🔹 Section 1: Variables and Data Types

✅ Interview Questions

1. What is a variable in Python?
    - A variable in Python is a name that refers to a value stored in memory. It doesn't require explicit declaration to reserve memory space.
2. How does Python handle variable declaration?
    - Python is dynamically typed, meaning you don't need to declare the data type of a variable. It's inferred at runtime.
3. What are the basic data types in Python?
    - int, float, bool, str, list, tuple, set, dict, and NoneType.
4. What is the difference between mutable and immutable data types?
    - Mutable: Can be changed after creation (e.g., list, dict, set).
    - Immutable: Cannot be changed after creation (e.g., int, float, string, tuple).
5. What is typecasting in Python?
    - Converting one data type to another using functions like int(), float(), str(), etc.
6. How do you check the type of a variable?
    - Using the type() function. Example: type(x)
7. What is the difference between is and == in Python?
    - == checks value equality, is checks identity (memory location).

🔹 Section 2: Operators in Python

✅ Interview Questions

1. What are operators in Python?
    - Operators are special symbols or keywords used to perform operations on variables and values.
2. List different types of operators in Python.
    - Arithmetic Operators
    - Comparison (Relational) Operators
    - Logical Operators
    - Assignment Operators
    - Bitwise Operators
    - Identity Operators
    - Membership Operators
3. What is the difference between = and == in Python?
    - = is an assignment operator (assigns value).
    - == is a comparison operator (checks equality of values).
4. Explain arithmetic operators with examples.
    - +, -, *, /, //, %, **

      ```
      10 + 2 = 12
      10 ** 2 = 100
      10 // 3 = 3  # Floor division
      ```
5. What is the difference between / and // operators?
    - / returns float division
    - // returns floor division (integer part only)
6. What are logical operators in Python?
    - and, or, not — used to combine conditional statements

True and False → False
not True → False

7. Explain the difference between is and ==.
    - is checks if two variables refer to the same object (identity)
    - == checks if the values are equal
8. What are membership operators in Python?
    - in and not in — used to check if a value exists in a sequence

      "a" in "apple" → True
      3 not in [1, 2, 4] → True
9. What are identity operators in Python?
    - is and is not — used to compare memory locations of two objects.
10. What are bitwise operators? Name them.
    - Bitwise operators perform operations on binary digits:
        - & (AND), | (OR), ^ (XOR), ~ (NOT), << (left shift), >> (right shift)
11. Explain the working of the not operator.
    - It returns the opposite boolean value:

      not True → False
      not 0 → True
12. Can operators be overloaded in Python?
    - Yes, Python allows operator overloading using special methods like __add__, __eq__, etc.
13. What is operator precedence in Python?
    - It determines the order in which operations are evaluated. Example:

      result = 10 + 3 * 2 # 10 + 6 = 16
14. What is associativity in Python operators?

- It determines the direction in which an expression is evaluated when operators have the same precedence (left-to-right or right-to-left).

🔹 Section 3: Control Flow (if, elif, else, loops)

✅ Interview Questions

1. What is control flow in Python?
   - Control flow refers to the order in which the code is executed, especially when using conditions and loops. It allows decisions and repetition in programs.
2. What are the different control flow statements in Python?
   - Conditional Statements: if, elif, else
   - Loops: for, while
   - Loop Control: break, continue, pass
3. Explain the use of if, elif, and else with syntax.

```
if condition:
    # code block
elif another_condition:
    # another code block
else:
    # fallback code block
```

4. Can elif be skipped in an if-elif-else ladder?
   - Yes, you can use if and else without elif, or just if alone.
5. What is the difference between = and == in an if condition?
   - = is an assignment operator; == is used to compare values in conditions.
6. What is the difference between for and while loops?

- for is used when the number of iterations is known (looping over sequences).
- while is used when the loop should continue until a condition becomes false.

