UNIT 4

Python File Operations

UNIT-4: Python File Operations: Reading files, writing files in python, Understanding read functions, read (), readline (), readlines (). Understanding write functions, write () and writelines () Manipulating file pointer using seek Programming, using file operations.

4.1 INTRODUCTION TO PYTHON FILE OPERATIONS:

In Python, **file operations** allow programs to **create**, **read**, **write**, and **modify** data stored in files. File handling is essential for storing information permanently, since data stored in variables is lost when the program ends.

The built-in open() function is used to work with files. Once opened, we can read/write data, and finally close the file to release resources.
Python supports two types of files:

- **Text files:** Store characters in human-readable format. (e.g., .txt)
- **Binary files:** Store raw bytes. (e.g., .bin, images, audio)

Basic File Handling Process:

1. **Open** the file using open()
2. **Perform** the required operation (read / write / append).
3. **Close** the file using close() (or use with statement to auto-close).

Example: Using with open() (auto-close)

```
# with automatically closes the file when the block ends
with open("hello.txt", "w") as f:
    f.write("This file auto closes")
```

Output:

*(file now contains **This file auto closes**)*

Explanation: with handles closing the file, so you don't need close().

Python also supports **file modes** like:

File Mode	Description
"r"	Read mode (default). Opens file for reading. Error if file doesn't exist.
"w"	Write mode. Creates new file or overwrites existing one.
"a"	Append mode. Adds data to the end of the file.
"r+"	Read and write mode. File must exist.
"rb"	Read in binary mode. Used for non-text files like images or audio.
"wb"	Write in binary mode. Used for non-text files like images or audio.

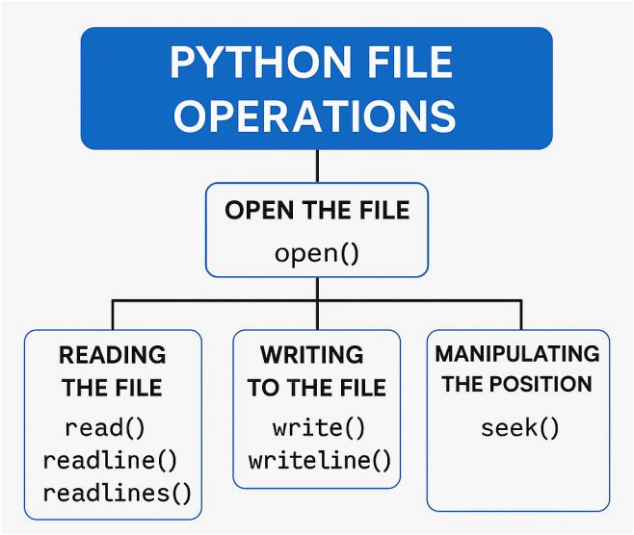

Figure 4.1: Python File Operations

4.2 BASIC FILE OPERATIONS SYNTAX

Opening Files:
Before performing any read or write action, a file must be opened. Python provides the open() function.

Syntax: Opening Files

```
file_object = open(filename, mode)
```

Common File Modes

Mode		Purpose
'r'	:	Read mode (default). Error if file doesn't exist.
'w'	:	Write mode. Creates new file or overwrites existing content.
'a'	:	Append mode. Adds data to the end of the file.
'r+'	:	Read and write mode. File must exist.
'x'	:	Exclusive creation. Fails if file already exists.
'b'	:	Binary mode (used with 'rb', 'wb' for images/audio).

Example: Open for reading

```
with open("readme.txt", "w") as f:
    f.write("Python is fun")
file = open("readme.txt", "r")
print(file.read())
file.close()
```

Output:
Python is fun

Explanation: File opened in "r" mode and read.

Example: Open for writing

```
file = open("write_demo.txt", "w")
file.write("First Line")
file.close()
```

Output:
(file contains First Line)

Explanation: "w" mode overwrites or creates file.

Example: Open for appending

```
file = open("append_demo.txt", "a")
file.write("Extra Line\n")
file.close()
```

Output:
(file adds Extra Line at end without deleting old data)

Explanation: "a" mode always appends.

Example: Using 'with' (no need to close manually)

```
with open("auto_close.txt", "w") as f:
    f.write("This file auto closes")
```

Output:
(file contains This file auto closes)

Explanation: with ensures file closes automatically.

Example: Error with exclusive create

```
file = open("unique.txt", "x")
file.write("New File Created")
file.close()
```

Output:
If unique.txt already exists → ***FileExistsError***

If not → (file created with text inside)

Explanation: "x" mode only works if file does not already exist.

Closing Files:
Always close the file after operations to release system resources.

Syntax: Closing Files

```
file_object.close()
```

Example: Opening and Closing Files

```
# Syntax: open(filename, mode)
file = open("example.txt", "w")          # Open file for writing
file.write("Hello File Handling")        # Perform operation
file.close()                             # Always close after use
```

Output:
(example.txt now contains Hello File Handling)

Explanation: open() creates/opens a file, write() saves data, and close() releases the file.

OPENING FILES AND CLOSING FILES

Before any read or write action, open the file

```
open(filename, mode)                # Always open file before reading
file = open("example.txt", "r")     # Open for reading
file.close()                        # Always close after use
```

4.3 READING FILES IN PYTHON:

Reading a file means accessing its content.

