UNIT V

TOPICS:

Exceptions in Python

Errors in a Python Program: Compile-Time Errors, Runtime Errors, Logical Errors, Exception Handling, Types of Exceptions, the Except Block.

Files in Python:

Types of Files in python, Opening a File, Closing a File, Working with Text Files Containing Strings, the seek () and tell () methods

Exception Handling

A software developer came across a lot of errors either in design of the software or in writing the code. The errors in the software are called 'bugs' and the process of removing them is called 'debugging'.

Errors in python program

In general, we can classify errors in a program into one of these three types:

1. Compile-time errors
2. Run-time errors
3. Logical errors

Compile-time errors

There are syntactical errors found in the code, due to which a program fails to compile. For example, forgetting a colon in the statements like if, while, for, def etc. will result in compile-time error. Such errors are detected by python compiler and the line number along with error description is displayed by the python compiler. Let's see program 1 to understand this better. In this program, we have forgotten to write colon in the if statement, after the condition. This will raise SyntaxError.

Program 1: A python program to understand the compile-time error.

```
#example for compile-time error
x = 1
```

```
if x == 1
print('where is colon?')
```

Output:

```
C:\>python ex.py
  File "ex.py", line 3
    if x == 1
  ^
SyntaxError: invalid syntax
```

Program 2:A python program to demonstrate compile-time error.

```
#another compile time error
x = 10
if x%2==0:
print(x,'is divisible by 2')
print(x,'is even number')
```

Output:

```
C:\>python ex.py
  File "ex.py", line 5
     print(x,'is even number')
     ^
IndentationError: unexpected indent
```

Runtime Errors

When PVM cannot execute the byte code, it flags runtime error. For example, insufficient memory to store something or inability of the PVM to execute some statement come under runtime errors. Runtime errors are not detected by the python compiler. They are detected by the PVM, only at runtime. The following program explains this further:

Program 3:A python program to understand runtime errors.

```
#example for runtime error
def concat(a, b):
print(a+b)
#callconcat() and pass arguments
concat('Hai', 25)
```

Output:

```
C:\>python ex.py
Traceback (most recent call last):
  File "ex.py", line 5, in <module>
concat('Hai', 25)
  File "ex.py", line 3, in concat
  print(a+b)
TypeError: can only concatenate str (not "int") to str
```

Program 4 is also an example for runtime error. In this program, we are creating a list with 4 elements. The indexes (or position numbers) of these elements will be from 0 to 3. When we refer to the index 4 which is not in the list, there will be Index Error during runtime.

Program 4:A python program to demonstrate runtime error.

```
#another runtime error
animal = ['dog','cat','horse','donkey']
print(animal[4])
```

Output:

```
C:\>python ex.py
Traceback (most recent call last):
  File "ex.py", line 3, in <module>
```

```
print(animal[4])
IndexError: list index out of range
```

Logical Errors

These errors depict flaws in the logic of the program. The programmer might be using a wrong formula or the design of the program itself. Logical errors are not detected either by python compiler or PVM. The programmer is solely **responsible**for them. In the following program, the programmer wants to calculate incremented salary of an employee, but he gets wrong output, since he uses wrong formula.

Program 5: A python program to increment the salary of an employee by 15%.

```
#logical error
def increment(sal):
   sal = sal *15/100
return sal
#call increment() and pass salary
sal = increment(5000.00)
print('Incremented Salary: %.2f'%sal)
```

Output:

```
C:\>python ex.py
Incremented Salary: 750.00
```

By comparing the output of a program with manually calculated results, a programmer can guess the presence of a logical error. In program 5, we are using the following formula to calculate the incremented salary:

```
sal =  sal  *   15/100
```

This is wrong since the formula calculates only the increment but it is not adding in the original salary. So, the correct formula would be:

```
sal =  sal  +  sal   * 15/100
```

Compile time errors and logical errors can be eliminated by the programmer by modifying the program source code. In case of runtime errors, when the programmer knows which type of error occurs, he has to handle them using exception handling mechanism. The runtime errors which can be handled by the programmer are called exceptions. Before discussing exception handling mechanism, we will first understand what type of harm and exception handling mechanism, we will first understand what type of harm and exception can cause. Program 6 will help us in this regard.

