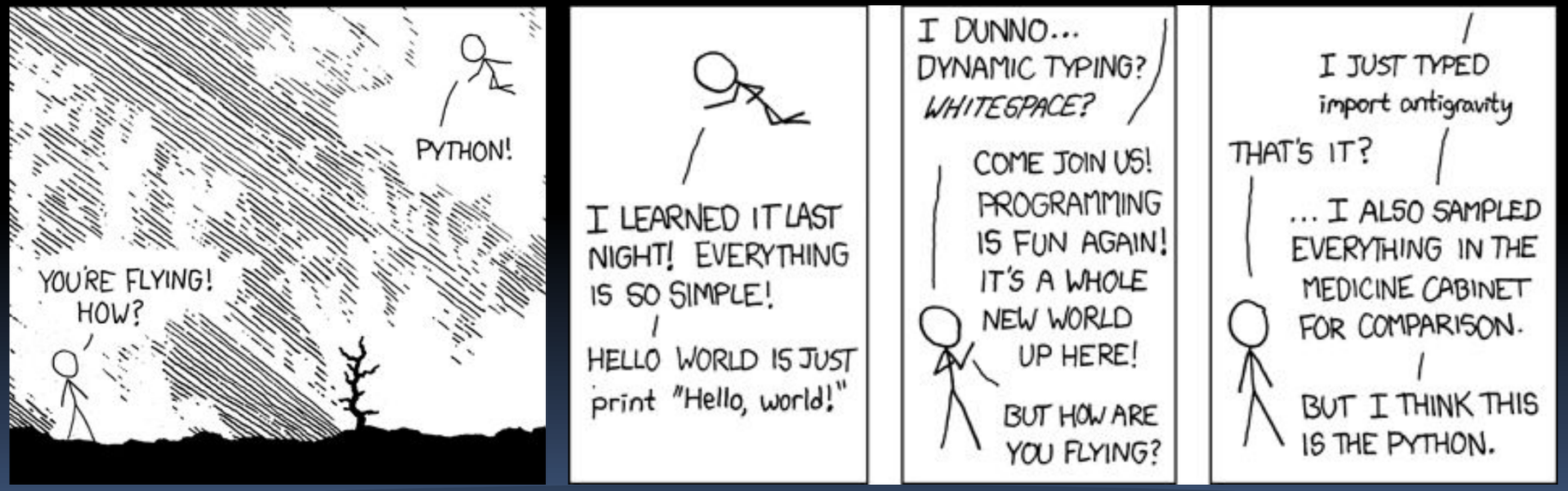
- Announcements

UC Berkeley
Teaching Professor
Dan Garcia

The Beauty and Joy of Computing

Python Object-Oriented Programming (OOP)