7. What are break and continue statements?
   - break terminates the loop entirely.
   - continue skips the current iteration and continues with the next one.
8. What is the use of pass statement?
   - pass is a null operation; it's used as a placeholder when a statement is syntactically required but no code is needed.
9. What is the purpose of the else clause with loops in Python?
   - The else block after a loop runs only if the loop is not terminated by a break.
     Example:

```
for i in range(3):
    print(i)
else:
    print("Completed without break")
```

10. Can you use else with if without using elif?
    - Yes. elif is optional. You can use just if and else.
11. Can you nest if statements in Python?
    - Yes, nested if statements are allowed.
      Example:

```
if age > 18:
    if age < 60:
        print("Eligible")
```

12. What happens if the condition in an if statement is not a boolean?
    - Python evaluates the condition using truthy or falsy rules:
      - Falsy values: 0, '', [], {}, None
      - Truthy: Everything else

13. How do break and continue behave in nested loops?
    - break and continue only affect the innermost loop they are written in.
14. What are infinite loops and how can they be avoided?
    - A loop that never ends due to a condition that never becomes false.
    - Always make sure loop conditions eventually fail.
15. What is a common use of the range() function in loops?
    - It is commonly used with for loops to generate a sequence of numbers:

```
for i in range(5):
    print(i)
```

🔹 Section 4: Loops in Python

✅ Interview Questions

1. What are the types of loops in Python?
    - Python supports two main types of loops:
        - for loop
        - while loop
2. How does a for loop work in Python?
    - A for loop iterates over a sequence (like a list, tuple, string, or range) one element at a time.
      Example:

```
for i in range(3):
    print(i)
```

3. How does a while loop work in Python?

- A while loop runs as long as the condition is True.
  Example:

```
i = 0
while i < 3:
    print(i)
    i += 1
```

4. What is the purpose of the break statement in loops?
   - break is used to exit the loop prematurely when a certain condition is met.
5. What is the use of the continue statement in loops?
   - continue skips the rest of the code inside the loop for the current iteration and jumps to the next iteration.
6. What is the purpose of the else clause with loops?
   - The else clause runs after the loop completes, only if the loop wasn't terminated by a break.
     Example:

```
for i in range(3):
    print(i)
else:
    print("Loop completed")
```

7. What is an infinite loop? How do you prevent it?
   - A loop that never ends due to a condition that always remains true.
   - Ensure that the loop's condition will eventually become false.
8. What are nested loops? Provide an example.
   - Loops inside loops. Useful for multidimensional data or combinations.
     Example:

```
\
for i in range(2):
    for j in range(2):
        print(i, j)
```

9. How can you loop through a string or list in Python?

- **Using a for loop:**

```
for char in "Python":
    print(char)
```

10. What is the range() function? How is it used in loops?

- Generates a sequence of numbers. Often used with for loops.

Example:

```
for i in range(1, 6, 2):  # Start=1, Stop=6, Step=2
  print(i)  # Output: 1, 3, 5
```

11. Can you use else with a while loop as well?

- Yes. Same as for loop — the else block runs if the loop terminates normally (not by break).

12. What is the difference between for and while loop usage?

- Use for when you know the number of iterations.
- Use while when iterations depend on a condition.

13. What happens if break is used inside nested loops?

- It only breaks the **innermost** loop in which it's written.

14. Can a for loop iterate over dictionaries or sets?

- Yes. In dictionaries, it iterates over keys by default:

```
for key in my_dict:
  print(key, my_dict[key])
```

15. What's the role of pass inside loops?

- It acts as a placeholder when no action is needed but syntactically a statement is required.

🔹 Section 5: Functions in Python

✅ Interview Questions

1. What is a function in Python?
    - A function is a reusable block of code that performs a specific task and can be executed when called.
2. What is the difference between a built-in function and a user-defined function?
    - Built-in: Already available in Python (e.g., len(), print(), type())
    - User-defined: Functions created by the programmer using the def keyword.
3. How do you define a function in Python?
    - Using the def keyword:

```
def greet(name):
    print("Hello", name)
```

4. What is the purpose of the return statement in a function?
    - It ends the function execution and optionally sends back a value to the caller.
5. What is the difference between return and print?
    - return sends a result back to the caller, while print displays it on the console but returns None.
6. What are positional, keyword, and default arguments?
    - Positional: Arguments passed in order.
    - Keyword: Arguments passed with a key-value pair (name="Om").
    - Default: Argument that has a default value if none is provided.
      Example:

```
def greet(name="Guest"):
    print("Hello", name)
```

7. **What are *args and **kwargs in Python?**
   - ***args allows passing a variable number of positional arguments.**
   - ****kwargs allows passing a variable number of keyword arguments.**
     Example:

```
def func(*args, **kwargs):
    print(args)
    print(kwargs)
```

