INDIA'S Finest

JOB GUARANTEED

# Full stack PYTHON BOOTCAMP

For Fresh Graduates
&
For Experienced Professionals

By our Lead Faculty

**Khaja**

Sr. cloud Architect

India's best full stack python boot program from Ihub offer you everything you need to become a master and to grab a job

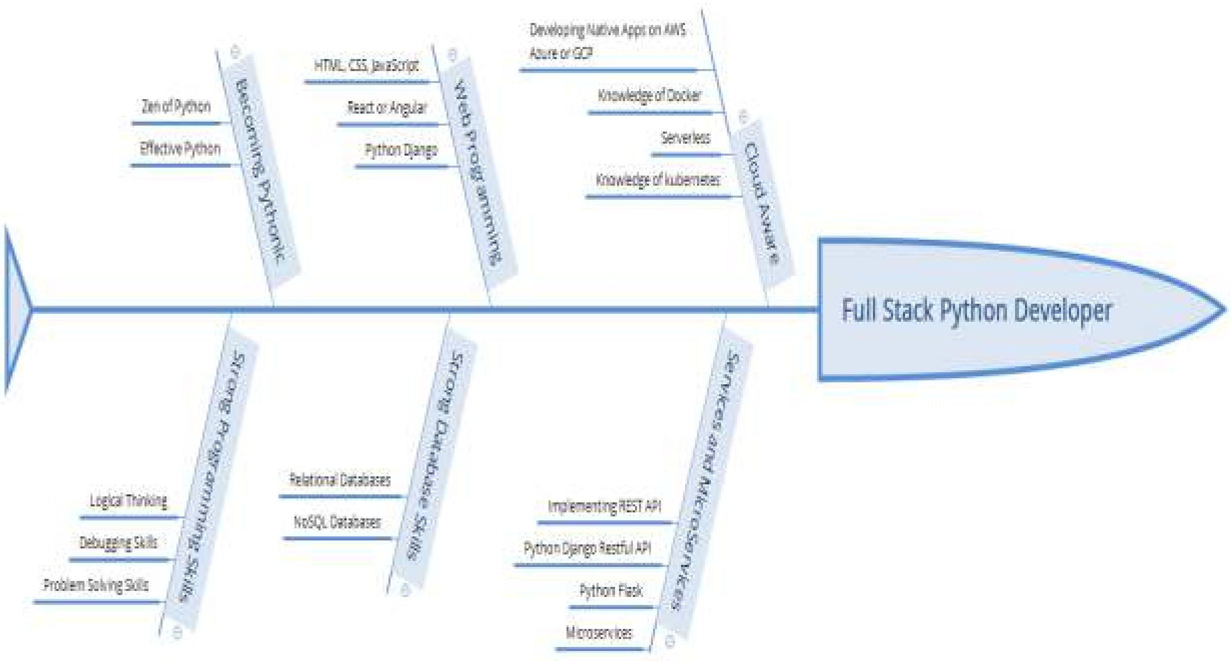
QualityThought®

Learn from Industry Experts

