Aula 1
03/11/2025

UNIVERSIDADE LUSÓFONA

Fundamentos de Programação (T)

Licenciatura em Videojogos

Ano 1 / Semestre 1

Filipe Pinto
filipe.pinto@ulusofona.pt

UNIVERSIDADE LUSÓFONA

Regras

- Tolerância: 10 minutos (Directiva do curso)
- Presenças: Minimo 75% (não trabalhador-estudante)
- Proíbido o uso de quaisquer ***LLMs/AIs*** em momentos de avaliação

UNIVERSIDADE LUSÓFONA

Avaliação

- Teórica 50% Prática 50%
- Nota mínima para aprovação – 9,5 valores em ambas (T e P)
- Avaliação continua:
  - 1 Teste Teórico Final => **penúltima semana de aulas -> Data a designar (10 ou 11/12)**
  - Assiduidade e participação
  - **A última semana de aulas será para a avaliação da componente prática -> *Live Coding***
- Em caso de reprovação, exame respectivo na componente em que se reprovou
  - A avaliação de cada componente é definida pelo Professor

UNIVERSIDADE LUSÓFONA

Sumário

- Tuplos
- Listas de listas

UNIVERSIDADE LUSÓFONA

Tipos de dados complexos

- Array/Listas são tipos de dados complexo, especificamente de Coleção

- **Existem outros:**

    - Um deles é o Tuplo

UNIVERSIDADE LUSÓFONA

Tuplos

- **Tuplos são uma coleção de valores**
  - Esses valores são ordenados e imutáveis
  - Comportam-se de forma a semelhante a Arrays noutras linguagens de programação
  - Embora possam conter dados de diferentes tipos
- **Sintaxe:**

```
variable_name = (value1, value2,....)
```

Tuplos

UNIVERSIDADE LUSÓFONA

Tuplos

- **Exemplos**

```
# 2D coordinates x, y
position = (22, 45)
print(position)

# 3D coordinates x, y, z
position = (22, 45, 12)
print(position)

# Test grades
grades = (13, 14, 6, 10, 16, 11, 19, 12, 7)
print(grades)
```

```
(22, 45)
```

```
(22, 45, 12)
```

```
(13, 14, 6, 10, 16, 11, 19, 12, 7)
```

UNIVERSIDADE LUSÓFONA

Tuplos

- Os elementos de um tuplo não precisam de ser do mesmo tipo

```
# Ficha de aluno: nome, idade, ocupação, altura, inscrição regularizada?

id_student = ("name", 20, "Estudante", 1.85, True)

print("Ficha de aluno:")

print(id_student)
```

```
Ficha de aluno:
('name', 20, 'Estudante', 1.85, True)
```

Tuplos

- Podemos também aceder a tuplos usando o índice

```
id_student = ("name", 20, "Estudante", 1.85, True)

print("Ficha de aluno:")

print(id_student)

print(id_student[3])
```

```
Ficha de aluno:
('name', 20, 'Estudante', 1.85, True)
1.85
```

Se tentarmos aceder a um indice que não existe, dá erro.

UNIVERSIDADE LUSÓFONA

Tuplos

- Ao contrário dos Arrays, Tuplos são imutáveis

```
id_student = ("name", 20, "Estudante", 1.85, True)

id_student[0] = "Smeagol"

print("Ficha de aluno:")

print(id_student)
```

```
ta/Local/Programs/Python/Python313/python.exe  c:/Users/filip/OneDrive/Ambiente de Trabalho/Aulas/IC/252
6/Aula9_exerc_Python/aula1_fp.py"
Ficha de aluno:
Traceback (most recent call last):
  File "c:\Users\filip\OneDrive\Ambiente de Trabalho\Aulas\IC\2526\Aula9_exerc_Python\aula1_fp.py", line
 24, in <module>
    id_student[0] = "Smeagol"
    ~~~~~~~~~~^^^
TypeError: 'tuple' object does not support item assignment
```

UNIVERSIDADE LUSÓFONA

Tuplos – Atribuição de variáveis

- Podemos usar um tuplo para rapidamente atribuirmos valores a variáveis

```
id_student = ("name", 20, "Estudante", 1.85, True)

print("Ficha de aluno:")

name, age, job, height, enrol = id_student

check_id = [name, age, job, height, enrol]

print(check_id)
```

```
Ficha de aluno:
['name', 20, 'Estudante', 1.85, True]
```

Expansão de Tuplos

- Não podemos modificar os tuplos, mas podemos manipulá-los?

