1. What is a class and what is an object?

Class:
A class is a **blueprint/template** used to create objects. It defines **attributes (data)** and **methods (functions)**.

Object:
An object is a **real instance** of a class that represents real-world entities.

Example: Student Class

```
class Student:
    def input_details(self, rollnum, name, branch, sem):
        self.rollnum = rollnum
        self.name = name
        self.branch = branch
        self.sem = sem

    def display_details(self):
        print(self.rollnum, self.name, self.branch, self.sem)
```

2. What is the use of `__init__()` method?

- It is a **constructor**
- Automatically called when an object is created
- Used to **initialize instance variables**

```
class Student:
    def __init__(self, rollnum, name):
        self.rollnum = rollnum
        self.name = name
```

3. What is the use of `seek()` method?

- Used to **change the file cursor position**
- Helps in random file access

```
file.seek(0)
```

4. What is the use of `tell()` method?

- Returns the **current position** of the file cursor

```
position = file.tell()
```

5. Describe file access modes: `'r'`, `'w'`, `'a'`

Mode	Description
`r`	Read only (file must exist)
`w`	Write (creates new or overwrites existing)
`a`	Append (adds data at end)

6. Explain mutable and immutable data types

Mutable: Objects whose state can be modified after creation (e.g., Lists, Dictionaries, Sets)
Examples: `list`, `set`, `dict`

```
lst = [1,2]
lst.append(3)
```

Immutable: Objects whose state *cannot* be modified after creation. Attempting to change them creates a new object (e.g., Integers, Strings, Tuples)
Examples: `int`, `float`, `tuple`, `string`

```
x = 10
```

7. What is the `pass` statement?

- A **null statement**
- Used when a statement is syntactically required but no action is needed

```
if x > 5:
    pass
```

8. Why is a base condition necessary in recursion?

- Prevents **infinite recursion**
- Defines when recursion should stop

```
def fact(n):
    if n == 0:
        return 1
```

9. What is the use of `try` and `except` blocks?

- Used for **exception handling**
- Prevents program crash

```
try:
    x = 10/0
except:
    print("Error")
```

10. Uses of `else` and `finally` in exception handling

- `else`: Executes when no exception occurs
- `finally`: Executes **always** (cleanup)

11. What is the use of `open()` function?

- Opens a file and returns a file object

```
file = open("data.txt", "r")
```

12. Difference between `while` loop and `for` loop

While	For
Condition-based	Sequence-based
Entry-controlled	Used for known iterations

13. Difference between loop-else and if-else

Loop-Else	If-Else
Executes if loop completes normally	Executes based on condition
Used with loops	Used with conditions

14. Significance of `if-elif-else`

- Handles **multiple conditions**
- Improves readability

15. Difference between `break` and `continue`

break	continue
Exits loop	Skips current iteration

16. Significance of `pass` in functions

- `pass` is a null statement. The interpreter reads it but executes nothing. It is used as a placeholder where valid syntax is required but no code is needed yet .

```
def my_function():
    pass  # Function defined but empty
```

17. Importance of `assert` statement

- Used for debugging. It tests a condition; if the condition is True, nothing happens. If False, it raises an `AssertionError` .

```
x = -1
assert x > 0, "x must be positive"  # Raises AssertionError
```

18. Benefit of `if __name__ == "__main__"`

- Ensures code runs **only when file is executed directly**
- Avoids execution during import

19. Difference between logical and relational operators

Relational	Logical
`<, >, ==`	`and, or, not`
Compare values	Combine conditions
Output: Boolean	Output: Boolean

20. Use of `global` and `nonlocal` keyword

- `global`: Access global variable inside function
- `nonlocal`: Access variable from enclosing function

21. Output-type questions cover

- Operators
- Scope
- `if` statements
- Loops
- Built-in math & string functions

22. Linear Search

- A simple search algorithm that checks every element in the list sequentially until a match is found.
- Sequential search technique
- Time Complexity: **O(n)**

```
def linear_search(arr, target):
    for i in range(len(arr)):
        if arr[i] == target:
            return i  # Return index if found
    return -1  # Return -1 if not found
```

23. Bubble Sort

- Sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order.
- Comparison-based sorting
- Time Complexity: **O(n²)**

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        # Last i elements are already in place
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                # Swap
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr
```

24. All Lab Assignments