Read Methods

Method	Description
read()	Reads the **entire file** as a single string (or given number of characters).
readline()	Reads **one line at a time**
readlines()	Reads **all lines** and returns them as a list of strings

Example:

```
file = open("example.txt", "r")
print(file.read())                  # Reads full content
# OR
print(file.readline())              # Reads first line only
# OR
print(file.readlines())             # Returns list of all lines
file.close()
```

Using read()

Reads the entire file or a specified number of characters.

SYNTAX:

```
file.read(size)  # size is optional
```

Example: Partial read with read(n)

```
# sample.txt contains: Hello Rahul
with open("sample.txt", "r") as f:
   print(f.read(5))   # read first 5 characters
```

OUTPUT:

```
Hello
```

Explanation: read(5) reads only 5 characters from the current pointer.

Example: Reading File Content

```
f = open("sample.txt", "r")
content = f.read()
print(content)
f.close()
```

OUTPUT:

```
Hello Rahul
Welcome to JSS Academy of Technical Education, Noida, Uttar Pradesh.
```

If sample.txt contains:

Hello Rahul

Welcome to JSS Academy of Technical Education, Noida, Uttar Pradesh.

Explanation:

- open() opens the file in read mode.
- read() fetches entire text from the file.
- close() frees system resources.

Using readline()

Reads one line at a time.

SYNTAX:

```
file.readline()
```

Example: Using readline()

```
f = open("sample.txt", "r")
line = f.readline()
print(line)
f.close()
```

OUTPUT:

Hello Rahul

If sample.txt contains:

Hello Rahul
Welcome to JSS Academy of Technical Education, Noida, Uttar Pradesh.

Explanation:

- Reads only the first line from the file.

Using readlines()

Reads all lines and returns them as a **list**.

SYNTAX:

```
file.readlines()
```

Example: Using readlines()

```
f = open("sample.txt", "r")
lines = f.readlines()
print(lines)
f.close()
```

OUTPUT:

['Hello Rahul\n', 'Welcome to JSS Academy of Technical Education, Noida, Uttar Pradesh.\n']

If sample.txt contains:

Hello Rahul
Welcome to JSS Academy of Technical Education, Noida, Uttar Pradesh.

Explanation:

- Returns each line as an element in a list, including newline characters.

WRITING FILES IN PYTHON

Two commonly used methods for writing to files are:

- **write():** Writes a single string to a file.
- **writelines():** Writes multiple strings from a list/iterable.

Using write()

- Writes the entire string to the file or Overwrites file if opened with 'w'.
- Does **not** add a newline (\n) automatically.
- Returns the number of characters written.

SYNTAX:

```
file.write(string)
```

Example: Using simple write()

```
f = open("output.txt", "w")
f.write("Python is powerful!\n")
f.write("File handling is easy.")
f.close()
```

Output in output.txt:

```
Python is powerful!
File handling is easy.
```

Explanation:

- "w" mode creates a new file or overwrites existing content.

Example: Using write() Return value

```
file = open("demo_write.txt", "w")
count = file.write("Python is fun!\n")
print("Characters written:", count)
file.close()

# File content: Python is fun!
```

Output:

```
Characters written: 15
```

Note: Python is fun!

Explanation:

The returned value (15) is the number of characters written, including the newline.

Using writelines()

The writelines() method writes multiple strings from a list (or other iterable) into a file.

- It **does not add newlines automatically**, so you must include \n if you want line breaks.
- Useful when writing multiple lines in one go.

SYNTAX:

```
file.writelines(list_of_strings)
```

Example: Writing list of lines

```
f = open("output.txt", "w")
lines = ["Line 1\n", "Line 2\n", "Line 3\n"]
```

```
f.writelines(lines)
f.close()
```

Output in output.txt:

```
Line 1
Line 2
Line 3
```

Explanation:

- Writes multiple lines in one go.

Example: Mix write() and writelines()

```
f = open("output.txt", "w")
f.write("Python is powerful!\n")
f.writelines(["Line 1\n", "Line 2\n"])
f.close()
```

Output in output.txt:

```
Python is powerful!
Line 1
Line 2
```

Explanation: write() adds one string, then writelines() writes multiple lines.

Example: Using writelines() - Writing fruit names

```
data = ["Apple\n", "Banana\n", "Cherry\n"]
file = open("demo_writelines.txt", "w")
file.writelines(data)
file.close()
```

Output:

```
Apple
Banana
Cherry
```

Explanation: Each list element is written exactly as it appears (with \n).

Example: Using writelines() - Writing and then reading back

```
# writelines() example
lines = ["First line.\n", "Second line.\n", "Third line.\n"]

file = open("writelines_example.txt", "w")          # open file for writing
file.writelines(lines)                              # write all lines from list
file.close()

# read the file
file = open("writelines_example.txt", "r")
print(file.read())
file.close()
```

Output:

```
First line.
Second line.
Third line.
```

Explanation:

- writelines() writes **multiple strings from a list** to the file.
- Each string in the list must have \n if you want it on a new line.

Example: Writing multiple lines manually

```
# write() example
file = open("write_example.txt", "w")  # open file for writing
file.write("Hello World!\n")        # write first line
file.write("Python is fun!\n")      # write second line
file.close()

# read the file
file = open("write_example.txt", "r")
print(file.read())
file.close()
```

Output:

Hello World!
*Python is fun***!**

Explanation:

- write() writes one string at a time.
- \n is added manually to move to the next line.

