12 EXCEPTION HANDLING IN PYTHON (NCERT CLASS 12)

INTRODUCTION

Software programs and applications do not always work flawlessly. When we write a program, we can make mistakes that cause errors when we run it. In Python, you may encounter two types of mistakes: syntax errors and exceptions. Before enabling the rest of the program to run, you may wish to test a specific block of code to ensure it works properly. Python try except blocks allow you to test your code and handle exceptions if they occur.

An exception is defined as an unexpected condition in a program that causes the program's flow to be interrupted.

When the Python interpreter attempts to execute invalid code, it throws an exception, and if the exception is not handled, it disturbs the regular flow of the program's instructions and outputs a traceback. Exceptions are a form of error in which the code has the correct syntax but contains a fault. There are many different types of exceptions, but some of the most prevalent are: ArithmeticError, ImportError, ZeroDivisionError, NameError, and TypeError.

As Python developers, we must consider various exception circumstances and incorporate error management into your code. Python, fortunately, includes a sophisticated error handling system. Python applications may determine the error type at run time and act accordingly by using structured exception handling and a collection of pre-defined exceptions. These actions can include adopting a different route, using default settings, or urging for accurate input.

SYNTAX ERRORS

(I) Syntax errors are detected when we have not followed the rules of the particular programming language while writing a program. These errors are also known as parsing errors.

(II) On encountering a syntax error, the interpreter does not execute the program unless we rectify the errors, save and rerun the program. When a syntax error is encountered while working in shell mode, Python displays the name of the error and a small description about the error.

(III) So, a syntax error is reported by the Python interpreter giving a brief explanation about the error and a suggestion to rectify it.

(IV) Similarly, when a syntax error is encountered while running a program in script mode, a dialog box specifying the name of the error and a small description about the error is displayed.

EXCEPTIONS

An exception is a Python object that represents an error. When an error occurs during the execution of a program, an exception is said to have been raised. Such an exception needs to be handled by the programmer so that the program does not terminate abnormally.

(I) Built-in Exceptions:- Commonly occurring exceptions are usually defined in the compiler/ interpreter. These are called built-in exceptions. Some of the commonly occurring built-in exceptions that can be raised in Python are explained in Table below.

S.No	Name of the Built-in Exception	Explanation
1.	SyntaxError	It is raised when there is an error in the syntax of the Python code.
2.	ValueError	It is raised when a built-in method or operation receives an argument that has the right data type but mismatched or inappropriate values.
3.	IOError	It is raised when the file specified in a program statement cannot be opened.
4.	KeyboardInterrupt	It is raised when the user accidentally hits the Delete or Esc key while executing a program due to which the normal flow of the program is interrupted.
5.	ImportError	It is raised when the requested module definition is not found.
6.	EOFError	It is raised when the end of file condition is reached without reading any data by input().
7.	ZeroDivisionError	It is raised when the denominator in a division operation is zero.
8.	IndexError	It is raised when the index or subscript in a sequence is out of range.

9.	NameError	It is raised when a local or global variable name is not defined.
10.	IndentationError	It is raised due to incorrect indentation in the program code.
11.	TypeError	It is raised when an operator is supplied with a value of incorrect data type.
12.	OverFlowError	It is raised when the result of a calculation exceeds the maximum limit for numeric data type.

A programmer can also create custom exceptions to suit one's requirements. These are called user-defined exceptions.

RAISING EXCEPTIONS

Each time an error is detected in a program, the Python interpreter raises (throws) an exception. Raising an exception involves interrupting the normal flow execution of program and jumping to that part of the program (exception handler code) which is written to handle such exceptional situations.

(I) **The raise Statement:** The raise statement can be used to throw an exception. The syntax of raise statement is :

raise exception-name[(optional argument)]

The argument is generally a string that is displayed when the exception is raised.

In addition to the error message displayed, Python also displays a stack Traceback. This is a structured block of text that contains information about the sequence of function calls that have been made in the branch of execution of code in which the exception was raised.

