

---

# Programação em Python

Prof. Fernando Nakagawa

---

---

# O que é Lógica de Programação?

É o estudo de sequências de ações ou comandos para atingir um objetivo.

Essas sequências de comandos se chama de ALGORITMOS .

---

---

---

# Algoritmos

É como se escrevem programas de computador de maneira prática. Essas sequência de comandos tem uma ordem e existe um fim.

---

---

# O que é Python?

É uma linguagem que foi criada em 1991 por Guido van Rossum para ser produtiva e legível.

Algumas características dessa linguagem:

- uso de indentação para determinar blocos
  - pouco uso de caracteres especiais
  - alto nível
  - interpretada
  - multiplataforma
  - multi-paradigma
-

---

# Para quê serve?

- Desenvolvimento de sistemas Web
  - Análise de dados, Inteligência Artificial, Machine Learning
  - Aplicativos
  - Sistemas Desktop
-

---

---

# Onde baixar o Python ?

<https://www.python.org/downloads/>

---

---

# Olá Mundo em Python

```
print("Ola Mundo!")
```

---

---

# Comentários

# Comentário de uma linha

"""

Bloco de comentários  
que se expandem por várias

linhas

"""

---



---

---

# Declaração de variáveis

No Python, qualquer variável inicia com a sua própria declaração e uma atribuição

A variável não pode iniciar com números e são  
CASE-SENSITIVE!

Variáveis podem ser declaradas em qualquer lugar do código!

---

---

# Case Sensitive

As palavras reservadas são case sensitive.

Os comandos devem ser escritos exatamente da mesma forma.

Declarar uma variável com um nome igual, porém diferenciando a caixa, é a mesma coisa que declarar variáveis diferentes:

```
j = 5
```

```
J = 6
```

```
print(j,J)
```

```
saída: 5 6
```

---

---

# Declaração de variáveis

```
i = 10
```

```
a = "Olá"
```

```
batata = True
```

```
temp1 = 3.1415
```

---

---

---

# print

Utilize print para mostrar conteúdo na tela.

```
print("Bom dia")
```

saída: Bom dia

---

---

# Tipos de dados

No Python as variáveis podem ser dos tipos:

- Inteiro (int)
    - `x = 1;`
  - Texto (str)
    - `txt = "Olá";`
  - Números Reais (float)
    - `y = 1.56;`
  - Booleano (Boolean)
    - `a = True;`
  - Números Complexos (complex)
    - `comp = 1j`
-

---

---

# type

Para verificar o tipo de uma variável utilize o type.

Para mostrar na tela utilize em conjunto com o print

```
print(type(x))
```

---

---

# input

Para realizar entrada de dados manual, utilize a função input.

o input retorna a string que o usuário digitar. Caso necessário converta a string para outro tipo

```
texto = input("Digite o seu nome:")
```

```
num = int(input("Digite um número:"))
```

---

---

# Números aleatórios

Para sortear um número importe a biblioteca random

```
import random
```

```
print(random.randrange(1,10))
```

---



---

# Trabalhando com textos

## Como pegar um caracter em uma posição

```
texto = "Bom dia"
```

```
print(texto[2])
```

---

---

# Trabalhando com textos

## Como pegar um pedaço de um texto

```
texto = "Bom dia"
```

```
print(texto[4:6])
```

```
print(texto[-5:-2])
```

---

---

# Trabalhando com textos

## Como pegar o tamanho do texto

```
texto = "Bom dia"
```

```
print(len(texto))
```

---

---

# Trabalhando com textos

## Como eliminar os espaços do início e do fim

```
texto = " Bom dia "  
  
print(texto.strip())
```

---

---

# Trabalhando com textos

## Como mostrar em caixa baixa e alta

```
texto = "Bom dia"  
  
print(texto.lower())  
  
print(texto.upper())
```

---

---

# Trabalhando com textos

## Como separar o texto em substrings através de um separador

```
texto = "Arroz,feijão,batata,macarrão"
```

```
print(texto.split(","))
```

Sendo o caractere , o separador.