Program 6:A python program to understand the effect of an exception.

```
#an exception example
#open a file
f = open("myfile",'w')
#do some processing on the file
#accept a, b values store the result  of a/b into the file
a, b = [int(x) for x in input("Enter two numbers: ").split()]
c = a/b
f.write("writing %d into myfile" %c)
#close the file
f.close()
print('File Closed')
```

Output:

```
C:\>python ex.py
Enter two numbers: 10 2
File Closed

C:\>python ex.py
Enter two numbers: >? 10 0
Traceback (most recent call last):
  File "ex.py", line 7, in <module>
   c = a/b
ZeroDivisionError: division by zero
```

Exceptions

An exception is a runtime error which can be handled by the programmer. That means if the programmer can guess an error in the program and he can do something to eliminate the harm caused by that error, then it is called an 'exception'. If the programmer cannot do anything in case of an error, then it is called an 'error' and not an exception.

All exceptions are represented as classes in python. The exceptions which are already available in python are called 'built-in' exceptions. The base class for all built-in exceptions is 'BaseException' class. From BaseException class, the subclass 'Exception' is derived. From Exception class, the subclasses 'StandardError' and 'Warning' we derived.

All errors (or exceptions) are defined as subclasses of StandardError. An error should be compulsorily handled otherwise the program will not execute. Similarly, all warnings are derived as subclasses from 'Warning' class. A warning represents a caution and even though it is not handled, the program will execute. So warnings can be neglected by errors cannot be neglected.

Just like the exceptions which are already available in python language, a programmer can also create his own exceptions, called 'user-defined' exceptions. When programmer wants to create his own exception class, he should derive his class from Exception class and not from 'BaseException' class. In figure we are showing important classes available in Exception hierarchy:

Exception Handling:

The purpose of handling errors is to make the program robust. The word 'robust' means 'strong'. A robust program doesnot terminate in the middle. Also, when there is an error in the program, it will display an appropriate message to the user and continue execution. Designing such programs is needed in any software development. For this purpose, the programmer should handle the errors. When the errors can be handled, they are called exceptions.

To handle exceptions, the programmer should perform the following three steps.

Step 1: The programmer should observe the statements in his program where there may be a possibility of exceptions. Such statements should be written inside a 'try' block. A try block looks like as follows:

```
try:
    statements
```

The greatness of try block is that even if some exception arises inside it, the program will not be terminated. When PVM understands that there is an exception, it jumps into an ‘except’ block.

Step 2: The programmer should write the ‘except’ block where he should display the exception details to the user. This helps the user to understand that there is some error in the program. The programmer should also display a message regarding what can be done to avoid this error. Except block looks like as follows:

```
exceptexceptionname:
    statements     # these statements from handler
```

The statements written inside an except block are called ‘handlers’ since they handle situation when the exception occurs.

Step 3: Lastly, the programmer should perform clean up actions like closing the files and terminating any other process which are running. The programmer should write this code in the finally block. Finally block looks like as follows:

```
finally:
    statements
```

The specially of finally block is that the statements inside the finally block are executed irrespective of whether there is an exception or not. This ensures that all the opened files are properly closed and all the running processes are properly terminated. So, the data in the files will not be corrupted and the user is at the safe-side.

Performing the above 3 tasks is called ‘exception handling’. Remember, in exception handling, as in many cases it is not possible. But the programmer is avoiding any damage that may happen to data and software. Let’s rewrite program 6 handle the ZeroDivisionError exception using try, except and finally blocks.

