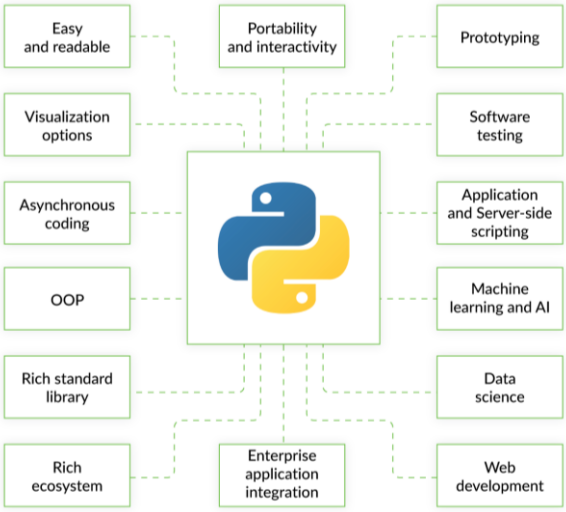
WEB DEVELOPMENT IN PYTHON

Web Development



Web development is the umbrella term for conceptualizing, creating, deploying and operating web applications and application programming interfaces for the Web.

The Web has grown a mindboggling amount in the number of sites, users and implementation capabilities since the first website went live in 1989. Web development is the concept that encompasses all the activities involved with websites and web applications.

How does Python fit into web development?

Python can be used to build server-side web applications. While a web framework is not required to build web apps, it's rare that developers would not use existing open source libraries to speed up their progress in getting their application working.

Python is not used in a web browser. The language executed in browsers such as Chrome, Firefox and Internet Explorer is JavaScript. Projects such as pyjs can compile from Python to JavaScript. However, most Python developers write their web applications using a combination of Python and JavaScript. Python is executed on the server side while JavaScript is downloaded to the client and run by the web browser.

Web Development Using Python Frameworks

Django Pyramid

TurboGears Flask

Another good thing about Python is that it has many frameworks that simplify the development process. Depending on what you're doing, you may need different frameworks.

Let's take a look at the most well-known Python frameworks.

- **Django**

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

This framework is great for fully-fledged web apps and mid-range scalable projects. It has built-in features and allows for code re-usage, coherent modification of different components of the code, and other functionality that simplifies web development. Django works well with Oracle SQL, PostgreSQL, MySQL, and other well-known databases.

- **Pyramid**

With this framework, you can start small and scale if needed. Pyramid can be used with various databases and applications or extended with plugins — developers can add whatever functionality they need. That's handy when you need to implement various solutions in one task.

- **TurboGears**

TurboGears consists of several components such as Repoze, WebOb, and Genshi, and is based on the MVC architecture. It's good for fast and efficient web application development, which is also more maintainable. With this framework, you can write small or complex applications using minimal or full-stack modes respectively.

- **Flask**

This framework's philosophy is to provide a simple and manageable solution that can be easily customized. Flask defines itself as a microframework and is most commonly applied to small solutions whose main priority is lean functionality. The framework is also used for creating prototypes.

Django

Django is a Web framework written in Python. A Web framework is a software that supports the development of dynamic Web sites, applications, and services. It provides a set of tools and functionalities that solves many common problems associated with Web development, such as security features, database access, sessions, template processing, URL routing, internationalization, localization, and much more.

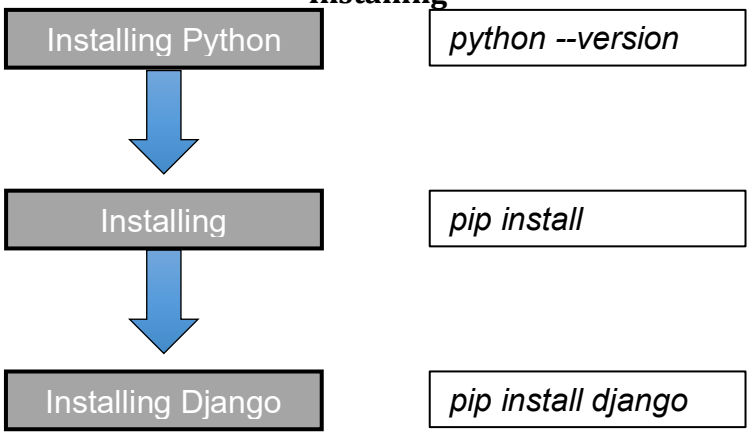
Django is a widely-used Python web application framework with a "batteries-included" philosophy. The principle behind batteries-included is that the common functionality for building web applications should come with the framework instead of as separate libraries.

For example, authentication, URL routing, a template engine, an object-relational mapper (ORM), and database schema migrations are all included with the Django framework. Compare that included functionality to the Flask framework which requires a separate library such as Flask-Login to perform user authentication.

The batteries-included and extensibility philosophies are simply two different ways to tackle framework building. Neither philosophy is inherently better than the other one.

Because of the Django's unique strength, there are multiple popular websites which are built with Python on top of the Django framework. There are some of the major sites which are fully or partially built based on Django.

Installing



The first thing we need to do is install some programs on our machine so to be able to start playing with Django. The basic setup consists of installing **Python, Virtualenv, and Django**.

