Python Web Training

Roadmap

We will divide our training process in five steps

Step 1 (1 week**)**

- Onboarding
- UI Training

Step 2 (4 week**)**

- Python Training and Practice Problems (5 days)
- Basics & web Revision (5 days)
- Django & Git Training (7 days)
- Django REST Framework (3 days)

Step 3 (1.5 weeks**)**

- Demo Project
- Final Review & Evaluation

Step 4 (2 week**) - MANDATORY**

- React Training
- React Demo Project
- Integrate the Demo Project with DRF Practice Project

Step 5 (Bonus**)**

- Flask Training (contact your buddy for more information on this)

Points to remember

We will learn the Language (Python) before jumping in Django
We'll assume that you will be learning the concepts by practical implementation yourself. Google and Official documentation of Django are your best friends for the next few weeks. So bookmark this page.

Note: We will use **Python 3.11** and **Django 5.0** for our training process.

Note: Following exact timeline with division on weeks/days is tentative, it's not strictly enforced or required to follow this exact timeline, if you need more time on some topics, take it, if you can finish early, do that.
This timeline is for just an average expectation, excluding any factors like prior experience in respected topics and even days off or time needed for other activities along the way to onboarding(like bank account opening etc).
But obviously, sooner the better.

First Things First:

1. Keep practicing the concepts you learn in all sections on your own, try to think of the examples on your own. You can't come in evaluation saying that i don't know how this thing works because there were no examples/exercises to solve. Making sure that you understand the concept is your responsibility.
2. If any kind of plagiarism is found anywhere during training, it will have severe consequences.
3. For any/all github repos you make, for python/django, don't commit in master, make a branch and add commits to it, and make a PR, so we can add review comments and keep track of the comments in there. Only merge in master after it is approved.

Timeline

Note: It is expected that you'll be working according to the following timeline under ideal conditions. If you think some topics are taking more time or you are stuck somewhere, timely let your buddy know.

Week 0 - Day 1 - 2 (On Boarding)

1. HR paperwork and orientation sessions, assets assignment, bank account, HRMS know-how and most importantly, go through all company SOPs and training SOPs.
2. **Machine setup**, you *can* follow this guide as a starting point.
   NOTE: This guide only shows what needs to be installed, if you face any issues, try to figure out the fixes yourself as well and bring the issues up with the buddy.

Week 0 - Day 3 - 5 (UI Training)

Frontend training

You will polish your basic skills around HTML and CSS in this section. We will use bootstrap to ease some burden, you will learn bootstrap as well as raw css techniques in these 2 days.

1. **FOR BUDDY**: When a trainee is assigned to you and his/her UI training is starting. Create a thread on slack and ask the Training Department to assign you a UI assignment on the LMS, (who is taking care of UI training and assignments).
2. All the communication related to assignment will happen in the thread unless there is something which needs to be addressed confidentially.
3. When you complete your assignment, have your buddy review if there is any improvement required. Make sure everything looks fine then deploy it on some server I.e Netlify/git-pages etc. Share the link in the shared thread which was created at the beginning of training, where Mohsin will final evaluate the assignment and give his feedback.
4. Partial Trainees will have 4 days and full trainees have 2 days for completing UI assignment and you will make sure that they will not take more time than mentioned, otherwise it will create training overlapping and lags.
5. You may **follow these resources and get yourself familiar with the basics of HTML and CSS(practice basics here)**, try to finish these in a limited time(~1 day).
    1. HTML basics, follow from W3Schools HTML Tutorial through to this section HTML Versus XHTML
    2. CSS basics, follow from W3Schools CSS Tutorial through to w3schools.com: CSS Specificity also cover flexbox in advanced section(flex is now a standard, no more an advanced thing though)
       Introduction to CSS layout - Learn web development | MDN
    3. Create a Complete Responsive Website Using Bootstrap
    4. HTML5 and CSS3 Responsive design with media queries (basic understanding)
    5. Difference in %, px, em, rem, vh, vw etc

6. After you are done with above resources(and practiced along), now you need to finalize the assigned UI page (~1 day including review/approval):
    1. You can use **Bootstrap 4** for layout, for Icons use **FontAwesome** and for Font Style use **GoogleFonts**
    2. Page should be fully **responsive** according to the provided designs, it should also cater for mid screens like ipads etc, check your implementation on free hand responsive tests built in to the browsers (1920px to 360px) don't use predefined screen sizes in the browser responsive test.
    3. Use proper naming conventions, make meaningful classes names, use comments.
    4. Don't use inline and Internal CSS, make separate CSS file, don't use IDs for styling the controls and don't apply styles directly on controls, make separate style classes for it.
    5. If any kind of **plagiarism** is found in evaluation, it will have severe consequences.
    6. Once implemented, deploy your page, you can use netlify or any other static site hosting for this.
    7. Once you complete page implementation, your **buddy will review** and after his approval, you will get an external evaluation.
    8. After buddy approves this page implementation's code, the **training department will evaluate your progress** and recommend if you need more improvements on this or you are good to go.
7. Once your UI clears the evaluation, please **ask your buddy to generate an email** with following instructions:
   From: Buddy
   Subject: UI Training Completed - ASE - NAME
   To: the training department, with you(trainee) and the UI reviewer in cc.`
   This email should contain a link of your static site deployment as well.

