ihub
INNOVATIVE PLACEMENT SOLUTIONS

Powered By
QualityThought®
Learn from Industry Experts

100 % JOB
GAURANTEE

SALARY RANGE
3.2 to 12+ LAKHS

WITH 100% JOB GAURUNTEE

Python Django

Python & django

COURSE + INTERNSHIP

www.ihubtalent.com

100% JOB GUARANTEED

PYTHON COURSE CURRICULUM

COURSE DURATION	SESSION HOURS
5 WEEKS	60 HOURS

REALTIME PRACTICE

CASE STUDIES + MOCK TESTS

KEY HIGHLIGHTS

- Only EdTech with 100% Internship based and job guarantee training
- Institute backed by strong IIT alumina Team
- More than 10,000 + success stories created
- Partnered and serving 100+ clients right from the inception of the firm
- Parent company Quality Thought No #1 providing placements since 2010
- Experienced IT Architects, subject matter experts as advisory in the board
- Guaranteed Internship Opportunity (3 to 6Months) for every participant
- Backed by strong Internal human resource team enabling quick placements
- Special guidance to students to win jobs with highest packages

TRAINING HIGHLIGHTS

- Industry ready course roaster designed by architects and subject matter experts
- Proven content, creating gateway to world of Job opportunities
- Practical & detailed approach, everyone can understand and perform to excel
- Subject matter experts as trainers
- Not just learning, Learn by doing methodology
- Access to Instant recorded live Class videos shared
- Personalized attention and guidance to accelerate the learning
- Member access to private group to get instant doubt clarifications
- Guest lectures from leading MNC Companies

PYTHON

the Future Technology

4th Most Used

Programming Language in 2021, 44.1% of developers worked with it

85% of the Time

Python was used as a primary programming Language for the devlopment

Most Popular Uses of Python

40% Machine Learning

55% Data Science

40% Web Development

An average python developer in India draws ₹779,449/- PA

Source PayScale

There are now 8.2 million developers in the world who code using Python and that Population is now larger than those who build in Java, who number 7.6 million

Source -SlashData

Python - Programming from Absolute Beginning - Introduction to Computer Programs

Introduction to Programming Languages - Translating code into something that the computer **understands - Syntax and the building blocks of a programming language**

Types of Applications

Standalone applications , Client-Server Applications, Web applications
Mobile Applications ,Distributed applications, Cloud-based applications

Software Projects and How We Organize Our Code

Working with software projects , Working with packages to share code
Avoiding conflicts with namespaces

Sequence – The Basic Building Block of a Computer Program

Working with Data – Variables

Declaring and initializing variables , Primitive data types
Composite type

Program Control Structures

Controlling the execution path ,
Selection with the if and switch statement
Iteration with the for loop , Iteration with the while loop
Iterating over sequences using for each

Understanding Functions

Deciding what goes into a function , Writing a function
Returning values from a function ,Function arguments,
Functions in action , Local and global variables

When Things Go Wrong – Bugs and Exceptions

Understanding software bugs , types of software bugs
Finding bugs using a debugger

Programming Paradigms

Understanding structured programming , object-oriented programming ,
functional programming - logic programming

Programming Tools and Methodologies

Understanding version control systems
Unit testing, Integration testing , Other types of tests

Software releases

Understanding software deployment Understanding software deployment

Deployment Automation
Code maintenance

Top Companies Using

Software deployment process methodologies

Waterfall development , Spiral model
Agile development

Code Quality

Defining code quality , Writing code with readability in mind
Writing code with efficiency in mind

Practical Version Controlling with Git,
Software defect management using JIRA
Understanding Agile project Management using Scrum

Introduction to Python

A Taste of Python

Mysteries , Little Programs
A Bigger Program , Python in the Real world
Why Python? , Why Not Python?
Installing Python , Running Python , Moment of Zen

Data: Types, Values, Variables, and Names

Python Data are objects , Types , Mutability, Literal Values
Variables , Assignment, Variables are Names, Not Places
Assigning to Multiple Names, Reassigning a Name
Copying, Choose Good Variable Names