Program 7:A python program to handle the ZeroDivisonError exception.

```
#an exception handling example
try:
   f = open("myfile","w")
a,b = [int(x) for x in input("Enter two numbers").split()]
   c = a/b
   f.write("wrting %d into myfile" %c)
except ZeroDivisionError:
print('Division By Zero happened')
print('Please do not enter 0 in input')
finally:
   f.close()
print('File Closed')
```

From the preceding output, we can understand that the ‘finally’ block is executed and the file is closed in both the cases, i.e. when there is no exception and when the exception occurred. In the previous discussion, we used try-catch-finally to handle the exception. However, the complete execution handling syntax will be in the following format:

```
try:
statements
except Exception1:
     handler1
except Exception2:
     handler2
else:
statements:
finally:
statements
```

The ‘try’ block contains the statements where there may be one or more exceptions. The subsequent ‘except’ blocks handle these exceptions. When ‘Exception1’ occurs, ‘handle1’ statements are executed. When ‘Exception1’ occurs, ‘handle1’ and so forth. If no exception is raised, the statements inside the ‘else’ block are executed. Even if the exception occurs or does not occur, the code inside ‘finally’ block is always executed. The following points are noteworthy:

1. A single try block can be followed by several except blocks.
2. Multiple except blocks can be used to handle multiple exceptions.
3. We cannot write except blocks without a try block.
4. We can write a try block without any except blocks.
5. Else block and finally blocks are not compulsory.
6. When there is no exception, else block is executed after try block.
7. Finally block is always executed.

Types of Exceptions

There are several exceptions available as part of python language that are called built-in exceptions. In the same way, the programmer can also create his own exceptions called user-defined exceptions. Table summarizes some important built-in exceptions in python. Most of the exception class name ends with the word 'Error'.

Exception Class Name	**Description**
Exception	Represents any type of exception. All exceptions are sub classes of this class.
Arithmetic Error	Represents the base class for arithmetic errors like OverflowError, ZeroDivisionError, FloatingPoint Error
Assertion Error	Raised when an assert statement gives error.
Attribute Error	Raised when an attribute reference or assignment fails.
EOFError	Raised when input() function reaches end of the file condition without reading any data.
FloatingPointError	Raised when a floating point operation fails.
GeneratorExit	Raised when generator's close() method is called.
IOError	Raised when an input or output operation failed. It raises when the file opened is not found or when writing data disk is full.
ImportError	Raised when an import statement fails to find the module being imported.
IndexError	Raised when a sequence index or subscript is out of range.
KeyError	Raised when a mapping (dictionary) key is not found in the set of existing keys.
KeyboardInterrupt	Raised when the user hits the interrupt key (normally Control-C or delete)
NameError	Raised when an identifier is not found locally or globally.
NotImplementedError	Derived from 'RuntimeError'. In user defined base class abstract methods should raise this exception when they require derived classes to override the method.
OverFlowError	Raised when the result of an arithmetic operation is too

	large to be represented. This cannot occur for ling integers(which would rather raise ’MemoryError’.
RuntimeError	Raised when an error is detected that doesn’t fall in any of the other categories.
StopIteration	Raised by an iterator’s next() method to signal that there are no more elements.
SyntaxError	Raised when the compiler encounters a syntax error. Import or exec statements and input() and eval() function may raise this exception.
IndentationError	Raised when indentation is not specified properly.
SystemExit	Raised by the sys.exit() function is applied to an object of inappropriate datatype.
TypeError	Raised when an operation or function is applied to an object of an inappropriate datatype.
UnboundLocalError	Raised when a reference is made to a local variable. In a function or method, but no value has been bound to that variable.
ValueError	Raised when a built-in operation or function receives an argument that has right datatype but wrong value.
ZerDivisonError	Raised when the denominator is zero in a division or modulus operation.

Program 8: A python program to handle syntax error given by eval() function.

```
#example of syntax error
try:
   date = eval(input("Enter date: "))
except SyntaxError:
print('Invalid date entered')
else:
print('you enterd : ',date)
```

Output:

```
C:\>python ex.py
Enter date: 2016, 10, 3
you enterd :  (2016, 10, 3)
```

```
C:\>python ex.py
Enter date: 2016, 10b,3
Invalid date entered
```

Program 9:A python program to handle IOError produced by open() function.

```
#9. example for IOError
#accept a filename
try:
    name = input('Enter filename: ')
    f = open(name, 'r')
except IOError:
print('File not found: ', name)
else:
    n = len(f.readlines())
print(name, 'has',n,'lines')
    f.close()
```

Output:

```
C:\>python ex.py
Enter filename: ex.py
ex.py has 12 lines

Enter filename:  abcd
File not found:  abcd
```

The Except Block

The except block is useful to catch an expression that is raised in the try block. When there is an exception in the try block, then only the except block is executed. It is written in various formats.

