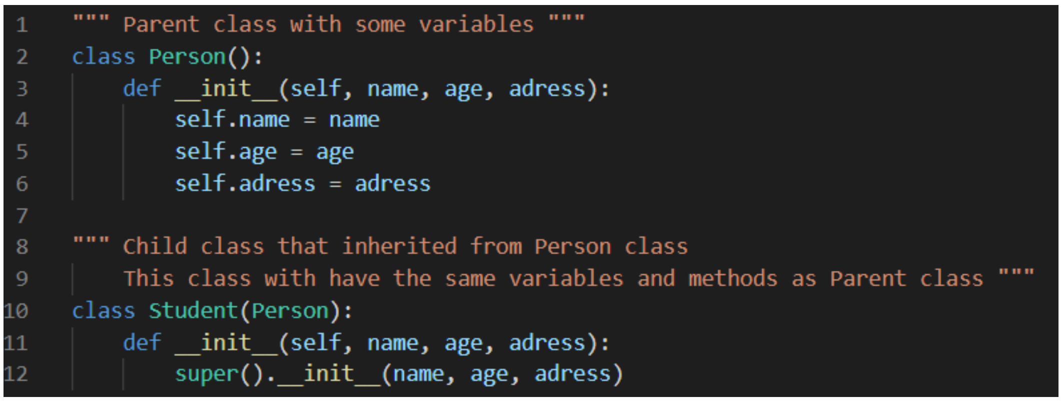
What is the difference between a class and an object?

- You could look at a class like a blueprint, it holds the details of what it can do, while an object is a collection of data and variables, described by the details of the class

What is inheritance? What is the Python syntax for inheritance?

- Inheritance is a concept that comes from OOP, it is used to share different variables and methods to other classes. The idea is that if you know that you are going to create at least two different classes that are almost the same. You can create a parent class which contains the information they have in common and share it to them.
  The syntax would look like this:



```python
""" Parent class with some variables """
class Person():
    def __init__(self, name, age, adress):
        self.name = name
        self.age = age
        self.adress = adress

""" Child class that inherited from Person class
    This class with have the same variables and methods as Parent class """
class Student(Person):
    def __init__(self, name, age, adress):
        super().__init__(name, age, adress)
```

What is the difference between a has-a and an is-a relationship?

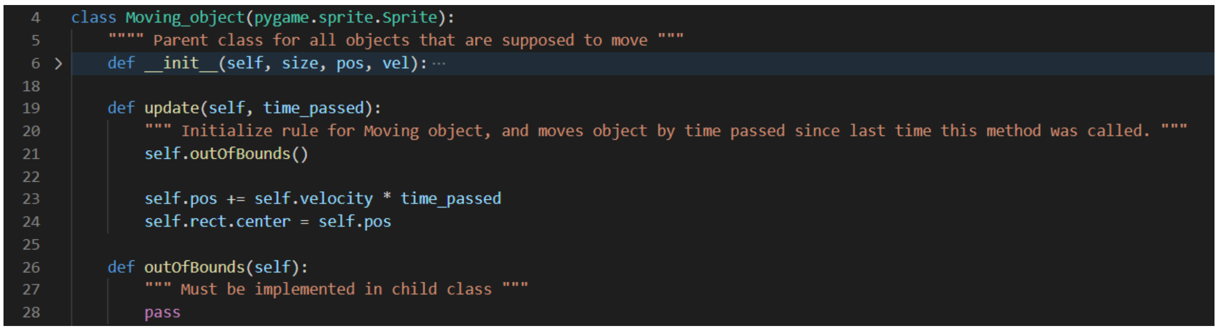
- The similarity between a has-a and an is-a relationship is that they support the ability to reuse code, but they do it in different ways.
  Is-a relationship is inheritance, if you have a parent class called Animal, you can use this class to create a child class called Horse, and extend this class to fit a horse, but the inheritance relationship states that the Horse class **Is-a** Animal class.
  Has-a relationship is composition, it enables you to add objects to other objects, opposite from inheriting the whole interface and implementation of other classes. If you would like to add a tail method to your horse and dog class, you could create a class called Traits, and create a tail method and a bunch of other traits. But your horse and dog class would only like a tail, saying that a Horse class **Has-a** tail.

What is encapsulation? How is encapsulation handled in Python?

- Encapsulation is the idea of bundling together variables and methods that work on variables into one unit. This also puts restriction on accessing variables and methods within the unit to prevent unwanted changes. Therefore, an object's variable can only be change by an object's method, meaning that the variable is private. In Python, there is no way to make a variable private. However, there is a convention that is being followed by most programs and programmers, that is a name with an underscore before it should be treated as a non-public variable/method (_name).

What is polymorphism? Give example of polymorphism from the precode and the Mayhem implementation.

- Polymorphism in programing means that a function/method with the same name can have many different forms. Meaning, that a function with the same name can have different appearances and give different results.
- Example from Mayhem implementation:
  - Parent class "Moving_object" with a method called "outOfBounds" that must be implemented in child class.



