1. Django basics and understanding the rules.

   Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. It follows the Model-View-Controller (MVC) architectural pattern, although in Django's case, it's slightly different: it uses a "Model-View-Template" (MVT) design pattern. Here are the basics and key concepts to understand about Django:

   MVT:(Model View Template)

   Model: The Model represents the data structure. It is responsible for handling the database logic and storing/retrieving data. Django provides an Object-Relational Mapping (ORM) layer that allows you to define your models using Python classes, which are then translated into database tables.

   View: The View contains the logic that accesses the model's data and handles user requests. It controls what data is presented to the user and interacts with templates to render the final output. Views are Python functions or classes that process incoming HTTP requests and return HTTP responses.

   Templates: The Template represents the presentation layer. It contains HTML files with placeholders (using Django's template language) for dynamic content. Templates are used to generate the user interface that is sent back to the client.

   Rules:

1. **DRY (Don't Repeat Yourself)**: Django encourages writing clean, reusable code by avoiding repetition. Use Django's features like template inheritance, mixins, and reusable components to keep your codebase concise.
2. **Convention over Configuration**: Django follows a "batteries-included" philosophy, providing sensible defaults and conventions. Follow these conventions to speed up development and make the code more maintainable.
3. **Security**: Django has built-in security features to prevent common web vulnerabilities like SQL injection, CSRF attacks, clickjacking, etc. Always use Django's security features and best practices to safeguard your application.
4. **Documentation and Community**: Django has excellent documentation and a strong community. Utilize these resources for learning, troubleshooting, and staying updated with best practices.
5. **Testing**: Writing tests is a crucial aspect of Django development. Django provides a robust testing framework to ensure the correctness of your code.

2. HTTP request and responses

HTTP (Hypertext Transfer Protocol) is the underlying protocol used for communication on the World Wide Web. HTTP requests and responses are fundamental concepts in web development:

HTTP Requests:

1. **GET Request**: This request method is used to retrieve data from a specified resource. It sends data in the URL, and it's visible in the browser's address bar. GET requests are generally used for fetching data and should not have any side effects on the server.
2. **POST Request**: The POST method is used to send data to the server to create or update a resource. It sends data in the body of the HTTP request. POST requests are commonly used when submitting forms, uploading files, or performing any action that modifies data on the server.
3. **Other Request Methods**: HTTP also supports other request methods like PUT, DELETE, PATCH, etc., each serving different purposes for interacting with resources on the web server.

HTTP Responses:

1. **Status Codes**: After receiving an HTTP request, the server responds with an HTTP status code indicating the outcome of the request. Examples include:
    - 200 OK: Request succeeded.
    - 404 Not Found: The requested resource does not exist on the server.
    - 500 Internal Server Error: Indicates a server-side error.
    - And many more status codes indicating different scenarios.
2. **Response Body**: The response might also include a body, containing the requested resource (in the case of successful requests) or error details.

3.Setting urls in Django.

In Django, URL patterns are defined using the **urls.py** file within each Django app. The **urls.py** file contains a list of URL patterns that map specific URLs to corresponding views or other URL patterns.

Basic setting:

Inside the created app directory (**myapp** in this case), create or modify the **urls.py** file.

```
from django.urls import path
from . import views

urlpatterns = [
   path('', views.home, name='home'),  # Mapping root URL to the 'home' view
   path('about/', views.about, name='about'),
```

Create Views for the Defined URLs:

Define the corresponding views (functions or classes) in **views.py** within the same app.

```
from django.http import HttpResponse


def home(request):
   return HttpResponse("This is the home page")


def about(request):
   return HttpResponse("This is the about page")
```

Include App URLs in Project URLs:

To use these app-specific URLs, include them in the project's main **urls.py** file.

```
from django.contrib import admin
from django.urls import include, path


urlpatterns = [
   path('admin/', admin.site.urls),
   path('', include('myapp.urls')),  # Including the URLs defined in 'myapp'
]
```