PASADENA CITY COLLEGE

CIS 112

Intro to Programming Using Python

Module 2 Part 1

Agenda for the Day!

- Intro to classes
    - Modularity
    - Classes
    - Instances

Class Basics

The next layer in modularity!

- A key element of scaling programs to accommodate increased functionality and complexity is to organize them in a way that allows for better modularity
  - We've seen this already with functions, allowing us to segment off self-contained code blocks that can be developed, tested, and then invoked on demand as part of a larger program structure
- However functions are limited in that they are strictly collections of executable code
  - This is limiting because we often have complex entities in our program that are collections of both functions and data oriented around a .
    - For instance:
      - A commerce platform will have products, that will include data like name, description, price, and inventory, as well as certain functions to update or display those data
      - Similarly a video game will have the concept of players, that will include data elements like name, job, powers, health level etc. and functions specific to managing those players as they navigate the game.
- The key in these examples is that we have these entities that exist in our program, often times with multiple copies, that we would like to organize in a modular and referentiable manner, such that we can easily create and invoke these on demand.

Class Defined

- A class expresses an idea; it’s a blueprint or recipe for an instance. The class is something virtual, it can contain lots of different details, and there is always one class of any given type.
  - Think of a class as a building blueprint that represents the architect’s ideas, and class instances as the actual buildings.
  - Another example might be a product in a retailer’s inventory
  - Another example is a player profile in a video game
- Classes describe attributes and functionalities together to represent an idea as accurately as possible.

Class Defined Cont.

- Classes are useful in two respects: 1. Organization, and 2. Modularity
    - **Organization:** Oftentimes data elements and functional elements will cluster around a single object (a videogame player will have certain data attributes (name, power level etc.) associated with it, along with certain functions (heal the player, promnote them etc.)storing and referencing them individually can be exhuastive, but clustering them within a class framework makes it easy to maintain and reference
    - **Modularity:** From the perspective of scaling program design, being able to bucket and organize repeating elements into self-contained structures is invaluable
- You can build a self-contained class from scratch or, employ inheritance to get a more specialized class based on another class.
- Additionally, your classes could be used as superclasses for newly derived classes (subclasses).
- Python’s class mechanism adds classes with a minimum of new syntax and semantics

Instances (and objects)

- Remember, when we define a class, we are simply creating the ‘mold’ or ‘recipe’ for an item in our program, not the actual item.
    - This is conceptually similar to a function. It is a recipe for an executable block of code, but doesn’t run until it’s invoked.
- Similarly class instances are only created once explicitly invoked.
- An **instance** is one particular actual instantiation of a class that occupies memory and has data elements. This is what 'self' refers to when we deal with class instances.
- An object is everything in Python that you can operate on, like a class, instance, list, or dictionary.
- The term instance is very often used interchangeably with the term object, because object refers to a particular instance of a class. It’s a bit of a simplification, because the term object is more general than instance.
- **The relation between instances and classes is quite simple: we have one class of a given type and an unlimited number of instances of a given class.**
- Each instance has its own, individual state (expressed as variables, so objects again) and shares its behavior (expressed as methods, so objects again).

Attributes

- An **attribute** is a general term that can refer to two major kinds of class traits:
  - **variables**, are containers of memory storing information about the class itself or a class instance; classes and class instances can own many variables;
  - **methods**, formulated as Python functions; represent a behavior that could be applied to the object.
- Each Python object has its own individual set of attributes. We can extend that set by adding new attributes to existing objects, change (reassign) them or control access to those attributes.
- It is said that methods are the 'callable attributes' of Python objects. By 'callable' we should understand anything that can be called; such objects allow you to use round parentheses () and eventually pass some parameters, just like functions.
- This is a very important fact to remember: methods are called on behalf of an object and are usually executed on object data.

Class basics!

Let’s recap classes!

- **class** – an idea, blueprint, or recipe for an instance;
- **instance** – an instantiation of the class; very often used interchangeably with the term 'object';
- **object** – Python's representation of data and methods; objects could be aggregates of instances;
- **attribute** – any object or class trait; could be a variable or method;
- **method** – a function built into a class that is executed on behalf of the class or object; some say that it’s a 'callable attribute';
- **type** – refers to the class that was used to instantiate the object.