07/10/2025

File Handling

File handling in Python allows you to create, read, write, and delete files.

It is used to store and manage data permanently even after the program ends.

Python provides a built-in function open() to work with files.

Syntax

open("file_path", "mode")

Parameters:

- "file_path" → The location (path) of the file.
- "mode" → The mode in which the file is opened (read, write, etc.).

Common modes:

"r" - read

"w" - write

"a" - append

"b" - binary

"t" - text mode

"rb"- read binary

"wb"- write binary

"wt"- write text

Types of File Paths

1) **Relative path**
   Path relative to the current working directory.
   Example:
   open("mydata.txt", "r")

2) **Absolute path**

   Full path including drive name.

   Example:

   open(r"C:\Users\Mahesh H N\Desktop\family.txt", "r")

Opening a File for Reading

d1 = open("mydata.txt","r")

- Opens the file mydata.txt in read mode.

Reading the File

d1.read()

o/p:

'thanushree h m\n22\nbengaluru\nkarntaka'

- Reads and returns the entire content as a string.

Closing the File

d1.close()

- Always close files after use to free system resources.

Error Example

If you try to read a file after closing:

d1.read()

o/p:

ValueError: I/O operation on closed file.

Example with Absolute Path

d2 = open(r"C:\Users\Mahesh H N\Desktop\family.txt", "r")

print(d2.read())

d2.close()

→ Reads and prints the content of the file located at that absolute path.

Writing to a File

To **create** or **overwrite** a file:

```
f1 = open("sample.txt", "w")

f1.write("hello good morning everyone")

f1.close()
```

- If the file exists → it will be overwritten.
- If it doesn't → a new file will be created.

o/p:

hello good morning everyone

Appending Data to a File

To add content **without deleting** existing data:

```
f2 = open("sample.txt", "a")

f2.write("\nhow are you")

f2.close()
```

o/p:

hello good morning everyone

how are you

Deleting a File

Python's os module can delete files.

```
import os

os.remove("sample.txt")
```

→ Deletes sample.txt permanently.

To check before deleting:

```
if os.path.exists("sample.txt"):
```

```
    os.remove("sample.txt")
else:
    print("File not found")
```

Writing with Absolute Path

```
f1 = open(r"C:\Users\Mahesh H N\Desktop\info.txt", "w")
f1.write("hello good morning everyone")
f1.close()
```

→ Writes data to the file at the specified path.

Errors and Exceptions

Errors (also known as exceptions) are issues that stop a program from running properly. Error handling means detecting and managing those errors gracefully instead of letting the program crash.

Types of Errors

1. **Syntax Error**

   Occurs when the Python code is written incorrectly.

2. **Name Error**

   Raised when you try to use a variable that hasn't been defined.

3. **Indentation Error**

   Occurs when spaces or tabs are not used correctly to define code blocks.

4. **Type Error**

   Raised when an operation is performed on an inappropriate data type.

5. **ZeroDivision Error**

   Occurs when a number is divided by zero.

6. **Index Error**

   Raised when trying to access an invalid index in a list or tuple.

7. **Value Error**

   Occurs when a function receives an argument of the right type but an inappropriate value.

8. **Key Error**

   Raised when trying to access a dictionary key that doesn't exist.

9. **FileNotFound Error**

   Happens when trying to open a file that doesn't exist.

Exception Handling Syntax

To prevent the program from crashing, we use the **try-except** block.

```
try:
    # Code that might cause an error
except:
    # Code to handle the error
```

Without Exception Handling

```
x = int(input("Enter a number: "))
print(10 / x)
```

o/p:

0

ZeroDivisionError: division by zero

With Exception Handling

```
try:
    x = int(input("Enter a number: "))
    print(10 / x)
```

```
except:
    print("Something went wrong")
```

o/p:

```
0
Something went wrong
```