Exemplo:

- Pegar num posição de um mapa num coordenada x, y e avançar para uma nova posição no mapa

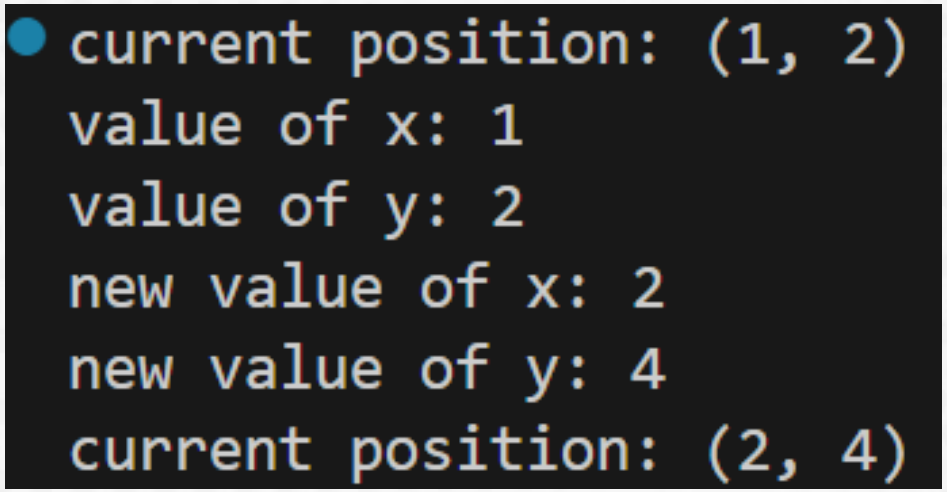
```
position = ( 1, 2 )

x, y = position
```

Expansão de Tuplos

- Como não podemos alterar os valores dos tuplos, podemos distribuí-los por variávies, que podem ser manipuladas.
- Exemplo:

```
player_position = (1, 2)
x, y = player_position
print ("current position:", player_position)
print ("value of x:", x)
print ("value of y:", y)
# player advances 1 in the x 2 in the y
x += 1
y += 2
print ("new value of x:", x)
print ("new value of y:", y)
player_position = (x, y)
print ("current position:", player_position)
```



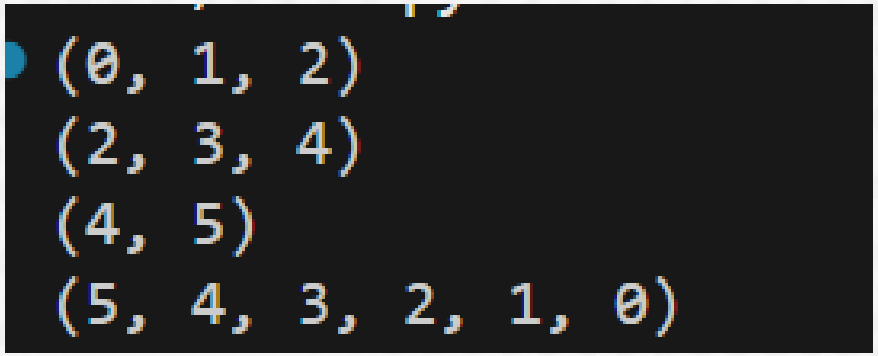
UNIVERSIDADE LUSÓFONA

Slicing de Tuplos

- Tal como as listas, os tuplos suportam slicing.
- Com listas criam-se sublistas, aqui permite criar subtuplos, o tipo mantém-se.

```
tuplo = (0, 1, 2, 3, 4, 5)

print(tuplo[:3])
print(tuplo[2:5])
print(tuplo[-2:])
print(tuplo[::-1])
```



UNIVERSIDADE LUSÓFONA

Exercícios

Exercícios - Tuplos

1. Cria um tuplo `hero` que contenha: nome, nível, vida, e experiência.

   Depois, imprime uma frase no formato: "O herói <nome> está no nível <nível> com <vida> HP e <experiência> XP."

2. Define um tuplo spawn_point com as coordenadas iniciais (0, 0, 0) e imprime cada valor num formato legível.

3. Cria um tuplo npc com um nome, uma idade e uma lista de itens que o NPC vende (3 strings).

   Depois imprime apenas o nome e a lista de itens.

