Aula 9 - Orientação a Objetos

June 10, 2021

1 Orientação a Objetos

1.1 Classes

Uma classe define uma estrutura de dados que conheta instâncias de atributos, instâncias de métodos e classes aninhadas.
Classes são estruturas de dados que possuem Atributos e Métodos para virem a serem implementadas por futuros objetos.
Classes, então, são representações computacionais de algo que queremos introduzir no nosso código.

[1]:
```
# Classes -> Representação
 # Atributos -> Variáveis
 # Métodos -> Funções
```

[10]:
```
# Classe Carro:
    # Modelo
    # Ano
    # Estado

    # liga_desliga()
    # acelerar()
    # test_drive()
    # comprar()


# Modelo, Ano e Estado => São Atributos
# Ligar e Desligar, Acelerar, Fazer Test Drive e Comprar => São Métodos
```

1.2 Objetos

Um objeto em linguagem de programação representa a posição onde será armazenada uma instancia daquela classe.
Objetos são instâncias (versões/criações/representação) da classe (o que o objeto tenta representar).
Todo objeto possui um tipo.

[13]:
```
# carro_ferrari
   # Modelo  -> Ferrari
   # Ano     -> 2019
   # Estado  -> Novo
```

```
   # liga_desliga() -> Método para ligar e desligar o carro
   # acelerar() -> Método para acelerar
   # test_drive() -> Método para testar o carro
   # comprar() -> Método para comprar o carro

# carro_bmw
   # Modelo -> BMW
   # Ano    -> 2016
   # Estado -> Semi-Novo

   # liga_desliga() -> Método para ligar e desligar o carro
   # acelerar() -> Método para acelerar
   # test_drive() -> Método para testar o carro
   # comprar() -> Método para comprar o carro
```

1.3 Resumindo

[17]:
```
# Classe = Carro
  # Atributos = Modelo , Ano e Estado
  # Métodos = liga_desliga() , acelerar(), test_drive() e comprar()

# Objeto = Instancias da Classe (representações/variaveis da classe) =>␣
 ↪Variável do tipo carro
```

[18]:
```
# Classes são representações computacionais de algo que queremos vir a utilizar␣
 ↪no decorrer do nosso código.
# Onde representamos os atributos e métodos de acordo com a nossa necessidade.

# Já os objetos são as nossas variaveis desse tipo de classe.
```

1.4 Exemplificando

[21]:
```
class Carro(object):
    estado = 'novo'

fusca = Carro() # Objeto
fusca.estado = 'novo' # Atributo

ferrari = Carro() # Objeto
ferrari.estado = 'usado' # Atributo

print('O valor do estado do fusca é',fusca.estado) # Valor do Atributo
print('O valor do estado da ferrari é',ferrari.estado) # Valor do Atributo
print('O tipo do fusca é',type(fusca)) # Tipo do Objeto
print('O tipo da ferrari é',type(ferrari)) # Tipo do Objeto
```

```
print('O tipo do atributo \'estado\' do fusca é',type(fusca.estado)) # Tipo do␣
 ↪Atributo
print('O tipo do atributo \'estado\' da ferrari é',type(ferrari.estado)) # Tipo␣
 ↪do Atributo
```

```
O valor do estado do fusca é novo
O valor do estado da ferrari é usado
O tipo do fusca é <class '__main__.Carro'>
O tipo da ferrari é <class '__main__.Carro'>
O tipo do atributo 'estado' do fusca é <class 'str'>
O tipo do atributo 'estado' da ferrari é <class 'str'>
```

2 Vamos deixar mais formal?

2.1 Orientação a objeto

Um objeto em linguagem de programação abstrata representa a posição onde será armazenada uma instancia (versão) daquela classe (o que o objeto representa). Os objetos em Python apresentam os seguintes atributos:

- Tipo: O tipo de um objeto determina os valores que o objeto pode receber e as operações que podem ser executadas nesse objeto.
- Valor: O valor de um objeto é o índice de memória ocupada por essa variável. Como os índices das posições da memória são interpretados, isto é determinado pelo tipo da variável.
- Tempo de vida: A vida de um objeto é o intervalo de tempo de execução de um programa em Python, é durante este tempo que o objeto existe.

Python define uma extensa hierarquia de tipos. Esta hierarquia inclui os tipos numéricos (tais como int, float e complex), seqüências (tais como a tupla e a lista), funções (tipo função), classes e métodos (tipos classobj e instancemethod), e as instâncias da classe (tipo instance).

2.2 Classes

Uma classe define uma estrutura de dados que contenha instância de atributos, instância de métodos e classes aninhadas. Em Python a classe de um objeto e o tipo de um objeto são sinônimos. Cada objeto do Python tem uma classe (tipo) que é derivada diretamente ou indiretamente da classe interna do objeto do Python. A classe (tipo) de um objeto determina o que é e como pode ser manipulado. Uma classe encapsula dados, operações e semântica.

