Aula 6
19/11/2025

UNIVERSIDADE LUSÓFONA

Fundamentos de Programação (T)

Licenciatura em Videojogos

Ano 1 / Semestre 1

Filipe Pinto
filipe.pinto@ulusofona.pt

UNIVERSIDADE LUSÓFONA

Sumário

- Stacks
- Queues
- Módulo
  - Importação e Utilização
- Número aleatórios
  - Random()

UNIVERSIDADE LUSÓFONA

Stacks & Queues

UNIVERSIDADE LUSÓFONA

Stacks & Queues

Stacks e Queues (ou pilhas e filas) são estruturas de dados que permitem inserir e remover elementos de forma ordenada

- Estruturalmente assemelham-se a listas normais (Arrays) mas a forma como o Python processa os dados é mais otimizada, permitindo uma melhor gestão de memória e *performace-hit* quando se manipula elementos.

- Usam o módulo *Deque()*
  - Vem do módulo collections e significa "fila de duas extremidades" (*Double-Ended Queue)*
  - Estrutura otimizada para inserir e remover elementos tanto no início como no fim
  - Listas normais são mais lentas para remover do início, pois têm de deslocar todos os elementos.
  - Em programas com muitas operações de entrada/saída de dados, *deque* é muito mais eficiente.

UNIVERSIDADE LUSÓFONA

Stacks & Queues

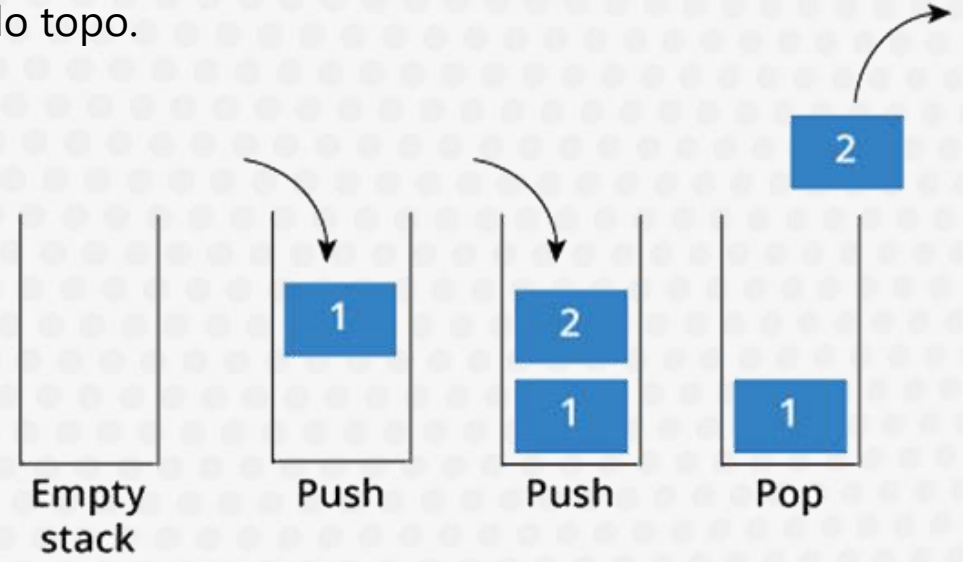
Diferencia-se pela forma como processam os dados:

- Stacks:
  - **LIFO** → *Last In, First Out*
  - Normalmente tem duas funções → push e pop
- Queues:
  - **FIFO** → *First In, First Out*
  - Normalmente tem duas funções → Enqueue e dequeue

UNIVERSIDADE LUSÓFONA

Stacks

- Stacks funcionam como se fossem pilhas de papéis.
- Adicionamos papéis à pilha.
- Quando vamos processar a papelada, fazêmo-lo removendo do topo.
- Logo, os últimos a entrar são os primeiros a saír.



UNIVERSIDADE LUSÓFONA

Stacks – Criação

```
from collections import deque
```

"deque" é abreviatura de double-ended queue

```
stack = deque()
print(stack)
stack.append(5)
print(stack)
```

Um *deque* não pertence à linguagem própriamente dita, mas pertence à biblioteca collections *default* do Python.