4. Cria um tuplo `moedas = (5, 10, 2)` que representa o número de moedas de ouro, prata e bronze.

   Cada moeda tem um valor diferente (5, 3, 1), no final imprime: Moedas de cada tipo; total de moedas, valor por moedas e valor total

5. Cria um tuplo `scores = ("Ana", "Bruno", "Carla", "David", "Eva")`

   Atribui as duas primeiras variáveis a uma lista `primeiros` e guarda o resto na lista `outros`. No final imprime as duas listas.

UNIVERSIDADE LUSÓFONA

Tuplos - Resumo

- O que são:
  - Colecções ordenadas e imutáveis de elementos.
  - Pertencem aos tipos de dados complexos (coleções).
  - Sintaxe: `tuplo = (valor1, valor2, valor3, ...)`

UNIVERSIDADE LUSÓFONA

Tuplos - Resumo

- Características:
  - Mantêm a ordem dos elementos e são acessíveis por índice (tuplo[0], tuplo[-1]).
  - Permitem valores de tipos diferentes.
  - Suportam slicing (tuplo[1:4] → devolve um subtuplo).
  - Imutáveis → não é possível alterar, adicionar ou remover elementos.
  - Podem ser desempacotados com atribuição múltipla.

Tuplos - Resumo

- Usos comuns:
  - Agrupar dados fixos (ex.: coordenadas, registos, configurações).
  - Retornar múltiplos valores de uma função.
  - Garantir que os dados não sejam modificados.

Um tuplo é como um array fixo, ordenado, indexável e eficiente,

mas é imutável, sendo ideal para dados que não devem ser alterados.

Arrays Bidimensionais

Voltando à aventura

```
while True:
    print("What now?")
    command = input("Choose the direction you want to go north, south, east, west) or exit\n")

    if (command == "north"):
        print("you move north...")
    elif (command == "south"):
        print("you move south...")
    elif (command == "west"):
        print("you move west...")
    elif (command == "east"):
        print("you move east...")
    elif (command == "exit"):
        break
    else:
        print("I don't understand " + command + "!")
```

```
What now?
Choose the direction you want to go north, south, east, west) or exit
north
you move north...
What now?
Choose the direction you want to go north, south, east, west) or exit
south
you move south...
What now?
Choose the direction you want to go north, south, east, west) or exit
whatever
I don't understand whatever!
What now?
Choose the direction you want to go north, south, east, west) or exit
exit
```

UNIVERSIDADE LUSÓFONA

Arrays bidimensionais

- Queremos criar uma estrutura para armazenar as descrições e ligações entre várias salas.
- O jogador tem uma posição X e Y, e começa na posição 2,2.
- Podiamos definir as descrições como um array normal:

```
room_desc = []
room_desc[sala22] = "..."
```

- Como determinar aquele index?
  - Converter x e y para um índice

```
n_colunas = 5
room_desc = []
sala22 = (2 * n_colunas) + 2
room_desc[sala22] = "..."
```

