



[Aula 7] Linguagem de Programação

Análise e Desenvolvimento de Sistemas

Orientação a Objetos em Python – Prof. Jean Zahn

jeanozahn@gmail.com

Na aula de hoje

- ▶ Programação Estruturada
- ▶ Orientação a Objetos
 - ▶ Conceitos básicos
 - ▶ Objetos
 - ▶ Classes
 - ▶ Relacionamentos entre classes
- ▶ Análise
- ▶ Projeto



Paradigma (*pa-ra-dig-ma*)

► Substantivo masculino cujo significado é :

1. **Modelo, padrão** ou
2. Termo com o qual Thomas Kuhn (v. kuhniano) designou as realizações científicas (p. ex., a dinâmica de Newton ou a química de Lavoisier) que geram modelos que, por período mais ou menos longo e de modo mais ou menos explícito, orientam o desenvolvimento posterior das pesquisas exclusivamente na busca da solução para os problemas por elas suscitados.



Motivação

- ▶ **Como você realmente escreve um grande software?**
 - ▶ Quanto tempo levará?
 - ▶ Como o código será organizado?
 - ▶ Dá para reaproveitar algum código?
 - ▶ Como será testado?
 - ▶ Será fácil depurar os bugs?
 - ▶ Como se dividem as tarefas entre mais programadores?
 - ▶ Como juntar todos os códigos ao final?
 - ▶ Funciona?