Week 1 - (Python Training)

(Day 1)

Prerequisites

Jupyter Notebook:
The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

1. What is a jupyter Notebook ?
2. How to use a jupyter notebook ?
3. Resources:
    a. Jupyter Notebook: An Introduction – Real Python
    b. Project Jupyter | Home

Why Python?

The python language is one of the most accessible programming languages available because it has simplified syntax and is not complicated, which gives more emphasis on natural language. Due to its ease of learning and usage, python codes can be easily written and executed much faster than other programming languages.

1. Features
2. Python 2 vs Python 3
3. Memory management in Python
4. GIL in Python
5. Resources
    a. Python Features
    b. Top 10 Reasons Why Python is So Popular With Developers in 2021
    c. Python 2 vs Python 3: Which One You Should Learn
    d. Python 3.11
    e. Memory Management in Python
    f. Python Memory management
    g. Python-GIL
    h. What is GIL in Python

Python Virtual Environment and Packages

At its core, the main purpose of Python virtual environments is to create an isolated environment for Python projects. This means that each project can have its own dependencies, regardless of what dependencies every other project has.

A Python package contains all the files you need for a module. Modules are Python code libraries you can include in your project.

1. What is Pip?
2. What is Virtualenv?
3. What is Pipenv?
4. Resources
    a. What Is Pip? A Guide for New Pythonistas – Real Python realpython.com
    b. A non-magical introduction to Pip and Virtualenv for Python beginners - Blog
    c. Pipenv: A Guide to the New Python Packaging Tool – Real Pythonrealpython.com
    d. Python Virtual Environments: A Primer – Real Python

Setup Python linting and formatting

Preferred tools for linting in python is Pylint and formatting is Black.

1. Resources
    a. Essential python tools
    b. Setup linting in VS Code
    c. Python linting
    d. Black
    e. Format with Black

Syntax

Python was designed to be a highly readable language. It has a relatively uncluttered visual layout and uses English keywords frequently where other languages use punctuation. Python aims to be simple and consistent in the design of its syntax.

2. Difference between Compiled and Interpreted Language
3. Loosely Typed vs Strongly Typed Languages
4. Indentation
5. Variables
6. Comments
7. Scopes
8. Docstrings
9. Resources
    a. Difference between Compiled and Interpreted Language
    b. Loosely typed vs strongly typed languages
    c. Execute Python Syntax
    d. How to Use Python: Your First Steps – Real Python In this step-by-step tutorial, you
    e. Learn Python Programming - Python Tutorial
    e.

Variables

Python supports different types of variables (data types) such as whole numbers, floating point numbers and text.
You do not need to specify the datatype of a variable, you can simply assign any value to a variable.

1. Dynamic Typing vs Static Typing
2. Resources
    a. Static vs. Dynamic Typing
    b. Variables and Types
    c. W3schools Python Variables
    d. Python Tutorial Python Variables, Constants and Literals

Operators

Operators are special symbols in Python that carry out arithmetic or logical computation. The value that the operator operates on is called the operand.

1. Arithmetic operators
2. Comparison operators
3. Logical operators
4. Bitwise operators
5. Assignment operators
6. Specials operators
    a. Identity operators
    b. Membership operators
7. Resources
    a. W3Schools Python Operators
    b. Python Operators: Arithmetic, Comparison, Logical and more.

Control Structures

Control Structures are the blocks that analyze variables and choose directions in which to go based on given parameters. The basic Control Structures in programming languages are: Conditionals (or Selection): which are used to execute one or more statements if a condition is met.

1. Iteration (for, while)
2. Selection (if, else)
3. Resources
    a. 1.10. Control Structures — Problem Solving with Algorithms and Data Structures
    b. loops in python

(Day 2)

Type Casting

Typecasting, or type conversion, is a method of changing an entity from one data type to another. It is used in computer programming to ensure variables are correctly processed by a function. An example of typecasting is converting an integer to a string.

