Objects Oriented Programming (OOPs) with Python

Dr. Sarwan Singh

Agenda

- OOPs – History, advantages
- Python
  - objects, self keyword
  - Constructor,
  - member functions, data
- e.g. Point class

References

- ocw.mit.edu
- Google colab
-

Python – History

- “**Object-Oriented Programming**” (**OOP**) was coined by Alan Kay circa 1966 or 1967 while he was at grad school. Ivan Sutherland's seminal Sketchpad application was an early inspiration for **OOP**. It was created between 1961 and 1962 and published in his Sketchpad Thesis in 1963.
- “The first programming language widely recognized as “object oriented” was Simula, specified in 1965.”

Objects in Python

- Python supports many different kinds of data
  - 1234 3.14159 "Hello" [1, 5, 7, 11, 13]
  - {"CA": "California", "MA": "Massachusetts"}
- each is an object, and every object has:
  - a type
  - an internal data representation (primitive or composite)
  - a set of procedures for interaction with the object
- an object is an instance of a type
  - 1234 is an instance of an int
  - "hello" is an instance of a string

Object in Python

- ***EVERYTHING IN PYTHON IS AN OBJECT*** (and has a type)
- can **create new objects** of some type
- can **manipulate objects**
- can **destroy objects**
  - explicitly using **del** or just “forget” about them
  - python system will reclaim destroyed or inaccessible objects – called “garbage collection”

Advantages of OOPs

- **bundle data into packages** together with procedures that work on them through well-defined interfaces
- **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

Creating & Using your own types with classes

- make a distinction between **creating a class** and **using an instance** of the class
- **creating** the class involves
  - defining the class name
  - defining class attributes
  - for example, someone wrote code to implement a list class
- **using** the class involves
  - creating new **instances** of objects
  - doing operations on the instances
  - for example, L=[1,2] and len(L)

Define your own types

- use the class keyword to define a new type

```
class Point (object):
        #define attributes here
```

- similar to **def**, indent code to indicate which statements are part of the **class definition**
- the word object means that **Point** is a Python object and **inherits** all its attributes
  - Coordinate is a subclass of object
  - object is a superclass of Coordinate