Inheritance

Sometimes, objects are closely related or have similar properties and functionality. Consider the Graphical User Interface (GUI) that makes up a modern operating system. A dialog box might provide several buttons for the user to click. These buttons share some properties (can be clicked, have coordinates in the window, display text, etc.), but some buttons have additional functionality (display icons, can be coloured, etc.). Instead of writing separate classes for each type of button (`IconButton`, `ColouredButton`) that contain the same attributes and methods as `Button`, it is possible to *reuse* the attributes and methods from `Button` as part of their definitions. This reduces the amount of code, and time, needed to make a program.

When a class uses properties from another class, this is called **inheritance**. A class which inherits methods and attributes from another is called a **derived class**, whereas the class from which it inherits is a **base class**. In the example above, `IconButton` and `ColouredButton` are derived classes, and Button is the base class. To derive properties from a base class, include the name of the base class inside brackets after the name of the derived class, such as `class IconButton(Button)`. Python supports inheritance from a single base class, or from multiple base classes.

All properties and methods from the base class will be available to the derived class. To call the initialization method from the base class, use `super()` as a prefix, as in `super().__init__(ARGS)`.

Definition of a Derived Class

Single inheritance.

```
class DERIVEDCLASSNAME(BASECLASSNAME):
      def __init__(self, ARGUMENTS):
            # initializer code goes here
      def METHODNAME(self, ARGUMENTS):
            # method code goes here
      ...
```

Multiple inheritance.

```
class DERIVEDCLASSNAME(BASECLASS1, BASECLASS2, ...):
      def __init__(self, ARGUMENTS):
            # initializer code goes here
      def METHODNAME(self, ARGUMENTS):
            # method code goes here
      ...
```

Inheritance

Answer the following questions.

1. What are some advantages and disadvantages of class inheritance with respect to code maintenance (adding, modifying, correcting code)?

2. What is the output of the following code?

```
class A:
    def __init__(self, x):
        self.x = x
    def calc(self):
        return 2*self.x
class B(A):
    def __init__(self, x):
        self.x = x
    def calc2(self):
        return 3*self.x
    def calc3(self):
        return self.calc()
```

```
b = B(5)
print(b.calc())
print(b.calc2())
print(b.calc3())
```

Write programs that accomplish each task, using appropriate programming conventions.

3. Create a class, `Hero`, that represents a character in a role-playing game. A `Hero` has the following integer-based attributes: `strength`, `dexterity`, and `intelligence`. Create two derived classes, `Warrior` and `Wizard`, that inherit from `Hero`. A `Warrior` has the additional attribute `endurance`, while a `Wizard` has the additional attribute `focus`. Create instances of a `Warrior` and a `Wizard`, and display their stats in a nicely formatted manner.

4. Create a class, `Animal`, that has the attributes `name` and `legs`. Create three derived classes that inherit from `Animal`: `Dog`, which has the method `bark()`; `Bird`, with the method `tweet()`; and `Snake`, with the method `hiss()`. Each method should contain a `print` statement indicating that the animal is making a particular sound. Create an instance of each type of animal, providing each instance with a name (e.g. German Shepherd) and number of legs (e.g. 4). Calling an animal's sound method should produce output similar to that below.

```
The 4-legged German Shepherd is barking.
```

5. Create a class, `Circle`, that has the "private" attribute `radius`, as well as the following methods: `get_radius()`, `set_radius()`, and get_area(). Create a derived class, `Cylinder`, that inherits from `Circle`. `Cylinder` has an additional "private" attribute, `height`, as well as the following methods: `get_height()`, `set_height()` and `get_volume()`. This last method should call `Circle`'s `get_area()` method as part of its calculations. Create an instance of `Cylinder` and display its volume.