xkcd.com/353

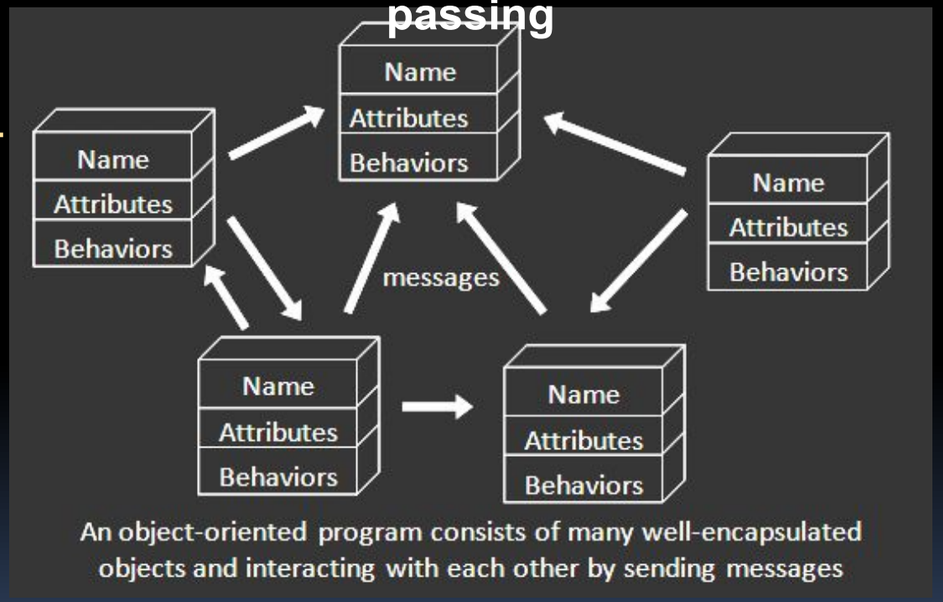
OOP Basics Snap! Demo

en.wikipedia.org/wiki/Object-oriented_programming

Review: Object-Oriented Prog. (OOP)

- One of 4 Programming Paradigms
  - Great for simulating independent elements
  - E.g., Minecraft (with people, skeletons, etc.)
- Classes are categories of things
  - "Factories" that produce objects
  - E.g., Dog, Cat, "Skeleton Spawner"..
- Objects (aka Instances) are examples of a class
  - With methods you ask of them
    - What can this object DO?
    - These are the behaviors
  - With local state, to remember
    - What does this object HAVE?
    - These are the attributes
  - E.g., Fluffy is instance of Dog
    - Behaviors: Bark, Sit, Roll over, etc.
    - Attributes: Name, Shots?, x,y,z location, etc.

Objects communicate via message passing



www3.ntu.edu.sg/home/ehchua/programming/java/images/OOP-Objects.gif

Let's now show you a Snap! demo…

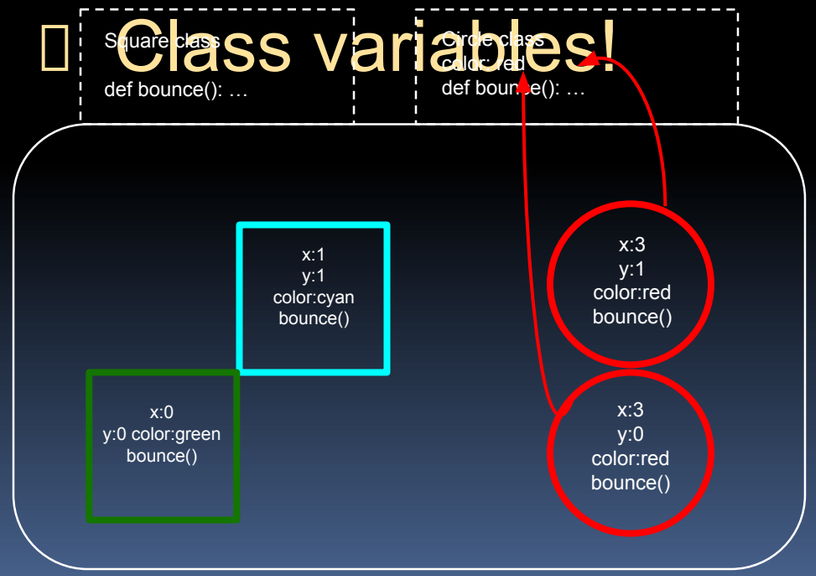
Why OOP?

Why OOP? Managing code, namespaces!

- Without OOP
  - Global lists of all objects and values
  - Tons of functions floating around, in same namespace
    - What if they conflict?
- With OOP
  - Each Class knows about different objects
  - Each Instance has data and functions within it
  - Class variables!

```
SQUARES = { 4: {"x":0,"y":0,"color":green},
            5: {"x":1,"y":1,"color":cyan}}
CIRCLES = { 1: {"x":3,"y":0,"color":red},
            2: {"x":3,"y":1,"color":red}}

def circle_bounce(): …
def square_bounce(): …
def change_all_circle_colors(newcolor): …
```