1. Implicit vs Explicit Type Casting
2. Resources
    a. Python Type Conversion and Type Casting (With Examples)
    b. Python Casting Tutorial

Type Hinting

Type hinting is a formal solution to statically indicate the type of a value within your Python code. It was specified in PEP 484 and introduced in Python 3.5. Although adding type hinting in python doesn't enforce it but packages like mypy (from pip) can be used to facilitate and enforce it to some extent

1. Resources
    c. Type Hinting
    d. Python type hinting
    e. MyPy

Exception Handling

Python has many built-in exceptions that are raised when your program encounters an error (something in the program goes wrong).
When these exceptions occur, the Python interpreter stops the current process and passes it to the calling process until it is handled. If not handled, the program will crash.

1. try-except
2. try-except-else
3. try-except-else-finally
4. Resources
    a. Python Tutorial Python Exception Handling Using try, except and finally statement
    b. Python Try Except – w3schools.com

Functions

In Python, a function is a group of related statements that performs a specific task. Functions help break our program into smaller and modular chunks. As our program grows larger and

larger, functions make it more organized and manageable. Furthermore, it avoids repetition and makes the code reusable.

1. Syntax
2. Partial functions
3. Anonymous/lambda functions
4. Recursive functions
5. Monkey patching
6. Resources
    a. Python Functions (def): Definition with Examples
    b. Partial Functions in Python
    c. Python Lambda (Anonymous) Function
    d. Thinking Recursively in Python – Real Python
    e. Monkey Patching in Python (Dynamic Behavior)

Collections

There are four collection data types in the Python programming language:

1. List is a collection which is ordered and changeable. Allows duplicate members.
    a. Slicing of a List
    b. List Comprehension
    c. Operations on List
    d. List Methods
    e. Resources
        i. Python Lists
        ii. Python List (With Examples)
        iii. Python - Lists
2. Tuple is a collection which is ordered and unchangeable. Allows duplicate members.
    a. Converting list to a Tuple
    b. Concatenation of Tuples
    c. Nesting of Tuples
    d. Slicing in Tuples
    e. Resources
        i. Detailed article on Tuples in Python
        ii. Python Tuple (With Examples)
        iii. Python - Tuples
3. Set is a collection which is unordered and unindexed. No duplicate members.
    a. Frozen sets
    b. Union, Intersection and Difference of Sets
    c. Resources
        i. Sets in Python
        ii. Python Set (With Examples)
        iii. Python - Sets
4. Dictionary is a collection which is ordered (prior to Python 3.6, dictionaries were unordered) and changeable. No duplicate members.

    a. Nested Dictionary
    b. pop() vs popItem()
    c. Resources
        i. Recent Articles on Python Dictionary
        ii. Python Dictionary (With Examples)
        iii. Python - Dictionary

Strings

In Python, Strings are arrays of bytes representing 1 character. However, Python does not have a character data type, a single character is simply a string with a length of 1. Square brackets can be used to access elements of the string.

1. Escape Sequencing in Python
2. Formatting of Strings
3. String Methods
4. Resources
    a. Recent Articles on Python String
    b. Python Strings (With Examples)
    c. Python - Strings
    d. Python String Interpolation

(Day 3)

Object Oriented Programming (OOP)

Object-Oriented Programming(OOP), is all about creating “objects”. An object is a group of interrelated variables and functions. These variables are often referred to as properties of the object and functions are referred to as the behavior of the objects. These objects provide a better and clear structure for the program.

Classes and Objects

A class is a collection of objects. Unlike the primitive data structures, classes are data structures that the user defines. They make the code more manageable.
When we define a class only the description or a blueprint of the object is created. There is no memory allocation until we create its object. The objector instance contains real data or information.

1. The Life Cycle of Python Instance Objects
2. Resources
    a. Object Oriented Programming in Python | OOPs Concepts Python
    b. Python - Object Oriented
    c. The Life Cycle of Python Instance Objects | by Yong Cui

Constructors and Class Methods

Constructors are generally used for instantiating an object.The task of constructors is to initialize(assign values) to the data members of the class when an object of class is created.In Python the __init__() method is called the constructor and is always called when an object is created.
A class method is a method which is bound to the class and not the object of the class. They have access to the state of the class as it takes a class parameter that points to the class and not the object instance. It can modify a class state that would apply across all the instances of the class.

1. Types of constructors
2. Resources
    a. classmethod() in Python
    b. Constructors in Python
    c. Python's Instance, Class, and Static Methods Demystified – Real Python

Inheritance and its types

Inheritance is the capability of one class to derive or inherit the properties from another class.

1. Single inheritance
2. Multi-level inheritance
3. Multiple inheritance
4. Hierarchical inheritance
5. Hybrid inheritance
6. Resources
    a. Types of Inheritance in Python | Python Inheritance [With Example]
    b. Inheritance in Python

Polymorphism

The literal meaning of polymorphism is the condition of occurrence in different forms. Polymorphism is a very important concept in programming. It refers to the use of a single type entity (method, operator or object) to represent different types in different scenarios.