Numbers

Boolean
Integers, Literal Integers, Integer Operations ,Integers and Variables
Precedence, Bases , Type Conversions, How Big is int?
Floats, Math Functions

Choose with if

Comment with #, Continue Lines with \
Compare with if, elif and else, What is True
Do Multiple Comparisons with in

Text Strings

Creating with Quotes
Creating with str(), Escape with \
Combine by Using + , Duplicate with +
Get a Character with [] , Get a Substring with a Slice
Get Length with len() , Split with strip()
Search and Select , Case , Alignment
Formatting, Oldstyle: % ,
New styles: {} and format() ,
Newest Style: f-string ,

Top Companies Using

Quora

Eventbrite

Recursion- Async Functions
Exceptions

Object-Oriented Design

What are Objects?

Simple Objects

Define a Class with class,
Attributes
Methods, Initialization

Inheritance

Inherit from a Parent Class, Override a Method
Add a Method, Get Help from your Parent with super()
Multiple Inheritance , Mixins

In Self Defense

Attribute Access

Direct Access
Getters and Setters
Properties for Attribute Access
Properties for Computed Values
Name Mangling for privacy
Class and Object Attributes

Method Types

Instance Methods
Class Methods
Static Methods
Duck Typing
Magic Methods
Aggregation and Composition
When to Use Objects or Something else
Named Tuples
Dataclasses
Attrs

Top Companies Using

INSTAGRAM

Objects Oriented Python

Object Oriented Design

Object-Oriented Design
Introducing object-oriented
Objects and classes
Specifying attributes and behaviors
Hiding details and creating the public interface
Composition
Inheritance - Case Study

Objects in Python

Objects in python
Creating python classes
Modules and packages

More String Things

Loop with while and for

Repeat with while

Cancel with break, Skip Ahead with continue

Check break Use with else ,

Iterate with for and in

Cancel with break, Skip Ahead with continue, Check break Use with else

Generate Number Sequences with range() , Other Iterators

Tuples and Lists

Tuples

Create with Commas and ()

Create with tuple()

Combine Tuples by Using +

Duplicate Items with *

Compare Tuples

Iterate with for and in

Modify a Tuple

Top Companies Using

Lists

Create with [] , Create or Convert with list()

Create from String with split()

Get an Item by [offset] , Get Items with a Slice

Add an item to the End with append(),

Add an Item by offset with insert()

Duplicate All items with *, Combine Lists by Using extend() or +

Change an item by [offset], Change Items with a Slice

Delete an Item by Offset with del,

Delete an Item by Value with remove()

Get an Item by Offset and Delete It with pop()

Delete All items with clear(),

Find an Item's Offset by Value with index()

Test for a Value with in.,

Count Occurrences of a Value with count()

Convert a List to a String with join()

Reorder Items with sort() or sorted()

Get Length with len()

Assign with =, Copy with copy(), list() or a Slice

Copy everything with deepcopy()

Compare Lists(), Iterate with for and in

Iterate Multiple Sequences with zip()

Create a List with a Comprehension- Tuples vs Lists

Dictionaries and Sets

Dictionaries

Create with {}

Create with dict()

- Convert with dict()
- Add or Change an Item by [key]
- Get All Keys with keys()
- Get All Values with values()
- Get All Key-Value Pairs with items()
- Get Length with len ()
- Combine Dictionaries with {**a, **b}
- Combine Dictionaries with update()
- Delete an Item by Key with del
- Get an Item by Key and Delete it with pop()
- Delete All Items with clear()
- Assign with =
- Copy with copy()
- Copy Everything with deepcopy()
- Compare Dictionaries
- Iterate with for and in
- Dictionary Comprehensions

Sets

- Create with set()
- Convert with set()
- Get Length with len()
- Add an Item with add()
- Delete an Item with remove()
- Iterate with for and in
- Combinations and Operators
- Set Comprehensions
- Create an Immutable Set with frozenset()