(II) **The assert Statement:** An assert statement in Python is used to test an expression in the program code. If the result after testing comes false, then the exception is raised. This statement is generally used in the beginning of the function or after a function call to check for valid input. The syntax for assert statement is:

assert Expression[,arguments]

On encountering an assert statement, Python evaluates the expression given immediately after the assert keyword. If this expression is false, an AssertionError exception is raised which can be handled like any other exception.

```
Program:- Use of assert statement
    print("use of assert statement")
    def negativecheck(number):
assert(number>=0), "OOPS... Negative Number"
        print(number*number)
print (negativecheck(100))
print (negativecheck(-350))
```

In case the number gets a negative value, AssertionError will be thrown, and subsequent statements will not be executed. Hence, on passing a negative value (-350) as an argument, it results in AssertionError and displays the message "OOPS.... Negative Number".

Handling Exceptions

Each and every exception has to be handled by the programmer to avoid the program from crashing abruptly. This is done by writing additional code in a program to give proper messages or instructions to the user on encountering an exception. This process is known as exception handling.

- **Need for Exception Handling:** Exception handling is being used not only in Python programming but in most programming languages like C++, Java, Ruby, etc. It is a useful technique that helps in capturing runtime errors and handling them so as to avoid the program getting crashed.

Important points regarding exceptions and their handling:

(I) Python categorises exceptions into distinct types so that specific exception handlers (code to handle that particular exception) can be created for each type.

(II) Exception handlers separate the main logic of the program from the error detection and correction code. The segment of code where there is any possibility of error or exception, is placed inside one block. The code to be executed in case the exception has occurred, is placed inside another block. These statements for detection and reporting the exception do not affect the main logic of the program.

(III) The compiler or interpreter keeps track of the exact position where the error has occurred.

(IV) Exception handling can be done for both user-defined and built-in exceptions.

- **Process of Handling Exception:** When an error occurs, Python interpreter creates an object called the exception object. This object contains information about the error like its type, file name and position in the program where the error has occurred. The object is handed over to the runtime system so that it can find an appropriate code to handle this particular exception. This process of creating an exception object and handing it over to the runtime system is called throwing an exception.

The runtime system searches the entire program for a block of code, called the exception handler that can handle the raised exception.

A runtime system refers to the execution of the statements given in the program. It is a complex mechanism consisting of hardware and software that comes into action as soon as the program, written in any programming language, is put for execution.

A runtime system first searches for the method in which the error has occurred and the exception has been raised. If not found, then it searches the method from which this method (in which exception was raised) was called. This hierarchical search in reverse order continues till the exception handler is found. This entire list of methods is known as call stack.

When a suitable handler is found in the call stack, it is executed by the runtime process. This process of executing a suitable handler is known as catching the exception. If the runtime system is not able to find an appropriate exception after searching all the methods in the call stack, then the program execution stops.