Como tal, temos de importá-la.

Stacks – Criação

```
from collections import deque

stack = deque()
print(stack)
stack.append(5)
print(stack)
```

`append()` é a função que se usa numa operação de push

```
deque([])
deque([5])
```

Stacks – Criação

```
from collections import deque

stack = deque()
stack.append(5)
stack.append(“abc”)
print(stack)
```

```
deque([5, 'abc'])
```

Stacks – Remover elementos

```
from collections import deque


stack = deque()
stack.append(5)
stack.append(“abc”)
print(stack)
value = stack.pop()
print("Removed", value, "from the stack")
print(stack)
```

```
deque([5, 'abc'])
Removed abc from the stack
deque([5])
```

Stacks – Remover elementos

```
from collections import deque


stack = deque()
print(stack)
value = stack.pop()
print("Removed", value, "from the stack")
print(stack)
```

```
deque([])
Traceback (most recent call last):
  File "c:\Users\filip\Desktop\Videojogos\2526\FP\Aulas\aula5_fp.py", line 104, in <module>
    value = stack.pop()
            ^^^^^^^^^^^
IndexError: pop from an empty deque
```

Stacks – Verificação

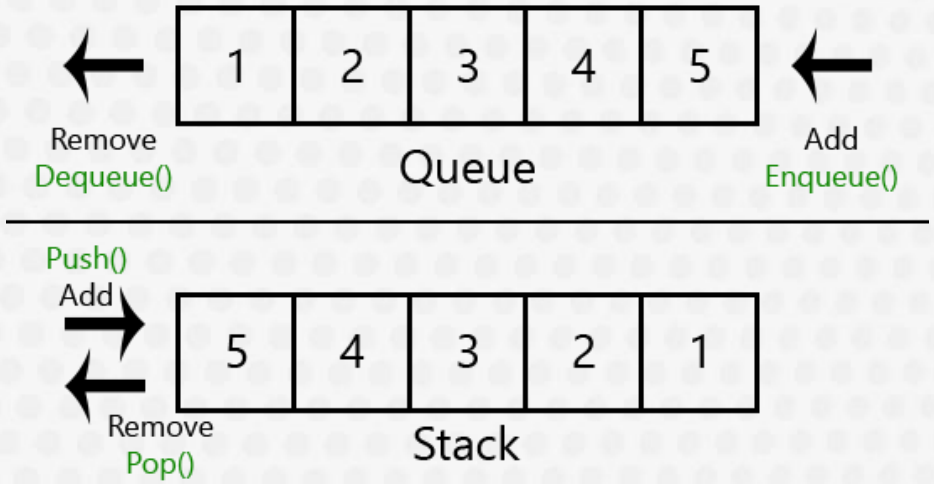
```
from collections import deque


stack = deque()
print(stack)
if (stack):
    value = stack.pop()
    print("Removed " + str(value) + " from stack")
else:
    print("Stack was empty!")

print(stack)
```

Um stack numa condição é convertido implicitamente para um booleano: `True` se tive elementos, `False` se estiver vazio.

```
deque([])
Stack was empty
deque([])
```

Queues

- Parece igual ao Stack, porque efectivamente é.
- Mundanças são semânticas e não estruturais.
- Funcionam exactamente como uma fila de um supermercado.
- Os elementos vão ser processados pela ordem de entrada.
- Os primeiros a entrar são os primeiros a saír.



Queues – Criação

```
from collections import deque
```

Tal como o *stack*, uma *queue* não pertence à linguagem própriamente dita.

