

# **Python – Orientação a Objetos – Parte 1**

Introdução à Programação

SI1

# Conteúdo

- Orientação a objetos
  - Objeto
  - Classe
  - Herança
  - Encapsulamento
  - Polimorfismo

# Motivação

- Realidade Atual
  - Sistemas de alta complexidade
  - Sistemas maiores, mais complexos e mais voláteis
- A mudança para alcançar a qualidade e produtividade está na ...

Reutilização

# Paradigma OO

- Um paradigma é uma **forma de abordar um problema**
- O paradigma da **orientação a objetos** surgiu no fim dos anos 60
- Hoje em dia, praticamente suplantou o paradigma anterior, o **paradigma estruturado...**

# Paradigma OO

- Um paradigma é uma **forma de abordar um problema**
- Alan Kay, um dos pais do paradigma da orientação a objetos, formulou a chamada **analogia biológica**
- “Como seria um sistema de **software** que funcionasse como um **ser vivo**?”

# Paradigma OO

- Cada “**célula**” interage com outras células através do envio de **mensagens** para realizar um objetivo comum
  - Cada célula se comporta como uma **unidade autônoma**
- De uma forma mais geral, Kay pensou em como construir um **sistema de software** a partir de **agentes autônomos** que **interagem** entre si
- Com isso, estabeleceu os princípios da **orientação a objetos**

# Análise e Programação OO

- Análise orientada a objetos
- Programação orientada a objetos
  - Consiste em utilizar objetos computacionais para implementar as funcionalidades de um sistema.

# **Princípios OO**

Tudo é um objeto!

# Objetos

- Entidades que possuem **dados** e **instruções** sobre como manipular estes dados.
- Estão ligados à solução do problema.

# Modelagem de Objetos

- Software Gráfico
  - Objetos: Círculos, Linhas, etc.
- Software BD
  - Objetos: Tabelas, Linhas, Campos, etc.
- Software Comercial
  - Objetos: Pedidos, Produtos, Clientes.

# Princípios OO

- **Tudo é um objeto**
- Pense em um objeto como uma **super variável**:
  - O objeto armazena dados, também pode-se fazer requisições a esse objeto, pedindo que ele execute operações
- **Elementos conceituais** no problema que você está tentando resolver (cachorros, livros, sócios, empréstimos, etc.) como um **objeto** de um programa

# Exemplo

```
>>> s = "quem parte e reparte, fica com a  
maior parte"
```

```
>>> s.find("parte")
```

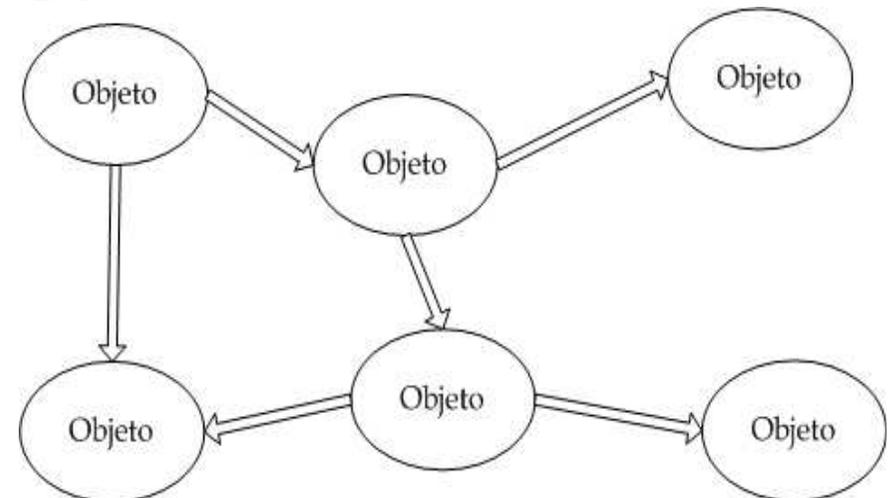
```
5
```

```
>>> lista=[1,2]
```

```
>>> lista.extend([3,4])
```

```
>>> lista
```

```
[1, 2, 3, 4]
```



# Princípios OO

- Um programa é uma **coleção de objetos** dizendo uns aos outros o que fazer
- Para fazer uma **requisição** a um objeto envia-se **uma mensagem** para este objeto
- Uma mensagem é uma chamada de um **método** pertencente a um objeto em particular

# Princípios OO

- **Todo objeto tem um tipo**
- Cada objeto é uma **instância** de uma classe, onde a classe define um tipo
  - **Classe professor, objeto Jones**

