HEXAVARSITY
Enhancing Excellence

Python Programming

Exceptions and File Handling

Session Objective

To understand the concepts of Exception handling and File operations in Python,

- Errors & Exceptions
  - Syntax Errors
  - Logical Errors
- Exceptions
  - Built-in Exception
  - Handling mechanism
  - User defined Exception
- File IO
  - File Modes
  - File Operation
  - File Methods

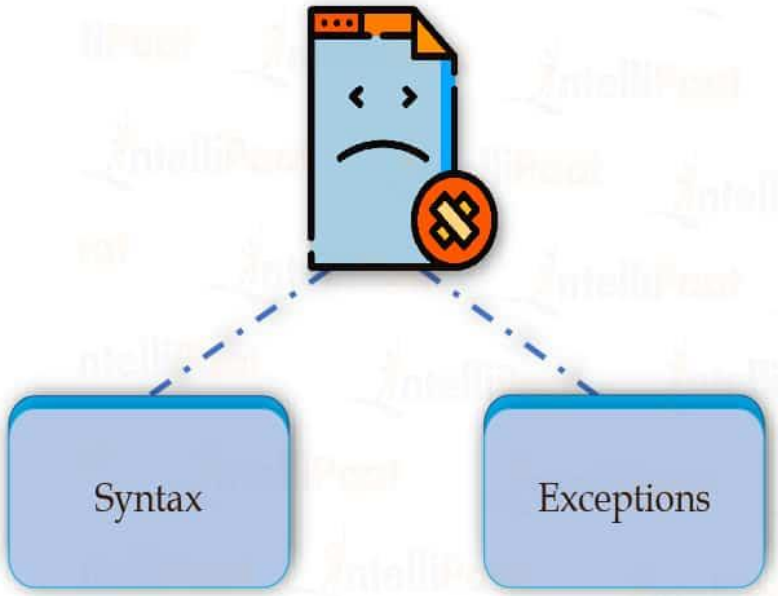
Exceptions

Errors and Exceptions

- Errors detected during execution are called *exceptions*
- A python program terminates as soon as it encounters an unhandled error.

These errors are 2 types:

1. Syntax errors
2. Logical errors (Exceptions)



Errors and Exceptions Contd...

Syntax Errors

- Syntax errors occur when the parser detects an incorrect statement.
- Error caused by not following the proper syntax of the language is called **syntax error** or **parsing error.**

```
>>> if a < 3

 File "<interactive input>", line 1
   if a < 3
        ^
SyntaxError: invalid syntax
```

Note: colon : is missing in the if statement.

Errors and Exceptions Contd...

Logical Errors - Exceptions

- Errors that occur at runtime are called exceptions or logical errors.
- Whenever these types of runtime errors occur, Python creates an exception object.
- If not handled properly, it prints a traceback to that error.

Example for Exceptions

- ✓ Open a file for reading that does not exist ： FileNotFoundError
- ✓ Divide a number by zero : ZeroDivisionError
- ✓ Import a module that does not exist ： ImportError

Errors and Exceptions

Contd...

Example for Exceptions:

```
>>> 2/0
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    2/0
ZeroDivisionError: division by zero

>>> '2'+4
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    '2'+4
TypeError: can only concatenate str (not "int") to str
```

Built-in Exceptions

Exceptions

Built-in Exceptions

- An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions.

- Illegal operations can raise exceptions.

Example:

i. **ZeroDivisionError:** Occurs when a number is divided by zero.

ii. **NameError:** It occurs when a name is not found. It may be local or global.

