Disciplina Lógica de Programação

Funções em Python

Téc. Informática Integrado ao Ensino Médio - 2018

Prof. Dr. Paulo César Rodacki Gomes
paulo.gomes@ifc.edu.br

Funções

- Uma função é um objeto que estabelece uma relação de entrada e saída

- Uma função recebe uma entrada e produz uma saída

- Se f é uma função cuja saída é b quando um valor de entrada é a, escrevemos:

$$f(a) = b$$

- É comum também usarmos o termo *mapeamento* para nos referirmos a funções

Funções

- O conjunto de possíveis entradas de uma função é chamado de **domínio** da função

- As saídas de uma função vêm de um conjunto chamado de **contradomínio**

- A notação para dizer que f é uma função com domínio D e contradomínio C é:

$$f : D \rightarrow C$$

Funções

- Podemos descrever funções de várias maneiras. Uma delas é com uma tabela listando todas as possíveis entradas e fornecendo a saída para cada entrada

- Exemplo: $f : \{0,1,2,3,4\} \rightarrow \{0,1,4,9,16\}$

n	*f(n)*
0	0
1	1
2	4
3	9
4	16

Funções

- Uma outra maneira de escrever uma função é por meio de um procedimento de cálculo do valor de saída a partir do valor de entrada

- Exemplo: $f(x) = x^2$

- Uma terceira maneira é criando um procedimento computacional para programar os passos necessários para obter a saída da função a partir de uma entrada

Sequências

- Uma sequência de objetos é uma lista desses objetos em uma determinada ordem

- Geralmente escrevemos sequências entre parêntesis

- Exemplo: (7, 21, 57) e (21, 7, 57) são sequências diferentes

Tuplas

- Sequências podem ser finitas ou infinitas
- Sequências finitas são chamadas de ***tuplas***
- Uma sequência de k objetos é chamada de ***k-tupla***

Funções

- Uma função pode receber mais de um valor, por exemplo: $f(a, b) = \sqrt{a^2 + b^2}$

- Quando o domínio de uma função f é $A_1 \times A_2 \times \ldots \times A_k$, os conjuntos $A_1, A_2, \ldots, A_k$, dizemos que a entrada de f é uma *k-tupla* $(a_1, a_2, \ldots, a_k)$ e chamamos os elementos a_i de argumentos ou parâmetros da função

Funções na programação

Na programação, funções são **blocos de código** que realizam determinadas **tarefas** que normalmente precisam ser executadas diversas vezes dentro de uma aplicação.

Quando surge essa necessidade, para que várias instruções não precisem ser repetidas, elas são agrupadas em uma função, à qual é dado um **nome** e que poderá ser **chamada/executada** em diferentes partes do programa.

Funções na programação

Não repita código.

Crie uma função!!!!!!!

Exemplo

- Se, num programa Python, tivermos a tarefa de “cantar parabéns para o Carlos”, faríamos o seguinte:

```
print("Parabéns pra você!")
print("Parabéns pra você!")
print("Parabéns para o Carlos!")
print("Parabéns pra você!")
```

Exemplo

- Se vc precisar cantar mais de uma vez, poderia dar um nome descritivo a esta tarefa, e mandar o Python executá-la varias vezes
- Se você chamar de parabensCarlos, ficaria assim:

```
def parabensCarlos():
    print("Parabéns pra você!")
    print("Parabéns pra você!")
    print("Parabéns para o Carlos!")
    print("Parabéns pra você!")
```

Sintaxe de uma função

```
def nome_da_funcao(parâmetros):
  """docstring"""
  código da função
```

Chamada de função

- Uma vez que a função está definida (criada), podemos chamá-la em nosso programa, ou até chamá-la de dentro de outra função

- “Chamar" uma função significa mandar o python executá-la

- para chamar uma função, simplesmente escrevemos o nome da função com os parâmetros apropriados

Chamada de função

- Exemplo

```
# definição de uma função
def minha_funcao():
    print("Bom dia")
    print("Boa tarde")
    print("Boa noite")

# chamada da função
minha_funcao()
```

Função quadrado

- Exemplo 1: recebe um valor numérico e imprime o quadrado do valor recebido

```python
def quadrado(n):
    resultado = n * n
    print(f"O quadrado de {n} é {resultado}")

num = int(input("Digite um número: "))
quadrado(num)
```

