Django Training with Digital Pathshala – Syllabus by Ishan Kafle

Cover page

**Table of Contents**

[1. Week 1: Python Basics (Days 1–6) 1](#_Toc190290031)

[1.1. Day 1: Python Fundamentals 1](#_Toc190290032)

[1.1.1. Variables 1](#_Toc190290033)

[1.1.2. Data Types in Python 1](#_Toc190290034)

[1.1.3. Basic Input/Output 1](#_Toc190290035)

[1.2. Day 2: Control Flow 1](#_Toc190290036)

[1.2.1. Control Flow in Python 2](#_Toc190290037)

[1.2.2. Conditional Statements (if, elif, else) 2](#_Toc190290038)

[1.2.3. Loops: for and while 2](#_Toc190290039)

[1.3. Day 3: Data Structures 3](#_Toc190290040)

[1.3.1. . Lists (list) – Ordered & Mutable 3](#_Toc190290041)

[1.3.2. Tuples (Ordered, Immutable Collection) 4](#_Toc190290042)

[1.3.3. Dictionaries (Key-Value Pairs) 4](#_Toc190290043)

[1.3.4. Sets in Python 4](#_Toc190290044)

[1.3.5. Basic List (Array) Operations 4](#_Toc190290045)

[1.3.6. Basic Dictionary Operations 5](#_Toc190290046)

[1.3.7. Basic Set Operations 5](#_Toc190290047)

[1.4. Day 4: Functions 6](#_Toc190290048)

[1.4.1. Introduction to Functions 6](#_Toc190290049)

[1.4.2. Parameters and Arguments 7](#_Toc190290050)

[1.4.3. Return Values 7](#_Toc190290051)

[1.4.4. Default Arguments 7](#_Toc190290052)

[1.5. Day 5: OOP Basics 8](#_Toc190290053)

[1.6. Day 6: Error Handling and File Handling 8](#_Toc190290054)

[2. Week 2: Django Basics (Days 7–12) 8](#_Toc190290055)

[2.1. Day 7: Introduction to Django 8](#_Toc190290056)

[2.2. Day 8: Django Project Structure 8](#_Toc190290057)

[2.3. Day 9: URLs and Views 8](#_Toc190290058)

[2.4. Day 10: Templates 8](#_Toc190290059)

[2.5. Day 11: Models and Migrations 9](#_Toc190290060)

[2.6. Day 12: Querying the Database 9](#_Toc190290061)

[3. Week 3: Django REST Framework Basics (Days 13–18) 9](#_Toc190290062)

[3.1. Day 13: Introduction to REST APIs and DRF 9](#_Toc190290063)

[3.2. Day 14: Serializers 9](#_Toc190290064)

[3.3. Day 15: Function-Based Views (FBVs) 9](#_Toc190290065)

[3.4. Day 16: Routers and URLs 9](#_Toc190290066)

[3.5. Day 17: Query Parameters 9](#_Toc190290067)

[3.6. Day 18: Class-Based Views (CBVs) 9](#_Toc190290068)

[4. Week 4: Advanced DRF (Days 19–24) 10](#_Toc190290069)

[4.1. Day 19: Viewsets 10](#_Toc190290070)

[4.2. Day 20: Authentication 10](#_Toc190290071)

[4.3. Day 21: Permissions 10](#_Toc190290072)

[4.4. Day 22: Relationships and Nested Serializers 10](#_Toc190290073)

[4.5. Day 23: Pagination 10](#_Toc190290074)

[4.6. Day 24: File Uploads 10](#_Toc190290075)

[5. Week 5: API Optimization and Deployment (Days 25–30) 10](#_Toc190290076)

[5.1. Day 25: Throttling and Rate Limiting 10](#_Toc190290077)

[5.2. Day 26: Middleware and Signals 10](#_Toc190290078)

[5.3. Day 27: Testing APIs 11](#_Toc190290079)

[5.4. Day 28: Deployment Basics 11](#_Toc190290080)

[5.5. Day 29: Capstone Project - Setup 11](#_Toc190290081)

[5.6. Day 30: Capstone Project - Build & Deploy 11](#_Toc190290082)

[6. Deploy the project and review 11](#_Toc190290083)

# Week 1: Python Basics (Days 1–6)

## Day 1: Python Fundamentals

* Introduction to Python and its uses.
* Setting up Python and IDE.
* Variables, data types, and basic input/output.

### Variables

A variable is a container that stores values.

Example**: A Water Bottle 💧**

A bottle can hold water, juice, or soda (just like a variable holds different data types).

You can change its contents, just like variables change values.

### Data Types in Python

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Real-World Example** | **Python Example** |
| **String (str)** | Name tag | name = "Alice" |
| **Integer (int)** | Number of apples | age = 25 |
| **Float (float)** | Weight in kg | height = 5.9 |
| **Boolean (bool)** | Light switch (On/Off) | is\_student = True |

### Basic Input/Output

* **input()** → Takes user input.
* **print()** → Displays output.

