EdX 6.00x Notes

Lecture 11:

- Objects
  - Python supports many types of data
    - ints: 1234
    - floats: 3.14
    - strings: “Hello”
    - lists: [1,2,3,4]
    - dictionaries: {“CA”, “California”, “MA”, Massachusetts”}
  - Each of the above is an **object**
  - Objects have:
    - A type (a particular object is said to be an **instance** of a type)
    - An internal data representation (primitive or composite)
    - A set of procedures for interaction with the object
- Object-oriented programming (OOP)
  - Everything is an **object** and has a **type**
  - Objects are a data abstraction that encapsulate
    - Internal representation
    - **Interface** for interacting with object
      - Defines behaviors, hides implementation
  - One can
    - Create new instances of objects (explicitly or using literal)
    - Destroy objects
      - Explicitly using del or just “forget” about them
      - Python system will reclaim destroyed or inaccessible objects
- Note:
  - Python has garbage collection.
  - Python does not have “data hiding”
    - Data Hiding: prevents access to private attributes
- Advantages of OOP
  - Divide-and-conquer development
    - Implement and test behavior of each class separately
    - Increased modularity reduces complexity
  - Classes make it easy to reuse code
    - Many Python modules define new classes
    - Each class has a separate environment (no collision on function names)
    - Inheritance allows subclasses to redefine or extend a selected subset of a superclass’ behavior

- Defining new types
    - In Python, the class statement is used to define a new type
        - Example: class Coordinate(object)
    - As with def, indentation used to indicate which statements are part of the class definition
    - Clases can inherit attributes from other classes, in this case Coordinate inherits from the object class. Coordinate is said to be a **subclass** of object, object is a **superclass** of Coordinate. One can override an inherited attribute with a new definition in the class statement
- Creating an instance
    - Usually when creating an instance of a type, we will want to provide some initial values for the internal data. To do this, define an __init__ method:
        - Method: Another name for a procedural attribute, or a procedure that "belongs" to a class
    - The "." operator is used to access an attribute of an object. So the __init__ method above is defining two attributes for the new Coordinate object: x and y
    - Data attributes of an instance are often called **instance variables**
- An environment view of classes
    - Class definition creates a binding of class name in global environment to a new frame or environment
    - That frame contains any attribute bindings, eithers variable or local procedures
    - That frame also knows the parent environment from which it can inherit
    - Using the Coordinate class as an example we can access parts of a class using Coordinate.__init__
    - Python interprets this by finding the binding for the first expression (which is a frame), and then using the standard rules to lookup the value for the next part of the expression in that frame
    - Suppose a class is invoked
    - A new frame is created (this is the instance)
    - The __init__ method is then called, with self bound to this object, plus any other arguments
    - Evaluating the body of __init__ creates bindings in the frame of the instance
- Print representation of an object
    - Left to its own devices, Python uses a unique but uninformative print presentation for an objects
    - One can define a __str__ method for a class, which Python will call when it needs a string to print. This method will be called with the object as the first argument and should return a str.
        - This is overriding the default string method.
- Type of an Object
    - We can ask for the type of an object
        - Print type(object)
    - Use isinstance() to check if an object is an instance of a type

- Adding other methods
    - Can add our own methods, not just change built-in ones
- Example: a set of integers
    - Create a new type to represent a set (or collection) of integers
        - Initially the set is empty
        - A particular integer appears only once in a set
            - This constraint, called a **representational invariant**, is enforced by the code in the methods.
        - Internal data representation
            - Use a list to remember the elements of a set
        - Interface
            - insert(e) – insert integer e into set if not there
            - member(e) – return True if integer e is in set, False else
            - remove(e) – remove integer e from set, error if present