Aula 8
26/11/2025

UNIVERSIDADE LUSÓFONA

Fundamentos de Programação (T)

Licenciatura em Videojogos

Ano 1 / Semestre 1

Filipe Pinto
filipe.pinto@ulusofona.pt

UNIVERSIDADE LUSÓFONA

Sumário

- Programação Orientada a Objectos:
  - Classes
  - Instâncias
  - Constutores

UNIVERSIDADE LUSÓFONA

Programação Orientada a Objectos

UNIVERSIDADE LUSÓFONA

Tipos de dados

- Até agora temos usados os tipos de dados que o Python tem disponível:
  - Números (ints e floats)
  - Strings
  - Tuplos
  - Listas
  - Etc.
- Mas quando o nosso programa começa a ficar mais complexo, torna-se útil criar novos tipos que agrupem dados relacionados.
- Este é o papel das classes: permitir-nos definir estruturas adaptadas ao problema que queremos resolver.

Porquê os nossos próprios tipos?

- Quando vários dados pertencem logicamente ao mesmo "conceito", mantê-los separados em variáveis independentes complica o código.
- Criar um novo tipo permite reunir todos os dados relevantes num único objeto, facilitando a escrita, leitura e manutenção do programa.
- No caso da nossa Aventura de Texto, as nossa salas são compostas por duas variáveis globais:
  - `room_desc`: Estrutura bidimensional que contém as descrições das salas.
  - `room_exits`: Estrutura tridimensional onde, além daposição, tem uma lista de valores booleanos que indicam as saídas possíveis.
- Ambos os arrays são indexados pelas coordenadas (x, y), estando os dados directamente correlacionados pela posição.
- Logicamente, estas duas estruturas representam informação da mesma sala, mas encontram-se separadas porque ainda não dispomos de um mecanismo que permita agrupar todos os dados da sala numa única entidade.

UNIVERSIDADE LUSÓFONA

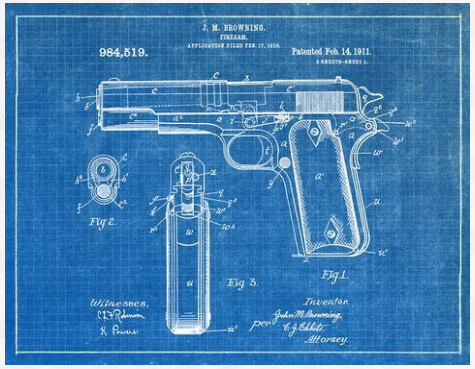
Um novo tipo – O tipo “Room”

- Portanto, o que gostávamos era de ter um tipo de dados “Room”
  - Este tipo iria armazenar a descrição da sala, e as saídas válidas.
  - Esta tipo poderia também incluir outras coisas que pudessem ser úteis em futuro desenvolvimento:
    - Loot, enemies, events, etc.
- Esta é uma aplicação tipica para o uso de classes

UNIVERSIDADE LUSÓFONA

Classes

- Uma classe é a descrição de uma estrutura de dados.
  - Pode conter diferentes variáveis (**atributos**) e funções (**métodos**).
- Uma classe pode ser **instanciada**, dando origem a objectos
- Podemos pensar numa classe como o plano de algo, e um objecto como o que se produziu a partir desse plano.



UNIVERSIDADE LUSÓFONA

Classes

- Podemos ver uma classe como um molde ou uma *blueprint* que descreve um novo tipo de objecto
- Definimos os dados (atributos) e operações (métodos) que esse tipo terá.
- Depois, sempre que criarmos uma instância dessa classe, obtemos um objeto real que segue essa estrutura.
- Exemplos comuns no uso de classes:
  - Enemies (diferentes tipos de inimigos com diferentes atributos e diferentes mecânicas)
  - Weapons (diferentes tipos de armas com diferentes atributos e diferentes tipos de mecânicas)

UNIVERSIDADE LUSÓFONA

Classe "Room" - declaração

```
class Room:
    pass
```

Usamos **`pass`** aqui porque estamos a definir uma classe vazia

- Aqui estamos apenas a declarar a classe
- O `pass` significa que "não há aqui nada para já"
- Mais tarde iremos preencher esta classe com atributos e métodos.

