Intro to Computer Science Project—Object-Oriented Programming

ASSIGNMENT OVERVIEW

In this assignment you'll be creating a short program called `oop.py`, which will define a class with attributes, accessor methods, and mutator methods, and include a brief example of using an object of that class in a main program.

This assignment is worth 50 points and is due on the crashwhite.polytechnic.org server at 23:59:59 on the date given in class.

BACKGROUND

Object-oriented programming (OOP) is a style of programming that organizes data, and the programs that manipulate that data, into *objects* that are organized into *classes*. Each object has *attributes*—values that describe the object—and *methods*—things that the object can "do". Methods may be *accessor* methods that allow you to examine the object's attributes, or *mutator* methods which cause a change in an object's attributes.

Object-oriented programming is built into Python—print `"Hi there".capitalize()` is a quick example of using an object, where the string object `"Hi there"` is retrieved by the accessor method `.capitalize()`. You may not have known you were using an object, and that's partly the point of OOP—a well-written program will allow you to manipulate objects without having to worry too much about what's happening behind the scenes, hidden in the "black box" of the class. This ability to conceal some of the underlying details of coding is called *encapsulation*.

Python provides ready-made classes (like `int`, `str`, and `list`) that can be manipulated using ready-made methods (like `float`, `len()`, and `.pop()`). The real power of objects, however, comes when a programmer is able to custom-design a class, often to represent a model of some Real World object. Some examples:

- A `BankAccount` class could be used to store and manipulate data for a series of bank account objects
- A `Planet` class could be used to store and manipulate data for a planet object that revolves around a star.
- A `Contact` class could be used to store and manipulate data for a contact object in an address book.
- A `Frog` class could be used to store and manipulate data for... a frog.

PROGRAM SPECIFICATION

Create a Python program that:

a. creates a user-defined **class** that can be used by a program to establish and manipulate some sample objects
b. includes a **constructor** class that establishes an object of the given class
c. includes at least three **accessor methods** for accessing the attributes of the class
d. includes at least two **mutator methods** for altering attributes of the class
e. includes a **main()** program that uses the class to create an object, and manipulate it using its accessor and mutator methods

DELIVERABLES

`oop.py`

You should keep a working copy of this file in your home folder on the server and a backup copy of the file elsewhere. To submit your assignment for grading, copy your file to your `/home/userID/forInstructor` directory at `crashwhite.polytechnic.org` before the deadline.

ASSIGNMENT NOTES

- Here's a full example of a program that creates a class `Frog` and then shows both how to use that class to create and manipulate models of a couple of different frogs.

```
#!/usr/bin/env python3
"""Demonstration on how to create and use an object in Python
Richard White, 2013-03-20"""

class Frog:
    """The Frog class demonstrates the use of
       attributes, constructor methods,
       accessor methods, and mutator methods."""

    def __init__(self, commonName = "bullfrog"):
        """This is the constructor method that constructs the Frog
        when we first create it."""
        self.name = commonName  # assigns parameter to attribute
        self.location = [0,0]   # establishes initial X,Y position
        self.age = "egg"        # establishes initial age

    def getName(self):          # Accessor methods used to
        return self.name        # look at state of attribute

    def getLocation(self):
        return self.location

    def getAge(self):
        return self.age

    def grow(self):             # Mutator method
        ages = ["egg","tadpole","adult","dead"]
        if self.age != "dead":
            self.age = ages[ages.index(self.age)+1]

    def vocalize(self):
        if self.name == "bullfrog": return "Brrrup!"
        elif self.name == "peeper": return "Peep!"
        else: return "Ribbit!"

    def jump(self,distance=[1,1]):
        self.location[0] += distance[0]
        self.location[1] += distance[1]

    # program continues on next page
```

```
def main():
    froggie1 = Frog()      # default frog is "bullfrog"
    froggie2 = Frog("leopard frog")
    print("My first frog is a",froggie1.getName())
    froggie2.grow()
    print("My",froggie2.getName(),"is a",froggie2.getAge())
    print(froggie2.vocalize())
    loudNoise = True
    if loudNoise:
        froggie1.jump()    # default distance is +1, +1
    froggie2.jump([-2,4])
    print(froggie1.getLocation())

if __name__ == "__main__":
    main()
```

Output:

```
My first frog is a bullfrog
My leopard frog is a tadpole
Ribbit!
[1, 1]
```

- If you're feeling adventurous you can think about designing a completely different type of class, with its corresponding attributes and methods. How about an `Automobile` class? A `Book` class? A `CoffeeMachine` class? Just make sure you can identify some attributes and methods that will be appropriate for your class.

- If you want to play it safer, come up with a class for some animal, and use the `Frog` class above as a model that you can alter as needed, with different attributes and methods appropriate for your animal.

- If you'd prefer to work off a specification, do *one* of these:
  - Create a class called `Human` that takes a name, gender, and age as parameters.
    - Attributes for the class include `name`, `gender`, and `age`.
    - Create three accessor methods: `getName`, `getGender`, and `getAge`.
    - Create two mutator methods: `celebrateBirthday` and `changeName`.
    - Write a main program that creates an object with your name, gender, and age. Print out the status of all three attributes, then celebrate a birthday and change your name, and print out the status of all three attributes again.
  - Create a class called `Contact` that takes a name as a parameter.
    - Attributes for the class include `name`, `phone`, `emailAddress`, and `birthday`.
    - Create three accessor methods: `getName`, `getPhone`, `getEmail`, and `getBirthday`.
    - Create two mutator methods: `changePhone`, and `changeEmail`.
    - Write a main program creates a contact with your name, phone, and email. Print out the status of all the contact's attributes, edits some of the attributes, and prints out the status of the attributes again.

GETTING STARTED

1. With paper and pencil, and perhaps in collaboration with a partner, identify what the main components are that you'll need to include in your program.
2. Sketch out the basic flow of your program using a flowchart, and write some pseudocode that you can use to begin implementing those main components.
3. Either on your personal computer or on the *crashwhite.polytechnic.org* server, open up two windows: a text editor to write the program (on the left), and a shell to perform test runs of your program on the right.
4. Using the `Frog` class above as a model, write the code that defines your class.
5. Write a test program—the `main()`—that will create one or two objects of the class that you've defined, and demonstrate how it may be manipulated.
6. Save your program with the required name.
7. Copy this initial version of the program to the *crashwhite.polytechnic.org* server (to make sure that the upload/copy process works).

```
$ scp oop.py studentID@crashwhite.polytechnic.org:
/home/studentID/forInstructor
```

8. Switching to the shell, run your program as new features are added, making sure to fix old problems before adding new components. You'll repeatedly run through this edit-run, edit-run process to find bugs and fix them.
9. When your program is completed (but before the deadline), copy it to the server as indicated above. The newer version of your program will replace the old one there.

QUESTIONS FOR YOU TO CONSIDER (NOT HAND IN)

1. We've looked at Python functions that accept integers and strings as parameters, so that data can be passed into a function. Is it possible to pass an entire list of values into a function so that the function can work with it? Is it possible to pass an object into a function so that the function can work on it?
2. In some programs we've seen a "list of lists." Is it possible to have an "object of objects?" What types of objects would compose a `SolarSystem` object? What larger object would many `SolarSystem` objects form?

SAMPLE INTERACTIONS

None