8. **What is the scope of a variable in Python?**
   - **Local scope: Defined inside a function, accessible only there.**
   - **Global scope: Defined outside all functions, accessible throughout.**
9. **How can you access a global variable inside a function?**
   - **Use the global keyword:**

```
x = 5
def func():
    global x
    x = 10
```

10. **Can a function return multiple values in Python?**

- Yes. Functions can return multiple values as a tuple:

```
def calc(a, b):
   return a+b, a*b
```

11. **What is a lambda function?**

- A small anonymous function defined using lambda keyword.

```
square = lambda x: x * x
print(square(5))  # Output: 25
```

12. **When would you use a lambda function instead of a regular function?**

- When you need a short, throwaway function — often used with functions like map(), filter(), and sorted().

13. What is recursion in Python?

- A function that calls itself to solve a smaller subproblem. Must have a base case to avoid infinite recursion.

14. What are function annotations?

- Optional metadata about the types of parameters and return values.

Example:

```
def greet(name: str) -> str:
    return "Hello " + name
```

15. What is a docstring in Python?

- A string literal used to document a function. Defined inside triple quotes.

Example:

```
def greet():
    """This function greets the user"""
    print("Hello!")
```

🔹 Section 6: Strings in Python

✅ Interview Questions Only

1. **What is a string in Python?**
    - **A string is a sequence of characters enclosed in single (') or double (") quotes.**
2. **How are strings stored in memory in Python?**
    - **Strings are stored as immutable sequences of Unicode characters.**
3. **Are strings mutable or immutable in Python?**

- **Strings are immutable, meaning they cannot be changed after creation.**

4. **How do you create a string in Python?**
   - **Using single, double, or triple quotes:**

```
s1 = 'Hello'
s2 = "World"
s3 = '''Multiline
string'''
```

5. **How do you access characters in a string?**
   - **Using indexing (zero-based):**

```
s = "Python"
print(s[0])  # Output: 'P'
```

6. **What is string slicing?**
   - **Extracting a part of the string:**

```
s = "Python"
print(s[1:4])  # Output: 'yth'
```

7. **What are some common string methods?**
   - **upper(), lower(), strip(), replace(), split(), join(), find(), count()**
8. **How do you concatenate strings in Python?**
   - **Using + operator:**
   - **"Hello" + " " + "World" → "Hello World"**
9. **How do you repeat a string multiple times?**
   - **Using * operator:**

```
"Hi" * 3 → "HiHiHi"
```

10. **What is the difference between is and == when comparing strings?**

- == compares values, is compares identities (memory locations).

11. **How do you check if a substring exists in a string?**

- Using the in keyword:

"py" in "python" → True

12. What does the strip() method do?

- Removes whitespace from the beginning and end of a string.

13. How do you format strings in Python?

- Using f-strings (Python 3.6+):

```
name = "Om"
print(f"Hello, {name}")
```

14. How do you convert a string to a list of words?

- Using the split() method:

"Data Science".split() → ['Data', 'Science']

15. What is the difference between split() and join()?

- split() divides a string into a list.
- join() merges a list into a string using a separator.

Example:

'-'.join(['a', 'b']) → "a-b"

16. Can strings be compared using relational operators?

- Yes. Strings are compared lexicographically based on Unicode values.

17. How do you check if a string starts or ends with a specific substring?

- Using startswith() and endswith() methods.

18. What does the find() and index() method do?

- Both return the index of the first occurrence of a substring.
- find() returns -1 if not found, index() raises a ValueError.

19. What is the use of escape sequences in strings?

- To include special characters like newline (\n), tab (\t), or quotes (\", \').

20. **Can a string contain both numbers and characters?**

- Yes, strings can contain letters, digits, symbols, or any combination.

