Programming and Database Fundamentals for Data Scientists

Classes and Objects

Varun Chandola

School of Engineering and Applied Sciences
State University of New York at Buffalo
Buffalo, NY, USA
chandola@buffalo.edu

Outline

Object Oriented Design

- Data-centric design instead of logic-centric
- Logic-centric design:
  - A program is organized as a logical procedure
  - Have functions as reusable logical blocks
- Data-centric design
  - A program is essentially a way to manipulate data
  - Data encapsulated as objects

How to do OOP?

- Identify objects that need to be manipulated in a program (data modeling)
- Define a `class` as a general description of the desired object
  - Example: Consider a banking application
  - Need to define customers
  - Each customer has a name, address, and multiple accounts
  - Each account has a type (checking or savings), current amount
  - Application: *Read data from* `csv` *files containing customer and account information and find all customers with more than $5,000 in their bank account*
  - A `class` will consist of the data and the methods needed to interact with the data

Encapsulation

- A fundamental tenet of **Object Oriented Programming**
- Allows programmers to control the flow of data in a program
- Every object has some data attached to it
- Not all data is acessible to the external program
- Encapsulation controls what methods and fields are visible and how

Python Classes

- Define a Python class using the keyword `class`
- During the program execution, you can *instantiate* objects of a certain class
- Each class has three entities:
  - A *constructor* (using a special function called `__init__`)
  - Fields containing various data elements (mutable or immutable)
  - Methods that let you manipulate the fields or perform any task
  - Fields and methods can be defined as *public* or *private*
    - Private – only accessible within the class definition
    - Public – accessible outside (`objectname.fieldname` or `objectname.methodname()`)

Python Scopes and Namespaces

Namespace

- Mapping from names to objects
- E.g., set of built-in function names, set of functions within a module, set of methods within an object definition
- One appends the module name or the object name, followed by a '.', followed by the object or method name

Scope of a Namespace

- A textual region within a Python program where a namespace is directly accessible without providing the qualifying object or module name

Global and Local Scope

- One can declare a global name using the keyword `global`

References