UNIT – IV

Exception and File Handling

Exception Handling-Difference between an error and Exception, Handling Exception, try except block, Raising Exceptions, User Defined Exceptions.

Files Handling- Signaficance of files, types of files,file path,file modes,Understanding read functions:read(),readline() and readlines() Understanding write functions:write() and wriletlines(),manipulating file pointer using seek.

Learning Material

Error

Errors are the problems in a program due to which the program will stop the execution.

The two common types of errors are ***syntax errors*** and ***logic errors***.

syntax errors :

When the proper syntax of the language is not followed then a syntax error is thrown.

Example:

```
>>> i=0
>>> if i == 0 print(i)
```

SyntaxError: invalid syntax

logic errors:

Logical error specifies all those type of errors in which the program executes but gives incorrect results.Logical error may occur due to wrong algorithm or logic to solve a particular problem.

Example:

```
# initialize the amount variable
marks = 10000

# perform division with 0
a = marks / 0
print(a)
```

Logical error:ZeroDivisionError: division by zero

Exception:

An exception is an event,which occurs during the execution of a program and disrupts the normal flow of the program's instructions.When a program raises an exception,it must handle the exception or the program will be immediately terminated.

Example:

```
>>> 5/0
```

```
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    5/0
```

ZeroDivisionError: division by zero

```
>>>"Rollno"+123
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
"Rollno"+123
```

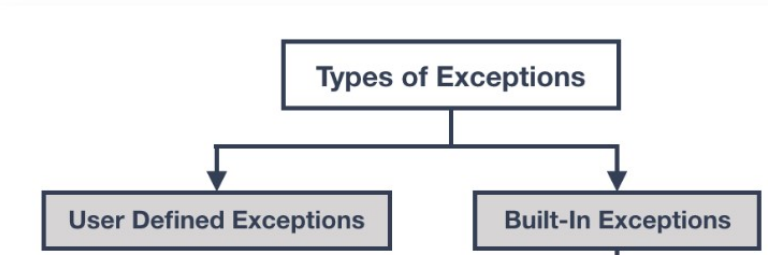
TypeError: Can't convert 'int' object to str implicitly

The string printed as the exception type(like TypeError) is the name of the built in exception that occur.

There are two types of Exceptions:

1.Built-in-Exceptions/Predefined Exceptions

2.User Defined Exceptions



4.1 Built-in-Exceptions/Predefined Exceptions:

Predefined Exceptions:

Also known as inbuilt exceptions. The exceptions which are raised automatically by Python virtual machine whenever a particular event occurs are called pre defined exceptions.

Eg 1: Whenever we are trying to perform Division by zero, automatically Python will raise ZeroDivisionError.

```
print(10/0)
```

Eg 2 : Whenever we are trying to convert input value to int type and if input value is not int value then Python will raise ValueError automatically.

```
x=int("ten")
```

ValueError

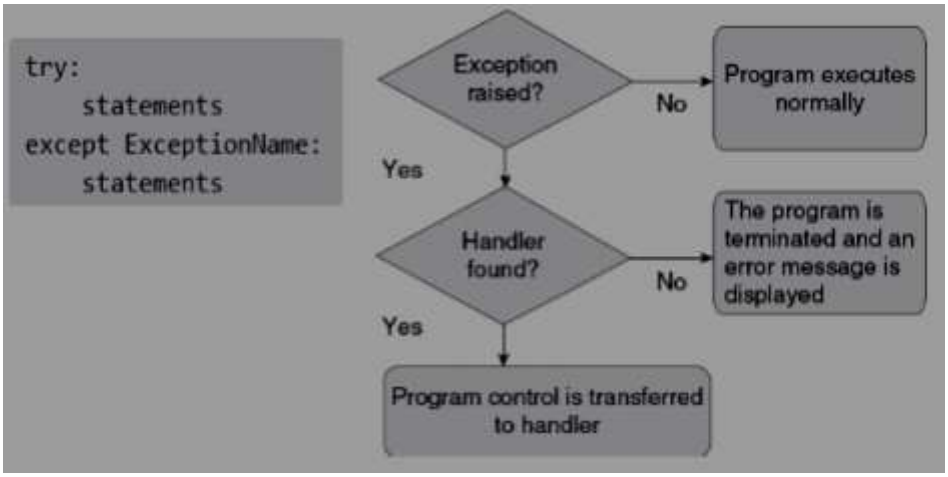
The following is the list of some of built-in exceptions.

Class	Description
Exception	A base class for most error types
AttributeError	Raised by syntax obj.foo, if obj has no member named foo
EOFError	Raised if "end of file" reached for console or file input
IOError	Raised upon failure of I/O operation (e.g., opening file)
IndexError	Raised if index to sequence is out of bounds
KeyError	Raised if nonexistent key requested for set or dictionary
KeyboardInterrupt	Raised if user types ctrl-C while program is executing
NameError	Raised if nonexistent identifier used
StopIteration	Raised by next(iterator) if no element; see Section 1.8
TypeError	Raised when wrong type of parameter is sent to a function
ValueError	Raised when parameter has invalid value (e.g., sqrt(−5))
ZeroDivisionError	Raised when any division operator used with 0 as divisor

4.2 Handling Exception

We can handle exceptions in our program by using try block and except block.

A critical operation which can raise exception is placed inside the try block and the code that handles exception is written in except block. The syntax for try–except block can be given as:



Example1:Python program to demonstrate the usage of NameError

```
#python program to handle NameError exception
try:
    print(x)
except:
print("X is not defined")
```

Output:

```
X is not defined
```

Example2 :Python program to demonstrate the usage of IndexError

```
#python program to handle IndexError exception

l=[1,2,3]
try:
    print(l[10])
except IndexError:
    print("You are not giving valid index")
```

output:-

```
"You are not giving valid index"
```

try except block:

The syntax for try except block :

```
try:
    Statements
except ExceptionName:
        Statements
```

- The try block is used to check some code for errors i.e the code inside the try block will execute when there is no error in the program.
- Whereas the code inside the except block will execute whenever the program encounters some error in the preceding try block.

