CSCI 1300
Intro to Computing

Gabe Johnson

Lecture 14 Feb 18, 2013

Classes and Objects in Python

Lecture Goals

1. Correcting some confusions
2. Keep on going with Objects

Upcoming Homework Assignment

HW #4 **Due: Friday, Feb 22**

While, Boolean, and Objects

This week we're adding in the finishing aspects of Python: while loops, boolean arithmetic, and very basic objects. Check the GitHub repository for some documentation. The questions are softballs, meant to give you some time to play with the more advanced stuff we're doing in class.

Confusions Part 1

‘name’ is not the same as ‘Name’.

Python (and any other respectable language I know of) makes a distinction between strings that have different capitalization.

Confusions Part 2

Everything that a function does must be included somewhere in the function definition. If you have code above the line that says ‘def’, or if you have code that is not in the function’s code suite, it isn’t part of the function.

Confusions, Part 2

This is wrong for a few reasons:

```
my_data = [10, 20, 30]
def process_data(input_list):
    print my_data

print input_list
```

First: ‘my_data’ is defined above the function.

Second: ‘input_list’ is only valid inside the function definition but we’re using it on the last line.

Confusions, Part 3

Every function you write for homework should have a return statement. *The return statement is payday.* That's what the grading script uses to determine what your function did. (In fact, that's what *any* code that uses your functions will use.)

The grading system (and any code that uses your functions) will completely ignore your *print* statements. It doesn't even know they're there.

Confusions, Part 3

```
# return the age the student will be a year
# from now.
def get_next_years_age(student):
    current_age = student[‘age’]
    next_age = current_age + 1
    print “The student will be “ + next_age
```

First: this function has the right idea, but because it isn’t returning anything, we never get the answer.

Second: that *print* statement is only helpful to the programmer to debug their code.

Today: Object Behavior

Earlier we made Phone objects. But all we did with them is store some silly information. Unlike dictionaries, we can use classes and objects to define behavior. Here’s a phone class that rings:

```
class Phone:
    def ring(self):
        print “Rrrrrriiiinnnnnngggggg!!!”

p1 = Phone()
p1.ring()
```

Trick out with Ringtones

```
class Phone:
    ring_tone = “Brrrring briingggg”
    def ring(self):
        print self.ring_tone

p1 = Phone()
p1.ring()
p1.ring_tone = “Woop woop woop”
p1.ring()
```

Object State

Objects contain state variables (just like the name and age and major). We can access these variables from inside an object's functions if we include the marker parameter 'self'.

```
class Phone:
    ring_tone = “wowowowowowooooo”
    # ‘self’ gives us access to ring_tone
    def ring(self):
        print self.ring_tone
```

Complex(ish) Phone

```
class Phone:
    model = "Unknown Phone"
    ring_tone = "Brrrrrrrring..."
    weight = 8
    # dictionary of names and phone numbers
    contacts = { }
    # owner is initially None.
    owner = None
```

... continued...

```
def ring(self):
    print ring_tone

# name is a person's name (string)
# phone_number is an integer
def add_contact(self, name, phone_number):
    if self.contacts.has_key(name):
        print "Updating number for '" + name + "'"
    self.contacts[name] = phone_number
    print "+Num: " + name + " = " + str(phone_number)

def has_owner(self):
    return owner is not None
```

And a Person Object

```
class Person:
    first_name = "??"
    last_name = "??"

    def __init__(self, first, last):
        self.first_name = first
        self.last_name = last

    def __str__(self):
        return self.first_name + " " + self.last_name
```

What are those funny __init__ and __str__ functions?

Init function

Inside a class definition, a function that looks like this:

```
def __init__(self, first, last):
    self.first_name = first
    self.last_name = last
```

... is called a *constructor*. This is called every time we make a new instance using *two parameters* called first and last. So we can make a person like this:

bob = Person(“Bob”, “Jones”)

str function

Inside a class definition, a function that looks like this:

```
def __str__(self):
    return self.first_name + " " + self.last_name
```

... is called a *to string* function. This is called every time we explicitly cast the object to a string with the `str()` function, or when it is implicitly cast, like when we just print the bare object.

```
bob = Person(“Bob”, “Jones”)
print “The object is: “ + str(bob)
```