A classe é o que faz com que Python seja uma linguagem de programação orientada a objetos. Classe é definida como um agrupamento de valores sua gama de operações. As classes facilitam a modularidade e abstração de complexidade. O usuário de uma classe manipula objetos instanciados dessa classe somente com os métodos fornecidos por essa classe.

Frequentemente classes diferentes possuem características comuns. As classes diferentes podem compartilhar valores comuns e podem executar as mesmas operações. Em Python tais relacionamentos são expressados usando derivação e herança.

2.2.1 Instâncias, Instância de Atributos e Métodos

Objetos são instanciados pelas classes. Cada instância (objeto) em uma programa Python tem seu próprio namespace.

Um classe criada é chamada de classe objeto (tipo classobj). Os nomes no namespace da classe objeto são chamados de atributos da classe. Funções definidas dentro de uma classe são chamadas de métodos.

Quando um objeto é criado, o namespace herda todos os nomes do namespace da classe onde o objeto está. O nome em um namespace de instância é chamado de atributo de instância.

Um método é uma função criada na definição de uma classe. O primeiro argumento do método é sempre referenciado no início do processo. Por convenção, o primeiro argumento do método tem sempre o nome self. Portanto, os atributos de self são atributos de instância da classe.

3 Pronto! Agora vamos voltar ao informal :)

3.1 Instâncias Abertas

Nem sempre precisamos definir tudo direto na classe.
Uma classe pode ter suas propriedades definidas diretamente nos objetos.
Dessa forma, os atributos são inseridos dinamicamente nos objetos.

[23]:
```
class Carro(object):
    pass
```

[24]:
```
carro = Carro()
type(carro)
```

[24]:
```
__main__.Carro
```

[31]:
```
fusca = Carro()
fusca.estado = 'novo' # Atributo da Instância
fusca.multas = 12 # Atributo da Instância
print(fusca.estado)
print(fusca.multas)
```

```
novo
12
```

[30]:
```
ferrari = Carro()
print(ferrari.estado) # Lembrete que o Carro não possui por padrão essa␣
 ↪propriedade,
                      #  então não podemos acessar a menos que criemos essa␣
 ↪propriedade
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-30-174cd454ec46> in <module>
```

```
      1 ferrari = Carro()
----> 2 print(ferrari.estado) # Lembrete que o Carro não possui por padrão essa
 ↪propriedade, então não podemos acessar a menos que criemos essa propriedade

AttributeError: 'Carro' object has no attribute 'estado'
```

3.2 Atributos de Classe

É um atributo que fica diretamente na classe.
Sendo assim, todos os objetos dessa classe terão essa propriedade (atributo).

[34]:
```
class Carro(object):
    estado = 'novo'

print(Carro.estado) # Atributos de classe podem ser acessados diretamente com a␣
 ↪classe (sem precisar de objetos)
```

```
novo
```

Note bem o código acima, reparou que não precisamos instanciar a classe ?
Simplesmente acessamos o atributo (que é da classe) diretamente Carro.estado.

Por outro lado, repare que a propriedade também se encontra disponível nas instâncias.

[41]:
```
fusca = Carro()
bmw = Carro()
ferrari = Carro()

print(fusca.estado)
print(bmw.estado)
print(ferrari.estado)
```

```
novo
novo
novo
```

[42]:
```
fusca.estado = 'usado'
bmw.estado = 'semi-novo'

print(fusca.estado)
print(bmw.estado)
print(ferrari.estado)
print(Carro.estado)
```

```
usado
semi-novo
novo
novo
```

Em Python precisamos atentar que “as coisas” ou são da classe ou são da instância.

3.3 Self

Para editarmos uma propriedade apenas do objeto que chama um método usamos a palavra reservada 'self'.

Portanto todo método deve o parâmetro 'self' e esse ser o primeiro

[45]:
```
class Carro(object):
    estado = 'novo'

    def dirigir(self):
        self.estado = 'usado' # Modifica apenas no objeto que chamar o método

# Exemplo 1
bmw = Carro() # Criou com o estado novo
bmw.dirigir() # Modifiquei o estado para usado
print(bmw.estado) # Passa a ser usado

# Exemplo
ferrari = Carro() # Criou com o estado novo
print(ferrari.estado) # Continua como novo
```

```
usado
novo
```

Caso não utilizarmos o self... Coisas diferentes do que esperamos podem acontecer