Example : Python program to demonstrate the usage of zero division error.

```
#python program to handle zero division exception

num=int(input("Enter numerator:"))
coef=num/0
print("coef:",coef)
```

Output:

```
Enter numerator:3
Traceback (most recent call last):
  File "\\192.168.0.180\lib\python fop\1.py", line 2, in <module>
    coef=num/0
ZeroDivisionError: division by zero
```

Example : Python program to demonstrate the usage of zero division error.

```
#python program to handle zero division exception

num=int(input("Enter numerator:"))
denm=int(input("Enter denominator:"))
try:
        coef=num/denm
        print("The quotient is:",coef)
except ZeroDivisionError:
   print("The denominator cannot be zero")
```

Output:

```
>>> Enter numerator:5
Enter denominator:0
The denominator cannot be zero
```

Example : Python program to demonstrate the usage of zero division error.

```
#python program to handle zero division exception
a=int(input("Enter the first number"))
b=int(input("Enter the second number"))
try:
    c=a/b
    print("The division result is",c)
except ZeroDivisionError:
    print("see your denominator value is zero")
```

```
finally:
    print("I Sucessfully handle zero divion exception")
```

output:-

```
Enter the first number4
Enter the second number0
see your denominator value is zero
I Sucessfully handle zero divion exception'''
```

Example : Python program to demonstrate the usage of Index Error

```
#python program to handle index error exception
l=[1,2,3]
try:
    print(l[10])
except IndexError:
    print("You are not giving valid index")
```

output:-

```
You are not giving valid index
```

4.3 try except block:

4.3.1 Multiple except blocks :

Python allows you to have multiple except blocks for a single try block. The block which matches with the exception generated will get executed. A try block can be associated with more than one except block to specify handlers for different exceptions. However, only one handler will be executed. Exception handlers only

handle exceptions that occur in the corresponding try block. The syntax for specifying multiple except blocks for a single try block is as follows.

```
try:
        Operations are done in this block
        ....................................................
except Exception1:
        If there is Exception1,then executes this block.
except Exception2:
        If there is Exception2,then executes this block.
        ....................................................
else:
        If there is  no Exception,then executes this block.
        ....................................................
```

Example1:

```
try:
    x=int(input("Enter first number"))
    y=int(input("Enter second number"))
    print(x/y)
except ZeroDivisionError:
    print("Denominator can't be Zero")
```

```
except ValueError:
    print("plz provide int value only")
```

Output 1 :

```
Enter first number 5
Enter second number 0
Denominator can't be Zero
```

Output 2:

```
Enter first number10
Enter second number five
plz provide int value only
```

If try with multiple except blocks available then the order of these except blocks is important.Python interprter will always consider from top to bottom until matched except block identified.

Example:

```
try:
    x=int(input("Enter first number"))
    y=int(input("Enter second number"))
    print(x/y)
except ArithmeticError:
    print("Arithmetic Error")
except ZeroDivisionError:
```

```
    print("Denominator can't be Zero")
```

Output:

```
Enter first number10
Enter second number0
Arithmetic Error
```

Example:-#python program to handle multiple exceptions using multiple except blocks

```
a=int(input("Enter first number:"))
b=int(input("Enter second number:"))
l=[1,2,3]
n=int(input("Enter index value:"))
try:
    c=a/b
    print("The division result is",a/b)
    print("The element for the given index",l[n])
except ZeroDivisionError:
    print("U once check the denominator")
except IndexError:
    print("please give valid Index")
```

Output:

```
Enter first number:4
Enter second number:2
```

Enter index value:6

The division result is 2.0

please give valid

Output :

Enter first number:4

Enter second number:0

Enter index value:5

U once check the denominator

4.3.2 Multiple Exceptions in single except block

Python allows to handle any number of exceptions in single except block.

The syntax for single except block is

```
try:
        Block of statements
except (Exception1,Exception2,....):
        Exception handling code
finally:#Optional block
```

#python program to handle multiple exceptions using single except block

Example1:

```
try:
    x=int(input("Enter First Number:"))
    y=int(input("Enter Second Number:"))
```

```
    print(x/y)
except (ZeroDivisionError,ValueError) as msg:
    print("Please provide valid numbers only and problem is :",msg)
```

Output1:

```
Enter First Number:4
Enter Second Number:0
Please provide valid numbers only and problem is : division by zero
```

Output2:

```
Enter First Number:4
Enter Second Number:two
Please provide valid numbers only and problem is : invalid literal for int()
with base 10: 'two'
```

#python program to handle multiple exceptions using single except block

```
a=int(input("enter the number"))
b=int(input("enter the second number"))
l=[1,2,3]
try:
    c=a/b
    print("the division result is",c)
    print(l[20])
except (ZeroDivisionError,IndexError):
    print("you can verify the code in try block")
finally:
    print("GoodBye!")
```

output:-

Enter the number4

Enter the second number2

The division result is 2.0

You can verify the code in try block

GoodBye!

Except: without exception

The single except block without exception will capable to handle all types of exceptions with these programmer need not remember all types of possible exceptions that try block will throw.

Syntax:

```
try:
        Write the operations here
        ……..……..
except:
        If there is any exception,then executes this block
else:
        If there is no exception,then executes this block
```

Example1:

```
try:
    print(10/0)
except:
```

```
    print("Default Except")
except ZeroDivisionError:
   print("ZeroDivisionError")
```

Output:

SyntaxError:default'except:'must be last

Example 2:

```
#python program to handle more than one exception without exception list
a=int(input("Enter the number"))
b=int(input("Enter the second number"))
l=[1,2,4]
try:
    c=a/b
    print("The division result is",c)
    print(l[5])
except:
   print("Their is some exception in your code")
```

Output:

```
Enter the number10
Enter the second number5
The division result is 2.0
Their is some exception in your code
```

4.3.3 The Else clause :

The try ... except block can optionally have an *else clause*, which, when present, must follow all except blocks. The statement(s) in the else block is executed only if the try clause does not raise an exception.

