Polymorphism

Polymorphism (literally "many forms") refers to the ability to process objects in different ways, depending on their data type or class. For example, a `Circle` may have a `get_area()` method that calculates the area using the formula $A=\pi r^2$, while a `Rectangle` may also have a `get_area()` method that uses $A=\ell w$ instead. Both objects use the same name for the method, but the implementations are different. This practice is known as **method overriding**.

In addition to using common names, it is possible to override operators and other functions. For example, the "+" symbol has two different meanings for different data types: for number types, it adds two values, while for sequence types, it concatenates them. To extend or change this functionality, the `__add__()` method can be redefined. A handy method to override is `__str__()`, which returns a string when `print()` is called on a class, as in the following example.

```
class MyClass:
    def __init__(self, x):
        self.x = x
    def __str__(self):
        return "This class contains the value "+str(self.x)
c = MyClass(5)
print(c)
```

Some programming languages allow for purely **abstract classes** (those that contain methods with definitions, but no implementation). Python does not inherently support abstract classes, but can implement them by importing the ABC (Abstract Base Class) module.

Typical Polymorphic Setup for Derived Classes

```
class BASECLASS:
      def __init__(self, ARGUMENTS):
            # initializer code goes here
      def MyMethod(self, ARGUMENTS):
            # method code goes here

class DERIVEDCLASS1:
      def __init__(self, ARGUMENTS):
            # initializer code goes here
      def MyMethod(self, ARGUMENTS):
            # overridden method code goes here

class DERIVEDCLASS2:
      def __init__(self, ARGUMENTS):
            # initializer code goes here
      def MyMethod(self, ARGUMENTS):
            # overridden method code goes here
```

Polymorphism

Answer the following questions.

1. What are some advantages of using polymorphism in your code?

2. What are some attributes and methods that are common to an `Employee` and an `Employer`? What are some attributes and methods that are specific to each class?

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

3. Modify your `Animal` class from the previous worksheet. Instead of separate methods `bark()`, `tweet()` and `hiss()`, create a `make_noise()` method for the `Animal` class that indicates that the animal is making a strange noise. Override this method in the `Dog` and `Bird` classes to use their output noises (barking and tweeting). Leave the `Snake` class alone. Create instances of each class and call `make_noise()`. Output should be similar to that below.

```
The German Shepherd is barking.                (overridden method)
The Parakeet is tweeting.                      (overridden method)
The Rattlesnake is making a strange noise.     (base class method)
```

4. Modify your `Hero` class from the previous worksheet. Both `Warrior` and `Wizard` inherit from `Hero` as described. Hero should have an additional attribute, `health`. Create an `attack()` method in both `Warrior` and `Wizard`. A `Warrior`, when hit by an enemy, should have damage taken reduced by 10% of the `endurance` value, rounded up to the nearest integer (e.g. if endurance is 17, `health` is reduced by 2 fewer points). A `Wizard`, when attacking with a spell, inflicts an additional point of damage for every 10 points of `focus` (e.g. if `focus` is 23, 2 additional points are scored). Implementation is up to you. Keep things simple at this point.

5. Trading card games, like *Pokémon* or *Yu-Gi-Oh*, use characters with various attributes and attacks. For example, Pikachu is an electric type with 35 HP, 55 attack, 40 defense, etc. You can read about *Pokémon* here. Create a base class, `Pokemon`, that defines the base attributes for a character. Create three (or more) derived classes of your choice for various *Pokémon*, that have `attack()` and `special_attack()` methods. Implement them as you see fit.