Programação Python

<u>Aula 05</u>: Introdução à Programação Orientada a Objetos

Prof. Eduardo Corrêa Gonçalves

25/03/2021

Sumário

Introdução (1/5)

- **O que é Programação Orientada a Objetos (POO)?**
  - Como o nome indica, no paradigma orientado a objetos, os principais "atores" são os **objetos**.
  - Cada objeto é uma **instância** (materialização) de uma **classe**.
  - A definição de uma classe deve especificar **atributos** e **métodos**.
    - **Atributos**: armazenam as características dos objetos de uma classe. Também chamados de **Propriedades** ou *Data Members.*
    - **Métodos**: ações que os objetos de uma classe podem executar. Também chamados de *Member Functions*.
  - Uma classe deve apresentar ao "mundo exterior" uma visão concisa e consistente dos objetos que são instância dessa classe.

Introdução (2/5)

- **Exemplo**
  - classe Usuario de um app de streaming de música.

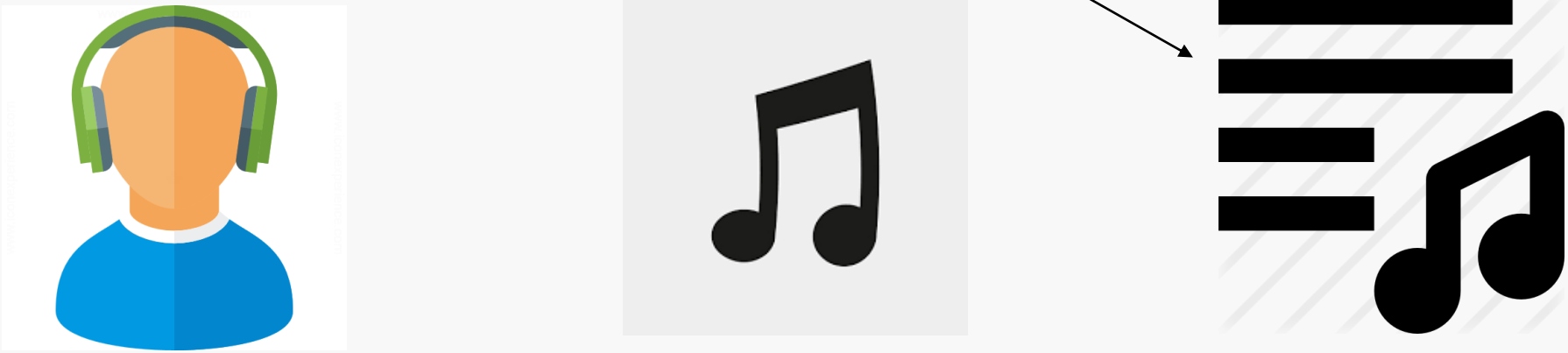
• **Atributos**: “nome”, “e-mail”, “cidade”, “data de nascimento”, ...

• **Métodos**: “buscar música”, “buscar artista”, “ouvir música”, “criar playlist”, “alterar forma de pagamento”, “remover playlist”, ...

- *Na POO você desenvolve pensando exclusivamente em classes e seus atributos e métodos !!!*

Introdução (3/5)

- **Sistema Orientado a Objetos**
  - É formado por um conjunto de objetos de diferentes classes que irão se comunicar (**trocar mensagens** entre si).
    - **Exemplo** – Um app de Streaming de Música poderia definir as classes Usuario, Musica e Playlist (entre outras).



  - Cada objeto possui diferentes responsabilidades e deverá realizar as operações para as quais está apto.

Introdução (4/5)

- **Classes**
  - Na POO, cada objeto é a instância de uma **classe**. A classe é uma espécie de *template* onde definimos os atributos e métodos de um tipo específico de objeto.
    - Em Python, utilizamos a palavra reservada **class** para definir uma classe:

```
class Musica:
    especificação dos atributos e métodos
```

Introdução (5/5)

- **Classes x Objetos**
  - Ou seja: classe e objeto **não** são a mesma coisa! A **classe** representa uma **estrutura** (*template*), mas não o conteúdo.
  - **Exemplo** – seja uma classe **Musica** com três atributos: “nome”, “artista” e “estilos”.
  - Podemos **instanciar** - criar na memória - diferentes **objetos** desta classe.
  - No exemplo ao lado, temos dois objetos instanciados, “m1” e “m2”. Cada objeto representa uma música específica.

