Funções em Python

Disciplina: Programação para Engenharia

Professores: Cristiane Pavei Martinello Fernandes
Douglas de Medeiros Deolindo
Giovani Martins Cascaes
Joel Barbosa Panchyniak
Marcelo Marcos Amoroso
Marcos Antônio Jeremias Coelho
Ramon de Souza Coan

O que são funções?

Funções são pequenos trechos de código reutilizáveis;

Elas permitem dar um nome a um bloco de comandos e executar esse bloco, a partir de qualquer lugar do programa;

Definições de funções auxiliam para automatizar procedimentos que são recorrentes no código, evitando a necessidade de escrevê-los várias vezes.

O que são funções?

```
matriz1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
matriz2 = [[2, 4, 3], [2, 1, 7], [4, 3, 2]]
matriz3 = [[4, 3, 2], [1, 1, 6], [2, 1, 4]]

#Imprime matriz 1
for nlinha in range(3):
    linha = ""
    for ncol in range(3):
        linha = linha + str(matriz1[nlinha][ncol])
    print(linha)

#Imprime matriz 2
for nlinha in range(3):
    linha = ""
    for ncol in range(3):
        linha = linha + str(matriz2[nlinha][ncol])
    print(linha)

#Imprime matriz 2
for nlinha in range(3):
    linha = ""
    for ncol in range(3):
        linha = linha + str(matriz3[nlinha][ncol])
    print(linha)
```

Como definir uma função em Python

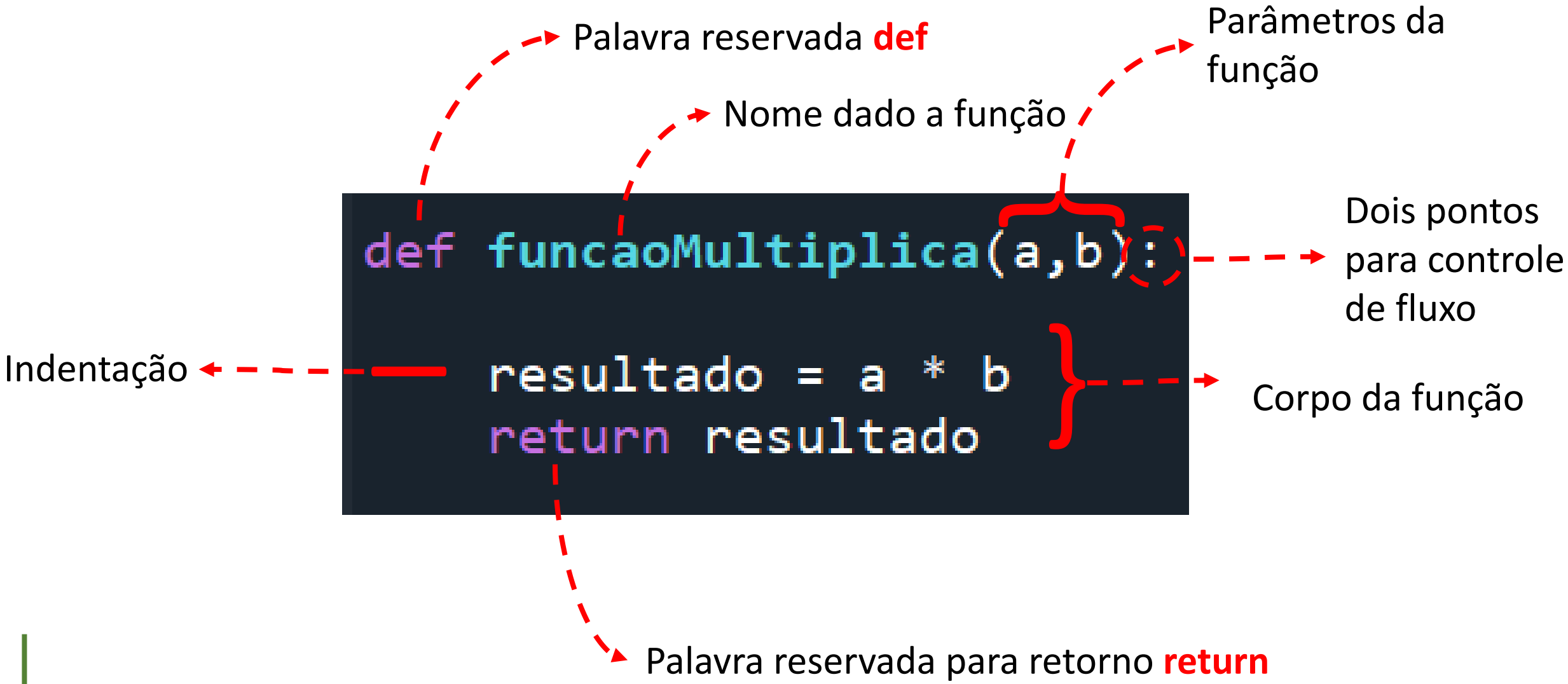
Bem como as outras estruturas do Python, a estrutura de definição de funções também segue as regras da indentação e dos dois pontos;

Para definir a função, usa-se a palavra reservada *def*, seguida do nome da função, dos parâmetros entre parênteses e dos dois pontos;

```
def funcao():
    print("Bloco de código")
```

Para retornar valores ou apenas para indicar o término da função, deve-se usar a palavra *return*.

