Paradigmas de Linguagem de Programação em Python

Paradigma Orientado a Objetos: Conceitos Básicos

Prof. Henrique Mota

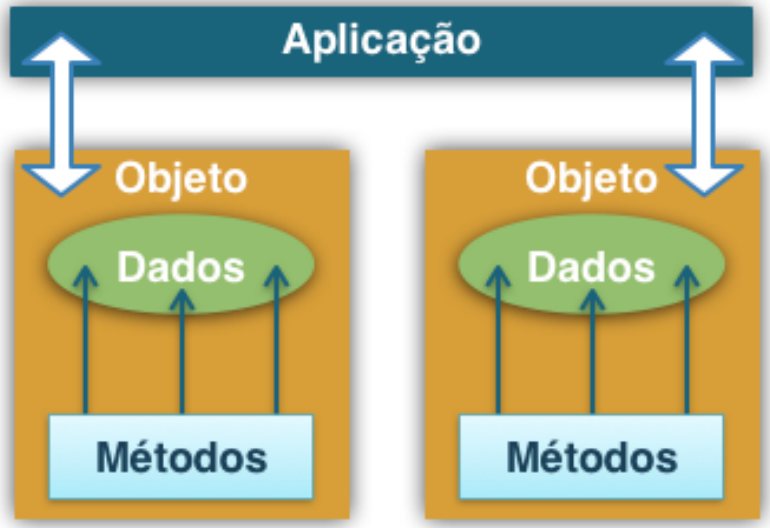
Objetivos

- **Aprender Conceitos Básicos** de **Orientação a Objetos** (Objetos, Atributos, Métodos e Classes | Construtores)
- **Aplicar** os Conceitos Aprendidos em um **Exemplo Prático**
- **Entender** o uso da palavra-chave ***self***
- **Entender** o **acesso** a **atributos** e **métodos**

Programação Imperativa

- Modularização dos programas baseada nas **funções** que um programa vai oferecer ao usuário.
- Dados normalmentes globais, podendo ser acessados por qualquer função

Programação Orientada a Objetos

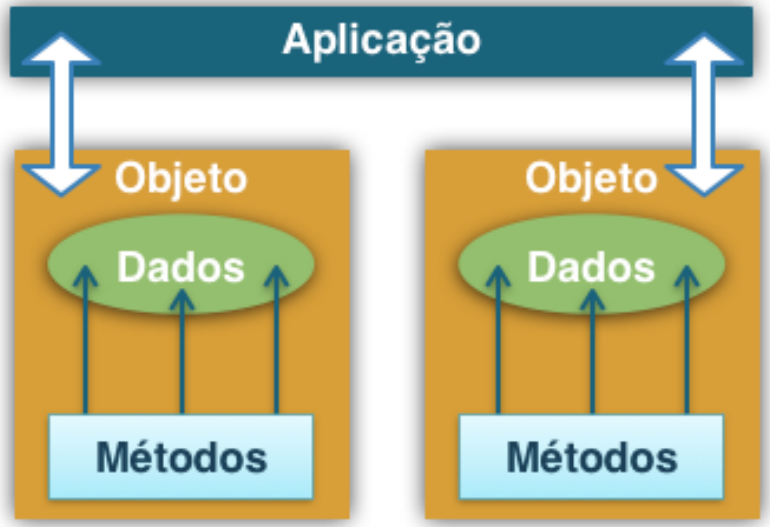


- Modularização dos programas baseada na definição de **objetos (dados)** a serem manipulados pelo sistema.
- Definição de **atividades** que um dado objeto poderá realizar

Diferenças

Programação Imperativa

Programação Orientada a Objetos



POO: Definição

- **Paradigma** que **usa** ao longo do processo de desenvolvimento de software, o **conceito** de **“Objetos”**.

- **Considera** os **sistemas computacionais** não como uma coleção estruturada de processos, mas sim como uma **coleção de objetos que interagem entre si**.

