Introdução ao Python

Funções

Henrique Y. Shishido

Departamento de Computação
Universidade Tecnológica Federal do Paraná

Introdução

- Uma função é uma sequência de instruções que somente é executada quando chamada;
- Funções promovem a reusabilidade, modularidade, manutebilidade e legibilidade de código;

Introdução

- Imagine um programa que necessita calcular o IMC (Índice de Massa Corporal) de diferentes pessoas
  - Sem o uso das funções, poderia haver a repetição de um mesmo bloco de instruções dentro do código-fonte do programa

```
#Cálculo do IMC de 3 pessoas
>>> altura = input("Digite a altura: ")
>>> peso = input("Digite o peso: ")
>>> imc = peso/(altura*altura)
>>> print("O IMC é {0}".format(imc))
>>> altura = input("Digite a altura: ")
>>> peso = input("Digite o peso: ")
>>> imc = peso/(altura*altura)
>>> print("O IMC é {0}".format(imc))
>>> altura = input("Digite a altura: ")
>>> peso = input("Digite o peso: ")
>>> imc = peso/(altura*altura)
>>> print("O IMC é {0}".format(imc))
```

Criação de uma função

- Uma função em Python é definida por meio da palavra-chave `def`:

```
>>> def imprimirHello():
>>>     print("----------------------------")
>>>     print("Olá essa é a primeira função")
>>>     print("----------------------------")
```

Chamada de função

- A chamada de função é realizada por meio do uso do nome da função seguido de parênteses. Exemplo: `imprimirHello()`:

```
>>> def imprimirHello():
>>>     print("----------------------------")
>>>     print("Olá essa é a primeira função")
>>>     print("----------------------------")
>>>
>>> imprimirHello()
----------------------------
Olá essa é a primeira função
----------------------------
```

Argumentos de uma função

- Uma função pode receber dados por meio de argumentos;
- Argumentos são passados à função dentro dos parênteses. Uma função pode receber quantos argumentos forem necessários, separados por vírgula
- Exemplo: `calcularArea(altura, largura)`:

```
>>> def calcularArea(altura, largura):
>>>     area = altura * largura
>>>     print("A área é {0} m2.".format(area))
>>>
>>> calcularArea(5, 3)
A área é 15 m2.
```

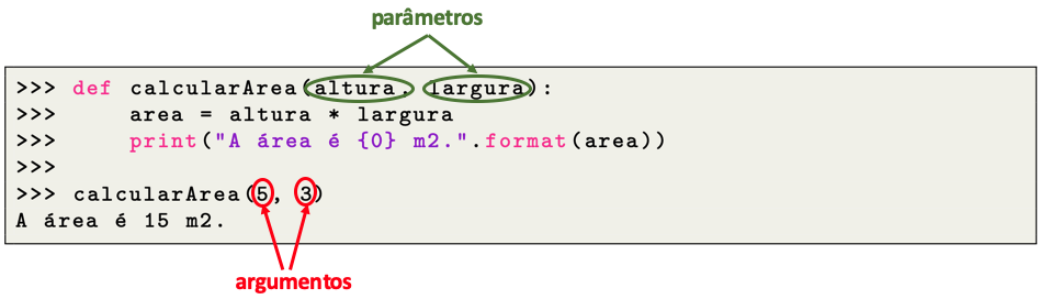
Argumentos de uma função

- Uma função pode receber dados por meio de argumentos;
- Argumentos são passados à função dentro dos parênteses. Uma função pode receber quantos argumentos forem necessários, separados por vírgula
- Exemplo: `calcularArea(altura, largura)`:

```
>>> def calcularArea(altura, largura):
>>>     area = altura * largura
>>>     print("A área é {0} m2.".format(area))
>>>
>>> calcularArea(5, 3)
A área é 15 m2.
```

Parâmetros ou argumentos?