QualityThought®

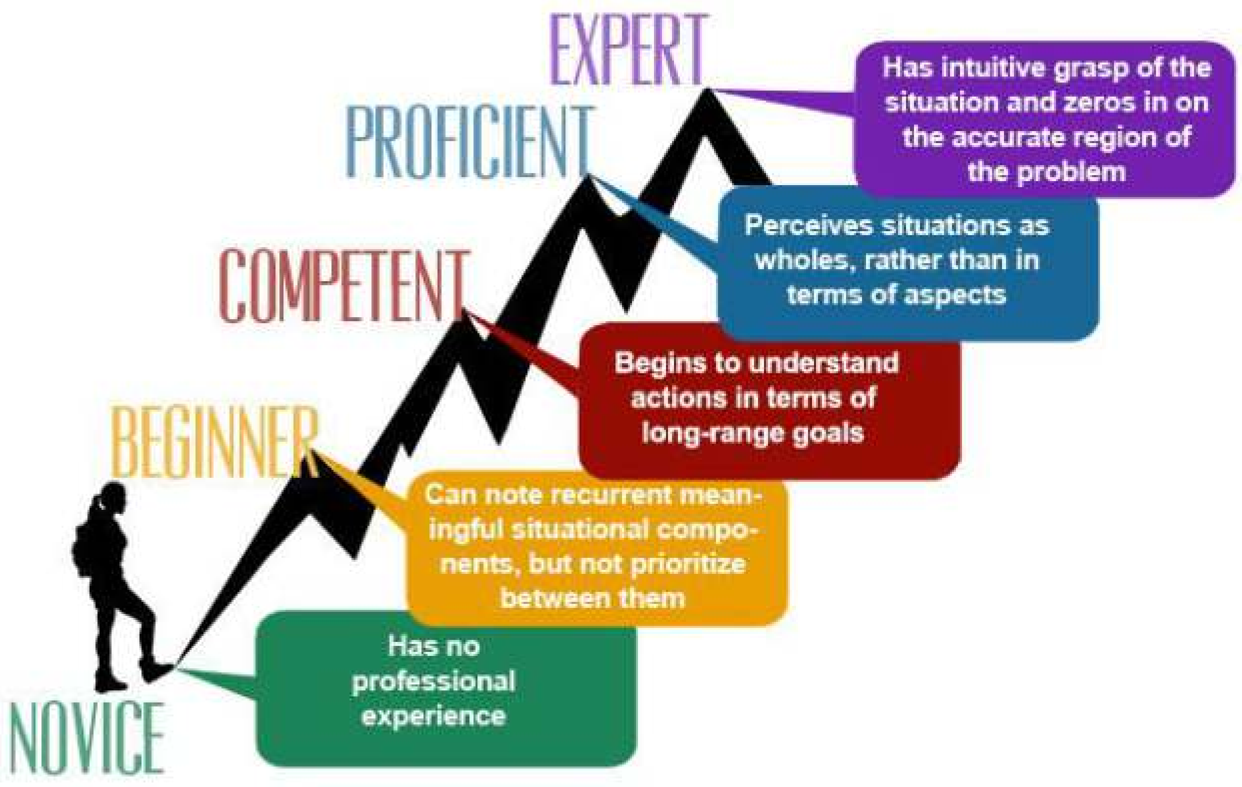
# Python Developer Placement Program

*How to land as Python Developer (Industry Expectations)*



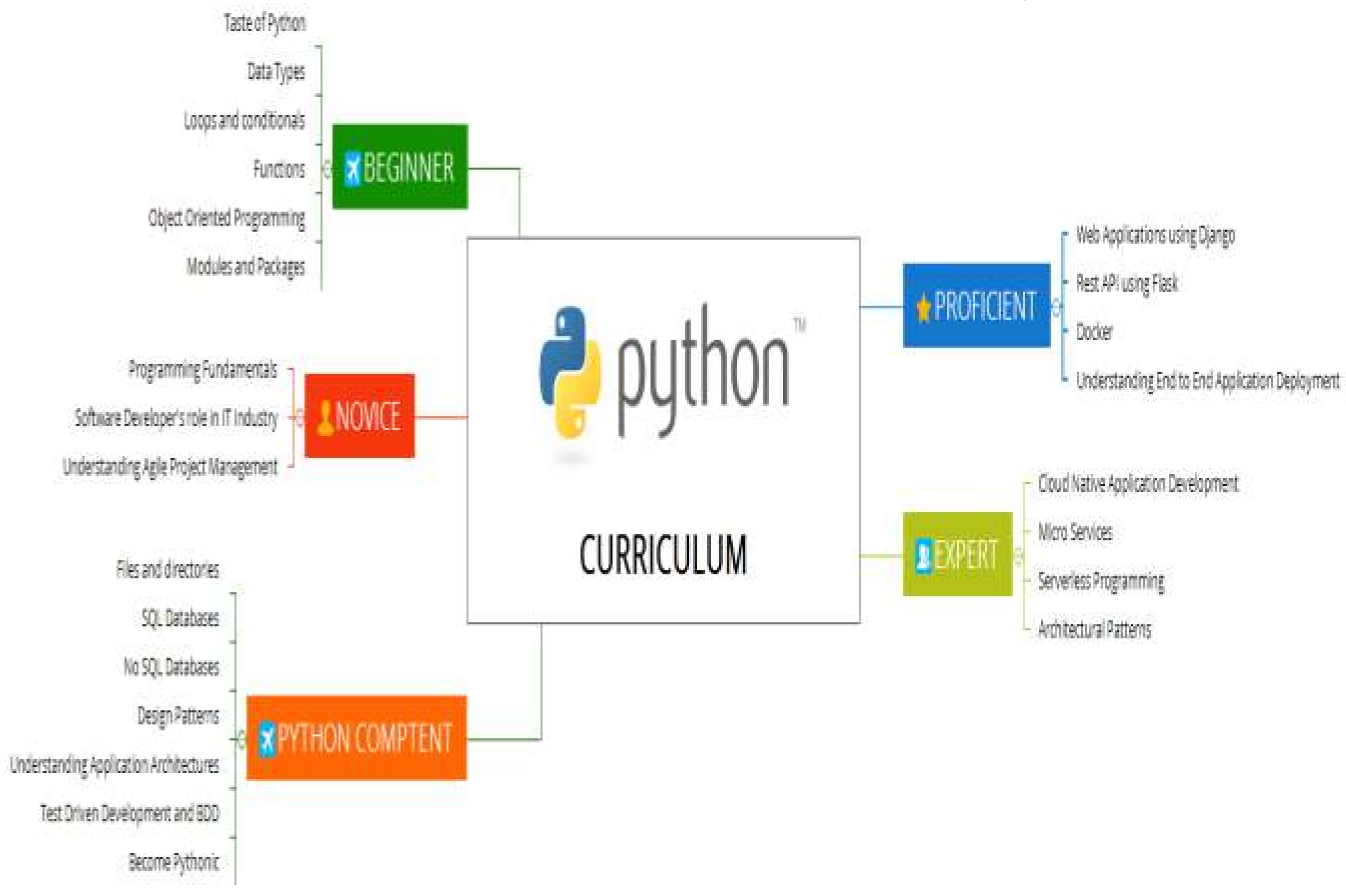
*This Placement Program fills the gaps required to make you the Successful Python Developer*

aiml@qualitythought.in


EXPERT
Has intuitive grasp of the situation and zeros in on the accurate region of the problem