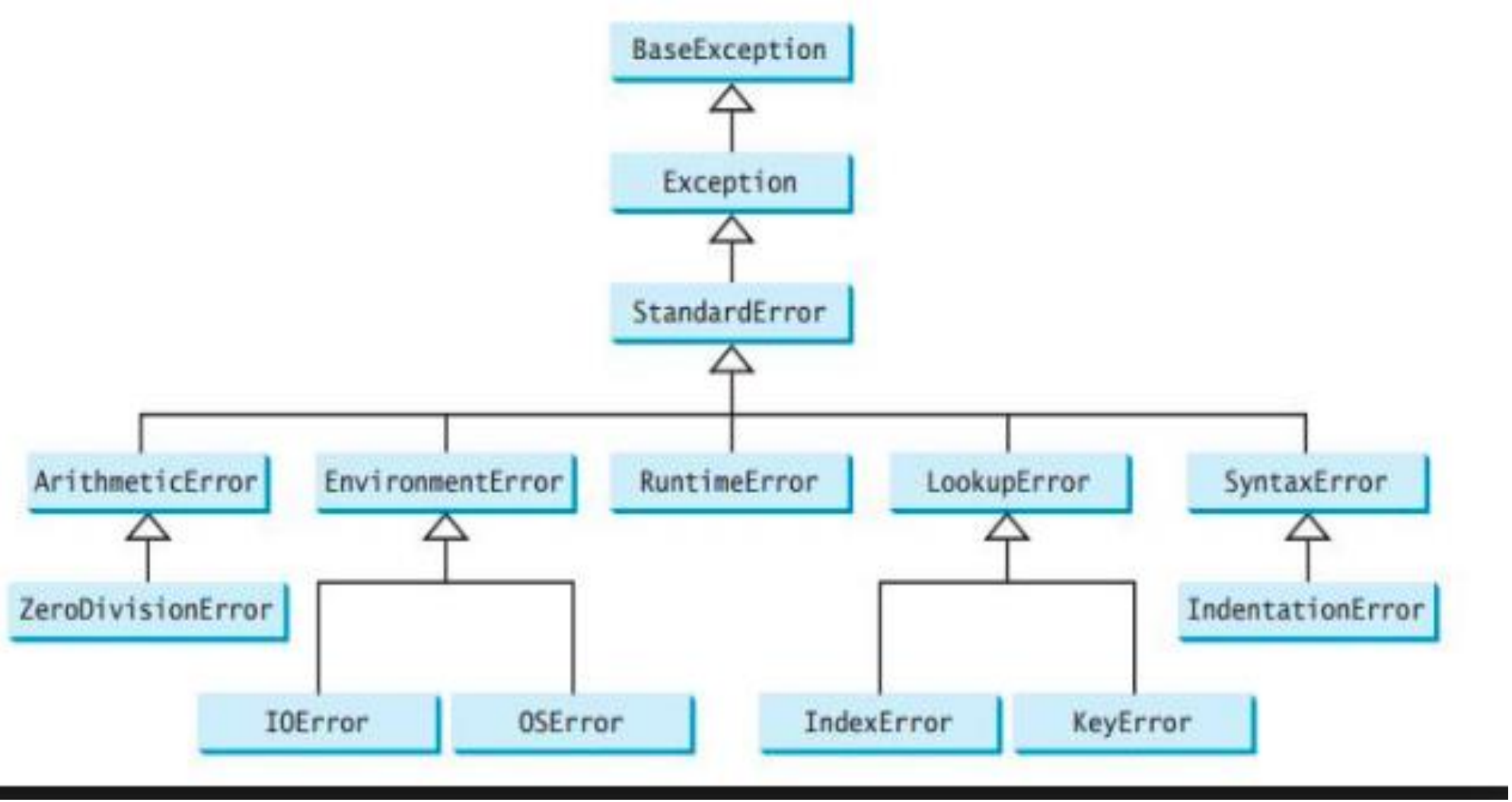
iii. **IndentationError:** If incorrect indentation is given.

iv. **IOError:** It occurs when Input Output operation fails.

v. **EOFError:** It occurs when the end of the file is reached, and yet operations are being performed.

Exceptions

Contd...



Exceptions Contd...

