Using Object-Oriented Design Principles with Pygame

When you are creating objects that are very similar to others, or when you are creating multiple variations of similar objects, consider using classes. Define attributes and methods that are common to all objects, and create derived classes that inherit from base classes. Using `Sprite` as a base class provides access to all of the attributes and methods (e.g. collision-detection) that are available in `Sprite`.

Consider a game in which there are several different types of monsters: soldiers, skeletons, goblins and trolls. Each of these have common traits (HP, attack power), as well as different abilities (attacking, wielding magic). We might create a base class, `Enemy`, from which we can derive other classes, such as `Soldier`, `Skeleton`, and so on. Below is code that defines `Enemy` and `Soldier` classes, and uses images in the spritesheet `enemies.png`.

```
ss = spritesheet.SpriteSheet('enemies.png')

class Enemy(pygame.sprite.Sprite):
    def __init__(self, HP, attack, evade, weapon):
        super().__init__()
        self.HP = HP
        self.attack = attack
        self.evade = evade
        self.weapon = weapon

class Soldier(Enemy):
    def __init__(self, HP, attack, evade, weapon, x, y):
        super().__init__(HP, attack, evade, weapon)
        self.image = ss.image_at((35, 0, 200, 250), -1)
        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.y = y
    # DEFINE YOUR METHODS HERE...
    def attack():
        # CODE GOES HERE...
        print(“Soldier is attacking!”)
```

Source: OpenGameArt.org

Obviously there is much to improve on here, but the basic idea is in place. Now a Soldier can be instantiated with a simple call like below.

```
S = Soldier(20, 12, 10, "sword", 100, 100)
```

Try blitting `S` to the surface to see the soldier. It is trivial to add additional soldiers, or to create soldiers with slightly different (random?) stats, as necessary for your game.

Using Object-Oriented Design Principles with Pygame

Answer the following questions.

1. What are some of the advantages of using OOP principles?

2. What are some of the disadvantages of using OOP principles?

Write programs that accomplish each task, using appropriate programming conventions. Download the file `sprites_oop_files.zip` for the images.

3. Create a `Ball` class, which draws a circle of a specified colour, radius and position. Draw three balls in random locations on the screen.

4. Modify your program above so that, if two balls are colliding, they are coloured red. Otherwise, colour them green.

5. Create the base class `Animal`, derived from the `Sprite` class. with the method `speak()`. Create derived classes `Cow`, `Pig` and `Sheep` that override `speak()` appropriately (e.g. “MOO!” for the cow). Use `cow.png`, `pig.png` and `sheep.png` for their images. Generate a random instance of an animal, display it to the surface, then have it `speak()`.

6. Create derived classes for `Skeleton`, `Goblin` and `Troll`, using the images from the spritesheet `enemies.png`. Use an image-editing program to determine the *x*- and *y*-offsets for each image. Have your program generate a random enemy, then display its image along with its stats.

7. Modify your program above so that when the user presses an arrow or WASD key, it moves the sprite in the appropriate direction.

8. Modify your program above so that it generates two sprites. One is stationary, while the other is controlled by the user. If the user presses a directional key that would cause the sprites to collide, do not move the sprite in that direction. Instead, play a sound and make the background of the surface red. If another key is pressed and the sprite is able to move, revert the background to its original colour.