Como tal, temos de importá-la.

```
queue = deque()
print(queue)
queue.appendleft(5)
print(queue)
```

`appendleft()` é o nome da função que se usa numa operação de *enqueue.*
Enquanto o `append()` acrescenta dados no fim da lista, o `appendleft()` acrescenta dados ao início da lista.

```
deque([])
deque([5])
```

Queues – Criação

```
from collections import deque


queue = deque()
print(queue)
queue.appendleft(5)
print(queue)
queue.appendleft("abc")
print(queue)
```

```
deque([])
deque([5])
deque(['abc', 5])
```

Queues – Remover elementos

```
from collections import deque

queue = deque()
queue.appendleft(5)
queue.appendleft("abc")
print(queue)

value = queue.pop()
print("Removed " + str(value) + " from queue")
print(queue)
```

```
deque(['abc', 5])
Removed 5 from queue
deque(['abc'])
```

Queues – Remover elementos

```
from collections import deque


queue = deque()
print(queue)


value = queue.pop()
print("Removed " + str(value) + " from queue")
print(queue)
```

```
deque([])
Traceback (most recent call last):
  File "c:\Users\filip\Desktop\Videojogos\2526\FP\Aulas\aula5_fp.py", line 118, in <module>
    value = queue.pop()
            ^^^^^^^^^^^
IndexError: pop from an empty deque
```

Queues – Remover elementos

```
from collections import deque

queue = deque()
print(queue)

if(queue):
    value = queue.pop()
    print("Removed", value, "from the queue")
else:
    print("Queue was empty")

print(queue)
```

Tal como a *stack*, uma *queue* numa condição é convertida implicitamente para um booleano: `True` se tive elementos, `False` se estiver vazio.

```
deque([])
Queue was empty
deque([])
```

Comparação com Listas

- Podiamos usar listas para fazer exatamente o mesmo
- Mas a funcionalidade optimizada das listas é diferente do *Deque*
- O módulo *Deque* está otimizado para lidar com as extremidades das listas → **FIFO** e **LIFO**
    - Estas operações com listas iriam requerem uma reindexação e movimentação de elementos (**mais dispendioso**)
- No entanto, se o objectivo é remover elementos ou índices específicos, a lista continua a ser a melhor forma.
- Estamos apenas a usar `append()`, `appendleft()` e `pop()`.
    - Poderiamos também usar `popleft()` com *Deque*.

UNIVERSIDADE LUSÓFONA

Exercícios

Exercícios

```
Action (walk/attack/heal/undo/exit): walk
Action: walk registered
Action (walk/attack/heal/undo/exit): Attack
Action: attack registered
Action (walk/attack/heal/undo/exit): heal
Action: heal registered
Action (walk/attack/heal/undo/exit): undo
Action: heal undone
Action (walk/attack/heal/undo/exit): attack
Action: attack registered
Action (walk/attack/heal/undo/exit): exit
Historic: deque(['walk', 'attack', 'attack'])
```

1. Simula o histórico de ações de um jogador num RPG.
   - Cada ação é adicionada ao topo.
   - Se o jogador usar o comando "**undo**", desfaz a última ação.

```
Combat with: Troll is starting.
Enemy: Troll  has been defeated.
Combat with: Orc is starting.
Enemy: Orc  has been defeated.
Combat with: Goblin is starting.
Enemy: Goblin  has been defeated.
```

2. Cria uma lista que representa inimigos a serem processados por ordem de entrada.
   - Os inimigos entram na lista quando aparecem e saem quando são derrotados.
   - `enemies = deque(["Goblin", "Orc", "Troll"])`

```
Processing event:  boss arrival
Processing event:  pause
Processing event:  start cutscene
Processing event:  update player position
Processing event:  detect collision
Processing event:  generate particles
```

3. Simula o motor de eventos de um jogo, onde:
   - Eventos normais seguem FIFO. Eventos prioritários seguem LIFO. Usando duas listas
   - `normal_events` → `["update player position", "detect collision", "generate particles"]`
   - `priority_events` → `["start cutscene", "pause", "boss arrival"]`
   - Usa uma unica deque para ambas

UNIVERSIDADE LUSÓFONA

Módulos

Módulos – Importação

```
from collections import deque
```

Acabámos de falar disto

```
stack = deque()
print(stack)
stack.append(5)
print(stack)
```

UNIVERSIDADE LUSÓFONA

Módulos

- Módulos são uma forma de organizar e agrupar código Python.
- Nós já usamos módulos:

```
from collections import deque
```

- Neste caso, específico a sintaxe apresentada diz que:
  - Estamos a importar o deque do módulo collections, que pertence à biblioteca standard do Python.
  - Apesar de fazer parte do Python requer importação.
  - Outros módulos pertencem a bibliotecas externas que têm que ser instaladas (ex: pygame).
- Vamos agora perceber um pouco melhor como funcionam.