1. Run-time
2. Compile-time
3. Resources
    a. Polymorphism in Python(with Examples)
    b. Polymorphism in Python | Object Oriented Programming (OOPs)

Class, Instance and Static variables/methods

In Python, class variables, instance variables, static methods, and instance methods contribute to building an object-oriented structure. Let's explore each concept. Explore each concept in detail.
Class or Static variables are the variables that belong to the class and not to objects. Class or Static variables are shared amongst objects of the class. All variables which are assigned a

value in the class declaration are class variables. And variables which are assigned values inside class methods are instance variables.

1. Class Method vs Static Method
2. Resources
    a. Python Static Variables and Methods
    b. class method vs static method in Python
    c. Class vs Static vs Instance methods

Function Overloading and Function Overriding

Function overloading is a feature of object oriented programming where two or more functions can have the same name but different parameters.
Function overriding is a feature that allows us to have the same function in the child class which is already present in the parent class. A child class inherits the data members and member functions of parent class, but when you want to override a functionality in the child class then you can use function overriding.

1. Advantages and Disadvantages of function overloading
2. Advantages and Disadvantages of function overriding
3. Resources
    a. Difference between Method Overloading and Method Overriding in Python
    b. Python Inheritance Tutorial- Method Overloading & Method Overriding

File Handling

Python treats files differently as text or binary and this is important. Each line of code includes a sequence of characters and they form a text file. Each line of a file is terminated with a special character, called the EOL or End of Line characters like comma {,} or newline character. It ends the current line and tells the interpreter a new one has begun.

1. Working of open() function
2. Using write along with with() function
3. Resources
        a. Basics of File Handling in Python
        b. Python File I/O: Read and Write Files in Python
        c. Python - Files I/O
        d. Context Managers and Python's with Statement

Iterators

Iterator in python is an object that is used to iterate over iterable objects like lists, tuples, dicts, and sets. The iterator object is initialized using the iter() method. It uses the next() method for iteration.

1. Building Custom Iterators
2. Python Infinite Iterators
3. Resources
        a. Python Iterators (__iter__ and __next__): How to Use it and Why?

    b. Python Iterators

Generators

A generator is a function that returns an object (iterator) which we can iterate over (one value at a time).

1. Differences between Generator function and Normal function
2. Python Generators with a Loop
3. Pipelining Generators
4. Reading Large File
5. Resources
    a. Python yield, Generators and Generator Expressions
    b. Generators in Python
    c. How to Use Generators and yield in Python – Real Python

(Day 4)

Decorators

A decorator is a design pattern in Python that allows a user to add new functionality to an existing object without modifying its structure. Decorators are usually called before the definition of a function you want to decorate.

1. First Class Objects
2. Chaining Decorators
3. Decorators with parameters in Python
4. Memoization using decorators in Python
5. Resources:
    a. Decorators in Python
    b. Python Decorators: How to Use it and Why?

Multithreading and Multiprocessing

Multithreading and multiprocessing offer a gateway to perform concurrent operations within a single process or across multiple processes.

1. Resources:
    a. Introduction to multithreading and multiprocessing in python
    b. Python - Multiprocessing vs Multithreading
    c. Difference betweeen multithreading and multiprocessing

Shallow Copy vs Deep Copy

A shallow copy constructs a new compound object and then (to the extent possible) inserts references into it to the objects found in the original. A deep copy constructs a new compound object and then, recursively, inserts copies into it of the objects found in the original.

1. Resources
    a. copy in Python (Deep Copy and Shallow Copy)
    b. copy — Shallow and deep copy operations — Python 3.9.5 documentation

Numpy

NumPy is the fundamental package for scientific computing in Python. NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data.

1. Numpy arrays and functions
2. Resources
    a. Introduction to NumPy
    b. NumPy quickstart — NumPy v1.21 Manual
    c. Python Numpy Tutorial | Learn Numpy Arrays With Examples

Pandas

Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool built on top of the Python programming language.

3. Resources
    a. Pandas
    b. Pandas Tutorial
    c. Learn Pandas on Kaggle

Scraping with python

Python provides powerful tools for web scraping. Widely used packages include BeautifulSoup, Scrapy and Selenium. It is recommended to get familiar with beautifulsoup and scrapy and get some hands-on experience.