#python program to demonstrate the else statement for exception handling

```
a=int(input("Enter the number"))
b=int(input("Enter the second number"))
try:
    c=a/b
    print("The division result is",c)
except:
    print("Their is some exception in your code")
else:
    print("this program is successfully executed without Exception")
```

output:-

```
Enter the number4
Enter the second number2
The division result is 2.0
this program is successfully executed without Exception
```

4.3.4 Finally block :

The try block has another optional block called finally which is used to define clean-up actions that must be executed under all circumstances. The finally block is always executed before leaving the try block. This means that the statements written in finally block are executed irrespective of whether an exception has occurred or not. The syntax of finally block can be given as,

```
try:
  Write your operations here
  ......................
  Due to any exception, operations written here will be skipped
finally:
  This would always be executed.
  ......................
```

Example1:

```
#case:1 If there is no exception
try:
    print("try")
except:
    print("except")
finally:
    print("finally")
```

Output:

try
Finally

Example2:

```
#case 2:If there is an exception raised and handled
try:
    print("try")
    print(10/0)
except ZeroDivisionError:
    print("except")
finally:
   print("finally")
```

Output:

try
except
finally

Example 3:

```
#case 3:If there is an exception raised but not handled
try:
    print("try")
    print(10/0)
```

```
except NameError:
    print("except")
finally:
    print("finally")
```

Output:

```
try
finally
ZeroDivisionError: division by zero   #Abnormal Termination
```

4.4 Raising Exceptions:

We can raise an exception using the raise keyword. The general syntax for the raise statement is,

raise [Exception [, args [, traceback]]]

Here, Exception is the name of exception to be raised (example, TypeError). *args* is optional and specifies a value for the exception argument. If args is not specified, then the exception argument is None. The final argument, *traceback*, is also optional and if present, is the traceback object used for the exception.

```
#python program to demonstrate the usage of raise
try:
    raise KeyError
except KeyError:
```

```
    print("The try block will throw KeyError Exception")
```

Output:-

```
The try block will throw KeyError Exception
```

Instantiating Exceptions

Python allows programmers to instantiate an exception first before raising it and add any attributes (or arguments) to it as desired. These attributes can be used to give additional information about the error. To instantiate the exception, the except block may specify a variable after the exception name. The variable then becomes an exception instance with the arguments stored in instance.args. The exception instance also has the __str__() method defined so that the arguments can be printed directly without using instance.args.

Example1:

```
try:
    raise Exception('Hello','World')
except Exception as cse:
    print(cse)
    print(cse.args)
    a,b=cse.args
    print(a)
```

```
    print(b)
```

output:

```
('Hello', 'World')
('Hello', 'World')
Hello
World
```

Re-raising an exception

Python allows programmers to re-raise an exception. For example, an exception thrown from the try block can be handled as well as re-raised in the except block using the keyword raise. The code given below illustrates this concept.

Example 1:

```
try:
    f=open("abc123.txt")
except:
    print("File does not exist")
   raise
```

Output:

```
File does not exist
Traceback (most recent call last):
  File "//192.168.0.180/lib/python fop/p6.py", line 2, in <module>
```

```
f=open("abc123.txt")
```

```
FileNotFoundError: [Errno 2] No such file or directory: 'abc123.txt
```

The Else Clause

We can use else block with try-except-finally blocks.

else block will be executed if and only if there are no exceptions inside try block.

try:

    Risky Code

except:

    will be executed if exception inside try

else:

    will be executed if there is no exception inside try

finally:

    will be executed whether exception raised or not raised and handled or not handled

Eg:

```
try:
        print("try")
        print(10/0)→ 1
except:
        print("except")
else:
        print("else")
finally:
        print("finally")
```

If we comment line-1 then else block will be executed b'z there is no exception inside try.

In this case the output is:

try
else
finally

If we are not commenting line-1 then else block won't be executed b'z there is exception inside try block. In this case output is:

try
except
finally

The try … except block can optionally have an else clause, which, when present, must follow all except blocks. The statement(s) in the else block is executed only if the try clause does not raise an exception.

```
try:
    file = open('File1.txt')
    str = file.readline()
    print(str)
except IOError:
    print("Error occurred during Input
...... Program Terminating...")
else:
    print("Program Terminating
Successfully.....")
```

OUTPUT

```
Hello
Program Terminating Successfully.....
```

```
try:
    file = open('File1.txt')
    str = f.readline()
    print(str)
except:
    print("Error occurred ...... Program
Terminating...")
else:
    print("Program Terminating
Successfully.....")
```

OUTPUT

```
Error occurred......Program
Terminating...
```

4.5 User Defined Exceptions :

Also known as Customized Exceptions or Programatic Exceptions.
Programmers may name their own exceptions by creating a new exception class. Exceptions need to be derived from the Exception class, either directly or indirectly. Although not mandatory, most of the exceptions are named as names that end in "**Error**" similar to naming of the standard exceptions in python. For example:

#program to generate the userdefined exception ShortInputException

```
class ShortInputException(Exception):
    def __init__(self,length):
        Exception(self)
```

```
        self.len=length
string=input("Enter the string")
length=len(string)
try:
    if length<3:
        raise ShortInputException(length)
except ShortInputException as s:
    print("The length of string is 3 or greater than 3 but your entered string length is ",s.len)
else:
   print("No Exception Proceed")
```

output:-

```
Enter the stringCSE
No Exception Proceed

Enter the stringhe
   The length of string is 3 or greater than 3 but your entered string length is  2'''
```

Some times we have to define and raise exceptions explicitly to indicate that something goes wrong, such type of exceptions are called User Defined Exceptions or Customized Exceptions.

Programmer is responsible to define these exceptions and Python not having any idea about these. Hence we have to raise explicitly based on our requirement by using "raise" keyword.

