

Programação

Objetos

Classes

Métodos

Fátima Leal



DEPARTAMENTO DE CIÊNCIA
E TECNOLOGIA



Até agora ...

- Fundamentos programação
- Variáveis, strings, listas, dicionários
- Condições e ciclos

Objetos

- Em Python, todos os valores são objetos: list, int, strings, *etc.*
- Cada objeto é mutável e tem:
 - um estado interno — informação acerca do próprio objeto, e que o distingue dos outros
 - uma coleção de métodos — ações que o objeto pode executar, possivelmente alterando o seu estado
- **Métodos:**
 - max()
 - min()
 - sum()
 - split()

Classes

- Classes são estruturas que permitem agrupar dados e métodos

Exemplo:

- Imagina uma empresa que pretende representar os seus funcionários no nosso código *Python*
 - Cada funcionário vai ter específicos atributos e métodos
 - Nome
 - Email
 - Salários
 - Etc.

Classes: Instâncias

- **Instâncias** de uma classe vão permitir aceder à estrutura definida na classe compartilhando todo o tipo de atributos e métodos.

Exemplo:

```
class Employee:  
    pass
```

```
emp_1 = Employee()  
emp_2 = Employee()
```

```
print(emp_1)  
print(emp_2)
```

Classes: Construtor

- O construtor permite definir atributos às instâncias

```
class Employee:
    def __init__(self, firstName, lastName, salary):
        #atributos das instancias
        self.firstName= firstName
        self.lastName= lastName
        self.salary= salary
        self.email = firstName + "." + lastName+"@company.com"
```

```
emp_1 = Employee("Diogo", "Gonçalves", 50000)
emp_2 = Employee("Pedro", "Marques", 100000)
```

```
print(emp_1)
print(emp_2)
```

Classes: Métodos

```
class Employee:
    def __init__(self, firstName, lastName, salary):
        self.firstName= firstName
        self.lastName= lastName
        self.salary= salary
        self.email = firstName + "." + lastName+"@company.com"
```

```
    def fullName(self): #método para construir o fullname
        return self.firstName + " " + self.lastName
```

```
emp_1 = Employee("Diogo", "Gonçalves", 50000)
emp_2 = Employee("Pedro", "Marques", 100000)
```

```
print(emp_1.fullName())
print(emp_1.firstName)
```

Uma vez que é um método
necessitamos dos parênteses

atributo

Classes: Variáveis das classes

```
class Employee:
    raise_amount = 1.04 # variáveis da classe
    def __init__(self, firstName, lastName, salary):
        self.firstName= firstName
        self.lastName= lastName
        self.salary= salary
        self.email = firstName + "." + lastName+"@company.com"

    def fullName(self):
        return self.firstName + " " + self.lastName

    def apply_raise(self): #método que utiliza a variável da classe
        return self.salary * self.raise_amount

emp_1 = Employee("Diogo", "Gonçalves", 50000)
emp_2 = Employee("Pedro", "Marques", 100000)
print(emp_1.apply_raise())
```


Classes: Variáveis das classes

```
class Employee:
    raise_amount = 1.04
    num_Employees = 0 #variável da classe
    def __init__(self, firstName, lastName, salary):
        self.firstName= firstName
        self.lastName= lastName
        self.salary= salary
        self.email = firstName + "." + lastName+"@company.com"
        Employee.num_Employees+=1 # diferença entre usar self e Employee

    def fullName(self):
        return self.firstName + " " + self.lastName

    def apply_raise(self):
        return self.salary * self.raise_amount

emp_1 = Employee("Diogo", "Gonçalves", 50000)
emp_2 = Employee("Pedro", "Marques", 100000)
print(Employee.num_Employees)
```

Classes: Exercício

- Considera que o teu programa recebe do utilizador as seguintes *strings*
 - “Pedro-Tavares-15000
 - “Tiago-Nunes-155000”
 - “Salome-Sol-100000”
- Cria instâncias da classe Employee que desenvolveste com os dados das strings.
- Repara que os atributos da instância estão todos na string separados por -

Classes: *staticmethod*

```
class Employee:
    num_Employees = 0
    def __init__(self, firstName, lastName, salary):
        self.firstName= firstName
        self.lastName= lastName
        self.salary= salary
        self.email = firstName + "." + lastName+"@company.com"
        Employee.num_Employees+=1

    def fullName(self):
        return self.firstName + " " + self.lastName

    @staticmethod
    def welcome(): # não contém referências às instâncias
        print("Welcome to the company!")
```

```
emp_1 = Employee("Diogo", "Gonçalves", 50000)
emp_2 = Employee("Pedro", "Marques", 100000)
Employee.welcome() #chama o static method
```

Staticmethod não tem referência aos elementos das instâncias

Não é necessário colocar o self como argumento dos métodos.


