



Aula 8

COMPUTATIONAL THINKING USING PYTHON

AGENDA



1. Introdução a Funções
2. Conhecendo as funções importadas
3. Definição de funções em Python
4. Argumentos e parâmetros de funções
5. Retorno de funções
6. Escopo de variáveis
7. Documentação e organização de código

Nesta aula iremos abordar o conceito de **funções** e como podemos *defini-las* de acordo com a necessidade



Funções



Ao criarmos soluções para diversos problemas, é comum que certas sequências de código se tornem frequentes... podemos pensar na **reutilização** de código!

Principais vantagens do uso de funções

- Abstração ("o que" precisa ser feito e "como" será feito)
- Reusabilidade
- Modularização (problema inicial ser dividido em problemas menores)
- Manutenção do código
- Maior legibilidade
- Separação de escopo

Funções importadas

- Funções integradas podem ser utilizadas em qualquer trecho de código
- São reconhecidas automaticamente pelo interpretador
- O Python possui um conjunto de módulos extras denominados **biblioteca padrão** (<https://docs.python.org/pt-br/3/library/>)
- Os módulos são instalados junto com o interpretador, mas não são carregadas automaticamente ao executarmos o Python, ou seja, é preciso **importá-los** (comando `import`)

• Exemplo:



```
>>> import math
>>> math.pi
3.141592653589793
>>> math.log(10)
2.302585092994046
```

Funções definidas pelo programador

- Para criar uma nova função é preciso defini-las
- Para definir uma função, utiliza-se o comando `def`

```
def <nome da função> (<parâmetros>) :  
    <bloco de código>
```

- Após o nome da função, deve-se colocar um **par de parênteses** e, entre eles, os **parâmetros**, se existirem
 - Parâmetros são variáveis locais (internas da função)

Exemplo

```
def soma(n1, n2):  
    s = n1 + n2  
    return s
```

```
>>> soma  
  
<function soma at 0x0000017FC3BB7CA0>
```

Endereço da função `soma` na memória

Em Python, o nome da função é uma variável que faz referência ao seu código. A referência é feita por meio do endereço da função, ou seja, a variável guarda o endereço de memória onde está o código da função para ser executado.

Exemplo

```
>>> imprime = print  
>>> imprime('Programando em Português!')  
Programando em Português!
```

Atribuição da referência de uma função à outra variável

Criação da função

- Definição da função (comando `def`)
- Invocação da função (“chamar a função”)
- Toda função deve estar definida antes de ser chamada
- Após definida, para executá-la, basta chamá-la de modo idêntico às funções integradas e importadas