Eg:

InSufficientFundsException

InvalidInputException

TooYoungException

TooOldException

How to Define and Raise Customized Exceptions:

Every exception in Python is a class that extends Exception class either directly or indirectly.

Syntax:

```
class classname(predefined exception class name):
        definit(self,arg):
                self.msg=arg

class TooYoungException(Exception):
        definit(self,arg):
                self.msg=arg
```

TooYoungException is our class name which is the child class of Exception

We can raise exception by using raise keyword as follows

```
raise TooYoungException("message")
```

4.6 Significance of Files:

- **A file** is a collection of data stored on a secondary storage device like **hard disk**.
- When a program is being executed,its data is stored in **RAM**. Though RAM can be accessed faster by the CPU,it is also **volatile,**If you want to use the data in future,then you need to store this data on a permanent or non-volatile storage media such as hard disk, USB drive and DVD e.t.c.,
- A file is basically used because real-life applications involve large amounts of data and in such situations the console oriented I/O operations pose two major problems:
- ***First***, it becomes cumbersome and time consuming to handle huge amount of data through terminals.
- ***Second***, when doing I/O using terminal, the entire data is lost when either the program is terminated or computer is turned off. Therefore, it becomes necessary to store data on a permanent storage (the disks) and read whenever necessary, without destroying the data.

4.7 Types of files :

- Python supports two types of files.They are:

  1.ASCII Text Files

  2.Binary Files

4.7.1 ASCII Text Files

- A ***text file*** is a stream of characters that can be sequentially processed by a computer in forward direction. For this reason a text file is usually opened for only one kind of operation (reading, writing, or appending) at any given time.
- Because text files can process characters, they can only read or write data one character at a time. In Python, a text stream is treated as a special kind of file.
- Depending on the requirements of the operating system and on the operation that has to be performed (read/write operation) on the file, the newline characters may be converted to or from carriage-return/linefeed combinations.
- Besides this, other character conversions may also be done to satisfy the storage requirements of the operating system. However, these conversions occur transparently to process a text file. In a text file, each line contains zero or more characters and ends with one or more characters.
- Another important thing is that when a text file is used, there are actually two representations of data- internal or external. For example, an integer value will be represented as a number that occupies 2 or 4 bytes of memory internally but externally the integer value will be represented as a string of characters representing its decimal or hexadecimal value.

4.7.2 Binary Files :

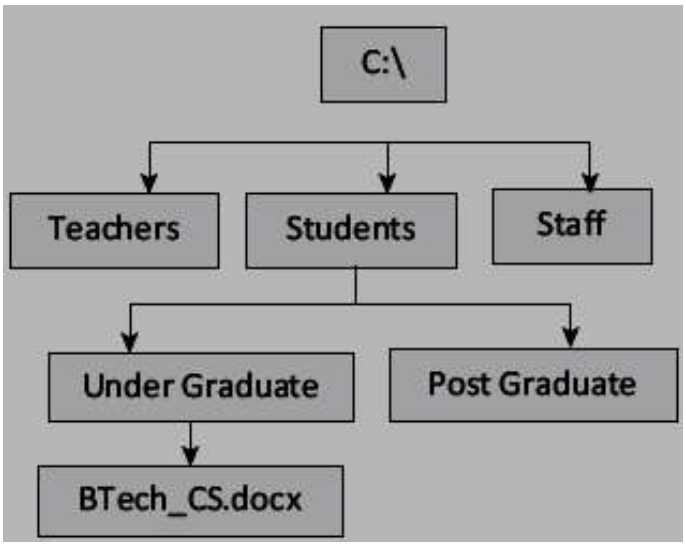
- A ***binary file***is a file which may contain any type of data, encoded in binary form for computer storage and processing purposes. It includes files such as word processing documents, PDFs, images, spreadsheets, videos, zip files and other executable programs.
- Like a text file, a binary file is a collection of bytes. A binary file is also referred to as a character stream with following two essential differences.
- A binary file does not require any special processing of the data and each byte of data is transferred to or from the disk unprocessed.
- Python places no constructs on the file, and it may be read from, or written to, in any manner the programmer wants.
- While **text files** can be processed **sequentially**, **binary files**, on the other hand, can be either processed **sequentially or randomly** depending on the needs of the application.

4.8 File Path:

- Files that we use are stored on a storage medium like the hard disk in such a way that they can be easily retrieved as and when required.
- Every file is identified by its path that begins from the root node or the root folder.
- In Windows, C:\ (also known as C drive) is the root folder but you can also have a path that starts from other drives like D:\, E:\, etc. The file path is also known as ***pathname***.
- In order to access a file on a particular disk we have two paths.

    1. Absolute Path

    2. Relative Path



4.8.1 A*bsolute path*:Absolute pathalways contains the root and the complete directory list to specify the exact location the file.

Example:To access BTech_CS.docx,The absolute path is

C:\Students\Under Graduate\BTech_CS.docx

4.8.2 Relative path: Relative path needs to be combined with another path in order to access a file. It starts with respect to the current working directory and therefore lacks the leading slashes.

Example: Suppose you are working on current directory Under Graduate in order to accessBTech_CS.docx,The Relative path is

Under Graduate\BTech_CS.docx

4.8.3 Differences between Absolute Path and Relative Path

Parameter of Comparison	Absolute Path	Relative Path
By definition	specifies the location from the root directory	related to the location from current directory
Function of delimiting character	Begins with a delimiting character	Never begins with a delimiting character
Navigates to	Content from other domains	Content from the same domain
URL used	Uses absolute URL	Used relative URL
Other names	Full-path or File path	Non-absolute path

File Operations

- When we want to read from or write to a file we need to open it first. When we are done, it needs to be closed, so that resources that are tied with the file are freed.
- Python has many in-built functions and methods to manipulate files. Hence, in Python, a file operation takes place in the following order.
    1. **Open a file**
    2. **Read or write (perform operation)**

3. Close the file

Opening a File

- Before reading from or writing to a file, you must first open it using Python's built-in open() function. This function creates a file object, which will be used to invoke methods associated with it.
- The Syntax of open() is:

fileObj = open(file_name [, access_mode])

Where file_name is a string value that specifies name of the file that you want to access. *access_mode* indicates the mode in which the file has to be opened, i.e., read, write, append, etc.

