Q1. Name different modes of Python

Ans:- Python has two basic modes:
- Script (Normal Mode)
- Interactive Mode

Q2. List file operations in Python.

Ans:-
- Opening file (using open() function)
- Reading file (using read() function)
- Writing file (using write() function)
- Copy files
- Delete files (using remove() function)
- Closing file (Using close() function)

Q3. With neat example differentiate between readline () and readlines ( ) functions in file-handling.

Ans:-

1) readline( ) Function

- Reads only one line from the file at a time.
- Each time it is called, it reads the next line from the file.
- Useful when reading large files line by line.
- Example:

```
file = open("example.txt", "r")  # Open file in read mode
print(file.readline())  # Reads the first line
print(file.readline())  # Reads the second line
file.close()
```

- Output:

```
Hello, welcome to Python!
This is a file handling tutorial.
```

(*Only two lines are read*)

2) readlines( ) Function

- Reads all lines at once and returns them as a list of strings.
- Each element in the list represents a line from the file.
- Useful when you need to store all lines in memory for processing.
- Example:

```
file = open("example.txt", "r")  # Open file in read mode
lines = file.readlines()  # Reads all lines and stores them in a list
print(lines)  # Prints the list of lines
file.close()
```

- Output:

```
['Hello, welcome to Python!\n', 'This is a file handling tutorial.\n', 'Have a great day!\n']
```

- Key Differences

Feature	readline( )	readlines( )
Reads how many lines?	One line at a time	All lines at once
Return type	String	List of strings
Performance	Memory efficient (reads one line at a time)	Consumes more memory (stores all lines)
Usage	Best for reading large files line by line	Best when you need all lines at once

Marathi (टीप)

- **readline( )** – एका वेळी फक्त एक ओळ वाचते. मोठ्या फायलींसाठी चांगले आहे.
- **readlines( )** – सर्व ओळी यादी (list) स्वरूपात वाचते. लहान फायलींसाठी चांगले आहे.

Q4. State the use of read() and readline () functions in python file handling.

Ans:- **1) read([n]) Method**

Functionality:

- Reads entire file content and returns it as a string.
- If a number n is specified, it reads only first n bytes/characters.
- If n is not given, it reads the whole file.

Example:

```
f = open("sample.txt", "r")  # Open file in read mode
print(f.read(5))  # Read first 5 characters
print(f.read())  # Read the rest of the file
f.close()  # Close the file
```

Output:

```
Pytho
n is fun!
Let's learn file handling.
```

(*First 5 characters read separately, then the rest of the file is read.*)

2) readline([n]) Method

Functionality:

- Reads one line at a time from the file.
- If n is specified, it reads only first n bytes/characters from the line.
- If EOF (End of File) is reached, it returns an empty string.

Example:

```
f = open("sample.txt", "r")  # Open file in read mode
print(f.readline())  # Read the first line
print(f.readline(3))  # Read first 3 characters of the second line
print(f.readline())  # Read the remaining part of the second line
f.close()  # Close the file
```

Output:

```
Python is fun!
Let
's learn file handling.
```

(*First line is fully read, then only first 3 characters of the second line, and finally, the rest of the second line.*)

Q5. Explain seek ( ) and tell ( ) function for file pointer manipulation in python with example.

Ans:-

1) seek() Function

Functionality:

- Used to **move** the file pointer (cursor) to a **specific position** in the file.
- Controls **where the next read or write operation** will start.

Syntax:

f.seek(offset, from_where)

- offset: Number of **bytes/characters** to move.
- from_where: The reference position (default = 0).
  - 0 → Beginning of the file
  - 1 → Current position (*only in binary mode*)
  - 2 → End of the file (*only in binary mode*)

Example:

```
f = open("demofile.txt", "r")  # Open file in read mode
f.seek(4)  # Move cursor to the 4th character from the start
print(f.readline())  # Read from the new position
f.close()
```

Example File (demofile.txt):

```
Hello, Python!
Welcome to file handling.
```

Output:

```
o, Python!
```

(*The first 4 characters 'Hell' are skipped, and reading starts from the 5th character.*)

2) tell() Function

Functionality:

- Returns the **current position** of the file pointer **in bytes** from the beginning.
- **Useful for tracking** where the file pointer is located.

Syntax:

```
file.tell()
```

Example:

```
f = open("demofile.txt", "r")  # Open file in read mode
print(f.tell())  # Prints 0 (File pointer at the start)
f.read(5)  # Read first 5 characters
print(f.tell())  # Prints 5 (File pointer moved to 5)
f.close()
```

Output:

```
0
5
```

(*Initially, the pointer is at position 0, and after reading 5 characters, it moves to 5.*)

- **Key Differences**

Feature	seek()	tell()
Purpose	Moves file pointer to a specific position	Returns current file pointer position
Arguments	seek(offset, from_where)	No arguments
Returns	Moves pointer, does not return value	Returns an **integer** (byte position)
Use Case	Used to **jump** to specific locations in a file	Used to **track** the file pointer position

Marathi (टीप)

- **seek(n)** – फाईल पॉइंटर (कर्सर) दिलेल्या n स्थानावर हलवतो.
- **tell()** – सध्याच्या फाईल पॉइंटरची स्थिती दर्शवतो.

