Complete Python Interview Questions Guide

1. Python Basics

Core Concepts

- **What is Python?** - Interpreted, high-level, general-purpose programming language
- **Python features** - Simple syntax, dynamic typing, interpreted, object-oriented, extensive libraries
- **Python vs other languages** - Comparison with Java, C++, JavaScript
- **PEP 8** - Python style guide and coding standards
- **Python versions** - Differences between Python 2 and Python 3

Data Types & Variables

- **Basic data types** - int, float, string, boolean, None
- **Mutable vs Immutable** - Lists vs tuples, strings, numbers
- **Type conversion** - Implicit and explicit type casting
- **Variable scoping** - Local, global, nonlocal, LEGB rule

2. Data Structures

Built-in Data Structures

- **Lists** - Creation, indexing, slicing, methods (append, extend, pop, remove)
- **Tuples** - Immutable sequences, when to use vs lists
- **Dictionaries** - Key-value pairs, methods (get, keys, values, items)
- **Sets** - Unique elements, operations (union, intersection, difference)
- **Strings** - Manipulation, formatting, methods

Advanced Operations

- **List comprehensions** - Syntax, nested comprehensions, filtering
- **Dictionary comprehensions** - Creating dicts efficiently
- **Set comprehensions** - Unique element generation

3. Control Flow

Conditional Statements

- **if/elif/else** - Basic and nested conditions
- **Ternary operator** - `x if condition else y`

- **Boolean operations** - and, or, not

Loops

- **for loops** - Iterating over sequences, range(), enumerate(), zip()
- **while loops** - Condition-based iteration
- **Loop control** - break, continue, else clause
- **Nested loops** - Performance considerations

4. Functions

Function Basics

- **Function definition** - def keyword, parameters, return values
- **Arguments** - Positional, keyword, default parameters
- **Variable arguments** - *args and **kwargs
- **Scope** - Local vs global variables, global keyword

Advanced Function Concepts

- **Lambda functions** - Anonymous functions, use with map, filter, reduce
- **Decorators** - Function modification, @property, @staticmethod, @classmethod
- **Generators** - yield keyword, memory efficiency
- **Closures** - Nested functions accessing outer scope

5. Object-Oriented Programming

Classes and Objects

- **Class definition** - Attributes, methods, constructor (**init**)
- **Instance vs class variables** - Scope and usage
- **Method types** - Instance, static, class methods
- **Special methods** - **str**, **repr**, **len**, **eq**

OOP Principles

- **Inheritance** - Single, multiple inheritance, super()
- **Encapsulation** - Private/protected members, property decorators
- **Polymorphism** - Method overriding, duck typing
- **Abstraction** - Abstract base classes

6. Exception Handling

Error Management

- **try/except blocks** - Catching specific exceptions
- **finally clause** - Cleanup code
- **else clause** - Code that runs when no exception occurs
- **Custom exceptions** - Creating your own exception classes
- **Common exceptions** - ValueError, TypeError, IndexError, KeyError

7. File Handling

File Operations

- **Opening files** - Different modes (r, w, a, rb, wb)
- **Reading files** - read(), readline(), readlines()
- **Writing files** - write(), writelines()
- **Context managers** - with statement for automatic cleanup
- **File paths** - os.path, pathlib

8. Libraries and Modules

Module System

- **import statements** - Different ways to import
- **Package structure** - **init**.py, relative imports
- **Standard library** - Common modules (os, sys, datetime, json, re)

Popular Libraries

- **NumPy** - Arrays, mathematical operations
- **Pandas** - Data manipulation, DataFrames
- **Requests** - HTTP requests
- **JSON handling** - Parsing and generating JSON
- **Regular expressions** - Pattern matching with re module

9. Advanced Python Concepts

Memory Management

- **Garbage collection** - Reference counting, cyclic references
- **Memory optimization** - **slots**, generators vs lists
- **Shallow vs deep copy** - copy.copy() vs copy.deepcopy()

