**Django** follows the **MVC** pattern closely, however it does use its own logic in the implementation. Because the “C” is handled by the framework itself and most of the excitement in **Django** happens in models, templates and views, **Django** is often referred to as an MTV framework.

MVC has been around as a concept for a long time, but has seen exponential growth since the advent of the Internet because it is the best way to design client-server applications. All of the best web frameworks are built around the MVC concept. At the risk of starting a flame war, I contest that if you are not using MVC to design web apps, you are doing it wrong. As a concept,  
the MVC design pattern is really simple to understand:

* The **model(M)** is a model or representation of your data. It’s not the actual data, but an interface to the data. The model allows you to pull data from your database without knowing the intricacies of the underlying database. The model usually also provides an *abstraction* layer with your database, so that you can use the same model with multiple databases.
* The **view(V)** is what you see. It’s the presentation layer for your model. On your computer,  
  the view is what you see in the browser for a Web app, or the UI for a desktop app. The view also provides an interface to collect user input.
* The **controller(C)** controls the flow of information between the model and the view. It uses programmed logic to decide what information is pulled from the database via the model and what information is passed to the view. It also gets information from the user via the view and implements business logic: either by changing the view, or modifying data through the model, or both.

Where it gets difficult is the vastly different interpretations of what actually happens at each layer – different frameworks implement the same functionality in different ways. One framework “guru”  
might say a certain function belongs in a view, while another might vehemently defend the need for it to be in the controller.

You, as a budding programmer who Gets Stuff Done, do not have to care about this because in the end, it *doesn’t matter*. As long as you understand how Django implements the MVC pattern, you are free to move on and get some real work done. Although, watching a flame war in a comment thread can be a highly amusing distraction…

Django follows the MVC pattern closely, however it does use its own logic in the implementation. Because the “C” is handled by the framework itself and most of the excitement in Django happens in models, templates and views, Django is often referred to as an *MTV framework*. In the MTV development pattern:

* **M stands for “Model,”** the data access layer. This layer contains anything and everything about the data: how to access it, how to validate it, which behaviors it has, and the relationships between the data. We will be looking closely at Django’s models in Chapter 4.
* **T stands for “Template,”** the presentation layer. This layer contains presentation-related decisions: how something should be displayed on a Web page or other type of document. We will explore Django’s templates in Chapter 3.
* **V stands for “View,”** the business logic layer. This layer contains the logic that accesses the model and defers to the appropriate template(s). You can think of it as the bridge between models and templates. We will be checking out Django’s views in the next chapter.

This is probably the only unfortunate bit of naming in Django, because Django’s view is more like the controller in MVC, and MVC’s view is actually a Template in Django. It is a little confusing at first, but as a programmer getting a job done, you really won’t care for long. It is only a problem for those of us who have to teach it. Oh, and to the flamers of course.