# DEFINING CLASSES IN PYTHON

Catapult—Rose Hulman Institute of Technology

# We've actually been using Objects

```
WIDTH = 400
HEIGHT = 50
REPEAT_COUNT = 20
PAUSE_LENGTH = 0.25
win = GraphWin('Giants Win!', WIDTH, HEIGHT)
p = Point(WIDTH/2, HEIGHT/2)
t = Text(p, 'NY Giants—2008 Super Bowl Champs!')
t.setStyle('bold')
t.draw(win)
nextColorIsRed = True
t.setFill('blue')
for i in range(REPEAT_COUNT):
    sleep(PAUSE_LENGTH)
    if nextColorIsRed:
        t.setFill('red')
    else:
        t.setFill('blue')
    nextColorIsRed = not nextColorIsRed
win.close()
```

Consider this graphics program…

It uses Objects!

# Object Terminology

- Objects are "*active data types*"
  - They **know stuff** — *instance variables*
  - They **can do stuff** — *methods*
- Objects are *instances* of some *class*
- Objects created by calling *constructors*

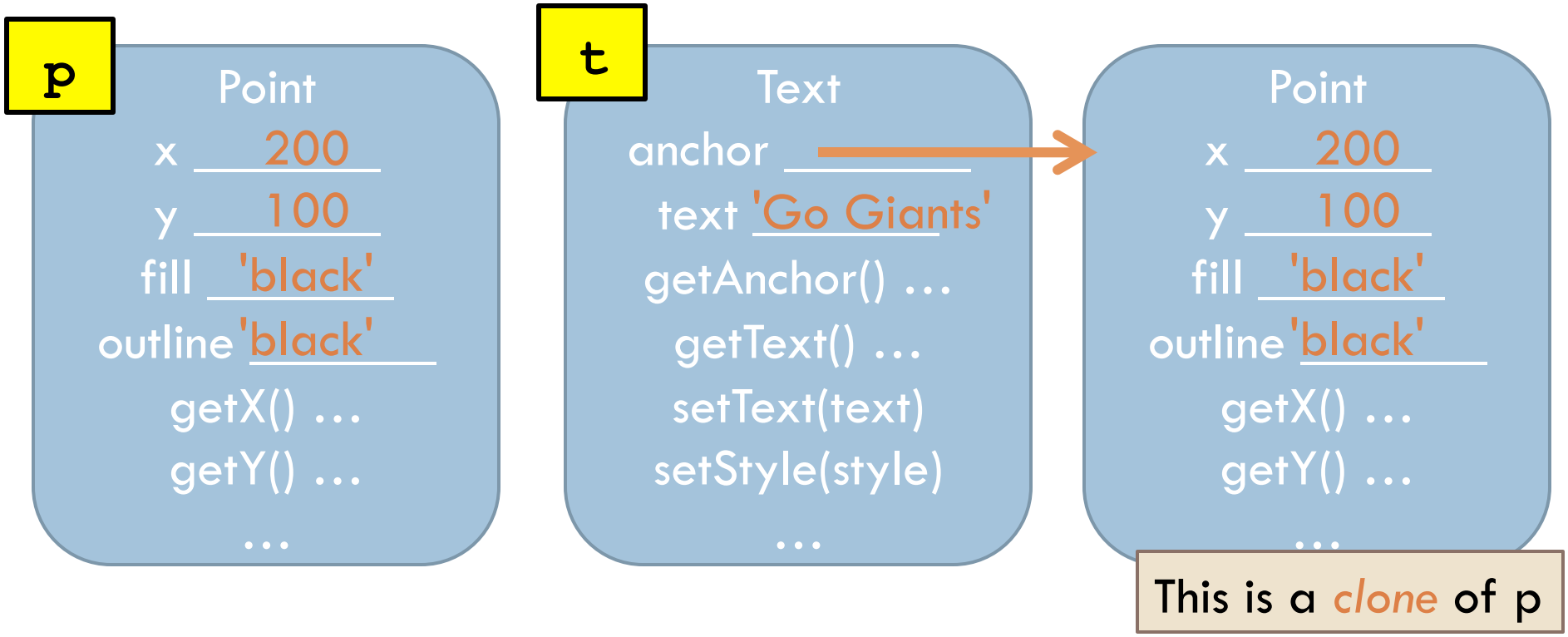
# Key Concept!

- A class is like an "object factory"
  - Calling the constructor tells the classes to make a new object
  - Parameters to constructor are like "factory options", used to set instance variables
- Or think of class like a "rubber stamp"
  - Calling the constructor stamps out a new object shaped like the class
  - Parameters to constructor "fill in the blanks". That is, they are used to set instance variables.

