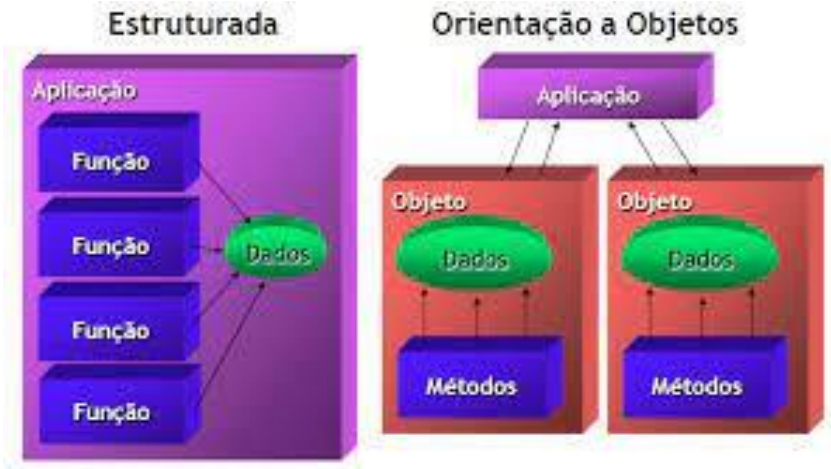
# Introdução à Programação Orientada a Objetos

## Algoritmos e Programação em Python

Prof. Fabio Fernando Kobs, Dr.

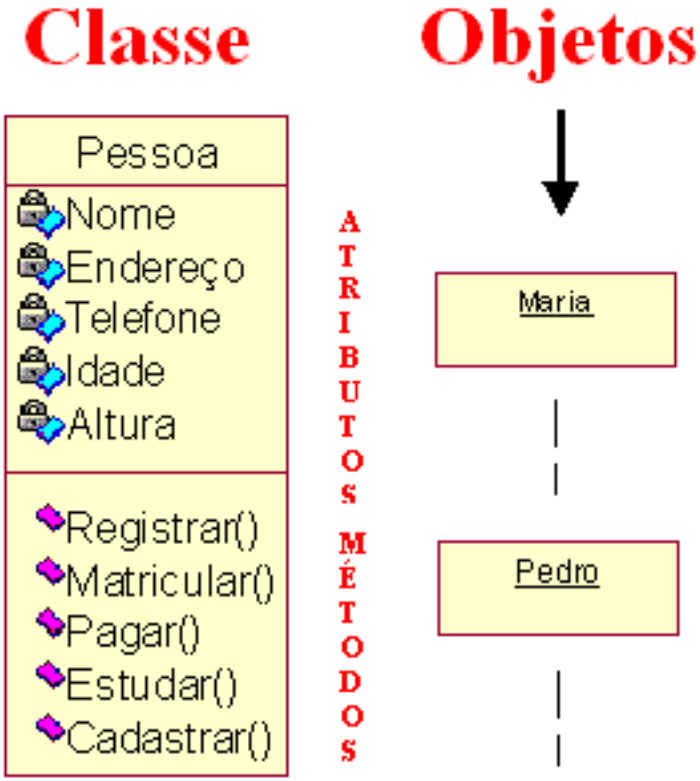


# Agenda

- Classes e Objetos
- Instâncias e Métodos construtor e destrutor
- Encapsulamento, métodos acessores e modificadores
- Herança simples

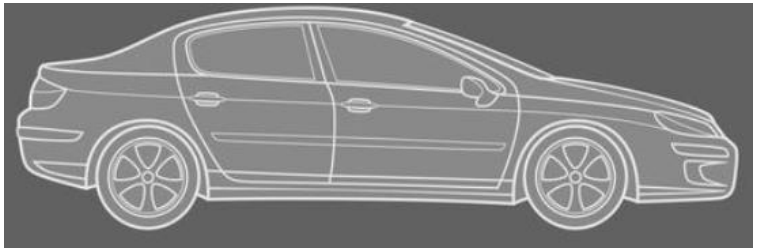
# Classe e Objeto – Conceito

- **Classes** são a definição de um novo tipo de dados que associa dados e operações em uma só estrutura.
- **Objeto** é uma variável cujo tipo é uma classe, ou seja, um objeto é uma instância de uma classe. Como a representação de um objeto do mundo real, escrita em uma linguagem de programação.