UNIVERSIDADE LUSÓFONA

Classe “Room”

```
class Room:
    description = ""
```

Quando declaramos uma variável temos que inicializá-la.

Esta inicialização é feita por uma questão de facilidade de leitura. Esta inicializa-se com o mesmo tipo que esperamos que a variável tenha no futuro, neste caso uma `string`.

UNIVERSIDADE LUSÓFONA

Classe "Room"

```
class Room:
        description = ""
        exits = [ False, False, False, False ]
```

O mesmo nesta declaração, `exits` recebe uma lista com 4 `bools`. Estes são inicializados como `false`, por convenção

- Estes atributos pertencem à classe, mas serão usados por cada instância individual.
- Inicializamos com valores padrão para indicar o tipo de dados esperado.

UNIVERSIDADE LUSÓFONA

Instanciação

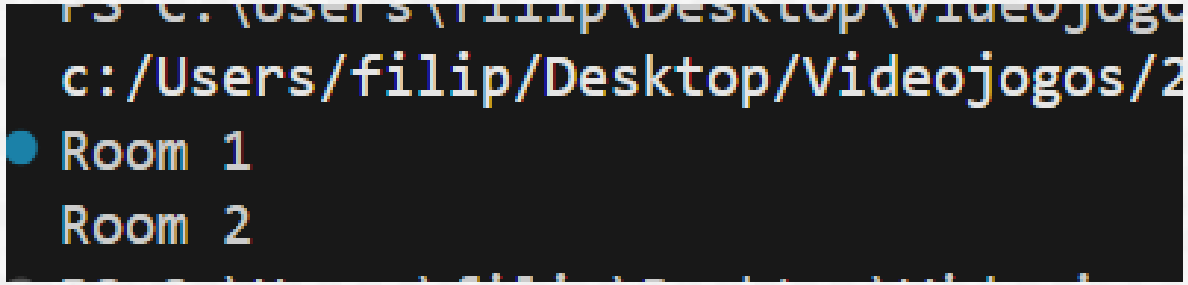
- Agora que temos a classe Room definida podemos começar a criar objectos desse tipo.
- Para isso vamos instanciá-la
- Criar uma instância significa gerar um objeto real, baseado no molde da classe.
- Cada instância tem a sua própria cópia dos atributos, que podemos modificar sem afetar outras instâncias.

```
r1 = Room()
r1.description = "Room 1"
print(r1.description)
```

```
c:/Users/filip/Desktop/Videoj
Room 1
PS C:\Users\filip\Desktop\Vid
```

Objectos instanciados são independentes

```
r1 = Room()

r1.description = "Room 1"

r2 = Room()

r2.description = "Room 2"



print(r1.description)

print(r2.description)
```



Objectos instanciados são independentes

```
r1 = Room()
r1.description = "Room 1"
r1.exits = [True, False, True, False]
r2 = Room()
r2.description = "Room 2"
r2.exits[2] = True

print(r1.description, r1.exits)
print(r2.description, r2.exits)
```

```
Room 1 [True, False, True, False]
Room 2 [False, False, True, False]
```

UNIVERSIDADE LUSÓFONA

Objectos instanciados são independentes

- Mesmo tendo sido criados a partir da mesma classe, cada objeto é independente.
- Alterar os atributos de um objeto não altera os atributos dos outros.
- Isto é fundamental para representar várias entidades do mesmo tipo no nosso jogo.

UNIVERSIDADE LUSÓFONA

Métodos

- Métodos são funções definidas dentro de uma classe e servem para manipular ou apresentar os dados do próprio objeto.

```
class Room:
    description = ""
    exits = [False, False, False, False]

    def Log(self):
        print(self.description)
        print(self.exits)
```

Define-se como qualquer função, dentro do *scope* da classe.

O primeiro parâmetro (`self`) representa a instância concreta que está a chamar o método, permitindo aceder aos seus atributos.

Métodos

```
class Room:
    description = ""
    exits = [False, False, False, False]

    def Log(self):
        print(self.description)
        print(self.exits)
```

Todos os métodos precisam de levar pelo menos um paramêtro, para contêr a instância do objecto

Muitas das linguagens de programação não precisam de fazer isto explicitamente, mas o paramêtro está lá escondido.