Example: Write a Program to print the details of file object

```
file=open("file1.txt","rb")
print(file)
```

Output:

<open file "file1.txt",mode 'rb' at 0X02A850D0>

Access mode is an optional parameter and the default file access mode isread(r).

4.9 File modes :

- Python supports the following access modes for opening a file those are

- r for reading – The file pointer is placed at the beginning of the file. This is the default mode.
- rb Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file.
- r+ Opens a file for both reading and writing. The file pointer will be at the beginning of the file.
- rb+ Opens a file for both reading and writing in binary format.
- w Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
- Wb opens a file in binary format for writing only.When a file is opened in w mode ,two things can happen.If the file does not exist,a new file is created for writing.
- w+ Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, it creates a new file for reading and writing.
- wb+ Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, it creates a new file for reading and writing.
- a Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
- ab Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.

- a+ Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
- ab+ Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

x ->To open a file in exclusive creation mode for write operation. If the file already exists then we will get FileExistsError.

The File Object Attributes

Once a file is successfully opened, a *file* object is returned. Using this file object, you can easily access different type of information related to that file. This information can be obtained by reading values of specific attributes of the file.

The Following table shows list attributes related to file object.

Attribute	Information Obtained
fileObj.closed	Returns true if the file is closed and false otherwise
fileObj.mode	Returns access mode with which file has been opened
fileObj.name	Returns name of the file

Example: Program to open a file and print its attribute values.

```
file = open("File1.txt", "wb")
print("Name of the file: ", file.name)
print("File is closed.", file.closed)

print("File has been opened in ", file.mode, "mode")
```

OUTPUT

```
Name of the file:  File1.txt
File is closed. False
File has been opened in wb mode
```

Various Properties of File Object:

Once we opend a file and we got file object, we can get various details related to that file by using its properties.

- **name → Name of opened file**
- **mode → Mode in which the file is opened**
- **closed → Returns boolean value indicates that file is closed or not**
- **readable()→ Retruns boolean value indicates that whether file is readable or not**
- **writable()→ Returns boolean value indicates that whether file is writable or not.**

```
1) f=open("abc.txt",'w')
2) print("File Name: ",f.name)
3) print("File Mode: ",f.mode)
4) print("Is File Readable:  ",f.readable())
5) print("Is File Writable:  ",f.writable())
6) print("Is File Closed : ",f.closed)
7) f.close()
8) print("Is File Closed : ",f.closed)
```

Output

```
D:\Python_classes>py test.py
File Name: abc.txt
File Mode:  w
Is File Readable: False
Is File Writable: True
Is File Closed: False
Is File Closed: True
```

Closing A File

- The *close() method* is used to close the file object. Once a file object is closed, you cannot further read from or write into the file associated with the file object.
- While closing the file object the close() flushes any unwritten information. Although, Python automatically closes a file when the reference object of a file is reassigned to another file, but as a good programming habit you should always explicitly use the close() method to close a file.
- The syntax of close() is

fileObj.close()

- The close() method frees up any system resources such as file descriptors, file locks, etc. that are associated with the file.
- Once the file is closed using the close() method, any attempt to use the file object will result in an error.

Example: Write a Python program to assess if a file is closed. **(April 2018 Regular)**

```
file = open('File1.txt','wb')
print('Name of the file :',file.name)
print('File is closed:',file.closed)
print('File is now being closed')
file.close()
print('File is closed',file.closed)
print(file.read())
```

Output:

```
Name of the file : File1.txt
File is closed: False
File is now being closed
File is closed True
Traceback (most recent call last):
  File "D:/Python/sample.py", line 7, in <module>
    print(file.read())
io.UnsupportedOperation: read
```

4.10 Understanding read functions : read( ) ,readline() and readlines( )

- The *read() method* is used to read a string from an already opened file.
- **The syntax of read() method is**

fileObj.read([count])

Where In the above syntax, count is an optional parameter which if passed to the read() method specifies the number of bytes to be read from the opened file.

- The read() method starts reading from the beginning of the file and if *count* is missing or has a negative value then, it reads the entire contents of the file (i.e., till the end of file).

Reading Character Data from Text Files

We can read character data from text file by using the following read methods.

read()->To read total data from the file

read(n) ->To read 'n' characters from the file

readline()->To read only one line

readlines()->To read all lines into a list

Example1: Program to print the first 8 characters of the file data.txt

```
file=open('data.txt','r')
print(file.read(8))
file.close()
```

data.txt

hellocsehope to enjoylearningpython programming

Have a nice day

Output:

```
hellocse
```

Example2: Program to display the content of file using for loop

```
file=open('data.txt','r')
for line in file:
    print(line)
file.close()
```

data.txt

hellocsehope to enjoylearningpython programming

Have a nice day

Output:

```
hellocsehope to enjoylearningpython programming
Have a nice day
```

readline() Method

- It is used to read single line from the file.
- This method returns an empty string when end of the file has been reached.

Example : Program to demonstrate the usage of readline() function

```
file=open('data.txt','r')
```

data.txt

```
hellocsehope to enjoylearningpython programming
Have a nice day
print('firtsline:',file.readline())
print('second line:',file.readline())
print('third line:',file.readline())
file.close()
```

Output:

```
firtsline: hellocsehope to enjoylearningpython programming
second line: Have a nice day
third line:
```

readlines() Method

readlines() Method is used to read all the lines in the file.

data.txt

```
hellocsehope to enjoylearningpython programming
Have a nice day
```

Example: Program to demonstrate readlines() function

```
file=open('data.txt','r')
```

```
print(file.readlines())
file.close()
```

Output:

```
['hellocsehope to enjoylearningpython programming\n', 'Have a nice day']
```

4.11 Understanding write functions: :Write(), writeline() and Writelines()

- The *write() method* is used to write a string to an already opened file. Of course this string may include numbers, special characters or other symbols.
- While writing data to a file, you must remember that the write() method does not add a newline character ('\n') to the end of the string.
- The syntax of write() method is:

fileObj.write(string)

Example: Program that writes a message in the file,data.txt

```
file=open('data.txt','w')
file.write('hello cse we are learning python programming')
file.close()
print('file writing successful')
```

data.txt

hello cse we are learning python programming

Output:

```
file writing successful
```

writeline() method:

- The *writelines() method* is used to write a list of strings.