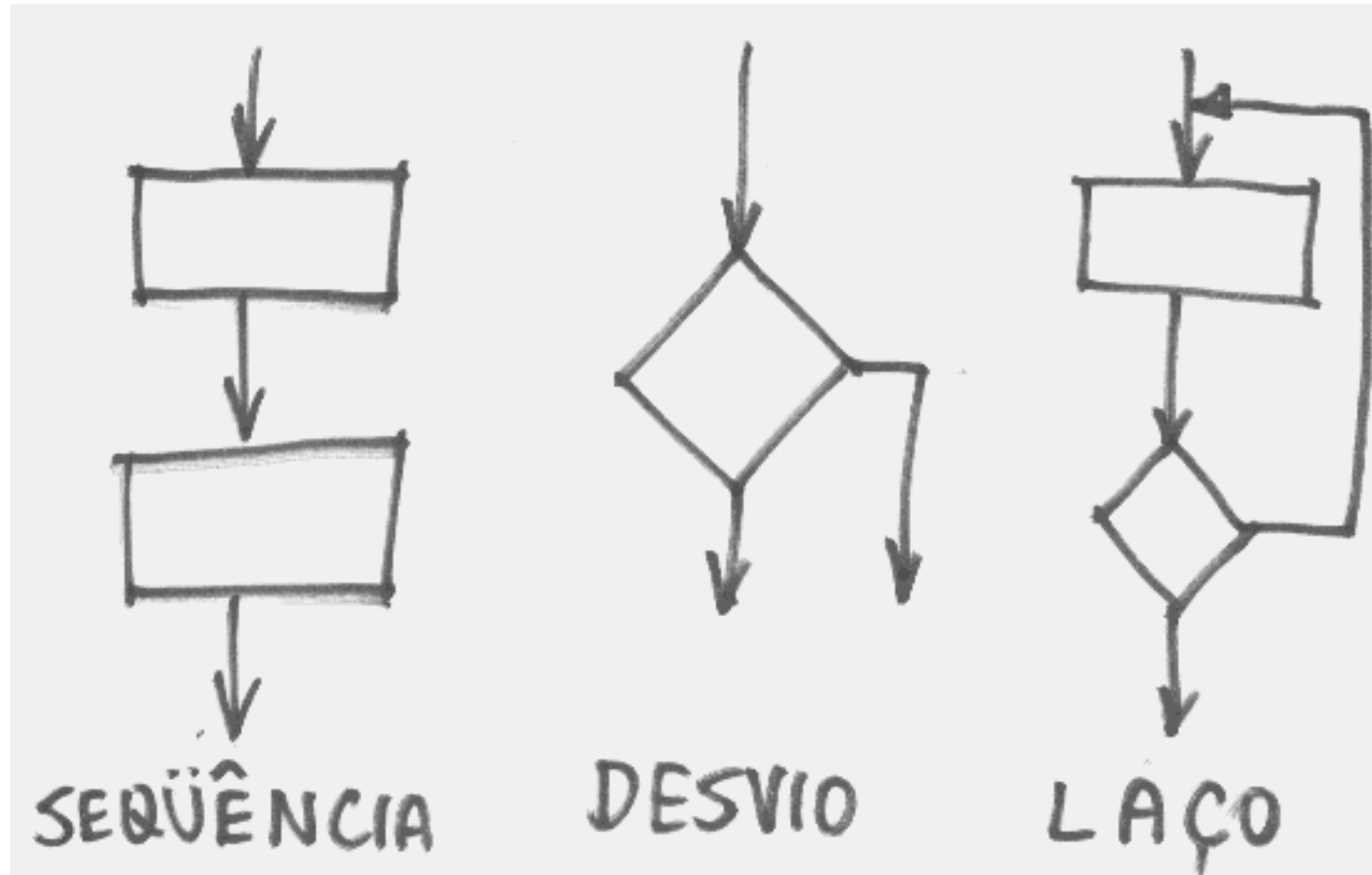




Programação Estruturada



Programação Estruturada



Programação Estruturada

- ▶ A programação estruturada tem como principal foco as **ações**
 - ▶ Procedimentos e Funções
- ▶ Fornece maior controle sobre o fluxo de execução de um programa
 - ▶ Estruturas de sequência;
 - ▶ Estruturas de decisão;
 - ▶ Estruturas de repetição.



Programação Estruturada


- ▶ As linguagens estruturadas são de entendimento relativamente fácil
 - ▶ Por isso são utilizadas em cursos introdutórios.
- ▶ No entanto, são focadas em **como** uma tarefa deve ser feita
 - ▶ E não em **o que** deve ser feito.
- ▶ Mistura **tratamento de dados** e **comportamento** do programa.



