Q 1: List and explain different types of files. Which type of file (text/binary) is more suitable for storing images and justify your answer.

Answer:
In Python, files are mainly of two types:

1. **Text Files (.txt, .csv, .py etc.)**
   - Store data in human-readable form (characters).
   - Each line is terminated by a special character `\n`.
   - Example: writing names, numbers, programs.
2. **Binary Files (.jpg, .png, .mp3, .exe etc.)**
   - Store data in the same format as in computer memory (0s and 1s).
   - Not human-readable, but exact copy of data is preserved.
   - Example: images, audio, video, executable files.

✅ **For storing images, binary files are more suitable** because images are made of pixels (binary data). Text files may corrupt image data, while binary files store the exact byte-by-byte format without loss.

Q2) Define and explain `seek()` and `tell()` methods in file handling.

Answer:
In Python, file handling provides methods to move and track the file pointer.

1. **`tell()` Method**
   - It returns the current position of the file pointer (in bytes) from the beginning of the file.
   - Example:
   - `f = open("demo.txt", "r")`
   - `print(f.tell())   # shows current position`
   - `f.close()`
2. **`seek(offset, from_where)` Method**
   - It moves the file pointer to a specific position.
   - `offset` → number of bytes to move.
   - `from_where` → reference position (0 = beginning, 1 = current, 2 = end).
   - Example:
   - `f = open("demo.txt", "r")`
   - `f.seek(5)        # moves pointer to 5th byte`
   - `print(f.tell())  # prints 5`
   - `f.close()`

✅ Together, `seek()` is used to change the position of the pointer, and `tell()` is used to know the current position.

Here's the **5-mark answer** for you 👇

Q3) Explain the difference between `read()` and `readlines()` methods with an example.

Answer:

1. **`read()` Method**
   - Reads the **entire content** of the file as a **single string**.
   - Example:
   - `f = open("demo.txt", "r")`
   - `data = f.read()`
   - `print(data)  # prints complete file as one string`
   - `f.close()`
2. **`readlines()` Method**
   - Reads the **whole file line by line** and returns a **list of strings** (each line as one element).
   - Example:
   - `f = open("demo.txt", "r")`
   - `lines = f.readlines()`
   - `print(lines)  # prints list: ['line1\n', 'line2\n', ...]`
   - `f.close()`

✅ **Difference:**

- `read()` → returns one long string.
- `readlines()` → returns a list of lines.

Want me to also make one on **`read()` vs `readline()`** (very commonly asked)?

Here's the **5-mark detailed answer** 👇

Q4) Explain how to write into files using `write()` and `writelines()` along with example.

Answer:

In Python, we use `write()` and `writelines()` methods to store data in files.

1. **`write()` Method**
   - Writes a **single string** into the file.
   - Example:
   - `f = open("demo.txt", "w")`
   - `f.write("Hello World\n")`
   - `f.write("Python File Handling")`

- `f.close()`
- Here, text is written line by line using `\n`.

2. **`writelines()` Method**
   - Writes a **list of strings** into the file at once.
   - Example:
   - `f = open("demo.txt", "w")`
   - `lines = ["Line1\n", "Line2\n", "Line3\n"]`
   - `f.writelines(lines)`
   - `f.close()`
   - Each list element is written as it is.

✅ **Difference:**

- `write()` → writes only one string.
- `writelines()` → writes multiple strings (list of lines).

Do you want me to also prepare a **small table (write vs writelines)** for easy last-minute revision?

Here's a **5-mark answer** ready for copy-paste 👇

Q5) Explain the difference between lists and tuples.

Answer:

Feature	List (`[]`)	Tuple (`()`)
Mutability	Mutable (can change elements)	Immutable (cannot change elements)
Syntax	`mylist = [1, 2, 3]`	`mytuple = (1, 2, 3)`
Methods	Many methods like `append()`, `remove()`, `pop()`	Few methods like `count()`, `index()`
Speed	Slower due to mutability	Faster due to immutability
Use Case	Use when data needs modification	Use for fixed data or keys in dictionary

Example:

```
# List
mylist = [10, 20, 30]
mylist.append(40)
print(mylist)  # Output: [10, 20, 30, 40]

# Tuple
mytuple = (10, 20, 30)
# mytuple[0] = 50  # Error: cannot modify
print(mytuple)  # Output: (10, 20, 30)
```

✅ **Summary:**

- Lists are **changeable**, tuples are **fixed**.
- Tuples are faster and safer for constant data.

I can continue preparing the next **OOP/Python file handling questions** in this same **exam-ready 5-mark format**.

Do you want me to do that?

Here's a **5-mark answer** you can copy-paste 👇

Q6) List and explain any two built-in generic data structures in Python

Answer:

Python provides several built-in data structures. Two common generic data structures are:

1. **List**
   - **Definition:** An ordered, mutable collection of elements.
   - **Characteristics:**
     - Can store elements of different types.
     - Allows duplicate elements.
     - Indexing and slicing are supported.
   - **Example:**
   - `mylist = [10, "Python", 3.14]`
   - `mylist.append("AI")`
   - `print(mylist)  # Output: [10, 'Python', 3.14, 'AI']`
2. **Dictionary**
   - **Definition:** An unordered collection of key-value pairs.
   - **Characteristics:**
     - Keys must be unique and immutable.
     - Values can be of any type.
     - Supports fast lookup using keys.
   - **Example:**
   - `mydict = {"name": "Chetan", "age": 25, "city": "Pune"}`
   - `print(mydict["age"])  # Output: 25`

✅ **Summary:**

- **List** → ordered and mutable.
- **Dictionary** → key-value pairs, fast lookup, keys unique.

If you want, I can also prepare the **next few OOP questions on class, object, inheritance, and polymorphism** in the same **5-mark ready format**.