4. Resources
    a. A guide to web scraping in Python
    b. https://nanonets.com/blog/web-scraping-with-python-tutorial/
    c. Web scraping with BeautifulSoup
    d. Scrape web using scrapy

(Day 5)

Practice Problems

Prerequisites

Python style guide, Linter and Formatter
Python style guide (PEP 8) gives coding conventions for the python code comprising the standard library in the main Python distribution. Linting highlights syntactical and stylistic

problems in your Python source code, which oftentimes helps you identify and correct subtle programming errors or unconventional coding practices that can lead to errors. For example, linting detects use of an uninitialized or undefined variable, calls to undefined functions, missing parentheses, and even more subtle issues such as attempting to redefine built-in types or functions.

a. Python style guide (PEP 8)
b. What is Linter ?
c. What is a formatter ?
d. Difference Between Linter and Formatter?
e. Resources:
    i. PEP 8 -- Style Guide for Python Code
    ii. Linting Python
    iii. Python Code Quality: Tools & Best Practices – Real Python

SOPs

1. Using all you have learnt about the language(basics), **build some example programs(suggestions below)**, workout with your buddy to decide which programs to implement.
   Preferably something which includes dealing with files + some kind of parsing(CSVs) + some data generation + some network calls.
2. Get these **codes reviewed** by your buddy. And these example codes should not be discarded, and might be reviewed by the training department.
3. **Once you are done with practice problems, upload them in a repo on github and send the repo link with your next daily standup(both in email and on slack).**
    a. Don't commit in master, make a branch named dev and add all commits to it, and make a PR, so we can add review comments and keep track of the comments in there.
    *b. Repo should be private with only your **buddy and training department.***
4. *Get the code you write for each problem reviewed by your buddy properly, fix all feedback received before moving to the next problem and before evaluation by the training department. Code should be modular, properly split in modules, classes/files etc.*

Problems

Following are **some suggestions for example programs,** work with your buddy to choose from these, buddy can assign multiple different exercises as well(***Doing all of them is not required, neither recommended*** as we only have limited time for training):
(If there are any suggestions for improving exercises, please suggest edits)

1. Random password generator, with gradually adding options of different kinds(include alphabets/numeric/special-characters/length etc as desired by the user). (Difficulty: Easy)
2. Weather man: (Difficulty: Hard)

3. Build a web-scraper (Ask your buddy about the package, page and information to scrape)

Week 2 - (Web and Basic Revision)

(Day 1 - 4)

For Basics Revision, you will be given a maximum of 5 days to complete basic training and have it evaluated. In-case, you are able to complete the document before the 5 day deadline; you need to move to the next part of training right away as the rest of the 2 days will be discarded. For example: If you have completed the document in 3 days then you will not be given the leverage of 2 days. If you have any confusion regarding this point, it is your responsibility to reach out to L&D Team for clarification.

(Day 5)

Prerequisites

Basics of web:

1. If you are not comfortable with basic web concepts, or you didn't had chance to take basic web course in your university education, you can use following resources to learn basic of web development concepts:
2. How does the internet work? History, terminologies, protocols, IP, HTTP(s), TCP, SLL, Future
3. Web Application Development
4. How the Web Works: A Primer for Newcomers to Web Development (or anyone, really) (Read both parts of the above article, link to second part is at the end of the provided link)
5. **Must know:** Http vs https, **UDP** vs TCP, DNS, Request life cycle, Different request types(get, post, put etc) and their differences, application server vs web server, how encryption works vs how hashing works etc. What are HTTP headers, what are comment headers?
6. **Good to know**(above average): Details of how SSL/**HTTPS** encryption actually works, how handshake works in http, and what nameservers are.
7. **You should be comfortable with at-least these basics of web development before you begin proper training outline.**

What is a Web Framework?

A web app framework or web framework is a software framework that is created to support the development of dynamic sites, web services, and web applications.

1. Types of Web Frameworks
2. Resources
    a. Web Frameworks: How To Get Started
    b. What is a Web Framework?

Week 3 - (Django)

(Day 1)

Django Apps

A Django application is a Python package that is specifically intended for use in a Django project. An application may use common Django conventions, such as having models, tests, urls, and views submodules.

1. Projects and applications
2. Django Admin App
3. Resources
    a. Applications
    b. How to Create an App in Django ?
    c. Django Admin Site
    d. Customize the Django Admin With Python – Real Python

Models

A model is the single, definitive source of data about your data. It contains the essential fields and behaviors of the data you're storing. Generally, each model maps to a single database table.

1. Field types
2. Field options
    a. NULL vs BLANK
3. Automatic primary key fields
4. Verbose field names
5. Relationships
    a. Many-to-one relationships
    b. Many-to-many relationships
    c. Extra fields on many-to-many relationships
    d. One-to-one relationships
6. Models across files
7. Field name restrictions

8. Custom field types
9. Proxy and Abstract models
10. Managed vs Unmanaged models
11. Resources
    a. Models
    b. Table 7-1 Django model data types and generated DDL by database
    c. Model Field Name restrictions in Django Framework
    d. Django Tutorial Part 3: Using models - Learn web development | MDN
    e. Django - Models
    f. One-to-one relationships
    g. Many-to-one relationships

Django Migrations

Migrations are Django's way of propagating changes you make to your models (adding a field, deleting a model, etc.) into your database schema. They're designed to be mostly automatic, but you'll need to know when to make migrations, when to run them, and the common problems you might run into.