Example: Writing numbers with writelines()

```
numbers = [str(n) + "\n" for n in range(1, 6)]
with open("nums.txt", "w") as f:
   f.writelines(numbers)
```

Output:

1
2
3
4
5

Explanation: Converts numbers into strings with \n and writes them all at once.

Example: Using a tuple

```
animals = ("Dog\n", "Cat\n", "Elephant\n")
with open("animals.txt", "w") as f:
   f.writelines(animals)
```

Output:

Dog
Cat
Elephant

Explanation: writelines() also works with tuples or any iterable of strings.

Example: Append mode ("a")

```
# If log.txt exists, this will add to the end; otherwise creates it
with open("log.txt", "a") as f:
   f.write("New entry\n")
```

Output:

(file keeps old content and adds New entry)

Explanation: "a" appends, it never overwrites existing content.

Example: write() + writelines() together

```
with open("mix.txt", "w") as f:
   f.write("Header\n")              # single string
```

```
    f.writelines(["Line 1\n", "Line 2\n"])   # list of strings

with open("mix.txt", "r") as f:
    print(f.read())
```

Output:

```
Header
Line 1
Line 2
```

Explanation: write() writes one string; writelines() writes many. Neither adds \n for you.

Differences Between write() and writelines()

Feature	write()	writelines()
Input Type	Single string	List or iterable of strings
Newline Handling	Must be added manually	Must be added manually
Use Case	Writing one string	Writing multiple strings
Returns	Number of characters written	None

4.4 Manipulating File Pointer Using seek()

The **file pointer** (or cursor) decides **where the next read/write happens** in a file.
By default, it starts at the **beginning** of the file. We can move it with **seek()**.

seek() in Python

The seek() method is used to manipulate the file pointer (also called the file cursor), which determines where the next read or write operation will occur. This is useful for:

- Reading or writing at specific positions
- Skipping parts of a file
- Rewinding to reread or overwrite data
- Accessing specific data without reading the entire file.

SYNTAX:

```
file_object.seek(offset, whence=0)
```

Explanation:

offset: Number of bytes to move the pointer (positive or negative).
whence **(optional)**: Reference point for offset:

0 → default: Beginning of file ***(default).***
1 → Current pointer position ***(binary mode only).***
2 → End of file ***(binary mode only).***

Return Value:

- None

Note:

- File must be opened in a mode that supports seeking: 'r', 'r+', 'w+', 'a+', 'rb', etc.
- In **text mode**, only seek(offset, 0) with non-negative offset is reliably portable.
- In **binary mode**, all whence values are supported.

Related Method: tell()

```
file_object.tell()
```

Explanation:

- Returns the current position of the file pointer (in bytes from the start).

Example: Checking position after reading using tell()

```
with open("demo.txt", "w") as f:
    f.write("Python")

with open("demo.txt", "r") as f:
    print(f.read(3))                    # Reads "Pyt"
    print("Pointer:", f.tell())
```

Output:

```
Pyt
Pointer: 3
```

Explanation: tell() shows the file pointer is now at byte 3 (just after "Pyt").

Example: Move pointer back and check using tell()

```
with open("demo.txt", "r") as f:
    print(f.read(2))                    # "Py"
    print("Pointer:", f.tell())
    f.seek(0)                           # reset to start
    print("Pointer:", f.tell())
```

Output:

```
Py
Pointer: 2
Pointer: 0
```

Explanation: tell()first shows 2 (after reading "Py"), then 0 after seek(0) resets to the beginning.

Example: Simple seek() + tell()

```
with open("demo.txt", "w") as f:
    f.write("Python")

with open("demo.txt", "r") as f:
    print(f.read(3))                    # 'Pyt'
    print("Pointer:", f.tell())  # 3
    f.seek(0)                           # jump back to start
    print(f.read())                     # 'Python'
```

Output:

```
Pyt
Pointer: 3
Python
```

Explanation: tell() shows the current byte position; **seek(0)** resets to the start.

Example: Read with seek() and tell()

```
with open("demo.txt", "w") as f:
    f.write("HelloWorld")

with open("demo.txt", "r") as f:
    print(f.read(5))                    # Reads "Hello"
    print("Pointer:", f.tell())         # Shows position 5
```

```
    f.seek(5)                    # Move to position 5
    print(f.read())              # Reads "World"
```

Output:

```
Hello
Pointer: 5
World
```

Explanation:

- tell() shows where the file pointer is.
- seek(5) moves it to position 5, so reading continues from "World".

Example: Read with seek() and tell() - Jump to middle

```
with open("sample.txt", "w") as f:
    f.write("ABCDEFGH")

with open("sample.txt", "r") as f:
    f.seek(4)                    # Jump to 4th position (E)
    print("Pointer:", f.tell())
    print(f.read(2))             # Reads "EF"
```

Output:

```
Pointer: 4
EF
```

Explanation:

- seek(4) moves pointer to 4 (0-based).
- Reading from there gives "EF".