Criação de Classes (1/7)

- **Sobre a aula de hoje:**
  - POO é um assunto que engloba muitos conceitos e técnicas.
  - Nessa aula apresentaremos apenas as os conceitos básicos para que você possa definir e utilizar as suas primeiras classes.
  - Como primeiro exemplo, trabalharemos na definição de uma classe **Musica**.

```
>>> class Musica:
        pass            # pass é um "stub": comando que você pode
                        # usar para informar que vai colocar código
                        # posteriormente.

>>> type(Musica)        # Em Python, class e type são
<class 'type'>          # são a mesma coisa
>>>
```

Criação de Classes (2/7)

- **Definindo a classe Musica**
  - Essa classe é bem simples: possui 3 atributos e 1 método.

```
class Musica:
    #construtor: inicializa os atributos
    def __init__(self, nome, artista, estilos):
        self.nome = nome
        self.artista = artista
        self.estilos = estilos

    def tocar(self):
        return "tocando '{}' por {}...".format(self.nome, self.artista)
```

  - Esse programa tem **apenas o *template*** que define a classe.
    - Mas ainda **não tem o código que instancia** (cria em memória) um ou mais objetos da classe.

Criação de Classes (3/7)

- **Classe Musica**
  - **Atributos**: “nome”, “artista” e “estilos”
  - **Método**: tocar()

```
class Musica:
    #construtor: inicializa os atributos
    def __init__(self, nome, artista, estilos):
        self.nome = nome
        self.artista = artista
        self.estilos = estilos

    def tocar(self):
        return "tocando '{}' por {}...".format(self.nome, self.artista)
```

  - Mas o que é **__init__** e o que é **self**?

Criação de Classes (4/7)

- **__init__**
  - Método especial, **chamado automaticamente** quando instancia-se um objeto da classe.
    - Chamado de **construtor** da classe.
    - Neste exemplo, ele foi utilizado para atribuir valores para os atributos no momento em que a classe foi instanciada.
      - *É a principal aplicação do construtor... Mas existem outras!*

```
class Musica:
    #construtor: inicializa os atributos
    def __init__(self, nome, artista, estilos):
        self.nome = nome
        self.artista = artista
        self.estilos = estilos

    def tocar(self):
        return "tocando '{}' por {}...".format(self.nome, self.artista)
```

Criação de Classes (5/7)

- **self (1/2)**
  - Em Python alguns nomes de métodos são reservados para uso pela própria linguagem e “__init__” é um deles.
  - Seu primeiro parâmetro (assim como ocorre em todo método) é a própria instância, que por convenção, denotamos por **self**.

```
class Musica:
    #construtor: inicializa os atributos
    def __init__(self, nome, artista, estilos):
        self.nome = nome
        self.artista = artista
        self.estilos = estilos

    def tocar(self):
        return "tocando '{}' por {}...".format(self.nome, self.artista)
```

Criação de Classes (6/7)

- **self (2/2)**
  - self representa a **própria instancia** do objeto.
    - Neste caso, uma música específica.
    - Ex.: self.nome refere-se ao valor do atributo “nome” de um objeto instanciado.

```
class Musica:
    #construtor: inicializa os atributos
    def __init__(self, nome, artista, estilos):
        self.nome = nome
        self.artista = artista
        self.estilos = estilos

    def tocar(self):
        return "tocando '{}' por {}...".format(self.nome, self.artista)
```

Criação de Classes (7/7)

- **Métodos**
  - Além do __init__, a classe Musica possui um único método, chamado “tocar”.
    - Veja que assim como o método __init__, ele deve receber a instância do objeto (self) como parâmetro.
      - Isso acontece porque o método precisa saber qual objeto Musica ele deve manipular, nesse caso qual música vai tocar (*podemos ter muitas musicas criadas no nosso sistema*)

```
class Musica:
    #construtor: inicializa os atributos
    def __init__(self, nome, artista, estilos):
        self.nome = nome
        self.artista = artista
        self.estilos = estilos

    def tocar(self):
        return "tocando '{}' por {}...".format(self.nome, self.artista)
```

Trabalhando com Objetos (1/9)

- **Instanciando Objetos**
  - Basta utilizar o nome da classe e passar as propriedades do objeto na ordem que você especificou no método construtor __**init**__.
    - Veja abaixo (em verde), como instanciamos duas músicas, “m1” e “m2”.

```
class Musica:
    #construtor: inicializa os atributos
    def __init__(self, nome, artista, estilos):
        self.nome = nome
        self.artista = artista
        self.estilos = estilos

    def tocar(self):
        return "tocando '{}' por {}...".format(self.nome, self.artista)

m1 = Musica("Wave","Tom Jobim",["Bossa Nova", "Jazz", "MPB"])
m2 = Musica("New Rules", "Dua Lipa",["Pop", "Dance"])
```

Trabalhando com Objetos (2/9)

- **Instanciando Objetos**
  - Ao criar uma música (objeto da classe Musica) , estamos pedindo para o Python criar uma nova instância de Musica na memória
  - Ou seja, o Python alocará memória suficiente para guardar todas as informações da música dentro da memória do programa.
  - O __init__, portanto, devolve uma referência, uma seta que aponta para o objeto em memória e é guardada nos nomes “m1” e “m2”.