1. migrate
2. makemigrations
3. sqlmigrate
4. Showmigrations
5. Reversing migrations
6. Fake migrations
7. Resources
    a. Django Migrations

(Day 2)

Django ORM

One of the most powerful features of Django is its Object-Relational Mapper (ORM), which enables you to interact with your database, like you would with SQL. In fact, Django's ORM is just a pythonic way to create SQL to query and manipulate your database and get results in a pythonic fashion.

1. Introduction to ORM
2. What is an ORM?
3. How does an ORM solve problems?
4. Difference between select_related and prefetch_related?
5. Explain Q objects in Django ORM?
6. Resources

    b. Getting to know the Django ORM
    c. Django ORM Tutorial - The concept to master Django framework
    d. https://www.youtube.com/watch?v=mO-pfdJpnBA
    e. https://www.youtube.com/watch?v=5-UN4YPDDQc
    f. https://swapps.com/blog/quick-start-with-django-orm/

Querying Data

Queries allow us to perform CRUD (create, read, update and delete) operations on our Django Models.
Query expressions describe a value or a computation that can be used as part of an update, create, filter, order by, annotation, or aggregate. When an expression outputs a boolean value, it may be used directly in filters. There are a number of built-in expressions that can be used to help you write queries. Expressions can be combined, or in some cases nested, to form more complex computations.

1. Making queries
2. Query Expressions
3. CRUD Operations
4. Field Lookups
5. Q objects
6. F expressions
7. Raw queries
8. Spanning multivalued relationships
9. Querying JSON field
10. Defer, only, values, value_list, prefetch_related and select_related, annotate
11. Resources
    a. Making queries
    b. Query Expressions

(Day 3)

Views

A view function is a Python function that takes a Web request and returns a Web response. This response can be the HTML contents of a Web page, or a redirect, or a 404 error, or an XML document, or an image, anything that a web browser can display.

1. Function based views
2. Class based views
3. Generic class based views
4. Async views
5. Resources

    a. Writing views
    b. Django : Class Based Views vs Function Based Views
    c. Views In Django | Python

Templates

A Django template is a text document or a Python string marked-up using the Django template language. Some constructs are recognized and interpreted by the template engine. A template is rendered with a context. Rendering replaces variables with their values, which are looked up in the context, and executes tags.

1. The Django template language
2. Variables
3. Filters
4. Tags
5. Template Inheritance
6. Include vs Extend
7. Resources
    a. The Django template language
    b. How to use the Extends and Include Django Template Tags

Url configuration/mapping/routing

Routing provides a mapping between URLs and views in your django application.

1. How Django processes a request
2. Path converters
3. Passing extra options to view functions
4. Reverse resolution of URLs
5. Resources
    a. URL dispatcher
    b. Django Routing Examples
    c. Django - URL Mapping
    d. Python RegEx (Used for urlpatterns)

(Day 4)

Django Forms

Django provides a special class which is used to create HTML forms. It describes a form and how it works and appears.

1. Form classes
2. Model Form
3. Difference between form and model form

4. Fields
5. Form Templates
6. Bounded vs Unbounded forms
7. Resources
    a. Working with forms
    b. Render Django Forms

Django Messages

Quite commonly in web applications, you need to display a one-time notification message (also known as “flash message”) to the user after processing a form or some other types of user input.
For this, Django provides full support for cookie- and session-based messaging, for both anonymous and authenticated users. The messages framework allows you to temporarily store messages in one request and retrieve them for display in a subsequent request (usually the next one). Every message is tagged with a specific level that determines its priority (e.g., info, warning, or error).

1. Resources
    c. Django Messages

Django Mixins

Mixins are essentially reusable pieces of code that can be added to a class to give it additional functionality. In Django, mixins are used to add functionality to views, forms, and models, allowing developers to reuse code and improve the efficiency of their applications. Mixins are typically defined as classes, and can be added to other classes by using inheritance.

1. Resources
    a. Mastering mixins in django

(Day 5)

Authentication and Authorization

Django comes with a user authentication system. It handles user accounts, groups, permissions and cookie-based user sessions.

1. Authentication vs Authorization
2. Permissions and authorization
3. Authentication in web requests
4. Managing users in the admin
5. Customizing Users and authentication
6. Password management in Django
7. Resources
    a. User authentication in Django
    b. Using the Django authentication system
    c. Customizing authentication in Django
    d. Password management in Django

Testing

Software Testing is evaluation of the software against requirements gathered from users and system specifications. Testing is conducted at the phase level in software development life cycle or at module level in program code. Software testing comprises Validation and Verification. Make sure you are familiar with testing basics before proceeding to testing in django.