Conceito: Objetos

- Ja vimos que **Python** possui vários tipos diferentes de dados:

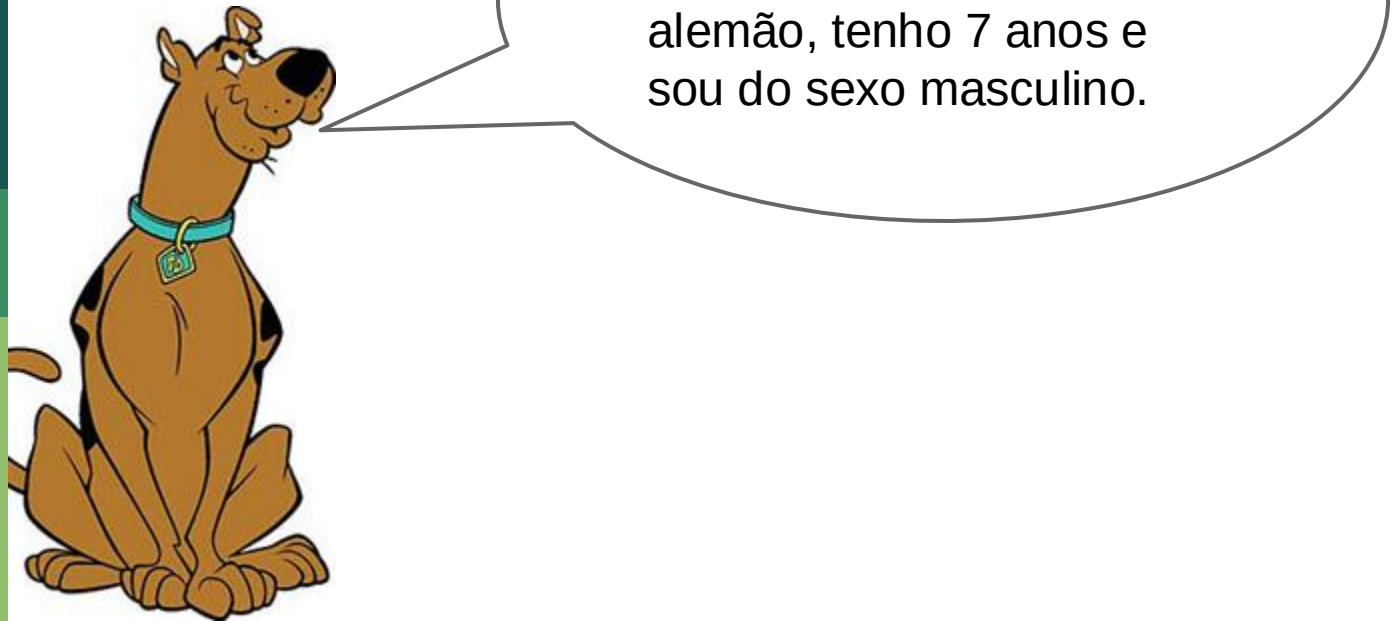
```
1234                                    3.14159
    "Bom dia"

[1, 5, 7, 11, 13]              {"CA": "California", "MA":
"Massachusetts"}
```

- Cada deles é um objeto, que possui:
  - um **tipo**
  - um **estado** (informação / representação interna)
    - primitivo x composto
  - um conjunto **operações** (comportamento) que permitem que seu **estado** seja **alterado.**

Conceito: Objetos

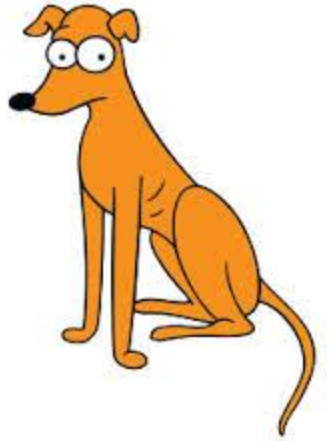
- Logo, um objeto é uma instância de um determinado tipo:
  - `1234` é uma instância de um **`int`**
  - `"Bom dia"` é uma instância de uma **`string`**
- **`Resumindo:`**
  - **`TUDO em Python é um objeto (e tem um tipo)`**
  - **`Objetos`** são abstrações (representações), que possuem:
    1) uma representação interna (atributos dos dados)
    2) uma interface de interação (métodos e funções)

Conceito: Objetos



Conceito: Objetos

Oi, meu nome é Ajudante de Papai Noel, eu sou um galgo inglês, ou greyhound, tenho 8 anos e sou do sexo masculino.



