

## ALGORITMIA E ESTRUTURAS DE DADOS

### MÓDULO III DECOMPOSIÇÃO MODULAR - FUNÇÕES

TECNOLOGIAS E SISTEMAS DE INFORMAÇÃO PARA A WEB

## 1. Decomposição Modular - Funções

- ❑ Conceito
- ❑ Criar uma Função
- ❑ Devolução de dados (*return*)
- ❑ Parâmetros com valor por defeito
- ❑ Número de parâmetros de entrada indefinido

```
primo.py > ...
1
2
3 def primo(numero): # função que dado um numero devolve True
4     primo = True
5     for i in range(numero-1, 1, -1):
6         resto = numero % i
7         if resto == 0:
8             primo = False
9             break
10    return primo
11
12
13
14    numero = int(input("Numero:"))
15    estado = primo(numero)
16    if estado == True:
17        print("O numero", numero, "é primo")
18    else:
```

## ❖ Funções | Conceito

- ❑ Conjunto de código (ou bloco de instruções), delimitado de forma clara, e que executa uma tarefa específica
- ❑ Uma função comporta-se da mesma forma que um programa, embora numa escala diferente:
  - ❑ É executada quando invocada pelo programa que o chama;
  - ❑ Quando termina, devolve a execução do programa, a partir do local onde foi chamada.
- ❑ O uso de funções permite desenvolver código de forma estruturada, facilitando:
  - ❑ Reutilização do código
  - ❑ Leitura / legibilidade do código
  - ❑ Abstração do código

## ❖ Funções | Conceito

- ❑ Uma função é um bloco reutilizável de instruções de programação projetado para executar uma determinada tarefa.
- ❑ Para definir uma função, o Python fornece a keyword **def**. A seguir está a sintaxe para definir uma função

The diagram illustrates the syntax of a Python function definition. It features a code block with the following structure:

```
def function_name(parameters):  
    "function docstring"  
    statement1  
    statement2  
    ...  
    ...  
    return [expr]
```

Labels with arrows point to specific parts of the code:

- Nome da função* points to `function_name`.
- Parâmetros de entrada (opcional)* points to `parameters`.
- Devolução de dados (opcional)* points to `return [expr]`.

## ❖ Funções | Conceito

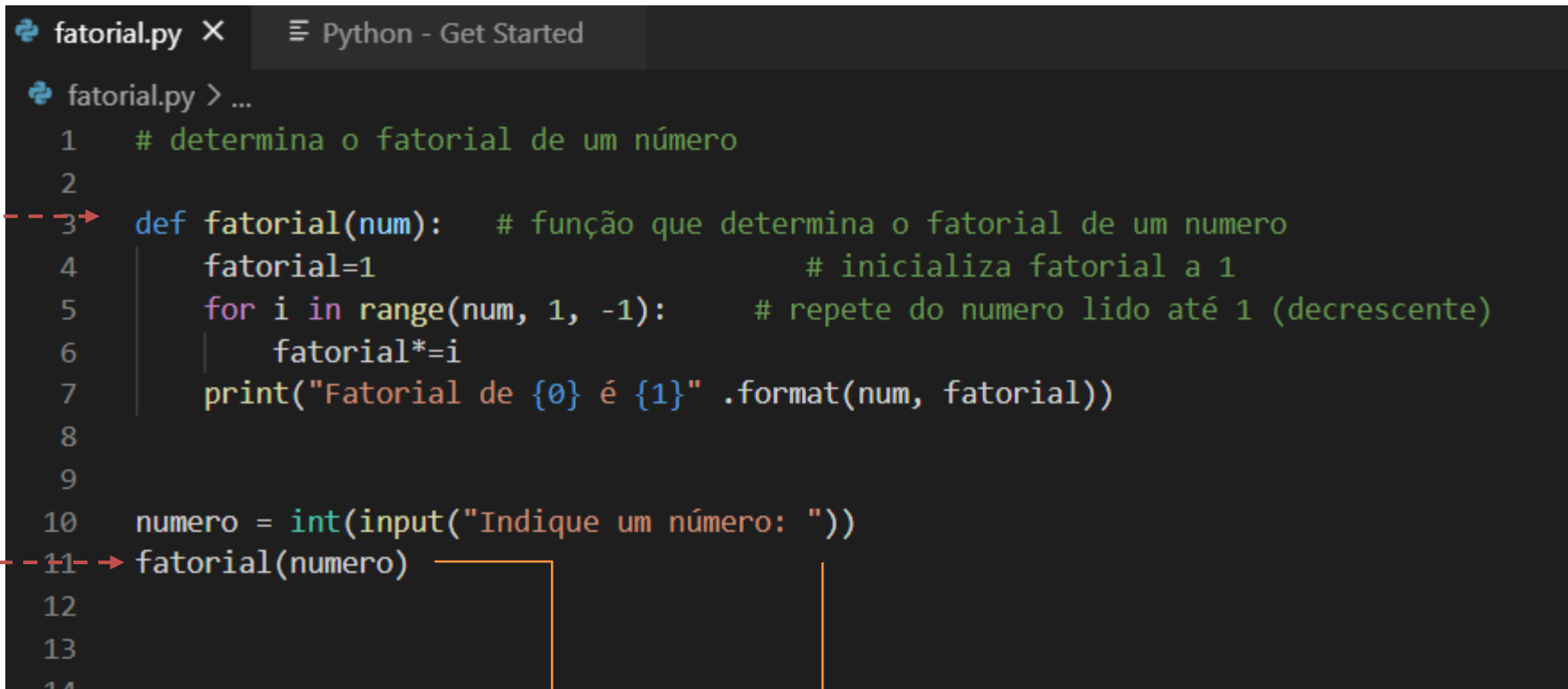
- ❑ As variáveis definidas no âmbito de uma função são **variáveis locais** - existem apenas dentro dessa função, em contraponto a **variáveis globais** - existem durante toda a execução do programa