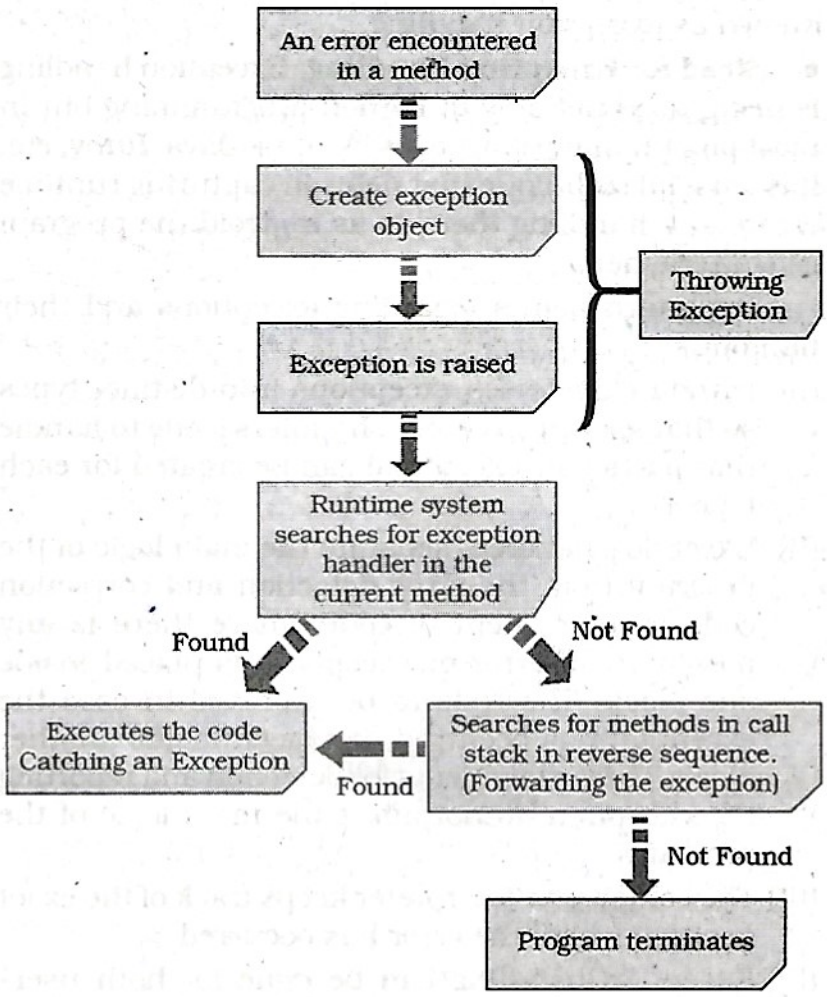

FIG:- Steps of handling exception

- **Catching Exceptions :** An exception is said to be caught when a code that is designed to handle a particular exception is executed. Exceptions, if any, are caught in the try block and handled in the except block.

While writing or debugging a program, a user might doubt an exception to occur in a particular part of the code. Such suspicious lines of codes are put inside a try block.

Every try block is followed by an except block. The appropriate code to handle each of the possible exceptions (in the code inside the try block) are written inside the except clause.

The syntax of try ... except clause is as follows:

```
try:
[ program statements where exceptions might occur]
except [exception-name]:
[ code for exception handling if the exception-name error is encountered]
```

- **try...except...else clause:** We can put an optional else clause along with the try...except clause. An except block will be executed only if some exception is raised in the try block.But if there is no error then none of the except blocks will be executed. In this case, the statements inside the else clause will be executed.

FINALLY CLAUSE

The try statement in Python can also have an optional finally clause. The statements inside the finally block are always executed regardless of whether an exception has occurred in the try block or not. It is a common practice to use finally clause while working with files to ensure that the file object is closed. If used, finally should always be placed at the end of try clause, after all except blocks and the else block.

Use of finally clause

```
print ("Handling exception using try...except...else...finally")
try:
numerator=50
denom=int(input("Enter the denominator: "))
quotient=(numerator/denom)
print ("Division performed successfully")
except ZeroDivisionError:
print ("Denominator as ZERO is not allowed")
except ValueError:
print ("Only INTEGERS should be entered")
else:
print ("The result of division operation is ", quotient)
finally:
print ("OVER AND OUT")
```

In the above program, the message "OVER AND OUT" will be displayed irrespective of whether an exception is raised or not.

(I) Recovering and continuing with finally clause:- If an error has been detected in the try block and the exception has been thrown, the appropriate except block will be executed to handle the error. But if the exception is not handled by any of the except clauses, then it is re-raised after the execution of the finally block.

After execution of finally block, Python transfers the control to a previously entered try or to the next higher level default exception handler. In such a case, the statements following the finally block is executed. That is, unlike except, execution of the finally clause does not terminate the exception. Rather, the exception continues to be raised after execution of finally.

□□□

13 FILE HANDLING IN PYTHON
(NCERT CLASS 12)

INTRODUCTION

Programs which we have done so far, are the ones which run, produce some output and end. Their data disappears as soon as they stop running. Next time when you use them, you again provide the data and then check the output. This happens because the data entered is stored in primary memory, which is temporary in nature.

What if the data with which, we are working or producing as output is required for later use? Result processing done in Term Exam is again required for Annual Progress Report. Here if data is stored permanently, its processing would be faster. This can be done, if we are able to store data in secondary storage media i.e. Hard Disk, which we know is permanent storage media.

Data is stored using file(s) permanently on secondary storage media. Data in Word processing applications, Spreadsheets, Presentation applications, etc. all of them created data files and stored your data, so that you may use the same later on. Apart from this you were permanently storing your python scripts (as .py extension) also.