Example: Program to write to afile using the writelines() method

data.txt

hellocsehope to enjoylearningpython programming

```
file=open('data.txt','w')
lines=['hello','cse','hope to enjoy','learning','python programming']
file.writelines(lines)
file.close()
print('file writing successful')
```

Output:

file writing successful

append() method:

- Once you have stored some data in a file,you can always open that file again to write more data or append data to it.
- To append a file, you must open it using 'a' or 'ab' mode depending on whether it is text file or binary file.
- Note that if you open a file with 'w' or 'wb' mode and then start writing data into it, then the existing contents would be overwritten.

Example: Program to append data to an already existing file

```
file=open('data.txt','a')
```

```
file.write('\nHave a nice day')

file.close()

print('Data appended successful')
```

Output:

```
Data appended successful
```

list() Method

- list() method is also used to display entire contents of the file.you need to pass the file object as an argument to the list() method.

Example: Program to display the contents of the file data.txt using the list() method

data.txt

```
hellocsehope to enjoylearningpython programming
Have a nice day
file=open('data.txt','r')
print(list(file))
file.close()
```

Output:

```
['hellocsehope to enjoylearningpython programming\n', 'Have a nice day']
```

```
 # reading ENTIRE content of file  is returned as string
f=open('write_demo.txt')
```

```
data=f.read()
print('Content in file is ',data)
f.close()
'''Content in file is  Python Programming

File Handling Programs'''

#read(n)==> read only 'n' chars from the file

f=open('write_demo.txt')

data=f.read(6)

print('6 chars from file: ',data)

data=f.read(19)  # counts '\n' chars also

print('19 chars from file: ',data)

f.close()

'''6 chars from file:  Python

19 chars from file:   Programming

File H'''

# if n is -ve==> reads the entire file content

f=open('write_demo.txt')

print('redaing with -ve n value')

data=f.read(-100)

print(data)

#readind data line-by-line
```

```
'''9014272890

priyankacstau@gmail.com

8019726567

priyadatta24986@gmail.com

read()==>entire

read(n)'''

#readline()

f=open('write_demo.txt')

line1=f.readline()

print('Mobile No: ',line1)

line2=f.readline()

print('Email ID: ',line2)

f.close()

Mobile No:  9014272890

Email ID:  priyankacstau@gmail.com

'''

f=open('write_demo.txt')

l=f.readlines()

# reads all lines into a list

print('lines as list', l)
```

```
for x in l:
    print(x,end='')
f.close()
'''
```

lines as list ['9014272890\n', 'priyankacstau@gmail.com\n', '8019726567\n', 'priyadatta24986@gmail.com']

9014272890

priyankacstau@gmail.com

8019726567

- priyadatta24986@gmail.com

Eg 1: To read total data from the file	**Output**
f=open("abc.txt",'r') **data=f.read()** print(data) f.close()	Sunny Bunny chinny vinny
Eg 2: To read only first 10 characters from the file f=open("abc.txt",'r')	**Output** sunny Bunn

data=f.read(10) print(data) f.close()	
Eg 3: To read data line by line from the file f=open("abc.txt",'r') **line1 = f.readline()** print(line1, end='') **line2 = f.readline()** print(line2, end='') **line3 = f.readline()** print(line3, end='') f.close()	**Output** Sunny Bunny chinny
Ex: to read all lines into list f=open("abc.txt",'r') **lines=f.readlines()** **for l in lines:** print(l, end='')	**Output** Sunny Bunny chinny vinny

f.close()	
Ex: to read all lines into list f=open("abc.txt",'r') print(f.read(3)) print(f.readline()) print(f.read(4)) print("remaining data") print(f.read())) f.close()	Sun Ny Bunn Remaining data y Chinny vinny

Opening a file using with keyword:

- It is good programming habit to use the ***with*** keyword when working with file objects.
- This has the advantage that the file is properly closed after it is used even if an error occurs during read or write operation or even when you forget to explicitly close the file.

```
with open("file1.txt", "rb") as file:
    for line in file:
        print(line)
print("Let's check if the file is
closed : ", file.close())
```

OUTPUT

```
Hello World
Welcome to the world of Python
Programming.
Let's check if the file is closed :  True
```

```
file = open("file1.txt", "rb")
for line in file:
      print(line)
print("Let's check if the file is
closed : ", file.close())
```

OUTPUT

```
Hello World
Welcome to the world of Python
Programming.
Let's check if the file is closed : False
```

Note: *When you open a file for reading, or writing, the file is searched in the current directory. If the file exists somewhere else then you need to specify the path of the file.*

Splitting Words:

- *Python allows you to read line(s) from a file and splits the line (treated as a string) based on a character. By default, this character is space but you can even specify any other character to split words in the string.*

Example*: Program to split the line into series of words and use space to perform the split operation*

data.txt

hellocsehope to enjoylearningpython programming

```
with open('data.txt','r') as file:
    line=file.readline()
    words=line.split()
    print(words)
```

Output:

['hellocsehope', 'to', 'enjoylearningpython', 'programming']

Some Other Useful File Methods:

- The following are some of additional methods which will work on files

Method	Description	Example
fileno()	Returns the file number of the file (which is an integer descriptor)	file = open("File1.txt", "w") print(file.fileno()) **OUTPUT** 3
flush()	Flushes the write buffer of the file stream	file = open("File1.txt", "w") file.flush()
isatty()	Returns True if the file stream is interactive and False otherwise	file = open("File1.txt", "w") file.write("Hello") print(file.isatty()) **OUTPUT** False
readline(n)	Reads and returns one line from file. n is optional. If n is specified then atmost n bytes are read	file = open("Try.py", "r") print(file.readline(10)) **OUTPUT** file = ope
truncate(n)	Resizes the file to n bytes	file = open("File.txt", "w") file.write("Welcome to the world of programming....") file.truncate(5) file = open("File.txt", "r") print(file.read()) **OUTPUT** Welco
rstrip()	Strips off whitespaces including newline characters from the right side of the string read from the file.	file = open("File.txt") line = file.readline() print(line.rstrip()) **OUTPUT** Greetings to All !!!