Métodos

```
class Room:
    description = ""
    exits = [False, False, False, False]

    def Log(self):
        print(self.description)
        print(self.exits)
```

Quando acedemos a variáveis da classe, precisamos dizer a que instância nos referímos (que normalmente é aquela que recebemos como parâmetro da função)

Métodos

```
class Room:
    description = ""
    exits = [False, False, False, False]

    def Log(self):
      print(self.description)
      print(self.exits)

r1 = Room()
r1.description = "Room 1"
r1.exits = [True, False, True, False]
r1.Log()
```

- Quando chamamos o método num objeto, não precisamos de passar o self.
- O Python adiciona esse parâmetro automaticamente.
- Assim, `r1.Log()` é o mesmo que chamar `Room.Log(r1)`.

UNIVERSIDADE LUSÓFONA

Métodos

```
class Room:
    description = ""
    exits = [False, False, False, False]

    def Log(self):
        print(self.description)
        print(self.exits)

r1 = Room()
r1.description = "Room 1"
r1.exits = [True, False, True, False]
r1.Log()
```

```
c:/Users/filip/Desktop/Videojogos/2526/FP/Aulas/
Room 1
[True, False, True, False]
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aul
```

UNIVERSIDADE LUSÓFONA

Métodos - Parâmetros

```
class Room:
    description = ""
    exits = [False, False, False, False]

    def Log(self, count):
        for i in range(count):
            print(self.description)
            print(self.exits)
r1 = Room()
r1.description = "Room 1"
r1.exits = [True, False, True, False]
r1.Log(3)
```

- Métodos podem receber parâmetros adicionais para alterar o seu comportamento.
- Também podem ter valores por defeito, tal como funções normais.

Métodos - Parâmetros

```
class Room:
    description = ""
    exits = [False, False, False, False]

    def Log(self, count):
        for i in range(count):
            print(self.description)
            print(self.exits)
r1 = Room()
r1.description = "Room 1"
r1.exits = [True, False, True, False]
r1.Log(3)
```

```
Room 1
[True, False, True, False]
Room 1
[True, False, True, False]
Room 1
[True, False, True, False]
```

Métodos - Parâmetros

```
class Room:
    description = ""
    exits = [False, False, False, False]

    def Log(self, count = 2):
        for i in range(count):
            print(self.description)
            print(self.exits)
r1 = Room()
r1.description = "Room 1"
r1.exits = [True, False, True, False]
r1.Log()
```

```
Room 1
[True, False, True, False]
Room 1
[True, False, True, False]
```

UNIVERSIDADE LUSÓFONA

Construtores

- Construtores são um método especial nas classes, que é invocado quando criamos um novo objecto.
- Esse método chama-se `__init__` :


```
class Room:
    description = ""
    exits = [False, False, False, False]

    def Log(self, count = 1):
      for i in range(count):
        print(self.description)
        print(self.exits)

    def __init__(self):
      self.description = "Room N"
```

Construtores

```
class Room:
    description = ""
    exits = [False, False, False, False]

    #def __init__(self):
      #self.description = "Room N"

r1 = Room()
r1.Log()
```

```
PS C:\Users\filip\Desktop\Videojogos\
c:/Users/filip/Desktop/Videojogos/252

[False, False, False, False]
```

UNIVERSIDADE LUSÓFONA

Construtores

```
class Room:
    description = ""
    exits = [False, False, False, False]

    def __init__(self):
      self.description = "Room N"

r1 = Room()
r1.Log()
```

```
Room N
[False, False, False, False]
```

UNIVERSIDADE LUSÓFONA

Construtores - Parâmetros

- Podemos definir parâmetros no construtor para garantir que cada objeto começa com os valores certos logo no momento da criação.
- Isto torna o código mais claro e evita erros causados por objetos parcialmente configurados.

```
class Room:
    description = ""
    exits = [False, False, False, False]

    def __init__(self, desc, north, south, east, west):
      self.description = desc
      self.exits = [north, south, east, west]
```

Construtores - Parâmetros

```
class Room:
    description = ""
    exits = [False, False, False, False]

    def __init__(self, desc, north, south, east, west):
      self.description = desc
      self.exits = [north, south, east, west]

r1 = Room()
r1.Log()
```

```
Traceback (most recent call last):
  File "c:\Users\filip\Desktop\Videojogos\2526\FP\Aulas\Aula8_FP.py", line 15, in <module>
    r1 = Room()
         ^^^^^^
TypeError: Room.__init__() missing 5 required positional arguments: 'desc', 'north', 'south', 'east', and 'west'
```