Conceito: Atributos

- São as **propriedades** (ou características) de uma classe.

- Seus valores descrevem o **estado** do objeto: é a informação que os diferencia;

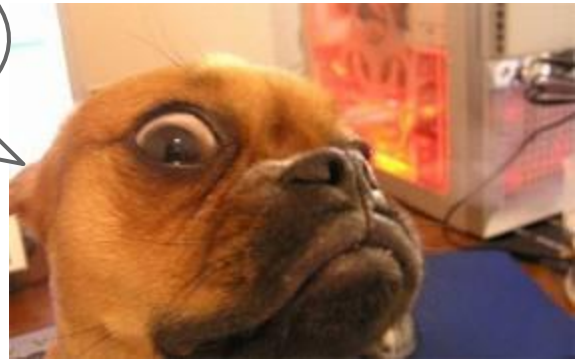
Conceito: Atributos

Considerando o que o Scooby-Doo e o Ajudante de Papai Noel disseram, quais seriam os atributos dos objetos?

Conceito: Atributos

Considerando o que o Scooby-Doo e o Ajudante de Papai Noel disseram, quais seriam os atributos dos objetos?

Seriam o **nome**, a **raça**, a **idade** e o **sexo**?



Conceito: Métodos

- São as **ações** que um objeto pode realizar: determina o seu comportamento.

- Podem ser usados para:
  - **Apresentar** ou **alterar o estado** de um objeto;
  - **Expor as ações** que um objeto pode executar.

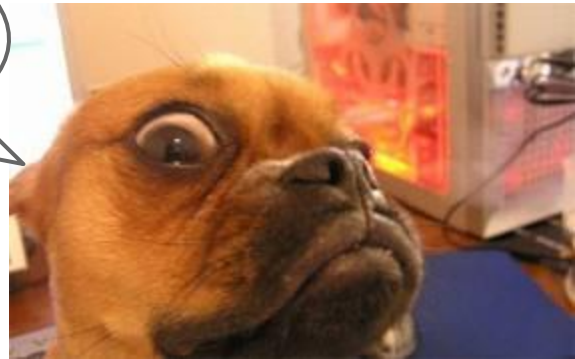
Conceito: Métodos

Considerando o Scooby-Doo e o Ajudante de Papai Noel, que ações eles podem fazer?

Conceito: Métodos

Considerando o Scooby-Doo e o Ajudante de Papai Noel, que ações eles podem fazer?

latir? **dormir**?
comer?
correr?



Conceito: Classe

- **Tipo de dado** que **agrupa** um **conjunto** de **variáveis** (**atributos**) e **funções** (**métodos**) que podem realizar ações;
- Possuem **características** e **comportamentos** em **comum;**
- Onde são realizadas as definições dos atributos e métodos;
- A **base** da **POO**: **um sistema é modelado pelas suas classes e não objetos.**

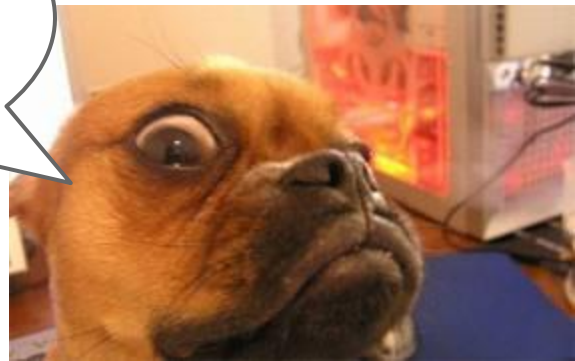
Conceito: Classe

Qual seria a classe de Scooby-Doo e do Ajudante de Papai Noel?