Variáveis locais  
(existem apenas dentro  
da função)

Variável global

```
fatorial.py X Python - Get Started
fatorial.py > ...
1  # determina o fatorial de um número
2
3  def fatorial(num):  # função que determina o fatorial de um numero
4      fatorial=1      # inicializa fatorial a 1
5      for i in range(num, 1, -1):  # repete do numero lido até 1 (d
6          fatorial*=i
7          print("Fatorial de {0} é {1}" .format(num, fatorial))
8
9
10 numero = int(input("Indique um número: "))
11 fatorial(numero)
12
13
14
```

## ❖ Criar uma Função



```
fatorial.py X Python - Get Started
fatorial.py > ...
1  # determina o fatorial de um número
2
3  def fatorial(num):  # função que determina o fatorial de um numero
4      fatorial=1      # inicializa fatorial a 1
5      for i in range(num, 1, -1):  # repete do numero lido até 1 (decrescente)
6          fatorial*=i
7      print("Fatorial de {0} é {1}" .format(num, fatorial))
8
9
10 numero = int(input("Indique um número: "))
11 fatorial(numero)
12
13
14
```

1. Início da execução do código

2. Invoca a função fatorial

## ❖ Criar uma Função



Por defeito, uma função deve ser chamada com o número correto de argumentos.

O que significa que se sua função **recebe** 3 argumentos, devo invocar a função com 3 argumentos

Exemplo1.py > ...

```
1
2 def soma(num1, num2, num3):
3     soma = num1+num2+num3
4     print("A soma é:", soma)
5     print("A média é", soma/3)
6
7
8
9     num1 = int(input("primeiro numero:"))
10    num2 = int(input("segundo numero:"))
11    num3 = int(input("terceiro numero:"))
12 → soma(num1, num2, num3)
13
14
15
```

C:\WINDOWS\py.exe

```
primeiro numero:10
segundo numero:20
terceiro numero:30
A soma é: 60
A média é 20.0
```

## ❖ Criar uma Função

```
primo.py > ...
1  # Ler um numero e verificar se é primo
2
3
4  def primo(numero): # função que dado um numero indica se é primo
5      primo = True
6      for i in range(numero-1, 1, -1):
7          resto = numero % i
8          if resto == 0:
9              primo = False
10             break
11     if primo==True:
12         print("O numero", numero, "é primo")
13     else:
14         print("O numero", numero, "não é primo")
15
16
17
18     numero = int(input("Numero:"))
19     primo(numero)
20
21
22
```