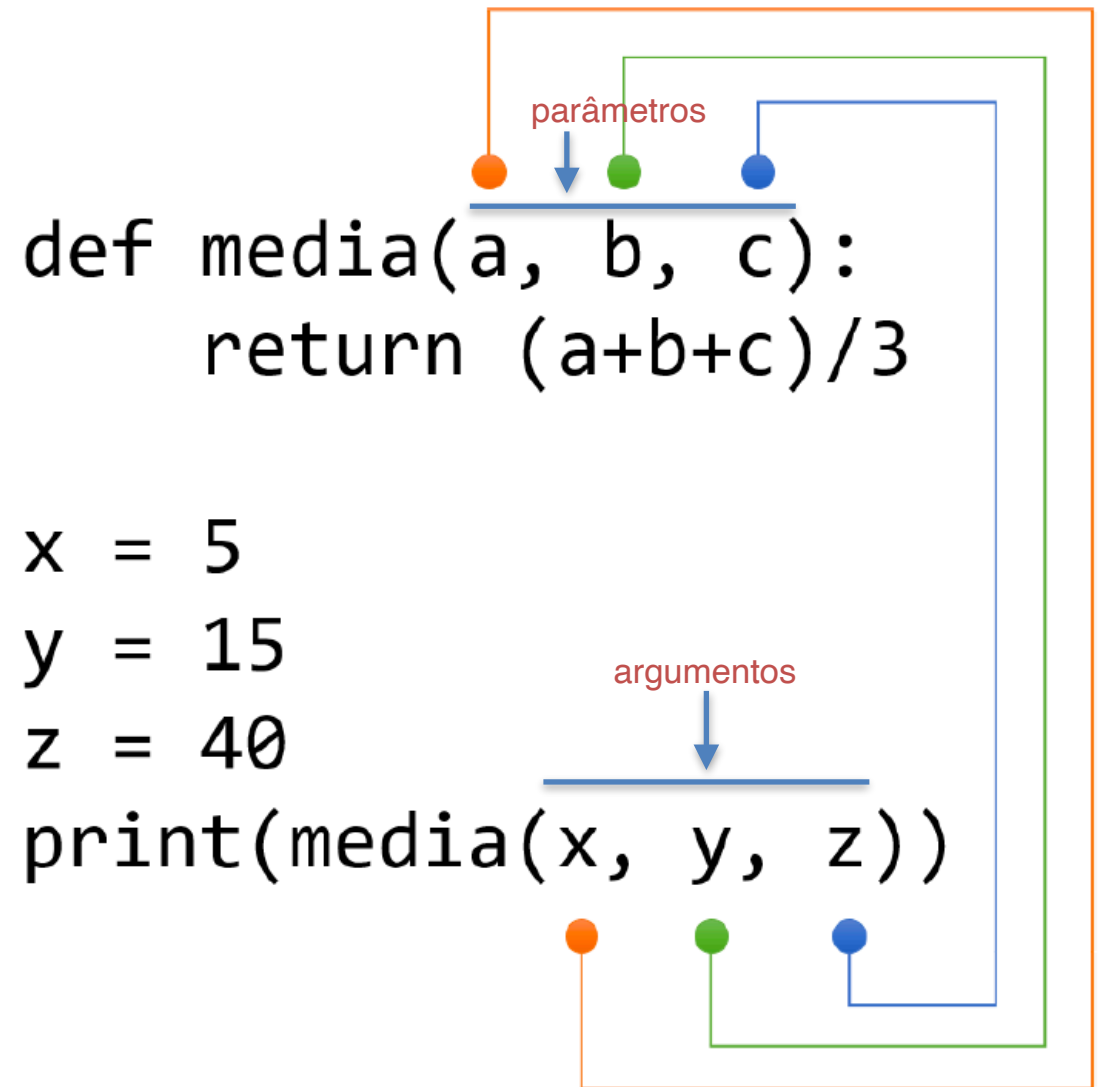
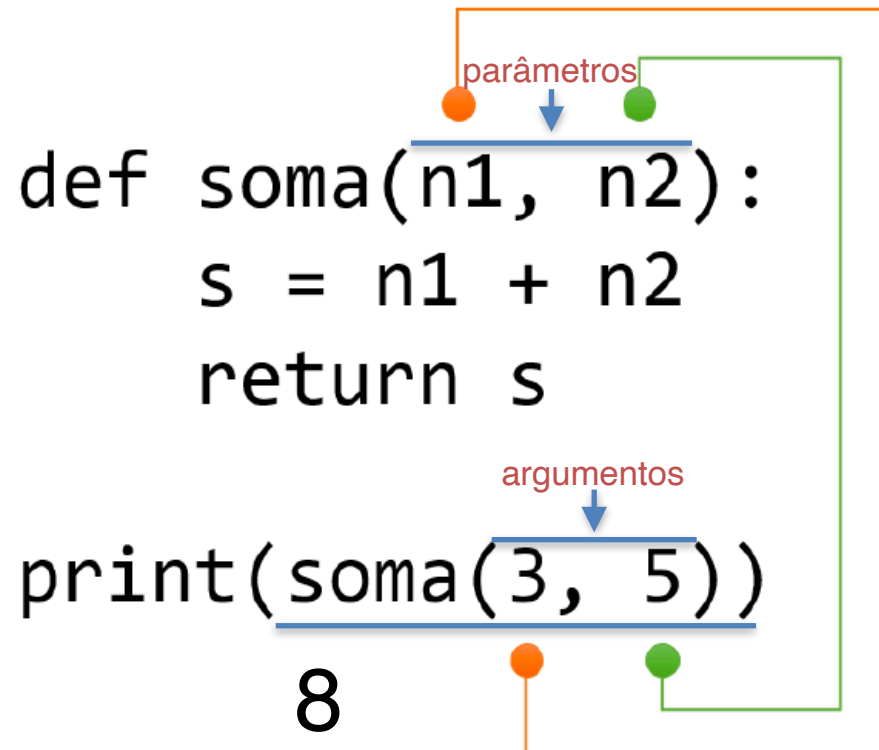
```
>>> soma (3, 7)
```

```
8
```

```
>>> soma (7, -9)
```


```
-2
```

Relação entre argumento e parâmetro



Exemplo

```
def soma(n1, n2):  
    print('Inicio do bloco da função')  
    s = n1 + n2  
    return s  
    print('Fim do bloco da função')  
print('Fora da função!')
```



```
>>> ...
```

```
Fora da função!
```

```
>>>
```

Exemplo

```
>>> soma(3, 5)
```

```
Início do bloco da função
```

```
8
```

```
>>> soma(7, -9)
```

```
Início do bloco da função
```

```
-2
```


Uma função pode ter mais de um return

- Pode ser útil caso estejam em blocos de códigos diferentes (estrutura de seleção, por exemplo)

```
def par(n) :  
    if n%2 == 0:  
        return True  
    else:  
        return False  
  
print(f'{4} é par? {par(4)}')  
print(f'{7} é par? {par(7)}')
```