# Example

- Consider:

```
p = Point(200, 100)
t = Text(p, 'Go Giants!')
```



# Creating Custom Objects: Defining Your Own Classes

- Custom objects:
  - Hide complexity
  - Provide another way to break problems into pieces
  - Make it easier to pass information around

# Code to Define a Class

Declares a class named Card

```
class Card:
    """This class represents a card from a standard deck."""
```

***docstring*** describes class, used by **help()** function

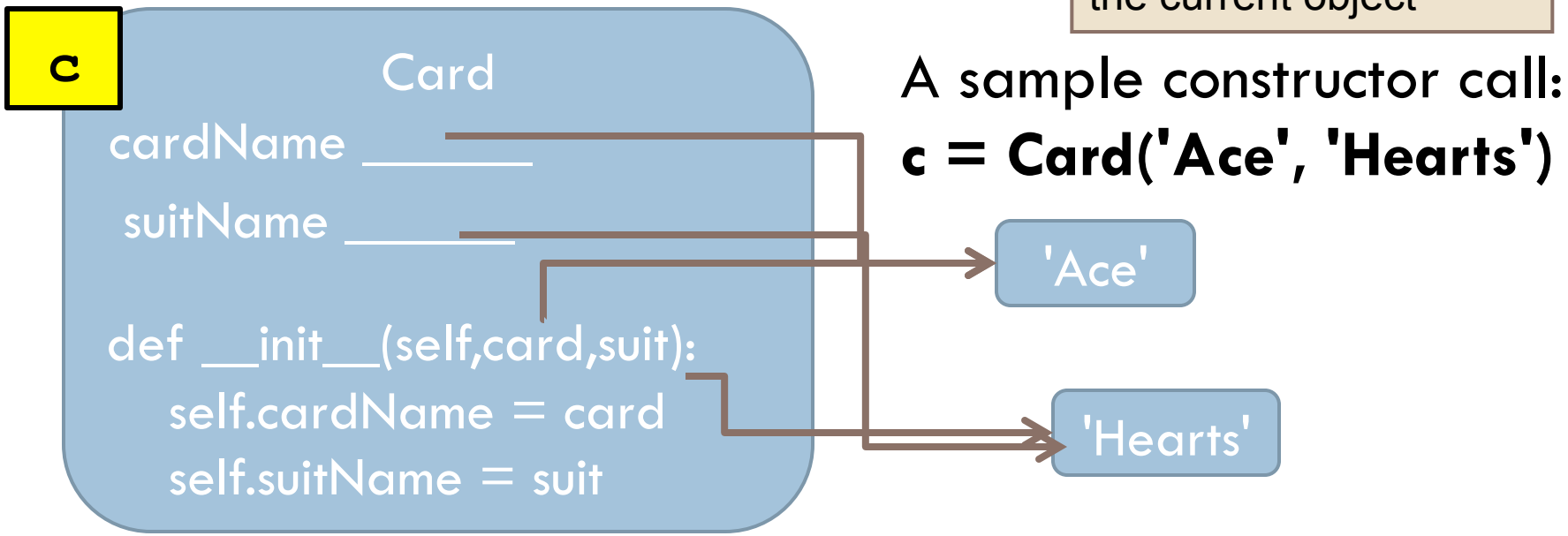
# Code to Define a Class

Special name, `__init__` declares a ***constructor***

```
class Card:
    """This class represents a card from a standard deck."""
    def __init__(self, card, suit):
        self.cardName = card
        self.suitName = suit
```

Create instance variables just by assigning to them

Special ***self parameter*** is the first formal parameter of each method in a class. **`self`** always refers to the current object




A sample constructor call:

**c = Card('Ace', 'Hearts')**

# Code to Define a Class

```
class Card:
    """This class represents a card from a standard deck."""
    def __init__(self, card, suit):
        self.cardName = card
        self.suitName = suit

    def getValue(self):
        """Returns the value of this card in BlackJack.
        Aces always count as one, so hands need to adjust
        to count aces as 11."""
        pos = cardNames.index(self.cardName)
        if pos < 10:
            return pos + 1
        return 10
```

***self parameter*** again, no other formal parameters

***docstring*** for method

use `self.<varName>` to ***read instance variable***

A sample method call:

**c.getValue()**

# Code to Define a Class

```
class Card:
    """This class represents a card from a standard deck."""
    def __init__(self, card, suit):
        self.cardName = card
        self.suitName = suit

    def getValue(self):
        """Returns the value of this card in BlackJack.
        Aces always count as one, so hands need to adjust
        to count aces as 11."""
        pos = cardNames.index(self.cardName)
        if pos < 10:
            return pos + 1
        return 10

    def __str__(self):
        return self.cardName + " of " + self.suitName
```

Sample uses of __str__ method:

```
print (c)
msg = "Card is " + str(c)
```

***Special* `__str__` *method*** returns a string representation of an object

# Stepping Through Some Code

Sample use:

```
card = Card('7','Clubs')
print (card.getValue())
print (card)
```

```
class Card:
    """This class represents a card
    def __init__(self, card, suit):
        self.cardName = card
        self.suitName = suit

    def getValue(self):
        """Returns the value of this card in BlackJack.
        Aces always count as one, so hands need to adjust
        to count aces as 11."""
        pos = cardNames.index(self.cardName)
        if pos < 10:
            return pos + 1
        return 10

    def __str__(self):
        return self.cardName + " of " + self.suitName
```

# Key Ideas

- *Constructor*:
  - Defined with special name `__init__`
  - Called like `ClassName()`
- *Instance variables (a.k.a fields)*:
  - Created when we assign to them
  - Live as long as the object lives
- `self` formal parameter:
  - Implicitly get the value *before the dot* in the call
  - Allows method of an object to "talk about itself"

# Let's look at an example!

- An employee class and some uses of it
  - http://www.tutorialspoint.com/python/python_classes_objects.htm