1. Testing Basics
2. Testing in Django
3. Faker
4. Factory Boy
5. Test coverage
6. Testing Models with Django using Faker and Factory Boy
7. Fixtures
8. Mocking responses and return values
9. Resources
    a. Software Testing Overview
    b. Testing in Django
    c. Faker · PyPI
    d. factory-boy · PyPI
    e. coverage · PyPI
    f. Testing Models with Django using Faker and Factory Boy

Week 4 - (Django & DRF)

(Day 1)

Signals

Django includes a “signal dispatcher” which helps allow decoupled applications to get notified when actions occur elsewhere in the framework. In a nutshell, signals allow certain senders to notify a set of receivers that some action has taken place. They’re especially useful when many pieces of code may be interested in the same events.

1. Defining Signals
2. Sending Signals
3. Receiver Functions
4. Listening to Signals
5. Resources
    a. Django Signals
    b. How to Create Django Signals Tutorial
    c. Intro to Django Signals

Sending Emails

Statistics tell us that people will open only those emails which are well optimized. So the style of writing an email matters a lot. With a small investment in emailing software, businesses can get a substantial return on investment. This helps to generate a multiplier effect and have the ability to tremendously improve sales.

1. Sending Email
2. Sending Email to specific Users
3. Different Email Backends
4. Email Templates
5. Resources
    a. Sending email
    b. Django - Sending E-mails
    c. How to Send Email in a Django App

Pagination

Django provides high-level and low-level ways to help you manage paginated data – that is, data that’s split across several pages, with “Previous/Next” links.

1. The Paginator class
2. Paginating a ListView
3. Resources

    a. Pagination
    b. How to Paginate with Django
    c. Using Django Pagination

Middlewares

In Django, middleware is a lightweight plugin that processes during request and response execution. Middleware is used to perform a function in the application. The functions can be security, session, CSRF protection, authentication etc.

1. What is middleware?
2. Uses of middlewares
3. Custom middleware
4. Activating a middleware
5. Marking middleware as unused
6. Middleware order and layering
7. Async Support
8. Resources
    a. Middleware
    b. Django Middleware - javatpoint
    c. A Comprehensive Guide to Django Middleware - DEV Community

(Day 2)

Async/Background tasks in Django

If a long-running process is part of your application's workflow, rather than blocking the response, you should handle it in the background, outside the normal request/response flow. Perhaps your web application requires users to submit a thumbnail (which will probably need to be re-sized) and confirm their email when they register. If your application processed the image and sent a confirmation email directly in the request handler, then the end user would have to wait unnecessarily for them both to finish processing before the page loads or updates. Instead, you'll want to pass these processes off to a task queue and let a separate worker process deal with it, so you can immediately send a response back to the client. The end user can then do other things on the client-side while the processing takes place. Your application is also free to respond to requests from other users and clients.
To achieve this, we'll walk you through the process of setting up and configuring Celery and Redis for handling long-running processes in a Django app.

1. Resources
    a. Django and celery
    b. Using celery with django
    c. Celery integration with django

Caching

A fundamental trade-off in dynamic websites is, well, they're dynamic. Each time a user requests a page, the web server makes all sorts of calculations – from database queries to template rendering to business logic – to create the page that your site's visitor sees. This is a lot more expensive, from a processing-overhead perspective, than your standard read-a-file-off-the-filesystem server arrangement.
For most web applications, this overhead isn't a big deal. Most web applications aren't washingtonpost.com or slashdot.org; they're small- to medium-sized sites with so-so traffic. But for medium- to high-traffic sites, it's essential to cut as much overhead as possible.
That's where caching comes in.
To cache something is to save the result of an expensive calculation so that you don't have to perform the calculation next time.

1. Resources
    d. Caching in Django

Deployments

There are many options for deploying your Django application, based on your architecture or your particular business needs, but that discussion is outside the scope of what Django can give you as guidance.
Django, being a web framework, needs a web server in order to operate. And since most web servers don't natively speak Python, we need an interface to make that communication happen.

1. ASGI vs WSGI
2. Application server vs Web server
3. Gunicorn
2. Resources
    e. How to deploy django

Setting up Environment Variables in django

In a Django project, there is information that needs to be kept secret; like a Secret key, a Database username and password, and API keys. This is because their exposure to foreign parties can put your project at risk of security attacks, especially if you upload your Django project to GitHub or something similar. In Django projects that are not meant for deployment, this may not be a big issue but for production-level projects, it is mandatory to keep the information safe. In Django, we do that by using environment variables.

