1. What is the difference between a class and an object?

A class is a blueprint. It's a recipe for how to create an instance of it.
An object is made from a blueprint. You could for example have a blueprint on how a house should be constructed, and if the house is built it is now an object.
The house is an object of the blueprint.

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

A class can inherit from another. That means it can inherit attributes and methods from this class. The syntax for inheritance is really simple. If you have a class named Fruit, and another class named Apple, you can inherit from fruit just by writing the sub-class this way:
Class Apple(Fruit)

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

The difference between a has-a and an is-a relationship is that is-a is based on inheritance and by that I mean that an apple is-a fruit.

A has-a usually refers to a property of a class. In python this would be the attributes the class has. A salmon has gills, a salmon is a fish.

4. What is encapsulation? How is encapsulation handled in Python?

Encapsulation means restricting the access to some of the objects component.
This means that you can't edit attributes of an object, unless you use its methods to do so.

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

Polymorphism is sharing a common interface for different types. But different types may different implementations. An example is that fruit can be eaten. But some fruit can be eaten in different ways.

In Python we can use polymorphism in inheritance by having the same interface (methods) for many classes, but some classes will do the same thing differently.

From the precode with the vectors we see functional polymorphism. In this case its with multiplication. It has implemented a way for it to multiply itself with a vector and a scalar. However, a scalar(int/float) does not know how to multiply itself with a vector. So the vector has to tell it how to.

Vector * Int

Here the vector knows how to multiply with a integer, so __mul__ is called and everything is fine.

Int * vector

Here __mul__ is called on int, but it won't know how to do the calculation. So python has to ask the vector. __rmul__ is then called on the vector and the multiplication is done.

Since multiplication can be done differently depending on the input we have polymorphism.