A file (i.e. data file) is a named place on the disk where a sequence of related data is stored. In python files are simply stream of data, so the structure of data is not stored in the file, along with data. Basic operations performed on a data file are:

(I) Naming a file
(II) Opening a file
(III) Reading data from the file
(IV) Writing data in the file
(V) Closing a file

Using these basic operations, we can process file in many ways, such as :

(I) Creating a file
(II) Traversing a file for displaying the data on screen
(III) Appending data in file
(IV) Inserting data in file
(V) Deleting data from file
(VI) Create a copy of file
(VII) Updating data in the file, etc.

Python allow us to create and manage two types of file :

(I) Text
(II) Binary

- **TEXT FILE :** A text file is usually considered as sequence of lines. Line is a sequence of characters (ASCII), stored on permanent storage media. Although default character coding in python is ASCII but using constant u with string, supports Unicode as well. As we talk of lines in text file, each line is terminated by a special character, known as End of Line (EOL). From strings we know that \n is newline character. So at the lowest level, text file will be collection of bytes. Text files are stored in human readable form and they can also be created using any text editor.
- **Binary File :** A binary file contains arbitrary binary data i.e. numbers stored in the file, can be used for numericaloperation(s). So when we work on binary file, we have to interpret the raw bit pattern(s) read from the file into correct type of data in our program. It is perfectly possible to interpret a stream of bytes originally written as string, as numeric value. But we know that will be incorrect interpretation of data and we are not going to get desired output after the file processing activity. So in the case of binary file it is extremely important that we interpret the correct data type while reading the file. Python provides special module(s) for encoding and decoding of data for binary file.

To handle data files in python, we need to have a file object. Object can be created by using open() function or file() function. To work on file, first thing we do is open it. This is done by using built in function open(). Using this function a file object is created which is then used for accessing various methods and functions available for file manipulation.

Syntax of open() function is:-

file_object = open(filename [, access_mode] [,buffering])

open() requires three arguments to work,i.e.,

(I) first one (filename) is the name of the file on secondary storage media, which can be string constant or a variable. The name can include the description of path, in case, the file does not reside in the same folder / directory in which we are working. We will know more about this in later section of chapter.
(II) The second parameter (access_mode) describes how file will be used throughout the program. This is an optional parameter and the default access_mode is reading.
(III) The third parameter (buffering) is for specifying how much is read from the file in one read. The function will return an object of file type using which we will manipulate the file, in our program.

When we work with file(s), a buffer (area in memory where data is temporarily stored before being written to file), is automaticallyassociated with file when we open the file. While writing the content in the file, first it goes to buffer and once the buffer is full, data is written to the file. Also when file is closed, any unsaved data is transferred tofile. flush() function is used to force transfer of data from buffer to file.

FILE ACCESS MODES

r will open the text file for reading only and rb will do the same for binary format file. This is also thedefault mode. The file pointer is placed at the beginning for reading purpose, when we open a file in thismode.

w will open a text file for writing only and wb for binary format file. The file pointer is again placed at the beginning. A non existing file will be created using this mode. Remember if we open an already existing file (i.e. a file containing data) in this mode then the file will be overwritten as the file pointer will be at the beginning for writing in it.

a mode allow us to append data in the text file and ab in binary file. Appending is writing data at the end of the file. In this mode, file pointer is placed at the end in an existing file. It can also be used for creating a file, by opening a non existing file using this mode.

r+ will open a text file and rb+ will open a binary file, for both reading and writing purpose. The file pointer is placed at the beginning of the file when it is opened using r+ / rb+ mode.

w+ opens a file in text format and wb+ in binary format, for both writing and reading. File pointer will be placed at the beginning for writing into it, so an existing file will be overwritten. A new file can also be created using this mode.

a+ opens a text file and ab+ opens a binary file, for both appending and reading. File pointer is placed at the end of the file, in an already existing file. Using this mode a non existing file may be created.

Example usage of open:-

file= open("Sample.txt","r+")

will open a file called Sample.txt for reading and writing purpose. Here the name (by which it exists onsecondary storage media) of the file specified is constant. We can use a variable instead of a constant as name of the file. Sample file, if already exists, then it has to be in the same folder where we are working now, otherwise we have to specify the complete path. It is not mandatory to have file name with extension. In the example .txt extension is used for our convenience of identification. As it is easy to identify the file as text file. Similarly for binary file we will use . dat extension.