4.12 Manipulating file pointer using seek.:

File Positions: tell() and seek()

- With every file, the file management system associates a pointer often known as ***file pointer***that facilitates the movement across the file for reading and/ or writing data.
- The file pointer specifies a location from where the current read or write operation is initiated. Once the read/write operation is completed, the pointer is automatically updated.

- Python has various methods that tells or sets the position of the file pointer.
- For example, the **tell()** method tells the current position within the file at which the next read or write operation will occur. It is specified as number of bytes from the beginning of the file.
- When you just open a file for reading, the file pointer is positioned at location 0, which is the beginning of the file.

Example:

```
f=open("abc.txt","r")
print(f.tell())
print(f.read(2))
print(f.tell())
print(f.read(3))
print(f.tell())
```

Output:

```
>>>
0
su
2
nny
5
```

seek():

We can use seek() method to move cursor (file pointer) to specified location. [Can you please seek the cursor to a particular location]

f.seek(offset, fromwhere) à offset represents the number of positions

The allowed Values for 2nd Attribute (from where) are

0 ->From beginning of File (Default Value)

1->From Current Position

2 >From end of the File

- The syntax for seek() function is

seek(offset[, from])

- The offset argument indicates the number of bytes to be moved and the from argument specifies the reference position from where the bytes are to be moved.

Note: Python 2 supports all 3 values but Python 3 supports only zero.

Example:

```
data="All students are STUPIDS"
f=open("abc.txt","w")
f.write(data)
with open("abc.txt","r+") as f:
    text=f.read()
    print(text)
```

```
print("The current cursor Position:",f.tell())
f.seek(17)
print("The current cursor position",f.tell())
f.write("GEMS!!!")
f.seek(0)
text=f.read()
print("Data After Modification")
print(text)
```

Output:

```
All students are STUPIDS
The current cursor Position: 24
The current cursor position 17
Data After Modification
All students are GEMS!!!
```

Example :

```
file=open("File.txt","rb")
print("position of file pointer before reading is:",file.tell())
print(file.read(10))
print("position of file pointer after reading is:",file.tell())
```

```
print("setting 3 bytes from the current position of file pointer")
file.seek(3,1)
print(file.read())
file.close()
```

Output:

```
position of file pointer before reading is:0
Hello All,
position of file pointer after reading is:10
setting 3 bytes from the current position of file pointer
Pe you are enjoying learning python
```

Example Programs

1.Write a program to print each line of a file in reverse order

input.txt

```
Hello hi
How are you
```

Program:

```
with open('input.txt','r') as fp:
    for line in fp:
        print (line[::-1])
```

Output:

```
ih olleH
```

uoy era woH

2. Write a program to compute the number of characters, words and lines in a file

Program:

data.txt

hellocsehope to enjoylearningpython programming

Have a nice day

```
fname = "data.txt"
num_lines = 0
num_words = 0
num_chars = 0
with open(fname, 'r') as f:
 for line in f:
        words = line.split()
        num_lines += 1
        num_words += len(words)
        num_chars += len(line)
print('The no of lines in a given file is',num_lines)
print('The no of words in a given file is',num_words)
print('The no of chars in a given file is',num_chars)
```

Output:

The no of lines in a given file is 2

The no of words in a given file is 8

The no of chars in a given file is 63

3. Write a program to copy contents of one file to another file

Program:

```
with open("data.txt","r") as f:
    with open("out.txt", "w") as f1:
        for line in f:
            f1.write(line)
```

Output:

data.txt

hellocsehope to enjoylearningpython programming

Have a nice day

out.txt

hellocsehope to enjoylearningpython programming

Have a nice day

4. Write a Python Program to count number of Vowels and Number of Consonants in a given file. (November 2018 Supplementary)

Program:

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

```
vowels = set("AEIOUaeiou")
cons = set("bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ")
countV = 0
countC = 0
for c in infile.read():
    if c in vowels:
        countV += 1
    elif c in cons:
        countC += 1
print("The no of Vowels are",countV)
print("The no of Consonants are",countC)
```

5.Write a Python program to print the percentage of vowels and Consonants count in the given text file.

```
fileName = input("Enter the file to check: ").strip()
infile = open(fileName, "r")
vowels = set("AEIOUaeiou")
cons =
set("bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ")
count=0;
countV = 0
countC = 0
for c in infile.read():
   count++
   if c in vowels:
      countV += 1
   elif c in cons:
      countC += 1
Vowelpercentage=(countV/count)*100
Conspercentage=(countC/count)*100
print("The vowel Percentage is",Vowelpercentage)
print("The Consonant Percentage is ",Conspercentage)
```

6. Write a Python program to convert text file content into upper case letters.

Program:

```
file=open('data.txt','r')
for line in file:
   print(line.upper())
file.close()
```

data.txt:

```
hai hello
gudlavalleru
engineering college
```

Expectedoutput:

```
HAI HELLO
GUDLAVALLERU
ENGINEERING COLLEGE
```

5.1 Command line Arguments:

- The Python sys module provides access to any command-line arguments via the ***sys.argv***. This serves two purposes –
  - ***sys.argv*** is the list of command-line arguments.
  - ***len(sys.argv)*** is the number of command-line arguments.
- Here ***sys.argv[0]*** is the program ie. script name.

