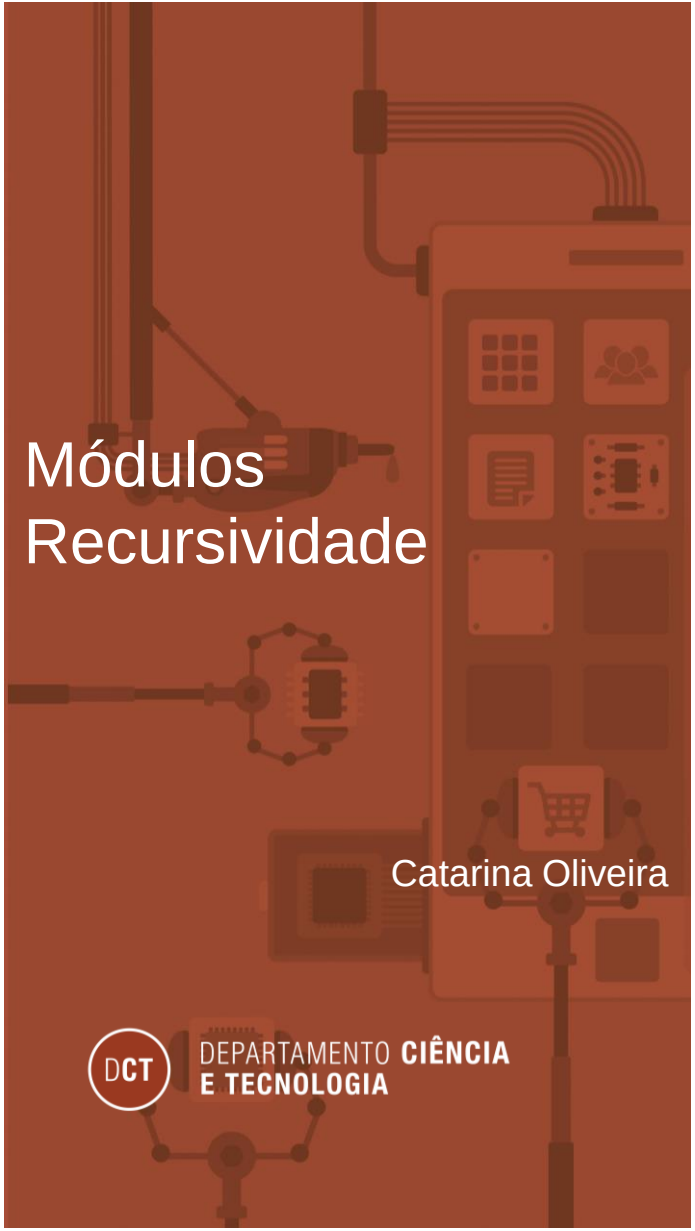

UNIVERSIDADE PORTUCALENSE

CONTEÚDO

1. Módulo
2. Exemplos:
    1. importar um módulo
    2. importar uma função de um módulo
    3. renomear um módulo
    4. organização de código de classes
3. _ _name _ _
4. Packages
5. Módulos pré-definidos
6. Recursividade

Módulo

- Ficheiro Python que contém funções que podem ser reutilizadas por outros ficheiros Python

Exemplo: importar um módulo

- Criar um ficheiro `funcoes.py` que contém uma função `soma(x,y)` que recebe dois valores e retorna a sua soma

```
def soma(x,y):
    return x+y
```

- Criar um ficheiro `main.py` que chama a função `soma(x,y)` importada a partir do ficheiro `funcoes.py`

```
import funcoes

print(funcoes.soma(2,3))
```

Exemplo: importar uma função de um módulo

- Cria a função `dobro(x)` no ficheiro `funcoes.py` em que retorna o dobro do valor que receber

```
def dobro(x):
    return 2*x
```

- Importar apenas a função dobro(x) no ficheiro `main.py`

```
from funcoes import dobro

print(dobro(2))
```

Exemplo: renomear um módulo

- O ficheiro `funcoes.py` contém as funções `soma(x,y)` e `dobro(x)`

```
def soma(x,y):
    return x+y

def dobro(x):
    return 2*x
```

- Usar as funções, renomeando o módulo

```
import funcoes as f
print(f.soma(2,3))
print(f.dobro(2))
```