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

UNIVERSIDADE LUSÓFONA

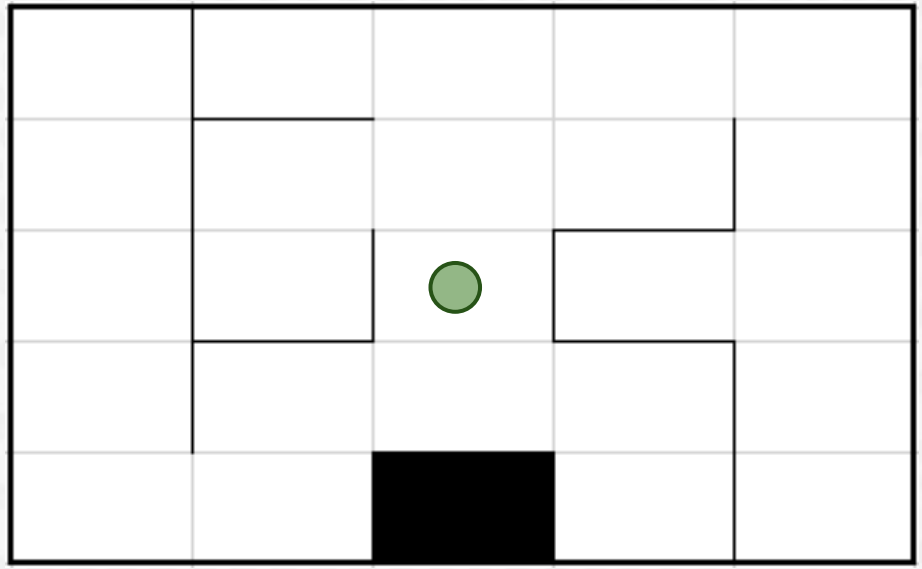
Arrays bidimensionais

- Necessitamos de uma estrutura onde possamos armazenas as descrições de cada sala consoante a coordenada
- Sabemos que o jogador tem uma posição X e Y, e começa na posição 2,2.

```
x, y = 2, 2
```

- Podiamos definir as descrições como um array normal.
- Mas o código seria mais legível se pudéssemos fazer:

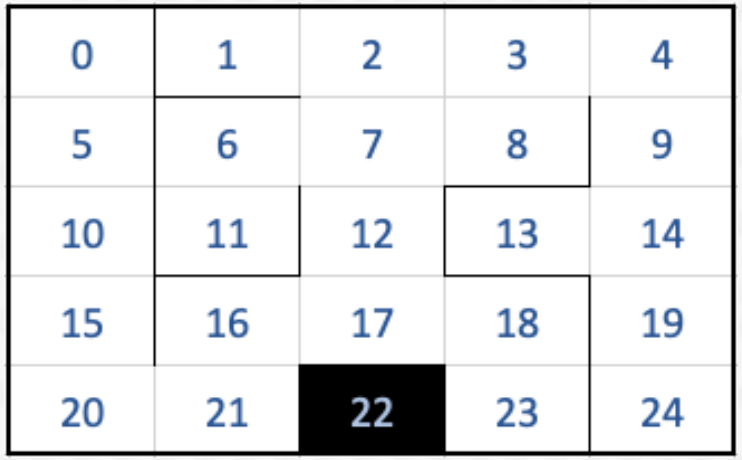
```
print(room_desc[x][y])
```



UNIVERSIDADE LUSÓFONA

Arrays bidimensionais

- Mas o Python não tem suporte nativo para arrays multi-dimensionais.
- Na realidade usamos listas de listas.
- A nossa lista ficava algo do genero:




```
mapa = [ [0, 1, 2, 3, 4], [5, 6, 7, 8, 9], [10, 11, 12, 13, 14], [15, 16, 17, 18, 19], [20, 21, 22, 23, 24] ]
```

Mas como é que podemos integrar isto?

Arrays bidimensionais

- Podemos criar uma lista de listas

```
n_columns = 5
n_rows = 5
room_desc = []
for i in range(0, n_columns):
    room_desc.append([])
    for j in range(0, n_rows):
        room_desc[i].append("Description of room x="+str(i)+", y="+str(j))
print(room_desc[3][2])
```

UNIVERSIDADE LUSÓFONA

Arrays bidimensionais

- Mais tarde vamos ver formas mais simples de fazer isto.
- O ciclo *for*, neste caso, gera a descrição automaticamente, não permite individualizar.
- Neste caso, queremos incluir uma descrição no de cada sala individualmente, por isso podemos a usar a inicialização de listas a nosso favor.
- O que o nosso duplo ciclo *for* faz, é isto:

```
room_desc = [
    [ "Room description x = 0, y = 0", "Room description x = 0, y = 1", "Room description x = 0, y = 2", "Room description x = 0, y = 3", "Room description x = 0, y = 4" ],
    [ "Room description x = 1, y = 0", "Room description x = 1, y = 1", "Room description x = 1, y = 2", "Room description x = 1, y = 3", "Room description x = 1, y = 4" ],
    [ "Room description x = 2, y = 0", "Room description x = 2, y = 1", "Room description x = 2, y = 2", "Room description x = 2, y = 3", "Room description x = 2, y = 4" ],
    [ "Room description x = 3, y = 0", "Room description x = 3, y = 1", "Room description x = 3, y = 2", "Room description x = 3, y = 3", "Room description x = 3, y = 4" ],
    [ "Room description x = 4, y = 0", "Room description x = 4, y = 1", "Room description x = 4, y = 2", "Room description x = 4, y = 3", "Room description x = 4, y = 4" ],
]
print(room_desc[3][2])
```

```
$ py test_array.py
Room description x = 3, y = 2
```