# Classe x Objeto

Classe:



| Veículo |
| --- |
| - modelo : string |
| - cor : string |
| - placa : string |
| |

Objetos:


Modelo: Gol
Cor: Verde
Placa: GOL 1983


Modelo: Fusca
Cor: Azul
Placa: KTU 1965

# Classe e Objeto – Self

- Tudo o que se sabe sobre funções, aplica-se para método. Porém, o método está associado a uma classe e atua sobre um objeto.
- O primeiro parâmetro do método é chamado ***self***, e representa a instância sobre a qual o método atuará. É por meio de ***self*** que se tem acesso aos outros métodos de uma classe, preservando todos os atributos dos objetos.
- Não precisa passar o objeto como primeiro parâmetro ao invocar um método. Python faz automaticamente, porém faz-se necessário declarar ***self*** como o primeiro parâmetro de seus métodos.

# Objeto – Instâncias

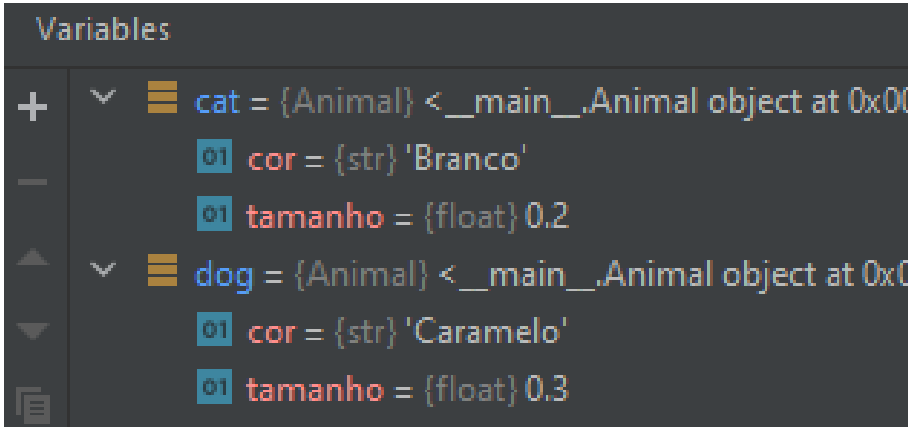
- Uma classe quando criada (instanciada) é chamada de classe objeto (tipo classobj).
- Objetos são instanciados pelas classes, e **cada objeto possui os atributos da classe**.
- Um método é uma função criada na definição de uma classe. O primeiro argumento do método é sempre referenciado no início do processo. **Por convenção, o primeiro argumento do método tem sempre o nome *self*. Logo, os atributos de *self* são atributos de instância da classe**.

```
class Animal:

    tamanho = None
    cor = None

    def __init__(self, t, c):
        self.tamanho = t
        self.cor = c


dog = Animal(0.30, "Caramelo")
cat = Animal(0.20, "Branco")
```



# Método Construtor - __init__

- O método *init* é um método especial para classes. O *init* é um método construtor, ele inicializa o estado de um objeto.
- O método *init* é invocado a cada nova instância de uma classe, ou seja, quando é criada.
- No Python funciona assim: tem-se uma classe A, e uma classe B que herda de A. Se a classe A tiver o método __init__, mas a classe B não tiver, ao se instanciar a classe B, o interpretador vai procurar o construtor, não achando na classe atual, vai subindo para as classes base, chamando o primeiro que encontrar, no caso o da classe A.