# Classes

- Podemos descrever o cachorro Bilú em termos de seus **atributos** físicos:
  - é pequeno
  - sua cor principal é castanha
  - olhos pretos
  - orelhas pequenas e caídas,
  - rabo pequeno



# Classes

- Podemos também descrever algumas **ações** que ele faz (temos aqui os métodos):
- balança o rabo
- foge e se deita quando leva reclamação
- late quando ouve um barulho ou vê um cão ou gato
- atende quando o chamamos pelo seu nome

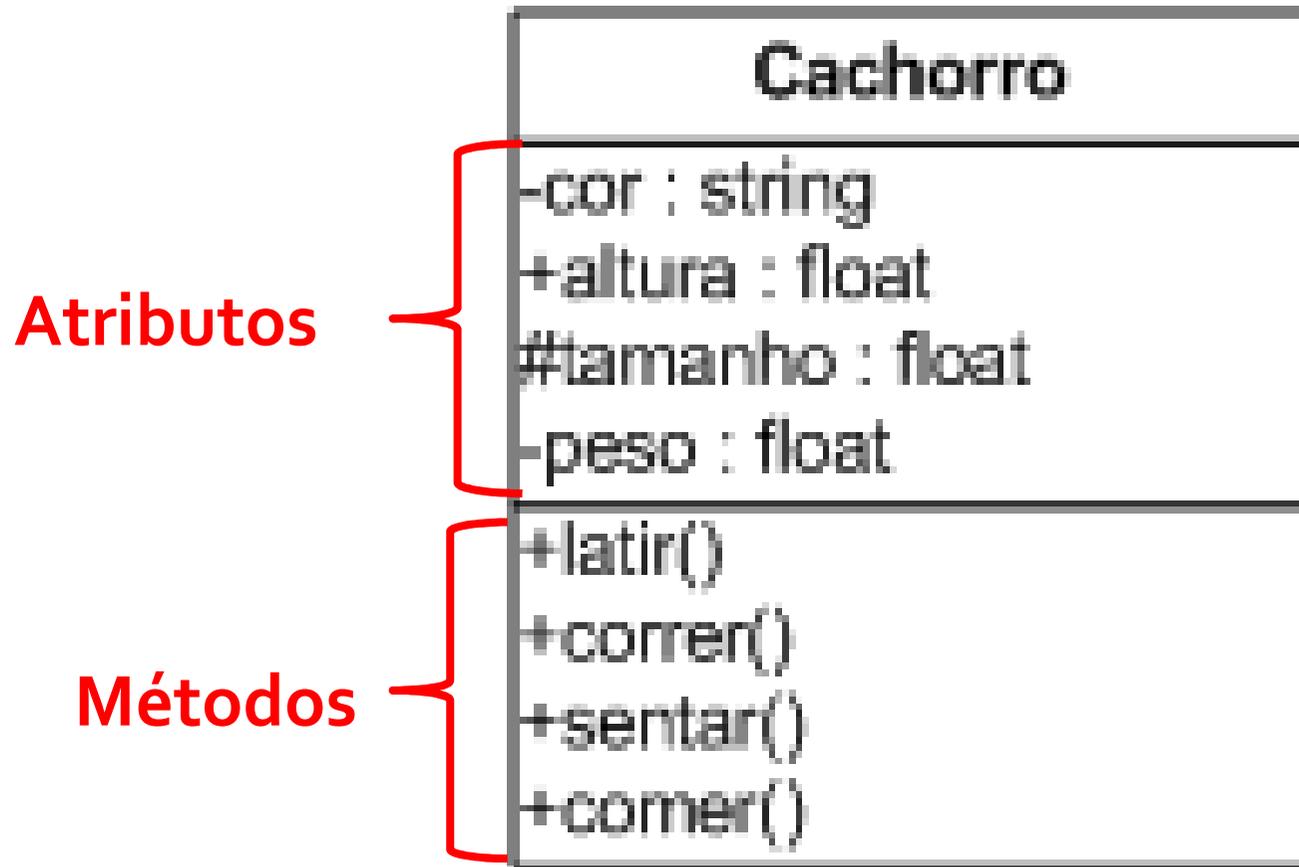


# Classes

- Representação do cachorro Bilú:
  - **Propriedades** : [Cor do corpo : castanha; cor dos olhos : pretos; altura: 15 cm; comprimento: 38 cm largura : 24 cm]
  - **Métodos** : [balançar o rabo , latir , correr, deitar , sentar ]