Conceito: Classe

Qual seria a classe de Scooby-Doo e do Ajudante de Papai Noel?

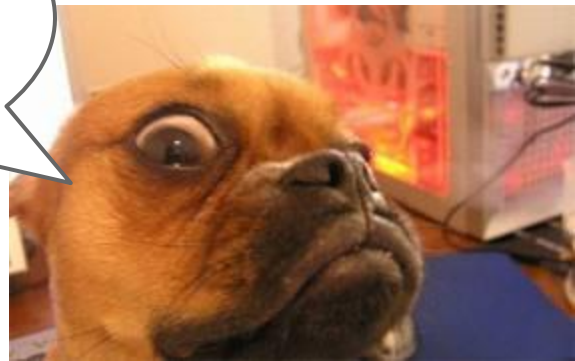
cachorro?! Quer dizer que eles são da mesma classe que a gente?



Conceito: Classe

Possivelmente...

achorro?!
Quer dizer que eles são da mesma classe que a gente?



Conceito: Objetos (Mais Precisamente)

- São representações reais de uma classe:
  - São **instâncias** de classes;

Sumarizando

- As **classes** são **tipos abstratos de dados** que definem **atributos** e **métodos:**
  - Como **regra geral**, os **atributos** (dados) devem ser **alterados** através dos **métodos** (operações).

Sumarizando

- Objetos são instanciados a partir das classes:
  - **Objetos** de uma **mesma classe** possuem os mesmos **atributos** e **métodos;**
  - Os **valores** dos **atributos** de cada **objeto variam.**

Representação

Classe

Cachorro
Atributos: String nome int idade String raça char sexo
Métodos: latir dormir comer correr

Objetos

Cachorro	
nome	"Scooby-Doo"
idade	7
raça	"Dogue Alemão"
sexo	M

Cachorro	
nome	"Ajudante de Papai Noel"
idade	8
raça	"Galgo inglês"
sexo	M

ATENÇÃO!!

- As notações visuais para classes e objetos não seguiram um notação padrão de forma proposital.

- A notação padrão é a **Unified Modeling Language (UML)**.

Aplicação
Definindo novo tipo (classe)

```
class Cachorro (object):
    # defina a classe!
    pass
```

- Similar ao `def` (que define funções/métodos), o **`class`** define uma Classe (tudo indentando pertence aquela Classe)
- **`Cachorro`** é o nome/tipo da Classe
- **`object`** indica que **`Cachorro`** é um objeto **`Python`** e herda todos os seus atributos (detalhes na próxima aula)
  - **`Cachorro`** é uma subclasse de **`object`**
  - **`Object`** é uma superclasse de **`Cachorro`**

Aplicação
Criando instancias

```
class Cachorro (object):
 # atributo de Classe
 attr1 = "mamifero"

 # Atributo de Instancia
 def __init__(self, nome):
   self.nome = nome
```

- Primeiro é preciso definir o método especial `__init__` (duplo _) para inicializar os atributos de cada instância
- **`self`** refere-se a um instância da classe
- **`nome`** é o parâmetro que inicializa o objeto **`Cachorro`**
- **`attr1`** é um atributo de classe

Mais sobre o *self*

- Ao referir-se a um instância da classe um objeto, ele é usado para:
  - **indicar que** os **atributos** ou **métodos que estão** sendo **acessados pertencem** ao **próprio objeto**;
  - diferir os atributos da classe em relação aos parâmetros e variáveis locais de um método

Aplicação
Criando instancias

```
class Cachorro (object):
 # atributo de Classe
 attr1 = "mamifero"
 # Atributo de Instancia
 def __init__(self, nome):
   self.nome = nome

# Inicialização de Objetos
Rodger = Cachorro("Rodger")
Tommy = Cachorro("Tommy")
# Acessando atributos de Classe
print("Rodger eh um {}".format(Rodger.__class__.attr1))
print("Tommy tambem eh um {}".format(Tommy.__class__.attr1))
# Acessando atributos de Instancia
print("Meu nome eh {}".format(Rodger.name))
print("Meu nome eh {}".format(Tommy.name))
```

