File & Exception Handling

Friday, 19 September, 2025 8:51 PM

Text Files :

Information is stored in ASCII or UNICODE characters.
EOL - End Of Line
The EOL error signifies that the Interpreter of Python reached the end of the line while scanning the string literal.
String literals (aka constants) must be enclosed in single or double quotation marks.

Binary Files :

Fast and easier for a program to read and write than text files (as it is stored in machine readable language).
Data cannot be directly read by humans, can be only read through Python program.

Opening & Closing Files :

To open a file, use open()

Direct assignment,

```
f=open("text file.txt", "mode")
```

With block (Automatic file closure),

```
with open("text file.txt", "mode") as f:
    #code
```

Note : f is the variable name assigned to the file object returned by open() (Can be named anything except for Python keywords).

"mode"

Text Files	Binary Files	Process	Description	Example for Text File	Example for Binary
r	rb	Read Only Mode	When you open a file in Read mode and the file doesn't exist, it raises an "FileNotFoundError" (I/O Error). By default, a file is opened in Read Mode.	f=open("file.txt") #default opens in 'r' mode	f=open("file.txt", "rb")
w	wb	Write Only Mode	When you open a file in Write mode and the file doesn't exist, it creates a file. When you use Write mode, it deletes all the existing data in the file and overwrites with the new data.	f=open("file.txt", "w")	f=open("file.txt", "wb")
a	ab	Append	When you open a file in Append mode and the file doesn't exist, it creates a file. When you use Append mode, it writes the new data at the end of the file with all the existing data in the file.	f=open("file.txt", "a")	f=open("file.txt", "ab")
r+	rb+	Read & Write	Opens the File for Both Reading & Writing. When file doesn't exist, "FileNotFoundError" (I/O Error) is raised.	f=open("file.txt", "r+")	f=open("file.txt", "rb+")
w+	wb+	Write & Read	Opens file for Both Reading & Writing. If the file doesn't exist, file is created. Writes the new data on top of the old data (Old existing data is removed).	f=open("file.txt", "w+")	f=open("file.txt", "wb+")
a+	ab+	Append & Read	Can read and write in the file. If the file doesn't exist, file is created. New data is written at the end of the file with the old existing data.	f=open("file.txt", "a+")	f=open("file.txt", "ab+")
x	-	Exclusive Creation Mode	Open the file only for writing, but only if the file doesn't exist already. If file exists, "FileExistsError" (I/O Error) is raised.	f=open("file.txt", "x")	-

To close a file you use f.close() where f is the file name you open with.

File Processing :

Counting lines in a file,

```
file=open("file.txt")  #File opens in 'r' mode by default
count = 0
for line in file:
    count+=1
print("Line Count : ", count)
```

Searching Through a file,

Using for Loop,

```
file=open("file.txt")
for line in file:
    if line.startswith('I'):
    print(line)
```

We can use .strip() function to remove whitespaces

Method	Description
x.strip()	Removes Whitespace from both sides of the string.
x.rstrip()	Removes Whitespace from the right side of the string.
x.lstrip()	Removes Whitespace from the left side of the string.

Note : Where 'x' is the string variable.

Reading a File :

Method	Description
f.read()	Reads the entire file in a string.
f.readline()	Reads one line at a time as a string. Each call moves to the next line.
f.readlines()	Reads the entire file as a list of strings. Where each string is 1 line.

Note : Where 'f' is the file object returned by open()

Writing into a Text File,

Method	Description	Example
f.write()	Writes a string into the file (new line is added manually using ' + "\n"').	x="this is a string" f.write(x)
f.writelines()	Writes a list of strings into the file. Each string is added as-is (New line is added only if it is included in the string itself).	x=["this", "is", "a", "list"] f.writelines(x)

Note : Where 'f' is the file object returned by open()

Exception Handling :

Syntax Errors,

Errors caused by not following proper syntax of the language.
These errors are detected before the program runs.
Eg:

```
def main()                    #Missing colon :
    a=10
    b=3
    print("Sum is ", a+b     #Missing )
```

Runtime Errors (Exceptions),

Exceptions what occur during the program execution/runtime.
Caused by some illegal operation taking place (operations which raises errors).
These errors mostly occur when the syntax is correct but the variables isn't assigned or the operation performed is incorrect .
Eg.i : file=open("file.txt", "r") #Raises "FileNotFoundError" as no file exists with that name
Eg.ii : a=5/0 #Raises "ZeroDivisionError" when you are trying to divide an expressio n by 0

Logical Errors,

Most annoying ones as the program runs without crashing but produces the incorrect results.
Caused by a mistake in the program's logic.
It won't generate the error message as no syntax or runtime error has occurred.

Eg:

Expression	Error Raised	Explanation
10 * (5 / 0)	ZeroDivisionError	Division by zero is not allowed.
10 * x / 5	NameError	Variable 'x' is not defined.
"2" + 10	TypeError	Can only concatenate str (not int) to str.
int("abc")	ValueError	Invalid Data type. Eg: Entering a str datatype 'abc' for int(input()) when it requires an integer input.
If x<3: x+=1	IndentationError	Improper indentation causes an error.
f=open("file.txt")	IOError	Raises when I/O operation (like reading or writing) fails.
(Reading from EOF)	EOFError	Raises an error when the cursor is at the end of the file, which can be fixed by using `f.seek(0)` before reading the file.
f=open("file.txt")	FileNotFoundError	Raises an error when the file does not exist.

Exception,

Even if an expression is syntactically correct, it may raise an error when you attempt to execute it.
Errors detected during execution are called exceptions.
An exception is an event that occurs during execution and disrupts the normal flow of the program.
If an exception is not handled, it will terminate the program and display an error message.

Try Block,

Let's you test a block of code for errors.

Except Block,

Let's you handle the error.

Finally Block,

Let's you execute the code, regardless of the result in try-except blocks.

Eg:

```
try:
  print(x)
except:
  print("An exception has occurred.")
finally:
  print("Done.")
```

If the code in try works, except is skipped.
If the try block raises an error, the except block will be executed.
A try statement can have multiple exceptions to handle different exceptions.
The finally block is run if any of the block runs.
Without the try block the program will crash and raise an error.

Logging Exceptions,

A method that keeps track of events during software execution.
It is important for software development, debugging, and maintenance.
To log an exception, we can use the logging module.
The logging module provides a set of functions for simple logging and follow-ups such as
debug(), info(), warning(), error(), critical()

To log an exception in Python with an error, use logging.exception() method.
This function logs a message with level ERROR on this logger.
The arguments are interpreted as for debug().
Exception information is added to the logging message.
This method should only be called from an except block.

Eg:

```
import logging
try:
   printf("Good Morning")
except Exception as Argument:
   logging.exception("Error occurred while printing Good Morning")
```