* **which are the read and write methods of a file object explain with example**

In Python, the **open()** function is used to create a file object, which can then be used to perform various operations on a file, including reading from and writing to it. Here are the commonly used read and write methods of a file object, along with examples:

**1. Reading Methods:**

**read(size=-1)**: Reads and returns the specified number of bytes from the file. If no size is specified, or if the size is negative, it reads the entire file.

pythonCopy code# Example of using read() to read the entire file

with open('example.txt', 'r') as file:

content = file.read()

print(content)

**readline()**: Reads and returns a single line from the file.

pythonCopy code

# Example of using readline() to read one line at a time

with open('example.txt', 'r') as file:

line1 = file.readline()

line2 = file.readline()

print(line1)

print(line2)

)**readlines()**: Reads all lines from the file and returns them as a list of strings.

# Example of using readlines() to read all lines into a list

with open('example.txt', 'r') as file:

lines = file.readlines()

for line in lines:

print(line)

**2. Writing Methods:**

**write(text)**: Writes the specified text to the file. Returns the number of characters written.

# Example of using write() to write content to a file

with open('output.txt', 'w') as file:

file.write("Hello, world!\n")

file.write("This is a new line.")

**writelines(lines)**: Writes a list of lines to the file. The lines should be provided as a list of strings.

# Example of using writelines() to write multiple lines to a file

lines = ["Line 1\n", "Line 2\n", "Line 3\n"]

with open('output.txt', 'w') as file:

file.writelines(lines)

**flush()**: Flushes the internal buffer, ensuring that any buffered data is written to the file.

# Example of using flush() to write buffered data to a file

with open('output.txt', 'w') as file:

file.write("This will be buffered.")

file.flush() # Flushes the buffer to ensure data is written immediately

Remember to use the appropriate mode ('r' for reading, 'w' for writing, 'a' for appending) when opening the file. Additionally, it's a good practice to use the **with** statement (context manager) to ensure that the file is properly closed after operations are performed on it.

* **list the functions that can used to write a CSV file**

To write data to a CSV (Comma-Separated Values) file in Python, you can use the built-in **csv** module. This module provides various functions to work with CSV files, making it easy to write data in a structured format. Here are some commonly used functions from the **csv** module to write data to a CSV file:

**csv.writer()**: This function returns a writer object that you can use to write rows to the CSV file.

import csv

data = [

['Name', 'Age', 'Country'],

['Alice', 28, 'USA'],

['Bob', 32, 'Canada'],

['Charlie', 23, 'UK']

]

with open('output.csv', 'w', newline='') as file:

csv\_writer = csv.writer(file)

csv\_writer.writerows(data)

**csv.DictWriter()**: This function returns a DictWriter object that lets you write dictionaries as rows to the CSV file. Each dictionary represents a row, where keys are used as fieldnames.  
import csv

data = [

{'Name': 'Alice', 'Age': 28, 'Country': 'USA'},

{'Name': 'Bob', 'Age': 32, 'Country': 'Canada'},

{'Name': 'Charlie', 'Age': 23, 'Country': 'UK'}

]

fieldnames = ['Name', 'Age', 'Country']

with open('output\_dict.csv', 'w', newline='') as file:

csv\_writer = csv.DictWriter(file, fieldnames=fieldnames)

csv\_writer.writeheader() # Writes the header row

csv\_writer.writerows(data)

**writerow(row)**: This method allows you to write a single row to the CSV file. You can use it with a **csv.writer()** object.

import csv

with open('output\_single\_row.csv', 'w', newline='') as file:

csv\_writer = csv.writer(file)

csv\_writer.writerow(['Name', 'Age', 'Country'])

csv\_writer.writerow(['Alice', 28, 'USA'])

* **how do we raise an exception in python**

As a python developer you can choose to throw(raise) an exception if a condition occurs. You can raise any of Python's built-in exceptions by using the **raise** statement followed by the exception class name. For example, to raise a **ValueError**, you can do:

def example\_function(value):

if value < 0:

raise ValueError("Value must be non-negative")

return value

try:

result = example\_function(-5)

except ValueError as e:

print("An exception occurred:", e)

**Demonstrate the multiple exception handlin python**

Multiple exception handling in Python involves using multiple **except** blocks to catch different types of exceptions that may occur in your code.

try:

num = int(input("Enter a number: "))

result = 10 / num

print("Result:", result)

except ValueError:

print("Invalid input. Please enter a valid number.")

except ZeroDivisionError:

print("Cannot divide by zero.")

except Exception as e:

print("An unexpected error occurred:", e)

else:

print("No exceptions were raised.")

finally:

print("Execution complete.")

The **except** blocks come after the **try** block and catch specific types of exceptions. In this case, we have separate blocks for **ValueError**, **ZeroDivisionError**, and a generic **Exception** (catches all other exceptions).

* **the program to read the length and breadth from a file "in.txt".Compute the area and write the result to a file "output.txt".**

def compute\_area(length, breadth):

return length \* breadth

try:

with open("in.txt", "r") as input\_file:

length = float(input\_file.readline())

breadth = float(input\_file.readline())

except FileNotFoundError:

print("Input file 'in.txt' not found.")

exit(1)

except ValueError:

print("Invalid data in 'in.txt'. Please provide valid numeric values.")

exit(1)

area = compute\_area(length, breadth)

try:

with open("output.txt", "w") as output\_file:

output\_file.write("Length: {}\n".format(length))

output\_file.write("Breadth: {}\n".format(breadth))

output\_file.write("Area: {}".format(area))

except Exception as e:

print("An error occurred while writing to 'output.txt':", e)

else:

print("Area successfully computed and written to 'output.txt'.")

* **develope an object oriented python program wich reads details of students who wrote an exam from a file namely in.txt Name ,Marks. Find the number of students who scored more than 80 marks**

class Student:

def \_\_init\_\_(self, name, marks):

self.name = name

self.marks = marks

def main():

students = []

# Read student details from file and create Student objects

try:

with open("in.txt", "r") as file:

for line in file:

name, marks = line.strip().split(',')

marks = int(marks)

student = Student(name, marks)

students.append(student)

except FileNotFoundError:

print("File 'in.txt' not found.")

return

except ValueError:

print("Invalid data in 'in.txt'.")

return

# Count students who scored more than 80 marks

count\_high\_scorers = 0

for student in students:

if student.marks > 80:

count\_high\_scorers += 1

print("Number of students who scored more than 80 marks:", count\_high\_scorers)

if \_\_name\_\_ == "\_\_main\_\_":

main()

* **develope an object oriented python program to implement a program which has the class that perform Read a number,display a number and add two numbers**

class Calculator:

def \_\_init\_\_(self):

self.number = 0

def read\_number(self):

try:

self.number = float(input("Enter a number: "))

except ValueError:

print("Invalid input. Please enter a valid number.")

def display\_number(self):

print("Number:", self.number)

def add\_numbers(self, num1, num2):

return num1 + num2

def main():

calc = Calculator()

calc.read\_number()

calc.display\_number()

num1 = float(input("Enter the first number: "))

num2 = float(input("Enter the second number: "))

result = calc.add\_numbers(num1, num2)

print("Sum:", result)

if \_\_name\_\_ == "\_\_main\_\_":

main()