

PPT09– Fundamentos de Python (UFCD 10793)

Sandra Liliana Meira de Oliveira



Funções

Funções

- Um aspeto importante na resolução de um problema complexo é conseguir dividi-lo em subproblemas menores.
- Sendo assim, ao criarmos um programa para resolver um determinado problema, uma tarefa importante é dividir o código em partes menores, fáceis de serem compreendidas e mantidas.
- As funções permitem-nos agrupar um conjunto de comandos, que são executados quando a função é chamada.
- Nas aulas anteriores vimos diversos exemplos de uso de funções (range, sum, min, len, etc) – Funções Internas.
- Agora vamos ver como podemos criar as nossas próprias funções.

Por que Utilizar Funções?

- Evitar que os blocos do programa fiquem grandes demais e, por consequência, difíceis de ler e entender.
- Separar o programa em partes que possam ser logicamente compreendidas de forma isolada.
- Permitir o reaproveitamento de códigos, implementados por você ou por outros programadores.
- Evitar que um pedaço de código seja repetido várias vezes dentro de um mesmo programa, evitando inconsistências e facilitando alterações.

Definindo uma Função

- Para criar uma nova função usamos o comando `def`.
- As regras para o nome das funções são as mesmas regras dos nomes de variáveis.

```
1 def imprime_mensagem():  
2     print("Minha primeira função")  
3  
4 imprime_mensagem()  
5 # Minha primeira função
```

Definindo uma Função

- Precisamos sempre definir uma função antes de utilizá-la

```
1 imprime_mensagem()  
2 # NameError: name 'imprime_mensagem' is not defined  
3  
4 def imprime_mensagem():  
5     print("Minha primeira função")
```

Redefinindo uma Função

- Uma função pode ser redefinida, para isso basta declararmos outra função utilizando o mesmo nome, mas não necessariamente com o mesmo comportamento.

```
1 def imprime_mensagem():
2     print("Minha função")
3
4 def imprime_mensagem():
5     print("Minha função foi redefinida")
6
7 imprime_mensagem()
8 # Minha função foi redefinida
```

Definindo uma função

- Uma função é um bloco de código que só é executado quando é chamado.
- Pode-se passar dados, conhecidos como parâmetros, para uma função.
- Uma função pode retornar dados como resultado.

```
def minha_funcao(): #Declaração da função
    print("A função foi chamada!")
#Início do programa
print("O programa começa aqui!")
minha_funcao() #Chamada da função
```


Funções Customizadas – Como criar?

Usar a palavra-chave **def** e com o nome da função para definir uma função.

De seguida definir o número de parâmetros se necessário (**opcional**)

```
def function_nome(parameter1, parameter2):  
    # function body #  
    #escrever Código  
    return Value
```

Por ultimo definir o corpo da função com um bloco de código - **ação a realizar**

Retornar Valor/Valores quando necessário

function_nome: O nome da função é o nome da função. Podemos dar qualquer nome à função.

parameter: Parâmetro é o valor passado para a função. Podemos passar qualquer número de parâmetros. O corpo da função usa o valor do parâmetro para executar uma ação

function_body: O corpo da função é um bloco de código que executa alguma tarefa. Este bloco de código nada mais é do que a ação que quer realizar.

valor de retorno : O valor de retorno é a saída da função

Funções Customizadas – Exemplo

Python Functions

In Python, the **function** is a block of code defined with a name

- A Function is a block of code that only runs when it is called.
- You can pass data, known as parameters, into a function.
- Functions are used to perform specific actions, and they are also known as methods.

Function Name Parameters

```
def add(num1, num2):  
    print("Number 1:", num1)  
    print("Number 2:", num1)  
    addition = num1 + num2  
  
    return addition
```

Function Body

Return Value

res = add(2, 4) → Function call
print(res)

Funções Customizadas – Sem parâmetros

```
def imprime_mensagem():  
    mensagem = "Variável local"  
    print(mensagem)  
  
imprime_mensagem()  
# Variável local  
print(mensagem)  
# NameError: name 'mensagem' is not defined
```

Funções Customizadas – Com parâmetros

Objetivo: função que recebe 2 parâmetros e exibe os seus valores

#Criar Função

```
def curso_func(name, course_name):  
    print("olá", name, "bem vindo")  
    print("nome do curso", course_name)
```

#Chamar Função

```
curso_func('Transformer', 'Python')
```

Funções Customizadas – Com parâmetros e Valor de Retorno

```
# function
def calculator(a, b):
    add = a + b
    # return the addition
    return add
```

```
# call function
# take return value in variable
```

```
res = calculator(20, 5)
print("Addition :", res)
```

Funções Customizadas – Chamar a Função

A função chama-se pelo seu nome.