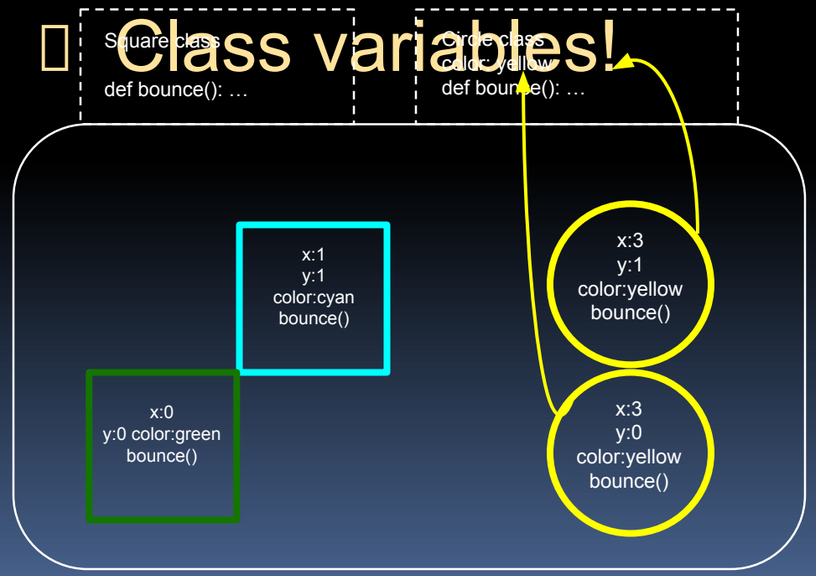
Why OOP? Managing code, namespaces!

- Without OOP
  - Global lists of all objects and values
  - Tons of functions floating around, in same namespace
    - What if they conflict?

```
SQUARES = { 4: {"x":0,"y":0,"color":green},
            5: {"x":1,"y":1,"color":cyan}}
CIRCLES = { 1: {"x":3,"y":0,"color":red},
            2: {"x":3,"y":1,"color":red}}

def circle_bounce(): …
def square_bounce(): …
def change_all_circle_colors(newcolor): …
```

- With OOP
  - Each Class knows about different objects
  - Each Instance has data and functions within it
  - Class variables!

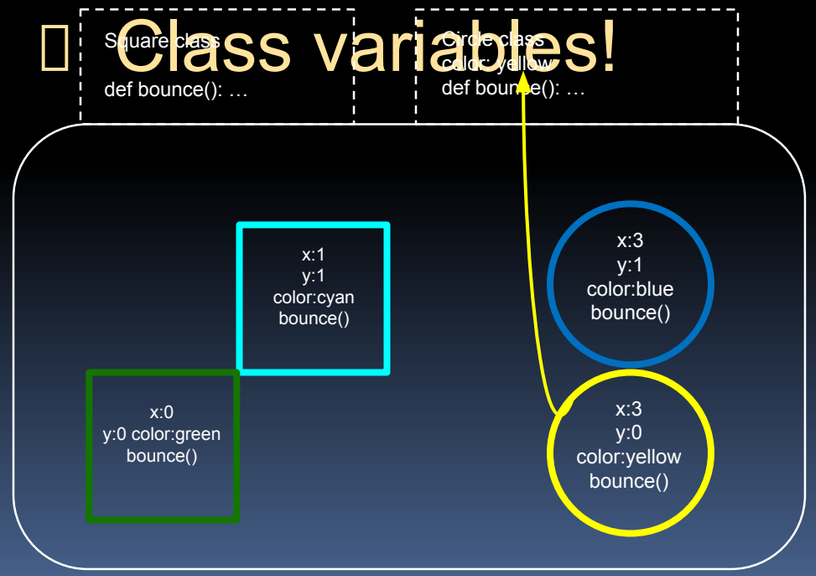


Why OOP? Managing code, namespaces!

- Without OOP
  - Global lists of all objects and values
  - Tons of functions floating around, in same namespace
    - What if they conflict?
- With OOP
  - Each Class knows about different objects
  - Each Instance has data and functions within it
  - Class variables!

```
SQUARES = { 4: {"x":0,"y":0,"color":green},
            5: {"x":1,"y":1,"color":cyan}}
CIRCLES = { 1: {"x":3,"y":0,"color":red},
            2: {"x":3,"y":1,"color":red}}

def circle_bounce(): …
def square_bounce(): …
def change_all_circle_colors(newcolor): …
```