Exception Handling

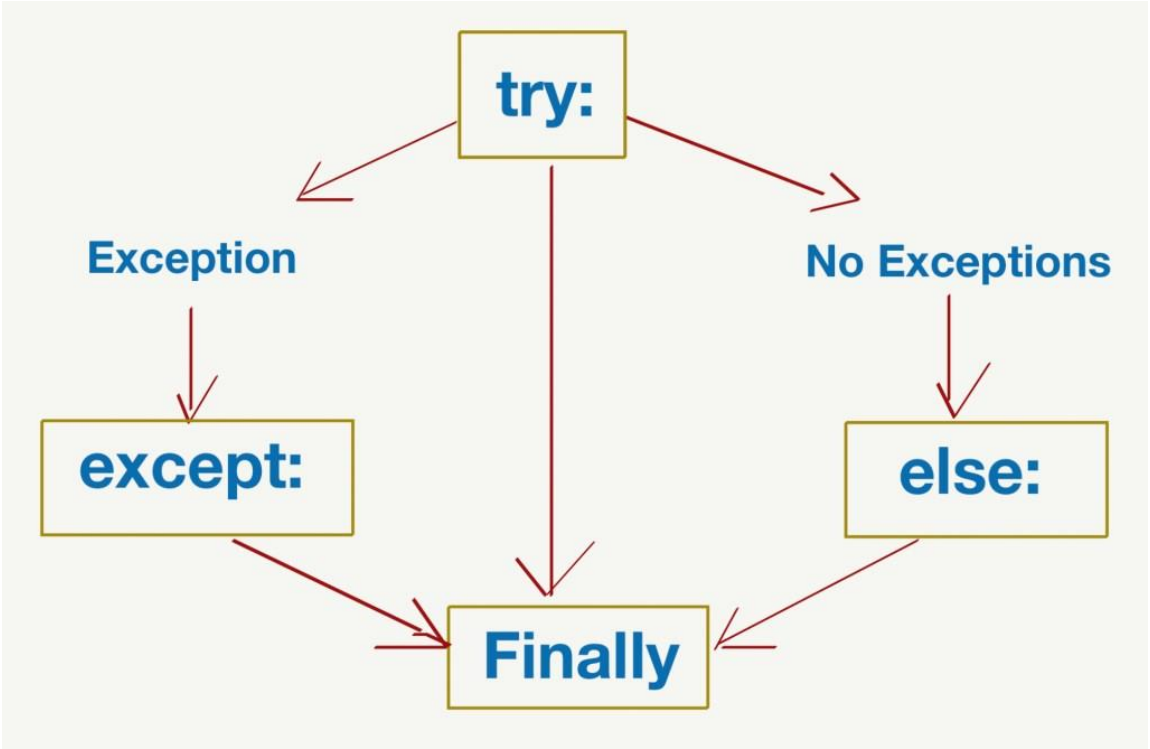
- When an exceptions occur, the Python interpreter stops the current process.
- Passes the unhandled exception to the calling process until it is handled.
- All statements are executed until an exception is encountered.

TRY

- Exceptions can be handled using a try statement.
- The critical operation which can raise an exception is placed inside the try clause.

Exception Handling

try:
{ Run this code
except:
{ Execute this code when there is an exception
else:
{ No exceptions? Run this code.
finally:
{ Always run this code.



Exceptions Contd...

EXCEPT

- The code that handles the exceptions is written in the except clause.
- Except is used to catch and handle the exceptions that are encountered in the try clause.
- Multiple exceptions Cn be declared in the except statement

ELSE

- Else lets the code sections to run only when no exceptions are encountered in the try clause.
- The statements that don't throw the exception should be placed inside the else block.

FINALLY

- Finally enables to execute sections of code that should always run, with or without any encountered exceptions.

Exceptions

Contd...