Programação Estruturada

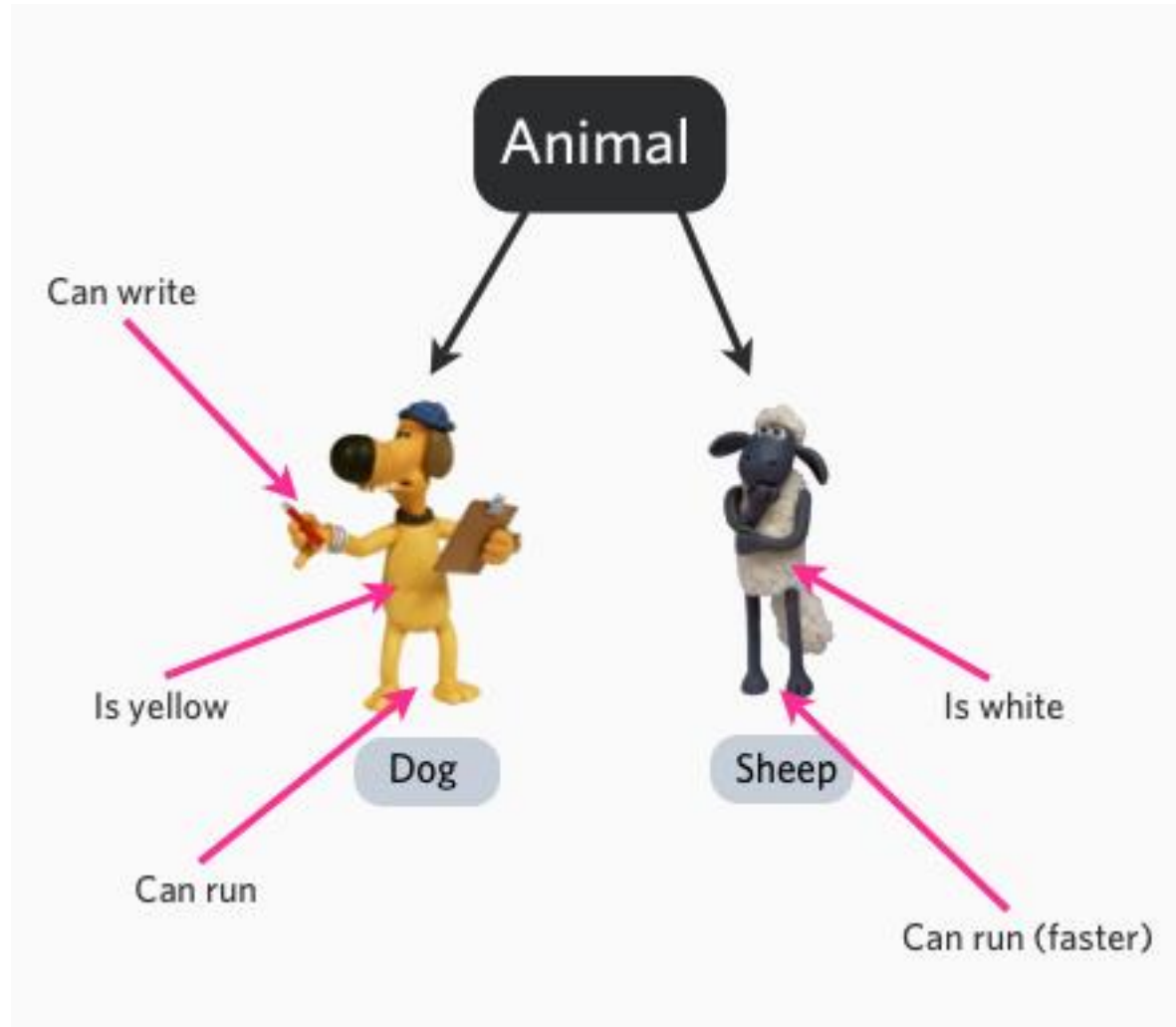
- ▶ A programação estruturada ainda é muito influente
 - ▶ Para cada situação uma ferramenta.
- ▶ Para problemas simples e diretos, ainda é a melhor solução.





Orientação a Objetos

Orientação a Objetos



Orientação a Objetos

- ▶ O conceito de Orientação a Objetos data do final da década de 60 e início da década de 70
 - ▶ Simula 67 (60's);
 - ▶ Smalltalk (70's);
 - ▶ C++ (80's).
- ▶ Surgiu da necessidade de modelar sistemas mais complexos.



Orientação a Objetos

- ▶ Como melhor modelar o mundo real utilizando um conjunto de componentes de software?
- ▶ Considerando que nosso mundo é composto de objetos, porquê não utilizá-los?
- ▶ A ideia é modelar utilizando objetos, determinando como eles devem se comportar e como deve interagir entre si.



Orientação a Objetos

- ▶ Este paradigma de programação tenta ser o mais óbvio, natural e exato possível;
- ▶ São conceitos essenciais:
 - ▶ Classes e objetos;
 - ▶ Atributos, Métodos e Mensagens;
 - ▶ Herança e Associação;
 - ▶ Encapsulamento;
 - ▶ Polimorfismo;
 - ▶ Interfaces.





Objetos

Objetos

- ▶ **abstrair** (*abs-tra-ir*)
 - ▶ Verbo transitivo cujo significado é:
 - ▶ Separar.V. i. Considerar separadamente.V. p. Concentrar-se.Alhear-se.
- ▶ Em outras palavras, captar a essência de um problema ou contexto e considerar o que realmente importa.



Objetos

- ▶ Objetos são a chave para entender a OO;
- ▶ Se olharmos em nossa volta, encontraremos vários exemplos de objetos reais:
 - ▶ Celular;
 - ▶ Mesa;
 - ▶ Computador;
 - ▶ Janela;
 - ▶ Lâmpada;
 - ▶ Etc.



