```
animais.add('gato') # Não adiciona um elemento que já existe no conjunto (não faz nada)
print(len(animais))
animais.remove('gato') # Remove um elemento
print(animais)
```

```
3
{'cão', 'peixe'}
```

✓ Tuplas

Uma tupla é uma lista ordenada (imutável) de valores. Uma tupla é em muitos aspectos semelhante a uma lista; uma das diferenças mais importantes é que tuplas podem ser usadas como chaves em dicionários e como elementos de conjuntos, enquanto listas não.

```
d = {(x, x + 1): x for x in range(10)} # Cria um dicionário com chaves tuplas
print(d)
t = (5, 6) # Cria uma tupla
print(type(t))
print(d[t])
print(d[(1, 2)])
```

```
{(0, 1): 0, (1, 2): 1, (2, 3): 2, (3, 4): 3, (4, 5): 4, (5, 6): 5, (6, 7): 6, (7, 8): 7, (8, 9): 8, (9, 10): 9}
<class 'tuple'>
5
1
```

✓ Funções

Funções em Python são definidas usando a palavra chave `def`.

```
def sinal(x):
    if x > 0:
        return 'positivo'
    elif x < 0:
        return 'negativo'
    else:
        return 'zero'
```

```
for x in [-1, 0, 1]:
    print(sinal(x))
```

```
negativo
zero
```

positivo

▼ Classes

A sintaxe para definir classes em Python é direta:

```
class Triangulo(object):

    def __init__(self, angulo1, angulo2, angulo3):
        self.angulo1 = angulo1
        self.angulo2 = angulo2
        self.angulo3 = angulo3

    def soma_angulos(self):
        return self.angulo1 + self.angulo2 + self.angulo3

    def check_angulos(self):
        if(self.soma_angulos() == 180):
            return True
        else:
            return False

figure = Triangulo(30, 40, 50)
print(figure.soma_angulos())
print(figure.check_angulos())
```

```
120
False
```

▼ Imports

Para importar um módulo utilizamos o import.

```
import math
print(math.sqrt(49))
```

```
7.0
```

O código acima importará todos os módulos de math, para importar apenas o necessário utilizamos from.

```
from math import sqrt
print(math.sqrt(49))
```

```
7.0
```