PROFICIENT
Perceives situations as wholes, rather than in terms of aspects
COMPETENT
Begins to understand actions in terms of long-range goals
BEGINNER
Can note recurrent meaningful situational components, but not prioritize between them
NOVICE
Has no professional experience

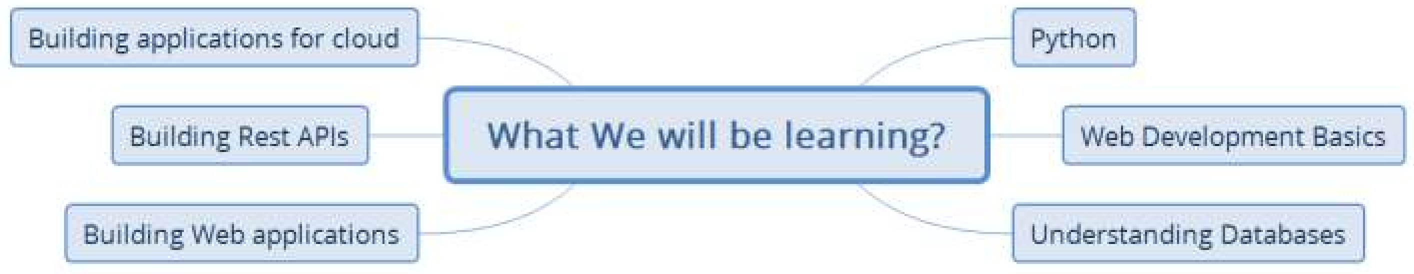
**THIS PROGRAM ASSUMES YOU ARE NOVICE.**