Iterators and Generators

- **Iterator protocol** - **iter** and **next** methods
- **Generator functions** - yield, generator expressions
- **Itertools module** - Advanced iteration tools

Concurrency

- **Threading** - GIL limitations, threading module
- **Multiprocessing** - True parallelism, process pools
- **Asyncio** - Asynchronous programming, coroutines

10. Common Coding Questions

Algorithmic Problems

- **Fibonacci sequence** - Recursive and iterative approaches
- **Palindrome check** - String manipulation
- **FizzBuzz** - Conditional logic
- **Prime numbers** - Number theory algorithms
- **Sorting algorithms** - Bubble sort, quick sort implementation

Data Structure Problems

- **Reverse a list/string** - Multiple approaches
- **Find duplicates** - Using sets, dictionaries
- **Two sum problem** - Dictionary lookup technique
- **Merge sorted lists** - Algorithm implementation
- **Binary search** - Efficient searching

String Manipulation

- **Anagram detection** - Character frequency counting
- **String compression** - Algorithm design
- **Longest substring** - Sliding window technique
- **Word frequency** - Dictionary usage

11. Python-Specific Questions

Language Features

- **Duck typing** - "If it walks like a duck..."

- **Monkey patching** - Dynamic attribute modification
- **List vs tuple performance** - When to use which
- **Dictionary implementation** - Hash tables, collision handling
- **Python's GIL** - Global Interpreter Lock implications

Best Practices

- **Code organization** - PEP 8, naming conventions
- **Error handling strategies** - When to catch exceptions
- **Performance optimization** - Profiling, algorithmic improvements
- **Testing** - Unit tests, docstrings, assertions

12. Debugging and Testing

Debugging Techniques

- **Print debugging** - Strategic print statements
- **Python debugger** - pdb module usage
- **IDE debugging** - Breakpoints, variable inspection
- **Logging** - Using logging module vs print

Testing

- **Unit testing** - unittest module, test structure
- **Test-driven development** - Writing tests first
- **Mocking** - unittest.mock for testing dependencies
- **Docstrings** - Documentation and simple tests

13. Web Development (if applicable)

Frameworks

- **Flask** - Lightweight web framework basics
- **Django** - MVC pattern, ORM, templates
- **FastAPI** - Modern API development
- **REST APIs** - HTTP methods, status codes

14. Database Integration

Database Concepts

- **SQL basics** - SELECT, INSERT, UPDATE, DELETE

- **Database connectivity** - sqlite3, SQLAlchemy
- **ORM concepts** - Object-relational mapping
- **Database design** - Normalization, relationships

Sample Interview Questions by Category

Beginner Level

1. Explain the difference between lists and tuples
2. How do you reverse a string in Python?
3. What is the difference between `==` and `is`?
4. How do you handle exceptions in Python?
5. What are lambda functions?

Intermediate Level

1. Explain decorators with an example
2. What is the difference between deep copy and shallow copy?
3. How does Python's garbage collection work?
4. Implement a function to find the second largest number in a list
5. What are generators and why are they useful?

Advanced Level

1. Explain the GIL and its implications
2. How would you implement a singleton pattern in Python?
3. Design a caching decorator
4. Explain metaclasses and when you might use them
5. How would you optimize a slow Python program?

Tips for Interview Success

Preparation Strategy

- Practice coding problems on platforms like LeetCode, HackerRank
- Review your own projects and be ready to discuss them
- Understand time and space complexity of your solutions
- Practice explaining concepts clearly and concisely

During the Interview

- Think out loud while solving problems

- Ask clarifying questions about requirements
- Start with a simple solution, then optimize
- Test your code with examples
- Discuss trade-offs of different approaches

Common Mistakes to Avoid

- Not handling edge cases
- Writing overly complex solutions initially
- Not testing your code
- Poor variable naming
- Not considering time/space complexity