Trabalhando com Objetos (3/9)

- **Instanciando Objetos**
  - **Ou seja:** *m1* e *m2* são dois diferentes objetos da classe (class ou type) Musica. Cada um deles:
    - Representa uma música específica.
    - Está armazenado em um local diferente da memória.

```
m1 = Musica("Wave","Tom Jobim",["Bossa Nova", "Jazz", "MPB"])
m2 = Musica("New Rules", "Dua Lipa",["Pop", "Dance"])
>>> type(m1)
<class '__main__.Musica'>
>>> type(m2)
<class '__main__.Musica'>
>>> id(m1)        #id retorna o identificador do objeto em memória
54559024
>>> id(m2)
60899600
```

Trabalhando com Objetos (4/9)

- **Instanciando Objetos – Observações Importantes (1/2)**
  - Da forma como definimos o construtor, o código não permite criar uma música sem os atributos nome, artista e estilos (todos os 3 são necessários)

```
>>> m3 = Musica("Total Eclipse ofthe Heart" , "Bonnie Tyler")

Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
TypeError: __init__() missing 1 required positional argument: 'estilos'
```

Trabalhando com Objetos (5/9)

- **Instanciando Objetos – – Observações Importantes (2/2)**
  - Veja que em **nenhum momento** chamamos o método __init__ para instanciar “m1” ou “m2”.
  - Quem está fazendo por baixo dos panos é o próprio Python quando cria estes objetos.

```
m1 = Musica("Wave","Tom Jobim",["Bossa Nova", "Jazz", "MPB"])
m2 = Musica("New Rules", "Dua Lipa",["Pop", "Dance"])
```

Trabalhando com Objetos (6/9)

- **Acessando um atributo**
  - Para acessar um atributo qualquer, basta especificar o nome do objeto, depois um **ponto** e depois o nome do atributo.

```
m1 = Musica("Wave","Tom Jobim",["Bossa Nova", "Jazz", "MPB"])
print(‘* artista: ', m1.artista)
print(‘* música: ', m1.nome)
print(‘* estilos: ', m1.estilos)
```

saída:

```
* artista:  Tom Jobim
* música:  Wave
* estilos:  ['Bossa Nova', 'Jazz', ‘MPB']
```

Trabalhando com Objetos (7/9)

- **Modificando um atributo**
  - Para modificar um atributo, basta fazer uma atribuição.

```
m3 = Musica("Juízo Final","Nelson Cavaquinho",["Samba"])

m3.nome = "Rugas"              #muda o valor do atributo nome de m3
m3.estilos.append("MPB")       #acrescenta um estilo à m3

print('* artista: ', m3.artista)
print('* música: ', m3.nome)
print('* estilos: ', m3.estilos)
```

saída:

```
* artista:  Nelson Cavaquinho
* música:  Rugas
* estilos:  ['Samba', 'MPB']
```

Trabalhando com Objetos (8/9)

- **Chamando um Método**
  - Depois de instanciar os objetos, podemos chamar (executar) os métodos que foram definidos em sua classe.
    - No caso da classe Musica, existe apenas um método definido além do __init__(). É o método “tocar()”.

```
>>> m1.tocar()
"tocando 'Wave' por Tom Jobim..."
>>> m2.tocar()
"tocando 'New Rules' por Dua Lipa..."
```

  - Veja que para chamar um método basta especificar o nome do objeto, um ponto e depois o nome do método.
  - Note ainda que não precisamos passar a própria música (self) como argumento. O Python faz isso de forma escondida.

Trabalhando com Objetos (9/9)

- **Tipo de um Objeto**
  - Veja de novo qual é o tipo de um objeto da classe Musica:

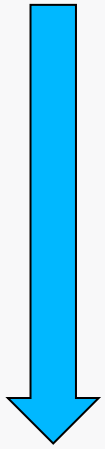
```
>>> type(m2)
<class '__main__.Musica'>
```

  - Obviamente, o tipo de um objeto da classe Musica é “Musica”!!!
  - Acima, o “__main__.” aparece porque o Python quer indicar que a classe Musica está definida no próprio programa em que o objeto “m1” foi instanciado.
    - Maiores explicações serão apresentadas na aula sobre Namespaces.