Other function, which can be used for creation of a file is file(). Its syntax and its usage is same as open().

Apart from using open() or file() function for creation of file, with statement can also be used for same purpose. Using with ensures that all the resources allocated to file objects gets deallocated automatically once we stop using the file. Its syntax is :

with open() as fileobject :

Example:

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

file manipulation statements

Let's know about other method's and function's which can be used with file object.

fileobject. close() will be used to close the file object, once we have finished working on it. The method will free up all the system resources used by the file, this means that once file is closed, we will not be able to use the file object any more. Before closing the file any material which is not written in file, will be flushed off. So it is good practice to close the file once we have finished using it. In case, if we reassign the file object to some other file, then python will automatically close the file.

Methods for reading data from the file are:

- readline() will return a line read, as a string from the file. First call to function will return first line, second call next line and so on. Remember file object keeps the track of from where reading / writing of data should happen. For readline() a line is terminated by \n (i.e. new line character). The new line character is also read from the file and post-fixed in the string. When end of file is reached, readline() will return an empty string.

  It's syntax is

  fileobject.readline()

  Since the method returs a string it's usage will be

  >>>x = file.readline()

  or

  >>>print file.readline()

  For reading an entire file using readline(), we will have to loop over the file object. This actually is memory efficient, simple and fast way of reading the file. Let's see a simple example of it

  >>>for line in file:

  ... print line

  Same can be achieved using other ways of looping.

- readlines()can be used to read the entire content of the file. You need to be careful while using it w.r.t. size of memory required before using the function. The method will return a list of strings, each separated by \n.

  It's syntax is:

  fileobject.readlines()

  as it returns a list, which can then be used for manipulation.

- read() can be used to read specific size string from file. This function also returns a string read from the file. At the end of the file, again an empty string will be returned.

Syntax of read() function is

fileobject.read([size])

Here size specifies the number of bytes to be read from the file. So the function may be used to read specific quantity of data from the file. If the value of size is not provided or a negative value is specified as size then entire file will be read. Again take care of memory size available before reading the entire content from the file.

Let's see the usage of various functions for reading data from file. Assuming we have a file data.txt containing hello world.\n this is my first file handling program.\n I am using python language.

Example of readlines():

```
>>>lines = []
>>>lines = file.readlines()
```

If we print element of lines (which can be done by iterating the contents of lines) we will get:

```
hello world.
this is my first file handling program.
I am using python language.
```

Can you notice, there are two blank lines in between every string / sentence. Find out the reason for it.

Example of using read():

```
lines = []
content = file.read() # since no size is given, entire file will be read
lines = content.splitlines()
print lines
```

will give you a list of strings:

['hello world.', 'this is my first file handling program.', 'I am using python language.']

For sending data in file, i.e. to create / write in the file, write() and writelines() methods can be used.write() method takes a string (as parameter) and writes it in the file. For storing data with end of line character, you will have to add \n character to end of the string. Notice addition of \n in the end of every sentence while talking of data.txt. As argument to the function has to be string, for storing numeric value, we have to convert it to string.

Its syntax is

fileobject.write(string)

Example

```
>>>f = open('test1.txt','w')
>>>f.write("hello world\n")
>>>f.close()
```

For numeric data value conversion to string is required.

Example

```
>>>x = 52
>>>file.write(str(x))
```

For writing a string at a time, we use write() method, it can't be used for writing a list, tuple etc. into a file.Sequence data type can be written using writelines() method in the file. It's not that, we can't write a string using writelines() method.

It's syntax is:

fileobject.writelines(seq)

So, whenever we have to write a sequence of string / data type, we will use writelines(), instead of write().

Example:

```
f = open('test2.txt','w')
str = 'hello world.\n this is my first file handling program.\n I am using python language"
f.writelines(str)
f.close()
```

let's consider an example of creation and reading of file in interactive mode

```
>>>file = open('test.txt','w')
>>>s = ['this is 1stline','this is 2nd line']
>>>file.writelines(s)
>>>file.close()
>>>file.open('test.txt') # default access mode is r
>>>print file.readline()
>>>file.close()
```

Will display following on screen

this is 1st line this is 2nd line

Let's walk through the code. First we open a file for creation purpose, that's why the access mode is w. Inthe next statement a list of 2 strings is created and written into file in 3rd statement. As we have a list of 2 strings, writelines() is used for the purpose of writing. After writing the data in file, file is closed.

