Computação II (MAB 225)

Fabio Mascarenhas - 2015.1

http://www.dcc.ufrj.br/~fabiom/pythonoo

Introdução

- No curso de Computação I vocês viram os conceitos básicos de programação em Python: expressões e comandos, funções, estruturas de dados simples, interação linear com o usuário usando o console

- Nesse curso vamos ver conceitos mais sofisticados de programação: como organizar um programa em *objetos*, como interagir com o usuário de modo não-linear, usando uma interface gráfica, e como usar Python para computação científica

- O objetivo é permitir a vocês saírem de pequenos programas exemplos para o desenvolvimento de aplicações reais, que sejam úteis na sua carreira como engenheiros

Orientação a Objetos

- Orientação a objetos é um paradigma de programação no qual estruturamos um programa como uma teia de *objetos* que se comunicam entre si através de *mensagens*

- Objetos são criados a partir de moldes que descrevem sua estrutura interna, e quais mensagens ele responde: as *classes*

- As mensagens às quais um objeto responde formam a sua *interface*

- Por fim, classes podem ser organizadas numa hierarquia em que classes derivam ou *herdam* de outras classes, promovendo o reuso de funcionalidade entre as classes

- Não se preocupem, nas próximas aulas voltaremos a cada um desses termos!

História da OO

- Sketchpad (1963)
  - Editor gráfico pioneiro, introduz o conceito de *objeto*
  - Desenhos e diagramas podiam ser agrupados e clonados: um desenho *mestre* podia ter várias *instâncias*, e mudanças no mestre eram refletidas nas instâncias
  - Instâncias são objetos, e os desenhos mestres são classes

História da OO

- Simula (1962-68)
  - Família de linguagens de programação para simulação de eventos discretos
  - Simula 67 introduz os conceitos de *classes*, *métodos virtuais*, *herança* e *subtipagem*
  - O modelo orientado a objetos é bastante natural para simulações, onde pode-se modelar cada elemento da simulação como um objeto responsável pelo seu próprio comportamento
  - Jogos também são simulações!

História da OO

- Smalltalk (1972-80)
  - Primeira linguagem OO “pura”
  - Qualquer coisa em Smalltalk é um objeto: uma *instância* de uma *classe* que responde a *mensagens* (*métodos virtuais*)
  - Inclusive coisas como as próprias classes, o contexto onde ficam armazenadas as variáveis, as partes do ambiente de programação...

História da OO

- Java (1995-?)
  - Criada originalmente para ser embutida em receptores de TV a cabo
  - Depois embutida em páginas web, como *applets*, mas ganhou maior uso como linguagem usada em sistemas web no lado do servidor, e em algumas aplicações desktop
  - Sintaxe inspirada em C, mas o modelo de execução está mais próximo de Simula e Smalltalk

Dados primitivos

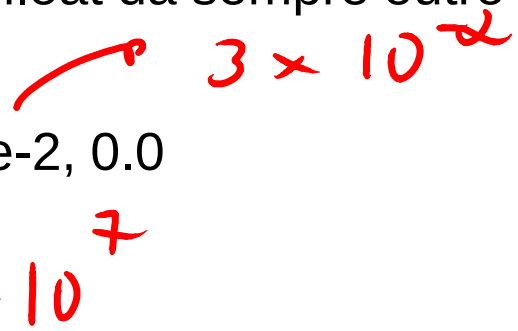
- Objetos concretos são compostos de *estado* e *comportamento*
- O estado pode ser composto por outros objetos, que por sua vez têm seu próprio estado, composto por outros objetos, que têm seu próprio estado, que...
- Alguma hora chegamos a tipos pré-definidos de Python
- Vamos relembrar os tipos que vocês viram em Computação I

Inteiros

- Sinalizados com o tipo `int`
  - Possuem sinal e 32 bits de precisão, então vão de ~-2.000.000.000 a ~2.000.000.000
  - Operações aritméticas: +, -, *, /
  - Exemplos: 2, 5, 42, 1000000, -1