POO – Discussão (1/2)

- Python é uma linguagem orientada a objetos, porém **não te obriga** a utilizar POO.
  - Python suporta outros paradigmas de programação, como o tradicional **paradigma procedural** (ou **estruturado**)
    - O paradigma procedural é aquele utilizado pelo Pascal, por exemplo.
    - Ele emprega uma abordagem **linear** e **top-down**: o **fluxo de execução** (ordem em que os comandos são executados) é de cima para baixo, em sequência.
    - E utiliza **procedures** e **funções** para organizar código reutilizável.
  - Devemos admitir que, em geral, é mais simples programar scripts de análise de dados utilizando o paradigma procedural.
    - Especialmente se você estiver trabalhando em um experimento que exija apenas a construção de um **programa pequeno**.

Procedural é top-down



POO – Discussão (2/2)

- Porém, a POO é muito útil para os cientistas de dados em algumas situações práticas.
  - **Exemplos**:
    - Como tudo em Python é objeto, ao entender a POO você se **torna mais capacitado** a entender as características do Python, suas EDs e pacotes.
    - Falando de Ciência de Dados especificamente, é importante observar que a maioria dos **pacotes estatísticos** do Python é desenvolvido em POO. Isso porque a POO favorece a manutenção, distribuição e tratamento de erros de código.
      - Sendo assim, se você quiser **entender melhor** o funcionamento de um pacote ou, até mesmo, **desenvolver o seu próprio pacote**, precisará conhecer POO.
    - Se o programa de seu experimento possui um **grande número de linhas**, considere adotar a POO, pois ela facilita a manutenção do código e identificação de erros.
    - Fora o fato de que a **POO é mais moderna**. Qualquer app atual é desenvolvido em POO e não na abordagem procedural.

Comentário Final

- **Programação Orientada a Objetos - POO**
  - O conteúdo que apresentamos nesta primeira aula representa apenas a “pontinha do iceberg” sobre POO.
    - Mas é o suficiente para que, a partir de agora, você possa se sentir mais confiante para estudar tópicos avançados:
      - Encapsulamento;
      - Herança;
      - Polimorfismo;
      - Classes abstratas;
      - *E outros...*
  - **IMPORTANTE***: ao definir uma classe, você está criando um novo tipo, criando a sua própria ED!!!*

Tarefa – Dojo: Classe “ProgressaoAritmetica”

- Uma progressão aritmética (PA) consiste em uma sucessão de números em que a diferença entre cada um deles, a partir do segundo, e o seu antecessor é sempre a mesma. Essa diferença é chamada de razão (r) da PA. Exemplos:

  (1, 3, 5, 7, 9, 11) → PA onde a1 = 1 (primeiro termo); r = 2; 6 termos.

  (0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0) → a1 = 0; r = ½; 9 termos.

  (10, 10, 10, 10, 10) → a1 = 10; r = 0; 5 termos.

  (9, 5, 1, -3, -7) → a1 = 9; r = -4; 5 termos.

- Crie uma classe chamada ProgressaoAritmetica com as seguintes características:
  - 2 Atributos: “a1” e “r”
  - 3 Métodos: “termo”, “soma” e “sequência”
    - “termo”: retorna o termo de ordem n, utilizando a fórmula $a_n = a_1 + (n - 1)r$
    - “soma”: retorna a soma dos n primeiros termos da PA. $S_n = (n(a_1 + a_n)) / 2$
    - “sequencia”: retorna uma lista contendo os n primeiros termos da PA.

  * * * DICA: o método “termo” pode ser utilizado pelos métodos “soma” e “sequência”.

Referências

- Corrêa, E. (2020). “Meu Primeiro Livro de Python”. V 2.0.0, edubd, 2020. (*capítulo 9*).
  - Disponível em: https://github.com/edubd/meu_primeiro_livro_de_python

- CAELUM (2020). Python e Orientação a Objetos – Curso PY-14 (*cap 7, 8, 10 e 11*).
  - https://www.caelum.com.br/apostila/apostila-python-orientacao-a-objetos.pdf

- Python 3 Tutorial – Object Oriented Programming.
  - https://www.python-course.eu/python3_object_oriented_programming.php