Aula 3
10/11/2025

UNIVERSIDADE LUSÓFONA

Fundamentos de Programação (T)

Licenciatura em Videojogos

Ano 1 / Semestre 1

Filipe Pinto
filipe.pinto@ulusofona.pt

UNIVERSIDADE LUSÓFONA

Sumário

- Scope
  - Global
  - Local
  - Nonlocal
- Strings

Scope

UNIVERSIDADE LUSÓFONA

Scope

- Scope define uma area no código onde um identificador pode ser acedido de forma não-ambigua.
- O Scope refere-se ao contexto em que uma variável é acessivel.
- Scope de um identificador é o bloco ou blocos onde esse identificador é válido.

Scope

- Pegando nesta função:

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

print(squares(5))
```

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas>
p/Videojogos/2526/FP/Aulas/Aula3.py
(25, [0, 1, 4, 9, 16])
```

UNIVERSIDADE LUSÓFONA

Scope

- E se tentarmos visualizar o valor das variáveis dentro da função?

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

print(squares(5))
```

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas>
p/Videojogos/2526/FP/Aulas/Aula3.py
16
4
(25, [0, 1, 4, 9, 16])
```

UNIVERSIDADE LUSÓFONA

Scope

- Scope de um identificador é o bloco ou blocos onde esse identificador é válido.
- Uma variável definida no interior de uma função está visível dentro dessa função.

Scope

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

print(squares(5))
```

UNIVERSIDADE LUSÓFONA

Scope

- E se colocarmos a visualização das variáveis fora da função?

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

print(squares(5))
print(square_value)
print(i)
```

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas> & C:/Users/filip/AppData/Local/Progr
p/Videojogos/2526/FP/Aulas/Aula3.py
(25, [0, 1, 4, 9, 16])
Traceback (most recent call last):
  File "c:\Users\filip\Desktop\Videojogos\2526\FP\Aulas\Aula3.py", line 10, in <module>
    print(square_value)
          ^^^^^^^^^^^^
NameError: name 'square_value' is not defined
```

UNIVERSIDADE LUSÓFONA

Scope

- Scope de um identificador é o bloco ou blocos onde esse identificador é válido.
- Uma variável definida no interior de uma função está visível dentro dessa função.
- Se uma variável for definida dentro de uma função, esta não estará visível fora dessa mesma função

UNIVERSIDADE LUSÓFONA

Scope

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

print(squares(5))
print(square_value)
print(i)
```

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas> & C:/Users/filip/AppData/Local/Progr
p/Videojogos/2526/FP/Aulas/Aula3.py
(25, [0, 1, 4, 9, 16])
Traceback (most recent call last):
  File "c:\Users\filip\Desktop\Videojogos\2526\FP\Aulas\Aula3.py", line 10, in <module>
    print(square_value)
          ^^^^^^^^^^^^
NameError: name 'square_value' is not defined
```

Scope

- E se inicializarmos a variável que vai ser usada na função

```
square_value = 0

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

print(squares(5))
print(square_value)
```

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas> &
p/Videojogos/2526/FP/Aulas/Aula3.py
16
(25, [0, 1, 4, 9, 16])
0
```

UNIVERSIDADE LUSÓFONA

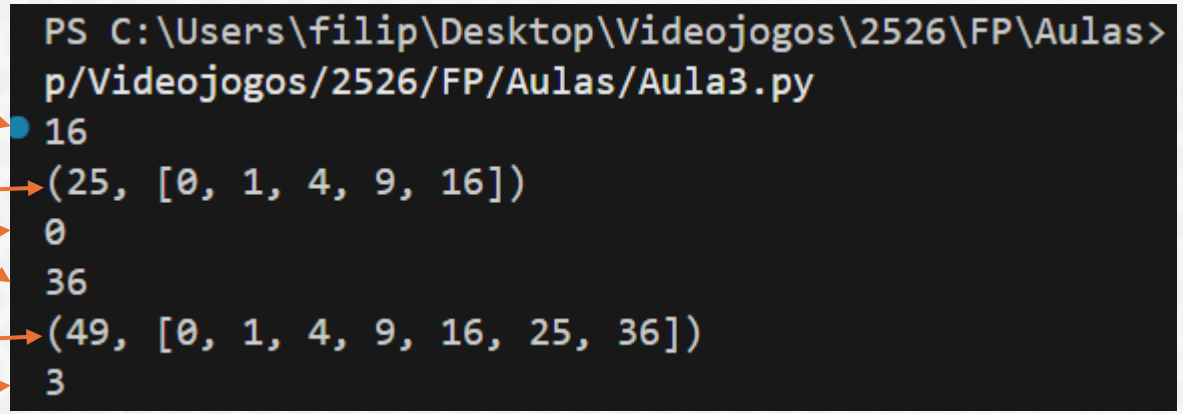
Scope

- Scope de um identificador é o bloco ou blocos onde esse identificador é válido.
- Uma variável definida no interior de uma função está visível dentro dessa função.
- Se uma variável for definida dentro de uma função, esta não estará visível fora dessa mesma função
- Uma variável definida no exterior de uma função não é afectada pelo interior de uma função

