**File Handling in Python**

File handling in Python involves interacting with files on your computer to read data from them or write data to them. Python provides several built-in functions and methods for creating, opening, reading, writing, and closing files. This tutorial covers the basics of file handling in Python with examples.

**The syntax for opening a file in Python is −**

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

Where, **filename** is the name of the file to open and **mode** is the mode in which the file is opened (e.g., 'r' for reading, 'w' for writing, 'a' for appending).

**File Access Modes**

| **Mode** | **Description** |
| --- | --- |
| r | It opens an existing file to read-only mode. The file pointer exists at the beginning. |
| rb | It opens the file to read-only in binary format. The file pointer exists at the beginning. |
| r+ | It opens the file to read and write both. The file pointer exists at the beginning. |
| rb+ | It opens the file to read and write both in binary format. The file pointer exists at the beginning of the file. |
| w | It opens the file to write only. It overwrites the file if previously exists or creates a new one if no file exists with the same name. |
| wb | It opens the file to write only in binary format. It overwrites the file if it exists previously or creates a new one if no file exists. |
| w+ | It opens the file to write and read data. It will override existing data. |
| wb+ | It opens the file to write and read both in binary format |
| a | It opens the file in the append mode. It will not override existing data. It creates a new file if no file exists with the same name. |
| ab | It opens the file in the append mode in binary format. |
| a+ | It opens a file to append and read both. |
| ab+ | It opens a file to append and read both in binary format. |

The following table shows different access modes we can use while opening a file in Python.

# Opening the file with absolute path

fp = open(r'E:\demos\files\sample.txt', 'r')

# read file

print(fp.read())

# Closing the file after reading

fp.close()

# path if you using MacOs

# fp = open(r"/Users/myfiles/sample.txt", "r")

-=============

# Opening a file in read mode

file = open("example.txt", "r")

# Opening a file in write mode

file = open("example.txt", "w")

# Opening a file in append mode

file = open("example.txt", "a")

# Opening a file in binary read mode

file = open("example.txt", "rb")

Reading a File in Python

* **read() −** Reads the entire file.
* **readline() −** Reads one line at a time.
* **readlines −** Reads all lines into a list.

**Using "with" Statement for Automatic File Closing**

The with statement is a best practice in Python for file operations because it ensures that the file is automatically closed when the block of code is exited, even if an exception occurs.

with open("example.txt", "r") as file:

content = file.read()

print(content)

--

with open("example.txt", "r") as file:

lines = file.readlines()

for line in lines:

print(line, end='')

**Write multiple lines**

lines = ["First line\n", "Second line\n", "Third line\n"]

with open("example.txt", "w") as file:

file.writelines(lines)

print ("Content added Successfully!!")

**Handling Exceptions When Closing a File**

try:

file = open("example.txt", "w")

file.write("This is an example with exception handling.")

finally:

file.close()

print ("File closed successfully!!")

### **Common Python Errors and Exceptions**

1. **SyntaxError**
   * **Description**: Raised when the Python parser encounters a syntax error.
   * **Example**: Missing a colon, unmatched parentheses.
   * **Example Code**:

print("Hello, world"  # SyntaxError: unexpected EOF while parsing

1. **IndentationError**
   * **Description**: Raised when there are issues with the indentation of the code.
   * **Example**: Mixing tabs and spaces, incorrect indentation levels.
   * **Example Code**:

def greet():  
print("Hello, world!")  # IndentationError: expected an indented block

1. **TypeError**
   * **Description**: Raised when an operation or function is applied to an object of inappropriate type.
   * **Example**: Adding a string to an integer.
   * **Example Code**:

result = "Age: " + 30  # TypeError: can only concatenate str (not "int") to str

1. **ValueError**
   * **Description**: Raised when a function receives an argument of the right type but inappropriate value.
   * **Example**: Passing a non-numeric string to a function expecting a number.
   * **Example Code**:

number = int("not\_a\_number")  # ValueError: invalid literal for int() with base 10: 'not\_a\_number'

1. **IndexError**
   * **Description**: Raised when a sequence subscript is out of range.
   * **Example**: Accessing an element outside the bounds of a list.
   * **Example Code**:

my\_list = [1, 2, 3]  
item = my\_list[5]  # IndexError: list index out of range

1. **KeyError**
   * **Description**: Raised when a dictionary key is not found.
   * **Example**: Accessing a non-existent key in a dictionary.
   * **Example Code**:

my\_dict = {"name": "Alice"}  
value = my\_dict["age"]  # KeyError: 'age'

1. **FileNotFoundError**
   * **Description**: Raised when a file operation (e.g., open) fails because the file does not exist.
   * **Example**: Trying to open a file that doesn’t exist.
   * **Example Code**:

with open('non\_existent\_file.txt', 'r') as file:  # FileNotFoundError: [Errno 2] No such file or directory: 'non\_existent\_file.txt'  
    content = file.read()

1. **ZeroDivisionError**
   * **Description**: Raised when dividing by zero.
   * **Example**: Performing division with zero as the divisor.
   * **Example Code**:

result = 10 / 0  # ZeroDivisionError: division by zero

1. **AttributeError**
   * **Description**: Raised when an attribute reference or assignment fails.
   * **Example**: Accessing or modifying an attribute that doesn’t exist.
   * **Example Code**:

my\_str = "hello"  
my\_str.append(" world")  # AttributeError: 'str' object has no attribute 'append'

1. **ImportError**
   * **Description**: Raised when an imported module cannot be found.
   * **Example**: Trying to import a non-existent module.
   * **Example Code**:

import non\_existent\_module  # ImportError: No module named 'non\_existent\_module'

### **Handling Exceptions**

You can handle exceptions using try and except blocks to prevent your program from crashing and to manage errors gracefully.

