Aula 2
05/11/2025

UNIVERSIDADE LUSÓFONA

Fundamentos de Programação (T)

Licenciatura em Videojogos

Ano 1 / Semestre 1

Filipe Pinto
filipe.pinto@ulusofona.pt

UNIVERSIDADE LUSÓFONA

Sumário

- Funções
  - Funcionalidade
  - Sintaxe
  - Invocação
  - Parâmetros
  - Retorno

UNIVERSIDADE LUSÓFONA

Funções

UNIVERSIDADE LUSÓFONA

Funções

- Uma função é um bloco de código que só é executado quando chamado
- A chamada de uma função também se pode chamar invocação
- Invocar uma função = Executar o código da função
- Funções servem para agregar código e funcionalidade com:
    - Melhor leitura
    - Mais modularidade
    - Mais reaproveitamento de código
    - Menos erros

UNIVERSIDADE LUSÓFONA

Funções

- Nós já usámos várias funções (built-in functions):
  - **print**: print("Hello world")
  - **input**: player_input = input()
  - **str**: print("Result = " + str(number))
  - **copy**: new_array = old_array.copy()
  - …

Mas podemos fazer as nossas, o que é mais interessante e vai ter mais utilidade.

Este tipo de função chama-se normalmente um "método"
Vamos ver mais sobre isto quando falarmos de classes

UNIVERSIDADE LUSÓFONA

Funções - Sintaxe

```
def <nome>(<parametros>):

    <bloco de instruções>
```

- Exemplo

```
def say_hello():

    print(“Hello”)
```

UNIVERSIDADE LUSÓFONA

Funções - Sintaxe

```
def say_hello():
    print("Hello")
```

- Não acontece nada porque só definimos a função, temos que chamá-la

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas> & C:/Users/filip/AppData/Local/Programs/Python/Python312/python.exe c:/U
sers/filip/Desktop/Videojogos/2526/FP/Aulas/Aula2_fp.py
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas>
```

UNIVERSIDADE LUSÓFONA

Funções - Invocação

```
def say_hello():
    print("Hello")

say_hello()
```

```
sers/filip/Desktop/Videojogos/2526/FP/Aulas/Aula2_fp.py
Hello
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas>
```

UNIVERSIDADE LUSÓFONA

Funções - Parâmetros

- A função `say_hello()` neste momento não recebe quaisquer parâmetros:
  - Os parentesis `()` encontram-se vazios
  - Significa que a função não usa nenhum valor externo
- E se quisermos adicionar um parâmetro para a função devolver um "Hello [nome]"?

```
def say_hello(name):
        print("Hello " + name + "!")

say_hello("Filipe")
```

```
sers/filip/Desktop/Videojogos/2526/FP/Aulas/Aula2_fp.py
Hello Filipe!
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas>
```

UNIVERSIDADE LUSÓFONA

Funções - Parâmetros

- Vamos criar de raiz uma função que dá o quadrado dos valores:

```
# Invocações de teste

square(5)

square(3)

square(-4)
```

UNIVERSIDADE LUSÓFONA

Funções - Parâmetros

```
def square(value):
    square_value = value * value
    print("Square of " + str(value) +  " is " + str(square_value))


square(5)
square(3)
square(-4)
```

Estes valores da invocação são copiados para a variável *value*

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas>
Square of 5 is 25
Square of 3 is 9
Square of -4 is 16
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas>
```

Funções - Parâmetros

- Podemos também usar variáveis como parâmetros:

```
def square(value):
    value = value * value
    print("Square is " + str(value))


input_value = 5
square(input_value)
print("Input value = " + str(input_value))
```

```
Square is 25
Input value = 5
```

Este como é um valor primitivo, a sua chamada faz uma cópia não modificada do original

UNIVERSIDADE LUSÓFONA

Funções - Parâmetros

- Podemos também usar tipos de dados complexos:

Qualquer tipo de valor pode ser passado a uma função

```
def squares(array_of_values):
    for i in range(0, len(array_of_values)):
        value = array_of_values[i] * array_of_values[i]
        print("Square of " + str(array_of_values[i]) + " is " + str(value))

squares([ 5, 3, -4 ])
```

```
Square of 5 is 25
Square of 3 is 9
Square of -4 is 16
```