Q6. Write a program function that accepts a string and calculate the number of uppercase letters and lower case letters.

Ans:-

- **Python Program to Count Uppercase and Lowercase Letters**

```
def count_case(s):
    upper_count = 0  # Counter for uppercase letters
    lower_count = 0  # Counter for lowercase letters

    for char in s:
        if char.isupper():
            upper_count += 1
        elif char.islower():
            lower_count += 1

    print(f" 🔹 Uppercase Letters: {upper_count}")
    print(f" 🔹 Lowercase Letters: {lower_count}")

# Example Usage
string = input("Enter a string: ")
count_case(string)
```

- **Output:**

Input:

Enter a string: Hello World!

Output:

Uppercase Letters: 2

Lowercase Letters: 8

- **isupper()** – अक्षर मोठ्या (A-Z) स्वरूपात आहे का ते तपासते.
- **islower()** – अक्षर लहान (a-z) स्वरूपात आहे का ते तपासते.

Q1. Describe various modes of file object ? Explain any two in detail.

Q2. Explain any four file modes in Python. Q3. List different modes of opening file in python.

(For all three question write the below answer)

- **Modes of File Handling in Python**
  - In Python, files can be opened in different modes depending on the required operation. The programmer needs to specify whether to read ('r'), write ('w'), or append ('a'). Additionally, files can be opened in text mode ('t') or binary mode ('b').
- **Types of File Modes:**
  1. Text Mode (Default)
     - Returns strings while reading from the file.
     - Used when working with text files such as .txt.
  2. Binary Mode
     - Returns bytes instead of strings.
     - Used for non-text files such as images or executable files.
  - The text (t) and binary (b) modes are combined with r, w, or a modes to perform file operations. The complete list of file modes in Python is shown in the table below:

Sr. No	Mode	Description
1	`r`	Opens a file for **reading only**. File pointer is placed at the **beginning**. *(Default mode)*
2	`rb`	Opens a file for **reading in binary format**. File pointer is at the **beginning**.
3	`r+`	Opens a file for **both reading and writing**. File pointer is at the **beginning**.
4	`rb+`	Opens a file for **both reading and writing in binary format**.
5	`w`	Opens a file for **writing only**. Overwrites if file exists, otherwise creates a new file.
6	`wb`	Opens a file for **writing in binary format**. Overwrites if file exists, otherwise creates a new file.
7	`w+`	Opens a file for **reading and writing**. Overwrites the existing file.
8	`wb+`	Opens a file for **reading and writing in binary format**. Overwrites the file if it exists.
9	`a`	Opens a file for **appending**. File pointer moves to the **end** of the file.
10	`ab`	Opens a file for **appending in binary format**. Pointer moves to the **end**.
11	`a+`	Opens a file for **both appending and reading**. Pointer is at the **end**.
12	`ab+`	Opens a file for **both appending and reading in binary format**.
13	`t`	Opens file in **text mode**. *(Default)*
14	`b`	Opens file in **binary mode**.
15	`+`	Opens file for **updating** (reading and writing).

(IMP : from below 3 program 1 program fix yeto so remember all three)

Q. Write a Python program to read contents from "a.txt" and write same contents in "b.txt."

```
with open('abs.txt','r') as firstfile, open('prq.txt','w') as secondfile:
        # read content from first file
    for line in firstfile:
        # write content to second file
    secondfile.write(line)
```

Q. Write a Python program to read contents of first.txt file and write same contents in second.txt file..

```
with open('first.txt', 'r') as f:              # Open the first file for reading
    contents = f.read()                        # Read the contents of the file

with open('second.txt', 'w') as f:             # Open the second file for writing
    f.write(contents)                          # Write the contents of the first file to the second file
```

Q. Write a program to open a file in write mode and append some content at the end of file.

```
file1 = open("myfile.txt", "w")
L = ["This is Delhi \n", "This is Paris \n", "This is London"]
file1.writelines(L)
file1.close()

# Append-adds at last
# append mode
file1 = open("myfile.txt", "a")

# writing newline character
file1.write("\n")
file1.write("Today")

# without newline character
file1.write("Tomorrow")

file1 = open("myfile.txt", "r")
print("Output of Readlines after appending")
print(file1.read())
print() file1.close()
```

Output:

```
Output of Readlines after appending
This is Delhi
This is Paris
This is London
TodayTomorrow
```

❖ 6.3 What is Exception Handling in Python? (Most IMP)

- In Python, an **exception** is an error that occurs during program execution and interrupts the normal flow of the program.
- Exception Handling is a mechanism that helps **handle such errors gracefully** so that the program doesn't crash.
- Python provides special keywords like **try**, **except**, **else**, and **finally** to handle exceptions properly.
- Some common exceptions in Python:
  - **ZeroDivisionError** → When you divide a number by zero.
  - **ValueError** → When you enter an invalid value (e.g., letters instead of numbers).
  - **FileNotFoundError** → When a file is not found.

