PIC 16, Winter 2018 – Preparation 3W

Assigned 1/22/2018. To be completed by lecture 1/24/2018.

Intended Learning Outcomes

By the end of this preparatory assignment, students should be able to:

- convert numerics and objects to strings;
- define a class, including class variables, instance variables, methods, and a custom constructor;
- instantiate objects, manipulate class and instance variables, and call object methods; and
- write "magic methods" to enable built-in operators, functions, and constructs to work with your classes.

Tasks

- ☐ Skim 7.1. The `str` and `repr` functions are important, and you should know where to go if you ever need fancy text output, but personally, I never remember how to format output. I look it up when I need it. I'd much rather spend the time we have on GUIs (Graphical User Interfaces) than command line output!
- ☐ If you're rusty on object oriented programming, you can take a look at this playlist I made for my 10A class to explain classes, objects, instance variables, etc…
- ☐ I think Chapter 9 of the Python tutorial has a lot more information than we need, and because of that it's too complex. Instead, read
  http://www.tutorialspoint.com/python/python_classes_objects.htm
  up until the part about Built-in Class Attributes.
- ☐ Note some substantial differences from C++/Java:
  - A "class variable" is like a static field / member variable in Java/C++
  - A "method" is like a member function in C++. It's the same concept as a method in Java.
  - The concept of "instance variable" is the same in C++/Java, but note that they are declared *within a method* (member function)! You do not list all your instance variables outside the methods like you do in C++/Java; you just initialize them inside the methods.
  - The `self` variable is similar to the `this` pointer/reference in C++/Java, but you have to include it as the first parameter in the definition of every method. Also, while `this` was not always necessary for referring to instance variables in C++/Java, in Python, you *always* have to use `self` in order to refer to an instance variable.
  - You will find different opinions online as to whether the `__init__` method is equivalent to a constructor in C++/Java[1]; the better name is probably "initializer". While `__init__` is called automatically when an object is created, it is possible to invoke it manually[2]. It also has exactly the same syntax as a regular method and it is not named the same as the class like in C++/Java.
  - Even after you have created an instance of a class (an object), you can add new "attributes" (instance variables) to it. They can even be used inside methods, provided that you initialize them before invoking the method. For instance, the following works:

```
class MyClass:
    def my_fun(self):
```

[1] There is also a different method called `__new__` (which we will not use) that is executed *before* `__init__`; this is probably more akin to a constructor from C++/Java.

[2] In fact, we'll need to invoke a superclass initializer manually when we start working with GUIs.

```
        print self.y
x = MyClass()
x.y = 10
x.my_fun()
```

I don't recommend that you do this. It's just interesting that you can.

- ☐ Practice working with classes and objects, starting with `Preparation 2F.py`. As highlighted above, not all behavior will be familiar from C++ or Java. Pay attention – it lulls you into a false sense of security before revealing some unusual behavior :)
- ☐ Resume http://www.tutorialspoint.com/python/python_classes_objects.htm (starting with Built-in Class Attributes) and continue until the end. The parts we're most interested in here are the sections "Base Overloading Methods" and "Overloading Operators", but we'll also need to use inheritance when we get to GUIs.
- ☐ Some additional differences from C++/Java:
  - Garbage collection works like in Java: when there are no more references to an object, it becomes eligible for (automatic) garbage collection; you do not have to de-allocate the space manually. Every reference is like smart pointer in C++. For those of you who only know C++ but haven't seen smart pointers, yes, this is new and very convenient!
  - Python supports multiple inheritance and operator overloading, like C++, but unlike Java.
  - Support for "private" variables that cannot be accessed from outside class methods is limited, as you read in the "Data Hiding" section. We won't even use this technique.[3]
- ☐ Skim http://minhhh.github.io/posts/a-guide-to-pythons-magic-methods to get an idea for all the magic methods you can use. We will practice overloading these in class.
- ☐ (Optional) You might wish to review all of Python Tutorial 9.4 and 9.5 to clarify some of the behavior you observe in `Preparation 2F.py`.
- ☐ (Optional) Skim 9.6 for more information about private fields if you're a fan of encapsulation.
- ☐ Skim 9.7 in case you ever need a struct. This also highlights that instance variables can be created (and destroyed) on the fly; they don't need to be included in the class definition, and not all objects of the class have to have the same instance variables. Don't worry about "Instance method objects…".

[3] Some interesting thoughts on "encapsulation" and Python: http://stupidpythonideas.blogspot.com/2014/01/python-doesnt-have-encapsulation.html