---

– Pode-se atribuir nulo para uma variável, e, depois, atribuí-la a uma instância de objeto ou constante.

*nome_variável = None*

# Método Construtor – Exemplo

Exemplo de um aparelho de TV. A TV tem uma marca e um tamanho de tela. Pode-se fazer com a TV, por exemplo, mudança de canal, ligá-la ou desligá-la.

```
class TV:
    def __init__(self):                 # método construtor; self é um objeto TV em si
        self.ligada = False             # é um valor de self, ou seja, do objeto TV
        self.canal = 2                  # ao especificar atributos do objeto, sempre usar self
##################################################################################
tv = TV()                   # cria-se um objeto tv utilizando a classe TV
print(tv.ligada)            # ou seja, tv é uma instância de TV
print(tv.canal)
tv_sala = TV()              # cria-se um objeto tv_sala
tv_sala.ligada = True
tv_sala.canal = 4
print(tv.ligada)
print(tv.canal)
print(tv_sala.ligada)
print(tv_sala.canal)
```

# Exemplo

```
class TV:

    def __init__(self, min, max):
        self.ligada = False
        self.canal = 2
        self.cmin = min
        self.cmax = max

    def muda_canal_para_baixo(self):
        if(self.canal-1 >= self.cmin):
            self.canal -= 1

    def muda_canal_para_cima(self):
        if (self.canal + 1 <= self.cmax):
            self.canal += 1
```

```
################################
tv = TV(1,99)
for x in range(120):
    tv.muda_canal_para_cima()
print(tv.canal)
for x in range(120):
    tv.muda_canal_para_baixo()
print(tv.canal)
```

# Classe e Objeto – Exercício

1. Defina uma classe para:
   a) Veículo
   b) Pessoa
   c) Animal
   d) Produto
   e) Smartphone
   f) Notebook
   g) TV
   h) Máquina de lavar roupas

# Encapsulamento

- Em muitas linguagens (por ex.: c++, java) existe o conceito de modificadores de acessos. Por exemplo, em C++ o modificador ***private*** determina que os métodos e atributos só podem ser acessados pela própria classe; o modificador ***protected*** determina que apenas as classes filhas podem acessar; o modificador ***public*** determina que qualquer classe pode acessar os elementos.
- Python não tem o conceito de métodos privados. Porém, para se poder ter o recurso de encapsulamento tem-se um conceito de variáveis ocultas no qual *não se restringe o acesso mas dificulta o acesso a determinadas variáveis*. Para o Python tem-se dois termos para classificar as variáveis privados - métodos e atributos fracamente privados e os fortemente privados. A seleção entre esses dois métodos é determinado pelo seu nome.

# Encapsulamento

- Métodos e atributos **fracamente privados** possuem um subtraço ou sublinhado (_) no início (à esquerda). Essa abordagem APENAS sinaliza que essas variáveis são privadas e outros programadores não devem usá-las em código externos, exceto se tratar de uma subclasse (herança). Porém, aos métodos fracamente privados não se impede que o código seja acessado de fora da classe. O efeito é que quando os módulos são importados as variáveis com sublinhado (_) não serão importadas. Exemplo de itens fracamente privados:

```
class ClasseFracamentePrivada():
    def __init__(self):
        self._atributoFracamentePrivado = 100
    def _MetodoFracamentePrivado(self):
        print("método fracamente privado")
#####################################################
p1 = ClasseFracamentePrivada()
p1._MetodoFracamentePrivado()
print(p1._atributoFracamentePrivado)
```

