

# Programação em Python

Ponteiros e recursividade



Prof. Daniel Di Domenico

[https://github.com/danidomenico/gex003\\_algprog](https://github.com/danidomenico/gex003_algprog)

# Ponteiros

- Permitem acesso ao endereço de memória onde determinada variável está alocada:
  - Quando declara-se uma variável, ela é alocada na memória do computador;
  - Por vezes, pode ser necessário acessar qual é este endereço de memória;
  - O programador não escolhe em qual endereço de memória ela será alocada;
  - Através do endereço de memória, tem-se uma referência para a variável.

# Ponteiros em Python

- Python é uma linguagem orientada a objetos:
  - Por isso, **não possui ponteiros**, visto que as variáveis apontam para objetos e não para endereços de memória;
  - Desta forma, em Python é possível apenas simular o uso de ponteiros;
- Tipos de dados para simular o uso de ponteiros:
  -  – Tipos simples ou primitivos (**int, float, string**):
    - Não são acessados como referência, logo, **não podem** ser utilizados para simular ponteiros;
  -  – Tipos complexos: **listas (vetores e matrizes) e classes (estruturas)**:
    - São acessados como referência, logo, **podem** ser utilizados para simular ponteiros.

# Ponteiros em Python

- **Simulando ponteiros** (tipos primitivos): 

```
nome = "Jhon Travolta"  
copia = nome  
copia = "Ben Stiller"
```

#Alteração de "copia" não afetou a variável "nome"

```
print(nome)  
Jhon Travolta
```

```
print(copia)  
Ben Stiller
```

# Ponteiros em Python

- **Simulando ponteiros (tipos complexos):** 

```
class Pessoa:
    nome = ""

#Código principal...
pessoa = Pessoa()
pessoa.nome = "Jhon Travolta"
copia = pessoa # "copia" recebe a referência de pessoa
copia.nome = "Ben Stiller" # Alteração da variável "copia"
                           # afetou a variável "pessoa"

print(pessoa.nome)
Ben Stiller
print(copia.nome)
Ben Stiller
```

# Ponteiros em Python - Funções

- **Passagem por valor:** acontece quando é passado para uma função apenas o valor da variável:
  - Em Python, isso ocorre com os tipos simples ou primitivos (**int**, **float**, **string**);
  - Caso ocorra alguma alteração no valor do parâmetro dentro da função, esta alteração **não será refletida** na variável de origem.

# Ponteiros em Python - Funções

- **Passagem por valor:**

```
def troca(a, b): #Função troca
    aux = a
    a = b
    b = aux

#Código principal...
x = 10
y = 20
troca(x, y) #A troca não afetará em x e y

print(x, y)
10 20
```

# Ponteiros em Python - Funções

- **Passagem por referência:** acontece quando é passado para uma função a referência da variável:
  - Em Python, isso ocorre com os tipos complexos: **listas (vetores e matrizes) e classes (estruturas)**;
  - Caso ocorra alguma alteração no valor do parâmetro dentro da função, esta alteração **será refletida** na variável de origem;
  - **Referência em Python:** é o objeto para o qual uma variável está apontando. Por isso, alterações nessa variável refletem em todas as outras que também apontam para o mesmo objeto (ou seja, possuem a mesma referência).



# Ponteiros em Python - Funções

- **Passagem por referência (exemplo 1):**

```
class Ponto: #Classe Ponto
```

```
    x = 0
```

```
    y = 0
```

```
def troca(p): #Função troca
```

```
    aux = p.x
```

```
    p.x = p.y
```

```
    p.y = aux
```

```
#Código principal...
```

```
ponto = Ponto()
```

```
ponto.x = 10
```

```
ponto.y = 20
```

```
troca(ponto) #A troca dentro da função afetará "ponto"
```

```
print(ponto.x, ponto.y)
```

```
20 10
```

# Ponteiros em Python

- **Passagem por referência (exemplo 2):**

```
def incrementa_lista(lista):  
    for i in range(len(lista)):  
        lista[i] = lista[i] + 1
```

#Código principal...

```
vetor = [1, 2, 8, 10, 25]  
incrementa_lista(vetor)
```

```
print(vetor)  
[2, 3, 9, 11, 26]
```

# Recursividade

- Consiste na chamada de uma função no corpo (dentro) dela mesma:
  - Se uma função possui uma chamada para si mesma, ela é denominada **recursiva**;
  - Por que utiliza-se recursividade:
    - Para executar repetições;
    - Para resolver problemas que possuem diversas etapas iguais:
      - Percorrer uma árvore;
      - Imprimir uma lista encadeada.
    - Será muito utilizado durante o curso.

# Recursividade

- Ler e somar 2 números:

```
def leitura_soma(conta, soma): #Função recursiva
    if conta == 2: #Condição de parada da recursão
        return soma

    num = int(input("Informe um número: "))
    #Chamada recursiva
    return leitura_soma(conta+1, num+soma)

#Código principal...
soma_numeros = leitura_soma(0, 0)
print(soma_numeros)
```

# Recursividade

- Fatorial (versão com laço):

```
N = int(input("Número: "))  
aux = N  
fatorial = 1  
  
while N > 0:  
    fatorial = fatorial * N  
    N -= 1  
  
print("O fatorial de {0} é {1}".format(aux, fatorial))
```

# Recursividade

- Fatorial (versão com recursão):

```
def fatorial(n):  
    if n == 1: #Condição de parada da recursão  
        return 1  
  
    return n * fatorial(n-1)  
  
#Código principal...  
N = int(input("Número: "))  
fat = fatorial(N)  
  
print("O fatorial de {0} é {1}".format(N, fat))
```

# Exercícios

- 1) Faça um programa que declare uma estrutura Carro (modelo, cor, ano), criando uma variável deste tipo no código principal. O programa deve possuir uma função que receba um parâmetro do tipo Carro. Nesta função, os membros da estrutura Carro devem ser lidos (informados pelo usuário), sendo que a função não deve retornar nada, apenas alterar o parâmetro. Por fim, imprima os valores dos membros da variável carro no código principal.
- 2) Faça um programa que leia um número N, e calcule a soma de todos os seus antecessores até 1 utilizando uma função recursiva. Ex.: se for informado 6, o programa deve somar  $6+5+4+3+2+1$  e imprimir o resultado.