Day 1: Learn the Basics

- Understanding Python basic syntax
- Variables and Data Types
- Conditionals
- Type Casting and Exceptions
- Functions and Built-in Functions
- Lists, Tuples, Sets, and Dictionaries

Questions for Review:

1. What are the basic data types available in Python?
2. How do you perform type conversion between different data types?
3. Can you write an example of a conditional statement in Python?
4. How would you handle an exception in a block of code?
5. How do you define a function with default parameters?
6. Create a list comprehension to generate a list of squares for numbers from 1 to 10.
7. What is the difference between a tuple and a list?
8. How can you add and remove elements from a set?
9. Demonstrate the use of a dictionary with keys and values.
10. Write a Python script that takes input from the user and prints out a message based on the input type.

Day 2: Advanced Topics

- Regular Expressions (Regex)
- Decorators
- Iterators
- Lambdas
- Object-Oriented Programming (OOP): Classes, Inheritance, Methods, and Dunder methods
- Modules: Built-in and Custom
- Package Managers: PyPI, Pip, Conda
- List Comprehension and Generator Expressions
- Different programming paradigms in Python

Questions for Review:

1. How do you use regular expressions in Python for pattern matching?
2. What is a decorator, and provide an example of a decorator that times a function.
3. Explain the difference between an iterator and a generator.
4. Provide an example of a lambda function that sorts a list of tuples by the second value.
5. How do you create a class in Python and what is inheritance?
6. What are magic or dunder methods in Python, and give an example?
7. How do you import a module in Python?
8. What is PEP 8 and why is it important?
9. Write a Python expression using generator expressions that generates even numbers up to 100.
10. Explain the concept of scope in Python.

Day 3: Data Structures and Algorithms

- Understanding arrays and linked lists
- Stacks, queues, and deques
- Hash tables
- Binary search trees
- Recursion
- Sorting algorithms (like quicksort, mergesort, etc.)

Questions for Review:

1. Explain the difference between arrays and linked lists.
2. How would you implement a queue in Python?
3. Describe a real-world application of a hash table.
4. What is a binary search tree and how does it work?
5. Provide an example of a recursive function in Python.
6. Explain the basic idea behind the mergesort algorithm.
7. How does a deque differ from a stack or a queue?
8. What is a binary heap and how is it used in sorting?
9. Describe the concept of memoization in the context of recursion.
10. Implement a simple search algorithm in Python.

Day 4: Testing Your Apps

- `doctest` and `unittest` : Writing test cases
- `nose` and `pytest` : Advanced testing frameworks
- Implementing Continuous Integration (CI) and Continuous Deployment (CD) pipelines

Questions for Review:

1. How do you create a test suite using `unittest` ?
2. What is the benefit of using `pytest` over `unittest` ?
3. Explain the difference between a test case and a test suite.
4. How do you run a `doctest` in Python?
5. What is a test fixture in `unittest` ?
6. How can `pytest` fixtures simplify test setup?
7. What is Continuous Integration, and how does it relate to testing?

8. How do you mock external services in unit tests?
9. Provide an example of a parameterized test with `pytest`.
10. How would you implement a test to check for an expected exception in `unittest`?

Day 5: Data Structures and Algorithms (Continued)

- Heaps, stacks, and queues
- Hash tables
- Binary search trees
- Recursion
- Sorting algorithms (like quicksort, mergesort, etc.)

Questions for Review:

1. How can you use a stack to check for balanced parentheses in an expression?
2. What are the typical operations associated with a heap data structure?
3. How do you handle collisions in a hash table?
4. Write a function that traverses a binary search tree in-order.
5. How would you sort a list of strings using a trie data structure?
6. Illustrate how quicksort algorithm works with an example.
7. What is the Big O notation, and why is it important in the context of algorithms?
8. Can you implement a basic binary search algorithm in Python?
9. What is the difference between iterative and recursive implementations of an algorithm?
10. How would you reverse a linked list, both iteratively and recursively?