Exemplo: organização de código de classes

- Com base no exemplo das classe e objetos, criar o ficheiro `funcionario.py` com a classe `Funcionario`

```python
class Funcionario:

    premioAnual = 1.5
    nFuncionarios = 0

    def __init__(self, primeiroNome, ultimoNome, salario):
        self.__primeiroNome = primeiroNome
        self.__ultimoNome = ultimoNome
        self.__salario = salario
        Funcionario.nFuncionarios += 1

    def __str__(self):
        return f"{self.nomeCompleto} [{self.email}]: {self.__salario}€"

    @property
    def nomeCompleto(self):
        return f"{self.__primeiroNome} {self.__ultimoNome}"

    @property
    def email(self):
        return self.__primeiroNome + "." + self.__ultimoNome + "@empresa.pt"


    def mostrarPremio(self):
        return Funcionario.premioAnual * self.__salario

    @staticmethod
    def boasVindas():
        return "Bem vindo à empresa"

    @staticmethod
    def mostrarProporcaoPremio():
        print(Funcionario.premioAnual)

    @nomeCompleto.setter
    def nomeCompleto(self, nome):
        primeiro, ultimo = nome.split(" ")
        self.__primeiroNome = primeiro
        self.__ultimoNome = ultimo

    @nomeCompleto.deleter
    def nomeCompleto(self):
        self.__primeiroNome = None
        self.__ultimoNome = None
```

Exemplo: organização de código de classes

- Criar o ficheiro `programador.py` (que importa a classe `Funcionario` do módulo `funcionario`) com a classe `Programador`

```python
from funcionario import Funcionario

class Programador(Funcionario):

    def __init__(self, primeiroNome, ultimoNome, salario, linguagem):
        super().__init__(primeiroNome, ultimoNome, salario)
        self.linguagem = linguagem

    def __str__(self):
        return f"{super().__str__()} => {self.linguagem}"
```

Exemplo: organização de código de classes

- Criar o ficheiro `main.py` que importa ambas as classes, e cria e imprime um funcionário e um programador

```
from funcionario import Funcionario
from programador import Programador

f1 = Funcionario("António", "Alves", 1000)
print(f1) # António Alves [António.Alves@empresa.pt]: 1000€
p1 = Programador("Bernardo","Bento",1000,"Python")
print(p1) # Bernardo Bento [Bernardo.Bento@empresa.pt]: 1000€ => Python
```

_ _name _ _

- Permite definir onde o código é executado

- Executar o ficheiro `modulo.py`

```
def funcao():
    if __name__=="__main__":
        print("dentro")
    else:
        print("fora")

funcao() # dentro
```

- Executar o ficheiro `main.py`, que importa o ficheiro `modulo.py`

```
def funcao():
    if __name__=="__main__":
        print("dentro")
    else:
        print("fora")
```

`modulo.py`

```
import modulo as m

m.funcao() # fora
```

`main.py`

Packages

- Em Python, podemos criar packages para organização de código

- Ao criar um package, é criado um ficheiro _ _init_ _.py,
  - Vazio
  - Assinala que a pasta é um package

Exemplo:

1. Criar um package chamado `auxiliar`
2. Mover o ficheiro `funcoes.py` para dentro do package
3. A chamada das funções passa a ser (automaticamente no IDE PyCharm):

```
from auxiliar import funcoes as f

print(f.soma(2,3))
print(f.dobro(2))
```

Exemplo:

1. Criar um package chamado `classesEmpresa`
2. Mover os ficheiros `funcionario.py` e `programador.py` para dentro do package
3. Imports atualizados

```
from classesEmpresa.funcionario import Funcionario
from classesEmpresa.programador import Programador

f1 = Funcionario("António", "Alves", 1000)
print(f1) # António Alves [António.Alves@empresa.pt]: 1000€
p1 = Programador("Bernardo","Bento",1000,"Python")
print(p1) # Bernardo Bento [Bernardo.Bento@empresa.pt]: 1000€ => Python
```

Módulos pré-definidos

- Exemplos:
  - math – funções matemáticas
  - numpy – cálculo de matrizes
  - pandas – tabelas e dados
  - sklearn – machine learning

- Lista de módulos predefinidos: https://docs.python.org/3/py-modindex.html

Recursividade

- Acontece quando uma função se chama a si própria
  - A função é recursiva

```
def recursiva(x):
    print(x)
    recursiva(x+1)

recursiva(1)
```



```
1
2
3
4

(...)

996
```

- A recursividade pode fazer com que se crie um ciclo infinito (o Python limita o nº de execuções)
  - Para saber o limite de recursividade:

```
import sys
print(sys.getrecursionlimit())
```

- É importante definir um **critério de paragem** (com um `if`)

```
def recursiva(x):
    if x<5:
        print(x)
        recursiva(x+1)

recursiva(1)
```



```
1
2
3
4
```

UNIVERSIDADE
PORTUCALENSE

Do conhecimento à prática.