**Example of Handling Exceptions:**

try:  
    # Code that might raise an exception  
    result = 10 / 0  
except ZeroDivisionError:  
    # Handle the exception  
    print("You can't divide by zero!")  
except FileNotFoundError:  
    print("File not found!")  
except Exception as e:  
    # Handle any other exception  
    print(f"An error occurred: {e}")  
else:  
    # Code to execute if no exceptions were raised  
    print("Operation successful!")  
finally:  
    # Code to execute regardless of whether an exception was raised or not  
    print("Execution completed.")

### **Creating Custom Exceptions**

You can also define your own exceptions by inheriting from the base Exception class.

**Example of a Custom Exception:**

class CustomError(Exception):  
    def \_\_init\_\_(self, message):  
        super().\_\_init\_\_(message)  
  
try:  
    raise CustomError("This is a custom error message.")  
except CustomError as e:  
    print(f"CustomError caught: {e}")

### **Summary**

* **SyntaxError**: Issues with code syntax.
* **IndentationError**: Issues with code indentation.
* **TypeError**: Incorrect data type usage.
* **ValueError**: Correct type but inappropriate value.
* **IndexError**: Out-of-range index.
* **KeyError**: Missing dictionary key.
* **FileNotFoundError**: File not found.
* **ZeroDivisionError**: Division by zero.
* **AttributeError**: Accessing non-existent attributes.
* **ImportError**: Module import issues.

What is Exception?

An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions. In general, when a Python script encounters a situation that it cannot cope with, it raises an exception. An exception is a Python object that represents an error.

When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.

Handling an Exception in Python

If you have some suspicious code that may raise an exception, you can defend your program by placing the suspicious code in a **try**: block. After the **try**: block, include an **except**: statement, followed by a block of code which handles the problem as elegantly as possible.

* The **try**: block contains statements which are susceptible for exception
* If exception occurs, the program jumps to the **except**: block.
* If no exception in the **try**: block, the **except**: block is skipped.

Syntax

Here is the simple syntax of **try...except...else** blocks −

try:

You do your operations here

......................

except ExceptionI:

If there is ExceptionI, then execute this block.

except ExceptionII:

If there is ExceptionII, then execute this block.

......................

<https://www.tutorialspoint.com/python/python_file_handling.htm>

<https://pynative.com/python/file-handling/>

[**os.chdir(path)**](https://www.tutorialspoint.com/python/os_chdir.htm)

Change the current working directory to path

[**os.chmod(path, mode)**](https://www.tutorialspoint.com/python/os_chmod.htm)

Change the mode of path to the numeric mode.

[**os.getcwd()**](https://www.tutorialspoint.com/python/os_getcwd.htm)

Return a string representing the current working directory

[**os.listdir(path)**](https://www.tutorialspoint.com/python/os_listdir.htm)

Return a list containing the names of the entries in the directory given by path.

[**os.makedirs(path[, mode])**](https://www.tutorialspoint.com/python/os_makedirs.htm)

Recursive directory creation function.

[**os.mkdir(path[, mode])**](https://www.tutorialspoint.com/python/os_mkdir.htm)

Create a directory named path with numeric mode mode.

[**os.remove(path)**](https://www.tutorialspoint.com/python/os_remove.htm)

Remove the file path.

[**os.removedirs(path)**](https://www.tutorialspoint.com/python/os_removedirs.htm)

Remove directories recursively.

[**os.rename(src, dst)**](https://www.tutorialspoint.com/python/os_rename.htm)

Rename the file or directory src to dst.

[**os.path.basename(path)**](https://www.tutorialspoint.com/python/os_path_basename.htm)

Returns the base name of pathname path.

[**os.path.dirname(path)**](https://www.tutorialspoint.com/python/os_path_dirname.htm)

Returns the directory name of pathname path.

[**os.path.exists(path)**](https://www.tutorialspoint.com/python/os_path_exists.htm)

Returns True if path refers to an existing path. Returns False for broken symbolic links.

[**os.path.join(path1[, path2[, ...]])**](https://www.tutorialspoint.com/python/os_path_join.htm)

Joins one or more path components intelligently.

[**os.path.split(path)**](https://www.tutorialspoint.com/python/os_path_split.htm)

Splits the pathname path into a pair, (head, tail) where tail is the last pathname component and head is everything leading up to that.

Pip install pickle

Regex

Beautifulsoup

Request

Numpy

Pandas

Matplotlib

Seaborn

Sklearn