---

---

# Trabalhando com textos

## Como verificar se um trecho está presente em uma variável

```
texto = "Bom dia"  
  
x = "dia" in texto  
  
print(x)
```

---

---

# Trabalhando com textos

## Como juntar strings e outras variáveis

```
texto1 = "Bom"  
texto2 = "dia"  
x = "dia" in txt  
print(x)
```

---



---

# Operadores Aritiméticos

- + (adição)
  - - (subtração)
  - \* (multiplicação)
  - / (divisão)
  - % (módulo - resto de uma divisão)
  - \*\* (exponenciação)
  - // (divisão arredondando para baixo)
-

---

# Operadores de atribuição

- = (é o que mais vamos utilizar!)
  - += (x += y é equivalente à x = x+y)
  - -=
  - \*=
  - /=
  - %=
-

---

# Operadores de Comparação

- ==
    - em  $x == y$ , se forem iguais, é verdade, senão é falso
  - !=
    - Diferente
-

---

# Operadores de Comparação

- >
    - em  $x > y$ , se  $x$  for maior que  $y$ , retorna verdade.
  - <
    - em  $x < y$ , se  $x$  for menor que  $y$ , retorna verdade.
  - >=
    - maior OU igual
  - <=
    - menor OU igual
-

---

# Operadores Lógicos

- **and**
    - em `x and y` é True se as duas condições forem True.
  - **or**
    - em `x or y` é True se uma das condições é True
  - **not**
    - inverte o valor booleano. Se era True vira False, se era False, vira True
-

---

# Operadores de identidade

- **is**
    - True, se x is y forem o mesmo objeto
  - **is not**
    - True, se x e y não forem o mesmo objeto
-

---

---

# Expressões Condicionais

Quando escrevemos códigos, sempre precisaremos que ações diferentes ocorram em situações diferentes. É possível utilizar expressões condicionais para atingir tal objetivo.

---

---

# if, elif, else

- expressão if (se)
    - Executa um código apenas se uma condição for verdadeira.
  - if... else (se... senão)
    - Executa um código se uma condição for verdadeira e outro código se a condição for falsa
  - if... elif... else (se... senão, se... senão)
    - elseif especifica uma nova condição se a primeira for falsa.
-



---

# if, elif, else

Exemplo 1:

```
a = 5
if a == 5:
    print("bom dia!")
```

#é necessário ter o espaço antes do print, pois o Python determina o escopo do if através de indentação!

Exemplo 2:

```
a = 6
if a == 5:
    print("bom dia!")
else:
    print("erro")
```

Exemplo 3:

```
a = 6
if a == 5:
    print("bom dia!")
elif a == 6:
    print("boa tarde!")
else:
    print("erro")
```

---

# if, elif, else

## Exemplo 4:

```
a = -1
if a > 0:
    print("numero positivo!")
```

## Exemplo 5:

```
a = -1
if a > 0:
    print("num positivo")
else:
    print("num negativo ou 0")
```

## Exemplo 6:

```
a = -1
if a > 0:
    print("num positivo")
elif a < 0:
    print("num negativo")
else:
    print("numero zero")
```

---

# if, else, elseif

## Exemplo 7:

```
a = -1
if a != 0:
    print("numero diferente de 0!")
```

## Exemplo 8:

```
a = -1
if a != 0:
    print("num diferente de 0")
else:
    print("num igual a 0")
```

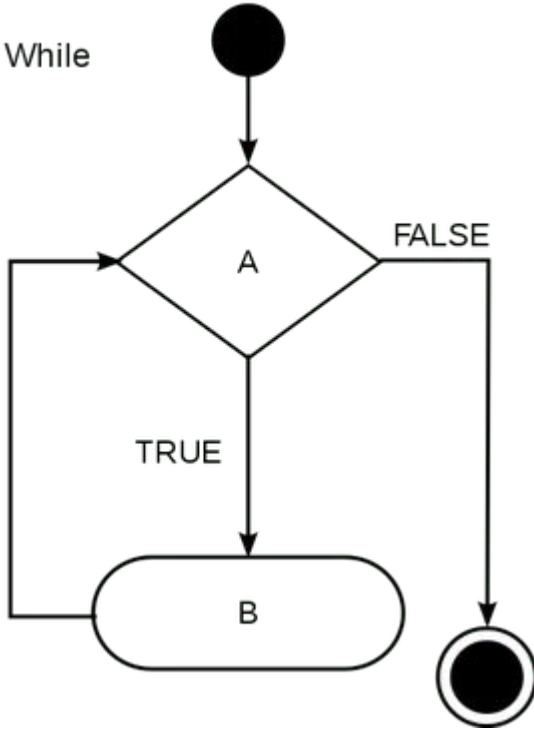
## Exemplo 9:

```
a = 12
if a < 12:
    print("bom dia!")
elif a != 13:
    print("hora de estudar!")
else:
    print("outros horarios")
```

---

# laço while

While (A = TRUE) Do  
    B  
End While



O laço while executa um bloco de código enquanto a condição especificada for verdadeira.