Exception Handling Example

```
projectid=[22,33,44]
 print(projectid[3])
```

```
Traceback (most recent call last):
File
"C:/Users/31410/PycharmProjects/Sa
mpleProject/Welcome.py", line 2, in
<module>
print(projectid[3])
IndexError: list index out of range
```

```
projectid=[22,33,44]
try:
   print(projectid[2])

except Exception:
   print("Exception is handled here.....")
```


Exceptions Contd...

Declaring Multiple Exceptions

- ✓ The Python allows to declare the multiple exceptions with the except clause.
- ✓ Declaring multiple exceptions is useful in the cases where a try block throws multiple exceptions.

```
try:
   #block of code

except (<Exception 1>,<Exception 2>,<Exception 3>,...<Exception n>)
   #block of code

else:
   #block of code
```


Exceptions

Contd...

Example for Exception Handling

```
projectid=[22,33,44]
projectinfo="Airlines"
try:
   print("Project id:",projectid[2])
except Exception as e:
   print("only elements from the list  ",e)
else:
   print("Project name",projectinfo)
finally:
   print("Always gets executed")
```

Exceptions

Contd...

Raising Exceptions

An exception can be raised by using the raise clause in Python.

raise Exception_class,<value>

```
try:
   age = int(input("Enter the age:"))
   if(age<18):
      raise ValueError
   else:
      print("the age is valid")
except ValueError:
   print("The age is not valid")
```


Exceptions Contd...

User defined Exception

- Custom exceptions can be derived by creating a new class.
- This exception class must be derived, either directly or indirectly, from the built-in Exception class.

```
class SalaryError(Exception):
   # Constructor or Initializer
   def __init__(self, salary):
      self.salary = salary
try:
   salary=3000
   if salary<10000:
      raise SalaryError(salary)

except SalaryError as error:
   print('A New Exception occured: ', error.salary)
```


File Input & Output

Files I/O

FILES

- Files are named locations on disk to store related information.
- They are used to permanently store data in a non-volatile memory e.g. hard disk.
- To read from or write to a file, open the file, manipulate it and close the file so that the resources that are tied with the file are freed.