Example1:Write a Python program to demonstrate the usage of Command Line Arguments

sample11.py

```
#!/usr/bin/python

import sys

print ('Number of arguments:', len(sys.argv), 'arguments.')

print ('Argument List:', str(sys.argv))
```

Output:

```
C:\Python34>sample11.py 1 2 3
Number of arguments: 4 arguments.
Argument List: ['C:\\Python34\\sample11.py', '1', '2', '3']
```

Example2:

```
#python program to demonstrate command line arguments
import sys
print("The name of the file is",sys.argv[0])#program name
print("The no of parameters is",len(sys.argv))
print("The first parameter is",sys.argv[1])# the first parameter
print("The second parameter is",sys.argv[2])# the second parameter
```

Output:

```
D:\Pythonprogramscseb>python add.py
Enter the first number23
Enter the second number34
The sum of two number is 57
```

Example3: Write a Python Program to add two numbers using Command line Arguments.

```
#Week1a write a python program to add two numbers using command line arguments
import sys
num1=int(sys.argv[1])
num2=int(sys.argv[2])
print("The num1 value is",num1)
print("The num2 value is",num2)
print("The sum of two numbers is",num1+num2)
```

Output:

```
D:\Pythonprogramscseb>python Week1a.py 4 9
The num1 value is 4
The num2 value is 9
The sum of two numbers is 13

D:\Pythonprogramscseb>python Week1a.py 20 -5
The num1 value is 20
The num2 value is -5
The sum of two numbers is 15
```

Example: Write a Python program to copy the content of one file to another using command line arguments.

input.txt

```
Hello hi
```

sample12.py

```
#!/usr/bin/python
```

```
import sys

print ('Number of arguments:', len(sys.argv), 'arguments.')

with open(str(sys.argv[1])) as f:

    with open((sys.argv[2]), "w") as f1:

        for line in f:

            f1.write(line)

print('File Copied Success')
```

output.txt

Hello hi

```
C:\Python34>sample12.py input.txt output.txt
Number of arguments: 3 arguments.
File Copied Success
```

Assignment-Cum-Tutorial Questions

A. Objective Questions

1._____ errors occur due to poor understanding of a problem and its solutions.

2._____ exceptions can be controlled by the program.

3. If no exception occurs, the _____ block is skipped.

4. The keyword used re-raise an exception is _____.

5.User-defined exceptions are created by inheriting the _____ class.

6.To handle an exception, try block should be immediately followed by which block?

[ ]

a) try: b)catch: c)else: d)except:

7.which statement raises exception if the expression is False? [ ]

a)Throw b)raise c)else d)assert

8. Identify The right way to close a file. [ ]

a) File.close() b)close(file) c)close("file") d)File.closed

9. A file is stored in _____ memory.
[ ]

a)primary b)secondary c)cache d)volatile

10. What will happen when a file is opened in write mode and then immediately closed.
[ ]

a)File contents are deleted

b) Nothing Happens

c) A Blank Line is written to the file

d) an error occurs

11. The default access mode of the file is _____.

12. Identify the delimiter in the Solaris file system
[ ]

a)/ b)\ c): d)|

13. By default a new file is created in which directory
[ ]

a)root b) current working c) Python directory d) D Drive

14. Identify the correct way to write "Welcome to Python" in a file
[ ]

a) write(file,"Welcome to python")

b) write("Welcome to Python",file)

c) file.write(“Welcome to Python”)

d) ”Welcome to Python”.write(file)

15. Predict the output of the following program [ ]

```
f = None
for i in range (5):
    with open("data.txt", "w") as f:
        if i > 2
            break
print(f.closed)
```

a)True b) False c) None d) Error

16. Fill in the blank to open a file,read its content and prints its length

```
file=_____(“file.txt”,”r”)
text=file._____()
print(_____(text))
file.close()
```

17. If count is missing or has a negative value in the read() method then, no contents are read from the file. [True/False]

18.os.path.abs() method accepts a file path as an argument and returns True if the path is an absolute path and False otherwise [True/False]

B. Descriptive Questions

1. Distinguish between error and exception. **[BL:2]**

2. Explain the syntax of try- except block. **[BL:2]**

3. How can we handle multiple exceptions in a program. **[BL:2]**

4. Explain any three built in exceptions with relevant examples.**[BL:1]**

5. How can we create your own exception in python.**[BL:3]**

6. What will happen if an exception generated in the try block is immediately followed by a finally block? Discuss both the cases(except block not present and except block present at the next higher level).**[BL:2]**

7. Define file. Explain about the importance of files in Python. **[BL:1]**

8. Define path. Distinguish between absolute and relative path with an example. **[BL:1]**

9.Discuss briefly about various types of file. **[BL:1]**

10.Explain in detail about various modes of file. **[BL:1]**

11. With an example program give an overview of file positions and its methods. **(oct-2021)[BL:2]**

12.Explain different file operations with suitable programming examples. **[BL:2]**

13.What is the purpose of opening a file using with keyword.**[BL:1]**

14.Develop a Python program to count number of vowels and consonants in a given text file. **[BL:3]**

15. Develop a Python program that reads data from a file and Calculates the percentage of vowels and consonants in the file. **[BL:3]**

16. Develop a Python program that copies one file to another in such away that all comment lines are skipped and not copied in the destination file. **[BL:3]**

17. Develop a python program to find no of lines, words and characters in a given text file. **[BL:3]**

18. Develop a Python program to combine each line from first file with the corresponding line in second file. **[BL:3]**

19. Develop a program that accepts file name as an input from the user. Open the file and count number of times a character appears in the file. **[BL:3]**

20. Develop a program that tells and sets the position of the file pointer. **[BL:3]**

21. Develop a python program to compare two text files and print first byte position where they differ. **(oct-2021) [BL:3]**

22. Develop a python program to read a file and capitalize the first letter of every word in the file. **(oct-2021) [BL:3]**

23. Develop a python program to demonstrate the usage of renaming and removing files. **(apr 2021) [BL:3]**

24. Develop a python program to read a string from the user and append it into a file. **[BL:3]**