UFCD 5417: Lab. 6

Funções

v1.1

Nelson Santos
nelson.santos.0001376@edu.atec.pt

ATEC — 25 de novembro de 2024

Índice

1 Introdução

Neste laboratório, vamos explorar uma série de exercícios práticos que envolvem a criação de funções mais avançadas em `Python`. Estes exercícios vão abordar tópicos como cálculo de dígitos de controlo de códigos de barras, entre outros desafios matemáticos e lógicos.

Antes da elaboração dos exercícios, é importante relembrar alguns conceitos básicos sobre funções em `Python`. As funções são blocos reutilizáveis de código que permitem organizar, simplificar e reutilizar partes de um programa. Uma função pode ser definida para realizar uma tarefa específica e ser chamada quantas vezes for necessário.

Cada exercício propõe um problema específico, que será resolvido utilizando funções personalizadas e manipulando estruturas matemáticas. A implementação destes problemas ajudará a consolidar os conhecimentos sobre funções, manipulação de números e operadores matemáticos em `Python`.

Cada um destes exercícios está descrito detalhadamente nas secções seguintes.

2 Funções em `Python`

As funções em `Python` são blocos reutilizáveis de código que permitem organizar, simplificar e reutilizar partes de um programa. Uma função pode ser definida para realizar uma tarefa específica e ser chamada quantas vezes for necessário.

2.1 Definir Funções

Para definir uma função, utiliza-se a palavra-chave `def`, seguida do nome da função, parêntesis (que podem conter argumentos) e dois pontos. O corpo da função é indentado.

```
def greeting():
    print("Welcome!")
```

No exemplo acima, a função `greeting()` imprime uma mensagem. Para a executar, basta chamá-la:

```
greeting()
```

2.2 Funções com argumentos

As funções podem aceitar argumentos, que são valores fornecidos no momento da chamada. Estes argumentos são utilizados dentro do corpo da função.

```
def sum(a, b):
    return a + b
```

A função `sum()` recebe dois argumentos (`a` e `b`) e devolve a soma dos mesmos. Para utilizá-la:

```
result = sum(3, 5)
print(result)  # output: 8
```

2.3 Valores padrão para os argumentos

É possível definir valores padrão para os argumentos, tornando-os opcionais ao chamar a função.

```
def greeting(name="Visitor"):
    print(f"Hello, {name}!")
```

Se nenhum argumento for fornecido, o valor padrão (`"Visitor"`) será utilizado:

```
greeting()  # output: Hello, Visitor!
greeting("Mike")  # output: Hello, Mike!
```

2.4 Funções com nº variável de argumentos

O Python permite criar funções que aceitam um número variável de argumentos, usando o operador `*`.

```
def list_items(*items):
    for item in items:
        print(item)
```

Neste caso, `list_items()` pode receber qualquer número de argumentos:

```
list_items("apple", "banana", "orange")
# output:
# apple
# banana
# orange
```

2.5 Retorno de valores

Uma função pode devolver valores utilizando a instrução `return`. Uma vez executada, a função termina e o valor é devolvido.

```
def pow(x):
    return x ** 2
```

```
result = pow(4)
print(result)  # output: 16
```

2.6 Funções anónimas (*Lambda*)

Em Python, é possível criar funções curtas, sem nome, utilizando a palavra-chave `lambda`. Estas funções são úteis para operações rápidas e temporárias.

```
double = lambda x: x * 2
print(double(5))  # output: 10
```

2.7 Boas práticas para a utilização de funções

- Escolha nomes significativos para as funções e argumentos, refletindo a sua finalidade
- Utilize documentação (`docstrings`) para explicar o propósito e os parâmetros da função
- Mantenha cada função focada numa única responsabilidade, promovendo a clareza e a reutilização

Exemplo de documentação com `docstrings`:

```
def divide(a, b):
    """
    Divide two numbers and returns the result.

    :param a: dividend
    :param b: divisor
    :return: result of the division
    """
    return a / b
```

3 Exercícios

3.1 Exercício 1: Verificação de Número de Cartão de Crédito

Implemente uma função que verifica se um número de cartão de crédito é válido utilizando o algoritmo de Luhn.