UNIVERSIDADE LUSÓFONA

Importar um Módulo

- Importar um módulo permite-nos usar um módulo no nosso código.
- A maneira mais fácil de importar um módulo é:

```
import collections
```

- Isto vai procurar nos paths definidos o módulo (que é basicamente um ficheiro chamado collections.py) e carregá-lo para o nosso programa.
- Agora poderiamos usar tudo o que está no módulo collections?

```
import collections

stack = deque()
```

```
Traceback (most recent call last):
  File "c:\Users\filip\Desktop\Videojogos\2526\FP\Aulas\Aula6_fp.py", line 3, in <module>
    stack = deque()
            ^^^^^
NameError: name 'deque' is not defined
```

UNIVERSIDADE LUSÓFONA

Porquê?

- Com a instrução `import`, o módulo é importado para dentro de um namespace com o nome do módulo
- Como importámos o módulo collections, para usar o deque precisamos de preceder o nome da classe com o nome do módulo:

```
queue = collections.deque()
```

- Isto acontece para evitar colisões de nomes e garantir que as variáveis, funções, classes, etc, ficam bem definidas e sem ambiguidades.

```
import collections

stack = collections.deque()

print(stack)
```

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas>
as/Aula6_fp.py
deque([])
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas>
```

UNIVERSIDADE LUSÓFONA

Importar para dentro de um *namespace*

- Anteriormente nós usámos o deque sem usar o nome do módulo.
- Porque importámos uma parte específica do módulo para o namespace principal:

```
from collections import deque
```

- Com esta instrução, estamos a pedir ao Python para importar o elemento deque que está no módulo collections para o nosso namespace.
- Assim já podemos importar o *deque* directamente

```
queue = deque()
```

- Isto deve ser feito com cuidado, para evitar ambiguidades.

UNIVERSIDADE LUSÓFONA

Nota importante

- Reparem que se fizermos import collections podemos usar qualquer elemento do módulo collections:

```
queue = collections.deque()

dict = collections.OrderedDict()
```

```
deque([])
OrderedDict()
```

- Mas se fizermos from collections import deque, só podemos usar o deque:

```
from collections import deque

q = deque()
d = OrderedDict()
```

```
deque([])
Traceback (most recent call last):
  File "c:\Users\filip\Desktop\Videojogos\2526\FP\Aulas\Aula6_fp.py", line 11, in <module>
    d = OrderedDict()
        ^^^^^^^^^^^
NameError: name 'OrderedDict' is not defined
```

UNIVERSIDADE LUSÓFONA

Nota importante

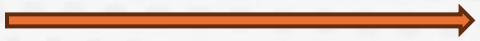
- Reparem que se fizermos import collections podemos usar qualquer elemento do módulo collections:

```
queue = collections.deque()

dict = collections.OrderedDict()
```

```
deque([])
OrderedDict()
```

- Mas se fizermos from collections import deque, só podemos usar o deque:

```
from collections import deque

q = deque()
d = collections.OrderedDict()
```



```
deque([])
Traceback (most recent call last):
  File "c:\Users\filip\Desktop\Videojogos\2526\FP\Aulas\Aula6_fp.py", line 11, in <module>
    d = collections.OrderedDict()
        ^^^^^^^^^^^
NameError: name 'collections' is not defined. Did you forget to import 'collections'?
```

UNIVERSIDADE LUSÓFONA

Nota importante

- É possível importar todos os elementos de um módulo para o nosso namespace:

```
from collections import *
```

Cuidado com este tipo de importação

UNIVERSIDADE LUSÓFONA

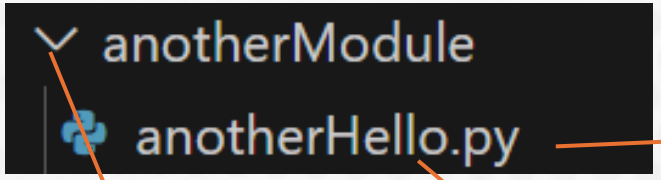
Criar um Módulo

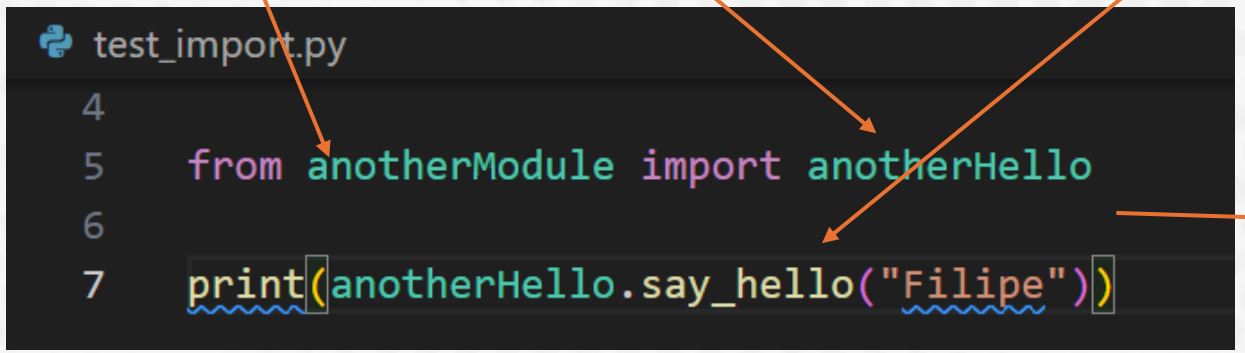
- Podemos também criar os nosso próprios módulos
- Isto pode ser especialmente util para tarefas especializadas
- Ou para reutilizar código antigo
- Basta criar um script em Python e importá-lo.

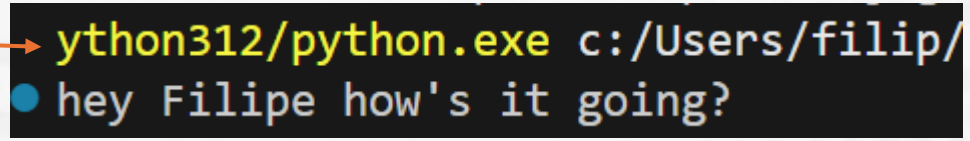
UNIVERSIDADE LUSÓFONA

Criar um Módulo numa subdirectoria

- Se quisermos colocar o nosso módulo numa subdirectoria, temos que colocar o nome da pasta no `from`







UNIVERSIDADE LUSÓFONA

Números Aleatórios

UNIVERSIDADE LUSÓFONA

Números aleatórios

- Um número aleatório é um número "ao calhas".
- Um computador não sabe fazer números aleatórios, por isso usa algoritmos sofisticados que simulam a aleatoriedade
- A maior parte desses algoritmos funciona da mesma forma na base.
  - Partindo de um número (*seed*/semente), vão gerando números a partir desse.
  - Sabendo a *seed*, consegue prever-se a sequência toda.
- Por isso chama-se a isto geradores de número pseudo-aleatórios
- O Python tem muitas formas diferentes de gerar números aleatórios, com características diferentes.

UNIVERSIDADE LUSÓFONA

Números aleatórios

- Provavelmente já ouviram falar de *seeds*

- Fixar a seed é extraordinariamente conveniente, especialmente na geração procedimental de conteúdo, etc...

UNIVERSIDADE LUSÓFONA

Números aleatórios

- Para usar random precisamos de importar o módulo

```
import random
```

- E usar a forma mais simples

```
value = random.random()
```

- Esta função devolve um número real no intervalo [0, 1[
  - Isto quer dizer que o 0 está incluído e o 1 não.