[46]:
```
class Carro(object):
    estado = 'novo'

    def dirigir(self):
        estado = 'usado' # Modifica apenas no objeto que chamar o método

# Exemplo 1
bmw = Carro() # Criou com o estado novo
bmw.dirigir() # Modifiquei o estado para usado
print(bmw.estado) # Passa a ser usado

# Exemplo
ferrari = Carro() # Criou com o estado novo
print(ferrari.estado) # Continua como novo
```

```
novo
novo
```

[47]:
```
class Carro(object):
    estado = 'usado'

    def dirigir(self):
        Carro.estado = 'usado' # Modifica apenas no objeto que chamar o método
```

```
# Exemplo 1
bmw = Carro() # Criou com o estado novo
bmw.dirigir() # Modifiquei o estado para usado
print(bmw.estado) # Passa a ser usado

# Exemplo
ferrari = Carro() # Criou com o estado novo
print(ferrari.estado) # Continua como novo
```

```
usado
usado
```

3.4 Diferenciando atributos da classe com atribudos da instância

[49]:
```
class Carro(object):
    estado = 'novo'

carro1 = Carro()
print('1:',carro1.estado)
carro1.estado = 'semi-novo' # Neste momento criamos um atributo de INSTÂNCIA

carro2 = Carro()

print('1:',carro1.estado)
print('2:',carro2.estado)

print('Carro:',Carro.estado)
Carro.estado = 'quebrado'
print('Carro:',Carro.estado)

carro3 = Carro()

print('1:',carro1.estado) # Carro 1 possui um atributo de INSTÂNCIA
print('2:',carro2.estado) # Carro 2 possui um atributo de CLASSE
print('3:',carro3.estado) # Carro 3 possui um atributo de CLASSE
```

```
1: novo
1: semi-novo
2: novo
Carro: novo
Carro: quebrado
1: semi-novo
2: quebrado
3: quebrado
```

3.5 Construtor de classe (init)

O constutor de calsse no Python é chamado de **init** e ele é usado para criar uma nova instância do objeto.
Caso usarmos algum parâmetro no construtor, torna-se obrigatório o envio desse parâmetro na hora da criação do objeto.
Com isso, podemos criar valores padrões para nossos objetos.

[65]:
```
# Criar + Modifica  => Cada um já tem seu próprio estado inicial

class Carro(object):
    def __init__(self, parametro_estado): # Constutor
        self.estado = parametro_estado

bmw = Carro('semi-novo')
ferrari = Carro('novo')
fusca = Carro('usado')

print("A BMW tem o estado:",bmw.estado)
print("A Ferrari tem o estado:",ferrari.estado)
print("O Fusca tem o estado:",fusca.estado)
```

```
A BMW tem o estado: semi-novo
A Ferrari tem o estado: novo
O Fusca tem o estado: usado
```

4 Desafio 9.1 - Desafio do Carro Nomeado

Criar uma classe Carro onde além de estado também tem um nome por padrão (Construtor).
Criar um método nessa classe Carro onde vai printar para gente o Nome e o Estado do carro.

[75]:
```
# método Printar() => Chamar por um carro
    # Retorna um print "O carro {Nome} tem o estado {Estado}"
         # O carro BMW tem o estado Semi-Novo
class Carro(object):
    def __init__(self, estado, nome): # Constutor
        self.estado = estado
        self.nome = nome

    def mostrar_informacoes(self):
        print(f"O carro {self.nome} tem o estado {self.estado}")

bmw = Carro('semi-novo','BMW')
ferrari = Carro(nome='Ferrari',estado='novo') # Parâmetro passado de forma␣
 ↪nomeada
fusca = Carro(estado='usado',nome='Fusca')
```

[76]:
```
bmw.mostrar_informacoes()
ferrari.mostrar_informacoes()
fusca.mostrar_informacoes()
```

```
O carro BMW tem o estado semi-novo
O carro Ferrari tem o estado novo
O carro Fusca tem o estado usado
```

4.1 Desafio ByLearner

[149]:
```
# Classe Carro:
    # Modelo => Definidos pelo construtor ("Quem define é a loja (classe)") => Tem um valor inicial para cada um
    # Ano => Definidos pelo construtor ("Quem define é a loja (classe)") => Tem um valor inicial para cada um
    # Estado => Definidos pelo construtor ("Quem define é a loja (classe)") => Tem um valor inicial para cada um
    # Comprado => Vai ser Falso para todos => Valor inicial Padrão => É uma variáel de classe (e não de instância)

    # liga_desliga() => Liga e Desliga o carro => Obrigatório antes de dirigir/fazer test drive
    # acelerar() => Só pode acelerar depois de comprar o carro... Mas vai tentar mesmo assim
    # test_drive() => Só pode fazer antes de comprar => Não posso acelerar
    # comprar() => Só pode comprar uma vez
    # dirigir() => Você pode somente depois de comprar o carro => Pode acelerar


# Modelo, Ano e Estado => São Atributos de Instância (Construtor)
# Comprado => Atributo da Classe (valor padrão inicial)
# Ligar e Desligar, Acelerar, Fazer Test Drive, Comprar e Dirigir => São Métodos
```