Example: Simple seek to beginning

```
with open("seek_demo.txt", "w") as f:
    f.write("Python Rocks!")

with open("seek_demo.txt", "r") as f:
    print(f.read(6))             # Reads "Python"
    f.seek(0)                    # Move back to start
    print(f.read(6))             # Reads "Python" again
```

Output:

```
Python
Python
```

Explanation: seek(0) resets the pointer to the start.

Example: Start reading from 10th character

[Content of sample.txt file here]

Python file operations are powerful.

```
with open('sample.txt', 'r') as file:
    file.seek(10)                # Move to the 10th character
    print("Pointer position:", file.tell())  # Current position
    print(file.read())           # Read from 10th character on
```

Output:

```
Pointer position: 10
le operations are powerful.
```

Explanation:

- The pointer skips the first 10 characters.
- tell() confirms the position.
- file.read() reads the file starting from character 10.

Example: Seek and overwrite

```
with open("edit.txt", "w+") as file:
    file.write("This is the original content.")

    file.seek(8)            # Move after "This is "
    file.write("modified")     # Overwrites starting at pos 8

    file.seek(0)
    print(file.read())
```

Output:

```
This is modified content.
```

Explanation: Writing after seek overwrites existing text.

Example: Using whence

```
with open("demo.txt", "w") as f:
    f.write("abcdefgh")

with open("demo.txt", "r") as f:
    f.seek(3, 0)         # From start → move to 3rd byte
    print(f.read(2))      # Reads "de"

    f.seek(-2, 2)         # From end → move 2 bytes back
    print(f.read())       # Reads "gh"
```

Output:

```
de
gh
```

Explanation:

- seek(3,0) jumps to position 3 → starts at "d".
- seek(-2,2) jumps 2 bytes before end → starts at "g".

Example: Binary mode with seek

```
with open("binary.txt", "wb") as f:
    f.write(b"1234567890")

with open("binary.txt", "rb") as f:
    f.seek(4, 0)                # Move to 5th byte
    print(f.read(3))              # Reads b'567'

    f.seek(-3, 2)                # 3 bytes before end
    print(f.read())               # Reads b'890'
```

Output:

```
b'567'
b'890'
```

Explanation: In **binary mode**, we can also use whence=1 and negative offsets.

Example: Very Simple Demo (Beginner Friendly)

```
with open("hello.txt", "w") as f:
    f.write("HelloWorld")

with open("hello.txt", "r") as f:
    print(f.read(5))   # Reads Hello
```

```
    f.seek(5)       # Jump to position 5
    print(f.read())   # Reads World
```

Output:

```
Hello
World
```

Explanation:

- f.read(5) reads the first 5 characters → "Hello".
- f.seek(5) moves the pointer to the 5th position (right after "Hello").
- f.read() then reads from that position until the end → "World".

Example: Using seek() to Read from a Specific Position

[Content of sample.txt file here]

```
Hello, this is a sample file.
Line 2: Python is great
```

```
# Program to demonstrate seek() for reading from a specific position
with open('sample.txt', 'w') as file:
  file.write('Hello, this is a sample file.\nLine 2: Python is great!')

with open('sample.txt', 'r') as file:
# Move the file pointer to the 7th byte (after "Hello, ")
  file.seek(7)
  content = file.read()
  print("Content read after seeking to position 7:")
  print(content)
```

Output:

```
Content read after seeking to position 7:
this is a sample file.
Line 2: Python is great!
```

[Content of sample.txt file here]

```
this is a sample file.
Line 2: Python is great!
```

Explanation:

- The program first creates a file sample.txt and writes two lines to it.
- The file is then opened in read mode ('r').
- The seek(7) call moves the file pointer to the 7th byte (0-based index), which is after "Hello, ".
- The read() method reads from the 7th byte to the end of the file, capturing everything after "Hello, ".
- The with statement ensures the file is properly closed after operations.

Example: Using seek() to Read from a Specific Position

[Content of test.txt file here]

```
Hello, Python!
```

```
# Create a file and read from a specific position
with open('test.txt', 'w') as file:
  file.write('Hello, Python!')

with open('test.txt', 'r') as file:
  file.seek(7)  # Move to position 7 (after "Hello, ")
```

```
content = file.read()
print("Content from position 7:", content)
```

Output:

```
Content from position 7: Python!
```

[Content of sample.txt file here]

```
Hello, Python!
```

Explanation:

- The file test.txt is created with the content "Hello, Python!" (13 bytes).
- The file is opened in read mode ('r').
- seek(7) moves the pointer to position 7 (after "Hello, "), where the space is.
- read() reads from position 7 to the end, outputting "Python!".

Example: Using seek() with Different whence Values

[Content of demo.txt file here]

```
# Demonstrate seek() with whence values

with open('demo.txt', 'w') as file:
  file.write('abcdefgh')

with open('demo.txt', 'r') as file:
  print("Initial position:", file.tell())
  file.seek(3, 0)  # Move to 3rd byte from start
  print("Position after seek(3, 0):", file.tell())
  print("Content:", file.read(2))  # Read 2 bytes

  file.seek(-2, 2)  # Move 2 bytes back from end
  print("Position after seek(-2, 2):", file.tell())
  print("Content:", file.read())
```

Output:

```
Initial position: 0
Position after seek(3, 0): 3
Content: de
Position after seek(-2, 2): 6
Content: gh
```

[Content of sample.txt file here]

```
abcdefgh
```

Explanation:

- The file demo.txt is created with "abcdefgh" (8 bytes).
- tell() shows the pointer starts at 0.
- seek(3, 0) moves to position 3 (character 'd').
- read(2) reads 'de'.
- seek(-2, 2) moves 2 bytes back from the end (8 - 2 = 6, character 'g').
- read() reads 'gh' to the end.