“Reais”

- Sinalizados pelo tipo `float`
  - Ponto flutuante com 64 bits, o que dá 52 bits de precisão (a mantissa)
  - Raramente um número com parte decimal tem representação exata como um float
  - Também possui as quatro operações aritméticas básicas, sendo que a divisão de um inteiro e um float dá sempre outro float
  - Exemplos: 1.23, 2.56e7, 3e-2, 0.0



Booleanos

- Sinalizados pelo tipo `bool`
  - Dois valores: verdadeiro (`True`) e falso (`False`)
  - Operações lógicas: `and` (e), `or` (ou), `not` (negação)
- Todo valor pode ser tratado como um booleano e normalmente é considerado verdadeiro, exceto por `None`, zeros inteiros ou reais, e strings, tuplas, listas ou dicionários vazios

Cadeias (strings)

- Sinalizados pelo tipo `str`
- Não são exatamente tão primitivos como os anteriores, pois já são objetos que também possuem métodos
- As funções do módulo `str` são métodos das strings
- Podem ser concatenados com +, criando uma nova string
- Não se pode mudar o conteúdo de uma string → IMUTÁVEIS

Listas

- Sinalizados pelo tipo `list`
- Listas são sequências de valores quaisquer: `[]`, `[1, 2, “foo”]`, `[[3, “a”],5]`
- Listas também são objetos, e as funções do módulo `list` são métodos dos objetos lista
- Podemos ler e substituir elementos de uma lista usando colchetes:

```
>>> l = [1, 3, 6]
>>> l[1]
3
>>> l[1] = 5
>>> l
[1, 5, 6]
```

Classes

- Uma das unidades básicas da programação OO

- Declaramos uma classe com o comando `class` seguido do nome da classe e um bloco, onde definimos os atributos dessa classe:

```
class MinhaClasse:
    pass
```

- Criamos uma instância de uma classe chamando a classe como se fosse uma função:

```
obj1 = MinhaClasse()
obj2 = MinhaClasse()
```

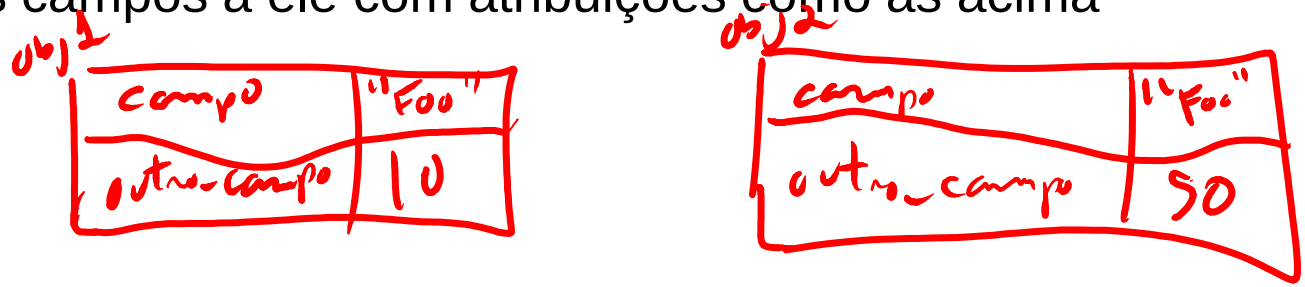
- Não podemos fazer muita coisa com instâncias de uma classe vazia; no mínimo precisamos de um *construtor*

Campos e Construtor

- Os dados que compõem um objeto ficam em *campos*
- Esses campos são declarados e inicializados dentro do *construtor* da classe
- O construtor de uma classe sempre se chama `__init__`

```
class MinhaClasse:
    def __init__(self):
        self.campo = "Foo"
        self.outro_campo = 10
```

- Dentro do construtor, o parâmetro `self` aponta para o objeto recém-criado, e adicionamos campos a ele com atribuições como as acima