UNIVERSIDADE LUSÓFONA

Construtores - Parâmetros

```
class Room:
    description = ""
    exits = [False, False, False, False]

    def __init__(self, desc, north, south, east, west):
      self.description = desc
      self.exits = [north, south, east, west]

r1 = Room("Room description 0, 0", False, False, True, False)
r1.Log()
```

```
Room description 0, 0
[False, False, True, False]
```

UNIVERSIDADE LUSÓFONA

Voltando à Aventura de Texto

UNIVERSIDADE LUSÓFONA

Na aventura de texto

- Como é que poderiamos implementar esta lógica?
- Primeiro, criando uma classe que tenha todos os dados relativos a cada sala
  - Descrição
  - Saídas
  - E uma função que faça a verificação se a saída existe

Na aventura de texto

- Criamos a classe Room:

```
class Room:
    description = ""
    exits = [False, False, False, False]

    def __init__(self, desc, north, south, east, west):
        self.description = desc
        self.exits = [north, south, east, west]

    def CanExit(self, dir):
        # Fazemos verificação de saída que
        # devolve um bool que vamos usar abaixo
        return self.exits[dir]
```

Na aventura de texto

- Criamos as instâncias de cada sala seguindo a lógica do contrutor:

```
Room("Description string", bool1, bool2, bool3, bool4)
```

```
rooms = [
    [
        Room("Room description 0, 0", False, False, True, False),
        Room("Room description 1, 0", False, True, False, False),
        Room("Room description 2, 0", False, True, True, True),
        Room("Room description 3, 0", False, True, True, True),
        Room("Room description 4, 0", False, False, True, True)
    ],
    [
        Room("Room description 0, 1", True, False, True, False),
        Room("Room description 1, 1", False, True, True, False),
        Room("Room description 2, 1", True, True, True, True),
        Room("Room description 3, 1", True, False, False, True),
        Room("Room description 4, 1", True, False, True, False)
    ],
    [
        Room("Room description 0, 2", True, False, True, False),
        Room("Room description 1, 2", True, False, False, False),
        Room("Room description 2, 2", True, False, True, False),
        Room("Room description 3, 2", False, True, False, False),
        Room("Room description 4, 2", True, False, True, True)
    ],
    [
        Room("Room description 0, 3", True, False, True, False),
        Room("Room description 1, 3", False, True, True, False),
        Room("Room description 2, 3", True, True, False, True),
        Room("Room description 3, 3", False, False, True, False),
        Room("Room description 4, 3", True, False, True, False)
    ],
    [
        Room("Room description 0, 4", True, True, False, False),
        Room("Room description 1, 4", True, False, False, True),
        Room("Room description 2, 4", False, False, False, False),
        Room("Room description 3, 4", True, False, False, False),
        Room("Room description 4, 4", True, False, False, False)
    ]
]
```