UNIVERSIDADE LUSÓFONA

Funções – Parâmetros por referência

- Podemos também usar referências:

```
def squares(list):
    for i in range(0, len(list)):
        value = list[i] * list[i]
        print("Square is " + str(value))

src = [ 5, 3, -4 ]
squares(src)
print(src)
```

```
Square is 25
Square is 9
Square is 16
[5, 3, -4]
```

UNIVERSIDADE LUSÓFONA

Funções – Parâmetros por referência

```
def squares(list):
    for i in range(0, len(list)):
        value = list[i] * list[i]
        print("Square is " + str(value))

src = [ 5, 3, -4 ]
squares(src)
print(src)
```

Listas são passadas por referência, não por valor: são exactamente os mesmos dados

```
Square is 25
Square is 9
Square is 16
[5, 3, -4]
```

Funções – Tipos de Parâmetros

- Primitivos:

```
def square(value):
    value = 4
    value = value * value
    print("Square is " + str(value))

input_value = 5
square(input_value)
print("Input value = " + str(input_value))
```

```
Square is 16
Input value = 5
```

Funções – Tipos de Parâmetros

- Complexos:

```
def squares(list):
    list.append(2)
    for i in range(0, len(list)):
        value = list[i] * list[i]
        print("Square is " + str(value))

src = [ 5, 3, -4 ]
squares(src)
print(src)
```

```
Square is 25
Square is 9
Square is 16
Square is 4
[5, 3, -4, 2]
```

UNIVERSIDADE LUSÓFONA

Funções – Passagem por referência / valor

- Tipos primitivos são passados por valor:
  - Números (inteiros, decimais e complexos)
  - *Strings*
  - Booleanos
  - Tuplos

  Não podem ser alterados pela função

- Tipos complexos são passados por referência:
  - *Arrays*/Listas
  - Dicionários
  - Objectos...

  Podem ser alterados pela função

UNIVERSIDADE LUSÓFONA

Funções – Parâmetros por omissão

```
def say_hello(name):
    print("Hello, " + name + "!")


say_hello("Filipe")
```

```
sers/filip/Desktop/Videojogos/2526/FP/Aulas/Aula2_fp.py
Hello Filipe!
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas>
```

UNIVERSIDADE LUSÓFONA

Funções – Parâmetros por omissão

```
def say_hello(name):
    print("Hello, " + name + "!")


say_hello()
```

```
Traceback (most recent call last):
  File "c:\Users\filip\Desktop\Videojogos\2526\FP\Aulas\Aula2_fp.py", line 149, in <module>
    say_hello()
TypeError: say_hello() missing 1 required positional argument: 'name'
```

UNIVERSIDADE LUSÓFONA

Funções – Parâmetros por omissão

```
def say_hello(name = "somebody"):

    print("Hello, " + name + "!")



say_hello()
```

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas> & C:/Users/filip/Desktop/Videojogos/2526/FP/Aulas/Aula2_fp.py
Hello somebody!
```

UNIVERSIDADE LUSÓFONA

Funções – Parâmetros por omissão

```
def say_hello(count, name = "somebody"):
    for i in range(0, count):
        print("Hello, " + name + "!")


say_hello(2)
```

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas> & C:
sers/filip/Desktop/Videojogos/2526/FP/Aulas/Aula2_fp.py
Hello somebody!
Hello somebody!
```

UNIVERSIDADE LUSÓFONA

Funções – Parâmetros por omissão

```
def say_hello(count, name1 = "some", name2 = "body"):
    for i in range(0, count):
        print("Hello, " + name1 + " " + name2 + "!")


say_hello(2)
```

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas> & C:
sers/filip/Desktop/Videojogos/2526/FP/Aulas/Aula2_fp.py
Hello, some body!
Hello, some body!
```

UNIVERSIDADE LUSÓFONA

Funções – Parâmetros por omissão

```
def say_hello(count, name1 = "some", name2 = "body"):
    for i in range(0, count):
        print("Hello, " + name1 + " " + name2 + "!")


say_hello(2)
```

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas> & C:
sers/filip/Desktop/Videojogos/2526/FP/Aulas/Aula2_fp.py
Hello, some body!
Hello, some body!
```

UNIVERSIDADE LUSÓFONA

Funções – Parâmetros por omissão

```
def say_hello(count, name1 = "some", name2 = "body"):
    for i in range(0, count):
        print("Hello, " + name1 + " " + name2 + "!")


say_hello(2, "no")
```