# Representação de Classe



# Objeto

- Um **objeto** é qualquer coisa, real ou abstrata, sobre a qual **armazenamos dados** e realizamos **operações** que manipulam tais dados
  - Pertencem a classes
- **Unidade básica** de modularização de um sistema OO
- Um objeto de uma classe possui:
  - **Atributos** → características ou propriedades que definem o objeto.
  - **Comportamento** → conjunto de ações pré-definidas (métodos)

# Objetos - Exemplos

- Pássaro



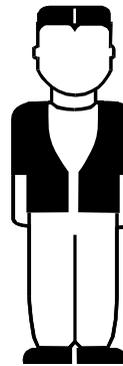
**Características:**

cores  
forma do bico  
tipo do vôo

**Comportamento:**

voar()  
piar()

- Pessoa



**Características:**

cor dos olhos: azuis  
data nascimento: 16/02/70  
peso: 70kg  
altura: 1,70m

**Comportamento:**

andar  
falar  
comer  
rir

# Exemplo

- Telefone



## Características:

cor: azul

discagem: tone

## Comportamento:

tocar()

discar()

# Exemplo

- Ônibus

**Características:**

cor amarela  
30 assentos  
a diesel

**Comportamento:**

frear  
andar  
correr  
buzinar  
acelerar



# OO

- Em resumo, a expressão **orientada a objetos** significa que
  - o aplicativo é organizado como uma coleção de objetos que incorporam tanto a estrutura como o comportamento dos dados
- Objetos pertencem à **classes**

# Classe

- Abstrações utilizadas para representar um conjunto de objetos com *características* e *comportamento idênticos*
- Uma classe pode ser vista como uma *“fábrica de objetos”*

# Classe

- Objetos são “**instâncias**” de uma classe
  - Todos os objetos são instâncias de alguma classe
- Todos os objetos de uma classe são **idênticos** no que diz respeito a sua interface e implementação
  - o que difere um objeto de outro é seu **estado** e sua **identidade**

# Classe - Exemplo

instância da  
classe (objeto)



## Características:

cor das penas: azuis

formato do bico: fino

velocidade de vôo: rápida

## Comportamento:

voar

piar

Pássaro
corPenas formatoBico velocidadeVoo
voar() piar()

# Classe - Exemplo

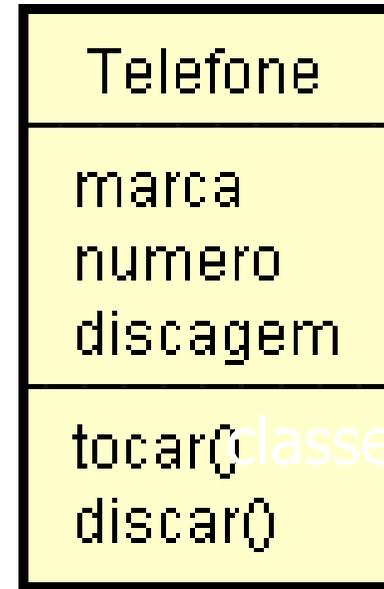


## Características:

marca: Siemens  
número: 2576-0989  
discagem: pulso

## Comportamento:

tocar  
discar



instância da classe  
(objeto)