```
import random

for i in range(0,10):
    print(random.random())
```

```
0.29559921290667757
0.009623103551031797
0.1558852040710813
0.3388853515319593
0.4504301289172019
0.494529438127845
0.7858571829731964
0.38581355912041226
0.5796584631455504
0.052903978031756727
```

```
0.9135453980877453
0.01432958835843745
0.6029200754222841
0.7448870649789023
0.6070873286353555
0.9981387610271173
0.51430253581292
0.6869738600834214
0.8176342448814985
0.529172199917753
```

UNIVERSIDADE LUSÓFONA

Números aleatórios

- Se fixarmos a seed, com o `random.seed(n)`

```python
import random

random.seed(1234567890)

for i in range(0,10):
    print(random.random())
```

```
0.920682628327498
5
0.635100201969301
8
0.443521143639848
4
0.806884434812499
3
0.892684845284852
9
0.808130125003583
4
0.254900201284270
27
0.083954412050385
12
0.138534135176515
25
0.431728088558569
9
```

```
0.920682628327498
5
0.635100201969301
8
0.443521143639848
4
0.806884434812499
3
0.892684845284852
9
0.808130125003583
4
0.254900201284270
27
0.083954412050385
12
0.138534135176515
25
0.431728088558569
9
```

UNIVERSIDADE LUSÓFONA

Números aleatórios

- Existem funções mais complexas:

```
import random

value = random.uniform(x,y)
```

- Esta função devolve um número real no intervalo [x,y]
  - O último número pode ou não ser incluído no intervalo dependendo do sistema de arredondamento

```
random.random = random.uniform(0, 1)
```

  - O 1 nunca é incluído porque o arredondamento nunca vai arredondar para 1.

Números aleatórios

- Se precisarmos de escolher um elemento de uma lista de forma aleatória:

```
import random

choices = ["abc", "def", "ghi", "jkl", "mno", "pqr", "stu", "vwx", "yz"]

value = random.choice(choices)
```

- Esta função devolve um elemento aleatório da lista fornecida

```
import random
choices = ["abc", "def", "ghi", "jkl", "mno", "pqr", "stu", "vwx", "yz"]

for i in range(0,10):
    print(random.choice(choices))
```

```
stu
pqr
pqr
vwx
stu
abc
pqr
mno
vwx
vwx
```

Números aleatórios

- Se quisermos baralhar uma lista:

```
import random

choices = ["abc", "def", "ghi", "jkl", "mno", "pqr", "stu", "vwx", "yz"]

random.shuffle(choices)
```

```
import random
choices = ["abc", "def", "ghi", "jkl", "mno", "pqr", "stu", "vwx", "yz"]