Functions

- Define a Function with def
- Call a Function with Parentheses

Arguments and Parameters

- None is useful
- Positional arguments
- Keyword Arguments
- Specify Default Parameter Values
- Explode/Gather Positional Arguments with *
- Explode/Gather Keyword Arguments with **
- Keyword-only Arguments
- Mutable and Immutable Arguments
- Docstrings
- Functions are First-Class Citizens
- Inner Functions
- Closures

Anonymous Functions: lambda

- Generators
- Decorators
- Namespaces and Scope
- Uses of _ and __ in Names

Top Companies Using

Organizing module content
Who can access my data? - Third-party Libraries - Case Study

When Objects are Alike

When Objects are Alike
Basic Inheritance
Multiple Inheritance
Polymorphism
Abstract base classes
Case Study

Exceptions

Raising exceptions
Case Study

When to Use Object-Oriented Programming

When to use Object-Oriented Programming
Treat objects as objects
Adding behaviors to class data with properties
Manager objects
Case Study

Python Data Structures

Python Data Structures
Empty Objects
Tuples and named Tuples
Data classes , Dictionaries
Lists, Sets, Extending built-in functions
Case Study

Python Object-Oriented Shortcuts

Python Object-Oriented Shortcuts
Python built in functions
An alternative to method overloading
Functions are objects too
Case Study

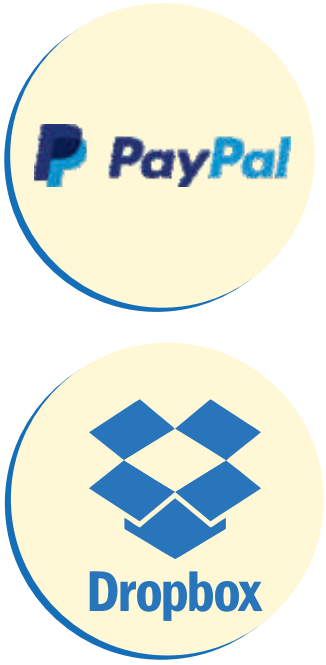
Strings and Serialization

Strings and Serialization
Strings
Regular expressions
Filesystem paths
Serializing objects
Case Study

The Iterator Pattern

The Iterator Pattern
Design patterns in brief

Top Companies Using



Iterators - Comprehensions
Generators - Coroutines - Case Study

Python Design Patterns

The decorator Pattern
The observer Pattern
The Strategy Pattern
The State Pattern
The Singleton Pattern
The template Pattern
The adapter Pattern
The facade Pattern
The flyweight Pattern
The command Pattern
The abstract factory Pattern
The composite Pattern

Top Companies Using

Testing Object-Oriented Programs

Testing Object-Oriented Programs
Why test? - Unit testing
Testing with pytest
Imitating expensive objects
How much testing is enough- Case Study

Concurrency

Concurrency
Threads
Multiprocessing
Futures
Aysnc IO - Case Study

Modules, Packages, and Goodies

Modules and import Statement

Import a Module
Import a Module with Another Name
Import Only What You want from a Module

Packages

The Module Search Path
Relative and Absolute Imports
Namespace Packages , Modules Vs Objects

Goodies in the Python Standard Library

Handle Missing Keys with setdefault() and defaultdict()
Count Items with Counter()
Order by Key with OrderedDict() - Deque
Iterate over Code Structures with itertools
Print Nicely with pprint() -Get Random
More Batteries: Get Other Python Code -

Virtual Environments

Software Testing and Test-Driven Development

Getting Started with Software Testing - Introducing software testing and quality control

Test plans
Introducing automatic tests and test suites
Multiple test cases, Organizing tests
Introducing test-driven development and unit tests
Test-driven development , Test units
Understanding integration and functional tests
Integration tests , Functional tests
Understanding the testing pyramid and trophy
The testing pyramid, The testing trophy
Testing distributions and coverage