## Day 2: Control Flow

* Conditional statements (if, else, elif).
* Loops: for and while.

### Control Flow in Python

**What is Control Flow?**

In real life, we make decisions based on conditions (e.g., "If it rains, take an umbrella; otherwise, wear sunglasses.").

Similarly, Control Flow in Python allows programs to make decisions and repeat tasks using:

* **Conditional Statements** (if, elif, else)
* **Loops** (for, while)

### Conditional Statements (if, elif, else)

**What are Conditional Statements?**

Conditional statements allow a program to **execute different code blocks based on conditions** (True/False).

Example: If it rains, I will take an umbrella. Else, I will go without it.

Let’s do it in python:

**Real-World Example: Traffic Light**

* Red = Stop
* Yellow = Slow Down
* Green = Go

**if** → First condition check.  
**elif** → Additional condition checks.  
**else** → Runs when no conditions match.

### Loops: for and while

Loops repeat a block of code multiple times, reducing redundancy.

**The for Loop**

* Used when we know how many times, we need to repeat something.
* The for loop is best when we have a fixed number of repetitions.
* It iterates over a sequence like a **list, tuple, or range**.

**Example 1**: Printing Numbers 1-5 used **for** loop

**Example 2**: Printing Each Character in a String

**The while Loop** (Looping Until a Condition is False)

* Used when we don’t know how many times the loop will run.
* A while loop runs **until a condition becomes False**

**Example 1**: Printing Numbers 1-5 Using while Loop

Both for and while loops are used for iteration, but they have **different use cases**.

Difference:

|  |  |  |
| --- | --- | --- |
| **Feature** | **for Loop** | **while Loop** |
| **Usage** | Used when the number of iterations is **known** beforehand. | Used when the number of iterations is **unknown** and depends on a condition. |
| **Syntax** | Iterates over a sequence (range, list, tuple, string, etc.). | Runs as long as a condition is True. |
| **Best for** | **Definite iteration** (fixed number of repetitions). | **Indefinite iteration** (unknown number of repetitions). |
| **May lead to infinite loop?** | No, because it iterates over a sequence with a defined end. | Yes, if the condition is never False. |
| **Example Use Case** | Iterating through a list, printing numbers from 1 to 10. | Asking the user for input until they enter the correct value. |

## Day 3: Data Structures

* Lists, tuples, and dictionaries.

### . Lists (list) – Ordered & Mutable

A list is an ordered collection of items that can be modified (mutable).

* **Ordered** – Elements are stored in a specific order.
* **Mutable** – Can be changed (elements can be added, removed, modified).
* **Allows Duplicates** – Can have repeated elements.
* **Supports Different Data Types** – A list can store integers, strings, and even other lists.

### Tuples (Ordered, Immutable Collection)

A tuple is an ordered collection like a list but immutable (cannot be changed after creation). Tuples are faster than lists and are used when data remain unchanged.

**Key Features of Tuples**

* **Ordered** – Elements have a fixed sequence.
* **Immutable** – Cannot be modified after creation.
* **Allows Duplicates** – Can store repeated values.
* **Supports Different Data Types** – Can hold various data types.

### Dictionaries (Key-Value Pairs)

A dictionary is an unordered collection that stores data as key-value pairs. Unlike lists and tuples, which use indexing, dictionaries allow data retrieval using keys.

**Key Features of Dictionaries**

* **Ordered (Python 3.7+)** – Elements have a fixed sequence.
* **Mutable** – Can be changed (add, update, or remove key-value pairs).
* **Key-Value Pairs** – Each value is accessed using a unique key.
* **Keys Must Be Unique** – No duplicate keys allowed.
* Basic operations: adding, removing, iterating.

### Sets in Python

A set is an unordered collection of unique elements. Unlike lists and tuples, sets do not allow duplicate values and are optimized for fast membership checks (whether an item is present or not).

**Key Features of Sets**

* **Unordered** – The elements are not stored in a specific order.
* **Unique Elements** – No duplicate values are allowed.
* **Mutable** – You can add and remove elements, but individual items cannot be modified.
* **Fast Membership Testing** – Checking if an item exists in a set is faster than in lists.

