Python Topics and Questions

Data Types

Theory

- What are the different data types in Python?
- What is the difference between list and tuple?
- Are integers and floats objects in Python?
- Is an integer mutable?
- What are the properties of sets, tuples, and dictionaries?
- What are falsy values in Python?
- What are the syntaxes to create lists, tuples, and sets properly (which bracket is used for each)?
- What are the datatypes of dictionary keys?
- What is NoneType?
- What is the difference between `is` and `==`?
- What is the `id()` function and object identity in Python?
- Are True and False objects in Python?
- What is dynamic typing?
- What is pass by assignment?
- What is call by sharing?
- What is the difference between a dictionary and a set?

Practical

- Learn the syntax of each data type, functions, and class to initialize and operate.
- Access values from each data type, update them, split, and merge lists, strings, and dictionaries.
- Identify data types.
- Convert a list to a linked list.
- Convert a list of tuples into a dictionary.
- Unpack a string.
- Unpack a list into 3 variables.
- Combine two lists using the splat operator.
- Create a nested dictionary.
- Split a dictionary into two equal pieces.
- Remove odd values from a dictionary.
- Remove non-string values from a dictionary.
- Add/remove keys in a dictionary.
- Check the existence of a key in a dictionary.
- Loop through a dictionary and print key and value pairs.
- Transform a list of dictionaries into a dictionary.

- Get JSON data at a URL as a Python dictionary.

Object-Oriented Programming (OOP)

Theory

- What are the core OOP concepts in Python?
- What is inheritance in Python?
- What is encapsulation in Python?
- What is abstraction in Python?
- What is polymorphism in Python?
- What is method overloading in Python?
- What is method overriding in Python?
- What is operator overloading?
- What is the purpose of `self` in Python classes?
- What is the difference between `__init__` and `__new__`?
- What is the Method Resolution Order (MRO)?
- What are access specifiers in Python?
- Why are private properties created?
- What is the purpose of a destructor?
- What is the difference between a class method, static method, and instance method?
- What is the `super()` function in Python?
- What is multiple inheritance in Python?
- What is duck typing in Python?
- What is the role of the `__str__` vs `__repr__` methods?
- What are access modifiers in Python?
- What is the purpose of the ABC (Abstract Base Class) in Python?

Practical

- Create a class with a constructor and attributes.
- Create a class with 2 attributes.
- Create a Bank class with 2 attributes and 3 methods (deposit, withdraw, balance check).
- Implement a class with 2 attributes.
- Create a Calculator class with an `__init__` method that initializes two values and a sum method to return their sum.
- Save constructor arguments as attributes.
- Access parent class properties.
- Practice programs with OOP concepts (creating classes, objects, methods, etc.).
- Demonstrate method overloading and overriding practically.

Functions

Theory

- What are pure functions and referential transparency?
- What are `*args` and `**kwargs` in Python?
- What are the types of arguments in Python?
- What is a lambda function, and how many arguments can it have?
- What is the least number of arguments a lambda function can have?
- Are functions objects in Python?
- What is a variadic function?
- What is the difference between `map()` and other functions like `filter()`?
- What is a closure in Python?
- What is currying in Python?

Practical

- Create a lambda function to generate square numbers using the math module.
- Create a lambda function to capitalize the first letter of a string.
- Create a lambda function to return the last element of a list.
- Create a lambda function to create a full address from dictionaries with optional fields.
- Create a lambda function to return random numbers between 0 and 100.
- Create a function that can only be executed once (throw errors on subsequent calls).
- Create a variadic function to return the sum and average of arguments.
- Use `map()` on a dictionary.
- Use `map()` to create a new list from an existing list with a lambda function.
- Create a program to get key-value pairs (dictionary) from a string where the key is an element of the string and the value is its count.
- Create a function to yield unique random numbers from a given list.
- Create a generator to count up to n numbers.
- Create a generator to yield random numbers between 0 and 50.
- Create a generator to yield prime numbers.
- Create a generator to yield random items from an iterable.
- Create a generator to generate multiples of a given number.

Decorators

Theory

- What are decorators in Python?
- What is the purpose of decorators with and without arguments?
- How do decorators work logically?

Practical

- Create a custom decorator.
- Create a decorator that can access a function with arguments.

Exception Handling

Theory

- What is exception handling in Python?
- What is the syntax for exception handling?
- What is the use of `else` in exception handling?
- What are the types of exceptions in Python?
- Why should you avoid inheritance from `BaseException`?
- What is the purpose of catching specific error types?

Practical

- Catch specific errors (avoid empty `except:` blocks, use at least `except Exception:`).
- Raise custom exceptions.
- Try-except with specific error types.

Context Managers

Theory

- What is a context manager in Python?
- What is the use of the `with` statement?
- How do context managers work?

Practical

- Use the `with` statement for file operations.
- Implement a custom context manager.

Generators and Iterators

Theory

- What are generators and how do they differ from iterators?
- What are the advantages of generators?
- How do you signal the end of iteration in an iterator?

Practical

- Create a generator to count up to n numbers.
- Create a generator to yield random numbers between 0 and 50.
- Create a generator to yield prime numbers.
- Create a generator to yield random items from an iterable.
- Create a generator to generate multiples of a given number.
- Implement a generator function.

List Comprehension

Theory

- What is list comprehension in Python?

Practical

- Create a list comprehension to filter strings with a length < 5.
- Create a list comprehension to extract strings.
- Create a list comprehension with an if-else condition (square even numbers and cube odd numbers).
- Create a list comprehension to filter dictionaries that don’t have a specific key.
- Create a list comprehension with a condition.

Dictionary Comprehension

Theory

- What is dictionary comprehension in Python?