Top Companies Using

Test Doubles

Introducing test doubles
Using dummy objects
Replacing components with stubs
Checking behaviors with spies
Using mocks
Replacing dependencies with fakes
Understanding acceptance tests and doubles
Managing dependencies with dependency injection
Using dependency injection frameworks

Test-Driven Development (TDD)

Starting projects with TDD
Building applications, the TDD way
Preventing regressions

Scaling the Test Suite

Scaling tests
Moving e2e to functional
Working with multiple suites
Compile suite , Commit tests, Smoke tests
Carrying out performance tests
Enabling continuous integration , Performance testing

PyTest for Python Testing

Running tests with PyTest
Writing PyTest fixtures , Using fixtures for dependency injection
Managing temporary data with tmp_path
Testing I/O with capsys, Running subsets of the test suites

Dynamic and Parametric Tests and Fixtures

- Configuring the test suite
- Generating fixtures
- Generating tests with parametric tests

Using Behavior-driven development

- Writing acceptance tests
- Writing first test, Defining a feature file
- Declaring the scenario, Running the scenario test
- Further setup with the And step
- Performing actions with the When step
- Assessing conditions with the Then step
- Embracing specifications by example

PyTest Essential Plugins

- PyTest Essential Plugins
- Using pytest-cov for coverage reporting
- Coverage as a service
- Using pytest-benchmark for benchmarking
- Comparing benchmark runs
- Using flaky to rerun unstable tests
- Using pytest-testmon to rerun tests on code changes
- Running tests in parallel with pytest-xdist

Managing Test Environments with Tox

- Introducing Tox
- Testing multiple python versions with Tox
- Using environments for more that Python Versions

Top Companies Using

Playing with data (text and binary)

Text Strings: Unicode

- Python 3 Unicode Strings
- UTF-8, Encode
- Decode, HTML Entities
- Normalization

Text Strings: Regular Expressions

- Find Exact Beginning Match with match()
- Find FirstMatch with search() , Find All Matches with findall()
- Split at Matches with split() , Replace at Matches with sub()
- Patterns: Special Characters, Patterns: Using specifiers
- Patterns: Specifying match() Output

Binary Data

- Bytes and bytearray
- Convert Binary Data with struct
- Other Binary Data Tools

Convert Bytes/String with binascii() - BitOperators

Calendars and Clocks

Leap Year, The datetime module , Using the time module
Read and Write Dates and times
All the Conversions, Alternative Modules

Files and Directories

File Input and Output

Create or Open with open(), Write a Text File with print()
Write a Text File with write() ,
Read a Text File with read(), readline(), or readlines()
Write a Binary File with read() ,Read a Binary File with read()
Close Files Automatically by using with, Change Position with seek()

Memory Mapping

File Operations

Check existence with exists()
Check Type with isfile()
Copy with copy()
Changing Name with rename()
Link with link() or symlink()
Change permissions with chmod(), Change Ownership with chown()
Delete a File with remove()

Directory Operations

Create with mkdir(),
Delete with rmdir()
List contents with listdir(),
Changing current directory with chdir()
List Matching Files with glob()
Pathnames, BytesIO and StringIO

Top Companies Using

Processes and Concurrency

Program and Processes

Create a Process with subprocess
Create a Process with multiprocessing
Kill a Process with terminate
Get System Info with os
Get Process Info with psutil

Command Automation

Invoke
Other Command Helpers

Concurrence

Queues
Processes
Threads - Concurrent.futures
Green Threads and gevent

Twisted
Asyncio, Redis - Beyond Queues

Effective and Performant Python

Pythonic Thinking

Follow PEP 8 Style Guide
Differences between bytes and str
Interpolated F-strings over C-style Format strings and str.format
Writing helper functions instead of complex expressions
Multiple Assignment Unpacking Over Indexing
Prefer enumerate over range
Using zip to process Iterators in Parallel
Avoid Else blocks after for & while loops
Prevent Repetition with Assignment Expressions

