PYTHON FILE HANDLING — FULL DETAILED NOTES

Python file handling allows you to **create**, **read**, **write**, and **modify** files stored on your computer.

---

1. What is a File?

A file is a collection of data stored in a computer.
It can be:

- Text file → `.txt`, `.csv`, `.log`
- Binary file → images, videos, PDFs, `.exe`

Python provides built-in functions to work with files.

---

2. Opening a File → `open()`

Syntax:

```
file = open("filename", "mode")
```

Modes of opening files:

Mode	Meaning	Description
`"r"`	Read	Opens file for reading (default). Error if file doesn't exist
`"w"`	Write	Opens file for writing, creates new file, **overwrites content**
`"a"`	Append	Opens file for writing, adds data at end (no overwrite)
`"x"`	Create	Creates a new file. Error if file already exists
`"r+"`	Read + Write	No overwrite
`"w+"`	Write + Read	File truncated (cleared)
`"a+"`	Append + Read	File pointer at end

---

3. Reading from File

3.1 `read()` — reads entire file as a string

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

3.2 `readline()` — reads one line at a time

```
f = open("data.txt", "r")
line1 = f.readline()
```

```
line2 = f.readline()
```

3.3 `readlines()` — returns list of all lines

```
lines = f.readlines()
```

---

4. Writing to a File

4.1 `write()` — write string

```
f = open("data.txt", "w")
f.write("Hello World")
f.close()
```

➜ Overwrites the entire file.

4.2 `writelines()` — write list of lines

```
f.writelines(["A\n", "B\n", "C\n"])
```

---

5. Appending to File

```
f = open("data.txt", "a")
f.write("\nNew line added!")
f.close()
```

---

6. Closing the File

Every file must be closed after work:

```
f.close()
```

But the best method is to use:

---

7. Using `with open()` — BEST PRACTICE

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

Advantages:

✓ Automatically closes file
✓ Cleaner syntax
✓ No need to call `close()`

---

8. File Cursor (Pointer)

Every file has a pointer that tracks:

> From where Python will read or write

Use:

Method	Use
`f.tell()`	Shows current pointer location
`f.seek(pos)`	Moves pointer to position

Example:

```
f.seek(0)   # Move to start of file
```

---

9. Working with Binary Files

Example: Images, videos, PDFs

```
with open("img.png", "rb") as f:
    data = f.read()
```

`"rb"` → read binary
`"wb"` ← write binary

---

10. Deleting Files

```
import os
os.remove("file.txt")
```

EXCEPTION HANDLING IN PYTHON — FULL DETAILED NOTES

Exception handling is used to handle **runtime errors** so the program does not crash.

---

1. What is an Exception?

An exception is an error during program execution.

Examples:

Error	Meaning
`ZeroDivisionError`	Division by zero
`ValueError`	Wrong value format
`TypeError`	Wrong data type
`FileNotFoundError`	File does not exist
`KeyError`	Key missing in dictionary

---

2. Why Exception Handling?

Without exception handling:

✕ Program crashes
✕ User gets confusing messages

With exception handling:

✓ Program continues
✓ Error is handled safely
✓ User gets safe messages

---

3. try–except Block

Basic structure:

```
try:
    risky code
except:
    code to execute when error happens
```

Example:

```
try:
    a = 10 / 0
except:
    print("Cannot divide by zero")
```

---

4. Catching Specific Exception

```
try:
    num = int(input("Enter number: "))
except ValueError:
    print("Invalid input! Enter a number only.")
```

---

5. Multiple Exceptions

```
try:
    f = open("abc.txt")
    data = f.read()
except FileNotFoundError:
    print("The file does not exist")
except PermissionError:
    print("Permission denied")
except Exception as e:
    print("Other error:", e)
```

---

6. else Block

Executed only when **no exception** occurs.

```
try:
    x = int(input("Enter a number: "))
except:
    print("Error!")
else:
    print("Thanks! No error.")
```

---

7. finally Block

Always executes
✓ Error or no error
✓ Used to close files, connections, etc.

```
try:
    f = open("data.txt")
    print(f.read())
except:
    print("Error!")
finally:
    print("Closing file...")
    f.close()
```

---

8. Raising Your Own Exception

You can forcefully trigger an error using:

```
raise Exception("Something went wrong!")
```

Or specific errors:

```
age = -5
if age < 0:
    raise ValueError("Age cannot be negative")
```

---

9. Custom Exception Class

```
class MyError(Exception):
    pass

try:
    raise MyError("Custom error occurred!")
except MyError as e:
    print(e)
```

DIFFERENCE BETWEEN ERROR & EXCEPTION

Error	Exception
Caused by wrong logic or syntax	Caused during runtime
Cannot be handled	Can be handled using try-except
Stops program	Handled safely

REAL-LIFE EXAMPLES

1. File not found

```
try:
    f = open("abc.txt")
except FileNotFoundError:
    print("File missing!")
```

---

2. Database connection

```
try:
    connect_db()
except ConnectionError:
    print("Cannot connect to database!")
```

---

3. User input error

```
try:
    age = int(input("Enter age: "))
except ValueError:
    print("Please enter a number!")
```

SHORT SUMMARY

✓ File Handling

- `open()` → open file
- `read()`, `write()`, `append()`
- `with open()` recommended
- Binary file handling
- Cursor control
- Remove files using `os.remove()`

✓ Exception Handling

- Errors handled using `try-except`
- `else` runs when no error
- `finally` always runs
- Raise your own exception
- Create custom exceptions