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas> & C:
sers/filip/Desktop/Videojogos/2526/FP/Aulas/Aula2_fp.py
Hello, no body!
Hello, no body!
```

UNIVERSIDADE LUSÓFONA

Funções – Parâmetros por omissão

```
def say_hello(count, name1 = "some", name2 = "body"):
    for i in range(0, count):
        print("Hello, " + name1 + " " + name2 + "!")


say_hello(2, name2 = "where")
```

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas> & C:
sers/filip/Desktop/Videojogos/2526/FP/Aulas/Aula2_fp.py
Hello, some where!
Hello, some where!
```

UNIVERSIDADE LUSÓFONA

Agora já conseguimos fazer funções…

- Em boa verdade, o que aprendemos a fazer foram procedimentos
- Procedimentos são funções que não devolvem nada, são só conjuntos de instruções
- Em matemática, quando falavam em funções, falavam em algo do género y = f(x)

UNIVERSIDADE LUSÓFONA

Funções - Motivação

Voltando à Aventura de texto

- Neste momento a aplicação verifica se o jogar se pode movimentar em qualquer direcção.
- Estamos a copiar código semelhante em todos os `ifs`/`elifs`
- Fazemos a mesma verificação 4 vezes, uma para cada direcção

```
    # Check direction
    if (room_exits[y][x][NORTH]):
        # Move direction
        print("you move north...")
        y -= 1
    else:
        print("Can't go north!")
elif (command == "south"):
    # Check direction
    if (room_exits[y][x][SOUTH]):
        # Move direction
        print("you move south...")
        y += 1
    else:
        print("Can't go south!")
elif (command == "west"):
    # Check direction
    if (room_exits[y][x][WEST]):
        # Move direction
        print("you move west...")
        x -= 1
    else:
        print("Can't go west!")
elif (command == "east"):
    # Check direction
    if (room_exits[y][x][EAST]):
        # Move direction
        print("you move east...")
        x += 1
    else:
        print("Can't go east!")
```

Funções - Motivação

- Estamos constantemente a fazer a mesma verificação para todas as direcções,
- Isto é má prática porque:
  - É muito propenso a erros
  - Acrescentar funcionalidades implica colocar esse código em todos os blocos
  - Se encontrarmos um erro, temos de corrigi-lo em todo o lado
  - Torna-se mais díficil ler o código, principalmente em trabalho de equipa
  - Há coisas virtualmente impossíveis de fazer desta forma

Funções - Motivação

- Para resolver isto podemos usar funções
- O uso de funções permite-nos aperfeiçoar e optimizar o código
- Precisamos de um função que:
    1. Verifica a direcção que o jogador escolheu
    2. Verifica se pode mover-se nessa direcção
    3. Move o jogador nessa direcção, se puder

UNIVERSIDADE LUSÓFONA

Funções - Motivação

```
def move_player(direction_text, direction_index, y_inc, x_inc):
    global position

    x, y = position
    if command == direction_text:
        if room_exits[y][x][direction_index]:
            y += y_inc
            x += x_inc
            position = (x, y)
            print("You move " + direction_text + " to room "  + room_descriptions[y][x])
        else:
            print("You can't move " + direction_text + "!")

        return True

    return False

# Ex.: move_player(“north”, NORTH, -1, 0), isto retorna True se o comando foi ”north”, e move o jogador se puder. Se não, retorna False.
```

Esta claúsula está relacionada com o *scope* da variável: basicamente é para podermos usar variáveis externas à função e modificá-las, mantendo-se as modificações fora da função

Ex.: if (command == “north”)

Ex.: if (room_exits[y][x][0])

y = y – 1
x = x + 0

Funções - Motivação

```
    # Check direction
    if (room_exits[y][x][NORTH]):
        # Move direction
        print("you move north...")
        y -= 1
    else:
        print("Can't go north!")
elif (command == "south"):
    # Check direction
    if (room_exits[y][x][SOUTH]):
        # Move direction
        print("you move south...")
        y += 1
    else:
        print("Can't go south!")
elif (command == "west"):
    # Check direction
    if (room_exits[y][x][WEST]):
        # Move direction
        print("you move west...")
        x -= 1
    else:
        print("Can't go west!")
elif (command == "east"):
    # Check direction
    if (room_exits[y][x][EAST]):
        # Move direction
        print("you move east...")
        x += 1
    else:
        print("Can't go east!")
```

vs.

```
if command == "exit":
    print("Exiting game...")
    break
elif move_player("north", NORTH, -1, 0):
    pass
elif move_player("south", SOUTH, 1, 0):
    pass
elif move_player("east", EAST, 0, 1):
    pass
elif move_player("west", WEST, 0, -1):
    pass
else:
    print("I don't understand " + command + "!")
```