3.1.1 Instruções

1. Crie uma função que receba um número de cartão de crédito como parâmetro.
2. A função deve verificar se o número é válido aplicando o algoritmo de `Luhn`, que envolve somar os dígitos do número de forma específica.
3. A função deve retornar `True` se o número for válido, ou `False` caso contrário.

3.1.2 Exemplos de Input e Output

- Input: 4532015112830366
  Output: `True` (cartão válido)
- Input: 1234567812345678
  Output: `False` (cartão inválido)
- Input: 4111111111111111
  Output: `True` (cartão válido)
- Input: 0000000000000000
  Output: `False` (número de cartão inválido)

3.2 Exercício 2: Conversão de Temperaturas

Implemente uma função que converte temperaturas entre Celsius e Fahrenheit.

3.2.1 Instruções

1. Crie uma função que receba uma temperatura e a unidade (Celsius ou Fahrenheit)
2. A função deve converter a temperatura para a outra unidade. Por exemplo, se o input for em Celsius, deve converter para Fahrenheit
3. A função deve retornar a temperatura convertida

3.2.2 Exemplos de Input e Output

- Input: 0°C
  Output: 32°F
- Input: 100°C
  Output: 212°F
- Input: 32°F
  Output: 0°C
- Input: -40°C
  Output: -40°F
- Input: "cinquenta"
  Output: `Erro` (input inválido)

3.3 Exercício 3: Verificação do Dígito de Controlo de um Código de Barras

Implemente uma função que verifique o dígito de controlo de um código de barras, usando o algoritmo `GTIN`.

3.3.1 Instruções

1. Crie uma função que receba um código de barras (número)
2. A função deve calcular o dígito de controlo com base no algoritmo `GTIN`
3. Verifique se o último dígito do código é igual ao dígito de controlo calculado

3.3.2 Exemplos de Input e Output

- Input: 12345670
  Output: `True` (código válido)
- Input: 12345678
  Output: `False` (código inválido)
- Input: 00000000
  Output: `False` (código inválido)

3.4 Exercício 4: Cálculo do IMC (Índice de Massa Corporal)

Implemente uma função que calcula o Índice de Massa Corporal (IMC) de uma pessoa.

3.4.1 Instruções

1. Crie uma função que receba o peso (em quilogramas) e a altura (em metros) de uma pessoa como parâmetros
2. A função deve calcular o IMC utilizando a fórmula:
$$IMC = \frac{peso}{altura^2}$$
3. A função deve retornar o valor do IMC e a sua classificação com base nos intervalos recomendados pela Organização Mundial de Saúde:
   - Abaixo de 18,5: Abaixo do peso
   - Entre 18,5 e 24,9: Peso normal
   - Entre 25 e 29,9: Excesso de peso
   - Entre 30 e 34,9: Obesidade grau 1
   - Entre 35 e 39,9: Obesidade grau 2
   - 40 ou mais: Obesidade grau 3

3.4.2 Exemplos de Input e Output

- Input: peso = 70, altura = 1.75
  Output: IMC = 22.86, Classificação: "Peso normal"
- Input: peso = 50, altura = 1.60
  Output: IMC = 19.53, Classificação: "Peso normal"
- Input: peso = 90, altura = 1.60
  Output: IMC = 35.16, Classificação: "Obesidade grau 2"
- Input: peso = -70, altura = 1.75
  Output: `Erro` (peso inválido)
- Input: peso = 70, altura = 0
  Output: `Erro` (altura inválida)

3.5 Exercício 5: Conversor de Unidades de Medida

Implemente uma função que converte unidades de medida entre quilómetros e milhas.

3.5.1 Instruções

1. Crie uma função que receba uma distância e a unidade (quilómetros ou milhas)
2. A função deve converter a distância para a outra unidade. Por exemplo, se o input for em quilómetros, deve converter para milhas
3. A função deve retornar a distância convertida

3.5.2 Exemplos de Input e Output

- Input: 10 km
  Output: 6.21 milhas
- Input: 5 milhas
  Output: 8.05 km
- Input: -10 km
  Output: `Erro` (distância negativa)
- Input: "dez"km
  Output: `Erro` (input inválido)