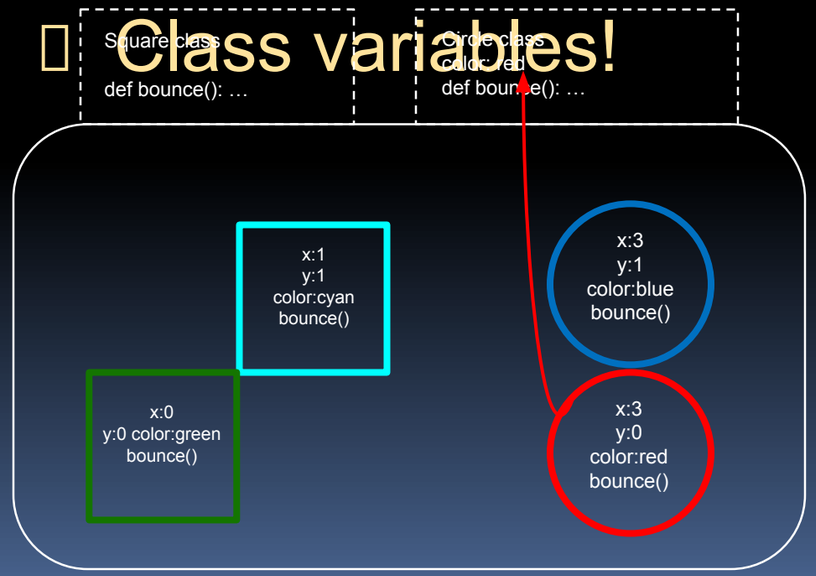


Why OOP? Managing code, namespaces!

- Without OOP
  - Global lists of all objects and values
  - Tons of functions floating around, in same namespace
    - What if they conflict?
- With OOP
  - Each Class knows about different objects
  - Each Instance has data and functions within it
  - Class variables!

```
SQUARES = { 4: {"x":0,"y":0,"color":green},
            5: {"x":1,"y":1,"color":cyan}}
CIRCLES = { 1: {"x":3,"y":0,"color":red},
            2: {"x":3,"y":1,"color":red}}

def circle_bounce(): …
def square_bounce(): …
def change_all_circle_colors(newcolor): …
```



Garcia

Classes in Python

Goal: A Sprite Simulator (ala Snap!)

- Class: Sprites
  - All sprites start out at 100% size
  - All sprites have a name & color *specific to that sprite*
  - All sprites have a “Say” method

- Instance: Alonzo sprite
  - This sprite starts out at 100% size
  - But it has a unique name (“Alonzo”), and a color (“yellow”)
  - Alonzo can say things

- Want the ability to change size of all sprites

Classes and Instances in Python

```
class Sprite:
    size = 100

    def __init__(self, name, color):
        self.name = name
        self.color = color

    def say(self, text):
        print(text)
```

Classes and Instances in Python

```
class Sprite:
    size = 100

    def __init__(self, name, color):
        self.name = name
        self.color = color

    def say(self, text):
        print(text)
```

Class Declaration

Object Constructor

Class Method

Class vs Instance Attributes and Methods

- **Class attributes and methods** are shared by all instances of the class
- **Instance attributes and methods** are unique to the particular instances
- Instance attributes and methods **of the same name** hold precedence over class attributes and methods

Classes and Instances in Python

```
class Sprite:
    size = 100

    def __init__(self, name, color):
        self.name = name
        self.color = color

    def say(self, text):
        print(text)
```

Class Attribute

Instance Attributes

Instantiating an Instance

- To create an instance of a class, you must call the constructor **(the __init__)**