Scope

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

square_value = 0
print(squares(5))
print(square_value)
square_value = 3
print(squares(7))
print(square_value)
```



UNIVERSIDADE LUSÓFONA

Scope

- Scope de um identificador é o bloco ou blocos onde esse identificador é válido.
- Uma variável definida no interior de uma função está visível dentro dessa função.
- Se uma variável for definida dentro de uma função, esta não estará visível fora dessa mesma função
- Uma variável definida no exterior de uma função não é afectada pelo interior de uma função
- O scope é dinâmico, só interessa no momento em que o código é executado

Scope

```
def squares(value):
    ret = []
    square_value = 999
    for i in range(value):
        square_value = i * i
        ret.append(square_value)

    print(square_value)

    return value * value, ret


print(squares(5))
```

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas>
p/Videojogos/2526/FP/Aulas/Aula3.py
16
(25, [0, 1, 4, 9, 16])
```

UNIVERSIDADE LUSÓFONA

Scope

- Scope de um identificador é o bloco ou blocos onde esse identificador é válido.
- Uma variável definida no interior de uma função está visível dentro dessa função.
- Se uma variável for definida dentro de uma função, esta não estará visível fora dessa mesma função
- Uma variável definida no exterior de uma função não é afectada pelo interior de uma função
- O scope é dinâmico, só interessa no momento em que o código é executado
- O scope não está associado ao nível de indentação

UNIVERSIDADE LUSÓFONA

Scope

- Questão?

De que tipo de Scope estivemos a falar até agora?

Local, é o tipo de Scope *by default*.

Outros tipos de Scope têm que ser declarados

Scope - Tipos

global e nonlocal test

```
def scope_test():
    def do_local():
        spam = "local spam"

    def do_nonlocal():
        nonlocal spam
        spam = "nonlocal spam"

    def do_global():
        global spam
        spam = "global spam"

    spam = "test spam"
    do_local()
    print("After local assignment:", spam)
    do_nonlocal()
    print("After nonlocal assignment:", spam)
    do_global()
    print("After global assignment:", spam)

scope_test()
print("In global scope:", spam)
```

Scope - Tipos

Em Python, é possível ter funções locais, isto é, funções que são definidas dentro de outras funções

```
def scope_test():
    def do_local():
        spam = "local spam"

    def do_nonlocal():
        nonlocal spam
        spam = "nonlocal spam"

    def do_global():
        global spam
        spam = "global spam"

    spam = "test spam"
    do_local()
    print("After local assignment:", spam)
    do_nonlocal()
    print("After nonlocal assignment:", spam)
    do_global()
    print("After global assignment:", spam)

scope_test()
print("In global scope:", spam)
```

Scope - Tipos

Se não colocarmos nenhuma indicação, a variável é definida como *local*, por defeito

```
After local assignment: test spam
```

```
def do_local():
    spam = "local spam"
    print("inside local function:", spam)
```

```
inside local function: local spam
After local assignment: test spam
```

```
def scope_test():
    def do_local():
        spam = "local spam"

    def do_nonlocal():
        nonlocal spam
        spam = "nonlocal spam"

    def do_global():
        global spam
        spam = "global spam"

    spam = "test spam"
    do_local()
    print("After local assignment:", spam)
    do_nonlocal()
    print("After nonlocal assignment:", spam)
    do_global()
    print("After global assignment:", spam)

scope_test()
print("In global scope:", spam)
```

Scope - Tipos

nonlocal diz ao interpretador que esta variável afecta o scope imediatamente acima.

```
After local assignment: test spam
After nonlocal assignment: nonlocal spam
```

```python
def scope_test():
    def do_local():
        spam = "local spam"

    def do_nonlocal():
        nonlocal spam
        spam = "nonlocal spam"

    def do_global():
        global spam
        spam = "global spam"

    spam = "test spam"
    do_local()
    print("After local assignment:", spam)
    do_nonlocal()
    print("After nonlocal assignment:", spam)
    do_global()
    print("After global assignment:", spam)

scope_test()
print("In global scope:", spam)
```