# Classe - Exemplo



## Características:

marca: Nokia  
número: 99193467  
discagem: tom

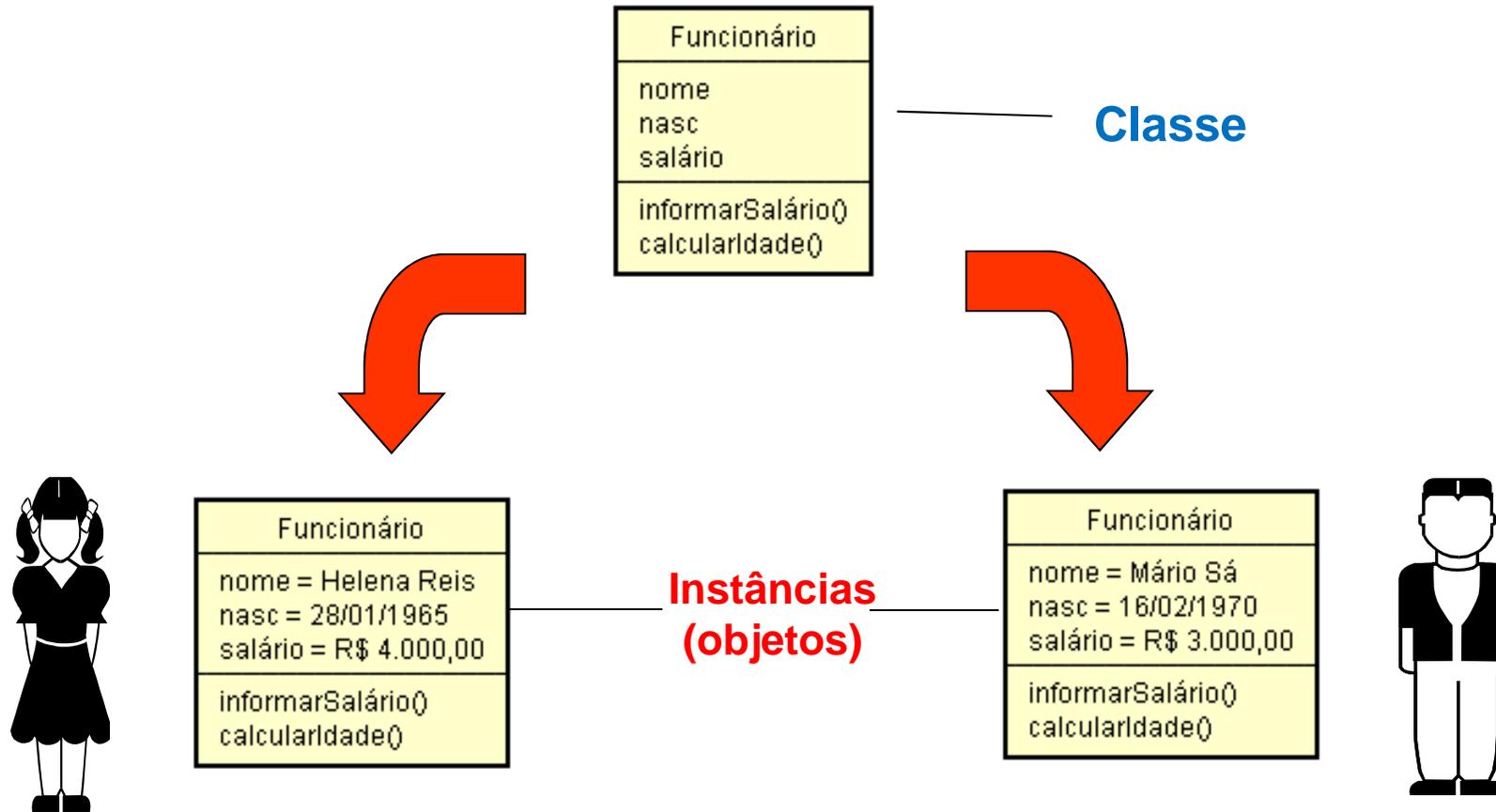
## Comportamento:

tocar  
discar

Telefone
marca numero discagem
tocar() discar()

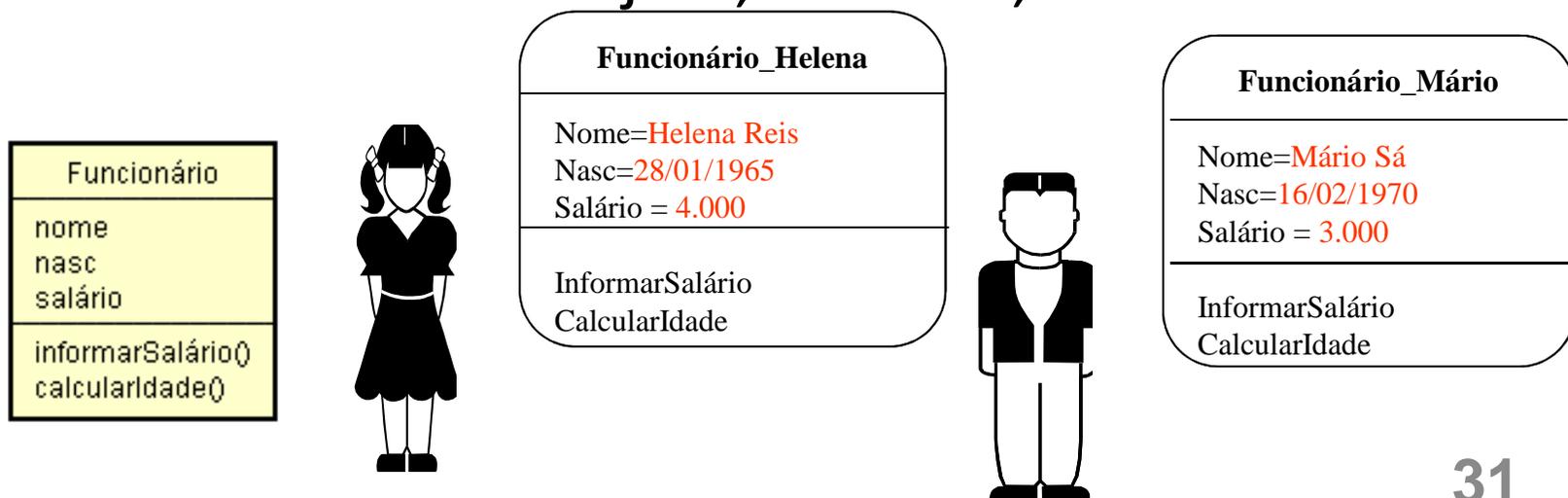
instância da classe  
(objeto)