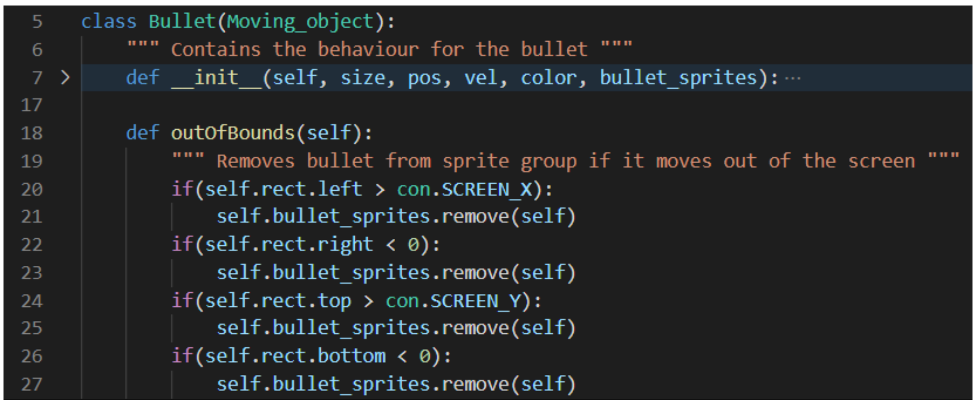
```python
class Moving_object(pygame.sprite.Sprite):
    """" Parent class for all objects that are supposed to move """
    def __init__(self, size, pos, vel):···

    def update(self, time_passed):
        """ Initialize rule for Moving object, and moves object by time passed since last time this method was called. """
        self.outOfBounds()

        self.pos += self.velocity * time_passed
        self.rect.center = self.pos

    def outOfBounds(self):
        """ Must be implemented in child class """
        pass
```

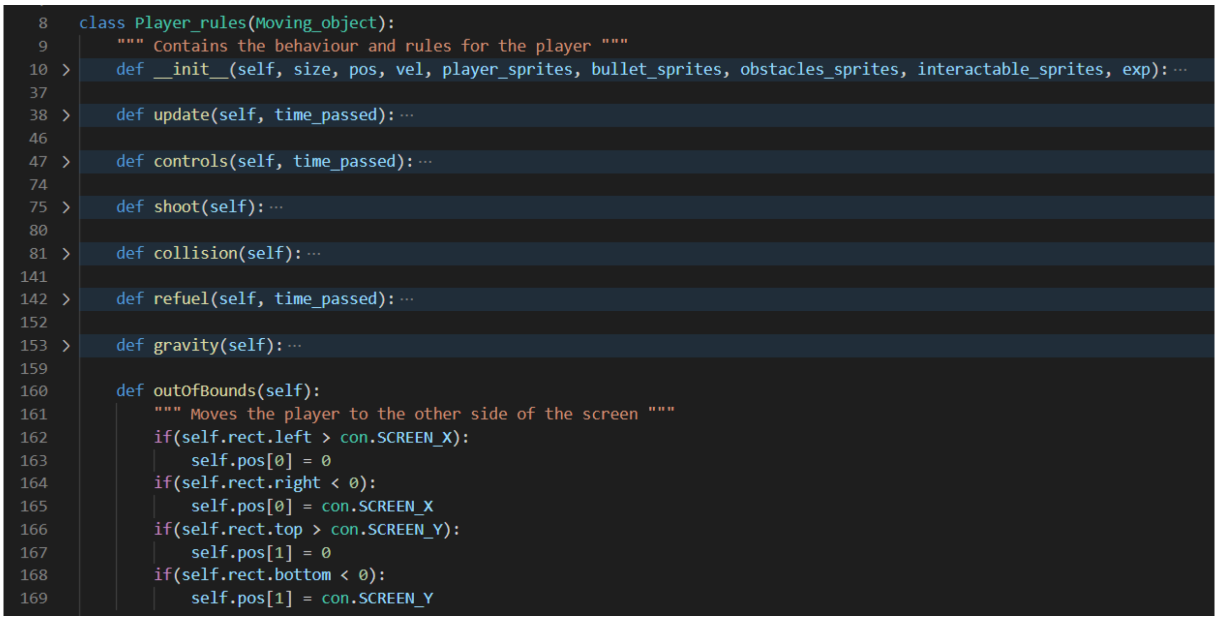
  - Child class "Bullet" with an implemented version of the "outOfBounds" method



```python
class Bullet(Moving_object):
    """ Contains the behaviour for the bullet """
    def __init__(self, size, pos, vel, color, bullet_sprites):···

    def outOfBounds(self):
        """ Removes bullet from sprite group if it moves out of the screen """
        if(self.rect.left > con.SCREEN_X):
            self.bullet_sprites.remove(self)
        if(self.rect.right < 0):
            self.bullet_sprites.remove(self)
        if(self.rect.top > con.SCREEN_Y):
            self.bullet_sprites.remove(self)
        if(self.rect.bottom < 0):
            self.bullet_sprites.remove(self)
```

- Child class “Player_rules” with a different implemented version of the “outOfBounds” method



```python
class Player_rules(Moving_object):
    """ Contains the behaviour and rules for the player """
    def __init__(self, size, pos, vel, player_sprites, bullet_sprites, obstacles_sprites, interactable_sprites, exp):···

    def update(self, time_passed):···

    def controls(self, time_passed):···

    def shoot(self):···

    def collision(self):···

    def refuel(self, time_passed):···

    def gravity(self):···

    def outOfBounds(self):
        """ Moves the player to the other side of the screen """
        if(self.rect.left > con.SCREEN_X):
            self.pos[0] = 0
        if(self.rect.right < 0):
            self.pos[0] = con.SCREEN_X
        if(self.rect.top > con.SCREEN_Y):
            self.pos[1] = 0
        if(self.rect.bottom < 0):
            self.pos[1] = con.SCREEN_Y
```

References

nikhilaggarwal3. (2022, Mar 17). *GeeksforGeeks.* Hentet fra Encapsulation in Python: https://www.geeksforgeeks.org/encapsulation-in-python/

Rodriguez, I. (2022, 04 04). *Real Python.* Hentet fra Inheritance and Composition: A Python OOP Guide: https://realpython.com/inheritance-composition-python/