**LET US LOOK INTO OVERVIEW OF WHAT WILL BE THE CURRICULUM OF THIS COURSE.**

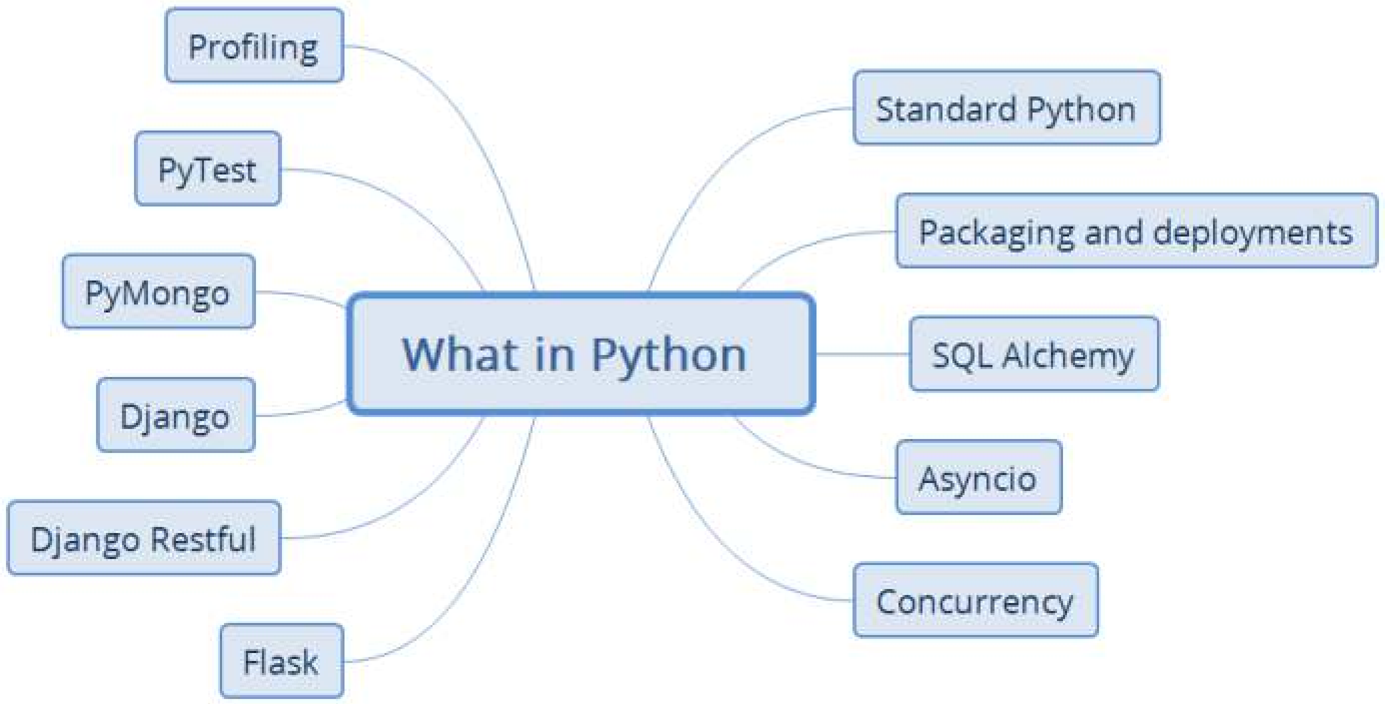


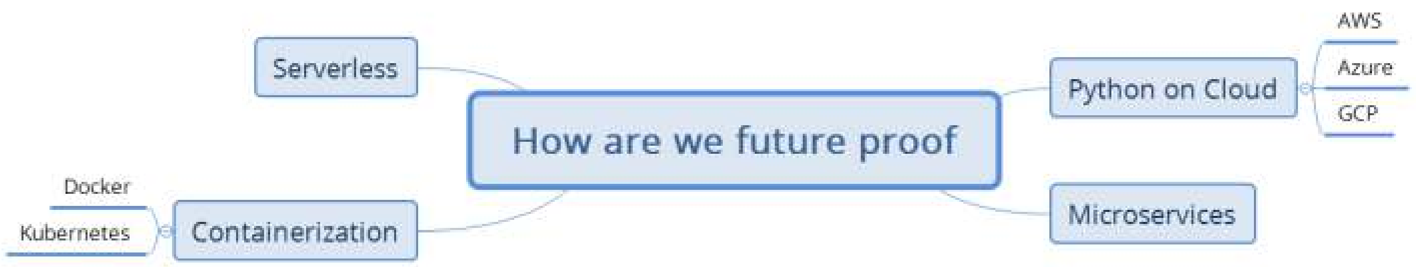
**We have all the tools to make you industry relevant in this age and time.**

**It's your decision which is pending.**


What We will be learning?
Building applications for cloud
Building Rest APIs
Building Web applications
Python
Web Development Basics
Understanding Databases

aiml@qualitythought.in


What in Python
Profiling
PyTest
PyMongo
Django
Django Restful
Flask
Standard Python
Packaging and deployments
SQL Alchemy
Asyncio
Concurrency


How are we future proof
Serverless
Containerization
Docker
Kubernetes
Python on Cloud
AWS
Azure
GCP
Microservices

QualityThought®

# CURRICULUM

## Programming from Absolute Beginning

### Introduction to Computer Programs

QualityThought®

A brief history of computing
A brief history of programming
What is a program?
Understanding the binary system

## Introduction to Programming Languages

Why do we have programming languages?
How programming languages have evolved?
The family tree of programming languages
Translating code into something that the computer understands

*Interpreting*
*Compiling*

Syntax and the building blocks of a programming language

*Keywords*
*Operators*
*Code Blocks*
*Relations to mathematics*

## 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

The importance of sequences

*Defining the problem*
*The solution to the problem*
*Understanding Statements*
*Compound statements*
*Separating statements*

## Working with Data – Variables

- Declaring and initializing variables
- Primitive data types
- Composite type

## Program Control Structures

- Controlling the execution path
  - *Selection statements*
  - *Iteration Statements*
  - *Conditional Statements*
- 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
- Understanding types of software bugs
- Finding bugs using a debugger

## Programming Paradigms

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

QualityThought®

## Programming Tools and Methodologies

Understanding version control systems

Unit testing

Integration testing

Other types of tests

Software releases

Understanding software deployment

*Deployment Automation*

*Code maintenance*

## 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

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*

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*

**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*

*Dictionaries and Sets*

*Dictionaries*

*Create with {}*
*Create with dict(*

***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*
*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*

*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
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

QualityThought®

## 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*
*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*

QualityThought®

## Software Testing and Test-Driven Development

QualityThought®

*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*

**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 testsuites*

**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*

## 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()*

QualityThought®

*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()*

QualityThought®

- - *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

## 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*
- Concurrency
  - *Queues*
  - *Processes*
  - *Threads*
  - *Concurrent.futures*
  - *Green Threads and gevent*
  - *Twisted*
  - *Asyncio*
  - *Redis*
  - *Beyond Queues*

## Persistent Storage

- Flat Text Files
- Padded Text Files
- Tabular Text Files
  - *CSV*
  - *XML*
  - *HTML*
  - *JSON*
  - *YAML*
  - *Tablib*