Practical

- Create a dictionary comprehension.
- Create a dictionary comprehension with an if-else condition (square even and cube odd numbers).
- Merge two dictionaries, adding values if the same key is present.
- Remove keys corresponding to the highest value in a dictionary.
- Remove dictionaries that don’t have a specific key.
- Remove non-string values from a dictionary.
- Remove odd values from a dictionary.
- Add/remove keys in a dictionary.
- Split a dictionary into two equal pieces.

File Operations

Theory

- What are the different file modes in Python (e.g., append mode)?
- What is the `open()` function in Python?
- How do you close a file handle?
- What is the purpose of the `with` statement in file operations?

Practical

- Open a text file and write a value to it.
- Perform file operations (open, read/write, modes).

Modules and Packages

Theory

- What is the difference between a module and a package?
- What is the mandatory module in a package?
- What can be written in `__init__.py`?
- What is the purpose of `__name__`?
- What is the difference between `import` and `import as`?
- Is it possible to import inside functions?
- What is the purpose of `pip`?
- What is PEP8?
- What are PEPs?

Practical

- Create a package and understand `__init__.py`.
- Write import statements at the top of a file.

Memory Management

Theory

- What is the Global Interpreter Lock (GIL)?
- Why is the GIL needed?
- What is generational memory management in Python?
- What is reference counting in Python?

- What is the Python memory model (heap/stack/GC)?
- How does Python manage memory?
- What is the difference between `.py` and `.pyc` files?
- What is `__pycache__`?
- What is manual memory management in Python?
- What is the garbage collection mechanism in Python?

Practical

- Understand `.pyc` files and `__pycache__`.

Metaclasses

Theory

- What is a metaclass in Python?
- What is the difference between a metaclass and Django's Meta class?
- What is the purpose of a metaclass?

Practical

- Implement a metaclass.
- Compare a metaclass with Django's ORM Meta class.

Pickling

Theory

- What is pickling in Python?
- Why use pickling?
- Can numbers be pickled?

Practical

- Perform pickling operations.

Environment Variables

Theory

- What are environment variables?
- Where are environment variables stored (not in `.env` files)?

- How do you read environment variables without using `dotenv`?

Practical

- Read environment variables.
- Access environment variables without `dotenv`.

Lambda Functions

Theory

- What is a lambda function in Python?
- How many expressions can be used in a lambda function?
- When to use a lambda function?

Practical

- Create a lambda function to generate square numbers using the math module.
- Create a lambda function to capitalize the first letter of a string.
- Create a lambda function to return the last element of a list.
- Create a lambda function to create a full address from dictionaries with optional fields.
- Create a lambda function to return random numbers between 0 and 100.

String Operations

Theory

- Are strings mutable or immutable?
- What is string slicing?
- What is the `re` package in Python?
- What is the purpose of f传来

Practical

- Reverse a string.
- Find the longest word in a sentence.
- Check whether a given string is a palindrome.
- Remove all whitespace characters in a string.
- Count the occurrence of each character in a string.
- Replace a character in a string without using the `.replace()` method.
- Convert a string to a list.
- Unpack a string.

- Store f-strings in a variable.
- Concatenate two object values using an f-string.

Pattern Problems

Practical

- Create a pyramid pattern.
- Create an inverted pyramid pattern.

Date and Time Handling

Theory

- How do you parse a string to a `datetime` object?
- What is `timedelta` in Python?

Practical

- Parse a date (string to `datetime`).
- Calculate the date 7 days ago.
- Calculate the date 45 days ago.
- Calculate the hours since New Year.
- Print the time without the date.

Miscellaneous

Theory

- What is the Python Virtual Machine (VM)?
- What are the advantages of CPython?
- What is the `eval()` function, and why should it be avoided?
- What is the walrus operator (`:=`)?
- What is the difference between `for` and `while` loops?
- What is the purpose of the `global` statement?
- What is the difference between `del` and `pop`?
- What is the largest integer in Python?
- What is the `match` statementintroduced in Python 3.10+?
- What is the `subprocess` module?
- What is bit shifting in Python?
- What is the difference between a thread and a process?

- What is the recommended indentation level in Python?
- What are docstrings?
- What is type annotation in Python?
- What is `xrange` in Python?
- What is the difference between `list.sort()` and `sorted()`?
- What are the HTTP status codes 401, 404, and 403?
- What is the purpose of WSGI?
- What are session-based authentication and CSRF in Django?

Practical

- Create a 6-digit OTP using `random`.
- Find the sum of numbers from an alphanumeric list.
- Filter odd numbers from a list.
- Find the second largest element in a list without modifying the list.
- Find the largest element in a list without modifying the list.
- Find the first non-repeating character in a string.
- Check if a number is prime.
- Calculate the factorial of a number.
- Remove duplicate elements from an array.
- Check if two strings are anagrams.
- Return the number of vowels in a string.
- Find common elements between two arrays.
- Reverse a list without using the `reverse()` method.
- Perform list sorting using a loop.
- Multiply two lists.
- Use the `all()` and `any()` built-ins.
- Throw an error if all arguments are None (defensive programming).
- Swap the value of two variables.
- Use a one-liner if-else statement.
- Use the walrus operator.
- Combine two lists.
- Crop an image (functionality).

Django-Specific

Theory

- What is the MVT (Model-View-Template) architecture in Django?
- What is the purpose and functionality of Django Forms?
- What are the advantages of ORM over raw SQL?
- What are the types of model inheritance in Django?
- What are `F` and `Q` objects in Django?

- What is `unique_together` in Django?
- What is the difference between `values()` and `values_list()` in Django?
- What are custom managers in Django?

Practical

- Write Django queries to find the average salary and fetch employees with salaries greater than a value.
- Implement relationships (one-to-one, one-to-many, etc.) in Django models.