Objetos

- ▶ Os objetos reais possuem duas características
 - ▶ Estado (Atributos);
 - ▶ Comportamento.
- ▶ Por exemplo, um cachorro
 - ▶ Estado: nome, cor, raça, fome...
 - ▶ Comportamento: latindo, abanando o rabo, comendo...
- ▶ Uma bicicleta
 - ▶ Estado: marcha atual, freio, rotação...
 - ▶ Comportamento: mudando de marcha, freando...



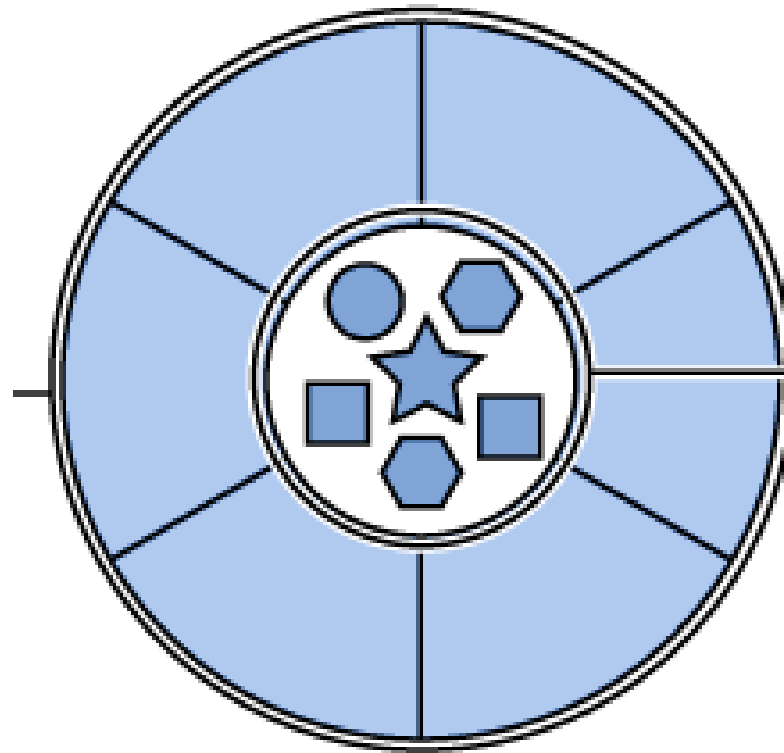
Objetos

- ▶ Quais são as características de uma lâmpada?
- ▶ Quais são as características de um projetor?
 - ▶ E como tratamos a lâmpada do projetor?
- ▶ Objetos variam em complexidade
 - ▶ Porém, os detalhes relevantes dependem do contexto;
 - ▶ Esta análise de características é traduzível em orientação a objetos.



Objetos

Métodos
(Comportamento)



Atributos
(Estado)