# Encapsulamento

- Métodos e atributos **fortemente privados** possuem dois subtraços ou sublinhados (__) no início (à esquerda). Essa abordagem faz com que o nome do método não possa ser encontrado/acessado fora da classe - os escondendo. Para acessar um método fortemente privado basta colocar um sublinhado (_) na frente do nome classe em seguida escrever o nome da variável que o método poderá ser acessado.
- Exemplo de itens fortemente privados:

```
class ClasseFortementePrivada():
    def __init__(self):
        self.__atributoFortementePrivado = 100
    def __MetodoFortementePrivado(self):
        print("método fortemente privado")
##########################################################
p1 = ClasseFortementePrivada()
p1._ClasseFortementePrivada__MetodoFortementePrivado()
print(p1._ClasseFortementePrivada__atributoFortementePrivado)
```

# Acessores e Modificadores

- Um atributo de classe promove o método de acesso e/ou método modificador. O método ***get*** é uma propriedade específica para **acessar** os atributos chamada de "*getter*". Já o método ***set*** é uma propriedade específica do método **modificador** chamado de "*setter*".
- Por convenção, adiciona-se a palavra *get* (obter, pegar) e *set* (por, colocar) antes do nome do atributo. Os métodos *get* e *set* promovem o acesso dos atributos. Um método que acessa uma instância mas não modifica a instância é chamado de acessor.
- Os métodos *set* promovem a modificação dos atributos. Um método que modifica uma instância é um modificador.
- O operador . (ponto) é utilizado para especificar o objeto em que o método também é invocado.

# Acessores e Modificadores – Exemplo

```
class Cliente:

    __nome = None
    __telefone = None

    def __init__ (self, nome, telefone):
         self.__nome = nome
         self.__telefone = telefone

    def getnome (self):
         return self.__nome

    def setnome (self, nome):
         self.__nome = nome

    def gettelefone (self):
         return self.__telefone

    def settelefone (self, telefone):
         self.__telefone = telefone
```

# Exercícios

2. **Classe Bola**: Crie uma classe que modele uma bola:
   Atributos: Cor, circunferência, material
   Métodos: trocaCor e mostraCor
3. **Classe Quadrado**: Crie uma classe que modele um quadrado:
   Atributos: Tamanho do lado
   Métodos: Mudar valor do Lado, Retornar valor do Lado e calcular Área;
4. **Classe Retângulo**: Crie uma classe que modele um retângulo:
   Atributos: LadoA, LadoB (ou Comprimento e Largura, ou Base e Altura, a escolher)
   Métodos: Mudar valor dos lados, Retornar valor dos lados, calcular Área e calcular Perímetro;
   Crie um programa que utilize esta classe. Ele deve pedir ao usuário que informe as medidas de um local. Depois, deve criar um objeto com as medidas e calcular a quantidade de pisos e de rodapés necessárias para o local.

5. **Classe Pessoa**: Crie uma classe que modele uma pessoa:
   Atributos: nome, idade, peso e altura
   Métodos: Envelhecer, engordar, crescer, emagrecer. Obs: Por padrão, a cada ano que a pessoa envelhece, sendo a idade dela menor que 21 anos, ela deve crescer 0,5 cm ao ano. A pessoa engorda 0,5 kg por ano.
   Faça uma projeção para X anos, informando o novo peso e/ou altura.
6. **Classe Conta Corrente**: Crie uma classe para implementar uma conta corrente. A classe deve possuir os seguintes atributos: número da conta, nome do correntista e saldo. Os métodos são os seguintes: alterarNome, depósito e saque. No construtor, saldo é opcional, com valor *default* zero e os demais atributos são obrigatórios. Faça um programa que realiza depósitos e saques diversos, informando o saldo a cada operação.
7. **Classe TV**: Faça um programa que simule um televisor criando-o como um objeto. O usuário deve ser capaz de informar o número do canal e aumentar ou diminuir o volume. Certifique-se de que o número do canal e o nível do volume permanecem dentro de faixas válidas.
8. **Classe Bichinho Virtual**: Crie uma classe que modele um Tamagushi (Bichinho Eletrônico): Atributos: Nome, Fome, Saúde e Idade b. Métodos: Alterar Nome, Fome, Saúde e Idade; Retornar Nome, Fome, Saúde e Idade. Obs: Existe mais uma informação que deve-se levar em consideração, o Humor do nosso Tamagushi, este humor é uma combinação entre os atributos Fome e Saúde, ou seja, um campo calculado, então não devemos criar um atributo para armazenar esta informação por que ela pode ser calculada a qualquer momento.