Scope - Tipos

nonlocal diz ao interpretador que esta variável afecta o scope imediatamente acima.

```
After local assignment: test spam
After nonlocal assignment: nonlocal spam
```

```python
def scope_test():
    def do_local():
        spam = "local spam"

    def do_nonlocal():
        nonlocal spam
        spam = "nonlocal spam"

    def do_global():
        global spam
        spam = "global spam"

    spam = "test spam"
    do_local()
    print("After local assignment:", spam)
    do_nonlocal()
    print("After nonlocal assignment:", spam)
    do_global()
    print("After global assignment:", spam)

scope_test()
print("In global scope:", spam)
```

Scope - Tipos

global diz ao interpretador que esta variável é global, isto é pertence ao scope mais acima de todos neste código

```
After local assignment: test spam
After nonlocal assignment: nonlocal spam
After global assignment: nonlocal spam
```

```
def scope_test():
    def do_local():
        spam = "local spam"

    def do_nonlocal():
        nonlocal spam
        spam = "nonlocal spam"

    def do_global():
        global spam
        spam = "global spam"

    spam = "test spam"
    do_local()
    print("After local assignment:", spam)
    do_nonlocal()
    print("After nonlocal assignment:", spam)
    do_global()
    print("After global assignment:", spam)

scope_test()
print("In global scope:", spam)
```

Scope - Tipos

global diz ao interpretador que esta variável é global, isto é pertence ao scope mais acima de todos neste código

```
After local assignment: test spam
After nonlocal assignment: nonlocal spam
After global assignment: nonlocal spam
In global scope: global spam
```

```
def scope_test():
    def do_local():
        spam = "local spam"

    def do_nonlocal():
        nonlocal spam
        spam = "nonlocal spam"

    def do_global():
        global spam
        spam = "global spam"

    spam = "test spam"
    do_local()
    print("After local assignment:", spam)
    do_nonlocal()
    print("After nonlocal assignment:", spam)
    do_global()
    print("After global assignment:", spam)

scope_test()
print("In global scope:", spam)
```

Voltando à função da última aula

Este *global* permite-nos aceder à variável global da posição do jogador, não só aceder mas modificá-la para ser vista no resto do código

```
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]:
            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
```

Voltando à função da última aula

Se não estivesse cá, as variáveis seriam desconhecidas e o resultado era este

```
(2, 2) : M
What now?
Choose the direction you want to go north, south, east, west) or exit
north
Traceback (most recent call last):
  File "c:\Users\filip\Desktop\Videojogos\2526\FP\Aulas\text_adventure.py", line 52, in <module>
    elif move_player("north", NORTH, -1, 0):
         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "c:\Users\filip\Desktop\Videojogos\2526\FP\Aulas\text_adventure.py", line 28, in move_player
    x, y = position
           ^^^^^^^^
UnboundLocalError: cannot access local variable 'position' where it is not associated with a value
```

```python
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]:
            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 + "!")
        turn True
        False
```

UNIVERSIDADE LUSÓFONA

Exercícios

UNIVERSIDADE LUSÓFONA

Exercícios

1.

```
x = 10

def test():
    x = 5
    print("Inside function:", x)

test()
print("Outside function:", x)
```

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas> &
ideojogos/2526/FP/Aulas/Aula3.py
Inside function: 5
Outside function: 10
```

UNIVERSIDADE LUSÓFONA

Exercícios

2.

```
count = 0

def outer():
    count = 10
    def inner():
        nonlocal count
        count += 1
        print("Inner:", count)
    inner()
    print("Outer:", count)

outer()
print("Result:", count)
```

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas>
p/Videojogos/2526/FP/Aulas/Aula3.py
Inner: 11
Outer: 11
Result: 0
```

Substituindo nonlocal por global →

```
Inner: 1
Outer: 10
Result: 1
```

UNIVERSIDADE LUSÓFONA

Exercícios

3.

```
x = 100

def update():
    global x
    x += 50
    print("Inside function:", x)

update()
print("Outside function:", x)
```

```
Inside function: 150
Outside function: 150
```

Exercícios

4.

```
value = 5

def test():
    value = 10
    def inner():
        value = 20
        print("Inner:", value)
    print("Outer1:", value)
    inner()
    print("Outer2:", value)

test()
print("Result:", value)
```

```
Outer1: 10
Inner: 20
Outer2: 10
Result: 5
```

