View logic
The view plays a pivotal role in Django's MVT architecture. On one side, Django's URL dispatcher invokes a corresponding view function that matches the URL pattern.
On the other side, the view interacts with both the model and template layers.

What does the view do?
The primary role of the view function is to fetch the data from the client's request, apply a certain processing logic to it and send an appropriate response back to the client.
It receives the request data in an object of class **`HttpRequest`**.

For simplicity, you can say models are equivalent of a database in Django. The view function interacts with the model in either of two ways. It either fetches all or certain objects from the model such as the database table mapped with the model.
Or the request parameters are used to add a new instance of the model thereby inserting a new row in the mapped table.
The client uses the HTTP GET method to provide the data from the model or delete a certain instance.
On the other hand, it uses the POST method to indicate that the data in the request is to be used to perform an insert or update operation.
While you will learn about models soon, it is good to know the fundamentals. In the upcoming modules of this course, you'll learn how to perform these model operations.
GET and POST methods
Schematically, this behavior is implemented as below:

```
from django.shortcuts import render

def myview(request):

    if request.method=='GET':
        #perform read or delete operation on the model

    if request.method=='POST':
        #perform insert or update operation on the model
```

```
from django.shortcuts import render

def myview(request):
    if request.method=='GET':
        val = request.GET['key']
        #perform read or delete operation on the model
    if request.method=='POST':
        val = request.POST['key']
        #perform insert or update operation on the model
```

At the end of performing any process, you would want to let the user know about the result.

The return value of the view function is a **`HttpResponse`** object containing the actual contents of default content_type as "**`text/HTML`**" and the status code.
Additionally, it contains some header information. However, you would also want the view to give a well-formatted response.
Since the web browser is the client of your web application, the response should be in HTML format as a web page, called a web template.
The Django view loads the template web page, inserts certain context data at the placeholders marked with tags, and returns it as the response.
View rendering template

```
from django.shortcuts import render

def myview(request):

     if request.method=='GET':
          #perform read or delete operation on the model

```

```
    if request.method=='POST':
        #perform insert or update operation on the model
        context={ } #dict containing data to be sent to the client

    return render(request, 'mytemplate.html', context)
```

Class based views
In the above discussion, **`myview`** is a regular Python function.

Such views are called **`function based views`**. The processing logic in it is very imperative in nature, hence it may be repetitive.
Also, it uses conditional blocks for GET and POST requests. Django offers a more concise alternative in the form of a class-based view.
You create a sub-class of the View class and override its **`get()`** and **`post()`** methods to separately and cleanly define GET and POST operations.

```
from django.views import View
class MyView(View):
    def get(self, request):
        # logic to process GET request
        return HttpResponse('response to GET request')

    def post(self, request):
        # <logic to process POST request>
        return HttpResponse('response to POST request')
```

Generic views
Django makes the view declaration process still easier with its generic class-based views. The **`django.views.generic`** module contains several view classes that provide the functionality required to perform tasks such as rendering a template, showing an instance, showing the list of instances, adding a new model instance, updating an instance and so on.
Some generic views are **`TemplateView`**, **`CreateView`**, **`ListView`**, **`DetailView`**, **`UpdateView`** to name a few.

You need to subclass the generic view and set the properties like **`model`** and **`template_name`**. Django will internally perform all the heavy lifting which you had to do by yourself in a function-based view.
You will be working with the class based views in a later module of this course. As this is an introductory course, understanding the fundamentals of building views is important with the function based views. Class based views and Generic views are however widely used and an important topic to be understood as you progress with your journey in web development.