# Classes



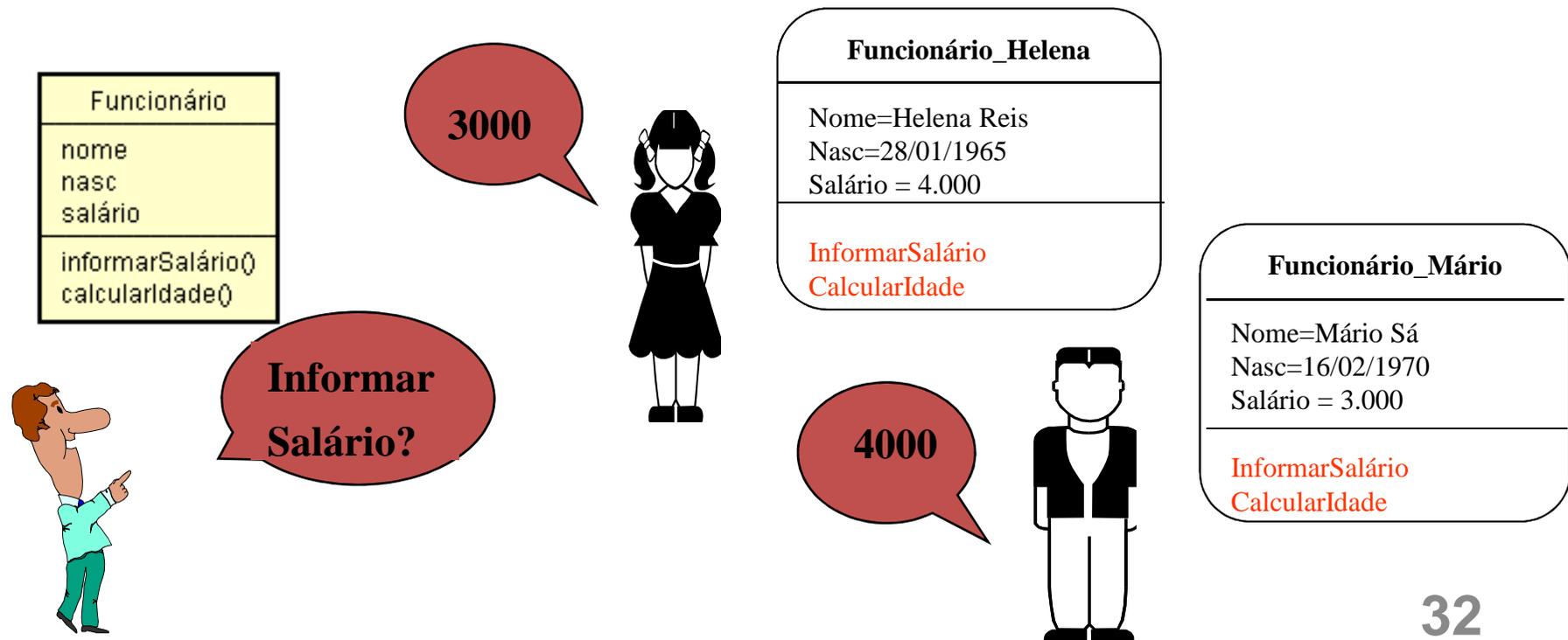
# Classe - Atributos

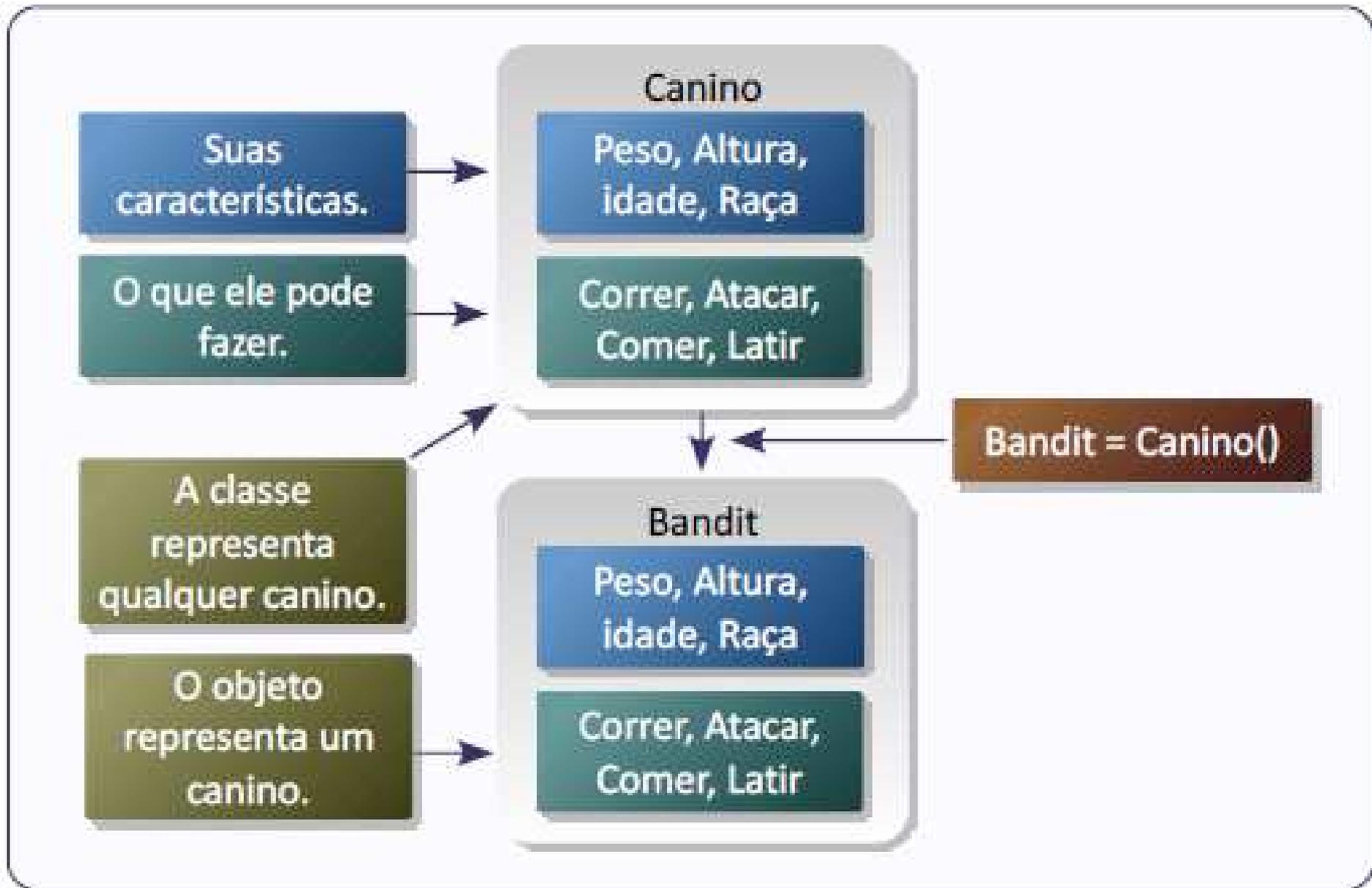
- Descrevem as **características** das instâncias de uma classe
- Seus valores definem o **estado** do objeto
- O estado de um objeto pode mudar ao longo de sua existência
- A identidade de um objeto, contudo, nunca muda



# Classe - Operações

- Representam o **comportamento** das instâncias de uma classe
- Correspondem às **ações** das instâncias de uma classe





# Classe em Python

- A maneira mais simples é:

```
class nomeClasse:
```

```
    var = valor
```

```
    ...
```

```
    var = valor
```

```
    def metodo (self, ... arg):
```

```
        ...
```

```
    def metodo (self, ... arg):
```

```
        ...
```

# Classe em Python

- Os métodos sempre têm **self** como primeiro argumento
  - **self** se refere a uma instância da classe
- Uma nova instância da classe é criada usando **nomeClasse()**

# Classe em Python

- As **variáveis** e os **métodos** são escritos precedidos pelo **nome da classe** e por um **ponto** (.)
  - A variável **v** definida numa classe **C** é escrita **C.v**
- Os métodos sempre têm **self** como primeiro argumento
  - **self** se refere a uma instância da classe
- Uma nova instância da classe é criada usando **nomeClasse()**

