Aula 08 – Funções

TECNOLOGIA em **Gestão de Dados**

Programação I

Prof. Dr. Wener Sampaio

ReUni DIGITAL · EAD Centro de Educação Aberta e a Distância · UFPI Universidade Federal do Piauí · MEC

Sumário

- Funções
- Variáveis locais e globais
- Passagem de parâmetros

Funções

- São blocos de comandos que recebem um apelido.
- Programação modular:
  - Dividir o programa em subunidades.
- Python:
  - Modular, imperativa, orientada a objetos e funcional
- Objetivos:
  - Simplificar o programa.
  - Facilitar correções e manutenção.
  - Reutilização de código.
  - Bibliotecas de funções.

Funções

- Tipos:
  - Funções externas: criadas por linguagens de programação imperativa.
    - print(), dir(), len(), id(), type(),...
  - Funções internas: criadas por linguagens orientadas a objetos.
    - ‘araraquara’.index(), str.count(), nome.upper()
  - Com retorno: produzem um valor ao final de sua execução.
    - tam=len(lista)
  - Sem retorno: apenas executam, mas não produzem valor.
    - print(“oi”)

Funções

- Objetivos:
  - Simplificar o programa.
  - Facilitar correções e manutenção.
  - Reutilização de código.
  - Bibliotecas de funções.
- Dica importante:
  - Funções devem fazer uma única coisa e muito bem feita.
  - Exemplo:
    - Não faça uma função que cria uma casa. Faça uma função que cria quartos, outra que cria banheiros, outra que cria sala, outra que cria um jardim, outra que cria um cozinha etc.
    - Monte a casa.
    - Monte diversas casas

Funções

- Até o momento, apenas utilizamos funções da linguagem python. Agora criaremos as nossas.

- Sintaxe: função sem retorno

```
def NOME_DA_FUNCAO(<LISTA_DE_PARAMETROS>) :
    <COMANDOS>
```

- Sintaxe: função com retorno

```
def NOME_DA_FUNCAO(<LISTA_DE_PARAMETROS>) :
    <COMANDOS>
    return valor
```

Funções

- Até o momento, apenas utilizamos funções da linguagem python. Agora criaremos as nossas.

- Sintaxe: função sem retorno

  def NOME_DA_FUNCAO**(**<LISTA_DE_PARAMETROS>**) :**

  <COMANDOS>

- Sintaxe: função com retorno

  def NOME_DA_FUNCAO**(**<LISTA_DE_PARAMETROS**>) :**

  <COMANDOS>

  return valor

Funções

- Trabalhando com módulos:
  - Importar apenas uma função do módulo:
    - From <módulo> import <função>
  - Importar todas as funções do módulo:
    - Import <módulo>

Funções

- Exercício
  - Fazer funções que calculam a média, mediana, variância, desvio padrão de uma lista. Salvar no arquivo estatisticas.py
- Módulos:
  - Crie um arquivo chamado principal.py que utiliza as funções de estatistica.py
  - Arquivos na mesma pasta
- Pacotes: organiza módulos em pastas diferentes
  - __init__.py deve estar presente em cada subpasta
  - Melhor organização

Funções

- Trabalhando com módulos e pacotes
  - Módulos melhoram a organização de um projeto.
  - Possuem funções de objetivos semelhantes.
  - Um grande projeto pode ter diversos módulos e pacotes.
  - Geralmente módulos contém apenas as declarações, ou seja, não executam comando algum.
  - Estas funções são utilizadas nas importações.
  - Assim, precisamos de um arquivo principal (do tipo __main__) que serve de ponto de partida.

Funções

- Trabalhando com módulos e pacotes
  - Como saber se um arquivo é do tipo __main__?
  - Todo arquivo python recebe um nome interno, que fica armazenado na variável oculta __name__
  - Se o arquivo foi importado então __name__ é igual ao nome do módulo.
  - C.C. __name__ é igual a “__main__”.
  - **Usamos o teste no final do arquivo principal**:

```
if __name__ == "__main__":
    <bloco_de_comandos>
```

Funções

- Variáveis globais e locais
  - Locais: só existem dentro da função/bloco.
  - Globais: existem no programa inteiro.
- Atenção:

```
1  def func():
2      print(x)
3
4  x = 42 #v. global
5  func()
```

```
1  def func():
2      print(x)
3      x=1 #local
4
5  x = 42 #v. global
6  func()
```

```
1  def func():
2      global x
3      x=1 #global
4      print(x)
5
6  x = 42 #v. global
7  func()
8  print(x)
```

```
1  def func():
2      x=1 #local
3      print(x)
4
5  x = 42 #v. global
6  func()
```

```
1  def func():
2      x=1 #local
3      print(x)
4
5  x = 42 #v. global
6  func()
7  print(x)
```

Evite o construtor global!
Evite variáveis globais!

Funções

- Parâmetros
  - São atributos que uma função pode receber.
  - Definem quais argumentos são aceitos por uma função.
  - Podem ou não ter um valor padrão.
  - Argumentos são os valores contidos nos parâmetros.

```
def calculadora_salario(valor, horas=1):
    return horas * valor

valor_total = calculadora_salario(299.25)
print(valor_total)

valor_total = calculadora_salario(299.25, 100)
print(valor_total)
```

Funções

- Parâmetros
  - Valores padrões são definidos na chamada da função, ao invés da definição.

```
x=100
def raiz(valor=x):
    return valor**0.5

x=4
print(raiz(x))
```

```
def soma(v1=0,v2=0,v3=0):
    return v1+v2+v3
x=5
print(soma(v2=10,v3=100,v1=x))
```

  - A ordem dos parâmetros padrões não é obrigatória.
  - Parâmetros obrigatórios devem vir primeiro.

Funções

- Parâmetros: passagem por valor
  - Internamente valores mutáveis são modificáveis. Valores imutáveis não são modificáveis.
  - Atenção às atribuições.

```
def modificar(x,y,z):
    x=23           #nova referência
    y.append(42)   #referência anterior
    z=[99]         #nova referência
    print(x,y,z)


a=77    #imutável
b=[99]  #mutável
c=[28]

modificar(a,b,c)
print(a,b,c)
```

```
23 [99, 42] [99]
77 [99, 42] [28]
```

Exercícios

- Crie uma função que retorna as raízes de uma equação do 2o grau: $ax^2 + bx + c=0$ (reais ou complexas). Crie antes uma função que calcula o delta.
- Crie uma função que receba 2 números e retorne o maior valor.
- Utilize a função anterior e crie uma função que calcula o maior entre três números.

Exercícios

- Crie uma função chamada dado() que retorna aleatoriamente um número de 1 até 6. Chame a função dado() mil vezes e imprima quantas vezes cada valor foi sorteado.
- Crie uma função que retorna todos os números primos até 1000.
- Crie uma função que recebe dois inteiros e retorna o seu MDC.