Parâmetros do construtor

- Um construtor pode ter outros parâmetros além do `self`

- Esses parâmetros extras correspondem a *argumentos* passados na criação do objeto, quando chamamos a classe, e usamos eles para ajudar a inicializar o objeto:

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

p1 = Ponto(1, 4)
p2 = Ponto(10, 20)
print p1.x + p2.y  # imprime 21
```

Identidade de objetos

- Todo objeto possui uma identidade; quando criamos um objeto, a identidade dele é diferente de todos os outros objetos, mesmo objetos da mesma classe, com o conteúdo dos campos idêntico

- Podemos comparar se dois objetos são a mesma instância (têm a mesma identidade) com os operadores `is` e `is not`

- Se não fazemos nada especial, os operadores `is` e `is not` correspondem a `==` e `!=`, mas podemos redefinir `==` para comparar objetos levando em conta sua estrutura e não apenas sua identidade

Identidade de objetos, cont.

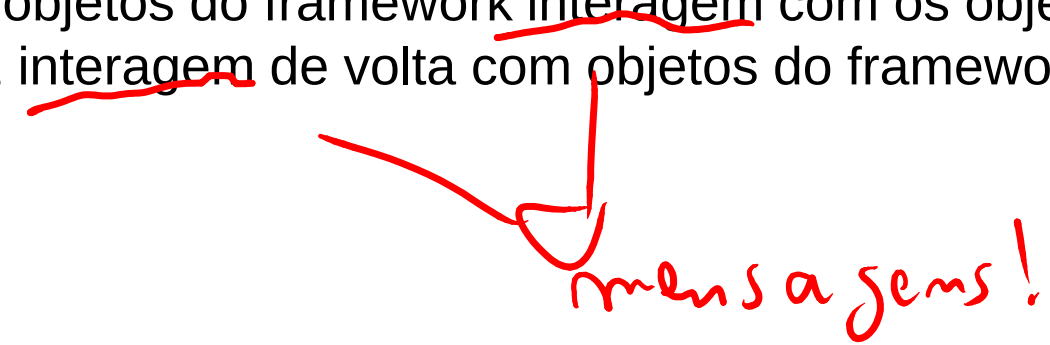
- Podemos ver um exemplo da diferença entre igualdade e identidade com listas:

```
>>> l1 = []
>>> l2 = []
>>> l1 == l2
True
>>> l1 is l2
False
>>>
```

- Colchetes são construtores para listas, então cada uso dos colchetes cria uma instância diferente de uma lista

Frameworks

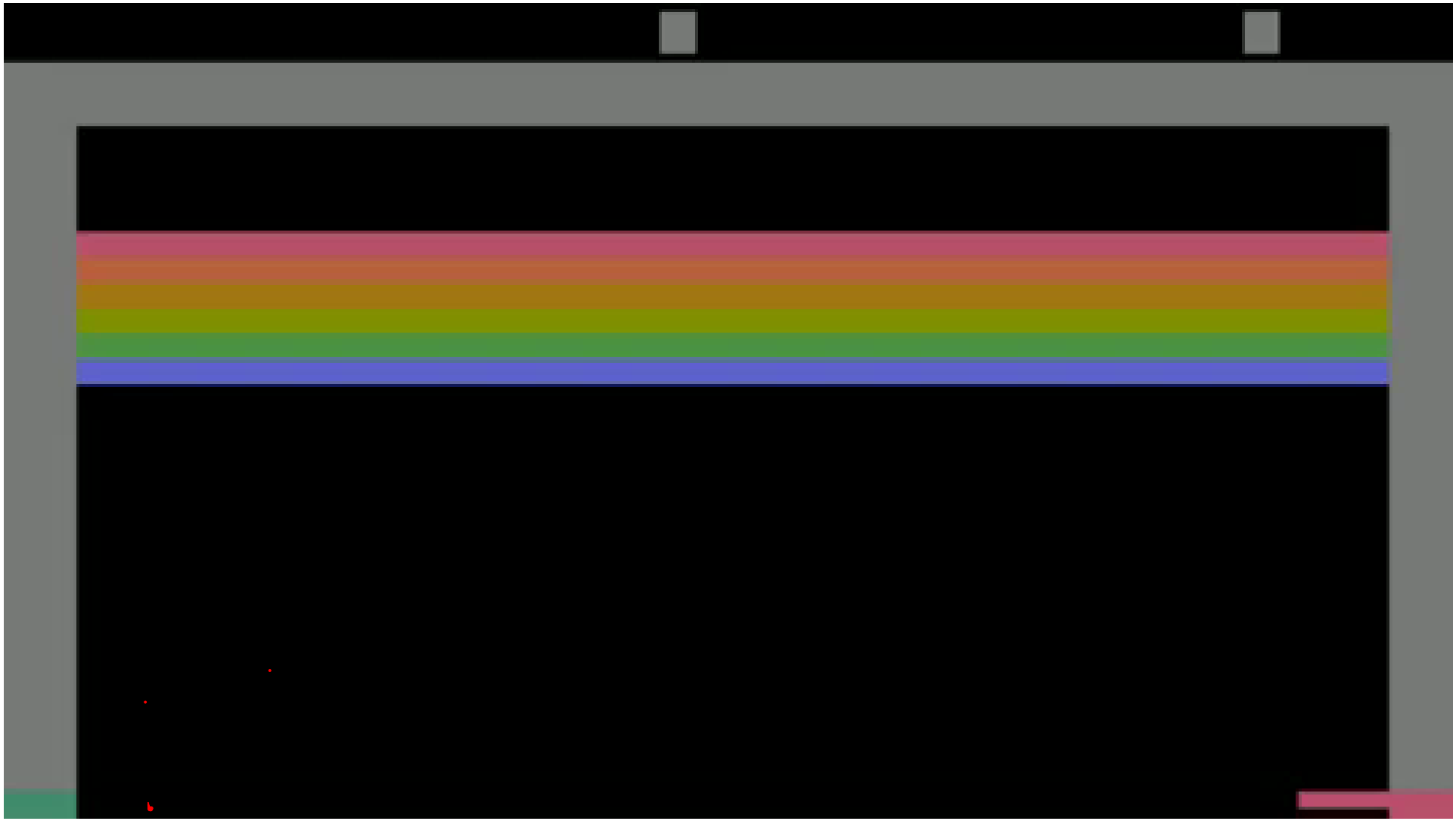
- É bastante comum que uma aplicação OO seja construída para usar um *framework* (arcabouço)
- Um framework é como uma máquina com algumas peças faltando, que serão fornecidas pela aplicação
- Essas “peças faltando” são instâncias de classes cuja estrutura é ditada pelas necessidades do framework
- Os objetos do framework interagem com os objetos da aplicação, que por sua vez interagem de volta com objetos do framework, para requisitar serviços



Um framework simples para jogos

- Vamos usar um framework bem simples para construir jogos 2D
- O framework fornece uma *tela* para desenhar figuras geométricas simples, além de texto
- A cada “tique” do relógio interno do framework, ele fornece uma tela vazia com fundo preto
- Ele também avisa a aplicação de *eventos* que acontecem: teclas pressionadas, e a própria passagem do tempo

Breakout



Componentes do Breakout

- Bola
- “Raquete”
- Tijolos
- Paredes
- Score
- Nem todos vão precisar de classes próprias para representa-los!

Bola

- Representamos a bola com uma posição e uma velocidade

- Tanto posição quanto a velocidade têm um componente horizontal e um vertical

```
class Bola:
    def __init__(self, x, y, vx, vy):
        self.x = x
        self.y = y
        self.vx = vx
        self.vy = vy
        self.raio = 5
        self.cor = (1.0, 1.0, 1.0)
```

VERM VERDE AZUL

- Tanto o raio quanto a cor da bola são fixos; a posição será do centro da bola