Request and Response Objects
A web application works on the principle of a request-response cycle in a client-server architecture, following the HTTP protocol.
Generally, a browser sends the request in the form of a URL. The web application forms a suitable response to the data contained in the request.
This Reading will provide more detailed information on the Request and Response Objects.

Django handles the request and response with the help of **`HttpRequest`** and **`HttpResponse`** classes in the **`django.http`** module.
Django obtains the **`HttpRequest`** object from the context provided by the server.

As a client request is received, Django's URL dispatcher mechanism invokes a view that matches the URL pattern and passes this **`HTTPRequest`** object as the first argument so that all the request metadata is available to the view for processing.
HttpRequest Object
The request object is characterized by its attributes and methods. They are used extensively in the processing logic of a Django view.
`request.method`

The view logic uses this attribute to identify how the client has approached the server. A browser submits its request using any HTTP methods or verbs – **`POST`**, **`GET`**, **`DELETE`**, and **`PUT`**.
Inside the view function, different conditional blocks may be executed depending on the value of the method attribute. For example:

```
1    if request.method == 'GET':
2        do_something()
3    elif request.method == 'POST':
4        do_something_else()
```

According to the **`REST`**(Representational State Transfer) principle, the **POST** method creates a new resource on the server.
To fetch one or more resources from the server, the **GET** method is used. Similarly, the **PUT** method is for updating an existing resource, and the **DELETE** method is used to remove a resource from the server.
`request.GET and request.POST`

The attributes return a dictionary-like object containing GET and POST parameters, respectively.

`request.COOKIES`

Along with the parameters, the browser also packs the request objects with cookies inserted by the server's previous interactions. It is a dictionary of string keys and values.
`request.FILES`

When the user uploads one or more files with a multipart form, they are present in this attribute in the form of UploadedFile objects. By appropriate logic in the view, these uploaded files are saved in the designated folder on the server.
`request.user`

The request object also contains information about the current user. This attribute is an object of **`django.contrib.auth.models.User`** class. However, if the user is unauthenticated, it returns **`AnonymousUser`**. Inside the view, you can lay down separated separate logic for either of them.

```
1    if request.user.is_authenticated():
2        # Do something for logged-in users.
3    else:
4        # Do something for anonymous users.
```

`request.has_key()`

This is a method available to the request object. It helps check whether the **`GET`** or **`POST`** parameter dictionary has a value for the given key.
Unlike the **`HttpRequest`** object, which is supplied by the server's context, the response object of **`HttpResponse`** class is instantiated inside the view function before it is returned to the client. For example:

```
1    from django.http import HttpResponse
2    def index(request):
3        return HttpResponse("Hello World")
```

Although it is possible to render a hardcoded HTML string as the response, Django offers a better alternative to render a template web page.

```
from django.http import HttpResponse
from django.template import loader
def index(request):
    template = loader.get_template('demoapp/index.html')
    context={}
    return HttpResponse(template.render(context, request))
```

You can pack additional headers or cookies in the response object.

HttpResponseObject
Some of the main attributes and methods of the **`HttpResponse`** object are:

`status_code`: returns the HTTP status code corresponding to the response

`content`: returns the byte string of the response.

`__getitem__()`: method that returns the value of a header

`__setitem__()`: method used to add a header

`write()`: This method creates a file-like object.

The following example demonstrates the attributes of the request and response objects. Add the following view function in **`views.py`** of the Django app.

```
    return HttpResponse(content)
```

Start the server and check the response for http://localhost:8000/demo/ as the URL.

In this Reading, you explored the properties of the **`HttpRequest`** and **`HttpResponse`** objects.