Exercícios

5.

```
counter = 0

def outer():
    count = 10
    def inner():
        global counter
        nonlocal count
        counter += 1
        count += 1
        print("Inner -> counter:", counter, "| count:", count)
    inner()
    print("Outer -> counter:", counter, "| count:", count)

outer()
print("Result -> counter:", counter)
```

```
Inner -> counter: 1 | count: 11
Outer -> counter: 1 | count: 11
Result -> counter: 1
```

Strings

UNIVERSIDADE LUSÓFONA

Strings

- Nós já vimos strings anteriormente
- São um tipo de dados primitivo

```
welcome = “Hello Filipe, welcome!”

print(welcome)
```

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas>
p/Videojogos/2526/FP/Aulas/Aula3.py
Hello Filipe, welcome!
```

UNIVERSIDADE LUSÓFONA

Strings

- Podemos usar ‘ ou “ para definir uma string:

```
welcome = ‘Hello Filipe, welcome!’

print(welcome)
```

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas>
p/Videojogos/2526/FP/Aulas/Aula3.py
Hello Filipe, welcome!
```

Strings

- E se quiséssemos escrever: “Yes!”, it isn’t what he said.?

```
string = '"Yes!", it isn't what he said.'

print(string)
```

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas> & C:/Users/filip/AppData/
p/Videojogos/2526/FP/Aulas/Aula3.py
  File "c:\Users\filip\Desktop\Videojogos\2526\FP\Aulas\Aula3.py", line 138
    string = '"Yes!", it isn't what he said.'
                                            ^
SyntaxError: unterminated string literal (detected at line 138)
```

Strings

- Temos de usar um código de escape:

```
string = '"Yes!", it isn\'t what he said.'

print(string)
```

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas>
p/Videojogos/2526/FP/Aulas/Aula3.py
"Yes!", it isn't what he said.
```

UNIVERSIDADE LUSÓFONA

Strings

- Outro código de escape útil é o \n:
- Já o vimos noutras aulas

```
string = '"Yes!",\n it isn\'t what he said.'

print(string)
```

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas>
p/Videojogos/2526/FP/Aulas/Aula3.py
"Yes!",
 it isn't what he said.
```

UNIVERSIDADE LUSÓFONA

Strings

- Strings podem ser tratadas (praticamente) como listas:

```
string = '"Yes!",\n it isn\'t what he said.'

print(string[1:5])
```

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas>
p/Videojogos/2526/FP/Aulas/Aula3.py
Yes!
```

UNIVERSIDADE LUSÓFONA

Strings

- Apesar de poderem ser tratadas como arrays, continuam a ser um tipo primitivo

```
string1 = "abcdefg"

string2 =  string1

string1 = string1 * 2

print(string1)

print(string2)
```

Multiplicar uma *string* por um inteiro resulta na *string* N vezes

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas>
p/Videojogos/2526/FP/Aulas/Aula3.py
abcdefgabcdefg
abcdefg
```

- Logo, um *assign* resulta na cópia e não na referência à string

UNIVERSIDADE LUSÓFONA

Strings

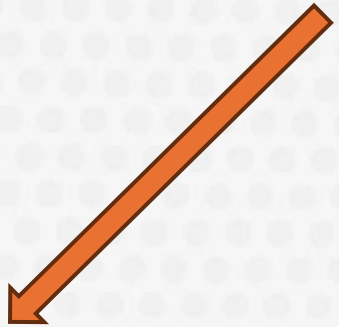
- Alteração de elementos

```
string1 = "abcdefg"

string1[2] = "C"

print(string1)
```

Ao contrário das listas,
não se pode alterar valores de strings



```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas> & C:/Users/filip/AppData/Local/Progra
p/Videojogos/2526/FP/Aulas/Aula3.py
Traceback (most recent call last):
  File "c:\Users\filip\Desktop\Videojogos\2526\FP\Aulas\Aula3.py", line 150, in <module>
    string1[2] = "C"
    ~~~~~~~^^^
TypeError: 'str' object does not support item assignment
```

UNIVERSIDADE LUSÓFONA

Strings

- Alteração de elementos

```
string = "\"Yes!\", he said."
print(string)
print(string.replace("he", "she"))
print(string.upper())
print(string.lower())
print(string.index("he"))
print(string.count("e"))
print(string.split(" "))
```

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas>
p/Videojogos/2526/FP/Aulas/Aula3.py
"Yes!", he said.
"Yes!", she said.
"YES!", HE SAID.
"yes!", he said.
8
2
['"Yes!",', 'he', 'said.']
```

UNIVERSIDADE LUSÓFONA

Voltando à aventura de texto

- Podemos usar alguns métodos que vimos hoje para limpar o input()