Atributos e Métodos

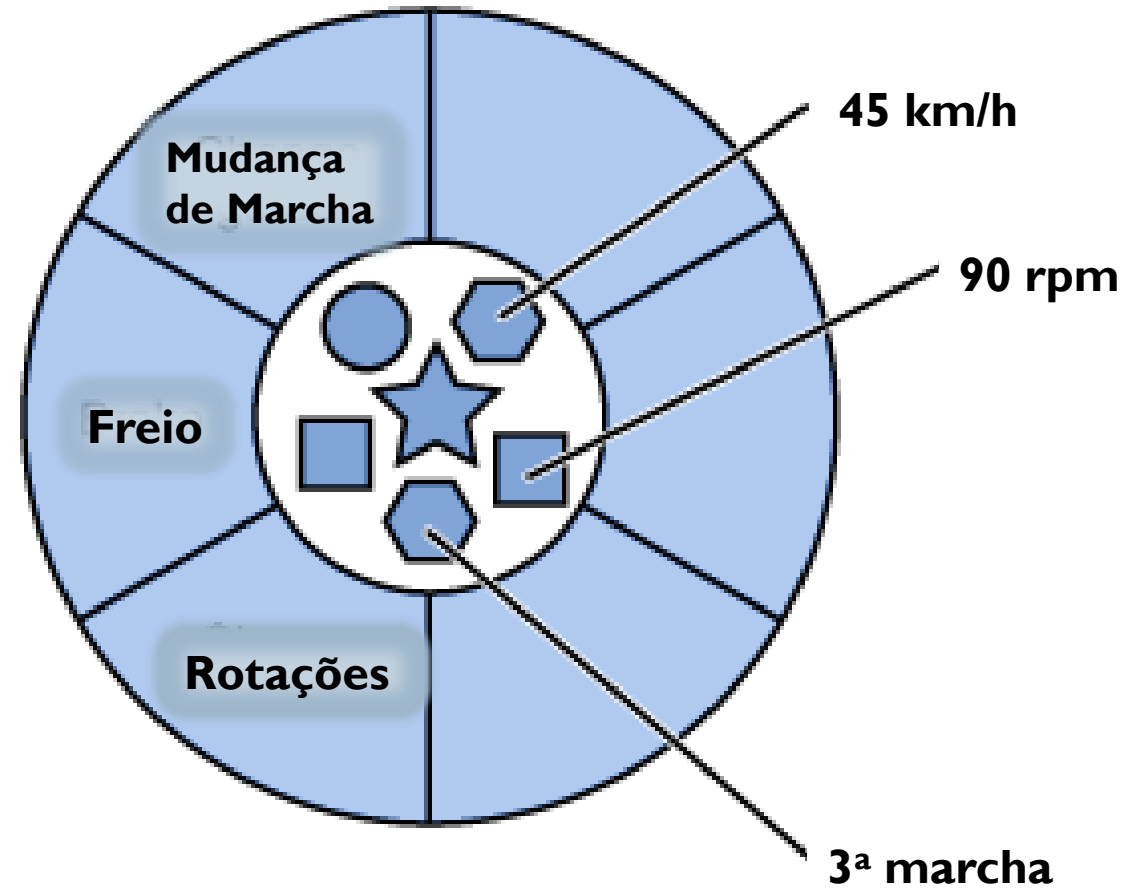


Atributos e Métodos

- ▶ Um objeto de software é **conceitualmente** similar aos objetos reais
- ▶ Objetos armazenam seu estado em **atributos**
 - ▶ Correspondentes às variáveis em programação estruturada.
- ▶ Objetos expõem seu comportamento através de **métodos**
 - ▶ Correspondentes às funções em programação estruturada.



Atributos e Métodos



Objetos

- ▶ Empacotar o código em objetos individuais fornece:
 - ▶ Modularidade
 - ▶ Objetos são independentes.
 - ▶ Ocultação de informação
 - ▶ Os detalhes da implementação de um objeto permanecem ocultos.
 - ▶ Reuso
 - ▶ Objetos podem ser reutilizados em diferentes programas.
 - ▶ Plugabilidade
 - ▶ Objetos podem ser substituídos em um programa, como peças.



Definindo uma classe

▶ Conceitos necessários

- ▶ Classe: Molde
- ▶ Objeto: Agente ativo na programação
- ▶ Método: Capacidade de ação do agente ativo
- ▶ Atributo: Características do agente ativo

▶ Exemplo:

- ▶ Classe: Humorista
- ▶ Objeto: Tiririca
- ▶ Método: Contar Piadas, Imitar Pessoas
- ▶ Atributo: Baixo, 50 anos



Métodos em classes

- ▶ Definir um **método** em uma **classe** , basta incluir a definição da função seguindo o escopo de bloco da classe.
- ▶ Em todos métodos associados à instância definido dentro de uma classe devem ter o argumento **self** definido como primeiro argumento.
- ▶ Há geralmente um método especial **__init__** definido na maioria das classes.