Estrutura:

while condição :

    código a ser executado

---

---

### Exemplo 1:

```
x = 0
while x <= 4:
    print("Olá")
    x += 1
```

#Bom dia será impresso 5 vezes. O laço while continuará a rodar enquanto x for menor ou igual a 4

### Exemplo 2:

```
x = 0
while x < 4:
    print("Olá")
    x += 1
```

#Bom dia será impresso 4 vezes. O laço while continuará a rodar enquanto x for menor que 4

### Exemplo 3:

```
x = 0
while x >= 0:
    print("Olá")
    x += 1
```

#CUIDADO com os loops infinitos!

---

# Laço for

O laço for é utilizado quando é sabido quantas vezes o laço irá executar ou se tem uma lista ou uma string.

Estrutura:

```
for <variável> in <teste>:
```

```
    código a ser executado
```

---

---

Exemplo 1:

```
for x in range(100):  
    print(x)
```

#Serão impressos os números  
de 0 à 99.

?>

Exemplo 2:

```
for batata in range(2,6):  
    print(batata)
```

#serão impressos os números de 2  
à 5

Exemplo 3:

```
for aaa in range(2,30,3):  
    print(aaa)
```

#Serão impressos os  
números: 2, 5, 8, 11, 14,  
17, 20, 23, 26 e 29

---

---

# Utilizando operadores lógicos nas condições

É possível utilizar operadores lógicos nas condições para unir condições que deve-se saber apenas seu resultado final.

---



---

Exemplo 1:

```
a = 5
```

```
if a == 5 and a==6:  
    print ("Bom dia!")
```

```
//"Bom dia nunca aparecerá  
na tela! Porque?"  
?>
```

Exemplo 2:

```
a = 20
```

```
if a == 10 or a==20:  
    print ("Bom dia!")
```

Exemplo 3:

```
a = 1
```

```
b = 2
```

```
if a != 3 and b != 4:  
    print ("Bom dia!")  
else:  
    print ("Boa noite!")
```

```
/*a primeira condição é  
verdadeira e a segunda  
também, Logo aparecerá Bom  
dia na tela.*/  
?>
```

---

Exemplo 4:

```
a = 1
b = 2
if a != 1 or b != 4:
    print("Bom dia!")
else:
    print( "Boa noite!")
```

#a primeira condição é falsa e a segunda verdadeira, Logo aparecerá Bom dia na tela.

Exemplo 5:

```
a = 0
b = 0
while a < 3 and b < 2:
    print("Bom dia!")
    a += 1
    a >= 2:
        a = 0
        b+=1
#Bom dia será impresso 4 vezes na tela.
```

Exemplo 6:

```
a = 0
b = 0
while a < 3 or b < 2:
    print ("Bom dia!")
    a+=1
    b+=1

#Bom dia será impresso 3 vezes na tela.
```

---

# Listas (vetores)

Um vetor pode guardar vários valores dentro de uma única variável. É uma variável especial. O Python possui 4 tipos diferentes que podem guardar mais de um valor em uma mesma variável (Lists, Tuples, Sets e Dictionaries). Além do mais, é possível utilizar outros tipos diferentes através da importação de bibliotecas.

---

---

# Lists (vetores)

Mas, por quê a utilização de listas?

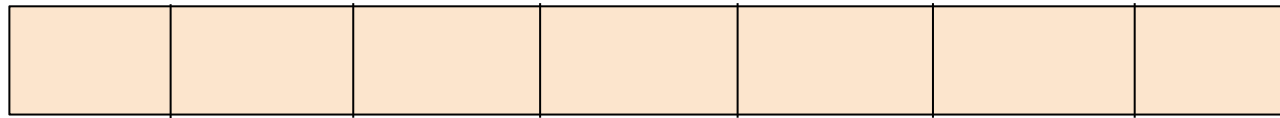
Imagine que você tenha 3 celulares:

celular1 = "Nokia"

celular2 = "Apple"

celular3 = "Xiaomi"

Você pode criar 3 variáveis. Mas e se você tiver 300?