# Encapsulamento – Exercício

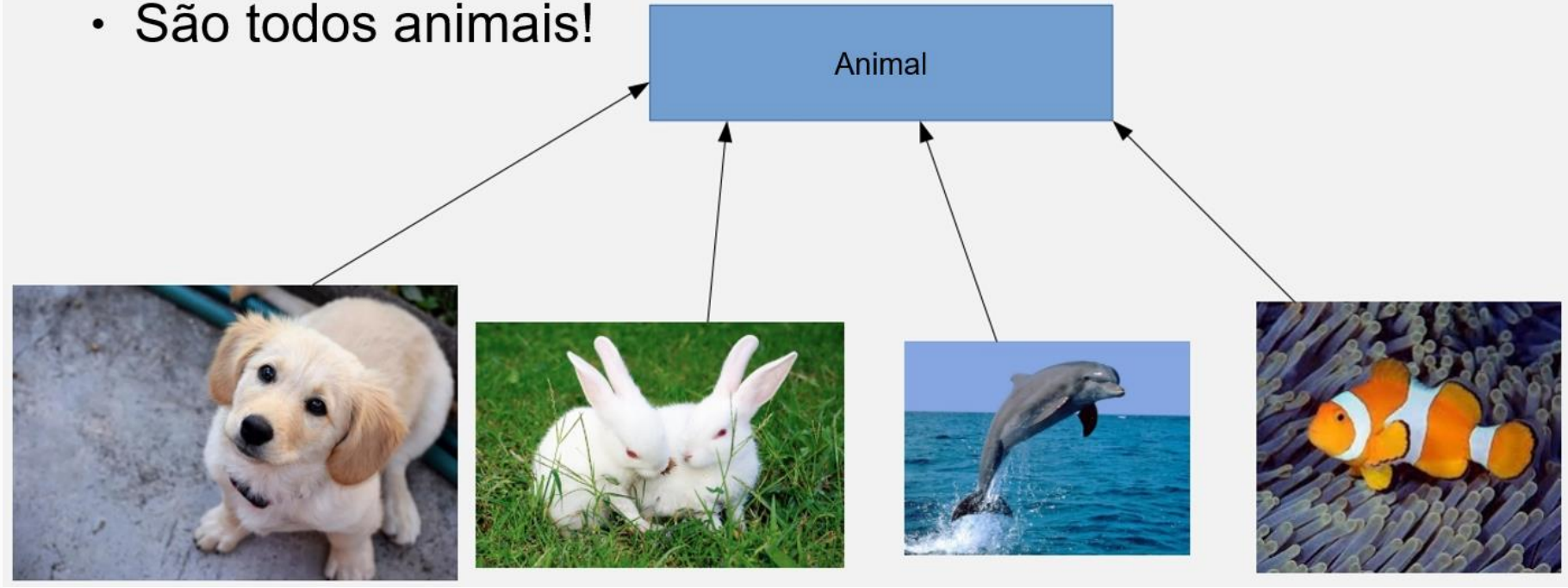
9. ~~Altere o programa do Banco X de forma que a mensagem saldo insuficiente seja exibida caso haja tentativa de sacar mais dinheiro que o saldo disponível.~~
10. ~~Modifique o método resumo da classe Conta para exibir o nome e o telefone de cada cliente.~~
11. ~~Crie uma nova conta, agora tendo João e José como clientes e saldo igual a 500.~~
12. Crie classes para representar estados e cidades. Cada estado tem um nome, sigla e cidades. Cada cidade tem nome e população. Escreva um programa de testes que crie três estados com algumas cidades em cada um. Exiba a população de cada estado como a soma da população de suas cidades.