🔹 Section 7: Lists, Tuples, Sets, and Dictionaries

✅ Interview-Based Questions

🔸 Lists in Python

1. **What is a list in Python?**
    - **A list is an ordered, mutable (changeable) collection of items.**
    - **Defined using square brackets [].**
2. **How do you create a list in Python?**

```
my_list = [1, 2, 3, "Python"]
```

3. **Are lists mutable or immutable?**
    - **Lists are mutable, meaning elements can be changed after creation.**
4. **How can you access, add, and remove elements in a list?**
    - **Access: list[0]**
    - **Add: append(), insert()**
    - **Remove: remove(), pop()**
5. **How do you iterate over a list?**

```
for item in my_list:
    print(item)
```

6. **What are common list methods?**
    - **append(), insert(), remove(), pop(), sort(), reverse(), extend(), index(), count()**

7. **What is the difference between append() and extend()?**
    - **append() adds one item, extend() adds multiple items from another list.**
8. **What is list comprehension?**
    - **A concise way to create lists:**

      [x2 for x in range(5)]**
9. **What happens if you access an index that doesn't exist?**
    - **You get an IndexError.**
10. **How can you sort a list in reverse order?**

my_list.sort(reverse=True)

- **Tuples in Python**
    1. **What is a tuple in Python?**
        - **A tuple is an ordered, immutable collection of items.**
        - **Defined using parentheses ().**
    2. **How do you create a tuple?**

       my_tuple = (1, 2, 3)
    3. **What is the difference between a list and a tuple?**
        - **Lists are mutable, tuples are immutable.**
        - **Tuples are generally faster and used for fixed data.**
    4. **Can a tuple contain mutable elements like lists?**
        - **Yes, but the tuple itself cannot be changed.**
    5. **How do you access elements of a tuple?**
        - **Using indexing: tuple[0]**
    6. **Can a tuple have one element?**
        - **Yes, but must include a comma:**

          single = (5,) # Not just (5)

7. What are common tuple methods?
    - count(), index()

- Sets in Python
    1. What is a set in Python?
        - A set is an unordered collection of unique items.
        - Defined using curly braces {} or the set() function.
    2. What are the key characteristics of sets?
        - Unordered, no duplicate values, mutable (but elements must be immutable).
    3. How do you create a set?

        ```
        my_set = {1, 2, 3}
        ```
    4. How are sets different from lists and tuples?
        - Sets are unordered and store unique elements only.
    5. What are some common set operations?
        - add(), remove(), discard(), union(), intersection(), difference(), clear()
    6. What is the difference between remove() and discard() in sets?
        - remove() raises an error if the item is not found; discard() does not.
    7. Can a set contain duplicate elements?
        - No, duplicates are automatically removed.
    8. What is a frozen set?
        - An immutable version of a set, created using frozenset().

- Dictionaries in Python
    1. What is a dictionary in Python?
        - A dictionary is a collection of key-value pairs.

- **Defined using curly braces {} with keys and values separated by a colon :.**

2. **How do you create a dictionary?**

```
my_dict = {"name": "Om", "age": 20}
```

3. **Are dictionary keys ordered?**
   - **As of Python 3.7+, dictionaries preserve insertion order.**
4. **Can dictionary keys be mutable?**
   - **No. Keys must be immutable types (like strings, numbers, tuples); values can be of any type.**
5. **How do you access, add, or update dictionary values?**

```
my_dict["name"]  # Access
my_dict["city"] = "Mumbai"  # Add/Update
```

6. **What are common dictionary methods?**
   - **keys(), values(), items(), get(), pop(), update(), clear()**
7. **What is the difference between get() and direct access ([]) for a key?**
   - **get() returns None if key doesn't exist, but [] raises a KeyError.**
8. **How do you iterate over a dictionary?**

```
for key, value in my_dict.items():
    print(key, value)
```

9. **Can dictionaries be nested in Python?**
   - **Yes, a dictionary can contain another dictionary as a value.**
10. **How do you remove an item from a dictionary?**

- Use pop(key) or del dict[key]

🔹 Section 8: Input and Output in Python

✅ Interview Questions

🔸 Input in Python

1. How do you take input from the user in Python?
    - Using the input() function:

      ```
      name = input("Enter your name: ")
      ```

2. What is the return type of input() function?
    - It always returns data as a string.
3. How do you take integer or float input from the user?
    - Convert the input using int() or float():

      ```
      age = int(input("Enter your age: "))
      price = float(input("Enter the price: "))
      ```

4. How do you take multiple inputs in one line?
    - Using split():

      ```
      x, y = input("Enter two numbers: ").split()
      ```

5. What does the split() function do in input handling?
    - Splits the input string into a list using a delimiter (space by default).
6. How do you take a list of integers in a single line input?

