**IS201 Fundamentals of Computing**

**HOP07 Object-Oriented Programming - Continued**

03/16/2020 Developed by Amrutha Vaidyanathan

10/25/2020 Reviewed by Kim Nguyen

Center for Information Assurance (CIAE) @City University of Seattle (CityU)

 

**Before You Start**

* The directory path shown in screenshots may be different from yours.
* Some steps are not explained in the tutorial**.** If you are not sure what to do:
  1. Consult the resources listed below.
  2. If you cannot solve the problem after a few tries, ask a TA for help.

**Learning Outcomes**

* Learn ways to pass information to functions
* How to write certain functions whose primary job is to display information and other functions designed to process data and return a value or set of values.
* Learn to store functions in separate files called modules to help organize your main program files.
* How to use debugger tools

**Resources**

Matthes, E. (2019). [Python Crash Course: A Hands-On, Project-Based Introduction to Programming, 2nd Edition](https://login.proxy.cityu.edu/sso/skillport?context=146803). No Starch Press. (ISBN 9781593279288)

**If you have completed all tasks in this document from previous weeks, move on to the Challenge document in Module 7 folder**

**Preparation**

In Visual Studio Code, open the private repository generated when you accepted the HOP07 assignment (If you cannot find that repository in your machine, you might have not cloned the repo, if so, please do before proceeding).

**Why OOP?**

* Modularity — Separating entities into separate logical units makes them easier to code, understand, analyze, test, and maintain.
* Data hiding (encapsulation) — The implementation of an object’s private data and actions can change without affecting other objects that depend on it.
* Code reuse through:
  + Composition — Objects can contain other objects
  + Inheritance — Objects can inherit state and behavior of other objects
* Easier design due to natural modeling

**Inheritance**

Inheritance is the process by which 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*.

It’s important to note that child classes override *or* extend the functionality (e.g., attributes and behaviors) of parent classes. In other words, child classes inherit all of the parent’s attributes and behaviors but can also specify different behavior to follow. The most basic type of class is an object, which generally all other classes inherit as their parent. When you define a new class, Python 3 it implicitly uses object as the parent class.

The below program is an example of single level inheritance where a child class inherits from only one parent class.

1. Create a file **Single\_Inheritence.py** and type the following code

A screenshot of a cell phone

Description automatically generated

Read the comments when working through this program to help you understand how inheritance works.

In the terminal type **python3 Single\_Inheritence.py** to see the output.

A close up of a sign

Description automatically generated

**Multiple Inheritance:**

When a child class inherits from multiple parent classes, it is called as multiple inheritance.  
Unlike Java and like C++, Python supports multiple inheritance. We specify all parent classes as comma separated list in bracket.

1. Create a file **Multiple\_Inheritence.py** and type the following code.

A close up of text on a black background

Description automatically generated

The dog is the child class or derived class which derives the values from the parent classes Animal and Color.

In the terminal type **python3 Multiple\_Inheritence.py** to see the output.

Graphical user interface, application

Description automatically generated

**Multilevel Inheritance:**

A subclass inherits from another sub class which in turn may inherit from another subclass thus having levels (parent – child – grandchild) of hierarchy.

1. Create a file **Multilevel\_Inheritence.py** and type the following code.

A screenshot of a cell phone

Description automatically generated

In the above code the Animal is the base class, Dog is the child class and pug is the grandchild class.

In the terminal type **python3 Multilevel\_Inheritence.py** to see the output.



There is other type of inheritances.

Hierarchical inheritance where more than one derived class are created from a single base

Hybrid inheritance is the combination of more than one form of inheritance. Basically, it is a blend of more than one type of inheritance.

**Encapsulation**

In Python, we can restrict access to methods and variables. This prevent data from direct modification which is called encapsulation. In Python, we denote private attribute using underscore as prefix i.e single “ \_ “ or double “ \_\_“.

1. Create a file **Data\_encapsulation.py** and type the following code

A screenshot of a cell phone

Description automatically generated

We defined a class Computer and use \_\_init\_\_() method to store the maximum selling price of computer. We tried to modify the price. However, we can’t change it because Python treats the \_\_maxprice as private attributes. To change the value, we used a setter function i.e setMaxPrice() which takes price as parameter.

In the terminal type **python3 Data\_encapsulation.py** to see the output.

A black sign with white text

Description automatically generated

**Polymorphism**

Polymorphism is an ability (in OOP) to use common interface for multiple form (data types).

1. Create a file **Polymorphism.py** and type the following code

A screenshot of a cell phone

Description automatically generated

we defined two classes Parrot and Penguin. Each of them have common method fly() method. However, their functions are different. To allow polymorphism, we created common interface i.e flying\_test() function that can take any object. Then, we passed the objects blu and peggy in the flying\_test() function, it ran effectively.

In the terminal type **python3 Polymorphism.py** to see the output.

A picture containing object, drawing, ball

Description automatically generated

There are 3 types of polymorphism: Method overloading, Method overriding and Operator overloading

**Method overloading:**

Method overloading in its traditional sense, where you can have more than one method having the same name within the class where the methods differ in types or number of arguments passed, is not supported in Python. Trying to have methods with same name won’t result in compile time error in Python but only the last defined method is recognized in such scenario, calling any other overloaded function results in an error. But you can still simulate polymorphism through method overloading by using default arguments in a method.

1. Create a file **method\_overloading.py** and type the following code

A close up of text on a screen

Description automatically generated

In the example there is one default argument in the method sum. If the method is called with 2 parameters for the third default value is used. If the method is called with 3 parameters passed value is used for the third parameter.

In the terminal type **python3 method\_overloading.py** to see the output.



**Method overriding:**

Method overriding provides ability to change the implementation of a method in a child class which is already defined in one of its super class. If there is a method in a super class and method having the same name and same number of arguments in a child class, then the child class method is said to be overriding the parent class method.

When the method is called with parent class object, method of the parent class is executed. When method is called with child class object, method of the child class is executed. So, the appropriate overridden method is called based on the object type, which is an example of Polymorphism. The below example shows how polymorphism is used through inheritance.

1. Create a file **method\_overriding.py** and type the following code

A screenshot of text

Description automatically generated

when the displayData() method is called with Person class object, displayData() of the Person class is executed. When displayData() method is called with Employee class object, displayData() of the Employee class is executed.

In the terminal type **python3 method\_overriding.py** to see the output.

A picture containing drawing, food, bird

Description automatically generated

**Operator overloading:**

Operator overloading means the ability to overload the operator to provide extra functionality in addition to its real operational meaning. Operator overloading is also an example of polymorphism as the same operator can perform different actions.

You can also overload operators to provide functionality for custom class objects. For all operators internally Python defines methods to provide functionality for those operators.

1. Create a file **operator\_overloading.py** and type the following code

A close up of a screen

Description automatically generated

The functionality for ‘+’ operator is provide by special method \_\_add\_\_(). Whenever ‘+’ operator is used internally \_\_add\_\_() method is invoked to do the operation. When you want any operator to work with custom objects you need to override the corresponding special method that provides functionality for that operator.

In the terminal type “python3 operator\_overloading.py” to see the output.



**Push your work to GitHub**

Open the terminal from the VSCode by hitting the “control” + “~” key and type the following command:

>>> git add .

>>> git commit -m “Submission for Module 7 – Your Name”

>>> git push origin master