Example: Using seek() to Read from a Specific Position

[Content of sample.txt file here]

abcdefghij

```
# Program to demonstrate seek() with different whence values

with open('sample.txt', 'w') as file:
   file.write('abcdefghij')

with open('sample.txt', 'r') as file:
   print("Initial position:", file.tell())                    # Get initial position
   file.seek(5, 0)                                            # Seek to 5th byte from
beginning
   print("Position after seek(5, 0):", file.tell())
   print("Content from position 5:", file.read(2))          # Read 2 bytes

   file.seek(-3, 2)                                          # Seek 3 bytes backward from end
   print("Position after seek(-3, 2):", file.tell())
   print("Content from position 7:", file.read())

   file.seek(2, 1)                                           # Seek 2 bytes forward from current
position
   print("Position after seek(2, 1):", file.tell())
  print("Content from position 9:", file.read())
```

Output:

```
Initial position: 0
Position after seek(5, 0): 5
Content from position 5: fg
Position after seek(-3, 2): 7
Content from position 7: hij
Position after seek(2, 1): 9
Content from position 9: j
```

[Content of sample.txt file here]

abcdefghij

Explanation:

- The file sample.txt is created with the content "abcdefghij" (10 bytes).
- The file is opened in read mode ('r').
- tell() shows the initial position as 0 (start of the file).
- seek(5, 0) moves the pointer to the 5th byte (position 5, character 'f').
- read(2) reads 2 bytes ('fg').
- seek(-3, 2) moves the pointer 3 bytes backward from the end (position 10 - 3 = 7, character 'h').
- read() reads from position 7 to the end ('hij').
- seek(2, 1) moves the pointer 2 bytes forward from the current position (7 + 2 = 9, character 'j').
- read() reads the remaining content ('j').

Example: Using seek() for Overwriting Specific Parts of a File

[Content of sample.txt file here]

abcdefghij

```
# Program to demonstrate seek() for overwriting part of a file

with open('edit.txt', 'w+') as file:
   file.write('This is the original content.')
   print("Initial content:", file.read())          # Won't read anything (pointer at end)

   file.seek(8)  # Move to position 8 (after "This is ")
   file.write('modified')                          # Overwrite starting at position 8

   file.seek(0)  # Move back to the beginning
   content = file.read()
   print("Content after modification:")
   print(content)
```

Output:

```
Initial content:
Content after modification:
This is modified content.
```

[Content of sample.txt file here]

This is modifiedinal content.

Explanation:

- The file edit.txt is opened in read-write mode ('w+'), which creates a new file or overwrites an existing one and allows both reading and writing.
- The initial write() writes "This is the original content.".
- The first read() returns nothing because the file pointer is at the end after writing.
- seek(8) moves the pointer to position 8 (after "This is ").
- The write('modified') overwrites the next 8 bytes with "modified", resulting in "This is modified content.".
- seek(0) moves the pointer back to the beginning, and read() reads the entire modified content.

Example: Using seek()

[Content of sample.txt file here]

abcdefghij

```
# Writing sample data into file
file = open("seek_example.txt", "w")
file.write("ABCDEFGH")  # 8 characters
file.close()

# Reading and moving the file pointer
file = open("seek_example.txt", "r")

print("Reading first 3 characters:")
print(file.read(3))   # Reads "ABC"

# Move pointer back to the start (offset=0, from beginning)
file.seek(0)
print("\nAfter seek(0):")
print(file.read(4))   # Reads "ABCD"
```

```
# Move pointer to position 5 (0-based index)
file.seek(5)
print("\nAfter seek(5):")
print(file.read())    # Reads from 'F' to end

file.close()
```

Output:

```
Reading first 3 characters:
ABC

After seek(0):
ABCD

After seek(5):
FGH
```

Explanation:

- First, the file seek_example.txt contains:
  A B C D E F G H
  0 1 2 3 4 5 6 7 (positions in bytes)
- **file.read(3)** → Reads "ABC", pointer now at position 3.
- **file.seek(0)** → Moves pointer to start, so reading starts again from "A".
- **file.seek(5)** → Moves pointer to "F", reading continues from there to end.

Example: Binary Mode with seek()

[Content of *'binary.txt'* file here]

```
binary data: 1234567890
```

```
# Seek in binary mode
with open('binary.txt', 'wb') as file:
  file.write(b'1234567890')

with open('binary.txt', 'rb') as file:
  file.seek(4, 0)                          # Move to 4th byte from start
  print("Content from position 4:", file.read(3).decode())
  file.seek(-3, 2)                         # Move 3 bytes back from end
  print("Content from position 7:", file.read().decode())
```

Output:

```
Content from position 4: 567
Content from position 7: 890
```

Explanation:

- The file binary.txt is created with 10 bytes (b'1234567890').
- In binary read mode ('rb'), seek(4, 0) moves to position 4 (byte '5').
- read(3) reads 3 bytes (b'567'), decoded to '567'.
- seek(-3, 2) moves to position 7 (10 - 3 = 7, byte '8').
- read() reads b'890', decoded to '890'.