Lists and Dictionaries

Know How to Slice Sequences
Avoid Striding and Slicing in a Single Expression, Prefer Catch-All unpacking over slicing
Sort by Complex Criteria Using the key parameter
Be Cautious when relying on dict insertion Ordering
Prefer get Over in and KeyError to Handle Missing Dictionary Keys
Prefer default dict Over setdefault to Handle Missing Items in Internal State
Know How to Construct Key-Dependent Default Values with __missing__

Functions

Never Unpack more than three variables when functions return multiple values
Prefer Raising exceptions to Returning None
Know How Closures interact with Variable Scope
Reduce Visual Noise with Positional Arguments
Provide Optional Behavior with Keywork Arguments
Use Node and Docstrings to Specify Dynamic Default Arguments
Enforce Clarity with Keyword-Only and Positional-Only Arguments
Define Function Decorators with functools.wraps

Comprehensions and Generators

Use Comprehensions Instead of map and filter
Avoid More Than Two Control Subexpressions in Comprehensions
Avoid Repeated Work in Comprehensions by Using Assignment Expressions
Consider Generators Instead of Returning Lists
Be Defensive When Iterating Over Arguments
Consider Generator Expressions for Large List Comprehensions
Compose Multiple Generators with yield from
Avoid Injecting Data into Generators with send
Avoid Causing State Transitions in Generators with throw
Consider itertools for Working with Iterators and Generators

Classes and Interfaces

Compose Classes Instead of Nesting Many Levels of Built-in Types
Accept Functions Instead of Classes for Simple Interfaces

Use @classmethod Polymorphism to Construct Objects Generically
Initialize Parent Classes with super
Consider Composing Functionality with Mix-in Classes
Prefer Public Attributes Over Private Ones
Inherit from collections.abc for Custom Container Types

Meta classes and Attributes

Use Plain Attributes Instead of Setter and Getter Methods
Consider @property Instead of Refactoring Attributes
Use Descriptors for Reusable @property Methods
Use __getattr__, __getattribute__, and __setattr__ for Lazy Attributes
Validate Subclasses with __init_subclass__
Register Class Existence with __init_subclass__
Annotate Class Attributes with __set_name__
Prefer Class Decorators Over Metaclasses for Composable Class Extensions

Concurrency and Parallelism

Use subprocess to Manage Child Processes
Use Threads for Blocking I/O, Avoid for Parallelism
Use Lock to Prevent Data Races in Threads
Use Queue to Coordinate Work Between Threads
Know How to Recognize When Concurrency Is Necessary
Avoid Creating New Thread Instances for On-demand Fan-out
Understand How Using Queue for Concurrency Requires Refactoring
Consider ThreadPoolExecutor -
When Threads Are Necessary for Concurrency
Achieve Highly Concurrent I/O with Coroutines
Know How to Port Threaded I/O to asyncio
Mix Threads and Coroutines to Ease the Transition to asyncio
Avoid Blocking the asyncio Event Loop to Maximize Responsiveness
Consider concurrent.futures for True Parallelism

Top Companies Using

Robustness and Performance

Take Advantage of Each Block in try/except/else/finally
Consider contextlib and with Statements for Reusable try
Use datetime Instead of time for Local Clocks
Make pickle Reliable with copyreg
Use decimal When Precision Is Paramount
Profile Before Optimizing
Prefer deque for Producer& Consumer Queues for Producer–Consumer Queues
Consider Searching Sorted Sequences with bisect
Know How to Use heapq for Priority Queues
Consider memoryview and bytearray for Zero-Copy Interactions with bytes

Testing and Debugging

Use repr Strings for Debugging Output
Verify Related Behaviors in TestCase Subclasses
Isolate Tests from Each Other with setUp, tearDown, setUpModule, and tearDownModule
Use Mocks to Test Code with Complex Dependencies

Encapsulate Dependencies to Facilitate Mocking and Testing
Consider Interactive Debugging with pdb
Use tracemalloc to Understand Memory Usage and Leaks