Exemplo:

```
def par_impar(n):  
    if n % 2 == 0:  
        print('Número Par')  
    else:  
        print('Número Impar')
```

```
par_impar(19) #chamada pelo nome
```

Funções Customizadas – Passagem de Argumentos – Parâmetros

O argumento é um valor, uma variável ou um objeto que passamos para uma função ou chamada de método.

Em Python, existem quatro tipos de argumentos permitidos.

1. Argumentos posicionais
2. Argumentos de palavra-chave
3. Argumentos padrão
4. Argumentos de comprimento variável

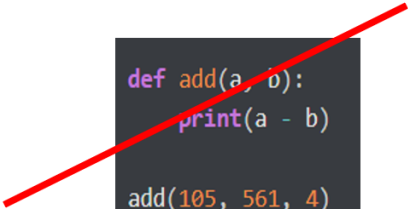
Funções Customizadas – Passagem de Argumentos – Parâmetros

- As funções podem receber dados para serem trabalhados dentro delas.
- Os argumentos são inseridos após o nome da função entre () e caso se pretenda adicionar mais do que um, basta separá-los por vírgulas.
- Uma função deve ser chamada com o número correto de argumentos, caso contrário dá erro.

```
def minha_funcao2(fname):  
    print("Nome: "+fname)  
  
minha_funcao2("Pedro")  
minha_funcao2("João")  
  
def minha_funcao3(fname, lname):  
    print(fname + " " + lname)  
  
minha_funcao3("Patrícia", "Almeida")
```


Funções Customizadas – Passagem de Argumentos – Parâmetros

Os argumentos posicionais são argumentos que são passados para funcionar na **ordem posicional apropriada**



```
def add(a, b):  
    print(a - b)  
  
add(105, 561, 4)
```

```
def add(a, b):  
    print(a - b)
```

```
add(50, 10)  
# Output 40  
add(10, 50)  
# Output -40
```

Funções Customizadas – Passagem de Argumentos – Parâmetros

Argumentos Arbitrários:

Quando não sabemos quantos argumentos são passados para a função.

```
def minha_funcao4(*kids):  
    #Recebe e trata como uma lista  
    print("Filho mais novo: " + kids[2])  
  
minha_funcao4("Emil", "Tobias", "Linus")
```

Argumentos de Palavra-Chave: Também podemos enviar argumentos com a sintaxe chave=valor

```
def minha_funcao5(child3, child2, child1):  
    print("Filho mais velho: " + child1)  
  
minha_funcao5(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
```

Funções Customizadas – Passagem de Argumentos – Parâmetros

Argumentos de Palavra-Chave: Quando não sabemos quantos argumentos palavras-chave são passados para a função.

```
def minha_funcao6(**kid):  
    print("O apelido é " + kid["lname"])  
  
minha_funcao6(fname = "Tobias", lname = "Antunes")
```

Argumento Padrão: Se a função for chamada sem parâmetro é usado o valor padrão.

```
def minha_funcao7(country = "Brasil"):  
    print("Eu sou de " + country)  
  
minha_funcao7("Portugal")  
minha_funcao7()
```

Funções Customizadas – Passagem de Argumentos – Parâmetros

Argumentos do tipo Coleções: Quando é enviada uma coleção como argumento ela ainda será uma coleção do mesmo tipo quando atingir a função

```
def minha_funcao8(food):  
    for x in food:  
        print(x)  
  
frutos = ["Maçã", "Banana", "Uvas"]  
  
minha_funcao8(frutos)
```

Funções Customizadas – Retorno de Valores

Permite que a função devolva algo para a execução do programa

```
def minha_funcao9(x):  
    return 5 * x  
  
print(minha_funcao9(3))  
print(minha_funcao9(5))  
print(minha_funcao9(9))
```

```
def minha_funcao10():  
    idade=35  
    nome = "Ines"  
    return nome, idade  
  
var_1, var_2 = minha_funcao10()  
print (var_1, var_2)
```

Variaveis em funções

Uma **variável local** é uma variável declarada **dentro da função** que **não é acessível de fora da função**. O scope (escopo) da variável local é limitado a essa função apenas onde ela é declarada.