Definição de uma classe

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
```



Instanciando Objetos

- ▶ Não há “**new**” como feito em Java!

- ▶ `a = student("Sheldon", 34)` (** sem o operador new!)

- ▶ `__init__` serve como construtor de uma classe. Geralmente faz o trabalho de inicialização.
- ▶ Não há limite para o número de argumentos passados para o método `__init__`. Como em qualquer outra função, os argumentos podem ser definidos com valores **default**, tornando-os assim opcionais ao chamador



Instanciando Objetos

- ▶ **self** : O primeiro argumento de qualquer método é a referência para a própria instância da classe
- ▶
- ▶ Em **__init__**, **self** referencia o objeto criado recentemente, e em outros métodos, referencia a instância de qual o método foi invocado.
- ▶ Similar ao **this** usado em Java ou C++
- ▶ Porém, Python usa mais **self** do que Java com **this**



`__init__`

```
Class Carro:
    def __init__(self):
        self._nrodas = 4

    def set_nrodas(self,n):
        self._nrodas = n
```

```
>>> gol = Carro()
```

```
>>> gol._nrodas
```

4

```
>>> gol.set_nrodas(10)
```

```
>>> gol._nrodas
```

10



self

- ▶ Não é necessário incluí-lo no método que faz a chamada do mesmo, apenas na definição!
- ▶ Python passa ele automaticamente.

```
a = Automovel()  
print a.get_placa()
```



Deletando instâncias

- ▶ Quando estiver finalizado com o objeto, você não precisa deletá-lo ou liberá-lo explicitamente.
- ▶ Python possui *garbage collection* de forma automática.
- ▶ Python irá automaticamente detectar quando todas as referências para um trecho de memória estiver não sendo mais referenciado. Automaticamente, a memória é liberada.
- ▶ Poucos *leaks* de memória, e **não há métodos destrutores** em Python!



Acessibilidade

- ▶ Acesso de métodos e atributos
- ▶ **Diretamente** `objeto.atributo` ou por algum método
 - ▶ `objeto.getAtributo()`

```
a = Automovel()  
print a.n_rodas
```





Acesso de atributos e métodos

Acessibilidade

▶ Atributos (class e ou instâncias)

▶ Privados

- ▶ Atributos e métodos só podem ser acessados dentro da classe, usa-se “__” no início do nome.

▶ Protected

- ▶ Apenas convenção e usa-se apenas um “_” no nome de métodos ou atributos

```
Class Carro:                                #Classe
    _nrodas =4                               #Privado
    __nparafusos = 3000 #Protected

>>> gol = Carro()                           #Instância
>>> go.__nparafusos                          #Error
```



Atributos

Como declarar os membros de uma classe?

- ▶ Exceto métodos, todos os demais dados dentro de uma classe são armazenados como atributos.
- ▶ **Atributos de instância**
 - ▶ Variáveis que pertencem a uma instância particular da classe
 - ▶ Cada instância tem o seu próprio valor para o atributo
 - ▶ Os mais frequentes em classes
- ▶ **Atributos de classe**
 - ▶ Variáveis que pertencem à classe como um todo.
 - ▶ Todas as instâncias da classe compartilham o mesmo atributo (valor).
 - ▶ Conhecidos como “estáticos” em outras linguagens



Atributos

- ▶ Atributos de instância são criados e inicializados pelo método `__init__()`
- ▶ Simplesmente atribuindo um valor a um rótulo
- ▶ Dentro da classe, referir-se ao atributo usando `self`
 - ▶ Exemplo: `self.full_name`



Atributos

- ▶ Atributos de classe são compartilhados (apenas uma cópia) por todas instâncias da classe.
 - ▶ Qualquer instância alterá-lo, o valor é alterado para todas instâncias.
 - ▶ Atributos de classe são definidas:
 - ▶ Dentro da definição de uma classe
 - ▶ Fora de quaisquer métodos da classe
- ▶ Já que estes atributos são compartilhados por todas instâncias de uma classe, eles são acessados através de uma notação diferente:
 - ▶ `self.__class__.name`



Exercícios

I. **Classe Triangulo:** Crie uma classe triangulo:

- ▶ Atributos: Lado A, Lado B e Lado C
- ▶ Métodos: Calcular Perímetro, getMaiorLado

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



Exercícios

- ▶ Crie uma classe **Livro** que possui os atributos **nome**, **qtdPaginas**, **autor** e **preço**
 - ▶ Crie os métodos **getPreco** para obter o valor do preço e o método **setPreco** para setar um novo valor ao preço.
 - ▶ Crie um código de teste





Relacionamento entre Classes



Herança

- ▶ Uma classe pode herdar a definição de outra classe
 - ▶ Permite o uso ou a extensão de métodos e atributos previamente definidos por outra classe.
 - ▶ Nova classe: **subclasse**. Original: **classe pai**, ancestral ou **superclasse**
 - ▶ Para definir uma subclasse, coloque o nome da superclasse entre parênteses depois do nome da subclasse na primeira linha da definição.
 - ▶ Python não tem a palavra 'extends' como em Java
 - ▶ Múltipla herança é suportada



Herança

