OBJECT ORIENTED PROGRAMMING

Professor Andrew Chapkowski

Objects

- Python support multiple data types

3.14 1000 “Hello World” [1,2,3,4] {‘a’:’b’} (‘a’, 1,5.2)

- Every **OBJECT** has:
  - A **type**
  - An internal **data representation**
  - A set of procedures for **interaction** with the object
- An object is an **instance** of a type

  “Johns Hopkins” is an instance of a string

  3.14 is an instance of a float

Object Oriented Programming

- **EVERYTHING IN PYTHON IS AN OBJECT**
- can create new objects of some type
- Can manipulate objects
- Can destroy objects
  - **del** explicitly destroys the object
  - **Python** reclaims destroyed objects using the “garbage collection” module

What are Object?

- They are a **data abstraction** that can:

1. Has an internal representation
2. Has an interface to which one interacts with the objects (functions/methods, etc…)

Example

Take a list with the following: [1,2,3,4]

- How are lists internally represented?
  - They are linked list of cells

  mylist = 1 -> 2 -> 3 -> 4->….. -> N
- Manipulation occurs through various method:
  - mylist[i], mylist[1:3], …
  - len(), min(), max(), del mylist[i]
  - Insert(), pop(), remove(), etc…
- Internal representation is private
- If we manipulate the internals, **errors can occur**

Advantages of OOP

- **Bundle data into packages** together with procedures that work on them through interfaces
- **Increase modularity** – reduces complexity
- **Code reuse**
  - Define once, use multiple times

Defining a Type with Classes

- Make a distinction between creating and using an instance of the class
- **Creating** involves:
  - Defining the class name
  - Defining class attributes
- **Using** involves:
  - creating a new instance of objects
  - Doing operations on the instance

Defining the Type

- Using the **class** keyword:

class Spoon(object):
    #define the attributes here

class name
Parent class

- This is similar to **def** and a method definition
- The word object means that **Spoon** inherits all of **object** attributes

Attributes

- Data and procedures that **belong** to the class are **attributes**
- **Data attributes**
  - The information inside the class
- **Methods**
  - Functions that perform an action on the data

Defining Initializers

- To create an instance of a class you must have a definition of what is required in a class for it to exist. This is called an **initializer**

```
class Spoon(object):
        def __init__(self, color, material):
                self.color = color
                self.material = material
```

Class parameters

References the class

Class Attributes

Adding and Using a Method

- A **method** is a function that works only within this class
- Python always passes the object as the first argument **self**
- Use a “**.**” off the class object to access both **attributes and methods**

```
class Spoon(object):
        def __init__(self, color, material):
                self.color = color
                self.material = material
        def wash(self, name):
                return f“{name} washed the {self.color} spoon”
```

Using Spoon’s wash()

```
>>> s = Spoon(“silver”, “plastic”)
>>> s.wash(“Andrew”)
Andrew washed the silver spoon
```

The Benefits of OOP

- **Bundle together objects** that share:
  - Common attributes
  - Common procedures
- Use **abstraction** to make the distinction between how to implement an object verse how to use it
- Build **layers** of object abstractions that inherit behaviors from other classes
- Create your own classes of object on top of existing basic classes