for i in range(0,10):
    random.shuffle(choices)
    print(choices)
```

```
['jkl', 'mno', 'abc', 'def', 'pqr', 'stu', 'vwx', 'yz', 'ghi']
['mno', 'pqr', 'jkl', 'ghi', 'vwx', 'abc', 'yz', 'def', 'stu']
['abc', 'jkl', 'stu', 'pqr', 'def', 'yz', 'ghi', 'vwx', 'mno']
['mno', 'ghi', 'pqr', 'abc', 'vwx', 'jkl', 'def', 'yz', 'stu']
['pqr', 'mno', 'jkl', 'vwx', 'ghi', 'stu', 'yz', 'def', 'abc']
['yz', 'abc', 'ghi', 'vwx', 'pqr', 'stu', 'jkl', 'mno', 'def']
['jkl', 'def', 'abc', 'mno', 'pqr', 'ghi', 'vwx', 'stu', 'yz']
['abc', 'jkl', 'def', 'stu', 'vwx', 'pqr', 'mno', 'ghi', 'yz']
['ghi', 'vwx', 'yz', 'jkl', 'abc', 'pqr', 'def', 'stu', 'mno']
['yz', 'abc', 'vwx', 'jkl', 'def', 'stu', 'ghi', 'mno', 'pqr']
```

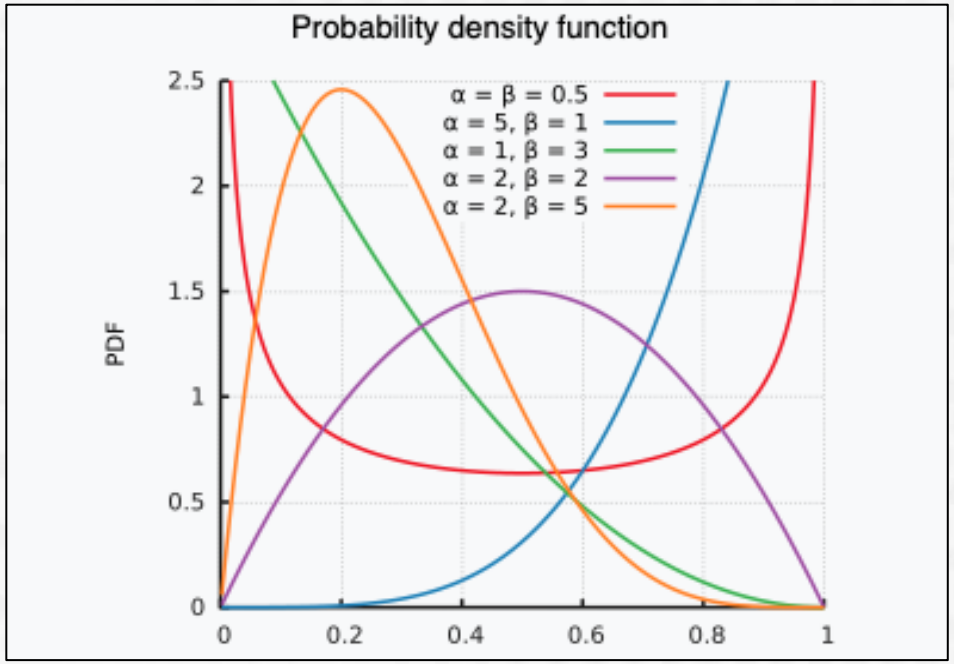
UNIVERSIDADE LUSÓFONA

Números aleatórios

- Existem outras funções para números aleatórios, com distribuições diferentes:
  - Como por exemplo a distribuição alpha ou beta:

```
import random

value = random.betavariate(alpha, beta)
```

- Mas estas são um bocado mais complexas



UNIVERSIDADE LUSÓFONA

Números aleatórios

- Podem ver mais detalhes sobre estas e outras funções na documentação do Python:

https://docs.python.org/3/library/random.html

UNIVERSIDADE LUSÓFONA

Exercícios

UNIVERSIDADE LUSÓFONA

Exercícios

1. Cria um módulo (`game_utils.py`) com 5 funções: saudação, classe, ataque, dano, e vida do inimigo (100)
   - Usa inputs no game loop
   - `player_class = [“warrior”, “mage”, “archer”]`
   - Dano base (40) com modifier baseado na classe = `[1.3, 1.5, 1.2]`
2. Nesse game_loop integra um loot system importado do `game_utils.py.`
   - Modifica o game loop para o jogo correr até o jogador saír
   - `loot = [“sword”, “shield”, “potion”, “armor”, “ring”]`
   - Quando o jogador derrota um inimigo, recebe um drop e adiciona ao inventário.
   - O inventário deve ser mostrado como stack
3. Integra um sistema aleatório de critical damage na função de dano no `game_utils.py`
   - Um critical hit de dobra o dano causado (base damage + class multiplier)
   - Existe 20% de possibilidade um critical hit
4. No `game_utils.py` cria um sistema de geração de diferentes inimigos
   - Lista de inimigos = `["Goblin", "Orc", "Troll", “Skeleton"]`
   - Eles têm força aleatória entre 5 e 20 (random.randint(a,b) → gera ints aleatórias no intervalo
   - Vida aleatória entre 30 e 100
5. Finalmente, no `game_utils.py,` gera um sistema de player health (100) e um sistema de defesa:
   - Se ambos atacam, ambos perdem health. Se um ataca e outro defende damage passa a metade.
   - Se o inimigo defende um ataque, o player perde a chance de critical hit
   - Fazer check do inventário não termina o turno
   - **O jogo agora deve correr até o jogador morrer ou saír.**

Aula 6
19/11/2025

Exercícios – Exemplos de execucão

1.

