Python Fundamentals (Part5)

Concepts : File I/O, Exception Handling, List Comprehensions, JSON module

File I/O

File I/O (Input/Output) means **reading data from files** and **writing data to files** using Python.
Python provides built-in functions to work with files.

Opening a File

We use the `open()` function:

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

Example

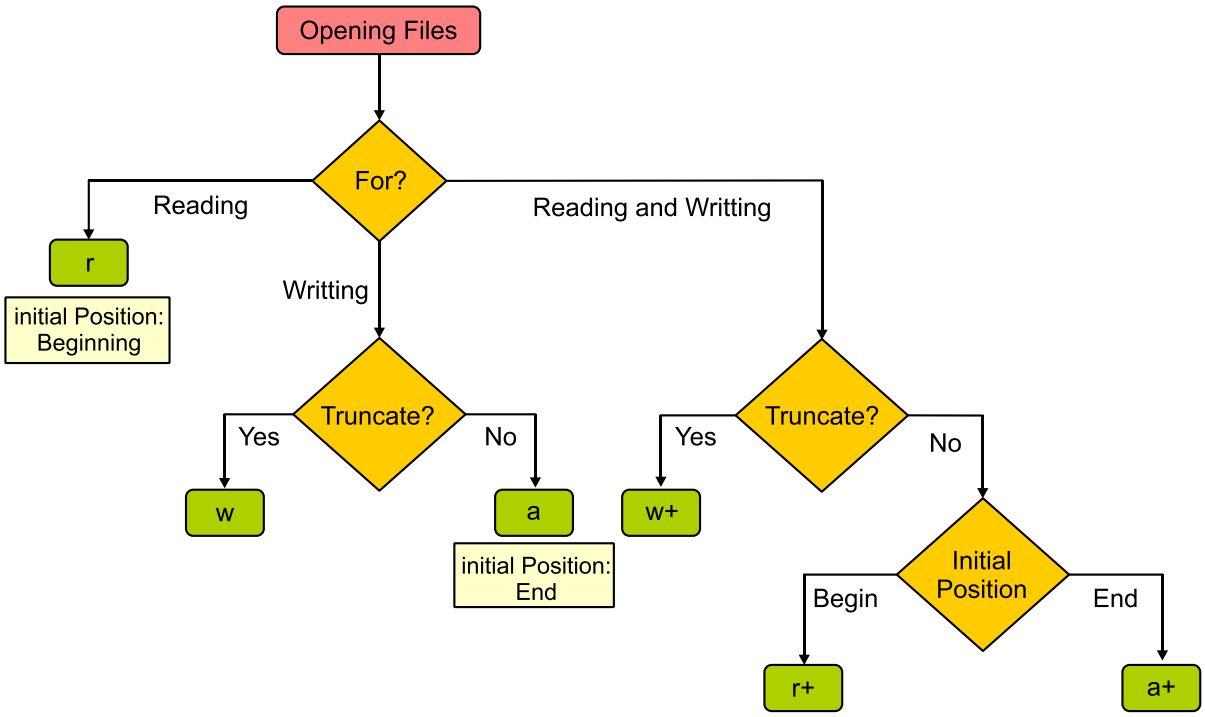
```
f = open("data.txt", "r")
```

Common Modes

Mode	Meaning	Description
`r`	Read	Opens file for reading (default). Error if file doesn't exist.
`w`	Write	Creates new file or overwrites existing file.
`a`	Append	Adds content at the end of the file.
`r+`	Read + Write	File must exist.
`w+`	Write + Read	Overwrites existing file.
`a+`	Append + Read	Reads + adds to end of file.
`b`	Binary Mode	For images, videos, etc. (use with other modes like `rb`).

What to use when?

Follow this flowchart:



Reading from a File

We have multiple functions to read content from a file.

1. `read()`

   Reads entire file as a single string.

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

2. `readline()`

   Reads one line at a time.

```
f = open("data.txt", "r")
line1 = f.readline()
line2 = f.readline()
f. close()
```

3. `readlines()`

Reads all lines into a list.

```
f = open("data.txt", "r")
lines = f.readlines()
```

Writing to a File

1. `write()`

Writes a string to a file.

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

2. `writelines()`

Writes multiple lines at once.

```
f = open("data.txt", "a")
f.writelines(["Line 1\n", "Line 2\n"])
f.close()
```

Always remember to close a file at the end to free system resources.

Recommended: Usage `with open()`

We use a context manager that automatically closes the file.

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

Deleting a File

To delete a file we use the `remove` function.

```
import os

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

Exception Handling

An **exception** is an error that occurs while a program is running. If not handled, the program crashes. Exception Handling allows you to manage errors gracefully so your program continues to run.

An exception occurs when Python encounters something it cannot handle during execution.

Examples:

- Dividing by zero - `ZeroDivisionError`
- Using an undefined variable - `NameError`
- Opening a missing file - `FileNotFoundError`
- Wrong data type - `TypeError`

Basic Syntax

```
try:
    # Code that may cause an error
except:
    # Code that runs if error occurs
```

Example:

```
try:
    x = 10 / 0
except:
    print("Error occurred!")
```

We can also throw specific exceptions:

```
try:
    print(10 / 0)
except ZeroDivisionError:
    print("You can't divide by zero!")
```

The `else` Block

`else` executes only if no exception happens.

```
try:
    x = int(input("Enter a number: "))
except ValueError:
    print("Invalid input!")
else:
    print("You entered:", x)
```

The `finally` Block

`finally` always executes, whether an exception occurs or not. It is used for cleanup tasks (closing files, releasing resources).

```
try:
    f = open("data.txt")
    print(f.read())
except FileNotFoundError:
    print ("File not found!")
finally:
    print("Execution completed.")
```

List Comprehensions

List Comprehension is a **short and elegant way** to create lists in Python.
It replaces long `for` loops with **one-line expressions**.

Basic Syntax

```
[expression for item in iterable]
```

Example: Create a list of numbers 1 to 5

```
nums = [x for x in range(1, 6)]
print(nums)    # [1, 2, 3, 4, 5]
```

List Comprehension with Condition

```
[expression for item in iterable if condition]
```

Example: Even numbers from 1 to 10

```
evens = [x for x in range(1, 11) if x % 2 == 0]
```

List Comprehension with if-else

```
[expression_if_true if condition else expression_if_false for item in iterable]
```

Example: Label numbers as “Even” or “Odd”

```
labels = ["Even" if x % 2 == 0 else "Odd" for x in range(1, 6)]
```

Working with JSON Module

JSON (**J**ava**S**cript **O**bject **N**otation) is a lightweight data format used to exchange data between programs, APIs, websites, etc. JSON format is very similar to Python dictionaries.

Python’s `json` module allows you to read, write, encode, and decode JSON.

Importing the Module

```
import json
```

Converting Python to JSON

We use `dumps()` to convert Python objects into JSON string.

```
data = {
    "name": "John",
    "age": 25,
    "marks": [85, 90, 92]
}

json_string = json.dumps(data)
print(json_string)
```

Converting JSON to Python

We use `loads()` to convert JSON string to Python dictionary.

```
json_data = '{"name": "John", "age": 25}'
python_obj = json.loads(json_data)
print(python_obj["name"])
```

Reading JSON from a File (`json.load`)

```
import json

with open("data.json", "r") as f:
    data = json.load(f)

print(data)
```

Writing JSON to a File (`json.dump`)

```
import json

data = {"name": "Aisha", "city": "Delhi"}

with open("data.json", "w") as f:
    json.dump(data, f, indent=4)
```

`indent=4` makes the JSON looks neat and readable.

> *Keep Learning & Keep Exploring!*