### Basic List (Array) Operations

|  |  |  |
| --- | --- | --- |
| **Operation** | **Example** | **Output** |
| Access Element | fruits[1] | "banana" |
| Modify Element | fruits[1] = "mango" | ["apple", "mango", "cherry"] |
| Add Element | fruits.append("orange") | ["apple", "banana", "cherry", "orange"] |
| Remove Element | fruits.remove("banana") | ["apple", "cherry"] |
| Iterate Over List | for fruit in fruits: print(fruit) | Prints all items |

### Basic Dictionary Operations

|  |  |  |
| --- | --- | --- |
| **Operation** | **Example** | **Output** |
| Access Value | student["name"] | "John" |
| Modify Value | student["age"] = 21 | {"name": "John", "age": 21, "grade": "A"} |
| Add Key-Value | student["city"] = "Kathmandu" | Adds "city": "Kathmandu" |
| Remove Key | del student["grade"] | Removes "grade" |
| Iterate Over Dict | for key, value in student.items(): print(key, value) |  |

### Basic Set Operations

|  |  |  |
| --- | --- | --- |
| **Operation** | **Example** | **Output** |
| Add an element | fruits.add("orange") | {"apple", "banana", "cherry", "orange"} |
| Remove an element | fruits.remove("banana") | {"apple", "cherry"} |
| Check if an element exists | "apple" in fruits | TRUE |
| Get set length | len(fruits) | 3 |
| Iterate through a set | for item in fruits: print(item) | Prints each item |
| Convert a list to a set (Remove Duplicates) | set([1, 2, 2, 3, 4, 4, 5]) | {1, 2, 3, 4, 5} |

## Day 4: Functions

* Defining and calling functions.
* Parameters, return values, and default arguments.

### Introduction to Functions

A function is a reusable block of code that performs a specific task.

**Why use functions?**

* Avoids code repetition (DRY principle).
* Improves readability and maintainability.
* Make debugging easier.

**Syntax of a Function**

A screen shot of a computer program

AI-generated content may be incorrect.

**Example**

A black background with white text

AI-generated content may be incorrect.

### Parameters and Arguments

* Parameters are variables in function definitions.
* Arguments are actual values passed to functions.

A screen shot of a computer code

AI-generated content may be incorrect.

### Return Values

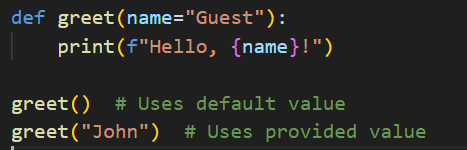
* A function can return a value using return.

A screenshot of a computer program

AI-generated content may be incorrect.

### Default Arguments

If no argument is provided, default values are used.

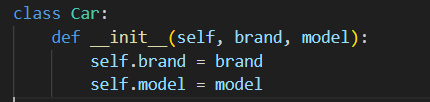


## Day 5: OOP Basics

* Classes, objects, and attributes.
* Methods and constructors.

### Class

A **class** is a blueprint for creating objects. It defines attributes (variables) and behaviors (methods).



### Object

An object is an **instance** of a class. When we create an object, we call the class as a function. The object holds data (attributes) and can access methods from the class.

A computer screen shot of text

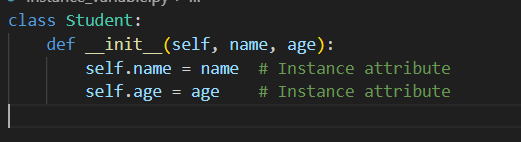
AI-generated content may be incorrect.

### Attributes:

Attributes are the variables that store the state of an object. Attributes can be instance attributes (unique to each object) or class attributes (shared among all objects of the class).

* **Instance Attributes:**

These are defined inside the **\_\_init\_\_** constructor method using self.



* **Class Attributes:**

These are defined **outside** any method but inside the class.

A black background with text

AI-generated content may be incorrect.

### Methods:

* **Instance Methods:**

Instance methods are **functions** defined inside the class that operate on the instance of the class. They typically have self as the first parameter, which represents the object.

A screen shot of a computer program

AI-generated content may be incorrect.

**Calling the Instance Method**

A screen shot of a computer

AI-generated content may be incorrect.

### Constructor (\_\_init\_\_ Method)

The constructor is a special method \_\_init\_\_() that is automatically called when an object is created. It is used to initialize the instance attributes with initial values.

A screenshot of a computer code

AI-generated content may be incorrect.

* **Calling the Constructor:**

A computer screen shot of a program

AI-generated content may be incorrect.

### Summary of the day:

**Class** = Blueprint,

**Object** = Building from the Blueprint,

**Attributes** = Characteristics of the Building (e.g., number of rooms),

**Methods** = Actions the Building can Perform (e.g., open doors).

### Inheritance

Inheritance is a fundamental concept in Object-Oriented Programming (OOP) where a class can inherit attributes and methods from another class. The class inheriting the properties is called the child class, and the class it is inheriting from is called the parent class.

**Parent Class:**

A parent class (also called the base class or superclass) defines attributes and methods that are inherited by the child class.

A computer screen shot of a computer code

AI-generated content may be incorrect.

**Child Class**

A child class inherits the properties and methods of the parent class. The child class can also have its own methods and attributes, or it can override methods from the parent class.