Example: Seek Backward from End

[Content of data.txt file here]
Python file seek example demo.

```
with open('data.txt', 'rb') as file:
file.seek(-5, 2)                    # Go 5 bytes before the end (binary mode)
print("Pointer position:", file.tell())
print(file.read().decode('utf-8'))  # Read the last 5 bytes
```

Output:

```
Pointer position: 26
demo.
```

Explanation:

- File is opened in 'rb' (binary) mode for negative offset.
- seek(-5, 2) means: Move backward 5 bytes from the end.
- file.read() gets the last 5 bytes, which is "demo.".

Note:

With the above examples, learners can:

1. *Reset pointer (seek(0))*
2. *Jump to a character position*
3. *Overwrite at specific place*
4. *Use whence for flexible navigation*
5. *Handle binary files safely*

Parameter in Python's seek() function

The from_what parameter in Python's seek() function controls where the file pointer's movement is calculated from. The values have these specific meanings:

Value		Typical Usage
0	**:**	The reference point is the beginning of the file. seek(offset, 0) moves the pointer to offset bytes from the start of the file, regardless of the current position.
1	**:**	The reference point is the current file position. seek(offset, 1) moves the pointer offset bytes forward (or backward, if offset is negative) from wherever it currently is. *This mode requires the file to be opened in binary mode* ('rb').
2	**:**	The reference point is the end of the file. seek(offset, 2) moves the pointer offset bytes from the file's end; usually, you use a negative offset to move backward. *This also requires binary mode*.

Example: Demonstrating seek() with Forward and Backward Movement

[Content of example.txt file here]

```
abcdefghij
klmnopqrst
uvwxyz
```

```
   # Create the file for demonstration (you can skip this in real usage)
with open("example.txt", "w") as f:
   f.write("abcdefghij\nklmnopqrst\nuvwxyz\n")

# Now, demonstrate seek() usage
with open("example.txt", "rb") as file:
   # 1. Forward Movement: Move to the 5th byte from start
   file.seek(5, 0)
   print("Position after forward seek:", file.tell())
   print("Reading after seeking forward:", file.read(5).decode())

   # 2. Backward Movement: Move 5 bytes back from the current position
   file.seek(-5, 1)
   print("\nPosition after backward seek:", file.tell())
   print("Reading after seeking backward:", file.read(5).decode())

   # 3. Move backwards from the end: 7 bytes before end
   file.seek(-7, 2)
   print("\nPosition 7 bytes before end:", file.tell())
   print("Reading the last 7 bytes:", file.read().decode())
```

Output:

```
Position after forward seek: 5
Reading after seeking forward: fghij

Position after backward seek: 5
Reading after seeking backward: fghij

Position 7 bytes before end: 26
Reading the last 7 bytes: tuvwxyz
```

Explanation:

Forward Movement:

- file.seek(5, 0): Jumps to the 5th byte from the start (0 = beginning).
- Reads 5 bytes: This prints "fghij".

Backward Movement:

- After reading, pointer moves forward; then file.seek(-5, 1) moves it 5 bytes back from the current position (1 = relative to current).
- Reads again: Prints "fghij", because the pointer was returned and the same chunk is read again.

Backward from End:

- file.seek(-7, 2): Moves pointer 7 bytes before the end of the file (2 = relative to end).
- Reads rest: Prints the last 7 bytes ("tuvwxyz").

Typical Values

Value	Reference Point	Description	Mode Needed
0	Beginning of file	Absolute positioning from file start	Text/Binary
1	Current file pointer	Relative movement to current pointer	Binary only
2	End of file	Relative movement from end (often negative offset)	Binary only

Example: *(Error manipulating)* Using seek() for Overwriting Specific Parts of a File

[Content of sample.txt file here]

Hello, this is a sample file.
Line 2: Python is great!

```
with open('sample.txt', 'r') as file:
    file.seek(5, 0)  # Move to position 5
    print("Content from position 5:", file.read())
    file.seek(-10, 1)                    # Try to move backward (may cause error)
except IOError as e:
    print(f"Error manipulating file pointer: {e}")
```

Output:

Content from position 5: this is a sample file.
Line 2: Python is great!
Error manipulating file pointer: [Errno 22] Invalid argument

Explanation:

- The program attempts to seek to position 5 and read the content, which succeeds.
- The seek(-10, 1) tries to move 10 bytes backward from the current position, which may fail if the resulting position is negative, triggering the IOError.

Table: Summary of File Modes

Mode	Purpose	Example
"r"	Read file (error if not exist)	open("file.txt", "r")
"w"	Write (create/overwrite file)	open("file.txt", "w")
"a"	Append (add to end)	open("file.txt", "a")
"r+"	Read & write (file must exist)	open("file.txt", "r+")
"x"	Create new file (error if exists)	open("new.txt", "x")
"rb"	Read binary (images/audio)	open("pic.jpg", "rb")
"wb"	Write binary (images/audio)	open("pic.jpg", "wb")

Exception Handling with File Operations

Using try–except–finally ensures that even if an error occurs (e.g., file not found), the program doesn't crash.