## ❖ Devolução de dados | return

Quando uma função **devolve um valor** (keyword **return**), ao invocarmos a função devemos atribuir a função a uma variável

```
primo.py > ...  
1  
2  
3 def primo(numero): # função que dado um numero devolve True se for primo, False se não é  
4     primo = True  
5     for i in range(numero-1, 1, -1):  
6         resto = numero % i  
7         if resto == 0:  
8             primo = False  
9             break  
10    return primo  
11  
12  
13  
14 numero = int(input("Numero:"))  
15 estado = primo(numero)  
16 if estado == True:  
17     print("O numero", numero, "é primo")  
18 else:  
19     print("O numero", numero, "não é primo")  
20  
21
```

Devolve um valor no final da execução da função

## ❖ Devolução de dados | return

Exemplo de função fatorial  
**com** devolução de resultado

```
1  # Função que determina o fatorial de um numero
2
3
4  def fatorial(num):
5      fatorial = 1
6      for i in range(num, 1, -1):
7          fatorial*=i
8      return fatorial
9
10
11
12  numero = int(input("Indique um número:"))
13  result=fatorial(numero)
14  print("Fatorial de {0} é {1}" .format(numero, result))
15
16
```

```
fatorial.py X Python - Get Started
fatorial.py > ...
1  # determina o fatorial de um número
2
3  def fatorial(num):  # função que determina o fatorial de
4      fatorial=1      # inicializa fatorial
5      for i in range(num, 1, -1):  # repete do numero lid
6          fatorial*=i
7      print("Fatorial de {0} é {1}" .format(num, fatorial))
8
9
10  numero = int(input("Indique um número: "))
11  fatorial(numero)
12
13
14
```

Exemplo de função fatorial  
**sem** devolução de resultado

## ❖ Parâmetros com valor por defeito



Parâmetros definidos por defeito podem ser omitidos quando chamo a função

```
Fatorial.py > fatorial
1  # Função que determina o fatorial de um numero
2  def fatorial(num= 0):
3      fatorial = 1
4      for i in range(num, 1, -1):
5          fatorial*=i
6      return fatorial
7
8
9
10 print("Fatorial de {0} é {1}" .format(5, fatorial(5)))
11
12 print("Fatorial de {0} é {1}" .format(3, fatorial(3)))
13
14 print("Fatorial de {0} é {1}" .format(0, fatorial()))
15
```

Valor por defeito, quando invoco a função se passar argumento

C:\WINDOWS\py.exe

Fatorial de 5 é 120  
Fatorial de 3 é 6  
Fatorial de 0 é 1

Invoco a função fatorial sem passar qualquer valor

## ❖ Parâmetros com valor por defeito

```
Fatorial.py > ...
1  # Função que determina o fatorial de um numero
2  def fatorial(num):
3      fatorial = 1
4      for i in range(num, 1, -1):
5          fatorial*=i
6      return fatorial
7
8
9
10 print("Fatorial de {0} é {1}" .format(5, fatorial(5)))
11
12 print("Fatorial de {0} é {1}" .format(3, fatorial(3)))
13
14 print("Fatorial de {0} é {1}" .format(0, fatorial()))
```

Neste caso dá erro, pois o parâmetro não contém nenhum valor por defeito

Exception has occurred: TypeError ×  
fatorial() missing 1 required positional argument: 'num'  
File "C:\Users\mario\OneDrive\AED\4 - Exercicios\Funções\Fatorial.py", line 14, in module  
print("Fatorial de {0} é {1}" .format(0, fatorial()))

## ❖ Numero de parâmetros de entrada indefinido

função `len()` conta o número de argumentos que foram passados para a função

```
Exemplo1.py > ...  
1  
2 def soma(*numero):  
3     soma=0  
4     for i in range (len(numero)):  
5         soma = soma + numero[i]  
6     print("A soma é:", soma)  
7  
8  
9 soma(10, 20)  
10 soma(10, 20, 30, 40)  
11  
12  
13
```

```
C:\WINDOWS\py.exe  
A soma é: 30  
A soma é: 100  
-
```