A computer screen shot of a program code

AI-generated content may be incorrect.

## Day 6: Error Handling and File Handling

* Exception handling (try, except).
* Reading and writing files.
* Introduction to pip and installation

### ****What Are Exceptions?****

Exceptions are runtime errors that occur during the execution of a program. When Python encounters an error that disrupts the normal flow of the program, it raises an exception. An exception is a signal that something unexpected happened, and it interrupts the normal operation of the program.

In Python, exceptions are objects that represent errors, and these errors can be caught and handled using try-except blocks to prevent the program from crashing unexpectedly.

A screen shot of a computer

AI-generated content may be incorrect.

* The code tries to divide x by y, but since y = 0, it raises a ZeroDivisionError.
* The except block catches this specific exception and prints the error message.

**Common Python Exceptions**:

* **ZeroDivisionError**: Raised when a number is divided by zero.
* **ValueError**: Raised when a function receives an argument of the correct type but an invalid value.
* **IndexError**: Raised when an invalid index is used to access a list or tuple.
* **KeyError**: Raised when trying to access a dictionary key that doesn't exist.
* **FileNotFoundError**: Raised when trying to access a file that does not exist.

### Handling Exceptions:

You can handle exceptions using the try-except block. This lets you attempt to run code and catch any exceptions that arise, allowing your program to recover from unexpected errors.

A computer screen shot of a program code

AI-generated content may be incorrect.

* **try block**: Code that may raise an exception.
* **except block**: Code that handles the exception.
* **Exception as e**: This catches any type of exception, and e holds the exception object with details.

**File Handling**

### Reading a File (r mode)

A computer screen shot of text

AI-generated content may be incorrect.

### Writing to a File (w mode)

### Appending to a File (a mode)

### Reading & Writing (r+ mode)

### Writing & Reading (w+ mode)

### Deleting a File (os.remove)

Reads the entire file content.

# Week 2: Django Basics (Days 7–12)

## Day 7: Introduction to Django

* What is Django, and why use it?
* Setting up a Django project and virtual environment.

## Day 8: Django Project Structure

* Understanding project and app structure.
* Creating and running a Django app.

## Day 9: URLs and Views

* Configuring URLs and writing views.
* Returning HTTP responses.

## Day 10: Templates

* Rendering HTML templates.
* Template syntax basics (variables, filters, loops).

## Day 11: Models and Migrations

* Defining models and creating migrations.
* Exploring the Django Admin panel.

## Day 12: Querying the Database

* Using Django ORM for basic CRUD operations.
* Querysets and filtering data.

# Week 3: Django REST Framework Basics (Days 13–18)

## Day 13: Introduction to REST APIs and DRF

* What are REST APIs?
* Setting up Django REST Framework.
* Creating your first API endpoint.

## Day 14: Serializers

* Introduction to DRF serializers.
* Writing basic serializers for models.

## Day 15: Function-Based Views (FBVs)

* Writing GET, POST, PUT, and DELETE APIs using FBVs.

## Day 16: Routers and URLs

* Configuring API URLs with DRF routers.

## Day 17: Query Parameters

* Handling query parameters in APIs.
* Filtering, searching, and ordering data.

## Day 18: Class-Based Views (CBVs)

* Writing APIs using CBVs.
* Difference between FBVs and CBVs.

# Week 4: Advanced DRF (Days 19–24)

## Day 19: Viewsets

* Simplifying APIs with ModelViewSet.

## Day 20: Authentication

* Implementing basic and token authentication.
* Securing endpoints.

## Day 21: Permissions

* Applying built-in permissions (AllowAny, IsAuthenticated, etc.).
* Creating custom permissions.

## Day 22: Relationships and Nested Serializers

* Serializing related fields (ForeignKey, ManyToMany).
* Writing nested serializers.

## Day 23: Pagination

* Adding pagination to APIs.
* Using PageNumberPagination and LimitOffsetPagination.

## Day 24: File Uploads

* Handling file and image uploads in APIs.

# Week 5: API Optimization and Deployment (Days 25–30)

## Day 25: Throttling and Rate Limiting

* Adding throttling to APIs.
* Customizing throttling rules.

## Day 26: Middleware and Signals

* Writing custom middleware.
* Using signals for real-time actions (e.g., email on user creation).

## Day 27: Testing APIs

* Writing tests for API endpoints using APITestCase.

## Day 28: Deployment Basics

* Preparing the project for deployment.
* Deploying APIs to Heroku or PythonAnywhere.

## Day 29: Capstone Project - Setup

* Define the project structure and requirements.
* Examples: Blog API, To-Do API, or E-Commerce backend.

## Day 30: Capstone Project - Build & Deploy

* Create CRUD endpoints for the project.
* Apply authentication, permissions, and pagination.

# Deploy the project and review