CSC 171 - Introduction to Computer Programming

Lecture #11 – A Brief Introduction to Objects

What is a Class?

- If you have done anything in computer science before, you likely will have heard the term object oriented programming (OO P)
- What is OOP, and why should I care?

What is OOP?

- The short answer is that object oriented programming is a way to think about "objects" in a program (such as variables, functions, etc)
- A program becomes less a list of instruction and more a set of objects and how they interact

Our First Class

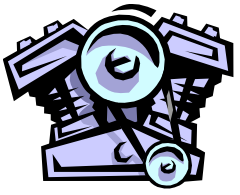
```
class Student(object):
    # Simple Student class

    # Initializes the object
    def __init__(self, first='', last='',id=0):
        self.first_name_str = first
        self_last_name_str = last
        self.id_int = id

    # String representation, eg, for printing
    def __str__(self):
        return "{} {}, ID:{}".format\
               (self.first_name_str,\
               self.last_name_str, self.id_int)
```

Objects Respond to “Messages”

- As a set of interacting objects, each object responds to “messages” sent to it
- The interaction of objects via messages makes a high level description of what the program is doing.



Everything In Python Is An Object

- In case you hadn't noticed, everything in Python is an object.
- Thus Python embraces OOP at a fundamental level.

Type Versus Class

There is a strong similarity between a type and a Python class

- Seen many types already: **`list`**, **`dict`**, **`str`**, …
- Suitable for representing different data
- Respond to different messages regarding the manipulation of that data

OOP Helps For Software Engineering

- **Software engineering** (SE) is the discipline of managing code to ensure its long-term use
- Remember: SE via refactoring
- Refactoring:
  - Takes existing code and modifies it
  - Makes the overall code simpler, easier to understand
  - Doesn't change the functionality, only the form!

More Refactoring

- Hiding the details of what the message entails means that changes can be made to the object and the flow of messages (and their results) can stay the same
- Thus the implementation of the message can change but its intended effect stay the same.
- This is ***encapsulation***

OOP Principles

- **Encapsulation**: hiding design details to make the program clearer and more easily modified later
- **Modularity**: the ability to make objects stand alone so they can be reused (our modules). Like the math module
- **Inheritance**: create a new object by inheriting (like father to son) many object characteristics while creating or over-riding for this object
- **Polymorphism**: (hard) Allow one message to be sent to any object and have it respond appropriately based on the type of object it is.

Class Versus Instance

- One of the harder things to get is what a class is and what an instance of a class is.
- The analogy of the cookie cutter and a cookie.



Template Versus Exemplar

- The cutter is a template for stamping out cookies, the cookie is what is made each time the cutter is used
- One template can be used to make an infinite number of cookies, each one just like the other.
- No one confuses a cookie for a cookie cutter, do they?

Same in OOP

- You define a class as a way to generate new instances of that class.
- Both the instances and the classes are themselves objects
- the structure of an instance starts out the same, as dictated by the class.
- The instances respond to the messages defined as part of the class.

Why A Class

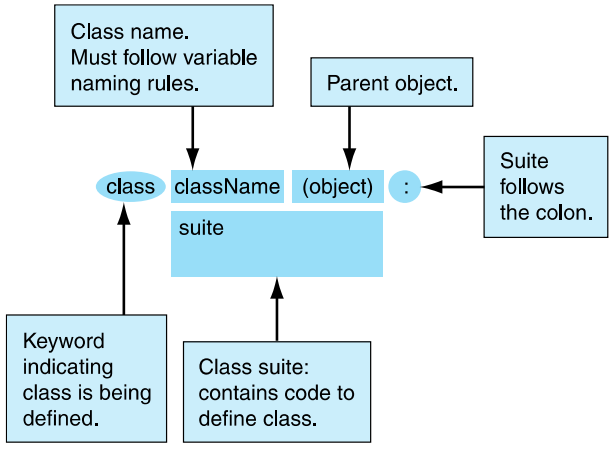
- We make classes because we need more complicated, user-defined data types to construct instances we can use.
- Each class has potentially two aspects:
  - The data (types, number, names) that each instance might contain
  - The messages that each instance can respond to.

Standard Class Names

The standard way to name a class in Python is called ***CapWords***:

- Each word of a class begins with a Capital letter
- No underlines
- Sometimes called ***CamelCase***
- Makes recognizing a class easier

The Basic Format of a Class Definition



`dir()` Function

The `dir()` function lists all the attributes of a class

- You can think of these as keys in a dictionary stored in the class.

`pass` Keyword

Remember, the `pass` keyword is used to signify that you have **intentionally** left some part of a definition (of a function, of a class) undefined

- By making the suite of a class undefined, we get only those things that Python defines for us automatically

Our First Class

```
>>> class MyClass (object):
        pass

>>> dir(MyClass)
'__class__', '__delattr__', '__dict__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__gt__', '__hash__',
'__init__', '__le__', '__lt__', '__module__', '__ne__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
'__sizeof__', '__str__', '__subclasshook__', '__weakref__']

>>> my_instance = MyClass()
>>> dir(my_instance)
['__class__', '__delattr__', '__dict__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__gt__', '__hash__',
'__init__', '__le__', '__lt__', '__module__', '__ne__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
'__sizeof__', '__str__', '__subclasshook__', '__weakref__']

>>> type(my_instance)
<class '__main__.MyClass'>
```

Constructor

- When a class is defined, a function is made *with the* ***same name as the class***
- This function is called the ***constructor***. By calling it, you can create an instance of the class
- We can affect this creation (more later), but by default Python can make an instance.

Dot Reference

- We can refer to the attributes of an object by doing a dot reference, of the form: *`object.attribute`*
- The attribute can be a variable or a function
- It is part of the object, either directly or by that object being part of a class

Examples

```
print(my_instance.my_val)
```

print a variable associated with the object **`my_instance`**

```
my_instance.my_method()
```

- Call a method associated with the object **`my_instance`**
- Variable versus method, you can tell by the parenthesis at the end of the reference

How to Make an Object-local Value

- once an object is made, the data is made the same way as in any other Python situation, by assignment
- Any object can thus be augmented by adding a variable

```
my_instance.attribute = 'hello'
```

New Attribute Shown In Dir

dir(my_instance)

- ['__class__', '__delattr__', '__dict__', '__doc__', '__format__', '__getattribute__', '__hash__', '__init__', '__module__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', attribute]