Retorno de valores vs. Exibição de valores

Exemplo

```
def soma(n1, n2):  
    print('Inicio do bloco da função')  
    s = n1 + n2  
    return s
```

```
print('Fora da função!')
```

```
soma(3, 5)
```

```
soma(7, -9)
```



Escopo de variáveis



Escopo de variáveis

- Escopo local - variáveis criadas no interior de uma função ou bloco (acessíveis em todo o código-fonte onde foram criadas)
- Escopo global - variáveis criadas no raiz do código (código principal - acessíveis em todo o código fonte)

```
def diga_ola():  
    print(f'Olá {nome}') #acessa a variável global  
  
nome = 'seu nome' #variável global  
diga_ola
```

Escopo de variáveis

- Variáveis locais tem prioridade sobre as globais (caso existam duas variáveis com o mesmo identificador, a local será utilizada por padrão)

```
def diga_ola(nome):          #parâmetros são variáveis locais
    print(f'Olá {nome}!')    #acessa a variável local

nome = 'José' #variável global

diga_ola('Maria')

print(f'Tchau {nome}!')    #acessa a variável global
```


Escopo local vs. escopo global

Python 3.6
([known limitations](#))

```
→ 1 def diga_ola(nome):      # parâmetros são variáveis locais
  2     print(f'Olá {nome}!') # acessa a variável local
  3
  4 nome = 'Megan'           # variável global
→ 5 diga_ola('Maria')
  6 print(f'Tchau {nome}!')  # acessa a variável global
```

[Edit this code](#)

→ line that just executed
→ next line to execute

Step 4 of 7

[Customize visualization](#) (NEW!)

Print output (drag lower right corner to resize)

ESCOPO GLOBAL

Frames

Objects

Global frame

diga_ola

nome "Megan"

function diga_ola(nome)

ESCOPO LOCAL

diga_ola

nome "Maria"

- O uso de variáveis globais em funções é desencorajado, pois viola o encapsulamento das funções, dificulta a leitura do código e correções.



Texto de documentação de funções

Documentação de funções

- Ao definirmos novas funções, é comum acoplarmos comentários para explicar o funcionamento para quem as usará
- Também conhecidos como *docstrings*
- Se feito conforme padrão, muitos editores de código poderão identificar essa documentação e mostrá-la de forma conveniente

Exemplo

```
1 def soma(n1, n2):  
2     """Retorna a soma de dois numeros.  
3  
4     Parametros  
5     -----  
6     n1, n2: int ou float  
7     |  
8     |     Numeros a serem somados.  
9  
10    Retorno  
11    -----  
12    int ou float  
13    |  
14    |     Retorno da soma de n1 com n2.  
15    |  
16    """  
17  
18    s = n1 + n2  
19    return s
```

Resumo do que a função faz

Informações sobre a função

```
*IDLE Shell 3.9.1*
File Edit Shell Debug Options Window Help
>>> soma (
    (n1, n2)
    Retorna a soma de dois números.
Ln: 1 Col: 9
```

nome da função seguida da abertura do parênteses

```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
>>> help(soma)
Help on function soma in module __main__:

soma(n1, n2)
    Retorna a soma de dois números.

    Parâmetros
    -----
    n1, n2: int ou float
           Números a serem somados.

    Retorno
    -----
    int ou float
           Resultado da soma de n1 com n2.

>>>
Ln: 17 Col: 4
```

usando a função help



Organização do código

Organização do código

- Recomendações de organização para melhor legibilidade, rastreamento de erros e manutenção de códigos-fonte
 - As importações de bibliotecas devem ser feitas no início do código-fonte
 - Após as importações, se definem as funções
 - Por último, escreve-se o código principal que chamará as funções
 - É comum denominar o código principal de **programa principal**

```
1  # Importacoes
2  import math
3
4
5  # Definicao de funcoes
6  def dobro(x):
7      return 2 * x
8
9  def saudacao(nome):
10
11      print(f'Bem-vindo {nome}!')
12
13
14  #Codigo principal
15  nome = input('Qual o seu nome? ')
16  saudacao(nome)
17  dobro_pi = dobro(math.pi)
18  print(f'0 dobro de {math.pi:.3f} {dobro_pi:.3f}')
```

Exercício 1

1. Escreva um programa que leia um número do teclado, calcule o dobro do número lido e imprima o resultado. O programa deve ter:

- **Função** para entrada de dados
- **Função** para calcular o dobro do número
- **Função** para imprimir o resultado
- **Programa Principal** para testar as funções acima

Exercício 2

1. Escreva um programa que leia dois números do teclado (x e y), verifique qual é o maior e imprima a mensagem: "o número {x} é maior do que {y}. O programa deve ter:

- **Função** para entrada de dados
- **Função** para verificar o maior número lido
- **Função** para imprimir a mensagem
- **Programa Principal** para testar as funções acima

Obrigado e até a próxima aula!