In next set of statements, first one is to open the file for reading purpose. In next statement we are readingthe line from file and displaying it on screen also. Last statement is closing the file.

Although, we have used readline() method to read from the file, which is suppose to return a line i.e. string at a time, but what we get is, both the strings. This is so, because writelines() does not add any EOL.

Character to the end of string. You have to do it. So to resolve this problem, we can create s using following statement

```
s = ['this is 1st line\n', 'this is 2nd line\n']
```

Now using readline(), will result into a string at a time.

All reading and writing functions discussed till now, work sequentially in the file. To access the contents of file randomly - seek and tell methods are used.

- tell() method returns an integer giving the current position of object in the file. The integer returned specifies the number of bytes from the beginning of the file till the current position of file object.

It's syntax is

fileobject.tell()

- fileobject.seek(offset [, from_what])

here offset is used to calculate the position of fileobject in the file in bytes. Offset is added to from_what (reference point) to get the position. Following is the list of from_what values:

Value	Reference point
0	beginning of the file
1	current position of file
2	end of file

Let's read the second word from the test1 file created earlier. First word is 5 alphabets, so we need to move to 5th byte. Offset of first byte starts from zero.

```
f = open('test1.txt','r+')
f.seek(5)
fdata = f.read(5)
print fdata
f.close()
```

will display world on screen.

Let's write a function to create and display a text file using one stream object.

```
def fileHandling():
file = open("story.txt","w+")
while True:
line = raw_input("enter sentence :")
file.write(line)
choice = raw_input("want to enter more data in file Y / N")
if choice.upper() == 'N' : break
file.seek(0)
lines = file.readlines()
file.close()
for l in lines:
print l
```

in this function after opening the file, while loop allow us to store as many strings as we want in the file.

once that is done, using seek() method file object is taken back to first alphabet in the file. From where we read the complete data in list object.

We know that the methods provided in python for writing / reading a file works with string parameters.

So when we want to work on binary file, conversion of data at the time of reading, as well as writing is required. Pickle module can be used to store any kind of object in file as it allows us to store python objects with their structure. So for storing data in binary format, we will use pickle module.

First we need to import the module. It provides two main methods for the purpose, dump and load. For creation of binary file we will use pickle.dump() to write the object in file, which is opened in binary access mode. Syntax of dump() method is:-

dump(object, fileobject)

Example:

```
def fileOperation1():
import pickle
l = [1,2,3,4,5,6]
file = open('list.dat', 'wb') # b in access mode is for binary file
pickle.dump(l,file) # writing content to binary file
file.close()
```

Example:

Example of writing a dictionary in binary file:

```
MD = {'a': 1, 'b': 2, 'c': 3}
file = open('myfile.dat', 'wb')
pickle.dump(MD,file)
file.close()
```

Once data is stored using dump(), it can then be used for reading. For reading data from file we will:

use pickle.load() to read the object from pickle file.

Syntax of load() is :

object = load(fileobject)

Note : we need to call load for each time dump was called.

```
# read python dict back from the file
ifile = open('myfile.dat', 'rb')
MD1 = pickle.load(ifile) # reading data from binary file
ifile.close()
print MD1
```

Results into following on screen:

{'a': 1, 'c': 3, 'b': 2}

To distinguish a data file from pickle file, we may use a different extension of file. .pk / .pickle are commonly used extension for same.

Files are always stored in current folder / directory by default. The os (Operating System) module ofpython provides various methods to work with file and folder / directories. For using these functions, we have to import os module in our program. Some of the useful methods, which can be used with files in os module are as follows:

1. getcwd() to know the name of current working directory.
2. path.abspath(filename) will give us the complete path name of the data file.
3. path.isfile(filename) will check, whether the file exists or not.
4. remove(filename) will delete the file. Here filename has to be the complete path of file.

5. rename(filename1,filename2) will change the name of filename1 with filename2.

Once the file is opened, then using file object, we can derive various information about file. This is doneusing file attributes defined in os module. Some of the attributes defined in it are:-

1. file.closed returns True if file is closed
2. file.mode returns the mode, with which file was opened.
3. file.name returns name of the file associated with file object while opening the file.