   ```
   nums = list(map(int, input().split()))
   ```

🔸 Output in Python

7. How do you display output in Python?
    - Using the print() function:

      ```
      print("Hello, world!")
      ```

8. **How do you print multiple variables in one line?**

```
name = "Om"
age = 20
print("Name:", name, "Age:", age)
```

9. **What are sep and end parameters in the print() function?**
    - **sep: Defines the separator between items (default is space)**
    - **end: Defines what is printed at the end (default is newline \n)**
      Example:

      print("A", "B", "C", sep="-", end="*")

10. **How do you print output without a newline at the end?**

- Use end="" in print():

```
print("Hello", end="")
```

11. **What is string formatting in Python?**

- Inserting values into strings using:
    - % formatting (old style)
    - str.format() (modern style)
    - **f-strings** (latest and recommended)

Example:

```
name = "Om"
print(f"Hello, {name}")
```

12. **What is the difference between print() and return?**

- print() displays the output.
- return is used in functions to pass a value back to the caller.

13. **Can you redirect output to a file using print()?**

- Yes, using the file parameter:

```
with open("output.txt", "w") as f:
    print("Hello, File!", file=f)
```

🔹 Section 9: File Handling in Python

✅ Interview Questions

🔸 Basics of File Handling

1. **What is file handling in Python?**
    - File handling is the ability to read from and write to files on the system using built-in functions.
2. **Which built-in function is used to open a file?**
    - open()
      Syntax:

      file = open("filename.txt", "mode")
3. **What are the different modes for opening a file?**
    - 'r' – Read (default)
    - 'w' – Write (overwrites if file exists)
    - 'a' – Append
    - 'x' – Create (fails if file exists)
    - 'b' – Binary mode
    - 't' – Text mode (default)
    - 'r+' – Read and write
4. **What is the difference between w and a mode?**
    - 'w' overwrites the file if it exists.
    - 'a' appends to the existing file without deleting content.
5. **How do you read the contents of a file?**

- **Using read(), readline(), or readlines():**

```
file.read()       # Reads the whole file
file.readline()   # Reads a single line
file.readlines()  # Reads all lines into a list
```

6. **How do you write to a file in Python?**
   - **Using the write() method:**

```
file.write("Hello, world!")
```

7. **Why is it important to close a file?**
   - **To free up system resources and ensure data is properly written to disk.**
8. **How do you close a file in Python?**
   - **Using file.close() or using a with statement (preferred):**

```
with open("file.txt", "r") as file:
    data = file.read()
```

9. **What are the advantages of using with open() instead of open()?**
   - **It automatically handles closing the file, even if an error occurs.**
10. **How do you check if a file exists before opening it?**

- Using the os module:

```
import os
os.path.exists("file.txt")
```

11. **What happens if you try to read a file that doesn't exist?**

- A FileNotFoundError is raised.

12. **How do you read a file line by line using a loop?**

```
with open("file.txt", "r") as f:
  for line in f:
    print(line)
```

13. **What is the use of seek() and tell()?**

- seek(position) moves the file cursor to the specified position.
- tell() returns the current cursor position.

14. How do you handle file-related errors in Python?

- Using try-except blocks:

```
try:
   file = open("file.txt", "r")
except FileNotFoundError:
   print("File not found.")
```

15. What is the difference between text and binary file modes?

- 't' mode is for text files (default).
- 'b' mode is for binary files (images, PDFs, etc.).

🔹 Section 10: Exception Handling in Python

✅ Interview Questions Only

🔸 Core Concepts

1. **What is exception handling in Python?**
    - **Exception handling is the process of responding to runtime errors (exceptions) in a controlled manner using try, except, finally, etc.**
2. **What is the difference between a syntax error and an exception?**
    - **A syntax error is a mistake in the code structure that prevents execution.**
    - **An exception is a runtime error that occurs during execution (e.g., division by zero).**
3. **What are the keywords used in exception handling?**
    - **try, except, else, finally, raise**
4. **What is the purpose of the try-except block?**

- **To catch and handle exceptions without crashing the program. Example:**

```
try:
    x = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero")
```

5. **What is the use of the else block in exception handling?**
    - **Code inside the else block runs if no exception is raised.**
6. **What is the purpose of the finally block?**
    - **It runs no matter what, whether an exception occurred or not.**
    - **Used for cleanup actions like closing a file or releasing resources.**
7. **What is a built-in exception in Python?**
    - **Predefined exceptions like ZeroDivisionError, ValueError, TypeError, IndexError, etc.**
8. **How can you handle multiple exceptions?**
    - **By using multiple except blocks:**

```
try:
    # some code
except ValueError:
    print("Value Error")
except TypeError:
    print("Type Error")
```

9. **What is a generic exception handler?**
    - **Catches any exception:**

```
except Exception as e:
    print("Error:", e)
```

10. **Can you nest try-except blocks in Python?**

- Yes, try-except blocks can be nested for complex error handling.