Acessando Atributos e Métodos

- Para acessar atributos de um objeto, **utiliza-se** o **nome do objeto** (**e não o da classe**), seguido do caractere **ponto (.)**, mais o **nome do atributo** ou **método** que deseja-se acessar

- Sintaxe:

  objeto.nomeAtributo; ou

  objeto.nomeMetodo()

Aplicação
Mais métodos

```
class Cachorro (object):
 # atributo de Classe
 attr1 = "mamifero"
 # Atributo de Instancia
 def __init__(self, nome):
   self.nome = nome

 def falar(self):
   print("Meu nome eh {}".format(self.nome))

# Inicialização de Objetos
Rodger = Cachorro("Rodger")
Tommy = Cachorro("Tommy")
# Acessando metodos
Rodger.falar()
Tommy.falar()
```

- Python sempre passa o objeto (**self**) como primeiro argumento de todos os métodos

Aplicação

- Precisamos usar o **`self`** dentro da própria classe para acessar os atributos

```
class Coordenadas (object):
  def __init__(self, x, y):
    self.x = x
    self.y = y

 def distance(self, p2):
    x_diff_sq = (self.x - p2.x) ** 2
    y_diff_sq = (self.y - p2.y) ** 2
    return (x_diff_sq + y_diff_sq) ** 0.5

 def __str__(self):
    return "<"+str(self.x)+","+str(self.y)+">"
```

Aplicação

```
class Coordenadas (object):
  def __init__(self, x, y):
    self.x = x
    self.y = y

 def distance(self, p2):
    x_diff_sq = (self.x - p2.x) ** 2
    y_diff_sq = (self.y - p2.y) ** 2
    return (x_diff_sq + y_diff_sq) ** 0.5

 def __str__(self):
    return "<"+str(self.x)+","+str(self.y)+">"
```

- Precisamos usar o **`self`** dentro da própria classe para acessar os atributos
- Podemos definir o método **`__str__`** para criar uma representação informativa das instâncias na hora do `print`
  - `__` (duplo underline) sempre indica **métodos especiais**

Mais métodos especiais

- `__add__(self, outro)` → `self + outro`
- `__sub__(self, outro)` → `self - outro`
- `__eq__(self, outro)` → `self == outro`
- `__lt__(self, outro)` → `self < outro`
- `__len__(self)` → `len(self)`
- `__str__(self)` → `print(self)`

- Os métodos especiais **sobrescrevem** os respectivos métodos e operações

As vantagens do uso de Orientação a Objetos

- Juntar objetos que compartilham
  - atributos em comuns
  - Operações que atuam sobre estes atributos
- Usar abstrações para estabelecer uma separação entre **implementar** um objeto x **usar** um objeto
- Criar camadas de abstrações através de conceitos de heranças
- Criar classes customizadas acima das classes básicas da linguagem

Exercício 1

1. Implementar a classe **Quadrado** com as seguintes definições:
   a. O atributo **lado**;
   b. O método **area()**;
   c. O método **comprimento()**;
   d. Implementar o código que **cria** um objeto do tipo **Quadrado**, a partir do lado informado pelo usuário, e **imprime** o valor de sua **área**, **comprimento** e **lado**.

Exercício 2

2. Implementar a classe **TrianguloRetangulo** com as seguintes definições:
   a. Os atributos **base**, **altura** e **hipotenusa**.
   b. O método **area()**;
   c. O método **comprimento()**;
   d. Implementar o código que **cria** um objeto do tipo **TrianguloRetangulo**, a partir dos valores informados pelo usuário, e **imprime** o valor de sua **área**, **comprimento** e **atributos.**

Obrigado!

Alguma dúvida?

Prof. Henrique Mota

profhenriquemota@gmail.com