Do you want me to do that?

Here's a **5-mark ready Python program** for your question 👇

Q7) Python program to create a text file and write student details

```
# Open a new file in write mode
f = open("student.txt", "w")

# Input student details from user
name = input("Enter student name: ")
roll_no = input("Enter roll number: ")
marks = input("Enter marks: ")

# Write details to the file
f.write("Name: " + name + "\n")
f.write("Roll Number: " + roll_no + "\n")
f.write("Marks: " + marks + "\n")

# Close the file
f.close()

# Display success message
print("Student details have been successfully written to student.txt")
```

Explanation:

1. `open("student.txt", "w")` → creates a new file or overwrites if it exists.
2. `input()` → takes user input for name, roll number, and marks.
3. `write()` → writes the data into the file with proper formatting.
4. `close()` → closes the file to save data.
5. Prints a success message after writing.

If you want, I can also make the **version using `with open()`** which is safer and more professional. This is usually preferred in exams and real projects. Do you want me to do that?

Here's a **5-mark ready Python program** for your question 👇

Q8) Python program to count lines, words, and characters in a file

```
# Accept absolute file path from the user
file_path = input("Enter the absolute file path: ")

try:
```

```
    # Open the file in read mode
    with open(file_path, "r") as f:
        lines = f.readlines()  # Read all lines

        # Count lines
        line_count = len(lines)

        # Count words and characters
        word_count = 0
        char_count = 0
        for line in lines:
            words = line.split()
            word_count += len(words)
            char_count += len(line)

    # Display the counts
    print("Number of lines:", line_count)
    print("Number of words:", word_count)
    print("Number of characters:", char_count)

except FileNotFoundError:
    print("Error: File not found. Please check the path.")
```

Explanation:

1. `input()` → gets absolute file path from user.
2. `with open(file_path, "r")` → safely opens the file in read mode.
3. `readlines()` → reads all lines into a list.
4. Counts:
    - **Lines:** `len(lines)`
    - **Words:** sum of words in each line
    - **Characters:** sum of characters in each line
5. Handles **FileNotFoundError** if the path is incorrect.

If you want, I can prepare **Q9, Q10…** for file handling and OOP in the same **exam-ready 5-mark format** so you'll have a full set.

Do you want me to do that?

Here's a **5-mark ready Python solution** for your question 👇

Q9) Python function `find_max(items)` to return maximum element

```
# Function to find maximum element in a list
def find_max(items):
    if not items:  # Check if list is empty
        return None
    max_item = items[0]
    for item in items:
        if item > max_item:
            max_item = item
```

```
    return max_item

# Demonstration with different types of lists

# List of integers
int_list = [10, 45, 23, 67, 5]
print("Maximum in integer list:", find_max(int_list))

# List of strings
str_list = ["apple", "banana", "orange", "mango"]
print("Maximum in string list:", find_max(str_list))
```

Explanation:

1. The function `find_max(items)` takes any list of **comparable elements** (int, float, str).
2. Initializes `max_item` with the first element.
3. Loops through the list and updates `max_item` if a larger element is found.
4. Returns the maximum element.
5. Demonstrated with **integer list** and **string list** to show its generic nature.

✅ **Key point:** Works with **any data type** that supports comparison using > operator.

I can prepare **Q10 onwards for common OOP topics** like class, object, inheritance, polymorphism, encapsulation, with **5-mark ready answers and examples**.

Do you want me to do that next?

Here's a **5-mark ready Python solution** for your question 👇

Q10) Python generic function `swap(a, b)`

```
# Generic function to swap two values
def swap(a, b):
    return b, a

# Demonstration with integers
x, y = 10, 20
x, y = swap(x, y)
print("After swapping integers:", x, y)  # Output: 20 10

# Demonstration with strings
str1, str2 = "Hello", "World"
str1, str2 = swap(str1, str2)
print("After swapping strings:", str1, str2)  # Output: World Hello
```

Explanation:

1. The function `swap(a, b)` **returns the values in reverse order**, effectively swapping them.

2. Works with **any data type** (int, float, string, etc.), making it generic.
3. Demonstrated with:
    - **Integers:** swaps 10 and 20 → 20 and 10
    - **Strings:** swaps "Hello" and "World" → "World" and "Hello"

✅ **Key point:** Python supports multiple return values, which makes writing a generic swap function simple.

I can continue with **next questions on class, object, inheritance, and other OOP concepts** in the same **5-mark ready format**.

Do you want me to do that?

Here's a **5-mark ready Python solution** using a generic class for your question 👇

Q11) Python generic class `Box[T]`

```
from typing import TypeVar, Generic

# Define a type variable
T = TypeVar('T')

# Generic class
class Box(Generic[T]):
    def __init__(self):
        self.items = []

    def add_item(self, item: T):
        self.items.append(item)

    def display_items(self):
        print("Items in the box:", self.items)

# Demonstration with integers
int_box = Box[int]()
int_box.add_item(10)
int_box.add_item(20)
int_box.display_items()  # Output: Items in the box: [10, 20]

# Demonstration with strings
str_box = Box[str]()
str_box.add_item("apple")
str_box.add_item("banana")
str_box.display_items()  # Output: Items in the box: ['apple', 'banana']
```

Explanation:

1. `TypeVar('T')` → defines a generic type variable.
2. `Box(Generic[T])` → generic class that works with any type `T`.
3. `add_item(item: T)` → adds an item of type `T` to the box.

4. `display_items()` → prints all items in the box.
5. Demonstrated with **integers** and **strings** to show generic behavior.

✅ **Key point:** Generic classes allow creating flexible and type-safe containers for any data type.

I can prepare **Q12: Python program demonstrating inheritance with generic or normal classes** next in the same exam-ready 5-mark format. Do you want me to continue?