Example: Exception Handling with File Operations

(if file does not exist):

```
try:
    f = open("missing.txt", "r")   # Trying to read a file that may not exist
    content = f.read()
    print(content)
except FileNotFoundError:
    print("File not found! Please check the filename.")
finally:
    print("File handling process completed.")
```

Output: (if file does not exist):

File not found! Please check the filename.
File handling process completed.

Explanation:

- **try** → attempts to run risky code.
- **except** → handles the error gracefully.
- **finally** → runs no matter what (good place to close files or clean up).

Log File Analyzer

Imagine a log file with system messages. We want to **count how many times "ERROR" occurs**.

Example: Count how many times "ERROR" occurs.

Sample file log.txt content:

```
INFO: System started
ERROR: Disk not found
INFO: User logged in
ERROR: Out of memory
```

```
error_count = 0

with open("log.txt", "r") as f:
    for line in f:
        if "ERROR" in line:
            error_count += 1

print("Total ERROR messages:", error_count)
```

Output:

```
Total ERROR messages: 2
```

Explanation:

- Opens the file safely with with open(...).
- Reads line by line and checks if "ERROR" exists.
- Counts total error messages → very common in server log analysis.

CSV Writing

CSV files are widely used for storing tabular data (Excel-like). Python makes it very simple.

Example: Writing Student Records into CSV

```
import csv

# Data to write
data = [
    ["Name", "Age", "Branch"],
    ["Bharat", 37, "ECE"],
    ["Rahul", 35, "ECE_VLSI"],
    ["Vimal", 18, "CSE"],
]

with open("students.csv", "w", newline="") as f:
    writer = csv.writer(f)
    writer.writerows(data)

print("CSV file created successfully!")
```

Console Output:

CSV file created successfully!

Content of students.csv (created file)

```
Name,Age,Branch
Bharat,37,ECE
Rahul,35,ECE_VLSI
Vimal,18,CSE
```

Explanation

1. **data list** → Each inner list is one row for the CSV file.
   - ✓ Row 1 → Column headers.
   - ✓ Row 2 → Bharat's details.
   - ✓ Row 3 → Rahul's details.
   - ✓ Row 4 → Vimal's details.
2. **open("students.csv", "w", newline="")**
   - ✓ "w" → Opens in write mode (overwrites existing file).
   - ✓ newline="" → Prevents blank lines in file.
3. **writer.writerows(data)** → Writes all rows into the file.
4. **Printed message** confirms successful completion.

Example: Reading the Entire CSV File

Content of students.csv:

```
Name,Age,Branch
Bharat,37,ECE
Rahul,35,ECE_VLSI
Vimal,18,CSE
```

```
import csv

with open("students.csv", "r") as f:
    reader = csv.reader(f)
    for row in reader:
        print(row)
```

Output:

```
['Name', 'Age', 'Branch']
['Bharat', '37', 'ECE']
['Rahul', '35', 'ECE_VLSI']
['Vimal', '18', 'CSE']
```

Explanation: Reads each row as a list.

Example: Reading Only Names from CSV File

Content of students.csv:

```
Name,Age,Branch
Bharat,37,ECE
Rahul,35,ECE_VLSI
Vimal,18,CSE
```

```
import csv

with open("students.csv", "r") as f:
```

```
    reader = csv.reader(f)
    next(reader)  # skip header
    for row in reader:
        print(row[0])
```

Output:

Bharat
Rahul
Vimal

Explanation: Accesses only the first column (Name).

Example: Using DictReader for Easy Access from CSV File

Content of students.csv:

```
Name,Age,Branch
Bharat,37,ECE
Rahul,35,ECE_VLSI
Vimal,18,CSE
```

```
import csv

with open("students.csv", "r") as f:
    reader = csv.DictReader(f)
    for row in reader:
        print(row["Name"], "is in branch", row["Branch"])
```

Output:

Bharat is in branch ECE
Rahul is in branch ECE_VLSI
Vimal is in branch CSE

Explanation: Automatically maps column headers → dictionary keys.

Example: Appending a New Student into CSV File

Content of students.csv:

```
Name,Age,Branch
Bharat,37,ECE
Rahul,35,ECE_VLSI
Vimal,18,CSE
```

```
import csv

with open("students.csv", "a", newline="") as f:
    writer = csv.writer(f)
    writer.writerow(["Manoj", 38, "ECE"])

print("New student added!")
```

Console Output:

New student added!

Content of students.csv after appending:

```
Name,Age,Branch
Bharat,37,ECE
Rahul,35,ECE_VLSI
Vimal,18,CSE
```

```
Manoj,38,ECE
```

Explanation

1. **open("students.csv", "a", newline="")**
   - ✓ "a" → append mode (adds data at the end, doesn't overwrite).
   - ✓ newline="" → prevents blank lines between rows.
2. **writer.writerow([...])**
   - ✓ Adds **one new row** into the CSV file.
3. **Print Statement** → Confirms success.

Example Python Programs

Q1. Read full file with read()

```
with open("file1.txt", "w") as f:
  f.write("Hello\nWorld")
with open("file1.txt", "r") as f:
  print(f.read())
```

Output:

Hello
World

Explanation: read() returns the whole file content as a single string.

