Classes em Python

Mário Leite

...

Uma das características, e vantagens, da Tecnologia de Orientação a Objeto é o fato do programador poder criar novos tipos de dados chamados, justamente, de “objetos”. Isto é, através de um “molde” (classe) é possível criar esses entes dinâmicos que passam a possuir características desse molde; é como se fossem “filhos” herdando os genes do pai (ou da mãe).

As linguagens que suportam essa tecnologia são as chamadas LOO (ou OOL em Inglês): “**Linguagens Orientadas a Objeto**”. Esta característica de “transmitir características” é o fenômeno da “herança“, e um dos “filhos” pode ser até um outro “molde”: uma nova classe que herda TUDO da classe-mãe, podendo incluir novas características próprias. É como um filho herdando TODAS as características do pai e ainda podendo adquirir outras que não existia no seu genitor; por exemplo, um pai que não entende nada de música, mas o filho pode ser um exímio violinista, adquirindo posteriormente essa habilidade sem depender do pai...

As linguagens “Procedurais” (não orientadas a objetos) não possuem esse recurso de “criar objetos”, mas por outro lado podem criar estruturas muito parecidas com classes que facilitam o manuseio de “variáveis” ligadas a um tipo de dado, também conhecidas como “registo”. Isto também facilita a programação quando é preciso fazer ligações (referências) à uma estrutura de dados. O Python, embora seja uma LOO, também possui características de “linguagem procedural/multiprograma”; e tal como o Pascal, um dos seus recursos é criar “records” (registros). Um registro funciona tal como uma classe em LOO, embora não seja uma classe no sentido real do termo, pois não possui a capacidade de criar ações (métodos). Do ponto de vista de codificação um *registro* é um recurso muito eficiente nos programas quando se deseja trabalhar com variáveis que são de um determinado tipo de dado. Abaixo estão duas maneiras de criar um *registro* que representa os dados de um funcionário...

Em Pascal:

```
Type TFuncionario = record
     Codigo = integer;
     Nome = string[40];
     Sexo = char;
     Nasc = string[10];
     Sal = real;
end;
```

Em Python:

```
class TFuncionario:
    def __init__(self, Codigo, Nome, Sexo, Nasc):
        self.code = Codigo
        self.nome = Nome
        self.sexo = Sexo
        self.nasc = Nasc
        self.sal =  Salario
```

O programa **“CriaClasse”** mostra como usar esse tipo de estrutura em Python, num exemplo bem simples. Observe que é bem parecido com uma codificação em Pascal para criar e usar os elementos de um registro. A **figura 1** mostra um exemplo de saída deste pro0grama.

==========

```
'''
CriaClasse.py
------------------------------------------------------------------------
Mostra como trabalhar com registros em Python.
------------------------------------------------------------------------
Autor: Mário Leite
Data: 10/09/2023
------------------------------------------------------------------------
'''
class TFuncionario:
    def __init__(self,Codigo=None,Nome=None,Sexo=None,Nasc=None,Salario=5000):
        self.code = Codigo
        self.nome = Nome
        self.sexo = Sexo
        self.nasc = Nasc
        self.sal = Salario

#Cria uma instância da classe TFuncionario sem fornecer os parâmetros obrigatórios
Func = TFuncionario()

#Define os valores dos atributos diretamente na instância
Func.code = 123
Func.nome = "Mário de Campos Leite"
Func.sexo = "M"
Func.nasc = "06/10/1948"
Func.sal = (Func.sal*1.08)  #calcula o atributo 'sal' com aumento de 8%

#Mostra os valores dos dados do funcionário
print(Func.code)
print(Func.nome)
print(Func.sexo)
print(Func.nasc)
print("fR${Func.sal}")
#Fim do programa "CriaClasse"----------------------------------------------
```

```
Python 3.10.0 (tags/v3.10.0:b494f59, Oct  4 2021, 19
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" f

========== RESTART: D:/TesteXY.py
123
Mário de Campos Leite
M
06/10/1948
R$ 5400.0
```

Figura 1-– Saída do programa “CriaClasse”