We use file object(s) to work with data file, similarly input/output from standard I/O devices is also performed using standard I/O stream object. Since we use high level functions for performing input/output through keyboard and monitor such as - raw_input(), input() and print statement we were not required to explicitly use I/O stream object. But let's learn a bit about these streams also.

The standard streams available in python are :

(I) Standard input stream
(II) Standard output stream and
(III) Standard error stream

These standard streams are nothing but file objects, which get automatically connected to your program'sstandard device(s), when you start python. In order to work with standard I/O stream, we need to import sys module. Methods which are available for I/O operations in it are:-

(I) read() for reading a byte at a time from keyboard
(II) write() for writing data on terminal i.e. monitor

Their usage is

```
import sys
print ' Enter your name :'
name = ''
while True:
c = sys.stdin.read()
if c == '\n' :
break
name = name + c
sys.stdout.write( 'your name is ' + name)
same can be done using high level methods also
name = raw_input('Enter your name :')
print 'your name is ',name
```

As we are through with all basic operations of file handling, we can now learn the various processes which can be performed on file(s) using these operations.

(I) Creating a file

Option 1 : An empty file can be created by using open() statement. The content in the file can be stored later on using append mode.

Option 2 : create a file and simultaneously store / write the content also.

Algorithm

1. Open a file for writing into it
2. Get data to be stored in the file (can be a string at a time or complete data)
3. Write it into the file (if we are working on a string at a time, then multiple writes will be required.)
4. Close the file

Code :

```
def fileCreation():
ofile = open("story.txt","w+")
choice = True
while True:
line = raw_input("enter sentence :")
ofile.write(line)
choice = raw_input("want to enter more data in file Y / N")
if choice == 'N' : break
ofile.close()
```

At the run time following data was provided

```
this is my first file program
writing 2nd line to file
this is last line
```

(II) Traversal for display

Algorithm

1. Open file for reading purpose.
2. Read data from the file (file is sequentially accessed by any of the read methods).
3. Display read data on screen.
4. Continue step 2 & 3 for entire file.
5. Close the file.

Program Code:

```
def fileDisplaying1():
for l in open("story.txt","r").readlines():
print l
file.close()
```

The above code will display

this is my first file programwriting 2nd line to filethis is last line.

(III)Creating a copy of file

Algorithm

1. Open the source file for reading from it.
2. Open a new file for writing purpose
3. Read data from the first file (can be string at a time or complete data of file.)
4. Write the data read in the new file.
5. Close both the files.

Program Code for creating a copy of existing file:

```
def fileCopy():
ifile = open("story.txt","r")
ofile = open("newstory.txt","w")
l = file.readline()
while l:
ofile.write(l)
```

```
l = file.readline()
ifile.close()
ofile.close()
```

Similarly a binary file can also be copied. We don't need to use dump() & load()methods for this. As wejust need to pass byte strings from one file to another.

(IV) Deleting content from the file

It can be handled in two ways

Option 1 (for small file, which can fit into memory i.e. a few Mb's)

Algorithm

1. Open the source file for reading from it.
2. Open a new file for writing purpose
3. Read data from the first file (can be string at a time or complete data of file.)
4. Write the data read in the new file.
5. Close both the files.
6. Open same file for writing into it
7. Write the modified list into file.
8. Close the file.

Program code for deleting second word from story.txt is:

```
def filedel():
    with open('story.txt','r') as file :
        l = file.readlines()
    file.close()
    print l
    del l[1]
    print l
    file.open('story.txt','w')
    file.writelines(l)
    file.close()
```

Similarly we can delete any content, using position of the data. If position is unknown then serach thedesired data and delete the same from the list.

Option 2 (for large files, which will not fit into memory of computer. For this we will need two files)

Algorithm

1. Get the data value to be deleted
2. Open the file for reading purpose
3. Open another (temporary) file for writing into it
4. Read a string (data value) from the file
5. Write it into temporary file, if it was not to be deleted.
6. The process will be repeated for entire file (in case all the occurrence of data are to be deleted)
7. Close both the files.
8. Delete the original file
9. Rename the temporary file to original file name.

