April 6th: Class 9 Object Oriented Programming

Object Oriented Programming

: A common concept in any modern programming language; Approach that grew out of a need to handle the increasing complexity of programming.
: A point of view that a program is a set of objects, where each object can interact with other program objects to accomplish the programmer's goal.

Each object **(noun)** will

- have some number of attributes that are stored within the object. **(adjectives)**
- respond to some methods (**verbs**) that are particular to that kind (**class**) of object (e.g., move forward, print)

A. Instance v.s. Class

- **class** is a template for making a new object
    - built-in classes we've been using are `int, str, bool, list, float`, etc.
- A specific bject made by a class template is called an **instance** of that class
    - The literal `3` is an instance of the `int` class, `"Hello"` is an instance of the `str` class

B. Defining class

```
class ClassName():
    def anymethods you need
```

C. Defining methods within class

1) `__init__(self, any other parameters…)`
    - Acts as constructor, or initializer, when class is invoked during object creation
    - `self` is always written as a parameter in the method definition so that it can be referenced to initialize instance variables for that particular instance
    - However, when the constructor is actually invoked, self is not specified as an argument; only the arguments after that are specified in the invocation
    - E.g. Compare constructor definition for `Card` in `Card.py` and how we call `x = Card(…)` constructor in `Deck.py`

2) `__str__(self)`
    - function that specifies what string should be printed when the print function is called on the instance
    - Again, `self` is a parameter only written in the method definition
    - The method is called whenever you have commands like `print(objectName)`

3) Any additional user created methods (note these do not need the underscores like above)

** Let's see the Card class as an example.

D. built-in `isinstance(object, class)` function

- the `isinstance` function can be called on any instance of any class
- It returns a true if the object is an instance of the class specified as the explicit argument
- Returns a false if it is not

E.g. Summing up everything so far

```
class Student():
    def __init__(self, first='', last='', id=0):

        self.firstname_str = first
        self.lastname_str = last
        self.id_int = id

    def update(self, first='', last='', id=0):
        if first:
            self.firstname_str = first
        if last:
            self.lastname_str = last
        if id:
            self.id_int = id

    def __str__(self):
        # print "In str method"
        return "{} {}, ID:{}".\
        format(self.firstname_str, self.lastname_str, self.id_int)
```

In separate file called `StudentTester.py`:

```
s1 = Student();  #blank constructor (everything will be default)
s1.update('Kelly','Ryu',2362)
print s1

s2 = Student('Kelly','Ryu',2362) #constructor initializing instance var's
print s2

s3 = Student('Kelly') #only specified first keyword argument
print s3

if (isinstance(s1,Student)):
    print str(s1)+" is a student"
```

```
Result:
Kelly Ryu, ID:2362   #s1
Kelly Ryu, ID:2362   #s2
Kelly , ID:0         #s3 (last name is an empty string, ID is default 0)
Kelly Ryu, ID:2362 is a student #if statement was true
```

E. When writing tester: Reminder on how to call on another file
- Make sure current file and class file is in the same directory
- In current file, first thing to do is write: `from` *`classFileName`* `import` *`ClassName`*
- So it's easiest for you even you save the class file as the same name as the class
- No quotations around anything in the from, import statement above