



Fundamentos de Programação (T)

Licenciatura em Videojogos
Ano 1 / Semestre 1



Sumário

- Funções como dados
- Lambdas
- Stacks
- Queues



Funções como dados

Funções como dados

- Em Python, funções são também dados

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

Quando fazemos isto, estamos a definir uma variável chamada `say_hello` que contém um valor que é a função em si.

```
print(say_hello)
```

Reparem que não estamos a chamar a função `say_hello` – isso seria `say_hello()` – estamos só a pedir ao Python para nos escrever que valor tem a variável `say_hello`.

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas> & C:/Users/filip.exe c:/Users/filip/Desktop/Videojogos/2526/FP/Aulas/aula5_fp.py  
<function say_hello at 0x000002C87ADC8A40>
```

Funções como dados

- Como podemos usar as funções como dados?

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

result = say_hello()
print(result)
```

O Python imprime o “Hello World” porque a função é chamada. Mas a variável result é nula.

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas>
aulas/aula5_fp.py
Hello World!
None
```

Funções como dados

- Como é que resolvemos isto?

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

```
result = say_hello()
print(result)
```

```
def say_hello():
    return "Hello World!"
```

```
result = say_hello()
print(result)
```

```
PS C:\Users\filip\Desktop\Vid
ulas/aula5_fp.py
> Hello World!
None
```

```
PS C:\Users\filip\Desktop\V
ulas/aula5_fp.py
> Hello World!
```

Funções como dados

- O facto de funções serem dados dá imenso jeito...
- Exemplo:

```
def square(array_of_values):
    ret = []
    for value in array_of_values:
        ret.append(value ** 2)
    return ret

def double(array_of_values):
    ret = []
    for value in array_of_values:
        ret.append(value * 2)
    return ret

values = [1,2,3]

print(square(values))
print(double(values))
```

Apenas estas duas linhas são diferentes

```
PS C:\Users\filip\Desktop\Videojogos\2526
ulas\aula5_fp.py
[1, 4, 9]
[2, 4, 6]
```



Funções como dados

- Neste exemplo temos duas funções praticamente idênticas
- O que significa que podemos otimizar ainda mais o código
- Se funções são dados, podemos escrever uma função que recebe uma outra função como parâmetro
- Assim podemos preencher as listas sem replicar código
- Para isso vamos definir primeiro as funções utilitárias:

```
def square(value):  
    return value ** 2  
  
def double(value):  
    return value * 2
```

Funções como dados

- E agora vamos definir a função que processa os dados:
- Tínhamos este código inicial: 
- E agora passamos a ter isto:

```
def square(array_of_values):  
    ret = []  
    for value in array_of_values:  
        ret.append(value ** 2)  
    return ret  
  
def double(array_of_values):  
    ret = []  
    for value in array_of_values:  
        ret.append(value * 2)  
    return ret
```

```
def process_values(array_of_values, process_function):  
    ret = []  
    for value in array_of_values:  
        ret.append(process_function(value))  
    return ret
```

Recebemos a função como uma variável normal

E usamos essa variável como se fosse uma função

Funções como dados

- E agora basta chamar a função:

```
values = [1,2,3]

print(process_values(values, square))
print(process_values(values, double))
```

Funções como dados

- O exemplo completo

```
def square(value):
    return value ** 2

def double(value):
    return value * 2

def process_values(array_of_values, process_function):
    ret = []
    for value in array_of_values:
        ret.append(process_function(value))
    return ret

values = [1,2,3]

print(process_values(values, square))
print(process_values(values, double))
```

```
PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas>
ulas/aula5_fp.py
[1, 4, 9]
[2, 4, 6]
```

Funções como dados

- Num dos exercícios da ultima aula tinhamos um sobre criar uma calculadora em que poderíamos ter usado esta lógica

```
# Operações
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b

def divide(a, b):
    if b == 0:
        print("Error: division by zero!")
        return None
    return a / b
```

Definiamos as operações

E usavamos essas funções como variáveis

```
# Seleção
def calculate(a, b, operation):
    if operation == "add":
        return add(a, b)
    elif operation == "subtract":
        return subtract(a, b)
    elif operation == "multiply":
        return multiply(a, b)
    elif operation == "divide":
        return divide(a, b)
    else:
        print("Unknown operation.")
        return None # equivalente a null em outras linguagens
```



Exercícios

Exercícios

```
execute_action(attack)
execute_action(defend)
execute_action(cure)
execute_action(inventory)
execute_action(flee)
```

1. Cria um sistema que executa as ações: ataque, defesa, cura, abrir inventário e fugir.
 - Usa uma função para gerir as ações.

```
• You attacked the enemy!
  You raised your shield!
  You gained 10 HP!
  You opened your inventory.
  You fled combat!
```

2. Usa uma lista `actions = []` para guardar as funções executá-las como no código do exercício anterior

```
• You attacked the enemy!
  You attacked the enemy!
  You gained 10 HP!
  You fled combat!
  You fled combat!
```

3. Define uma função que executa a mesma acção duas vezes usando o código do exercício anterior

4. Cria uma função que, conforme parametro recebido retorna uma função diferente. Depois, chama a função de return

5. Usando um dicionário. Cria um menu para executar o código todo do exercício usando input do jogador.

- Verifica se a opção existe
- Usa funções de string para limpar o input

```
menu = {
    "a": attack,
    "d": defend,
    "c": cure,
    "i": inventory,
    "f": flee
}
```

```
Select command a/d/c/i/f: a
You attacked the enemy!

Select command a/d/c/i/f: b
invalid option

Select command a/d/c/i/f: c
You gained 10 HP!

Select command a/d/c/i/f: d
You raised your shield!

Select command a/d/c/i/f: f
You fled combat!

PS C:\Users\filip\Desktop\Videojogos\2526\FP\Aulas>
```