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas> & C:/Users/filip/App
p/Videojogos/2526/FP/Aulas/text_adventure.py
(2, 2) : M
What now?
Choose the direction you want to go north, south, east, west) or exit

I don't understand !
(2, 2) : M
What now?
Choose the direction you want to go north, south, east, west) or exit
```

Limpar o input

- Podemos começar por trabalhar no while

```
command = “”
...
while command =! “exit”
...

    if (command == “exit”):
        print("Exiting game...")
        continue
```

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas> & C:/Users/filip/App
p/Videojogos/2526/FP/Aulas/text_adventure.py
(2, 2) : M
What now?
Choose the direction you want to go north, south, east, west) or exit
north

You move north

Room description: H

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

You move east

Room description: I

(3, 1) : I
What now?
Choose the direction you want to go north, south, east, west) or exit
exit
Exiting game...
```

UNIVERSIDADE LUSÓFONA

Limpar o input

- Neste momento se não colocarmos nada no input ele reconhece como uma string vazia
- Se colocarmos um espaço antes de um comando conhecido ele também não reconhece

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas> & C:/Users/filip/Ap
p/Videojogos/2526/FP/Aulas/text_adventure.py
(2, 2) : M
What now?
Choose the direction you want to go north, south, east, west) or exit

I don't understand        !
(2, 2) : M
What now?
Choose the direction you want to go north, south, east, west) or exit
   north
I don't understand    north!
(2, 2) : M
What now?
Choose the direction you want to go north, south, east, west) or exit
```

Limpar o input

- Idealmente, se tivermos uma string vazia deveriamos reiniciar o loop em vez de dar feedback
- Esta seria um opção:

```
command = ""
while (command != "exit"):
    print("What now?")
    command = input(“…”)
    if (command != ""):
        …
```

Desta forma teríamos que indentar o código todo e colocar as outras condições em `else:`

Limpar o input

```
command = ""
while (command != "exit"):
    print("What now?")
    command = input("…")
    if (command == ""):
        continue
```

UNIVERSIDADE LUSÓFONA

Cláusula *continue*

- Uma cláusula *continue* volta ao princípio do bloco de código do ciclo que o contém.
- No caso de um ciclo for, avança uma posição também

Limpar o input

```
command = ""
while (command != "exit"):
    print("What now?")
    command = input("…")
    if (command == ""):
        continue
```

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas> & C:/Users/filip/App
p/Videojogos/2526/FP/Aulas/text_adventure.py
(2, 2) : M
What now?
Choose the direction you want to go north, south, east, west) or exit

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

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

Limpar o input

Mas podemos continuar a usar espaços para dar a volta

```
command = ""

while (command != "exit"):
    print("What now?")
    command = input("…")
    if (command == ""):
        continue
```

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

I don't understand  !
(2, 2) : M
What now?
Choose the direction you want to go north, south, east, west) or exit
```

Limpar o input

Strings têm o método `string.strip()` que podemos usar aqui:

```
command = ""

while (command != "exit"):
    print("What now?")
    command = input("…")
    command = command.strip()
    if (command == ""):
        continue
```

`string.strip()` remove os caracters em branco do inicio e do fim da *string*, preservando os caracteres em branco no meio da *string*

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

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

You move north

Room description: H
```

Limpar o input

Para fechar a limpeza do input:

```
command = ""

while (command != "exit"):
    print("What now?")
    command = input("…").lower()
    command = command.strip()
    if (command == ""):
        continue
```

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas> & C:/Users/filip/App/Videojogos/2526/FP/Aulas/text_adventure.py
(2, 2) : M
What now?
Choose the direction you want to go north, south, east, west) or exit
NORTH

You move north

Room description: H

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

You move north

Room description: C
```

UNIVERSIDADE LUSÓFONA

Exercícios

UNIVERSIDADE LUSÓFONA

Exercícios

```
Original string: The enemy is strong!

New string: The friend is strong!
```

1. Modificar string
2. Cria uma variável `map_description` com o texto `"A dark cave with a small torch on the wall"`.

   Escreve um programa que:

   1. Conta quantas vezes aparece a letra "a"
   2. Imprime o resultado.
   3. Mostra todas as palavras da descrição numa lista.
3. Cria uma lista com três nomes de NPCs ([“maul", “plagueis", “bane"]).

   Escreve um programa que:

   1. Escreve cada nome com a primeira letra maíuscula
   2. Imprime `"Darth [Nome]"` para cada um.
4. Cria um mini-sistema de codificação que:

   1. Pede ao jogador uma palavra
   2. Substitui todas as vogais por `"*"`
   3. Imprime a versão codificada.