Q2. Read one line with readline()

```
with open("file2.txt", "w") as f:
  f.write("First line\nSecond line")
with open("file2.txt", "r") as f:
  print(f.readline())
```

Output:

First line

Explanation: readline() reads only the first line.

Q3. Read all lines with readlines()

```
with open("file3.txt", "w") as f:
  f.write("A\nB\nC")
with open("file3.txt", "r") as f:
  print(f.readlines())
```

Output:

['A\n', 'B\n', 'C']

Explanation: readlines() returns list of lines.

Q4. Write single string with write()

```
with open("file4.txt", "w") as f:
  f.write("ECE Dept")
```

Output:

(file contains ECE Dept)

Explanation: write() saves a string to file.

Q5. Write multiple lines with writelines()

```
with open("file5.txt", "w") as f:
  f.writelines(["Apple\n", "Banana\n", "Cherry\n"])
```

Output:
(file contains three lines)
Explanation: writelines() writes list of strings (newline must be added manually).

Q6. Append to file

```
with open("file6.txt", "w") as f:
  f.write("Line 1\n")
with open("file6.txt", "a") as f:
  f.write("Line 2\n")
with open("file6.txt", "r") as f:
  print(f.read())
```

Output:

```
Line 1
Line 2
```

Explanation: mode 'a' appends without overwriting.

Q7. Seek and Tell demo

```
with open("file7.txt", "w") as f:
  f.write("Hello Python")
with open("file7.txt", "r") as f:
  f.seek(6)
  print("Position:", f.tell())
  print(f.read())
```

Output:

```
Position: 6
Python
```

Explanation: seek() moves pointer, tell() gives current position.

Q8. Overwrite bytes in binary file

```
with open("file8.bin", "wb") as f:
  f.write(b"ABCDEFGH")
with open("file8.bin", "r+b") as f:
  f.seek(2)
  f.write(b"zz")
  f.seek(0)
  print(f.read())
```

Output:

```
b'ABzzEFGH'
```

Explanation: Binary mode allows overwriting exact bytes.

Q9. Copy file with readlines and writelines

```
with open("src.txt", "w") as f:
  f.write("one\ntwo\nthree\n")
with open("src.txt", "r") as src, open("dst.txt", "w") as dst:
  dst.writelines(src.readlines())
with open("dst.txt", "r") as f:
  print(f.read())
```

Output:

```
one
two
three
```

Explanation: Copies lines from one file to another.

Q10. Handle missing file

```
try:
  with open("nofile.txt", "r") as f:
    print(f.read())
except FileNotFoundError:
  print("File not found")
```

Output:

```
File not found
```

Explanation: Shows exception handling.

Q11. Count words in a file

```
with open("words.txt", "w") as f:
  f.write("Python is simple")
with open("words.txt", "r") as f:
  print(len(f.read().split()))
```

Output:

```
3
```

Explanation: Splits by spaces to count words.

Q12. Read first 5 characters

```
with open("alpha.txt", "w") as f:
  f.write("ABCDEFGHIJ")
with open("alpha.txt", "r") as f:
  print(f.read(5))
```

Output:

```
ABCDE
```

Explanation: read(n) reads only n characters.

Q13. Iterate file line by line

```
with open("poem.txt", "w") as f:
    f.write("Line1\nLine2\nLine3")
with open("poem.txt", "r") as f:
    for line in f:
        print(line.strip())
```

Output:

Line1
Line2
Line3

Explanation: File is iterable; each loop gives a line.

Q14. Write numbers 1–5

```
with open("nums.txt", "w") as f:
    for i in range(1, 6):
        f.write(str(i) + "\n")
```

Output:
(file contains numbers 1–5 each on new line)
Explanation: Loop writes numbers line by line.

Q15. Seek to middle of file

```
with open("mid.txt", "w") as f:
    f.write("ABCDEFGHIJ")
with open("mid.txt", "r") as f:
    f.seek(5)
    print(f.read())
```

Output:

FGHIJ

Explanation: Pointer jumps to 5th index.

Q16. Overwrite file content

```
with open("overwrite.txt", "w") as f:
    f.write("Old Content")
with open("overwrite.txt", "w") as f:
    f.write("New Content")
```

Output:
(file contains New Content)
Explanation: Opening in 'w' overwrites the file.

Q17. Use with block automatically closes file

```
with open("auto.txt", "w") as f:
    f.write("Hello")
```

Output:
(file contains Hello)
Explanation: with closes file automatically.

Q18. Check file pointer after read

```
with open("ptr.txt", "w") as f:
    f.write("Python")
with open("ptr.txt", "r") as f:
    f.read()
    print(f.tell())
```

Output:

```
6
```

Explanation: Pointer at end after reading all 6 characters.

Q19. Write list with join()

```
fruits = ["apple", "banana", "cherry"]
with open("fruits.txt", "w") as f:
    f.write("\n".join(fruits))
```

Output:
(file contains apple, banana, cherry on separate lines)
Explanation: join() adds \n between list items.

Q20. Copy binary file

```
with open("img.bin", "wb") as f:
    f.write(b"12345")
with open("img.bin", "rb") as src, open("img_copy.bin", "wb") as dst:
    dst.write(src.read())
with open("img_copy.bin", "rb") as f:
    print(f.read())
```

Output:

```
b'12345'
```

Explanation: Reads binary data and writes to another file.