Collaboration

Know Where to Find Community-Built Modules
Use Virtual Environments for Isolated and Reproducible Dependencies
Write Docstrings for Every Function, Class, and Module
Use Packages to Organize Modules and Provide Stable APIs
Consider Module-Scoped Code to Configure Deployment Environments
Define a Root Exception to Insulate Callers from APIs
Know How to Break Circular Dependencies
Consider warnings to Refactor and Migrate Usage
Consider Static Analysis via typing to Obviate Bugs

Understanding Performant Python

The Fundamental Computer System

Computing Units
Memory Units
Communications Layers

Putting the Fundamental Elements Together

dealized Computing Versus the Python Virtual Machine
So Why Use Python?

How to Be a Highly Performant Programmer

Good Working Practices

Profiling to Find Bottlenecks

Profiling Efficiently
Introducing the Julia Set
Calculating the Full Julia Set
Simple Approaches to Timing—print and a Decorator
Simple Timing Using the Unix time Command
Using the cProfile Module
Visualizing cProfile Output with SnakeViz
Using line_profiler for Line-by-Line Measurements
Using memory_profiler to Diagnose Memory Usage
Introspecting an Existing Process with PySpy

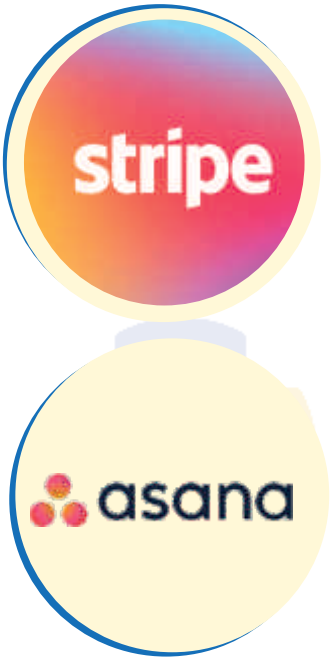
Bytecode: Under the Hood

Using the dis Module to Examine CPython Bytecode
Different Approaches,
Different Complexity

Unit Testing During Optimization to Maintain Correctness

No-op @profile Decorator
Strategies to Profile Your Code Successfully

Top Companies Using



Asynchronous I/O

- Introduction to Asynchronous Programming
- How Does async/await Work?

Serial Crawler

- Gevent
- Tornado
- Aiohttp

Shared CPU–I/O Workload

- Serial
- Batched Results
- Full Async

The multiprocessing Module

- An Overview of the multiprocessing Module

Estimating Pi Using the Monte Carlo Method

Estimating Pi Using Processes and Threads

- Using Python Objects
- Replacing multiprocessing with Joblib
- Random Numbers in Parallel Systems
- Using numpy

Finding Prime Numbers

- Queues of Work

Verifying Primes Using Interprocess Communication

- Serial Solution
- Naive Pool Solution
- A Less Naive Pool Solution
- Using Manager.Value as a Flag
- Using Redis as a Flag
- Using RawValue as a Flag
- Using mmap as a Flag
- Using mmap as a Flag Redux

Sharing numpy Data with multiprocessing

Synchronizing File and Variable Access

- File Locking
- Locking a Value

Clusters and Job Queues
Using Less RAM
Lessons from the Field

Top Companies Using

Python Django

Persistent Storage

- Flat Text Files
- Padded Text FilesTabular Text Files
  - CSV - XML, HTML
  - JSON , YAML , Tablib
  - Pandas, Configuration Files
- Binary Files
  - Padded Binary Files and Memory Mapping
  - Spreadsheets , HDF5 , TileDB
- Relational Databases
  - SQL, DB-API
  - SQLite , MySQL
  - PostgreSQL, SQLAlchemy
- NoSQL Datastores
  - The dbm Family
  - Memcached
  - Redis, Document Databases
  - Time Series Databases
  - Graph Databases
  - Other NoSQL
- Full-Text Databases