1. Resources
    a. Setup environment variables in django

(Django Rest Framework)

(Day 3)

What is an API?

API is the acronym for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other. Each time you use an app like Facebook, send an instant message, or check the weather on your phone, you're using an API.

1. Working
2. Examples
3. Modern API Standards
4. Resources
    a. "What is an API? (Application Programming Interface)".
    b. What is an API? | (API) Application Program Interface Definition

What is a REST API?

A REST API (also known as RESTful API) is an application programming interface (API or web API) that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services. REST stands for representational state transfer and was created by computer scientist Roy Fielding.

1. What is REST?
2. REST vs SOAP
3. Resources
    a. What is REST?
    b. SOAP Vs. REST: Difference between Web API Services
    c. What is a REST API?
    d. Difference between API and REST API

Django Rest Framework

Django Rest Framework (DRF) is a library which works with standard Django models to build a flexible and powerful API for your project.

1. Why use DRF?
2. Difference between Django and DRF
3. Django vs Flask for creating APIs
4. Resources
    a. What is Django Rest Framework and why you should learn it
    b. Django Best Practices and Tips for Beginners
    c. Why you should choose Django over Flask for your next API project

(Day 4)

Requests and Responses

REST framework introduces a Request object that extends the regular HttpRequest, and provides more flexible request parsing. The core functionality of the Request object is the request.data attribute, which is similar to request.POST, but more useful for working with Web APIs.
REST framework also introduces a Response object, which is a type of TemplateResponse that takes unrendered content and uses content negotiation to determine the correct content type to return to the client.

1. Request object
2. Response object
3. Status codes
4. Resources
    a. 2 - Requests and responses
    b. Working with request.data in Django REST framework

Serialization

Serialization is the process of converting objects into a stream of data. The serialization and deserialization process is platform-independent, it means you can serialize an object in a platform and deserialize in a different platform.

1. Serializer
2. BaseSerializer
3. ModelSerializer
4. ListSerializer
5. HyperlinkedModelSerializer
6. Resources
    a. 1 - Serialization
    b. Serializers - Django REST Framework

Views

Django's class-based views are a welcome departure from the old-style views.
REST framework provides an APIView class, which subclasses Django's View class.

1. APIView vs View
2. Class Based Views
3. Generic Views
4. Resources
    a. Views
    b. 3 - Class based views

(Day 5)

Authentication and Permissions

In DRF, permissions, along with authentication and throttling, are used to grant or deny access for different classes of users to different parts of an API. Authentication and authorization work hand in hand. Authentication is always executed before authorization.

1. View level vs Object level permissions
2. Built-in DRF Permission Classes
3. Custom Permission Classes
4. Resources
    a. Permissions in Django Rest Framework
    b. 4 - Authentication and permissions
    c. Difference between session and token based authentication

JWT in Django Rest Framework

JSON Web Token is a fairly new standard which can be used for token-based authentication. Unlike the built-in TokenAuthentication scheme, JWT Authentication doesn't need to use a database to validate a token. A package for JWT authentication is djangorestframework-simplejwt which provides some features as well as a pluggable token blacklist app.

1. What is JWT?
2. How is JWT used for authentication in DRF?
   Resources
    a. How to Use JWT Authentication with Django REST Framework
    b. JWT(JSON Web Token) With DRF

Viewsets and Routers

REST framework includes an abstraction for dealing with ViewSets, that allows the developer to concentrate on modeling the state and interactions of the API, and leave the URL construction to be handled automatically, based on common conventions.
A ViewSet class is only bound to a set of method handlers at the last moment, when it is instantiated into a set of views, typically by using a Router class which handles the complexities of defining the URL conf for you.

1. Binding ViewSets to URLs
2. Using Routers
3. Trade-offs between views vs viewsets
4. Resources
    a. 6 - Viewsets and routers
    b. Understanding Routers in Django-Rest-Framework | by MicroPyramid | Medium

Week 5 - Day 1 (Pre Test Project Evaluation)

- Refer to this document for general guidelines about the evaluation. **(If you have any confusion to understand this, kindly reach out to L&D department)**
- Follow this document for evaluation criteria.

Week 5 - Day 3 to Week 6- Day 5 - Test Project

The Test Project will be assigned on the LMS by the training team please reach out to the Training team for more information**.**

Week 7 - Day 1 (Test Project Evaluation)

Week 7 - Day 2 to Week 9 - Day 2 (React Training)

- React Training React Demo Project
- Integrate the Demo Project with DRF Practice Project

NOTE: Please skip UI, Git Training and Basics Revision from the document. React Thoery and Project Evaluation is mandatory. Feel free to reach out L&D Department in case of any confusion.

Best of Luck!