```
class Veiculo:
    def andar(self):
        print("andei")
```

```
class Carro(Veiculo):
    _nrodas = 4
```

```
>>> gol = carro()
```

```
>>> gol.andar()
```

andei



Redefinindo métodos

- ▶ Você pode redefinir métodos declarados na superclasse
- ▶ O mesmo vale para o método `__init__`
- ▶ Geralmente você encontra algo assim no método `__init__` das subclasses:
 - ▶ `parentClass.__init__(self, x, y)` onde `parentClass` é o nome da classe pai.



Redefinindo métodos

```
class Veiculo:
    def andar(self):
        print("andei")

class Carro(Veiculo):
    _nrodas = 4
    def andar(self):
        print("andei de carro")

>>> gol = Carro()
>>> gol.andar()
andei de carro
```



Redefinindo métodos

```
class Veiculo:
    def andar(self):
        print("andei")
```

```
class Carro(Veiculo):
    _nrodas = 4
    def andar(self):
        Veiculo.andar(self)
```

```
>>> gol = Carro()
```

```
>>> gol.andar()
```

andei



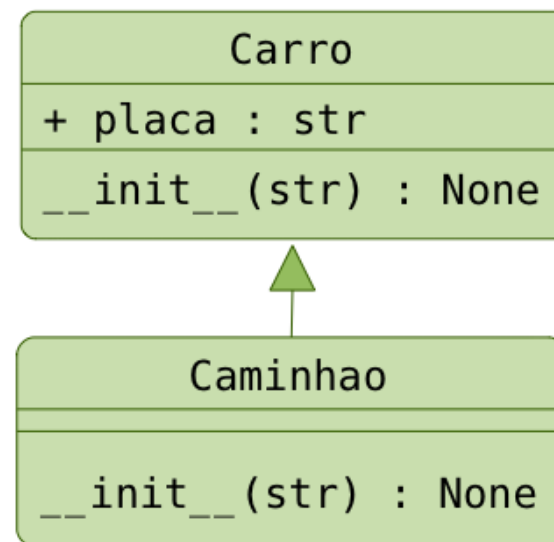
Herança

- Java

```
public class Caminhao extends Carro {  
    public Caminhao(String placa) {  
        super(placa);  
    }  
}
```

- Python