Networks

- TCP/IP
- Networking Patterns
- The Request-Reply Pattern
  - Zero MQ
  - Other Messaging tools
- The Publish-Subscribe Pattern
  - Redis
  - ZeroMQ
  - Other Pub-Sub Tools
  - Internet Services- DNS- Python Email Modules
  - Web Services and APIS
- Data Serialization
  - Serialize with pickle
  - Other Serialization Formats
- Remote Procedure Calls
  - XML RPC- JSON RPC- Zerorpc- gRPC- Twirp

Top sites & products built with Python & Django

Web applications and Services

HTML Web Development
CSS, JavaScript. SQL Databases (mysql)
NoSQL Databases (mongo db)
SQL Databases and Python (SQLAlchemy)
NoSQL Databases and Python (PyMongo)
Responsive Web Design
ReactJS

Top sites & products built with Python & Django

Introduction to Django

Introduction
Scaffolding a Django Project and App
Creating a Project and App, and Starting the Dev Server
Model View Template
Models
Views
Templates

Introduction to HTTP
Processing a Request
Django Project
The myproject Directory
Django Development Server
Django Apps
PyCharm Setup
Project Setup in PyCharm
View Details
URL Mapping Detail
Writing a View and Mapping a URL to It
GET, POST, and QueryDict Objects
Exploring GET Values and QueryDict
Exploring Django Settings- *Using Settings in Your Code*
Finding HTML Templates in App Directories
Creating a Templates Directory and a Base Template
Rendering a Template with the render Function
Rendering a Template in a View
Rendering Variables in Templates
Using Variables in Templates
Debugging and Dealing with Errors
Exceptions -Generating and Viewing Exceptions -
Debugging
Creating a Site Welcome Screen

Customizing the Admin Interfaces

Serving Static Files

- Introduction
- Static File Finders
- App Directories Finder
- Static File Namespacing
- FileSystemFinder
- Custom Storage Engines

Forms

- Introduction
- The <form> element
- Types of Input
- Form Security with Cross-Site Forgery Protection
- Accessing Data in the View
- Choosing b/w GET and POST
- Django Form's Library
- Validating Forms & Retrieving Python Values

Advanced Form Validation and Model Forms

- Introduction
- Custom Field Validation & Cleaning

Media Serving and File Uploads

- Setting up Media Uploads & Serving
- Context Processors & using MEDIA_URL in Templates
- File Uploads using HTML Forms
- Storing Files on Model Instances

Sessions and Authentication

- Middleware Modules
- Implementing Authentication Views & Templates
- Password Storage in Django
- The Profile Page
- request.user in Django
- Authentication Decorators - Redirection
- Enhancing Templates with Authentication Data
- Session Engine
- Pickle or JSON Storage
- Storing Data in Sessions

Top sites & products built with Python & Django

Models and Migrations

Introduction

Databases

Relational Databases
Non-Relational Databases
Database Operations
Using SQL Data Types in Relational databases

SQL CRUD Operations

SQL Create Operations
SQL Read Operations
SQL Update Operations
SQL Delete Operations
Django ORM
Database Configuration and Creating Django Applications
Django Apps
Django Migration
Creating Django Models and Migrations
Field Types
Field Options