# Herança

- O que as seguintes fotos possuem em comum?

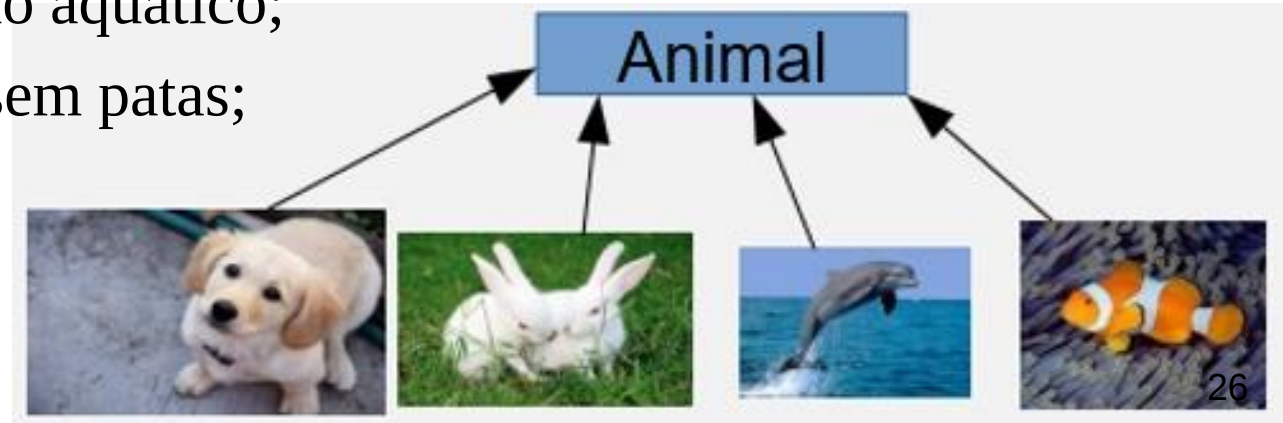
# Herança

- São todos animais!



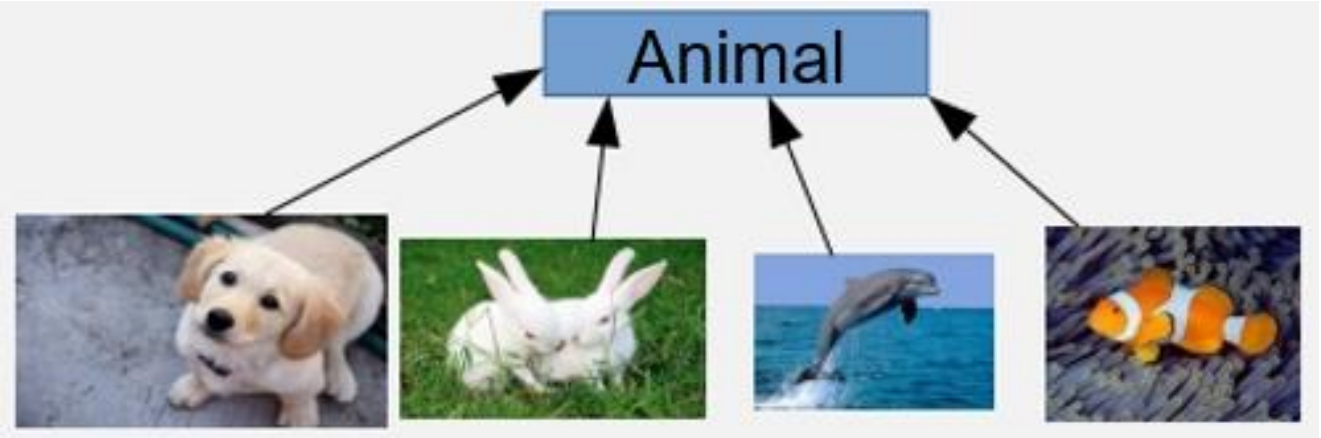
# Herança

- Possuem atributos semelhantes:
  - Idade;
  - Sexo;
  - Peso;
  - Outros.
- Possuem atributos diferentes:
  - Mamífero e não mamífero;
  - Aquático e não aquático;
  - Com patas e sem patas;
  - Outros.