Uma **variável global** é uma variável declarada fora da função. O scope de uma **variável global é amplo**. É acessível em todas as funções do mesmo módulo.

Escopo de uma Variável – Variável Local

- O escopo de uma variável é o local do programa onde ela é acessível.
- Quando criamos uma variável dentro de uma função, ela só é acessível nesta função – Variável LOCAL.

```
1 def imprime_mensagem():  
2     mensagem = "Variável local"  
3     print(mensagem)  
4  
5 imprime_mensagem()  
6 # Variável local  
7 print(mensagem)  
8 # NameError: name 'mensagem' is not defined
```

Escopo de uma Variável – Variável Global

- Quando criamos uma variável fora de uma função, ela também pode ser acessada dentro da função. Essas variáveis são chamadas de globais.

```
1 mensagem = "Variável global"
2 def imprime_mensagem():
3     print(mensagem)
4
5 imprime_mensagem()
6 # Variável global
7 print(mensagem)
8 # Variável global
```


Escopo de uma Variável

- Uma variável local com o mesmo nome de uma global, “esconde” a variável global.

```
1 a = 1
2 def imprime():
3     a = 5
4     print (a)
5
6 imprime()
7 # 5
8
9 print (a)
10 # 1
```

Escopo de uma Variável

- Uma variável local com o mesmo nome de uma global, “esconde” a variável global.

```
1 a = 1
2 def incrementa():
3     a = a + 1
4     print(a)
5
6 incrementa()
7 # UnboundLocalError: local variable 'a' referenced before
8     assignment
9
```

Escopo de uma Variável

- Uma variável local com o mesmo nome de uma global, “esconde” a variável global.

```
1 a = 1
2 def incrementa():
3     a = 12
4     a = a + 1
5     print(a)
6
7 incrementa()
8 # 13
9 print(a)
10 # 1
```

Escopo de Uma Variável e Argumentos

- Na medida do possível devemos evitar o uso de variáveis globais dentro de funções, que dificultam a compreensão, manutenção e reuso da função.
- Se uma informação externa for necessária, ela deve ser fornecida como argumento da função.
- Podemos definir argumentos que devem ser informados na chamada da função.

```
1 def imprime_mensagem(mensagem) :  
2     print(mensagem)  
3  
4 bomdia = "Bom dia"  
5 imprime_mensagem(bomdia)  
6 # Bom dia
```

- O **escopo dos argumentos** é o mesmo das variáveis criadas dentro da função (**variáveis locais**).

Variáveis Locais Versus Variáveis Globais

- Toda variável utilizada dentro de uma função é considerada local, isto é, ela não é “vista” por outras funções nem pelo programa principal.
- Se houver uma variável com o mesmo nome fora da função, será uma outra variável, sendo completamente independentes entre si.
- Para criar uma variável global para ser “vista” por outras funções e pelo programa principal utiliza-se a instrução **global** para declarar variável em todas os locais onde se pretende que ela esteja acessível.

Funções – Variáveis Globais - Exemplos

```
#total é utilizada duas vezes como local
def soma(x,y):
    total=x+y
    print("Total soma = ",total)
#programa principal
total=10
soma(3,5)
print("Total principal = ",total)
```

```
Total soma = 8
Total principal = 10
```

```
#total é utilizada com global
def soma(x,y):
    global total #Definir como global
    total=x+y
    print("Total soma = ",total)
#programa principal
global total
total=10
soma(3,5)
print("Total principal = ",total)
```

```
Total soma = 8
Total principal = 8
```

A Função `main`

- Para manter o código bem organizado, podemos separar todo o programa em funções.
- Neste caso, a ultima linha do código contém uma chamada para a função principal (por convenção chamada de `main`).

```
1 def main():  
2     print("Execução da função main")  
3  
4 main()  
5 # Execução da função main
```

A Função main

Como a chamada da função `main` fica no final do código, não precisamos nos preocupar com a ordem em que as outras funções são definidas.

```
1
2 def função2():
3     print("Execução da função 2")
4
5 def função1():
6     print("Execução da função 1")
7
8 def main():
9     função1()
10    função2()
11
12
13 main()
    # Execução da função 1
    # Execução da função 2
```