Top sites & products built with Django & Python

Relationships

One-to-One
Many-to-One
Many-to-Many

Django's Database

CRUD Operations

URL Mapping, View and Templates

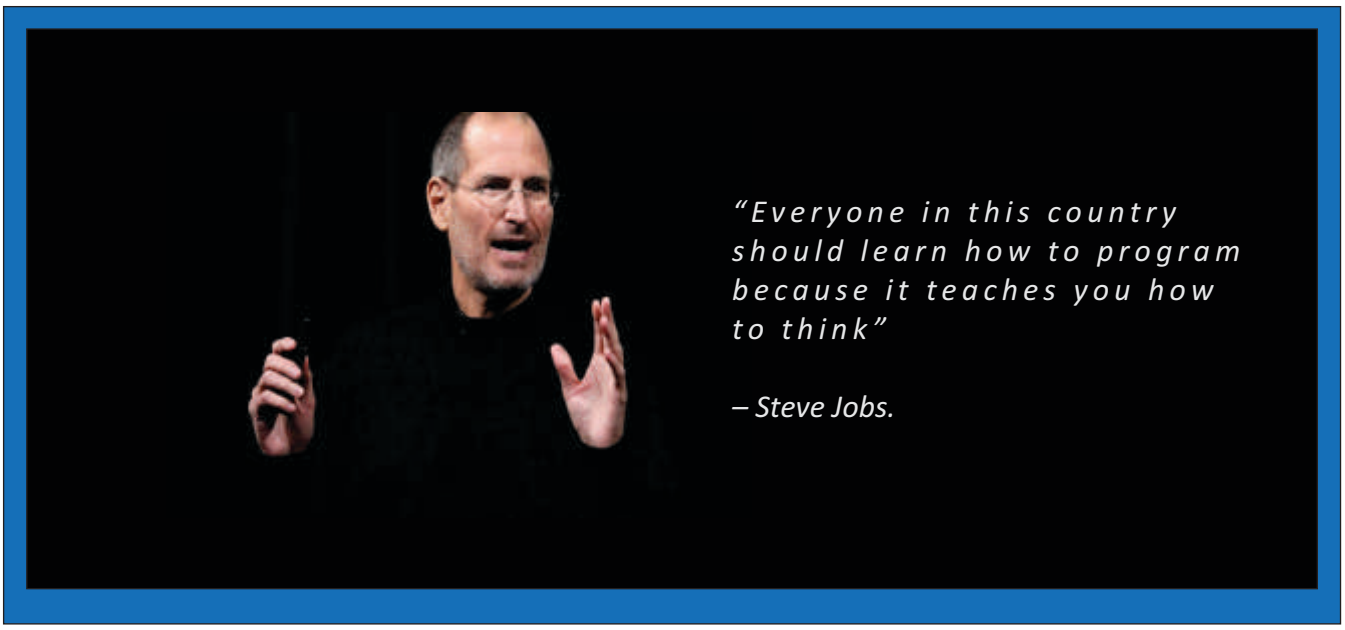
Function Based Views
Class Based Views
URL Configuration
Templates

Django Template Language

Template Variables
Template Inheritances
Template Styling with Bootstrap

Introduction to Django Admin

Introduction
Creating a Superuser Account
CRUD Operations Using Django Admin App
Registering the Model



Advanced Django Admin & Customizations

- Customizing Admin Site -
- Adding Views to the Admin Site

Advanced Templating & Class Based Views

- Template Filters
- Custom Template Filters
- Template Tags
- Django Views
- Class Based Views

Generating CSV PDF and Other Binary Files

- Working with Python's CSV Module
- Working with Excel Files in Python
- Working with PDF files in Python
- Playing with Graphs in Python
- Integrating Visualizations with Django

Testing

- Automation Testing
- Testing in Django
- Testing Django Models
- Testing Django Views
- Django Request Factory
- Test Case Classes in Django

Using Frontend JavaScript Libraries with Django

- JavaScript Frameworks - React and its Components

Top sites & products built with Python & Django

Bitbucket

Udemy

ihub
INNOVATIVE PLACEMENT SOLUTIONS

iHub's KEY CLIENTS

100 + Clients and Adding More

OTHER JOB GUARANTEE COURSES

ADVANCED
DATA SCIENCE & AI

100 % JOB
GAURANTEE

FULL STACK
DIGITAL MARKETING

talend
ETL Course
A Cloud Data Integration

PYTHON
D JANGO

UiPath

Full Stack
MERN
DEVLOPER

Full Stack
MEAN
DEVLOPER

CLOUD
ENGINEER

ihub
INNOVATIVE PLACEMENT SOLUTIONS
www.ihubtalent.com

iHub Talent

#515, Nilgiri Block, Beside Metro Station
Ameerpet, Hyderabad -16.

81069 61963