1. Explain what happens when a class inherits from another class.
    a. When a class inherits from another class it becomes a parent child situation. The parent is the super class or the class that is being inherited from while the child is a sub class that inherits methods and attributes from it's parents. This allows the sub class to use attributes and methods created in it's parents class as well as any that are created within it's own class.

2. Explain polymorphism and give an example. How is this helpful and what problem does it solve?
    a. Polymorphism is the modification of methods and/or attributes that a sub class inherits from the super class. This is helpful because it allows for adjustments of methods/attributes without having to have excess code. For example if a sub class inhearited this function:

    ```
    def print_statment(self):
        print 'Hello my name is'
    ```

    It could modify the code through polymorphism to this:

    ```
    def print_statment(self):
        print 'Hello my name is' + user_name
    ```

3. Compare and contrast Aggregation and Composition. Include examples in your explanation.
    a. Composition is when parts of the whole are dependent on the whole. For example a house can exist without a room but a room cannot exist without a house. Aggregation is when the parts that add up to the whole can exist independently of each other. For example a class is made up of students but if the class did not exist the students would still exist.

4. Explain encapsulation and it's purpose. How do access modifies and getters and setters aid encapsulation? What problem does it solve?
    a. Encapsulation is a way to hide information from users as well as our future selves; its purpose is to protect information. Access modifiers help to aid encapsulation by allowing the programmer to determine which variables are public, protected and private. Getters help to return information in a read only format and setters produce write only. This helps to protect application variables.

5. What is an abstract class? Include an example. How is an abstract class helpful?
    a. An abstract class is a class broad enough that it could be a blueprint for many subclasses. It contains attributes and methods that are standard amongst the sub classes being created off of it. For example:

```
class Shoes(object):
    def __init__(self):
        self.color = ' '
        self.brand = ' '
        self.size = ' '
```

6. What does the MVC initials stand for and explain how this design pattern is used in the organization of code.
    a. MVC stands for model, view and, controller. Breaking code into MVC allows for more organized code. All of the data information is held in the model, the view contains the information that we are showing to the user and the controller controls how the model and view interact with each other.