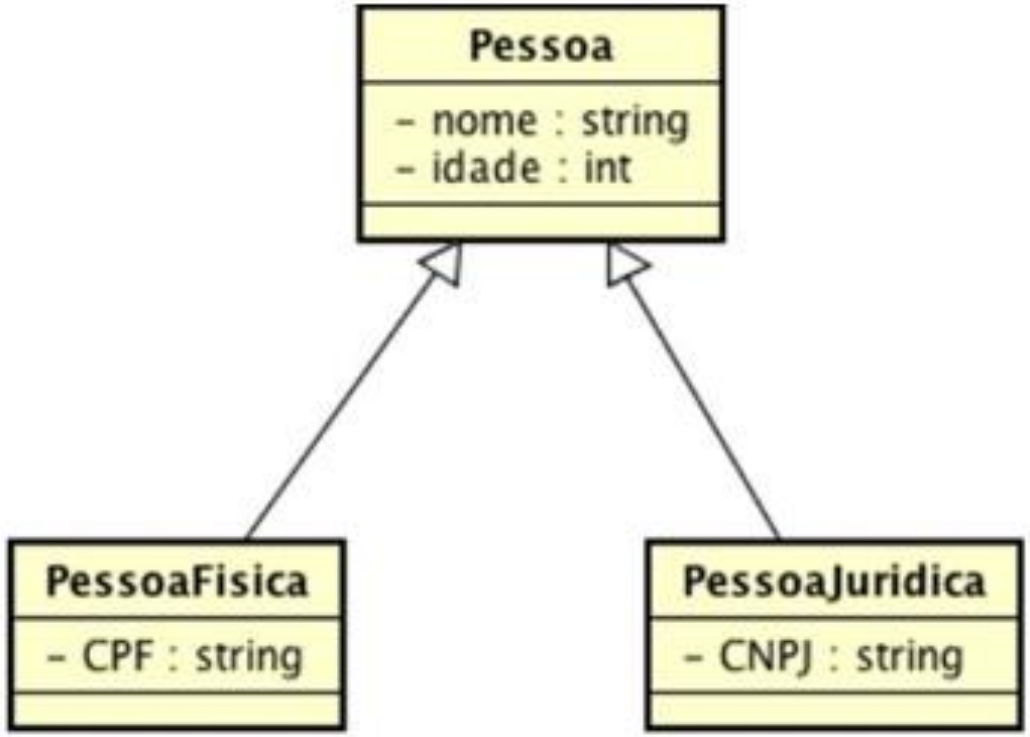
# Herança

- Todos são animais, mas cada um possui suas particularidades;
- Pode-se dizer que cachorro estende de animal, pois possui todos os atributos de animal, mais os seus atributos particulares; assim como o coelho estende de animal; o golfinho estende do animal; e o peixe estende do animal.



# Herança

- O mesmo ocorre entre Pessoa Física e Pessoa Jurídica, ou seja, possuem atributos semelhantes que podem ser estendidos para uma classe Pessoa.
- Exemplo:



# Herança

- A orientação a objetos permite modificar as classes, adicionando ou modificando atributos e métodos, tendo como base outra classe, ou seja, permite que o programador defina classes novas estendendo classes existentes.
- Usando classes derivadas, o programador pode explorar as variações que existem entre as classes de um programa. As classes diferentes podem compartilhar valores e operações.
- Quando uma ou mais classes “são do tipo” (por exemplo a classe PessoaFísica) significa que elas herdam características de uma classe “menos especializada” ou “mais genérica” (por exemplo a classe Pessoa).
- Em Python, usa-se o método *super* para invocar o construtor da classe pai (ou mãe, ou menos especializada, ou mais genérica).

```
class Pessoa:
    def __init__(self, nome, idade):
        self.__nome = nome
        self.__idade = idade

    def setnome(self, nome):
        self.__nome = nome

    def setidade(self, idade):
        self.__idade = idade

    def getnome(self):
        return self.__nome

    def getidade(self):
        return self.__idade

class PessoaFisica(Pessoa):
    def __init__(self,CPF,nome,idade):
       super().__init__(nome, idade)
       self.__CPF = CPF
    def getCPF(self):
       return self.__CPF
    def setCPF(self, CPF):
       self.__CPF = CPF
```

```
class PessoaJuridica(Pessoa):
    def __init__(self,CNPJ,nome,idade):
        super().__init__(nome, idade)
        self.__CNPJ = CNPJ
    def getCNPJ(self):
        return self.__CNPJ
    def setCNPJ(self, CNPJ):
        self.__CNPJ = CNPJ

pf = PessoaFisica('123.456.789-11',
'Fulano da Silva', 18)

pj = PessoaJuridica('12.123.123/0001-
01', 'Empresa XYZ Ltda.', '11/01/2001')

print('CPF:', pf.getCPF(), '\nNome:',
pf.getnome(), '\nIdade/Dt Fundação:',
pf.getidade())

print('\nCNPJ:', pj.getCNPJ(),
'\nNome:', pj.getnome(), '\nIdade/Dt
Fundação:', pj.getidade())
```

# Referências


- BORGES, Luiz Eduardo. **Python para Desenvolvedores**. São Paulo: Novatec, 2014.
- MENEZES, Nilo Ney Coutinho. **Introdução à Programação com Python**: Algoritmos e lógica de programação para iniciantes. 2 ed. 5 reimp. São Paulo: Novatec, 2017.
- PYTHON BRASIL. **Python e Programação Orientada a Objeto**. Disponível em: <https://wiki.python.org.br/ProgramacaoOrientadaObjetoPython>. Acesso em 20 mar. 2019.

# Exercícios

13. Crie uma classe chamada Ingresso que possui um valor em reais e um método imprimeValor().
    a) crie uma classe VIP, que herda Ingresso e possui um valor adicional. Crie um método que retorne o valor do ingresso VIP (com o adicional incluído).
    b) crie uma classe Normal, que herda Ingresso e possui um método que imprime: "Ingresso Normal".
    c) crie uma classe Camarote (que possui a localização do ingresso e métodos para acessar e imprimir esta localização). A classe herda a classe VIP.

14. Crie a classe Imovel, que possui um endereço e um preço.
    a) crie uma classe Novo, que herda Imovel e possui um adicional no preço. Crie métodos de acesso e impressão deste valor adicional.
    b) crie uma classe Velho, que herda Imovel e possui um desconto no preço. Crie métodos de acesso e impressão para este desconto.

# Exercícios

15. **Classe Bomba de Combustível**: Faça um programa utilizando classes e métodos que:

**a)** Possua uma classe chamada bombaCombustível, com no mínimo esses atributos:

i. tipoCombustivel.

ii. valorLitro

iii. quantidadeCombustivel

**b)** Possua no mínimo esses métodos:

i. abastecerPorValor( ) – método onde é informado o valor a ser abastecido e mostra a quantidade de litros que foi colocada no veículo

ii. abastecerPorLitro( ) – método onde é informado a quantidade em litros de combustível e mostra o valor a ser pago pelo cliente.

iii. alterarValor( ) – altera o valor do litro do combustível.

iv. alterarCombustivel( ) – altera o tipo do combustível.

v. alterarQuantidadeCombustivel( ) – altera a quantidade de combustível restante na bomba.

*OBS.*: Sempre que acontecer um abastecimento é necessário atualizar a quantidade de combustível total na bomba.