# Construtores

- O método **inicia** foi usado para inicializar atributos e é conhecido como **construtor** da classe
- Python suporta construtores que podem ser chamados **automaticamente** na criação de instâncias
  - Basta definir na classe um método chamado **`__init__`**
  - Este método é chamado **automaticamente** durante a criação de uma nova instância da classe, sendo que os **argumentos** são passados **entre parênteses** após o nome da classe

# Atributos

- Um atributo `attr` associado a uma instância `obj` tem nome `obj.attr`
- Se queremos nos referir a um atributo `attr` de um objeto **dentro da própria classe**, usamos o nome `self.attr`

# Exemplo

```
>>> class Exemplo:  
    def __init__(self,a=2,b=3):  
        self.a = a  
        self.b = b  
    def f(self,x):  
        return self.a*x+self.b
```

```
>>> obj1 = Exemplo()  
>>> obj1.a  
2  
>>> obj1.b  
3  
>>> obj2 = Exemplo(8,1)
```

```
>>> obj2.b  
1  
>>> obj1.f(7)  
17  
>>> obj2.f(7)  
57
```

# Exemplo

```
>>> class Retangulo:
    lado_a = None
    lado_b = None
    def __init__(self, lado_a, lado_b):
        self.lado_a = lado_a
        self.lado_b = lado_b
        print "Criada uma nova instância Retangulo"
    def calcula_area(self):
        return self.lado_a * self.lado_b
    def calcula_perimetro(self):
        return 2 * self.lado_a + 2 * self.lado_b
```

# Exemplo

```
>>> obj = Retangulo(4,6)
```

Criada uma nova instancia Retangulo

```
>>> obj.lado_a
```

4

```
>>> obj.lado_b
```

6

```
>>> obj.calcula_area()
```

24

```
>> obj.calcula_perimetro()
```

20

# Exemplo

```
>>> class ContaCorrente:
    def __init__(self, numero):
        self.numero = numero
        self.saldo = 0.0

    def debitar(self, valor):
        self.saldo = self.saldo - valor

    def creditar(self, valor):
        self.saldo = self.saldo + valor
```

# Exemplo

```
>>> c = ContaCorrente("1234")
```

```
>>> c.saldo
```

```
0.0
```

```
>>> c.creditar(1000)
```

```
>>> c.saldo
```

```
1000.0
```

```
>>> c.debitar(342)
```

```
>>> print(c.numero, c.saldo)
```

```
1234 658.0
```

# Classe – Exemplo 1

```
class Pessoa:
    nome = None
    idade = None

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

    def getAnoNascimento(self, anoAtual):
        return anoAtual - idade

pessoa = Pessoa("Pedro", 21)
print(pessoa.getAnoNascimento(2013))
```

# Classe – Exemplo 2

Automovel
+ placa : str
__init__(str) : None
get_placa() : str
dirigir(int) : None

métodos

```
class Automovel:
```

construtor

```
    def __init__(self, placa='XX-123'):  
        self.placa = placa
```

```
    def get_placa(self):  
        return self.placa
```

self

```
    def dirigir(self, velocidade):  
        print('Estou dirigindo a %d' \  
              ' km/h' % velocidade)
```

# Classe – Exemplo 3

```
class Matriz:
    dic = None
    tamanho = None

    def __init__(self):
        self.dic = {}
        self.tamanho = 0

    def __init__(self, matriz, tamanho):
        self.dic = matriz
        self.tamanho = tamanho

    def imprimeMatriz(self):
        for i in range(tamanho):
            stLinha = ""
            for j in range(tamanho):
                stLinha += dic[(i,j)] + " "
            print(stLinha)
```

# Encapsulamento

- Na terminologia da orientação a objetos, diz-se que um objeto possui uma **interface**.
- A interface de um objeto é como ele **aparece** para os demais objetos:
  - Suas características, sem **detalhes internos**
- A interface de um objeto define os **serviços** que ele pode realizar e conseqüentemente as **mensagens que ele recebe**
  - **Um objeto é “visto” através de seus métodos**

# Encapsulamento

- Encapsulamento é a **proteção** dos atributos ou métodos de uma classe.
- Em Python existem somente o **public** e o **private** e eles são definidos no próprio nome do atributo ou método.
- Atributos ou métodos iniciados por no máximo **dois sublinhados** (underline) são **privados** e todas as outras formas são **públicas**