O código ficou muito mais simples e legível

UNIVERSIDADE LUSÓFONA

Funções - Motivação

- Como exemplo, agora como fariamos se quisessemos mostrar apenas a descrição da sala?
- Dantes teríamos que mudar os blocos todos de movimento, agora só precisamos de modificar a função

```
def move_player(direction_text, direction_index, y_inc, x_inc):
    global position
    global room_desc
    global room_exits


    x, y = position
    if command == direction_text:
        if room_exits[y][x][direction_index]:
            print("\nYou move " + direction_text)
            y += y_inc
            x += x_inc
            position = (x, y)
            print("\nRoom description: "  + room_descriptions[y][x] + "\n")
        else:
            print("You can't move " + direction_text + "!")
        return True
    return False
```

E já temos a aplicação a usar a funcionar com as modificações

```
(2, 1) : H
What now?
Choose the direction you want to go north, south, east, west) or exit
south

You move south

Room description: M

(2, 2) : M
What now?
Choose the direction you want to go north, south, east, west) or exit
east
You can't move east!
(2, 2) : M
What now?
Choose the direction you want to go north, south, east, west) or exit
west
You can't move west!
(2, 2) : M
What now?
Choose the direction you want to go north, south, east, west) or exit
south

You move south

Room description: R

(2, 3) : R
What now?
Choose the direction you want to go north, south, east, west) or exit
east

You move east

Room description: S

(3, 3) : S
What now?
Choose the direction you want to go north, south, east, west) or exit
```

UNIVERSIDADE LUSÓFONA

Funções - Retorno

```
# Para retornar valores, usamos a clausula return


return <valor>
```

UNIVERSIDADE LUSÓFONA

Funções - Retorno

- Voltando ao exemplo dos quadrados, tinhamos

```
def square(value):
    value = value * value
    print(value)
square(5)
```

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas>
p/Videojogos/2526/FP/Aulas/Aula2_fp.py
25
```

- Podemos simplificar a função definindo um retorno:

```
def square(value):
    return value * value
print(square(5))
```

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas>
p/Videojogos/2526/FP/Aulas/Aula2_fp.py
25
```

UNIVERSIDADE LUSÓFONA

Funções - Retorno

- **Podemos devolver qualquer tipo de dados:**

```
def squares(value):
    ret = []
    for i in range(0, value):
        ret.append(i * i)
    return ret


print(squares(5))
```

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas>
p/Videojogos/2526/FP/Aulas/Aula2_fp.py
[0, 1, 4, 9, 16]
```

UNIVERSIDADE LUSÓFONA

Funções - Multiplo Retorno

- **Podemos devolver também vários valores:**

```
def squares(value):
    ret = []
    for i in range(0, value):
        ret.append(i * i)
    return value * value, ret

sq, array_of_values = squares(5)
print(sq)
print(array_of_values)
```

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas>
p/Videojogos/2526/FP/Aulas/Aula2_fp.py
25
[0, 1, 4, 9, 16]
```

Estamos a fazer um assign de uma variável a cada um dos returns de squares()

UNIVERSIDADE LUSÓFONA

Funções - Multiplo Retorno

- **Mas cuidado**, quando não há variáveis suficientes para receber o *Return*, é criado um **tuplo** com os valores:

```
def squares(value):
    ret = []
    for i in range(0, value):
        ret.append(i * i)
    return value * value, ret

sq = squares(5)
print(sq)
```

