Objetos em Python

Genericamente um paradigma é um modelo que mostra como as coisas são feitas. Na programação, um paradigma é uma maneira como se aborda e resolve um problema. Seguindo esta definição, na programação temos alguns paradigmas:

imperativo O inicial, mais simples e direto e usualmente usado como primeiro: aqui se diz ao computador o que fazer e quando. Ótimo para problemas simples, começa a se tornar inadequado para grandes e complexos projetos. A quantidade de interações e condições rapidamente tende ao ingovernável.

estruturado Uma sub-especificação do paradigma imperativo no qual está proibido o desvio irrestrito e ele é substituído pelas estruturas de repetição que vem a ser desvios mais *bem comportados*.

funcional Um paradigma maravilhoso: nele toda a funcionalidade do software deve ser tratada como a matemática trata suas funções, através da imagem e do domínio. (Relembrando em $y = f(x)$ os valores de x são o domínio e os de y são a imagem da função). Aqui não há contacto entre y e x e este fato diminui o efeito colateral responsável pela maioria dos erros de programação que é o compartilhamento de variáveis globais.

declarativo Um nicho especializado usado em engenhos de I.A. usados para gerar e manusear conhecimento.

orientado a objetos Mais um passo na direção do encapsulamento de dados. Agora as entidades do software protegem sofregamente seus dados e impedem agentes externos de manuseá-los. De quebra este paradigma tende a se aproximar da realidade que descreve os elementos que o software modeliza. Novamente não há variáveis globais: o que há são mensagens trocadas entre objetos.

Não é interessante tratar cada paradigma isoladamente. Fazer isso seria artificializar a solução. O que há na prática é uma certa salada na qual se apanha de cada paradigma o que se precisa e se quer. Um exemplo disso é dada pelo Python. Como uma linguagem moderna (recente) ele percorre os paradigmas acima airosamente. Pode ser usada como imperativa (sem o desvio que inexiste, portanto melhor dizer estruturada), como funcional e finalmente como O.O. (orientada ao objeto). É esta última característica que vai ser estudada aqui.

Objeto desde Aristóteles, nosso conhecimento do mundo parece surgir ao manusear objetos e suas definições (segundo ele um objeto é a soma de *genus* e *diferentia*). Somos rodeados por objetos: celular, agendas, aulas, livros, estradas, amores, fome, deslumbramentos, etc, etc. Formalmente falando um objeto é uma entidade do universo, caracterizado por

estado a situação na qual o objeto se encontra

comportamentos o que o objeto sabe fazer

Esta definição é integralmente introduzida no mundo do software: um objeto aqui é um elemento do software que tem um nome, algumas variáveis internas ao qual só ele tem acesso (estado) e algumas funções que manuseiam os dados de entrada, internos e de saída (comportamento).

As 4 pernas Se a O.O. for uma mesa retangular, ela se sustenta sobre 4 pernas que são:

- Abstração
- Encapsulamento
- Herança
- Polimorfismo

Abstração Um modelo é descrito abstratamente com a cláusula `class`. Nela se descreve o nome, as variáveis internas e as funções do objeto. A classe não é diretamente manuseável, mas ela serve como modelo para a criação de objetos reais. Aqui a abstração. Há um método particular denominado **construtor** que é chamado sempre que um novo objeto é criado. Exemplo:

```
class <nome>
   atributos
   construtor
   métodos
```

Define-se como instância a criação de um objeto a partir de sua classe.

```
variável = Classe()
```

Veja um exemplo completo

```
class gato():
  patas = 4
  nome = None
  def miar(self):
    print('miau')
b = gato()
b.nome = "Binho"
print(b.patas)
b.miar()
```

O construtor sempre tem o nome de `__init__(self,...)` e ele é chamado quando a instância ocorre. Veja um exemplo completo

```
class Conta :
  numero = None
  saldo = None
  def __init__ (self , numero):
    self.numero = numero
    self.saldo = 0.0
  def consultar(self):
    return self.saldo
  def creditar(self,valor):
    self.saldo = self.saldo + valor
  def debitar(self,valor):
    self.saldo = self.saldo - valor
  def transferir (self,conta,valor):
    self.debitar(valor)
    conta.creditar(valor)
conta1 = Conta (1)
conta1 . creditar (100)
conta2 = Conta (2)
conta2 . creditar (51)
print(conta1.consultar())
print(conta2.consultar())
conta1.transferir(conta2,13)
print(conta1.consultar())
```

Encapsulamento Tanto as variáveis como as funções são de uso exclusivo do objeto. Ninguém mais pode meter a mão lá. Esta a grande qualidade da O.O. Em Python existem 2 tipos de modificadores de acesso para atributos e métodos. O público e o privado. Atributos ou métodos iniciados por dois sublinhados são privados. Todos os outros são públicos

```
class Conta :
def __init__ (self , numero ):
self . __numero = numero
self . __saldo = 0.0
def consultar_saldo ( self ):
return self . __saldo
def creditar (self , valor ):
self . __saldo += valor
def debitar (self , valor ):
self . __saldo -= valor
def transferir (self , conta , valor ):
self . __saldo -= valor
conta . __saldo += valor
conta = Conta (1)
conta . creditar (100)
conta . __saldo = 200.0
    #Não é possível alterar o saldo da conta
print ( conta . consultar_saldo ())
```

Herança Novamente voltando a Aristóteles, nosso conhecimento de mundo exige uma hierarquia de objetos: o Binho é um gato. (Binho → gato → animal). Tudo o que for compartilhado deve ser descrito no mais alto nível possível. A classe definida pode herdar atributos e métodos de uma ou mais classes, usando o comando

```
class <nome>(classe-pai1, classe-pa2, ...):

class Poupanca ( Conta ):
def __init__ (self , numero ):
super (). __init__ ( numero )
self . __rendimento = 0.0
def consultar_rendimento ( self ):
return self . __rendimento
def gerar_rendimento (self , taxa ):
self.__rendimento + =
   super().consultar_saldo()*taxa/100
conta = Poupanca (1)
conta . creditar (200.0)
conta . gerar_rendimento (10)
print ( conta . consultar_saldo ())
print ( conta . consultar_rendimento ())
```

Polimorfismo O mesmo método pode fazer coisas diferentes dependendo da maneira como for chamado.

Instância Uma instância é um objeto criado com base em uma classe anterior. A classe é apenas um esqueleto e não pode ser usada diretamente, mas ela serve para criar objetos reais. Dentro de uma classe o construtor informa o que deve ser feito quando o objeto é criado. O construtor tem pelo menos um parâmetro (idealmente chamado de `self`), mas pode ter outros.

Métodos Representam o comportamento da classe. Eles acessam os atributos tanto para leitura como para gravação. Podem retornar ou não algum valor e podem possuir (ou não) parâmetros, além so `self`.

☞ Para você fazer

1. Crie a classe `point` em duas dimensões. Depois escreva a função distancia que recebe dois pontos e devolve a distância euclidiana entre eles.

2. Crie a classe `point` em duas dimensões. A equação de uma reta é $y = ax + b$. Escreva um método na classe point que receba outra instância de point e devolva os coeficientes a e b da reta que liga os 2 pontos.