Day 6: Working with Databases

- SQL databases: MySQL, PostgreSQL
- ORM: SQLAlchemy, Django ORM
- NoSQL databases: MongoDB, Cassandra
- Interacting with databases through Python

Questions for Review:

1. How do you perform a SELECT query to retrieve data from a SQL database using Python?
2. Explain the difference between a NoSQL and a SQL database.
3. What is an ORM, and how does it simplify database interactions?
4. Write a basic class using SQLAlchemy ORM to model a User entity.
5. How would you handle database migrations in a Python application?
6. Describe a use case for using a NoSQL database like MongoDB.
7. How do you ensure the ACID properties in a database transaction?
8. What is database normalization, and why is it used?
9. How can you prevent SQL injection attacks in Python?
10. Explain the concept of indexing and its impact on database performance.

Day 7: Asynchronous Programming

- Understanding the async/await syntax
- Event loop management
- Asynchronous I/O operations
- Utilizing async frameworks like aiohttp, FastAPI

Questions for Review:

1. What is the difference between synchronous and asynchronous execution?
2. How do you define an asynchronous function in Python?
3. What is an event loop, and how does it work in the context of async programming?
4. Provide an example of using the `asyncio` library for handling a non-blocking I/O operation.
5. How does the `await` keyword work in Python?
6. What are coroutines, and how are they used in asynchronous programming?
7. Describe how backpressure is handled in an asynchronous system.
8. Explain how an asynchronous web framework like FastAPI can improve web application performance.
9. How do you run multiple coroutines concurrently using `asyncio`?
10. What challenges might you encounter when using asynchronous programming, and how can you mitigate them?

Day 8: Version Control Systems

- Understanding Git
- Working with GitHub or other remote repositories
- Branching, Merging, and Pull Requests
- Collaborative development and resolving conflicts

Questions for Review:

1. How do you clone a repository from GitHub using Git?
2. What is the purpose of branching in version control, and how do you create a new branch in Git?
3. Explain the difference between a fast-forward merge and a three-way merge.
4. How would you resolve a merge conflict in a file using Git?
5. Describe the typical workflow of making a pull request on GitHub.
6. What are some best practices for commit messages?
7. How can you revert a commit that has already been pushed to a remote repository?
8. What are Git hooks, and give an example of how they can be used?
9. Describe how you can synchronize your local repository with changes from a remote branch.
10. What is a rebase, and when might you prefer it over a merge?

Day 9: Testing and Documentation

- Writing unit tests with unittest, pytest
- Integration and system testing
- Documentation standards like Docstrings
- ReadTheDocs and markdown for documentation

Questions for Review:

1. How do you write a basic unit test in Python using pytest?
2. What is the difference between unit testing, integration testing, and system testing?
3. Explain the purpose of test fixtures and how you use them in pytest.
4. How would you mock an external API call in your unit tests?
5. Describe the significance of code coverage in testing.
6. What are Python docstrings, and how do you use them to document a function?
7. Explain the process of generating HTML documentation from docstrings in Python.
8. What is continuous integration, and how does testing fit into it?
9. Describe a scenario where you would use a markdown file for documentation.
10. How do you ensure that your code and tests are maintainable and readable by others?

Day 10: Deployment and Continuous Integration/Continuous Deployment (CI/CD)

- Docker containers and orchestration with Kubernetes
- Deploying Python applications to cloud platforms (AWS, GCP, Azure)
- GitHub Actions or GitLab for CI/CD pipelines
- Monitoring and logging

Questions for Review:

1. How would you containerize a Python web application using Docker?
2. What is Kubernetes, and why is it used for container orchestration?
3. Describe the benefits of using a cloud platform like AWS for deploying Python applications.
4. What are the steps to set up a basic CI/CD pipeline using GitHub Actions?
5. Explain the concept of Infrastructure as Code and mention a tool that can be used for it.
6. How would you monitor the health of a live Python application?
7. What are some common indicators you might use for application logging?
8. Describe the process of scaling a Python application in the cloud.
9. What are the advantages of using a load balancer with a Python web application?
10. How can you ensure zero-downtime deployment for a critical Python application?