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas>
p/Videojogos/2526/FP/Aulas/Aula2_fp.py
(25, [0, 1, 4, 9, 16])
```

UNIVERSIDADE LUSÓFONA

Funções Recursivas

- Uma função recursiva é uma função que se chama a si própria
- Um exemplo clássico é o factorial:

$$n! = n * (n-1) * (n-2) * \cdots * 1$$

- Mas o mais correto seria algo deste género:

$$n! = \begin{cases} 1, & se\ n = 1 \\ n * (n-1)!, & se\ n > 1 \end{cases}$$

UNIVERSIDADE LUSÓFONA

Funções Recursivas

- Como podemos definir isto em Python:
- Se vissemos um factorial como:

$$n! = n * (n-1) * (n-2) * \cdots * 1$$

- Então o código seria:

```
def factorial(n):
    r = n * factorial(n-1)
    return r

print(factorial(5))
```

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas> & C:/Users/filip/AppData/Local/Programs/Py
p/Videojogos/2526/FP/Aulas/Aula2_fp.py
Traceback (most recent call last):
  File "c:\Users\filip\Desktop\Videojogos\2526\FP\Aulas\Aula2_fp.py", line 194, in <module>
    print(factorial(5))
          ^^^^^^^^^^^^
  File "c:\Users\filip\Desktop\Videojogos\2526\FP\Aulas\Aula2_fp.py", line 191, in factorial
    r = n * factorial(n-1)
            ^^^^^^^^^^^^^^
  File "c:\Users\filip\Desktop\Videojogos\2526\FP\Aulas\Aula2_fp.py", line 191, in factorial
    r = n * factorial(n-1)
            ^^^^^^^^^^^^^^
  File "c:\Users\filip\Desktop\Videojogos\2526\FP\Aulas\Aula2_fp.py", line 191, in factorial
    r = n * factorial(n-1)
            ^^^^^^^^^^^^^^
  [Previous line repeated 996 more times]
RecursionError: maximum recursion depth exceeded
```

UNIVERSIDADE LUSÓFONA

Funções Recursivas

- Precisamos de uma condição de saída
- Tendo em consideração a forma mais correcta de representar um factorial:

$$n! = \begin{cases} 1, & se\ n = 1 \\ n * (n-1)!, & se\ n > 1 \end{cases}$$

- Temos a nossa condição de saída em `n == 1`:

```
def factorial(n):
    if (n == 1):
        return 1
    r = n * factorial(n-1)
    return r
print(factorial(5))
```

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas>
p/Videojogos/2526/FP/Aulas/Aula2_fp.py
120
```

Funções Recursivas

- *"Maximum recursion depth exceeded"*
- Este erro é normalmente chamado *"stack overflow"*
- O *stack* é onde é armazenado o estado local de uma chamada de uma função.
- Isto é, se uma função tem uma variável xpto e gualter, cada chamada a essa função vai criar no *stack* espaço para armazenar essas variáveis.
- Quando se excede o espaço de armazenamento do *stack*, dá-se o *stack overflow*
- Normalmente quer dizer que não fizemos a nossa função bem e falta-lhe o critério de saída ou condição de terminação
- Mesmo que tivéssemos memória infinita, normalmente é sinal que a função está presa na sua recursão

UNIVERSIDADE LUSÓFONA

Exercícios

UNIVERSIDADE LUSÓFONA

Exercícios

1. Objetivo: criar uma calculadora com funções para cada operação.
   - Funções para as operações e cálculos

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas> & C:/U
p/Videojogos/2526/FP/Aulas/exerc_sol.py
Enter first number: 123
Enter second number: 5
Enter operation (add, subtract, multiply, divide): divide
Result = 24.6
```

No final, pede ao utilizador dois números e a operação e imprime o resultado

---

2. Objetivo: criar um mini-mapa interactivo (3x3) combinando funções, parâmetros, retorno e estruturas de dados.
   Exemplo:

```
rooms =         [
                ["A", "B", "C"],
                ["D", "E", "F"],
                ["G", "H", "I"]
                ]
position = (1, 1)
```

   - A aplicação deve permitir o movimento do utilizador, usando os limites da matriz para limitar o movimento

```
You are in room E
Choose direction (north, south, east, west) or 'exit': north

You are in room B
Choose direction (north, south, east, west) or 'exit': north
You can't move north!

You are in room B
Choose direction (north, south, east, west) or 'exit': west

You are in room A
Choose direction (north, south, east, west) or 'exit': west
You can't move west!

You are in room A
Choose direction (north, south, east, west) or 'exit': exit
Game over.
```