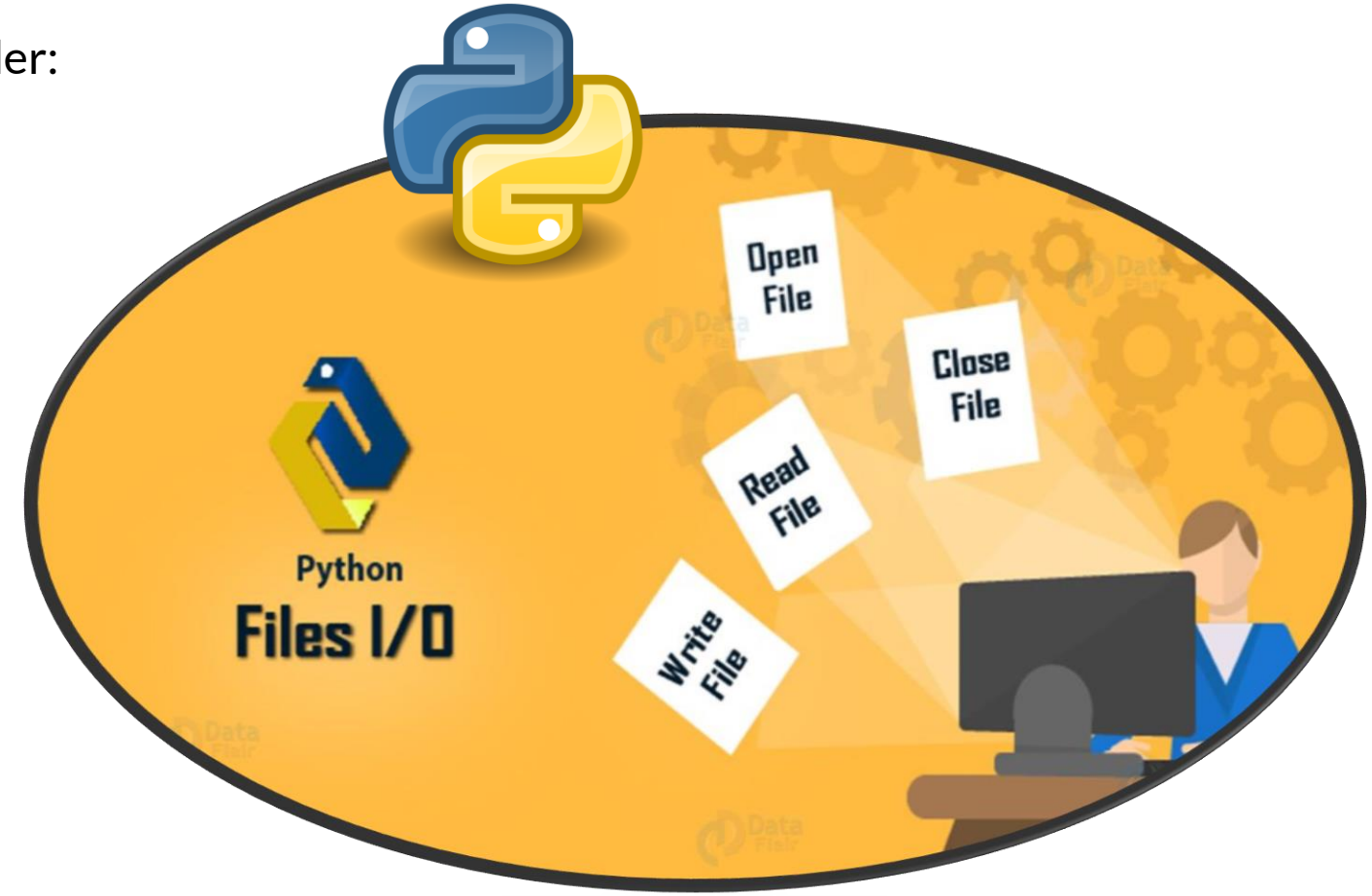
File Operation

A file operation takes place in the following order:

1.Open a file

2.Read or write – Manipulate

3.Close the file



Opening Files

- Python has a built-in open() function to open a file.
- This function returns a file object, also called a handle, as it is used to read or modify the file .

```
>>>f = open("test.txt")

>>> f = open("C:/Users/31410/Desktop/Python_Demo/test.txt")

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

open file in current directory

specifying full path

write in text mode

Files I/O

Contd...

- File mode can be specified while opening a file.
- The file can be opened in text mode or binary mode.
- The default is reading in text mode.
- Binary mode returns bytes, and this is the mode to be used when dealing with non-text files like images or executable fi
- Write mode creates a new file if it does not exist or truncates the file if it exists.

File Modes

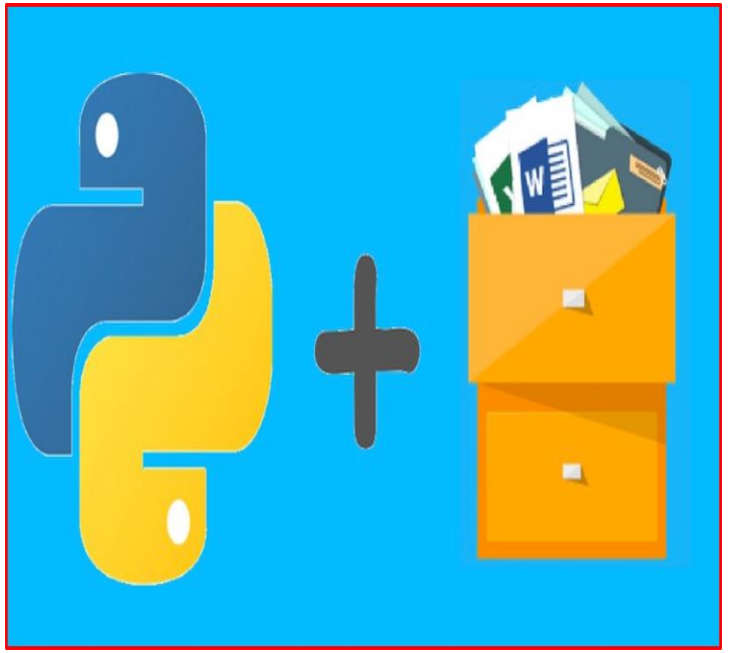
The different modes used for File Operation are:

Mode	Description
r	Opens a file for reading
w	Opens a file for writing
a	Opens a file for appending
t	Opens in text mode
b	Opens in binary mode.
+	Opens a file for updating - reading and writing

Files with Exception

- When an exception occurs while performing some operation with the file, the code exits without closing the file.
- A safer way is to use file operations is with the try...finally block.

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



Contd...

Writing Files

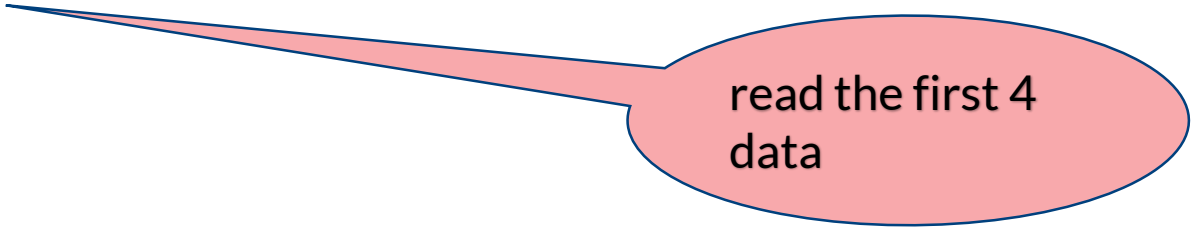
- To write into a file, open the file in write “w”, append “a” or exclusive creation “x” mode.
- In w mode, it will overwrite into the file if it already exists or creates a new file if the file does not exist.
- Writing a string or sequence of bytes for binary files is done using the write() method. This method returns the number of characters written to the file.

```
with open("test.txt",'w',encoding = 'utf-8') as f:
 f.write("my first file\n")
```

Reading Files

- To read a file in Python, open the file in reading “r” mode.
- To read the number of data, use read(size) method.
- If the size parameter is not specified, it reads and returns up to the end of the file.
- The read() method returns a newline as '\n'.

```
f = open("test.txt",'r',encoding = 'utf-8')
f.read(4)
```



Closing Files

- The Opened file instance can be closed using close() method

```
f.close()
```

- The with statement ensures that the file is closed when the block inside the with statement is exited.
- Explicit call to the close() method is not required. It is done internally.

```
with open("test.txt", encoding = 'utf-8') as f:
  # perform file operations
```


File Methods

Methods	Description
flush()	Flushes the write buffer of the file stream.
readline(n=-1)	Reads and returns one line from the file.
readlines(n=-1)	Reads and returns a list of lines from the file.
seek(offset,from=SEEK_SET)	Changes the file position to offset bytes, in reference to from (start, current, end).
tell()	Returns the current file location.
writelines(lines)	Writes a list of lines to the file.
truncate(size=None)	Resizes the file stream to size bytes.

Think and Answer

1. How many except statements can a try-except block have?
2. When will the else part of try-except-else be executed?
3. When is the finally block executed?
4. Which method is used to read the entire contents of the file as a string from a file object “fb”?
5. Which of the following command is used to open a file “c:\temp.txt” for writing in binary format only?

Think and Answer

1. One or more
2. when no exception occurs
3. Always
4. fb.read()
5. open(“c:\\temp.txt”, “wb”)

Thank you

Innovative Services

Passionate Employees

Delighted Customers