Program code for deletion of the line(s) having word (passed as argument) is :

```
import os
def fileDEL(word):
file = open("test.txt","r")
newfile = open("new.txt","w")
while True:
line = file.readline()
if not line :
break
            else :
            if word in line :
                pass
            else :
            print line
            newfile.write(line)
        file.close()
        newfile.close()
    os.remove("test.txt")
    os.rename("new.txt","test.txt")
```

(V) Inserting data in a file

It can happen in two ways. Insertion at the end of file or insertion in the middle of the file.

Option 1 Insertion at the end. This is also known as appending a file.

Algorithm

1. open the file in append access mode.
2. Get the data value for to be inserted in file, from user.
3. Write the data in the file
4. Repeat set 2 & 3 if there is more data to be inserted (appended)
5. Close the file.

Program code for appending data in binary file:-

```
def binAppend():
import pickle
file = open('data.dat','ab')
while True:
y = int(raw_input())
pickle.dump(y,file)
ans = raw_input('want to enter more data Y
/ N')
if ans.upper()== 'N' : break
file.close()
```

Option 2 Inserting in the middle of file

There is no way to insert data in the middle of file. This is a limitation because of OS. To handle such insertions re-writing of file is done.

Algorithm

1. Get the data value to to be inserted with its position.
2. Open the original file in reading mode.
3. Open another (temporary) file for writing in it.
4. Start reading original file sequentially, simultaneously writing it in the temporary file.

5. This is to be repeated till you reach the position of insertion of new data value.
6. Write the new value in temporary file.
7. Repeat step 4 for remaining data of original file.
8. Delete original file
9. Change the name of temporary file to original file.

Code for inserting data in the file with the help of another file

It will be similar to the code used for deletion of content using another file. Here instead of not writing the content add it in the file.

An alternative to this is, first read the complete data from file into a list. Modify the list and rewrite the modified list in the file.

(VI) Updating a file

File updation can be handled in many ways. Some of which are

Option 1 - Truncate write

Algorithm

1. Open the file for reading from it
2. Read the content of file in an object (variable) usually list
3. Close the file
4. Get the details of data to be modified
5. Update the content in the list
6. Re open the file for writing purpose (we know that now opening the file for writing will truncate the existing file).
7. Write the list back to the file.

Program Code for this will be similar to following:

```
with open("sample.txt","r") as file:
content = file.read()
file.close()
content.process()
wit h open ("sample.txt","w") as file :
file.writelines(content)
file.close()
```

Option 2 - Write replace

Algorithm

Open the original file for reading

Open temporary file for writing

Read a line / record from the file

If this was not to be modified copy it in temporary file otherwise copy the modified line / record in the temporary file.

Repeat previous two steps for complete file.

This way of processing a file using python has been handled earlier.

Option 3 - In place updation

Algorithm

1. Open file for reading and writing purpose
2. Get the details of data value to be modified
3. Using linear search, reach to the record / data to be modified
4. Seek to the start of the record
5. Re write the data
6. Close the file.

Updating a text file in this manner is not safe, as any change you make to the file may overwrite thecontent you have not read yet. This should only be used when the text to be replaced is of same size. In place updation of a binary file is not possible. As this requires placing of fileobject to the beginning of the record, calculating size of data in dump file is not possible. So updating a data file using third option is not recommended in python.

Let's create a data file storing students record such as Admission number, Name, Class and Total marks.

Data to be stored contains numeric data, hence will be stored in binary file. We will use dictionary data type to organize this information.

```
from pickle import load, dump
import os
import sys
def bfileCreate(fname):.
l = []
sd = {1000:['anuj',12,450]}
with open(fname,'wb') as ofile :
while True :
dump(sd,ofile)
ans = raw_input("want to enter more data Y
/ N")
if ans.upper() == 'N' : break
x = int(raw_input("enter admission number
of student"))
l = input("enter name class and marks of
student enclosed in []")
sd[x] = l
ofile.close()
def bfileDisplay(fname):
if not os.path.isfile(fname) :
print "file does not exist"
else:
ifile = open(fname,'rb')
try :
while True:
sd = {}
sd = load(ifile)
print sd
except EOFError:
                    pass

ifile.close()
```

Use the code to store records of your class mates. Once the file is created, use bfileDisplay() to see the result. Do you find some problem in the content displayed? Find and resolve the problem? □□□