---

# Lists (vetores)

Uma lista pode guardar vários valores dentro de uma única variável. Ela é:

- dinâmica: pode receber e remover itens;
- heterogênea: recebe valores de tipos diferentes (ex.: ints com strings na mesma lista);
- permite duplicatas;
- possui ordem;
- Sintaxe com [ ].

```
lista = ["item1", "item2", "item3", 83, "item5", True, "item7"]
```

```
print (lista)
```

	lista						
	"item1"	"item2"	"item3"	83	"item5"	True	"item7"
índice	0	1	2	3	4	5	6

---

# Lists

Para pegar um valor de uma lista você pode utilizar os índices.  
Considere a primeira lista do capítulo:

```
lista = ["item1", "item2", "item3", "item4",  
"item5", "item6", "item7"]  
x = lista[0]
```

#x vai ser igual à "item1"

---

---

# Lists

Para modificar um valor, é só utilizar o índice também

```
lista = ["item1", "item2", "item3", "item4",  
"item5", "item6", "item7"]  
lista[0] = "batata"  
  
print(lista)
```

---

---

---

# Listas - len

Para obter o valor inteiro do tamanho da lista:

```
x = len(lista)
```

---



---

# Listas - type

Para obter o tipo da lista:

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

```
print( type(x) )
```

```
#list
```

---

---

# Listas - min e max

Para obter o menor e maior valor da lista:

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

```
print( max(x) )
```

```
#3
```

---

---

## Listas - loop

É possível utilizar o for in para criar um loop para percorrer todos os elementos da lista:

```
x = [4,5,6,7]
```

```
for num in x:
```

```
    print(num*2)
```

O Python oferece suporte para mostrar a lista inteira através de um print.

```
print(x)
```

---

---

# Listas - append

É possível adicionar elementos na lista com `append()`

```
x = [4,5,6,7]
```

```
x.append(8)
```

---

---

# Listas - extend

É possível adicionar vários elemento de uma vez com extend()

```
x = [4,5,6,7]
```

```
x.extend([8,9,10])
```

---

---

# Listas - clear

Limpa a lista toda

```
x = [4,5,6,7]
```

```
x.clear()
```

---

---

## Listas - remove

É possível remover elementos na lista com `remove()`. Insira o conteúdo que se deseja remover. Ele apenas remove o primeiro elemento encontrado.

```
x = ["batata", "melao", "cenoura", "batata"]
```

```
x.remove("batata")
```

#elimina somente o primeiro "batata"

Pense: como remover o primeiro elemento da lista independente de seu conteúdo?

---

---

# Listas - pop

É possível remover elementos na lista com pop(). Insira o valor do índice que se deseja remover

```
x = [4,5,6,7]
```

```
x.pop(2)
```

#elimina o terceiro elemento da lista

```
x.pop()
```

#elimina o último elemento da lista

---



---

# Tuplas

Tuplas são coleções de itens que são imutáveis. Permitem duplicatas e possuem ordem e são representadas por ( ).

```
meses_ano = ("Janeiro", "Fevereiro", "Março")
```

Se você tentar:

```
meses_ano[1] = "Abril"
```

Terá o seguinte erro:

```
TypeError: 'tuple' object does not support item assignment
```

---

---

# Sets

Sets são coleções de dados de não ordenados que não permitem duplicatas e nem mudanças em seus elementos.

Pode ser inicializado com

```
x = set()
```

Utilize `add()` para adicionar um item e `remove` para remover.

ou

```
x = {1, 2, 3}
```

---

---

# Sets

Funções disponíveis:

- `add()`
  - `update()`: adicionar vários elementos no set com []
  - `remove()` ou `discard()`
  - `clear()`
  - `union()`: une dois sets
  - `copy()`
  - `difference()`: retorna a diferença entre dois sets
  - `difference_update()`
  - `intersection()`
  - `intersection_update()`
-

---

---

# Sets

- `isdisjoint()`
  - `issubset()`
  - `issuperset()`
  - `symmetric_difference()`
-

---

# Dictionaries

São coleções de itens que não permitem duplicatas (da mesma chave), mas são mutáveis e possuem ordem (a partir do Python 3.7). Também são representados através de { }.

A maior diferença entre as lists, tuples e sets é a utilização da estrutura chave: conteúdo.