Função quadrado

- Exemplo 2: recebe um valor numérico e **retorna** quadrado do valor recebido

```
def quadrado(n):
    resultado = n * n
    return resultado

num = int(input("Digite um número: "))
res = quadrado(num)
print(f"O quadrado de {num} é {res}")
```

Função quadrado

- Agora, podemos usar o retorno da função diretamente dentro de expressões

```
def quadrado(n):
    resultado = n * n
    return resultado

x = int(input("Digite o valor de x:"))
y = int(input("Digite o valor de y:"))
z = quadrado(x) + quadrado(y)
print(f"O resultado é {z}")
```

Função retornando mais de um valor

- Em python, funções podem retornar mais de um valor
- Neste caso, os valores de retorno são “empacotados" em um tupla, que pode ser desempacotada fora da função

Função retornando mais de um valor

- Exemplo1: ordenação de 3 valores numéricos, desempacotando a tupla (com as variáveis x, y e z)

```
def ordena(a, b, c):
    menor = min(a, b, c)
    maior = max(a, b, c)
    meio = a + b + c - maior - menor
    return menor, meio, maior

x, y, z = ordena(3, 2, 1)
print(x, y, z)
```

Função retornando mais de um valor

- Exemplo 2: ordenação de 3 valores numéricos, recebendo a tupla inteira

```
def ordena(a, b, c):
    menor = min(a, b, c)
    maior = max(a, b, c)
    meio = a + b + c - maior - menor
    return menor, meio, maior

t = ordena(3, 2, 1)
print(t)
print(t[0], t[1], t[2])
```

- Saída:

```
(1, 2, 3)
1 2 3
```

Escopo de uma variável

- Escopo de uma variável é a porção do programa no qual a variável é reconhecida

- Parâmetros e variáveis definidas dentro de uma função **não são visíveis fora da função**

- Portanto, dizemos que essas variáveis e parâmetros possuem um **escopo local**

Ciclo de vida de uma variável

- O ciclo de vida de uma variável é o período no qual a variável existe na memória

- O ciclo de vida de uma variável dentro de uma função é o mesmo da execução da função. Quando a função termina sua execução, a variável deixa de existir na memória

- As variáveis locais são destruídas assim que a função retorna seu(s) valor(es). Portanto, uma função não lembra dos valores de suas variáveis de chamadas anteriores

Escopo e ciclo de vida

- Exemplo 1:

```
def soma(a, b):
    total = a + b
    print("Total local = ", total)

soma(4, 9)
print("Total principal = ", total)
```

- Saída:

```
Total local =  13
Traceback (most recent call last):
  File "/Users/rodacki/Desktop/funcoes.py", line 55, in
<module>
    print("Total principal = ", total)
NameError: name 'total' is not defined
```

Escopo e ciclo de vida

- Exemplo 2:

```
total = 0
def soma(a, b):
    total = a + b
    print("Total local = ", total)

soma(4, 9)
print("Total principal = ", total)
```

- Saída:

```
Total local =  13
Total principal = 0
```

Escopo e ciclo de vida

- Exemplo 2:

```
total = 0
def soma(a, b):
    global total
    total = a + b
    print("Total local = ", total)

soma(4, 9)
print("Total principal = ", total)
```

- Saída:

```
Total local =  13
Total principal = 13
```

Chamando uma função dentro de outra função

```python
import math

def quadrado(n):
    resultado = n * n
    return resultado

def hipotenusa(a, b):
    resultado = math.sqrt(quadrado(a) + quadrado(b))
    return resultado

print(hipotenusa(3,4))
```

- Saída:

```
5.0
```

Resumo

- funções eliminam repetição de código
- uma função deve realizar uma única tarefa
- evite colocar prints e inputs dentro da função (a não ser que o objetivo específico da função seja fazer prints e inputs)
- observe sintaxe, parâmetros e valores de retorno
- cuidado com escopo e ciclo de vida de variáveis

Disciplina Lógica de Programação

Funções em Python

Téc. Informática Integrado ao Ensino Médio - 2018

Prof. Dr. Paulo César Rodacki Gomes
paulo.gomes@ifc.edu.br