Na aventura de texto

```
rooms = [
    [
        Room("Room description 0, 0", False, False, True, False),
        Room("Room description 1, 0", False, True, False, False),
        Room("Room description 2, 0", False, True, True, True),
        Room("Room description 3, 0", False, True, True, True),
        Room("Room description 4, 0", False, False, True, True)
    ],
    [
        Room("Room description 0, 1", True, False, True, False),
        Room("Room description 1, 1", False, True, True, False),
        Room("Room description 2, 1", True, True, True, True),
        Room("Room description 3, 1", True, False, False, True),
        Room("Room description 4, 1", True, False, True, False)
    ],
    [
        Room("Room description 0, 2", True, False, True, False),
        Room("Room description 1, 2", True, False, False, False),
        Room("Room description 2, 2", True, False, True, False),
        Room("Room description 3, 2", False, True, False, False),
        Room("Room description 4, 2", True, False, True, True)
    ],
    [
        Room("Room description 0, 3", True, False, True, False),
        Room("Room description 1, 3", False, True, True, False),
        Room("Room description 2, 3", True, True, False, True),
        Room("Room description 3, 3", False, False, True, False),
        Room("Room description 4, 3", True, False, True, False)
    ],
    [
        Room("Room description 0, 4", True, True, False, False),
        Room("Room description 1, 4", True, False, False, True),
        Room("Room description 2, 4", False, False, False, False),
        Room("Room description 3, 4", True, False, False, False),
        Room("Room description 4, 4", True, False, False, False)
    ]
]
```


```
rooms = [
    [
        Room("Room description 0, 0", False, False, True, False),
        Room("Room description 1, 0", False, True, False, False),
        Room("Room description 2, 0", False, True, True, True),
        Room("Room description 3, 0", False, True, True, True),
        Room("Room description 4, 0", False, False, True, True)
    ],
    [
        Room("Room description 0, 1", True, False, True, False),
        Room("Room description 1, 1", False, True, True, False),
        Room("Room description 2, 1", True, True, True, True),
        Room("Room description 3, 1", True, False, False, True),
        Room("Room description 4, 1", True, False, True, False)
    ],
    [
        Room("Room description 0, 2", True, False, True, False),
        Room("Room description 1, 2", True, False, False, False),
        Room("Room description 2, 2", True, False, True, False),
        Room("Room description 3, 2", False, True, False, False),
        Room("Room description 4, 2", True, False, True, True)
    ],
    [
        Room("Room description 0, 3", True, False, True, False),
        Room("Room description 1, 3", False, True, True, False),
        Room("Room description 2, 3", True, True, False, True),
        Room("Room description 3, 3", False, False, True, False),
        Room("Room description 4, 3", True, False, True, False)
    ],
    [
        Room("Room description 0, 4", True, True, False, False),
        Room("Room description 1, 4", True, False, False, True),
        Room("Room description 2, 4", False, False, False, False),
        Room("Room description 3, 4", True, False, False, False),
        Room("Room description 4, 4", True, False, False, False)
    ]
]
```

UNIVERSIDADE LUSÓFONA

Na aventura de texto

- E depois integramos a lógica no jogo

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

    x, y = position

    current_room = rooms[y][x]

    if (current_room.CanExit(direction_index)):
        print("\nYou move " + direction_text)
        y += y_inc
        x += x_inc
        position = (x, y)
        current_room = rooms[y][x]
        print("\nPosition: "  + str(position))
        print("\nRoom description: "  + current_room.description + "\n")
    else:
        print("You can't move " + direction_text + "!\n")
```

Na aventura de texto

- E depois integramos a lógica no jogo

```
while command != "exit":
    print("What now?")

    command = input("Choose the direction you want to go north, south, east, west) or exit\n").lower()
    command = command.strip()

    if command == "":
        continue

    if (command in command_processor):
        command_processor[command]()
    elif command == "exit":
        print("Exiting game...")
        continue
    else:
        print("I don't understand " + command + "!")
```

Exercícios

UNIVERSIDADE LUSÓFONA

Exercícios

1. Jogador com posição e stamina
    - Cria uma classe Player com:
        - atributos: name, x, y, stamina
        - construtor que recebe name, x, y e stamina
        - métodos:
            - log() → imprime posição e stamina
            - move(dx, dy) → altera x e y e reduz stamina em 1 por cada movimento (mesmo que seja diagonal)
            - is_exhausted() → devolve True se stamina <= 0, caso contrário False
    - **Programa principal:**
        1. Cria um Player na posição (0, 0) com stamina 5. Pede input para o nome
        2. Dentro dum while usa o input do player para se movimentar:
            - Pode se mover na vertical, horizontal e diagonal
            - Usar um dicionário com comandos e lambdas para definir a direcção
        3. Para cada movimento:
            - Se o jogador não estiver exausto, chama move
            - Se já estiver exausto, imprime "Too tired to move" e pára o ciclo
        4. No fim, chama log().

Exemplo de execução

```
Type your name: Filipe
Player: Filipe
Position: ( 0 , 0 )
Stamina: 5
Directions: n s e w ne nw se sw
q = quit