11. **How do you raise a custom exception?**

- Using the raise keyword:

raise ValueError("Invalid input")

12. **What is the use of the assert statement?**

- Used for debugging; raises AssertionError if the condition is False.

Example:

assert x > 0, "x must be positive"

13. **How can you define a custom exception?**

- By creating a class that inherits from Exception:

class MyError(Exception):
  pass

14. **What happens if you don't handle an exception?**

- The program crashes and shows a traceback with the error.

15. **What is the best practice for handling exceptions?**

- Catch only expected exceptions, avoid using a generic except unless necessary, and always clean up with finally.

🔹 Section 11: Object-Oriented Programming (OOP) Basics in Python

✅ Interview-Based Questions Only

🔸 Fundamentals

1. **What is Object-Oriented Programming (OOP)?**
    - **OOP is a programming paradigm that organizes code using objects (instances of classes), which bundle data (attributes) and functions (methods) together.**

2. **What are the main principles of OOP?**
    - **Encapsulation: Binding data and methods that operate on data in one unit (class).**
    - **Abstraction: Hiding internal implementation and showing only the essential details.**
    - **Inheritance: A mechanism for a class to inherit properties and behaviors from another class.**
    - **Polymorphism: The ability of different objects to respond to the same method in different ways.**
3. **What is a class in Python?**
    - **A blueprint for creating objects. It defines attributes and methods.**
      Example:

```
class Car:
    def __init__(self, brand):
        self.brand = brand
```

4. **What is an object in Python?**
    - **An instance of a class. It has actual values for the attributes defined in the class.**
      Example:

```
car1 = Car("Toyota")
```

5. **What is the __init__ method in Python?**
    - **It is a special method called a constructor used to initialize object attributes.**
6. **What is the difference between a class and an object?**
    - **A class defines structure and behavior.**
    - **An object is an instantiation of the class with actual data.**
7. **What are instance variables and class variables?**
    - **Instance variables are unique to each object (self.variable).**
    - **Class variables are shared across all objects of the class.**
8. **What is the role of self in Python OOP?**

- **self refers to the current instance of the class. It is used to access instance variables and methods.**

9. How do you create methods in a class?

- **Methods are defined inside a class and use self as the first argument.**

```
class Person:
    def greet(self):
        print("Hello")
```

10. What is encapsulation?

- Hiding the internal state of an object and requiring all interaction to be performed through methods.
- Achieved using **private members** (_var or __var).

◆ OOP Additional Concepts

11. What is abstraction in Python?

- Abstraction hides unnecessary implementation details and shows only the relevant information.
- Can be implemented using **abstract base classes** from the abc module.

12. What is inheritance in Python?

- The process where a class (child/derived) inherits attributes and methods from another class (parent/base).

Example:

```
class Animal:
    def sound(self):
        print("Animal sound")

class Dog(Animal):
    def bark(self):
        print("Woof")
```

13. What are the types of inheritance in Python?

- Single Inheritance
- Multiple Inheritance

- Multilevel Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

14. What is polymorphism?

- The ability to use a common interface for different data types or classes.

Example:

```
class Bird:
   def fly(self):
      print("Bird can fly")

class Airplane:
   def fly(self):
      print("Airplane can fly")
```

15. What are magic methods in Python?

- Special methods that start and end with double underscores (__init__, __str__, __len__, etc.)
- Used to define object behavior like constructors, string representation, operator overloading, etc.

16. What is operator overloading in Python?

- Redefining the way operators work for user-defined classes using magic methods like __add__, __lt__, etc.

17. How do you make attributes private in a class?

- Use double underscores (__) before the attribute name:

```
self.__balance = 1000
```

18. What is the difference between @classmethod, @staticmethod, and instance methods?

- **Instance method**: Takes self as first argument.
- **Class method**: Takes cls as first argument, defined using @classmethod.
- **Static method**: No self or cls, defined using @staticmethod.