Classes: *Inheritances*

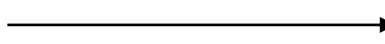
- Subclasses permitem criar instâncias mais específicas das super classes
- Exemplo:

A classe funcionário tem como atributos *firstName*, *lastName*, *salary* e *email*. Imagina que é necessário guardar a linguagem de programação que um *developer* trabalha.

Para isso, será necessário criar uma subclasse *Developer* que irá herdar todos os atributos da super classe *Employee*


Classes: *Inheritances*

```
class Employee:  superclasse
    def __init__(self, firstName, lastName, salary):
        self.firstName= firstName
        self.lastName= lastName
        self.salary= salary
        self.email = firstName + "." + lastName+"@company.com"
```


```
class Developer(Employee):  subclasse
    def __init__(self, first, last, salary, prog_lang):
        Employee.__init__(self,first, last, salary)
        self.prog_lang=prog_lang
```

```
emp_1 = Employee("Diogo", "Gonçalves", 50000)
emp_2 = Employee("Pedro", "Marques", 100000)
dev_1 = Developer("Teresa", "Neto", 150000, "Python")
print(dev_1.__dict__)
```

Herança de
atributos da
super classe



Imprime os dados da instância
dev_1 num dicionário



Subclasses: Exercício

- Cria a subclasse Manager que herda todos os atributos e métodos da superclasse Employee.
- A subclasse Manager necessita como adicional o Nome do departamento que gere
- Cria instâncias da subclasse Manager e imprime o seu conteúdo como um dicionário.

Classes: *atributos privados*

- Para maior segurança, os atributos do construtor podem ser protegidos.
- Para isso, é antecedido por um _

```
class Employee:
    def __init__(self, firstName, lastName, salary):
        self._firstName= firstName
        self._lastName= lastName
        self._salary= salary
        self._email = firstName + "." +
lastName+"@company.com"
```

Classes: Property Decorators

- Para definirmos getters and setters o Python disponibiliza Property Decorators
- Tomemos como exemplo o código inicial da classe Employee

```
class Employee:
    def __init__(self, firstName, lastName):
        self.firstName= firstName
        self.lastName= lastName
        self.email = firstName + "." + lastName+"@company.com"

    def fullName(self):
        return self.firstName + " " + self.lastName
```

- Para considerar o método fullname() como um atributo temos @property

Classes: property (getter)

```
class Employee:
    def __init__(self, firstName, lastName):
        self.firstName= firstName
        self.lastName= lastName

    @property
    def fullName(self):
        return self.firstName + " " + self.lastName

    @property
    def email(self):
        return self.firstName + "." + self.lastName+"@company.com"

emp1=Employee("Fatima", "Leal")
emp1.firstName="Luis"
print(emp1.email)
print(emp1.fullName)
```

→

Com o property o email e o fullName são atributos e não métodos.

Classes: setters

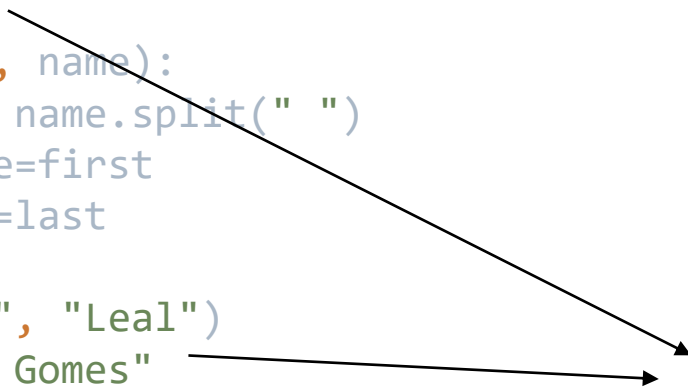
- Imagina que necessitas de modificar o fullname diretamente sem passar pelo atributo first and last. Para isso necessitamos de definir o fullname como um setter

... •

```
@property
def fullName(self):
    return self.firstName + " " + self.lastName
```

```
@fullName.setter
def fullName(self, name):
    first, last = name.split(" ")
    self.firstName=first
    self.lastName=last
```

```
emp1=Employee("Fatima", "Leal")
emp1.fullName = "Luis Gomes"
print(emp1.fullName)
print(emp1.firstName)
```



Fullname definido como um setter

Classes: deleters

- Para remover atributos, é necessário definir o mesmo como um deleter.

...

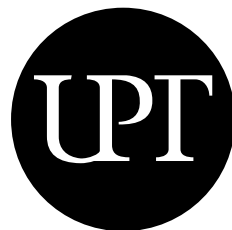
```
@property
def fullName(self):
    return self.firstName + " " + self.lastName
```

```
@fullName.deleter
def fullName(self):
    print("Delete Name!")
    self.firstName=None
    self.lastName=None
```

```
emp1=Employee("Fatima", "Leal")
```

```
del emp1.fullName
print(emp1.firstName)
```

Fullname definido como
um deleter



UNIVERSIDADE
PORTUCALENSE

Do conhecimento à prática.