```
Enter your name: Filipe
Welcome, Filipe!
Available classes: ['warrior', 'mage', 'archer']
Choose your class: wa
Invalid choice. Please select one of: ['warrior', 'mage', 'archer']
Choose your class: ARCHER
You are a archer with base damage 40
You are now fighting a Training Dummy with 100 HP.
Type 'attack' to attack or 'exit' to quit: attack
You attack and deal 48 damage!
Enemy HP: 52
Type 'attack' to attack or 'exit' to quit: attack
You attack and deal 48 damage!
Enemy HP: 4
Type 'attack' to attack or 'exit' to quit: Attack
You attack and deal 48 damage!
Enemy HP: 0
You defeated the Training Dummy !
```

```
Enter your name: Filipe
Welcome, Filipe!
Available classes: ['warrior', 'mage', 'archer']
Choose your class: Warrior
You are a warrior with base damage 40
You are now fighting a Training Dummy with 100 HP.
Type 'attack' to attack or 'exit' to quit: exit
You decided to retreat.
```

2.

```
--- Game Start ---
Type 'attack' to fight, 'inventory' to check your items, or 'exit' to quit.


A new Training Dummy appears with 100 HP!
Command (attack/inventory/exit): inventory
Inventory is empty.
Command (attack/inventory/exit): attack
You attack and deal 52 damage!
Enemy HP: 48
Command (attack/inventory/exit): attack
You attack and deal 52 damage!
Enemy HP: 0
Enemy defeated!
You received: sword
Inventory (stack): ['sword']

A new Training Dummy appears with 100 HP!
Command (attack/inventory/exit): inventory
Current inventory (stack): ['sword']
Command (attack/inventory/exit):
```

3.

```
A new Training Dummy appears with 100 HP!
Command (attack/inventory/exit): attack
CRITICAL HIT!
You attack and deal 120 damage!
Enemy HP: 0
Enemy defeated!
You received: potion
Inventory (stack): ['potion']
```

4.

```
A Troll appears! (HP: 51 , Strength: 8 )
Command (attack/inventory/exit): attack
You attack and deal 48 damage!
Enemy HP: 3
Command (attack/inventory/exit): attack
You attack and deal 48 damage!
Enemy HP: 0
You defeated the Troll!
You received: armor
Inventory (stack): ['armor']

A Orc appears! (HP: 45 , Strength: 8 )
Command (attack/inventory/exit): exit

You chose to leave the battlefield.
Final inventory (stack): ['armor']
Game over.
```

5.

```
A Goblin appears! (HP: 88 , Strength: 7 )
Your action (attack/defend/inventory/exit): defend
Enemy action: defend
HP -> You: 73 | Goblin: 88
Your action (attack/defend/inventory/exit): defend
Enemy action: attack
Enemy dealt 3 damage to you.
HP -> You: 70 | Goblin: 88
Your action (attack/defend/inventory/exit): attack
Enemy action: defend
You dealt 26 damage.
HP -> You: 70 | Goblin: 62
Your action (attack/defend/inventory/exit): attack
Enemy action: defend
You dealt 26 damage.
HP -> You: 70 | Goblin: 36
Your action (attack/defend/inventory/exit): attack
Enemy action: defend
You dealt 26 damage.
HP -> You: 70 | Goblin: 10
Your action (attack/defend/inventory/exit): attack
Enemy action: attack
You dealt 52 damage.
Enemy dealt 7 damage to you.
HP -> You: 63 | Goblin: 0

You defeated the Goblin!
You received: ring
Inventory (stack): ['armor', 'ring']

A Goblin appears! (HP: 79 , Strength: 9 )
Your action (attack/defend/inventory/exit): attack
Enemy action: attack
You dealt 104 damage (CRITICAL).
Enemy dealt 9 damage to you.
HP -> You: 54 | Goblin: 0

You defeated the Goblin!
You received: shield
Inventory (stack): ['armor', 'ring', 'shield']

A Orc appears! (HP: 48 , Strength: 14 )
Your action (attack/defend/inventory/exit): attack
Enemy action: defend
You dealt 26 damage.
HP -> You: 54 | Orc: 22
Your action (attack/defend/inventory/exit): inventory
Inventory (stack): ['armor', 'ring', 'shield']
Your action (attack/defend/inventory/exit):
```