*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
    - *ZeroMQ*
    - *Other Mesaging 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*

## 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

QualityThought®

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 defaultdict 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

## Robustness and Performance

Take Advantage of Each Block in try/except/else/finally
Consider contextlib and with Statements for Reusable try/finally Behavior
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
*Idealized Computing Versus the Python Virtual Machine*
So Why Use Python?
How to Be a Highly Performant Programmer
*Good Working Practices*

## 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*

## Profiling to Find Bottlenecks

Profiling Efficiently
Introducing the Julia Set

Using Less RAM

**Lessons from the Field**

## 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

QualityThought®

## Introduction to Django

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*

*Primary Keys*

**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 Inheritance*

*Template Styling with Bootstrap*

## *Introduction to Django Admin*

Introduction

Creating a Superuser Account

CRUD Operations Using Django Admin App

Registering the Model

Customizing the Admin Interfaces

## Serving Static Files

Introduction

Static File Finders

AppDirectoriesFinder

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

QualityThought®

## Sessions and Authentication

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

## 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

QualityThought®

## Introduction to Django RESTful Web Services

- Installing the Required Software and Tools
- Creating a virtual environment with Python 3.x and PEP 405
- Installing Django and Django REST frameworks in an isolated environment
- Creating an app with Django
- Installing tools

## Working with Models, Migrations, Serialization and Deserialization

- Working with Models, Migrations, Serialization, and Deserialization
- Defining the requirements for our first RESTful Web Service
- Creating our first model
- Running our initial migration
- Analyzing the database
- Controlling, serialization, and deserialization
- Working with the Django shell and diving deeply into serialization and deserialization

## Creating API Views

- Creating API Views
- Creating Django views combined with serializer classes
- Understanding CRUD operations with Django views and the request methods
- Routing URLs to Django views and functions
- Launching Django's development server
- Making HTTP POST requests with Postman

## Using Generalized Behavior from the APIView Class

- Using Generalized Behavior from the APIView Class
- Taking advantage of model serializers
- Understanding accepted and returned content types
- Making unsupported HTTP OPTIONS requests with command-line tools
- Understanding decorators that work as wrappers
- Using decorators to enable different parsers and renderers
- Taking advantage of content negotiation classes
- Making supported HTTP OPTIONS requests with command-line tools
- Working with different content types
- Sending HTTP requests with unsupported HTTP verbs

## Understanding and Customizing Browsable API Feature

- Understanding and Customizing the Browsable API Feature

Understanding the possibility of rendering text/HTML content
Using a web browser to work with our web service
Making HTTP GET requests with the browsable API
Making HTTP POST requests with the browsable API
Making HTTP PUT requests with the browsable API
Making HTTP OPTIONS requests with the browsable API
Making HTTP DELETE requests with the browsable API

## Working with Advanced Relationships and Serialization

Working with Advanced Relationships and Serialization
Defining the requirements for a complex RESTful Web Service
Creating a new app with Django
Configuring a new web service
Defining many-to-one relationships with models.ForeignKey

## Installing PostgreSQL

Running migrations that generate relationships
Analyzing the database
Configuring serialization and deserialization with relationships
Defining hyperlinks with serializers.HyperlinkedModelSerializer
Working with class-based views
Taking advantage of generic classes and viewsets
Generalizing and mixing behavior
Working with routing and endpoints
Making requests that interact with resources that have relationships

## Using Constraints, Filtering, Searching, Ordering and Pagination

Using Constraints, Filtering, Searching, Ordering, and Pagination
Browsing the API with resources and relationships
Defining unique constraints
Working with unique constraints
Understanding pagination
Configuring pagination classes
Making requests that paginate results
Working with customized pagination classes
Making requests that use customized paginated results
Configuring filter backend classes
Adding filtering, searching, and ordering