- Mas isto também é pouco intuitivo,
  - não é assim que lemos grelhas, fazia mais sentido definir o y primeiro e só depois o x

UNIVERSIDADE LUSÓFONA

Arrays bidimensionais

- O que podemos fazer?
- Vamos introduzir a lista manualmente na inicialização.
- **Por exemplo:**

```
room_description = [
    ["A", "B", "C", "D", "E"],
    ["F", "G", "H", "I", "J"],
    ["K", "L", "M", "N", "O"],
    ["P", "Q", "R", "S", "T"],
    ["U", "V", "W", "X", "Y"]
    ]
```

UNIVERSIDADE LUSÓFONA

Arrays bidimensionais

- Temos também que definir uma posição inicial do *player*

```
position = (2,2)
```

- E uma forma de vericarmos a posição e a descrição dentro do while

```
x, y = position

description = room_descriptions[y][x]

print(position, ":", description)
```

UNIVERSIDADE LUSÓFONA

Arrays bidimensionais

Arrays bidimensionais

- Agora vamos integrar o movimento para mover o jogador

```
if (command == "north"):
   # Check if we can move north
   print("You move north...")
   y = y - 1
```

- Fazemos o mesmo para as restantes direcções....

- No final actualizamos a posição

```
# At the end of the code, move x and y back to the
# position tuple
position = (x, y)
```

UNIVERSIDADE LUSÓFONA

Verificação de saídas

- Neste momento existe movimento, mas sem limites
- Precisamos de "construir" as paredes do mapa para não saírmos das listas
- Podemos usar uma lista de listas de tuplos para verificar saídas
- Para a parte da "lista de lista", usamos a mesma estratégia que antes
- Em vez de usarmos uma string, usamos um tuplo para armazenar as saídas possíveis
- Para começar temos que criar keys que permitam aceder ao índice de direcção

```
NORTH = 0
EAST = 1
SOUTH = 2
WEST = 3
```

Constantes são declarados como variáveis com valores que se mantêm estáticos, são representados por CAPS, por convenção

UNIVERSIDADE LUSÓFONA

Verificação de saídas

- Agora podemos começar a criar os nossos tuplos baseando-nos no layout do mapa:

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O
P	Q	R	S	T
U	V		X	Y

```
room_exits = [
    [(False, False, True, False), (False, True, False, False), (False, True, True, True), (False, True, True, True), (False, False, True, True)],
    [(True, False, True, False), (False, True, True, False), (True, True, True, True), (True, False, False, True), (True, False, True, False)],
    [(True, False, True, False), (True, False, False, False), (True, False, True, False), (False, True, False, False), (True, False, True, True)],
    [(True, False, True, False), (False, True, True, False), (True, True, False, True), (False, False, True, False), (True, False, True, False)],
    [(True, True, False, False), (True, False, False, True), (False, False, False, False), (True, False, False, False), (True, False, False, False)]
    ]
```

UNIVERSIDADE LUSÓFONA

Verificação de saídas

- E para finalizar, usamos a lista de lista de tuplos para fazermos a verificação de saída dentro de cada direcção

```
if (command == "north"):
        if (room_exits[y][x][NORTH]):
            print("you move north...")
            y -= 1
        else:
            print("Can't go north!")
```

- E repetimos para todas as direcções

Arrays bidimensionais

- No final devemos ter algo como isto:

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

UNIVERSIDADE LUSÓFONA

Exercícios

UNIVERSIDADE LUSÓFONA

Exercícios

1. Localização no mapa

   Cria uma matriz 3x3 chamada mapa, onde cada célula contém uma string com a descrição da sala (ex.: "Sala (x,y)").

   Depois, cria um tuplo posicao = (1, 2) e imprime a descrição correspondente.
2. Usando o mesmo mapa 3x3, define uma posição inicial (1,1) e implementa as seguintes regras de movimento:

   "w" → sobe / "s" → desce / "a" → esquerda / "d" → direita

   O programa deve:
   - Pedir um comando (w, a, s, d),
   - Atualizar a posição,
   - Mostrar a nova descrição da sala.
   - Guardar o histórico da salas onde o jogador esteve e imprimi-la na saída
3. Expande o exercício anterior.

   Se o jogador tentar sair do mapa, o programa deve: impedir o movimento e mostrar a mensagem: “Não podes sair do mapa!”