[152]:
```
class Carro(object):
    comprado = False

    def __init__(self,modelo,ano,estado):
        self.modelo = modelo
        self.ano = ano
        self.estado = estado

    def comprar(self):
        if(self.comprado):
            print("Você já comprou, não pode comprar novamente!")
            return # Interrompe o ciclo / Para a execução ('estilo um break')
```

```
        self.comprado = True
        print("Você comprou o carro")

    def test_drive(self):
        if(not self.comprado):
            print("Você vai fazer o test drive")
            self.liga_desliga(True)
            print("Você está fazendo o test drive")
            if(self.acelerar()):
                print("Você está acelerando")
            else:
                print("Você não pode acelerar")
            self.liga_desliga(False)
            print("Você terminou de fazer o test drive")
        else:
            print("Você não pode fazer o test drive pois comprou o carro, vá␣
↪dirigir!")

    def dirigir(self):
        if(self.comprado):
            print("Você vai dirigir")
            self.liga_desliga(True)
            print("Você está dirigindo")
            if(self.acelerar()):
                print("Você está acelerando")
            else:
                print("Você não pode acelerar")
            self.liga_desliga(False)
            print("Você terminou de dirigir")
        else:
            print("Você só pode dirigir se comprar o carro!")

    def acelerar(self):
        return self.comprado

    def liga_desliga(self, status):
        if status: # Booleano
            print("Você ligou o carro")
        else:
            print("Você desligou o carro")
```

[155]:
```
ferrari = Carro('Ferrari','2019','Novo')
bmw = Carro(ano='2017',estado='Semi-Novo',modelo='BMW')
```

[165]:
```
ferrari.dirigir()
```

```
Você vai dirigir
Você ligou o carro
```

```
Você está dirigindo
Você está acelerando
Você desligou o carro
Você terminou de dirigir
```

4.2 Desafio 9.2 Calculadora

Classe Calculadora que contém os seguintes métodos:
- somar - subtrair - multiplicar - dividir

PS: Esses métodos funcionam apenas com dois números
PS2: Todos os métodos retornam o valor

A classe também contém as propriedades (atributos):
- primeiro_valor - segundo_valor

PS3: Os valores são passados em parâmetro
PS4: Os valores vão ser lidos do usuário (input)

[174]:
```
class Calculadora(object):
    def soma(self,primeiro_valor,segundo_valor):
        return primeiro_valor + segundo_valor;

    def subtracao(self,primeiro_valor,segundo_valor):
        return primeiro_valor - segundo_valor

    def divisao(self,primeiro_valor,segundo_valor):
        return primeiro_valor / segundo_valor

    def multiplicacao(self,primeiro_valor,segundo_valor):
        return primeiro_valor * segundo_valor
```

[175]:
```
calc = Calculadora()
calc.multiplicacao(4,2)
```

[175]: 8

4.3 Desafio 9.3 Media do Aluno

Classe Aluno possui os atributos:
- Nome => string - Status => Aprovado ou Não Aprovado (booleano) - Nota1 => float - Nota2 => float - Media => float

Classe também possui um método:
- Mostrar Informações (mostrar_informacoes) => Fala o nome do aluno e se ele foi aprovado ou não - Calcular Média (calcular_media) => Calcula e retorna a média do aluno - Inserir Nota (inserir_nota) => Adiciona valor nas notas do aluno (2 parâmetros de nota)

Regras:
- Para passar ele precisa de 6 - Nome será enviado no construtor - Nota1 e Nota2 será enviado por

parâmetro => Inserir Nota

[186]:
```
class Aluno(object):
    status = False

    def __init__(self, nome):
        self.nome = nome

    def inserir_notas(self,nota1,nota2):
        self.nota1 = nota1
        self.nota2 = nota2

    def calcular_media(self):
        return (self.nota1 + self.nota2) / 2

    def mostrar_informacoes(self):
        status = (self.calcular_media() >= 6)
        if(status):
            print(f"O aluno {self.nome} foi aprovado")
        else:
            print(f"O aluno {self.nome} não foi aprovado")
```

[187]:
```
felipe = Aluno('Felipe')
```

[188]:
```
felipe.inserir_notas(10,9)
```

[189]:
```
felipe.mostrar_informacoes()
```

```
O aluno Felipe foi aprovado
```

[190]:
```
haynes = Aluno('Haynes')
haynes.inserir_notas(0,1)
haynes.mostrar_informacoes()
```

```
O aluno Haynes não foi aprovado
```

[ ]:
```

```