CS521 O2
Information Structures with Python

Lecture 11

Guanglan Zhang

guanglan@bu.edu

Some slides adapted from Prof. Eugene Pinsky

Table of Content

Objected-oriented programming (OOP)

- OOP is an approach of structuring programs to bundle relevant properties and behaviors in objects
- An object interact with other objects to complete tasks
- Objects are at the center of OOP
- Another programming approach is procedural programming - structure a program to run step-by-step, in the form of functions and code blocks, that are executed sequentially to complete a task
- Python is object-oriented.
- The characteristics of the OOP approach
  - Encapsulation
  - Polymorphism
  - Inheritance

Objected-oriented programming (OOP)

- Encapsulation – one does not need to know the underlying details of a class in order to use it
  - ✓ Hides details of implementation to aid reading and understanding
  - ✓ Provides modularity that makes it easier to use a class
  - ✓ Provides an interface in the form of methods to access and manipulate a class instance
  - ✓ A new class should be consistent with the rules and syntax of the language
  - ✓ A new class should respond to “standard methods”

- Polymorphism – using an operator or method to perform different operations depending on the types that invoke them

- Inheritance – allows one class takes on the attributes and methods of another. Newly formed classes are called child classes, and the classes that child classes are derived from are called parent classes.

Python Class

- A **class** is a template for making a new object. Creating a new class creates a new **type**
- In Python, *object* is the root of all classes
- An **object** made by a class template is called an **instance** of the class
- The class template defines the structure and operations of an instance:
  - ✓ the attributes that might be contained in each instance
  - ✓ the methods define the operations that act on those attributes
- Python built-in classes include list, str, dict, set, tuple, etc.
- Class definitions start with the *class* keyword, followed by the name of the class, parentheses, name of the parent class, and a colon. Code that is indented below the class definition is the class's body

Methods are the interface to a class instance

- Functions that define the operations that can be done on an object are called methods
- Methods are defined in the suite of a class
- The method parameter *self* must always be placed as the first parameter in any method definition
- *Self* references to the calling object
- Python reserves some special methods that begin and end with two underscores __
  - __init__() initializes each new instance of a class
  - __str__() casts an instance of a class to a str object. It is for the end user.
  - __repr__() is the “official” string representation of an object, describing how you would make an object of the class
  - __copy__() allows copying of an object

Class attributes (static variables) & Instance attributes

```
class Student(object):
    school = "Brookline High“

    def __init__(self, first='', last='', id=0):
        self.first_name = first
        self.last_name = last
        self.id = id
```

- first_name, last_name, and id are instance attributes (or instance variables)
- Each instance has its own copy of instance attributes
- school is a class attribute (or static variable)
- It is defined directly beneath the first line of the class name (before methods)
- Class attributes must always be assigned an initial value
- It is the same across all instances
- Access a class attribute as className.class_attribute or instance.class_attribute

Public vs. Private

In OOP terminology

- Public attributes (variables, methods) are available to everyone
- Private attributes are only available to class designers
- Python has no enforcement mechanism for data privacy
- All attributes are public so that both class designers and programmers have access
- Privacy is indicated using double underscore __
- An attribute with 2 leading underscore __ is indicated as a private attribute
- To prevent accidently modification of private attributes, Python mangles an attribute named __attribute to _ClassName__attribute for outside use

Operator Overloading – magic methods

- Built-in types have common operators, such as +, <, ==
- The same built-in operator or function shows different behavior for objects of different classes
- This is called operator overloading or function overloading respectively
- We can override built-in methods using magic methods
- Python associates special method names with operators of the following 3 classes:
  - ✓ Arithmetic operators, such as +, -, *, /
  - ✓ Collection operators, such as [], len
  - ✓ General class operators, such as methods for printing and construction

Math-like operators

Operator	Method
+	__add__(self, other)
-	__sub__(self, other)
*	__mul__(self, other)
**	__pow__(self, other)
/	__truediv__(self, other)
//	__floordiv__(self, other)
%	__mod__(self, other)
<	__lt__(self, other)
<=	__le__(self, other)
>	__gt__(self, other)
>=	__ge__(self, other)
==	__eq__(self, other)
! =	__ne__(self, other)
&	__and__(self, other)
|	__or__(self, other)
+=	__*iadd*__(self, other)

Sequence operators

Operator	Method	Description
len()	__len__(self)	length of the sequence. Python requires the function to return an integer
x in y	__contains__(self, x)	Does the sequence y contain x
x[key]	__getitem__(self, key)	access element key of sequence x
x[key]=y	__setitem__(self, key, y)	set element key to value y

Exercises

Create a class named *Circle*.

- It take a parameter radius to construct an instance of the class.
- The default radius is 1.
- print(instance of Circle) should print out appropriate message
- Include a method area() that calculates area of a circle
- Implement the following:
  1. '+': new circle with r = max(r1, r2)
  2. '-': new circle with r = min(r1, r2)
  3. '==': True if both radii are the same
  4. '>': True if r1 > r2

- Run the following code

```
c1 = Circle(2)
c2 = Circle(5)
c3 = c1 + c2
print(c1 == c2)
print(c1 > c2)
```

Key takeaways

- The characteristics of the OOP approach
  - Encapsulation
  - Polymorphism
  - Inheritance

- In Python, *object* is the root of all classes

- Python has no enforcement mechanism for data privacy. All attributes are public so that both class designers and programmers have access

- Privacy is indicated using double underscore __