```
class Caminhao (Carro):  
    def __init__(self, placa):  
        Carro.__init__(self, placa)
```



Alguns métodos e atributos especiais nativos

Membros nativos

- ▶ As classes contêm métodos e atributos especiais que são incluídos por Python mesmo se você não os defina explicitamente.
- ▶ A maioria destes métodos são invocados automaticamente a partir de alguma ação ou evento por meio de operadores ou uso da classe.
- ▶ Alguns atributos nativos definem informações que devem ser armazenadas para todas as classes.
- ▶ Todos os membros nativos tem 2 *underscores* ao redor dos nomes: `__init__` , `__doc__`



Membros nativos

- ▶ Alguns métodos, como por exemplo `__repr__`, existem para todas as classes e você pode sempre redefini-las.
- ▶ A definição deste método especifica como tornar a instância de uma classe em uma String.
- ▶ `print f` algumas vezes chama `f.__repr__()` para chamar a representação em *string* do objeto `f`
- ▶ Se você digitar `f` e pressionar ENTER, então você também está chamando `__repr__` para informar ao *display* o que deve ser exibido ao usuário



Métodos nativos

- ▶ Você pode redefinir estes métodos também:
 - ▶ `__init__` : O construtor da classe
 - ▶ `__cmp__`: Define como `==` funciona para a classe
 - ▶ `__len__` : Define como `len(obj)` funciona
 - ▶ `__copy__` : Define como copiar uma classe
- ▶ Outros métodos nativos permitem você dar a classe o poder de usar notação `[]` como um array ou `()` como uma chamada de função.



Métodos nativos

```
class Carro:
    def __init__(self, nr):
        self.__nrodas = nr
    def __add__(self, car):
        return self.__nrodas + car.get_nr()
    def __repr__(self):
        return "Eu sou um carro de %d rodas!" % self._nrodas
    def __cmp__(self, car):
        return cmp(self.__nrodas, car.get_nr())
    def get_nr(self):
        return self.__nrodas
```

```
>>> a = Carro(4); b = Carro(6)
```

```
>>> a + b
```

```
10
```

```
>>> a > b
```

```
False
```

```
>>> print a
```

```
Eu sou um carro de 4 rodas
```



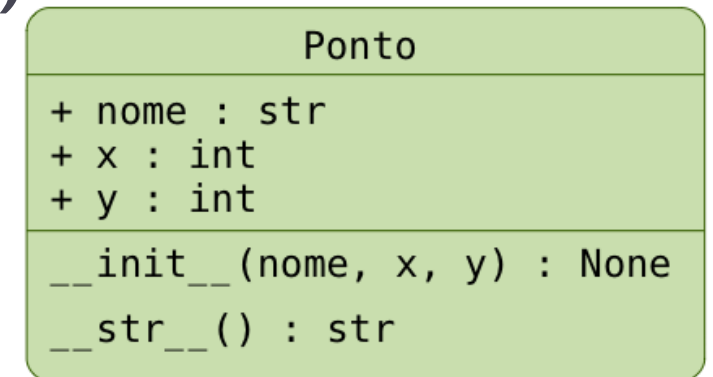
Atributos especiais

- ▶ Estes atributos existem para todas as classes.
 - ▶ `__doc__` : Armazena a documentação (String) para a classe.
 - ▶ `__class__` : Retorna a referência à classe de qualquer instância dela.
 - ▶ `__module__` : Retorna a referência ao módulo que aquela classe em particular foi definida.
- ▶ Outro método bem útil `dir(x)` retorna a lista de todos os métodos e atributos definidos pelo objeto `x`.



Exercícios

1. Crie uma classe **Ponto** conforme o diagrama ao lado. Salve o código num arquivo de nome ponto.py
 - ▶ O método `__str__` retorna uma string se alguém der um “**print objeto**”. Faça com que mostre os dados do objeto no formato: “**Nome: (x, y)**”



2. Crie outro script que importe **ponto**, e leia um arquivo contendo informações sobre vários pontos, criando um objeto **Ponto** para cada entrada lida.



Exercícios

3. Coloque cada objeto Ponto numa lista.
4. Imprima cada elemento da lista.

pontos.txt

A

100 200

B

130 150

C

500 239

OutroPonto

199 54





[Aula 7] Linguagem de Programação

Análise e Desenvolvimento de Sistemas

Orientação a Objetos em Python – Prof. Jean Zahn

jeanozahn@gmail.com