Funções

Sumário

Funções - Motivação

O código ao lado calcula a soma, o maior e o menor valor da uma lista:

```
valores = [5, 3, 7, 2, 8, 6, 5, 3]

# Soma os valores da lista
soma = 0
n = len(valores)
for i in range(0, n):
    soma += valores[i]

# Encontra o maior valor
maior = valores[0]
for i in range(0, n):
    if valores[i]>maior:
        maior = valores[i]

# Encontra o menor valor
menor = valores[0]
for i in range(0, n):
    if valores[i]<menor:
        menor = valores[i]

print(f"Soma: {soma}")
print(f"Maior: {maior}")
print(f"Menor: {menor}")
```

Funções - Motivação

Podemos organizar esse código em funções que realizam tarefas específicas:

```
def soma_valores(valores):
    n = len(valores)
    soma = 0
    for i in range(n):
        soma += valores[i]
    return soma

def menor_valor(valores):
    n = len(valores)
    menor_valor = valores[0]
    for i in range(0, n):
        if valores[i]<menor_valor:
            menor_valor = valores[i]
    return menor_valor

def maior_valor(valores):
    n = len(valores)
    maior_valor = valores[0]
    for i in range(0, n):
        if valores[i]>maior_valor:
            maior_valor = valores[i]
    return maior_valor
```

Funções - Motivação

As funções criadas podem ser utilizadas no restante do código a qualquer momento:

```
valores = [5, 8, 7, 2, 7, 10, 21, 1, 8]

soma = soma_valores(valores)
maior = maior_valor(valores)
menor = menor_valor(valores)

print(f"Soma: {soma}")
print(f"Maior: {maior}")
print(f"Menor: {menor}")
```

Funções

Quando temos um código que executa uma tarefa específica (somar valores, encontrar uma linha da tabela, identificar valores repetidos, etc), é útil criarmos uma função com esse código.

Essa função pode ser utilizada diversas vezes e a qualquer momento.

A criação de funções permite a modularização de um código, o que torna o código mais fácil de ser entendido e de ser reaproveitado.

Função - Sintaxe

Para definir funções utilizamos a seguinte sintaxe:

"define"

Parâmetros recebidos pela função, separados por vírgula. Pode ser qualquer número de parâmetros

```
def nome_funcao(parametro1, parametro2, ...):

    comando1
    comando2
    ...

    return valor1, valor2, ...
```

Valores retornados pela função, separados por vírgula. Pode ser qualquer número de valores

Função - Sintaxe

Chamamos a função da seguinte forma:

Argumentos recebidos pela função, separados por vírgula. O número de argumentos precisa ser igual ao número de parâmetros

```
valor1, valor2, ... = nome_funcao(argumento1, argumento2, ...)
```

Variáveis que recebem os valores retornados pela função, separadas por vírgula. O número de variáveis precisa ser igual ao número de valores retornados

Função - Sintaxe

Exemplo:

```
def soma(valor1, valor2):
    resultado = valor1 + valor2
    return resultado

val1 = 3
val2 = 8
res = soma(val1, val2)
```

Função - Sintaxe

Exemplo:

```
def soma(valor1, valor2):
    resultado = valor1 + valor2
    return resultado

val1 = 3
val2 = 8
res = soma(val1, val2)
```

Quando o comando `res = soma(val1, val2)` é executado, ocorre o seguinte:

1. As variáveis **valor1** e **valor2** recebem uma cópia dos valores das variáveis val1 e val2;
2. A soma dos valores é calculada e atribuída à variável **resultado**;
3. A função retorna o conteúdo da variável **resultado**;
4. A variável **res** recebe uma cópia do conteúdo da variável **resultado**.

Função - Sintaxe

- Para funções, sempre utilizamos parênteses ()
- É comum fazermos confusão entre funções e listas, que usam colchetes []

- O que está ocorrendo nos códigos abaixo?

  - res = soma[val]
  - res = valores(indice)

*Os nomes das variáveis foram definidos de forma não intuitiva de propósito

Função - Sintaxe

- Para funções, sempre utilizamos parênteses ()
- É comum fazermos confusão entre funções e listas, que usam colchetes []

- O que está ocorrendo nos códigos abaixo?

soma é uma lista de valores, e o valor no índice **val** está sendo acessado

- res = soma[val]

valores é uma função que recebe como argumento o valor da variável **indice**

- res = valores(indice)

Função - Sintaxe

- Os parâmetros de funções e os valores retornados podem ser qualquer tipo de dado

```
def compara(lista1, lista2):
    ...
    return valor

def maior_palavra(string1, string2):
    ...
    return maior_string
```

