File Handling in Python

- Python too supports file handling and allows users to handle files i.e., to read and write files, along with many other file handling options, to operate on files.
- The concept of file handling has stretched over various other languages, but the implementation is either complicated or lengthy, but alike other concepts of Python, this concept here is also easy and short.
- Python treats file differently as text or binary and this is important. Each line of code includes a sequence of characters and they form text file.
- Each line of a file is terminated with a special character, called the EOL or End of Line characters like comma {,} or newline character.
- It ends the current line and tells the interpreter a new one has begun. Let's start with Reading and Writing files.

Working of open() function

- We use open () function in Python to open a file in read or write mode.
- As explained above, open ( ) will return a file object.
- To return a file object we use open() function along with two arguments, that accepts file name and the mode, whether to read or write. So, the syntax being: open(filename, mode).
- There are three kinds of mode, that Python provides and how files can be opened:
    - " r ", for reading.
    - " w ", for writing.
    - " a ", for appending.
    - " r+ ", for both reading and writing

In [ ]:

```
f = open("test.txt", 'r') # open file in current directory
f
```

In [ ]:

```
f = open('Test.txt')
f
```

In [ ]:

```
f1 = open("C:/Test.txt")  # specifying full path
```

In [ ]:

```
f1
```

In [ ]:

```
import os
os.getcwd()
```

In [ ]:

```
# f = open("test.txt")      # equivalent to 'r' or 'rt'
f = open("testnow.txt",'w')  # write in text mode
```

In [ ]:

```
f = open('Test2.txt', 'w')
```

In [ ]:

```
# Closing the file
f.close()
```

In [ ]:

```
#Exceptions
try:
    f = open("testnow.txt",'w')
    f.write('This is my first line\n')
    f.write('This is my second line\n')
finally:
    f.close()
```

In [ ]:

```
# Using With Method
```

```
with open("test.txt") as f:
    print(f.readlines())
```

In [ ]:

```
# Writing contents inside the file

with open("testnew.txt",'w') as f:
    f.write("my first file\n")
    f.write("This file\n")
    f.write("contains three lines\n")
```

In [ ]:

```
f = open('testnew.txt')
f.readlines()
```

In [ ]:

```
f = open("Test.txt",'r')
f.read(4)   # read the first 4 data
```

In [ ]:

```
f.read(10)
```

In [ ]:

```
f = open("test.txt",'r')

x = f.readlines()
x
```

In [ ]:

```
f.read(4)    # read the next 4 data
```

In [ ]:

```
f.read(10)     # read in the rest till end of file
```

In [ ]:

```
f.read()  # further reading returns empty sting
```

In [ ]:

```
f.tell()   # get the current file position
```

In [ ]:

```
f.seek(0)   # bring file cursor to initial position
```

In [ ]:

```
f.read(5)
```

In [ ]:

```
print(f.read())  # read the entire file
```

In [ ]:

```
for line in f:
    print(line, end = '')
```

In [ ]:

```
f.readline()
```

In [ ]:

```
f.readline()
```

In [ ]:

```
f.readline()
```

In [ ]:

```
f.readline()
```

In [ ]:

```
f.readlines()
```

In [ ]:

```
f = open("myfile1.txt", "x")
```

In [ ]:

```
f.write("my first file\n")
f.write("This file\n")
f.write("contains three lines\n")
```

In [ ]:

```
f=open('myfile1.txt','r')
```

In [ ]:

```
f.readlines()
```

In [ ]:

```
import os
os.chdir('..')
os.remove("myfile1.txt")
```

In [ ]:

```
import os
if os.path.exists("myfile.txt"):
    os.remove("myfile.txt")
else:
    print("The file does not exist")
```

In [ ]:

```
f = open('Test.txt')
```

In [ ]:

```
f
```

In [ ]:

```
f = open('Test.txt','a+')
```

Exceptions Handling

- Like other languages, python also provides the runtime errors via exception handling method with the help of try-except.
- Some of the standard exceptions which are most frequent include IndexError, ImportError, IOError, ZeroDivisionError, TypeError.

Try and Except

- A try statement can have more than one except clause, to specify handlers for different exceptions.
- Please note that at most one handler will be executed.

Else Clause:

- In python, you can also use else clause on try-except block which must be present after all the except clauses.
- The code enters the else block only if the try clause does not raise an exception.

Raising Exception:

- The raise statement allows the programmer to force a specific exception to occur.
- The sole argument in raise indicates the exception to be raised. This must be either an exception instance or an exception class (a class that derives from Exception).

try...finally

- The try statement in Python can have an optional finally clause. This clause is executed no matter what, and is generally used to release external resources.

- For example, we may be connected to a remote data center through the network or working with a file or working with a Graphical User Interface (GUI).

- In all these circumstances, we must clean up the resource once used, whether it was successful or not.

- These actions (closing a file, GUI or disconnecting from network) are performed in the finally clause to guarantee execution.

In [ ]:

```
if a < 3
```

In [ ]:

```
a = 10
```

In [ ]:

```
1 / 0
```

In [ ]:

```
open("imaginary.txt")
```

In [ ]:

```
try:
    print(1/0)
except NameError:
    print("Variable x is not defined")
except:
    print("Something else went wrong")
```

In [ ]:

```
X = 'New'
x = 'new'
```

In [ ]:

```
try:
  print(X*x)
except:
  print("Something went wrong")
else:
  print("Nothing went wrong")
```

In [ ]:

```
try:
  print(Y)
except:
  print("Something went wrong")
finally:
  print("The 'try except' is finished")
```

In [ ]:

```
a = [1, 2, 3]
try:
    print ("Second element = %d" %(a[1]))
    print ("Fourth element = %d" %(a[3]))
except IndexError:
    print ("An error occurred")
```

In [ ]:

```
# to handle multiple errors with one except statement
try :
    a = 3
    if a < 4 :
        b = a/(a-3)  # throws ZeroDivisionError for a = 3
    print ("Value of b = ", b)  # throws NameError if a >= 4

# note that braces () are necessary here for multiple exceptions
except(ZeroDivisionError, NameError):
    print ("Error Occurred and Handled")
```

In [ ]:

```
# to depict else clause with try-except

# Function which returns a/b
def AbyB(a , b):
    try:
        c = ((a+b) / (a-b))
    except ZeroDivisionError:
        print ("a/b result in 0")
    else:
        print(c)

# Driver program to test above function
AbyB(2, 3)
AbyB(3, 3)
```

In [ ]:

```
try:
    f = open('missing')
except FileNotFoundError:
        print('File not Found')
except OSError:
        print('OS Failed')
```

In [ ]:

```
f = open('missing')
```

In [ ]:

```
try:
    a = 10/0
    print(a)
except ArithmeticError:
        print ("This statement is raising an arithmetic exception")
else:
    print ("Success")
```

In [ ]:

```
# import module sys to get the type of exception
import sys

randomList = ['a', 0, 2]

for entry in randomList:
    try:
        print("The entry is", entry)
        r = 1/int(entry)
        break
    except:
        print("Oops!",sys.exc_info()[0],"occured.")
        print("Next entry.")
        print()
print("The reciprocal of",entry,"is",r)
```

In [ ]:

```
try:
    a = int(input("Enter a positive integer: "))
    print(a)
    if a <= 0:
        raise ValueError("That is not a positive number!")
    else:
        print(a*25)
except ValueError as ve:
    print(ve)
```

In [ ]:

```
try:
    f = open("test.txt",'w', encoding = 'utf-8')
    f.write('gutyjbkj')
    f.write('gjhguytuytufuty')
   # perform file operations
finally:
    f.close()
```