Current Position -> x: 0 y: 0
Move: ne
Current Position -> x: 1 y: -1
Move: ne
Current Position -> x: 2 y: -2
Move: e
Current Position -> x: 3 y: -2
Move: ne
Current Position -> x: 4 y: -3
Move: se
Current Position -> x: 5 y: -2
Too tired to move
Player: Filipe
Position: ( 5 , -2 )
Stamina: 0
```

UNIVERSIDADE LUSÓFONA

Exercícios

2. Itens e Inventário com limite duplo

- Cria uma classe Item com:
  - name, weight, value
- Construtor recebe estes três parâmetros.
- Cria uma classe Inventory com:
  - max_items (número máximo de itens), max_weight (peso máximo total), items (lista de Item, começa vazia)
- Métodos de Inventory:
  - total_weight() → devolve a soma dos pesos de todos os itens
  - total_value() → devolve a soma dos valores de todos os itens
  - can_add(item) → devolve True se não exceder nem o número máximo de itens nem o peso máximo total
  - add_item(item) → se can_add(item) for True, adiciona o item; caso contrário, imprime "Cannot add <nome>: inventory limit reached"
- **Programa principal:**
  1. Cria um inventário com max_items = 4 e max_weight = 15.
  2. Cria pelo menos 5 itens diferentes (espada, escudo, poção, ouro, armadura, etc.).
  3. Usa o random para teres 3 objectos para escolher e input para apanhares 1 deles.
  4. O while corre até o player atingir um dos limites ou fizer quit
  5. No fim, imprime todos os nomes dos items no inventário e total_weight() e total_value().

Exemplo de execução

```
=== INVENTORY ===
- Shield (w:6, v:8)
- Sword (w:5, v:10)
Items: 2 / 4
Weight: 11 / 15
Total value: 18
================

Items available:
1 - Sword (weight: 5, value: 10)
2 - Ring (weight: 1, value: 25)
3 - Shield (weight: 6, value: 8)
Choose an item number to pick or 'q' to quit
>> q

=== INVENTORY ===
- Shield (w:6, v:8)
- Sword (w:5, v:10)
Items: 2 / 4
Weight: 11 / 15
Total value: 18
================
```

```
You open a chest...


=== INVENTORY ===
(empty)
Items: 0 / 4
Weight: 0 / 15
Total value: 0
================

Items available:
1 - Shield (weight: 6, value: 8)
2 - Ring (weight: 1, value: 25)
3 - Potion (weight: 1, value: 4)
Choose an item number to pick or 'q' to quit
>> 1
Shield added to inventory

=== INVENTORY ===
- Shield (w:6, v:8)
Items: 1 / 4
Weight: 6 / 15
Total value: 8
================

Items available:
1 - Sword (weight: 5, value: 10)
2 - Potion (weight: 1, value: 4)
3 - Ring (weight: 1, value: 25)
Choose an item number to pick or 'q' to quit
>> 1
Sword added to inventory

=== INVENTORY ===
- Shield (w:6, v:8)
- Sword (w:5, v:10)
Items: 2 / 4
Weight: 11 / 15
Total value: 18
================

Items available:
1 - Ring (weight: 1, value: 25)
2 - Gold (weight: 2, value: 15)
3 - Sword (weight: 5, value: 10)
Choose an item number to pick or 'q' to quit
>> 3
Cannot add Sword: inventory limit reached
```

UNIVERSIDADE LUSÓFONA

Exercícios

3. Sistema de quests com estados

- Cria uma classe Quest com:
  - Title, description, status (string que pode ser "locked", "active" ou "completed")
- Construtor recebe title e description e inicia o status como "locked".
- Métodos:
  - unlock() → se status for "locked", passa a "active"
  - complete() → se status for "active", passa a "completed"
  - log() → imprime título e estado atual

- **Programa principal:**
  - Cria uma lista com 3 Quest:
    - `Quest("Speak with villagers", "Talk to the people in the village.")`
    - `Quest("Clear dungeon", "Defeat all enemies in the old dungeon.")`
    - `Quest("Defeat area boss", "Kill the boss that controls the region.")`
  - Usa input para Desbloquear a primeira quest e depois marca-a como completa.
  - Quando uma quest ficar "completed", desbloqueia automaticamente a próxima.
  - Imprime o estado de todas as quests em três momentos:
    - antes de qualquer alteração
    - depois de completar uma quest
    - No final, imprime o log e sai do loop se todas as quests estiverem completas ou o input for *quit*

Exemplos de execução