Mesmo utilizando { } igualmente nos sets, os dictionaries se diferenciam pela necessidade de haver uma chave e conteúdo em cada registro.

---

---

# Dictionaries

Exemplo:

```
computador = {  
    "marca": "Dell",  
    "processador": "Core i7",  
    "memoria_gb": 32,  
    "perifericos": ["monitor", "teclado", "mouse"]  
}
```

---

---

---

# Funções

Funções são blocos de código que podemos utilizar quando quisermos, de forma repetida, evitando escrever repetidamente os mesmos códigos em determinados tempos diferentes.

---

---

# Funções

**FIQUE ATENTO!**

- Ela só é executada se for CHAMADA!
- Seu nome não pode começar com números (igualmente nas variáveis)

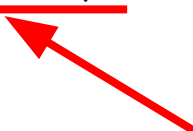
**Estrutura:**

```
def <nome>(<parâmetros>):  
    códigos
```



### Exemplo 1:

```
def dizerOi():  
    print("oi")
```



```
dizerOi()
```

#chamada da função. Aqui o bloco de código é executado.

```
?>
```

### Exemplo 2:

```
<?php  
def dizerOi():  
    print("oi")  
  
/#"oi" aparecerá na tela? Por  
quê?  
?>
```

### Exemplo 3:

```
def escreverNumerosPares():  
    for x in range(100):  
        if x%2 == 0:  
            print(x)  
  
escreverNumerosPares()
```

---

# Parâmetros

Para melhorar ainda mais as funcionalidades das funções, podemos passar valores por parâmetros.

Os parâmetros são especificados dentro dos parênteses e podem ser mais de um, separados por vírgula.

---

Exemplo 1:

```
def digaAlgumaCoisa(coisa):  
    print(coisa)
```

```
digaAlgumaCoisa("Bom dia, tudo bem?")  
digaAlgumaCoisa("Olá")  
digaAlgumaCoisa("Você poderia me ensinar  
algoritmos?")
```

Exemplo 2:

```
def calcularPrimo(num):  
    ehprimo = True  
    if(num % 2 == 0):  
        ehprimo = False  
    else:  
        for x in range(3,num//2):  
            if(num % x == 0):  
                ehprimo = False  
    if(ehprimo):  
        print("O número",num,"é primo!")  
    else:  
        print("O número",num,"não é primo!")  
  
calcularPrimo(123)  
calcularPrimo(1843)  
calcularPrimo(9873)
```

Exemplo 3:

```
def somar(n1, n2):  
    resp = n1 + n2  
    print(resp)
```

```
somar(5,7)
```

Exemplo 2:

```
def funcao(pais = "Brasil"):  
    print("Venho de um país chamado",pais)
```

```
funcao("Argentina")  
funcao()
```

#note que a atribuição no parâmetro define um valor padrão, caso não seja inserido nenhum valor

---

---

# Retorno de valores

Assim como podemos ter a entrada de valores dentro de uma função através de parâmetros, podemos ter a saída de valores, que é chamado de retorno.

---

Exemplo 1:

```
def somar(n1, n2):  
    resp = n1 + n2  
    return resp
```

```
x = somar(5,7)  
print(x)
```

Exemplo 1b:

```
def somar_e_multiplicar(n1, n2):  
    soma = n1 + n2  
    mult = n1 * n2  
    return soma, mult
```

```
x = somar_e_multiplicar(3,4)  
print(x)
```

Exemplo 2:

```
def calcularPrimo(num):  
    ehprimo = True  
    if(num % 2 == 0):  
        ehprimo = False  
    else:  
        for x in range(3,num//2):  
            if(num % x == 0):  
                ehprimo = False  
    return ehprimo
```

```
if(calcularPrimo(123)):  
    print(123, "é primo")
```

```
print(calcularPrimo(1843))
```

```
calcularPrimo(9873)
```

#esta última chamada perde o retorno, pois ninguém recebe seu retorno

Exemplo 3:

```
def bomDia():  
    return "Bom dia!"
```

```
print(bomDia())
```

Exemplo 4:

```
def numerosPares(inicio, fim):  
    lista = []  
    for x in range(inicio, fim):  
        if (x % 2 == 0):  
            lista.append(x)  
  
    return lista
```

```
x = numerosPares(3,31)  
print(x)
```