19. **What is the purpose of super()?**

- It is used to call a method from the parent class.

Example:

```
super().__init__()
```

20. **Can Python support multiple inheritance?**

- Yes, a class can inherit from multiple classes.

🔹 Section 12: Modules and Packages in Python

✅ Interview Questions

🔸 Modules in Python

1. **What is a module in Python?**
   - **A module is a file containing Python code (functions, classes, variables) that can be reused in other Python programs. It usually has a .py extension.**
2. **How do you import a module in Python?**
   - **Using the import statement:**

     ```
     import math
     ```
3. **What is the difference between import module and from module import?**
   - **import module: Imports the whole module, access functions with module.name.**
   - **from module import name: Imports specific items directly. Example:**

     ```
     from math import sqrt
     ```
4. **How do you import a module with an alias?**

- **Using as:**
- **import numpy as np**

5. **What are some built-in modules in Python?**
    - **math, random, datetime, os, sys, json, time, etc.**
6. **How can you view the functions available in a module?**
    - **Using the dir() function:**

```
import math
print(dir(math))
```

7. **What happens if you try to import a non-existent module?**
    - **A ModuleNotFoundError is raised.**
8. **What is the purpose of the __name__ == "__main__" condition?**
    - **It ensures that certain code runs only when the script is executed directly, not when it is imported as a module.**

- **Packages in Python**

9. **What is a package in Python?**
    - **A package is a collection of modules organized in directories that include a special __init__.py file (optional in Python 3.3+).**
10. **How is a package different from a module?**

- A **module** is a single .py file.
- A **package** is a directory containing multiple modules and possibly sub-packages.

11. **How do you import a module from a package?**

- Syntax:

```
from package.module import function
```

12. **What is the role of __init__.py in a package?**

- It indicates that the directory is a package. It can also execute initialization code.

13. Can packages contain sub-packages?

- Yes, Python supports nested packages.

14. How can you create a custom module or package?

- Create a .py file (for a module) or a folder with Python files (for a package). Use import to access them.

15. What is the Python Standard Library?

- A collection of modules and packages that come pre-installed with Python, providing tools for file handling, math, date/time, etc.

16. How do you install external packages in Python?

- Using pip:

```
pip install package_name
```

17. Where are installed packages stored in Python?

- Usually in the site-packages directory inside the Python environment.

18. How do you list all installed packages?

- Run:

```
pip list
```

19. Can you import a module from a specific file path?

- Yes, using the sys module:

```
import sys
sys.path.append('/path/to/module')
```

20. What is a namespace in the context of modules?

- A namespace is a container where names (identifiers) are mapped to objects. Modules help create separate namespaces to avoid name conflicts.

🔹 Section 13: Basic Python Libraries

✅ Interview Questions

Covers math, random, and datetime modules

🔸 1. math Module

1. **What is the math module in Python used for?**
    - **It provides mathematical functions like square root, power, trigonometry, factorial, etc.**
2. **How do you import the math module?**

    ```
    import math
    ```

3. **How do you find the square root of a number?**

    ```
    math.sqrt(25)  # Output: 5.0
    ```

4. **How do you calculate the power of a number?**

    ```
    math.pow(2, 3)  # Output: 8.0
    ```

5. **How do you get the value of π (pi)?**

    ```
    math.pi
    ```

6. **What's the difference between math.floor() and math.ceil()?**
    - **floor() returns the largest integer ≤ x.**
    - **ceil() returns the smallest integer ≥ x.**
7. **How to calculate the factorial of a number?**

    ```
    math.factorial(5)  # Output: 120
    ```

8. **What is the difference between math.pow() and the ** operator?**
    - **math.pow() returns a float, ** may return an int if inputs are int.**

🔸 2. random Module

9. **What is the purpose of the random module?**
    - **Used to generate random numbers, shuffle data, pick random items, etc.**
10. **How do you import the random module?**

```
import random
```

11. **How to generate a random number between 0 and 1?**

```
random.random()
```

12. **How to generate a random integer between two numbers?**

```
random.randint(10, 20)  # Inclusive
```

13. **How to select a random element from a list?**

```
random.choice(['apple', 'banana', 'cherry'])
```