◆ Python Code to Check for Zero Division Error Exception

To **avoid crashing**, we can use **try-except** to catch the error and display a proper message.

```
try:
    num1 = int(input("Enter a number: "))
    num2 = int(input("Enter another number: "))
    result = num1 / num2  # This may cause ZeroDivisionError
    print("Result:", result)
except ZeroDivisionError:
    print("Error: Cannot divide by zero!")  # Handles the error gracefully
```

Output 1 (No Error, User enters 10 and 2)

```
Enter a number: 10
Enter another number: 2
Result: 5.0
```

Output 2 (Error, User enters 10 and 0)

```
Enter a number: 10
Enter another number: 0
Error: Cannot divide by zero!
```

Now, the program does not crash! Instead, it shows a friendly error message.

◆ Try-Except-Else-Finally Block in Python (Step-by-Step Explanation)

Python provides **four blocks** for exception handling:

1. try block:

- This is where we write the code that **may cause an error**.
- If an error occurs, the control moves to the except block.
- If no error occurs, the program continues normally.

2. except block:

- This block **catches the error** and prevents the program from crashing.
- We can have multiple except blocks for different types of errors.

3. else block (optional):

- This block **executes only if there is no exception** in the try block.
- Used when we want to confirm that the operation was successful.

4. finally block (optional):

- This block **always executes**, whether there was an exception or not.
- Used for cleanup operations, like closing files or freeing resources.

Python Program Using Try-Except-Else-Finally

```
try:
    a = int(input("Enter first number: "))
    b = int(input("Enter second number: "))
    result = a / b  # May cause ZeroDivisionError
except ZeroDivisionError:
    print("Error: Cannot divide by zero!")  # Handles division by zero error
except ValueError:
    print("Error: Please enter valid numbers!")  # Handles invalid input
else:
    print("Division successful! Result:", result)  # Executes if no error
finally:
    print("Execution completed!")  # Always executes
```

Case 1: No Error (User enters 10 and 2)

```
Enter first number: 10
Enter second number: 2
Division successful! Result: 5.0
Execution completed!
```

✔ Everything works fine!

Case 2: ZeroDivisionError (User enters 10 and 0)

```
Enter first number: 10
Enter second number: 0
Error: Cannot divide by zero!
Execution completed!
```

✔ The program **doesn't crash**, and the finally block still runs.

Case 3: ValueError (User enters 10 and abc)

```
Enter first number: 10
Enter second number: abc
Error: Please enter valid numbers!
Execution completed!
```

✔ The program **handles invalid input** properly.

Why Exception Handling is Important?

- Prevents the program from crashing.
- Makes the program more **user-friendly**.
- Helps in debugging and identifying errors.
- Ensures smooth execution even when unexpected input is given.

Marathi Tip (मराठीत संक्षिप्त माहिती)

- **Exception Handling** म्हणजे **कार्यक्रमात येणाऱ्या त्रुटी (errors) हाताळण्याची पद्धत** आहे.
  - try मध्ये त्रुटी येऊ शकणारा कोड लिहिला जातो.
  - except मध्ये जर त्रुटी आल्यास, ती योग्य प्रकारे **हाताळली जाते** आणि प्रोग्राम क्रॅश होत नाही.
  - else फक्त **त्रुटी नसल्यास** चालतो.
  - finally हा **नेहमी चालतो**, त्रुटी आली तरी किंवा आली नाही तरी.

प्रोग्रॅम क्रॅश टाळण्यासाठी Exception Handling वापरणे गरजेचे आहे! 🚀

Real-World Example: ATM Cash Withdrawal System

Let's take a real-world example where exception handling is used in an **ATM system**:

```
try:
    balance = 5000
    withdraw = int(input("Enter amount to withdraw: "))
    if withdraw > balance:
        raise ValueError("Insufficient balance!")
    balance -= withdraw
    print("Transaction successful! Remaining balance:", balance)
except ValueError as e:
    print("Error:", e)
finally:
    print("Thank you for using our ATM!")
```

- **Output 1 (Valid case - withdrawal possible)**

```
Enter amount to withdraw: 2000
Transaction successful! Remaining balance: 3000
Thank you for using our ATM!
```