Working with different types of Django filters
Making requests that filter results
Composing requests that filter and order results
Making requests that perform starts with searches
Using the browsable API to test pagination, filtering, searching, and ordering

## Securing the API with Authentication and Permissions

Securing the API with Authentication and Permissions
Understanding authentication and permissions in Django, the Django REST framework, and RESTful Web Services
Learning about the authentication classes
Including security and permissions-related data to models
Working with object-level permissions via customized permission classes
Saving information about users that make requests
Setting permission policies
Creating the superuser for Django
Creating a user for Django
Making authenticated requests
Making authenticated HTTP PATCH requests with Postman
Browsing the secured API with the required authentication
Working with token-based authentication
Generating and using tokens

## Applying Throttling Rules and Versioning Management

Applying Throttling Rules and Versioning Management
Understanding the importance of throttling rules
Learning the purpose of the different throttling classes in the Django REST framework
Configuring throttling policies in the Django REST framework
Running tests to check that throttling policies work as expected
Understanding versioning classes
Configuring a versioning scheme
Running tests to check that versioning works as expected

## Automated Tests

Automating Tests
Getting ready for unit testing with pytest
Writing unit tests for a RESTful Web Service
Discovering and running unit tests with pytest
Writing new unit tests to improve the tests' code coverage

QualityThought®

Running unit tests again with pytest

**Building APIs using Flask**

Introduction

Understanding API

RESTful API

*REST Constraints/Principles*

HTTP Protocol

HTTP Methods and CRUD

The JSON Format

HTTP Status Codes

*Commonly used HTTP Status Codes*

Open API

The Flask Web Framework

Building a Simple Recipe Management Application

*Virtual Environment*

Using curl or httpie to Test All the Endpoints

Postman

*The Postman GUI*

*Sending a GET Request*

*Sending a POST Request*

*Saving a Request*

Introduction to Flask

What is Flask-RESTful?

Virtual Environment

Creating a Recipe Model

Configuring Endpoints

Making HTTP Requests to the Flask API using curl and httpie

**Manipulating Database using SQL Alchemy**

Databases

*Database Management System*

SQL

ORM

Defining Our Models

Password Hashing

**Authentication Services and Security with JWT**

JWT

Flask-JWT-Extended

Designing the Methods in the Recipe Model

Refresh Tokens

## Introduction to cloud computing

GET /api/v1/users
GET /api/v1/users/[user_id]
POST /api/v1/users
DELETE /api/v1/users
PUT /api/v1/users

*Building resource tweets methods*

GET /api/v2/tweets
POST /api/v2/tweets
GET /api/v2/tweets/[id]

Testing the RESTful API

*Unit testing*

## Building a Web Application in Python

Getting started with applications

Creating application users

*Working with Observables and AJAX*

*Binding data for the adduser template*

Creating tweets from users

*Working on Observables with AJAX for the addtweet template*

*Data binding for the addtweet template*

CORS - Cross-Origin Resource Sharing

Session management

## Interacting Data Services

*MongoDB - How it is advantageous, and why are we using it?*

*MongoDB terminology*

Setting up MongoDB

*Initializing the MongoDB database*

*Integrating microservices with MongoDB*

*Working with user resources*

*Working with the tweets resources*

## Building WebViews with React

Understanding React

Setting up the React environment

*Installing node*

*Creating package.json*

Building webViews with React

*Integrating webView with microservices*

User authentication

*Login user*

QualityThought®

*Sign up user*

*User profile*

*Log out users*

Testing the React webViews

## Creating UIs to Scale with Flux

Understanding Flux

*Flux concepts*

*Adding dates to UI*

*Building user interfaces to Flux*

*Actions and dispatcher*

Learning Event Sourcing and CQRS (Kafka)

Securing the Web application

Dockerizing Services

Implementing and Deploying on the AWS Platform

Implementing and Deploying on the Azure Platform

Implementing and Deploying on the GCP Platform

Using Python to Implement Serverless on AWS, Azure and GCP