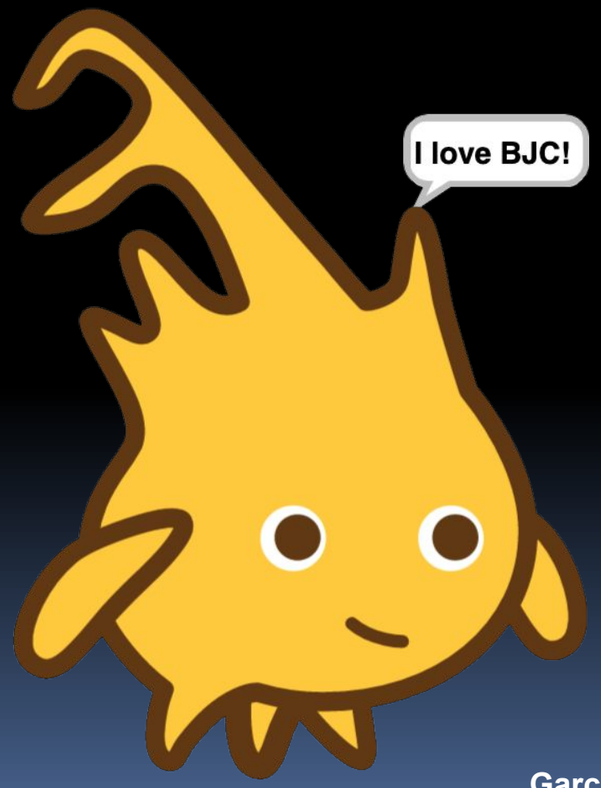
```
>>> alonzo = Sprite('Alonzo G.', 'yellow')
```

- Now our **alonzo** variable is bound to a Sprite object that has the name 'Alonzo G.' and the color 'yellow'
  - Note these are all the inputs we passed into the constructor!

Accessing Object Data

- Now that we have our alonzo object, how to get its attributes & methods?
  - Dot notation again!

```
>>> alonzo.name
'Alonzo G.'
>>> alonzo.size
100
>>> alonzo.say("I love BJC!")
I love BJC!
```



Mutating Your Objects (1/2)

- Remember that objects are mutable! You can change object attributes …
- Reassign attributes the way you would assign variables

```
>>> apple = Sprite('Another Sprite', 'green')
>>> apple.color
'green'
>>> apple.color = 'red'
>>> apple.color
'red'
```

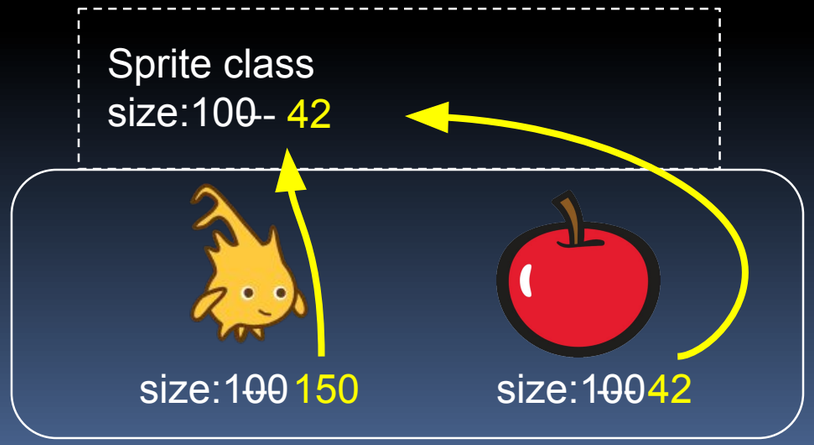
Mutating Your Objects (2/2)

- Changing an instance attribute only changes the attribute for that instance

```
>>> alonzo.size = 150
>>> apple.size
100
```

- Changing a class attribute changes it for all objects that haven't been changed

```
>>> Sprite.size = 42
>>> apple.size
42
>>> alonzo.size
150
```



What happened to self?

```
class Sprite:
    size = 100

    def __init__(self, name, color):
        self.name = name
        self.color = color

    def say(self, text):
        print(text)
```

Bound Methods

- Procedures defined in a class are bound methods
  - These automatically pass the object in as the first input
  - So the ‘self’ input doesn’t actually go away!

- When you call this:

```
>>> alonzo.say("I love BJC!")
```

…you’re actually calling this:

```
>>> Sprite.say(alonzo, “I love BJC!”)
```

Vector Class Demo

When poll is active, respond at **pollev.com/ddg**

Text **DDG** to **22333** once to join

L19 "I understood the vector OOP demo"

- Strongly agree
- Agree
- Neutral
- Disagree
- Strongly disagree