- **Output 2 (Insufficient balance case)**

```
Enter amount to withdraw: 6000
Error: Insufficient balance!
Thank you for using our ATM!
```

❖What is a User-Defined Exception in Python? (Most IMP)

- **User-Defined Exceptions** are custom exceptions created by programmers to handle specific errors in a program.
- Python allows us to define our own exception classes by inheriting from the built-in Exception class.
- This helps in providing meaningful error messages and better debugging.
- The raise keyword is used to trigger a user-defined exception when a condition is met.
- **Example of When to Use a User-Defined Exception:**
  - Checking **valid passwords** (minimum 8 characters, must contain numbers & special characters).
  - Validating **age for voting** (should be 18+).
  - Ensuring **bank transactions** do not exceed available balance.

◆ Python Program: User-Defined Exception for Password Validation

- **Scenario:**
  - We will create a user-defined exception called InvalidPasswordError. If the password does not meet the required conditions, an error will be raised.
- **Python Code:**

```
# Define a custom exception for password validation
class InvalidPasswordError(Exception):
    def __init__(self, message="Password is invalid! It must be at least 8 characters long and
      contain a number & special character."):
        self.message = message
        super().__init__(self.message)


# Function to check password validity
def check_password(password):
    if len(password) < 8 or not any(char.isdigit() for char in password) or not any(char in
      "!@#$%^&*()" for char in password):
```

```
        raise InvalidPasswordError()
    else:
        print("Password is valid!")

# User input and exception handling
try:
    user_password = input("Enter your password: ")
    check_password(user_password)
except InvalidPasswordError as e:
    print("Error:", e)
```

- **Output Examples:**

✅ **Valid Password Case:**

```
Enter your password: Abc@1234
Password is valid!
```

❌ **Invalid Password Case (Too Short):**

```
Enter your password: abc12
Error: Password is invalid! It must be at least 8 characters long and contain a number & special character.
```

📝 **Marathi Tip (मराठीत सोपे समजावून सांगितले आहे!)**

- **User-defined Exception म्हणजे प्रोग्रामरने स्वतः तयार केलेली त्रुटी (Error) आहे.**
  - raise कीवर्डचा वापर करून आपल्याला गरजेनुसार एरर टाकता येते.
  - वर दिलेल्या उदाहरणात, पासवर्ड योग्य नसल्यास InvalidPasswordError नावाची एरर निर्माण होते.
  - यात पासवर्ड **8 कॅरॅक्टरचा असला पाहिजे, नंबर आणि स्पेशल कॅरॅक्टर असावा** अशी अट दिली आहे.
  - अशा प्रकारे **कस्टम एरर वापरून डेटा व्हॅलिडेशन सोपे होते!**

UNIT 06. FILE I/O HANDLING AND EXCEPTION HANDLING.

❖ Summer 2022

1. Name different modes of python.. **(2marks)**
2. List different modes of opening file in python. **(2marks)**
3. Write a program to open a file in write mode and append some contents at the end of file **(6marks)**
4. Explain Try-except block used in exception handling in python with example. **(6marks)**

❖ Winter 2022

1. List file operations in Python. **(2marks)**
2. Explain how try-catch block is used for exception handling in python. **(4marks)**
3. Write python program to read contents of abc.txt and write same content to pqr.txt.. **(4marks)**

❖ Summer 2023

1. With neat example differentiate between readline( ) and readlines( ) functions in file-handling. **(4marks)**
2. Explain any four file modes in Python.. **(4marks)**
3. Write a program to show user defined exception in Python. **(4marks)**

❖ Winter 2023

1. State the use of read ( ) and readline ( ) functions in python file handling.. **(2marks)**
2. Describe various modes of file object ? Explain any two in detail. **(4marks)**
3. Explain seek ( ) and tell ( ) function for file pointer manipulation in python with example. **(4marks)**
4. WAP to read contents of first.txt file and write same content in second.txt file. **(4marks)**

❖ Summer 2024

1. Write the syntax of fopen.. **(2marks)**

2. Write a program function that accepts a string and calculate the number of uppercase letters and lower case letters. **(4marks)**
3. Write a Python program to create user defined exception that will check whether the password is correct or not. **(6marks)**
4. Describe various modes of file object. Explain any three in detail. **(6marks)**

❖ Winter 2024

1. List different modes of opening file in Python. **(2marks)**

2. Describe various modes of file object ? Explain any two in details . **(4marks)**

3. Write python code to check for zero division errors exception . **(4marks)**

4. Write a Python program to read contents from “a.txt” and write same contents in “b.txt.” **(4marks)**

5. Write a Python program to read contents of first.txt file and write same contents in second.txt file.. **(6marks)**

6. Explain Try-except-else-finally block used in exception handling in Python with example. **(6marks)**