1. **Describe the concept of object oriented programming.**

A paradigm designed to simplify code. It breaks up complex pieces of code into modular pieces that are independent and interact with other blocks of code. It’s purpose is to simplify and manage large amounts of code; to keep it all organized.

1. **Describe the difference between an object and a class.**

An object has characteristics; an inherent property that describes them. Think of an object as a cup, or an apple. Each one is an object. Each has different properties, such as does the cup have liquid in it. Is the apple red or green. In software, an object can also just be a well defined idea, such as a bank account.

Objects and classes go hand in hand. In software, a class describes an object. It isn’t the object itself however, it’s just a blueprint; it describes all of its properties/data. It describes its behavior as well in the way of methods.

1. **Define instantiation.**

Instantiation is the process of creating objects through a class. That is, entering information into a class is known as creating an instance of a class; instantiation. Each separate object, is just an instance of a class.

1. **Describe the concept of abstraction and why it is important in object oriented programming.**

Abstraction means we focus on the particular qualities, or properties of something, rather than a specific example. This allows us to build multiple objects, from one main idea. It’s the core idea of object oriented programming. It lays out the blueprint of an idea.

1. **Define encapsulation and what purpose it serves in object oriented programming.**

The idea of surrounding something. It keeps everything self contained. It keeps the private data about a class to itself, and keeps other classes/objects from affecting it. Also known as black boxing. The purpose is to reduce dependencies from different parts of that application. That is, if something needs to be changed, it only must be changed within the class now, not everywhere in the codebase.

1. **Define inheritance and tell why it is beneficial.**

It simplifies code, preventing the need for duplicating a class only to make a small change. It allows a new class that’s like the old one, to just inherit all the previous class’ methods and data. This means that if I make a change to the main class that the other classes are inheriting from, it will automatically apply that change to each of the subclasses.

1. **Define polymorphism. Give an example of why you might want to use it (you may use an example from the video.)**

Polymorphism means many forms. The idea is taking a class and inheriting from a parent class and changing the original methods from the parent class. This defines new behavior in the new class from the original parent. It’s known as overriding.

A good example is overriding the + operator. Maybe you have some base class whose original purpose defines adding two numbers together, along with some other logic. Maybe you want to define a sub class that defines something similar, but instead of adding two numbers, they add two strings. The sub class would override that + operator and change it to adding two strings together. This is the purpose of polymorphism in a nut shell.

In C++, I’ve used it to define a base window class, which controls the concept of a window. From there, I’ve made sub classes which define a window for different platforms, but still use the same methods, just different API’s. This allows functions that accept a pointer to an instance of a base class, call the different methods, but not know the underlying implementation. This might not be a particularly great example, but it is one I’ve used before. Similar things could be done for all sorts of other objects.