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Using virtual environments is not mandatory, but it's highly recommended. If you are just getting started, it's better to start with the right foot.

When developing Web sites or Web projects with Django, it's very common to have to install external libraries to support the development. Using virtual environments, each project you develop will have its isolated environment. So the dependencies won't clash. It also allows you to maintain in your local machine projects that run on different Django versions.

The first thing we should install the latest **Python** distribution.

For the next step, we are going to use **pip**, a tool to manage and install Python packages, to install **virtual environment**.

The last step is to install **Django**.

Projects and Apps

```
myproject/                <-- higher level folder
 |-- myproject/           <-- django project folder
 |    |-- myproject/
 |    |    |-- __init__.py
 |    |    |-- settings.py
 |    |    |-- urls.py
 |    |    |-- wsgi.py
 |    +-- manage.py
 +-- venv/                <-- virtual environment folder
```

Filename	Description/Purpose
__init__.py	Specifies to Python that this is a package
urls.py	Global URL configuration ("URLconf")
settings.py	Project-specific configuration
manage.py	Command-line interface for applications

In the Django philosophy we have two important concepts:

app: is a Web application that does something. An app usually is composed of a set of models (database tables), views, templates, tests.

project: is a collection of configurations and apps. One project can be composed of multiple apps, or a single app.

It's important to note that you can't run a Django app without a project. Simple websites like a blog can be written entirely inside a single app, which could be named **blog** or **weblog** for example.

Our initial project structure is composed of five files:

- **manage.py**: a shortcut to use the django-admin command-line utility. It's used to run management commands related to our project. We will use it to run the development server, run tests, create migrations and much more.
- **__init__.py**: this empty file tells Python that this folder is a Python package.
- **settings.py**: this file contains all the project's configuration. We will refer to this file all the time!
- **urls.py**: this file is responsible for mapping the routes and paths in our project. For example, if you want to show something in the URL /about/, you have to map it here first.
- **wsgi.py**: this file is a simple gateway interface used for deployment. You don't have to bother about it. Just let it be for now.

Django Apps

manage.py

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

django-admin startapp boards

```
myproject/
 |-- myproject/
 |    |-- boards/              <-- our new django app!
 |    |    |-- migrations/
 |    |    |    +-- __init__.py
 |    |    |-- __init__.py
 |    |    |-- admin.py
 |    |    |-- apps.py
 |    |    |-- models.py
 |    |    |-- tests.py
 |    |    +-- views.py
 |    |-- myproject/
 |    |    |-- __init__.py
 |    |    |-- settings.py
 |    |    |-- urls.py
 |    |    |-- wsgi.py
 |    +-- manage.py
 +-- venv/
```

To create our first app, go to the directory where the manage.py file is and executes the following command: ***django-admin startapp boards***.

Notice that we used the command **startapp** this time.

This will give us the following directory structure as in the picture.

So, let's first explore what each file does:

• **migrations/**: here Django store some files to keep track of the changes you create in the models.py file, so to keep the database and the models.py synchronized.

• **admin.py**: this is a configuration file for a built-in Django app called Django Admin.

• **apps.py**: this is a configuration file of the app itself.

• **models.py**: here is where we define the entities of our Web application. The models are translated automatically by Django into database tables.

• **tests.py**: this file is used to write unit tests for the app.

• **views.py**: this is the file where we handle the request/response cycle of our Web application.

settings.py

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

add boards

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'boards',
]
```

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Now that we created our first app, let's configure our project to use it.

To do that, open the **settings.py** and try to find the **INSTALLED_APPS** variable.

As you can see, Django already come with 6 built-in apps installed. They offer common functionalities that most Web applications need, like authentication, sessions, static files management (images, javascripts, css, etc.) and so on.

Hello, World!

views.py

```
from django.http import HttpResponse

def home(request):
    return HttpResponse('Hello, World!')
```

urls.py

```
from django.conf.urls import url
from django.contrib import admin
from boards import views

urlpatterns = [
    url(r'^$', views.home, name='home'),
    url(r'^admin/', admin.site.urls),
]
```

Open the **views.py** file inside the **boards** app, and add the code as in the picture.

Views are Python functions that receive an **HttpRequest** object and returns an **HttpResponse** object. Receive a **request** as a parameter and returns a **response** as a result.

So, here we defined a simple view called **home** which simply returns a message saying **Hello, World!**.

Now we have to tell Django ***when*** to serve this view. It's done inside the **urls.py** file.

Hello, World!

In a Web browser, open the **URL** as in the picture.

Congratulations! You've just created your first website and run it using a web server!

References

1. Link: [https://www.fullstackpython.com/web-development.html]
2. Link: [https://djangostars.com/blog/python-web-development/]

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

3. A Complete Beginner's Guide to Django by Vitor Freitas. Link: [https://simpleisbetterthancomplex.com/series/2017/09/04/a-complete-beginners-guide-to-django-part-1.html]
4. Link:[https://www.tutorialspoint.com/python_web_development_libraries/python_web_development_libraries_django_framework.htm]
5. Core Python Applications programming (Third edition) by Wesley J.Chun

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.