14. **How to randomly shuffle elements in a list?**

```
random.shuffle(my_list)
```

15. **How to generate a random float between a given range?**

```
random.uniform(1.5, 9.8)
```

- **3. datetime Module**

16. **What is the purpose of the datetime module in Python?**
    - To work with dates and times — current time, formatting, date arithmetic, etc.
17. **How to import the datetime module?**

```
import datetime
```

18. How to get the current date and time?

```
datetime.datetime.now()
```

19. How to get only the current date?

```
datetime.date.today()
```

20. How to create a custom date object?

```
datetime.date(2024, 12, 25)
```

21. How to calculate the difference between two dates?

```
d1 = datetime.date(2024, 12, 25)
d2 = datetime.date(2024, 12, 20)
delta = d1 - d2  # Output: datetime.timedelta(days=5)
```

22. How do you format a datetime object as a string?

```
now = datetime.datetime.now()
now.strftime("%Y-%m-%d %H:%M:%S")
```

23. What does strftime() do?

- Converts a datetime object to a string using format codes.

24. How do you convert a string to a datetime object?

```
datetime.datetime.strptime("2024-12-25", "%Y-%m-%d")
```

25. What is a timedelta in Python?

- It represents the difference between two datetime or date objects.

🔹 Section 14: Medium to Tough Python Interview Questions

🎯 Covers All Core Python Basics

🔸 1. Variables & Data Types

1. What are mutable and immutable types in Python? Give examples.
2. Explain the difference between is and == with examples.
3. What is the difference between a shallow copy and a deep copy in Python?

🔸 2. Operators

4. Explain operator precedence and associativity with a custom example.
5. What is the difference between and, or, and not vs bitwise &, |, ~?
6. What happens if we do "5" + 5 in Python? How can it be corrected?

🔸 3. Control Flow

7. How does Python handle if-elif-else compared to nested if?
8. Write a program to find the largest of 3 numbers using only conditional expressions (no if).

🔸 4. Loops

9. What is the use of the else block in a loop? Give a real-world example.
10. Write a Python program to print prime numbers from 1 to 100 using a for loop.

🔸 5. Functions

11. Explain the difference between default, keyword, and variable-length arguments.
12. Write a recursive function to calculate the nth Fibonacci number.

13. What are *args and **kwargs? Write a function using both.

- **6. Strings**

14. Explain the difference between isalpha(), isdigit(), isalnum().
15. Write a Python function to check if a string is a palindrome (ignore case and spaces).
16. Count the number of vowels and consonants in a string using dictionary mapping.

- **7. Lists, Tuples, Sets, Dictionaries**

17. What is the difference between list comprehension and generator expressions?
18. How does a set handle duplicate values internally?
19. Write a Python program to merge two dictionaries and sort by value.
20. Create a nested dictionary from two lists:
    keys = ["a", "b", "c"] and values = [1, [2, 3], {"d": 4}]

- **8. Input/Output**

21. What is the difference between input() and eval(input())?
22. How would you take multiple inputs in one line and store them as integers in a list?

- **9. File Handling**

23. Write a Python program to read a file and count word frequencies.
24. How can you append text to a file without overwriting it?
25. Explain file modes in Python with examples: 'r', 'w', 'a', 'r+', 'w+'.

- **10. Exception Handling**

26. How does Python's try-except-else-finally block work?
27. What is the use of raise keyword in Python? Give an example.

28. Write a function that validates user input (integer only), raises an exception otherwise.

11. Object-Oriented Programming (OOP)

29. What is the difference between instance variables, class variables, and static variables?
30. Implement polymorphism using method overriding in Python.
31. How would you make a class attribute private and access it safely?

12. Modules & Libraries

32. Explain how modules help in code reusability with an example.
33. What is the difference between a module and a package in Python?
34. How does Python resolve the name of a module during import? (Explain module resolution order - sys.path)

13. Math, Random, Datetime Libraries

35. Write a program that randomly selects a password from a list and logs the timestamp.
36. Calculate the number of days between two given dates using datetime.
37. Generate a list of 10 random even numbers between 1 and 100.

14. Combined Logic Problems (Tougher)

38. Write a function that accepts a string of words and returns the longest word and its length.
39. Given a list of integers, return only those that appear more than once without using collections.Counter.
40. Create a mini phone book where you can add, search, and delete contacts using dictionaries.