Função - Sintaxe

- Uma função também pode não receber nenhum parâmetro, nem retornar nenhum valor:

```
def despedida():
    print("Até mais!")

valor = int(input("Digite um valor: "))
print(f"Valor ao quadrado: {valor**2}")
despedida()
```

- É importante lembrar que mesmo que a função não receba nenhum parâmetro, ainda precisamos utilizar () na definição e na chamada da função

Documentação de funções

- É importante sempre incluirmos um comentário explicando o que a função faz, quais os parâmetros da função e o que ela retorna.
- Em Python, o padrão é usar comentários criados com três aspas duplas no começo da função:

```
def soma_valores(valores):
    """Soma os valores de uma lista.
    Parâmetros:
        valores: Lista contendo números.
    Retorna:
        soma: Soma dos valores."""
    n = len(valores)
    soma = 0
    for i in range(n):
        soma += valores[i]
    return soma
```

*Em alguns slides a documentação completa da função não será incluída para que o código caiba no slide.

Teste de funções

Em todos os códigos que faremos no restante da disciplina, sempre criaremos uma função testes(), e chamaremos ela ao final do código para testar uma função criada:

```
def soma_valores(valores):
    """Soma os valores de uma lista."""
    n = len(valores)
    soma = 0
    for i in range(n):
        soma += valores[i]
    return soma

def testes():
    valores = [5, 2, 6, 8, 4]
    soma = soma_valores(valores)
    print(f"Soma: {soma}")

    valores = []
    soma = soma_valores(valores)
    print(f"Soma: {soma}")

testes()
```

Exercício 1 - Contagem

Faça uma função chamada **contagem** que receba como entrada uma lista de valores e um número. A função retorna quantas vezes o número ocorre na lista.

Solução

```
def contagem(valores, num):
    """Conta o número de ocorrências de um número em uma lista.
    Parâmetros:
        valores: Lista de valores
        num: Número a ser procurado na lista
    Retorna:
        cont: Quantas vezes num ocorre em valores
    """
    n = len(valores)
    cont = 0
    for i in range(0, n):
        if valores[i]==num:
            cont += 1

    return cont
```

Testes da solução

```
def testes():
    # Valor ocorre várias vezes
    valores = [4, 1, 3, 7, 6, 5, 1, 6, 4, 7, 8, 5, 1, 3]
    valor = 1
    ocorrencias = contagem(valores, valor)
    print(ocorrencias) # Valor 3 esperado

    # Lista vazia
    valores = []
    valor = 1
    ocorrencias = contagem(valores, valor)
    print(ocorrencias) # Valor 0 esperado

    # Valor não ocorre na lista
    valores = [4, 1, 3, 7, 6, 5, 1, 6, 4, 7, 8, 5, 1, 3]
    valor = 12
    ocorrencias = contagem(valores, valor)
    print(ocorrencias) # Valor 0 esperado
```

Código completo

```
def contagem(valores, num):
    """Conta o número de ocorrências de um número em uma lista."""
    n = len(valores)
    cont = 0
    for i in range(0, n):
        if valores[i]==num:
            cont += 1

    return cont

def testes():
    # Valor ocorre várias vezes
    valores = [4, 1, 3, 7, 6, 5, 1, 6, 4, 7, 8, 5, 1, 3]
    valor = 1
    ocorrencias = contagem(valores, valor)
    print(ocorrencias) # Valor 3 esperado

    # Lista vazia
    valores = []
    valor = 1
    ocorrencias = contagem(valores, valor)
    print(ocorrencias) # Valor 0 esperado

    # Valor não ocorre na lista
    valores = [4, 1, 3, 7, 6, 5, 1, 6, 4, 7, 8, 5, 1, 3]
    valor = 12
    ocorrencias = contagem(valores, valor)
    print(ocorrencias) # Valor 0 esperado

testes()
```

Escopo de variáveis

Variáveis definidas dentro de uma função são válidas apenas dentro da função:

```
def alguma_funcao(numero):
    resultado = 20

    return resultado

valor = 10
res = alguma_funcao(valor)
# Os comandos print abaixo darão erro, pois as variáveis
# numero e resultado não existem no escopo fora da função
print(numero)      # Erro variável inexistente
print(resultado)   # Erro variável inexistente
```

Escopo de variáveis

Funções sempre recebem uma cópia dos valores passados como argumento:

```
def alguma_funcao(valor):
    # A variável valor acima definida como parâmetro da função
    # receberá uma cópia do conteúdo da variável valor definida
    # fora da função.

    print(valor) # Imprime número 10
    valor = 25
    print(valor) # Imprime número 25

    return valor

valor = 10
print(valor)  # Imprime número 10
alguma_funcao(valor)
print(valor) # Imprime número 10, pois valor não foi modificada
```