UNIVERSIDADE PORTUCALENSE

CONTENT

1. Module
2. Examples:
    1. import a module
    2. import a function from a module
    3. rename a module
    4. class code organization
3. _ _ name _ _
4. packages
5. predefined modules
6. recursion

Module

- Python file that contains functions that can be reused by other Python files

Example: import a module

- `funcoes.py` file that contains a function `sum( x,y )` that takes two values and returns their sum

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

- `main.py` file that calls the `sum( x,y ) function` imported from the `funcoes.py file`

```
import funcoes

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

Example: import a function from a module

- `double(x)` function in the `funcoes.py` file which returns twice the value it receives

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

- Import only the double(x) function in the `main.py file`

```
from funcoes import dobro

print(dobro(2))
```

Example: Rename a module

- The `funcoes.py file` contains the `sum( x,y )` and `double(x) functions`

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

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

- Using the functions, renaming the module

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

Example: Class Code Organization

- Based on the example of the classes and objects, create the `employee.py file` with the `Employee class`

```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
```

Example: Class Code Organization

- `programmer.py` file (which imports the `Employee class from the employee` module) with the `Programmer class`

```
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}"
```

Example: Class Code Organization

- `main.py` file that imports both classes, and creates and prints an employee and a scheduler

```
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 _ _

- Lets you define where the code runs

- Run the `module.py file`

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

funcao() # dentro
```

- `main.py` file , which imports the `module.py`

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

`module.py`

```
import modulo as m

m.funcao() # fora
```

`main.py`

packages

- In Python, we can create packages for code organization

- init _ _ file is created . py ,
  - Empty
  - Indicates that the folder is a package

Example:

1. Create a package called `helper`
2. `funcoes.py` file into the package
3. Calling the functions becomes (automatically in the PyCharm IDE):

```
from auxiliar import funcoes as f

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

Example:

1. Create a package called `classesEmpresa`
2. `employee.py` and `programmer.py` files into the package
3. updated

```
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
```

predefined modules

- Examples:
  - math - math functions
  - numpy - matrix calculation
  - pandas - tables and data
  - sklearn – machine learning

- List of predefined modules: https://docs.python.org/3/py-modindex.html

recursion

- Happens when a function calls itself
  - The function is recursive

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

recursiva(1)
```



```
1
2
3
4

(...)

996
```

- Recursion can create an infinite loop (Python limits the number of executions)
  - To know the recursion limit:

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

- It is important to define a **stopping criterion** (with an `if`)

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

recursiva(1)
```



```
1
2
3
4
```

UNIVERSIDADE
PORTUCALENSE

Do conhecimento à prática.