Python Fullstack

WEB DEVELOPMENT WITH PYTHON AND DJANGO

Vikash Patel

Part 0

The Overview

Introduction:

- What is full-stack development?
- Why Python and Django?
- Course overview and learning objectives.

What is Fullstack?

- Frontend + Backend = Fullstack
- Builds both the user-facing interface and the behind-the-scenes logic of web applications.

Why Python?

- Readable and beginner-friendly syntax.
- Extensive standard library and third-party libraries.
- Cross-platform compatibility - works on Windows, Mac, and Linux.
- Large and supportive developer community for assistance.

Why Django?

- Batteries-included.
- Model-View-Controller (MVC) architecture for organized code.
- Secure by design
- Scalable to handle complex web applications as they grow.
- Large community and extensive documentation for support.

Course Overview:

- Understand the fundamentals of full-stack development.
- Gain an introduction to Python programming.
- Explore key concepts of the Django web framework.
- Build a basic web application using Python and Django.

Learning Objectives:

- Explain the concept of full-stack development.
- Be able to write Python code.
- Understand core functionalities of Django.
- Construct a simple web application using Python and Django.
- Understand concepts related to deployment and monitoring

Introduction to Python

- Simple Syntax: just like reading english
- Interpreted Language: line by line execution
- Multipurpose Language: web development, data science, machine learning, desktop development
- Dynamic Typing: don’t need to create the variables with a specific type (e.g.: *int number*)
- Multi-platform: windows, linux, macos, unix, and arm, x86(32-bit), x86_64 (64-bit).
- It runs almost everywhere
- Large Community and Ecosystem

Variables, Operators, and Expressions

- Variables: easy to create variables `name = “Python”`
- Operators: (+, -, *, /), comparison (==, !=, <, >), logical (and, or, not), and assignment (=).
- Expressions: x+y, a=b-c etc.
- Data Types: Integer, Float, String, Boolean
- Data Structures: DIct (dictionary/hashmap/map), List, Set, Tuple, Class, Object

Control Flow Statements

1. **Branching (if, else, elif)**:
   - **If Statement**: execution of code based on a specified condition.
   - **Else Statement**: alternative execution path when the condition in the if statement is not met.
   - **Elif Statement**: Short for "else if," allows for checking multiple conditions sequentially after the initial if statement. (also known as if-else ladder)
2. **Loops (for, while)**:
   - **For Loop**: Iterates over a sequence (such as a list, tuple, or string) or an iterable object, executing a block of code for each item in the sequence.
   - **While Loop**: Repeatedly executes a block of code as long as a specified condition is true, allowing for flexible looping based on changing conditions.

Functions and Modules

- Function: block of reusable code
- Module: collection of many function (basically a file having many function)
- Both of them are used to organize our code.

```
def greet(name):

     message = "Hello, " + name

     return message
```

Hands-on Coding Exercise

- Write a program to calculate area of a rectangle.

```
# Function to calculate the area of a rectangle
def calculate_area(length, width):
    return length * width

# Input length and width from the user
length = float(input("Enter the length of the rectangle: "))
width = float(input("Enter the width of the rectangle: "))

# Calculate the area
area = calculate_area(length, width)

# Display the result
print("The area of the rectangle is:", area)
```

Part 1

Django Web Framework: The web development

Introduction to Django (MVT architecture)

- **MVT: means Model-View-Templating**
- **Model**: In Django, the model layer represents the data structure of the application.Each model class typically represents a specific database table. And to manage it we have ORM in django.
- **View**: Views in Django are responsible for processing user requests and returning appropriate responses. They act as the bridge between the model and the template layers.
- **Template**: Templates are used for generating HTML dynamically and rendering the data obtained from views.These are written in HTML and we can use django templating engine (like Jinja, Pug, Ejs) to create dynamic templates.

Setting up a Django Project

- `pip install django`
- `django-admin startproject <project-name>`
- `cd <project-name>`
- `python manage.py startapp <app-name>`
- `python manage.py makemigrations`
- `python manage.py migrate`
- `python manage.py collectstatic`
- `python manage.py createsuperuser`
- `python manage.py runserver`

Creating Models (defining data structure)

```
# biolinks/models.py
from django.db import models
class BioLink(models.Model):
    name = models.CharField(max_length=100)
    link = models.URLField()
    owner = models.ForeignKey('users.CustomUser',
related_name='bio_links', on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    def __str__(self):
        return f'{self.name}({self.link})'
```

Views (handling user requests)

```python
# biolinks/views.py

from biolinks.models import BioLink

def bio_links_view(request):

    bio_links = BioLink.objects.all()

    context = {'bio_links': bio_links}

    return render(request, 'biolinks/index.html', context)
```

Templates (presenting data with HTML)

```
<!-- templates/biolinks/index.html -->

….

{% for bio_link in bio_links %}
  <li>
      <a href="{{ bio_link.link }}" target="_blank">{{ bio_link.name 
}}</a>
  </li>
{% endfor %}


…
```

URL Routing (mapping URLs to views)

```
# biolinks/urls.py

urlpatterns = [

    path('', bio_links_view, name='Bio Links home'),

]


# core/urls.py (your configuration application)

urlpatterns = [

    path('', include('biolinks.urls')),

]
```

Part 2: Building a Simple Web App

- Project Idea and Planning
- Implementing Models (e.g., biolinks model)
- Creating Views (e.g., displaying links)
- Designing Templates (HTML with Django template tags)
- Running the Application and Testing

Part 3: Conclusion and Next Steps

Recap of key concepts learned.

Resources for further learning (Django documentation, tutorials)

Q&A session.

Get the code from here (GitHub repo)

https://github.com/vk4s/advanced-django-features-demo

Thank you