Estrutura de uma função



Parâmetros e argumentos das funções

Parâmetros são as variáveis que podem ser incluídas nos parênteses das funções;

Quando a função é chamada são passados valores para essas variáveis. Esses valores são chamados argumentos;

O corpo da função pode utilizar essas variáveis, cujos valores podem modificar o comportamento da função.

Parâmetros e argumentos das funções

```
def funcaoSoma(a,b):

    resultado = a+b
    return resultado
```

Parâmetros da função

Argumentos passados

```
valor1 = float(input('Digite o primeiro valor para soma: '))
valor2 = float(input('Digite o segundo valor para soma: '))

resultadoSoma = funcaoSoma(valor1, valor2)

print(resultadoSoma)
```

Retorno de valores

O comando return é usado para retornar um valor de uma função e encerrá-la.

Caso não seja declarado um valor de retorno, a função retorna o valor None (que significa nenhum, sem valor).

Retorno de valores e variáveis

```
def funcaoSoma(a,b):

    resultado = a+b
    print (resultado)

valor1 = 25
valor2 = 36

A = funcaoSoma(valor1,valor2)
print(A)
```

```
61
None
```

Retorno de múltiplos valores

Funções também podem retornar múltiplos dados.

```python
def soma_dois_numeros_e_calcula_media(valor1, valor2):
    soma = valor1 + valor2
    media = (valor1 + valor2)/2

    return soma, media


valor_soma = soma_dois_numeros_e_calcula_media(32, 15)
print(valor_soma)
print(soma_dois_numeros_e_calcula_media(50, 10))
```

```
(47, 23.5)
(60, 30.0)
```

Boas práticas com Funções

Uso da função **main()**

__main__ é o nome do ambiente principal no qual o código é executado

“Ambiente de código principal” é o primeiro módulo Python definido pelo usuário que começa a ser executado

É considerado principal porque ele importa todos os outros módulos que o programa precisa

Boas práticas com Funções

Uso da função **main()**

```
"""Teste funções"""

def media(s, n):
    """Função Media"""
    if n != 0:
        return s/n
    else:
        return None
```

Boas práticas com Funções

Uso da função **main()**

```
def main():
    """Função principal"""
    contador = 0
    soma = 0

    while contador < 4:
        num = int(input('Digite um numero: '))
        soma = soma + num
        contador = contador + 1

    print(f'Media é {media(soma, contador)}')


if __name__ == '__main__':
    main()
```

Praticando com o Python ...

1) Fazer uma função que receba três notas de um aluno e que retorne a média dessas 3 notas.

2) Agora, faça uma função de acordo com a média da função anterior informe o status do aluno de acordo com a tabela a seguir:

- Média acima de 6: “Aprovado”;
- Média entre 4 e 6 :“Verificação Suplementar”;
- Média abaixo de 4 : “Reprovado.

Escopos de variáveis

Toda variável utilizada dentro de uma função tem escopo local, isto é, ela não será acessível por outras funções ou pelo programa principal.

Se houver variável com o mesmo nome fora da função, será uma outra variável, completamente independentes entre si;

```
def funcaoMultiplica(a,b):

    resultado = a * b
    return resultado

resultado = 20
valorRetornado = funcaoMultiplica(30, 50)
```

Exemplo de escopo de variáveis

```
def funcaoMultiplica(a,b):

    resultado = a * b
    return resultado

resultado = 20
valorRetornado = funcaoMultiplica(30, 50)

print(resultado)
print(valorRetornado)
```

```
20
1500
```

Escopos de variáveis

Para uma variável ser compartilhada entre diversas funções e o programa principal, ela deve ser definida como variável global.

Para isto, utiliza se a instrução global para declarar a variável em todas as funções para as quais ela deva estar acessível, o mesmo vale para o programa principal.

Exemplo de escopo de variáveis

```
def funcaoMultiplica(a,b):
    global resultado
    resultado = a * b
    return resultado

global resultado
resultado = 20
valorRetornado = funcaoMultiplica(30, 50)

print(resultado)
print(valorRetornado)
```

```
1500
1500
```

Valores padrões

É possível definir um valor padrão para os parâmetros da função.

Neste caso, quando o valor é omitido na chamada da função, a variável assume o valor padrão.

Valores padrões

```
def funcaoSoma(a = 5,b = 6):

    resultado = a+b
    print (resultado)

funcaoSoma()
```

```
11
```

Parâmetro *args

Caso se queira desenvolver uma função que recebe um número variável de parâmetros, você pode utilizar o parâmetro ***args**!
Dessa forma, a função receberá os argumentos em forma de Tupla e você poderá processá-los com um loop for por exemplo!

```
def maior_30(*args):
    print(args)
    print(type(args))

    for num in args:
        if num > 30:
            print(num)


maior_30(10, 20, 30, 40, 50, 60)
```

```
(10, 20, 30, 40, 50, 60)
<class 'tuple'>
40
50
60
```

Parâmetro *args

Observação:

O nome ***args** é uma convenção, ou seja uma boa prática entre programadores Python!

