1) What is the difference between framework, library and module?

A **library** is *just* a collection of related functionality. Nothing more, but also nothing less. The defining characteristic of a library is that *you* are in control, *you* call the library.

The defining characteristic of a **framework** is *Inversion of Control*. The framework calls *you*, not the other way round. (This is known as the *Hollywood Principle*: "Don't call us, we'll call you.") The framework is in control. The flow of control and the flow of data is managed by the framework.

With a library, *you* write the application, and you leave out the *boring* details, which gets filled in by a *library*.
With a framework, the *framework writer* writes the application, and leaves out the *interesting* details, which *you* fill in.

The defining characteristic of a **module** is *information hiding*. A module has an *interface*, which explicitly, but abstractly specifies both the functionality it provides as well as the functionality it depends on. (Often called the *exported* and *imported* functionality.) This interface has an *implementation* (or multiple implementations, actually), which, from the user of a module are a black box. A module is any file with a .py extension.

- **library**: collection of related functionality
- **framework**: Inversion of Control
- **module**: abstract interface with explicit exports and imports, implementation and interface are separate, there may be multiple implementations and the implementation is hidden

2) Explain IoC with an example.

Inversion of Control (IoC) is a design principle (although, some people refer to it as a pattern). As the name suggests, it is used to invert different kinds of controls in object oriented design to achieve loose coupling. Here, the control means any additional responsibilities a class has other than its main responsibility, such as control over the flow of an application, control over the flow of an object creation or dependent object creation and binding.
IoC is all about inverting the control. To explain in layman's term, suppose you drive a car to your work place, it means you control the car. IoC principle suggests to invert the control, meaning instead of driving the car yourself, you hire a cab where another person will drive the car. Thus it is called inversion of the control from you to the cab driver. You don't have to drive a car yourself and let the driver do the driving so that you can focus on your main work.
IoC principle helps in designing loosely coupled classes which make them testable, maintainable and extensible.

3) What is MVT Architecture? How is it different from MVC?

The Model-View-Template (MVT) is slightly different from MVC. In fact the main difference between the two patterns is that Django itself takes care of the Controller part (Software Code that controls the interactions between the Model and View), leaving us with the template. The template is a HTML file mixed with Django Template Language (DTL).
The developer provides the Model, the view and the template then just maps it to a URL and Django does the magic to serve it to the user.

MVC is popular as it isolates the application logic from the user interface layer and supports separation of concerns. Here the Controller receives all requests for the application and then works with the Model to prepare any data needed by the View. The View then uses the data prepared by the Controller to generate a final presentable response. The MVC abstraction can be graphically represented as follows.

4) What is virtual environment? Why is it needed?

A virtual environment is a tool that helps to keep dependencies required by different projects **separate** by creating isolated **python** virtual environments for them. This is one of the most important tools that most of the **Python** developers use.

8) What is the difference between an app and a project? Create an app named greet and add it into your project greetings.

A project refers to the entire application and all its parts.
An app refers to a submodule of the project. It's self-sufficient and not intertwined with the other apps in the project such that, in theory, you could pick it up and plop it down into another project without any modification. An app typically has its own models.py (which might actually be empty). You might think of it as a standalone python module. A simple project might only have one app.