- O termo **parâmetro** e **argumento** são usados para representar a(s) informação(ões) que serão passadas a função. Porém, perante a perspectiva de uma função, há uma diferença:
  - **Parâmetro**: é a variável que permite acessar a informação passada dentro da função.
  - **Argumento**: é o valor que é enviado na chamada da função



Número de argumentos

- Por padrão, uma função deve ser chamada passando o número de correto de argumentos. Se uma função foi implementada aguardando dois argumentos, é preciso passar exatamente dois argumentos
- Contudo, se não for possível determinar o número de argumentos que deverão ser passados para a função, pode-se utilizar o carácter * antes do nome do argumento.

```
>>> def somarLados(*lados):
>>>     soma = 0
>>>
>>>     for l in lados:
>>>         somaLados += l
>>>
>>>     print(soma)
>>>
>>> somarLados(1,2,3,4)
10
>>> somarLados(5,1)
6
```

Palavras-chave de argumentos

- Os argumentos podem ser enviados para a função por meio de pares `chave:valor`:

```
>>> def printProdutos(p3, p2, p1):
>>>     print("O valor do parametro p3 é: {}".format(p3))
>>>
>>> printProdutos(p1 = "alface", p2 = "goiaba", p3 = "tomate")
O valor do par metro p3 é tomate
```

Valor padrão de um parâmetro

- Pode-se chamar uma função sem argumento, desde que se defina o valor padrão na assinatura da função:

```
>>> def printProduto(nome = "nenhum produto"):
>>>     print("O seguinte produto foi selecionado: {}".format(nome))
>>>
>>> printProduto()
O seguinte produto foi selecionado: nenhum produto
>>> printProduto("alface")
O seguinte produto foi selecionado: alface
```

Passagem de uma lista como parâmetro

- É possível enviar qualquer tipo de argumento para uma função, incluindo estruturas como listas, dicionários, sets e tuplas.

```
>>> def printList(lista):
>>>     for item in lista:
>>>         print(item)
>>>
>>> cores = ["amarelo", "vermelho", "azul"]
>>> printList(cores)
amarelo
vermelho
azul
```

Retorno de valores

- Uma função pode retornar valores para a instrução que a chamou por meio da instrução `return`:

```
>>> def calculaArea(altura, largura):
>>>     area = altura * largura
>>>     return area
>>>
>>> resultado = calculaArea(5,10)
>>> print(resultado)
50
>>> resultado = calculaArea(3,5)
>>> print(resultado)
15
```

Retorno de mais de um valor

- É possível retornar múltiplos valores no Python utilizando estrutura de dados:

```
>>> def raizesBascara(a, b, c):
>>>     x1 = (-b + math.sqrt(b*b - 4*a*c))/2*a
>>>     x2 = (-b - math.sqrt(b*b - 4*a*c))/2*a
>>>     return (x1, x2)
>>>
>>> resultado = raizesBascara(3,-15,12)
>>> print("X1: {}, X2: {}".format(resultado[0], resultado[1]))
X1: 4, X2: 1
```

Função lambda

- Uma função lambda é uma pequena função anônima que pode ter vários argumentos, porém com uma única expressão.

  Sintaxe: `lambda argumentos : expressão`

```
>>> #Função normal
>>> def square_area(lado):
>>>     return lado * 4
>>>
>>> print(square_area(10))
40
>>> #Função lambda
>>> square_area = lambda lado : lado * 4
>>> print(square_area(10))
40
```

Prós e contras da função lambda

- **Prós**:
  - Boa para operações lógicas que são de fácil entendimento. Isso aprimora a legibilidade do código;
  - Boa para quando a função será utilizada poucas vezes.
- **Contras**:
  - Função lambda pode executar uma única expressão;
  - Ruim para operações que gastam mais de uma linha (ex.: operações condicionais aninhadas);
  - Ruim porque não se pode escrever uma documentação explicando todas as entradas, operações e saídas como faria em una função normal.

Exemplos de uso de função lambda

```
>>> #Valor escalar
>>> valor = lambda y : y * 2
>>> print(valor(10))
>>> 20
```

```
>>> #Lista com maiores de que 18 anos
>>> idade = [5, 8, 10, 15, 20, 30, 40, 50]
>>> maioridade = list(filter(lambda x : x > 18, idade))
>>> print(maioridade)
[20, 30, 40, 50]
```