Contudo, nada te impede de alterar esse nome para ***números** por exemplo. Dessa forma, a definição da função seria:

def maior_30(*números):

Parâmetro **kwargs

Agora, se quiser desenvolver uma função com número variado de parâmetros nomeados, utilize ****kwargs**.
Dessa forma, todos os dados passados à função serão guardados nessa variável ****kwargs**, em formato de um **dicionário**.

```
def dados_pessoa(**kwargs):
    print(type(kwargs))

    for chave, valor in kwargs.items():
        print(f"{chave}: {valor}")


dados_pessoa(nome='João', idade=35, carreira='Desenvolvedor Fullstack')
```

Parâmetro **kwargs

```
<class 'dict'>
nome: João
idade: 35
carreira: Desenvolvedor Fullstack
```

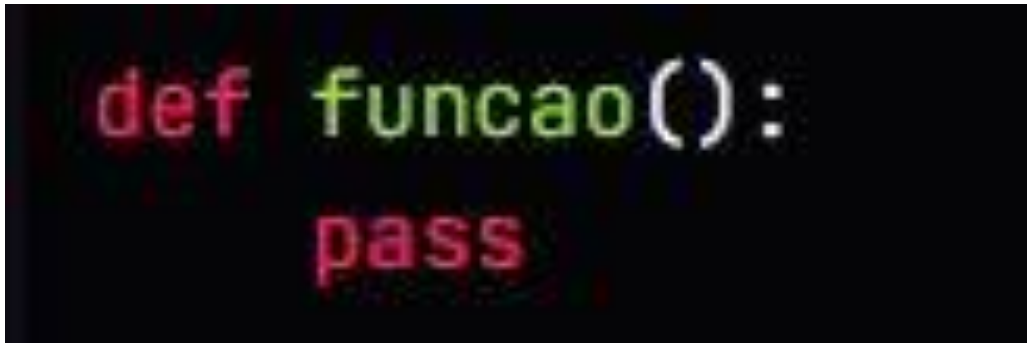
Observação:

O nome ****kwargs** é uma convenção, ou seja uma boa prática entre programadores Python! Contudo, nada te impede de alterar esse nome para ****pessoa** por exemplo.

Palavra reservada **pass**

Caso se deseje definir uma função sem corpo nenhum, ou seja, sem código, saiba que isso irá disparar o erro ***IndentationError***, pois funções não podem estar vazias.

Porém se por algum motivo precisar use a palavra reservada **pass**, da seguinte forma:



Recursividade de funções

A recursividade é uma característica de certas funções que diz respeito a possibilidade de chamar a si mesmas.

Nesses casos, ao definir uma função recursiva é importante atentar-se para impor uma condição que termine a recursão, porque caso contrário, a recursão será infinita e não retornará o resultado desejado.

Exemplo de recursividade

	Função
Chamada 1	fatorial(5)
Chamada 2	5 * fatorial(4)
Chamada 3	4 * fatorial(3)
Chamada 4	3 * fatorial(2)
Chamada 5	2 * fatorial(1)
Chamada 6	return 1
Retorno chamada 5	2 * 1 = 2
Retorno chamada 4	3 * 2 = 6
Retorno chamada 3	4 * 6 = 24
Retorno chamada 2	5 * 24 = 120
Retorno chamada 1	120

```python
def fatorial(numero):
    if numero == 1:
        return 1

    return numero * fatorial(numero - 1)

resultado = fatorial(5)

print(resultado)
```

```
120
```

Utilizando Docstrings

A primeira *string* após a função é chamada de *string* de documento ou **docstring** em resumo.

Isso é usado para descrever a funcionalidade da função.

O uso de **docstring** em funções é opcional, mas é considerado uma boa prática.

```
def evenOdd(x):
    """Function to check if the number is even or odd"""

    if (x % 2 == 0):
        print("even")
    else:
        print("odd")


# Driver code to call the function
print(evenOdd.__doc__)
```

Praticando com o Python ...

3) Escreva uma função para imprimir o nome e salário de um funcionário usando as seguintes condições.

Deve aceitar o nome e o salário do funcionário.

Se o salário estiver faltando na chamada de função, atribua o valor padrão 9000 ao salário.

4) Elabore uma função que recebe como entrada um número inteiro positivo n e retorne a soma de todos os inteiros positivos menores ou iguais a n.

Praticando com o Python ...

4) Desenvolva uma função que recebe dois números (a e b) como parâmetro e retorna **True** caso a soma dos dois seja maior que um terceiro parâmetro, chamado limite.

5) Crie uma função que recebe como entrada uma lista de números e retorna **True** se um número passado como parâmetro está presente na lista.

6) Elabore uma função que recebe como entrada um número ano e retorna **True** caso ano seja bissexto. Caso contrário, retorne **False**.


Obrigado!