# Exemplo

```
class Teste1:
```

```
    a = 1 # atributo publico
```

```
    __b = 2 # atributo privado da  
    classe Teste1
```

```
class Teste2(Teste1):
```

```
    __c = 3 # atributo privado da  
    classe Teste2
```

```
    def __init__(self):
```

```
        print self.a
```

```
        print self.__c
```

```
>>> t1 = Teste1()
```

```
>>> print(t1.a)
```

```
1
```

```
>>> t2 = Teste2()
```

```
1
```

```
3
```

```
>>> print(t2.__b)
```

```
# Erro, pois __b é privado a  
    classe A.
```

```
>>> print(t2.__c)
```

```
# Erro, __c é um atributo privado,  
    somente acessado pela classe
```

# EXERCÍCIOS

# Exercícios

1. **Classe Triângulo:** Crie uma classe que modele um triângulo:

- Atributos: LadoA, LadoB, LadoC
- Métodos: calcular Perímetro, getMaiorLado;

Crie um programa que utilize esta classe. Ele deve pedir ao usuário que informe as medidas de um triângulo. Depois, deve criar um objeto com as medidas e imprimir sua área e maior lado.

# Exercícios

2. **Classe Funcionário:** Implemente a classe Funcionário. Um funcionário tem um nome e um salário. Escreva um construtor com dois parâmetros (nome e salário) e o método aumentarSalario (porcentualDeAumento) que aumente o salário do funcionário em uma certa porcentagem. Exemplo de uso:

```
harry=funcionário("Harry",25000)  
harry.aumentarSalario(10)
```

Faca um programa que teste o método da classe.

3. Crie uma classe Livro que possui os atributos nome, qtdPaginas, autor e preço.
  - Crie os métodos getPrice para obter o valor do preço e o método setPreco para setar um novo valor do preço.

Crie um código de teste

# Exercício

4. Implemente uma classe Aluno, que deve ter os seguintes atributos: nome, curso, tempoSemDormir (em horas). Essa classe deverá ter os seguintes métodos:

- estudar (que recebe como parâmetro a qtd de horas de estudo e acrescenta tempoSemDormir )
- Dormir (que recebe como parâmetro a qtd de horas de sono e reduz tempoSemDormir )

Crie um código de teste da classe, criando um objeto da classe aluno e usando os métodos estudar e dormir. Ao final imprima quanto tempo o aluno está sem dormir

# Exercícios

5. **Classe carro:** Implemente uma classe chamada Carro com as seguintes propriedades:

- Um veículo tem um certo consumo de combustível (medidos em km / litro) e uma certa quantidade de combustível no tanque.
- O consumo é especificado no construtor e o nível de combustível inicial é 0.
- Forneça um método andar( ) que simule o ato de dirigir o veículo por uma certa distância, reduzindo o nível de combustível no tanque de gasolina. Esse método recebe como parâmetro a distância em km.
- Forneça um método obterGasolina( ), que retorna o nível atual de combustível.
- Forneça um método adicionarGasolina( ), para abastecer o tanque.
- Faça um programa para testar a classe Carro. Exemplo de uso:  
meuFusca = Carro(15); # 15 quilômetros por litro de combustível.  
meuFusca.adicionarGasolina(20); # abastece com 20 litros de combustível.  
meuFusca.andar(100); # anda 100 quilômetros.  
meuFusca.obterGasolina() # Imprime o combustível que resta no tanque.

# Exercícios

6. Crie uma classe *Aluno*, que possui como atributo um nome e cpf. Crie outra classe chamada *Equipe*, que possui como atributo uma lista de participantes do tipo *Aluno* e outro atributo chamado *projeto*.

Crie uma terceira classe chamada *GerenciadorEquipes*. Essa classe possui como atributo uma lista de todas as equipes formadas. Ela deverá possuir o método *criarEquipe*, que recebe uma lista de alunos de uma equipe e diz se a equipe pode ser formada ou não. Caso não haja nenhum aluno da equipe a ser formada em uma outra equipe com o mesmo projeto, então a equipe é criada e acrescentada à lista. Caso contrário é informada que a equipe não pode ser criada.

# Bibliografia

- Livro “Como pensar como um Cientista de Computação usando Python” – Capítulo 12
  - <http://pensarpython.incubadora.fapesp.br/portal>
- Python Tutorial
  - <http://www.python.org/doc/current/tut/tut.html>
- Dive into Python
  - <http://www.diveintopython.org/>
- Python Brasil
  - <http://www.pythonbrasil.com.br/moin.cgi/DocumentacaoPython#head5a7ba2746c5191e7703830e02d0f5328346bcaac>