1. To catch the exception which is raised in the try block, we can write except block with the exceptions name as:
2. We can catch the exception as an object that contains some description about the exception.
3. To catch multiple exceptions , we can write multiple catch blocks. The other ways is to use a single except block and write all the exceptions as a tuple inside parentheses as:

   except (Exceptionalclass1, Exceptioncla2, ……..):
4. To catch any type of exception where we are not bothered about which type of exception it is, we can write except block without mentioning any

   Exceptionclass name as:
   except:

In the previous program 10, we are catching two exceptions using two except blocks. The same can be written using a single except block as:

```
except (TypeError, ZeroDivisionError):
print('Either TypeError or ZeroDivisionError occurred. ')
```

the other ways is not writing any exception name in except block. This will catch any type of exception, but the programmer cannot determine specifically which exception has occurred. For example,

```
except:
print('some exceptions occurred')
```

in program 11, we are finding inverse of a given number. In this program , we are using try block alone, we need to follow it with a finally block. Since we are not using except block .since we are not except block, it is not possible to catch the exception.

Program 11:A Python program to understand the usage of try with finally blocks.

```
#try without except block
try:
   x = int(input('Enter a number'))
   y = 1/x
finally:
```

```
print("we are not catching the exception.")
print("The inverse is: ",y)
```

Output:

```
C:\>pyton ex.py
Enter a number 5
we are not catching the exception.
The inverse is:  0.2
```

FILES

1. Data is very important. Every organization depends on its data for continuing its business operations. If the data is lost, the organization has to be closed. This is the reason computers are primarily created for handling data, especially for storing and retrieving data. In later days, programs are developed to process the data that is stored in the computer.
2. To store data in a computer, we need files. For example, we can store employee data like employee number, name and salary in a file in the computer and later we use it whenever we want. In computer's view, a file is nothing but collection of data that is available to a program. Once we store data in a computer file, we can retrieve it and use it depending on our requirements. Figure shows the file in our daily life and the file stored in the computer.

There are four important advantages of storing data in a file:

1. When the data is stored in a file, it is stored permanently. This means that even though the computer is switched off, the data is not removed from the memory since the file is stored on hard disk or CD. This file data can be utilized later, whenever required.
2. It is possible to update the file data. For example, we can add new data to the existing file, delete unnecessary datafrom the file and modify the available data of the file. This makes the file more useful.
3. Once the data is stored in a file, the same data can be shared by various programs. For example, once employee data is stored in a file, it can be used in a program to calculate employees' net salaries or in another program to calculate income tax payable by the employees.
4. Files are highly useful to store huge amount of data. For example, voter's list or census data.

Types of files in python

In python, there are two types of files. They are:

1. Text files
2. Binary files

Text files store the data in the form of characters. For example, if we store employee name "Ganesh". It will stored as 6 characters and the employee salary 8900.75 is stored as 7 characters. Normally, text files are used to store characters or strings.

Binary files store entire data in the form of bytes i.e. a group of 8bits each. For example, a character is stored as a byte and an integer is stored in the form of 8 bytes (on a 64 bit machine). When the data is retrieved from the binary file, the programmer can retrieve the data as bytes. Binary files can be used to store text, images , audio and video.

Image files are generally available in .jpg, .gif or .png formats. We cannot use text lines to store images as the images do not contain characters. On the other hand, images contain pixels which are minute dots with which the picture is composed of. Each pixel can be represented by a bit, i.e either 1 or 0. Since these bits can be handled by binary files, we can say that they are highly suitable to store images.

It is very important to know how to create files, store data in the files and retrieve the data from the files in python. To do any operation on files, first of all we should open the files.

Opening a File

Weshould use open() function to open a file. This function accepts 'filename' and 'openmode' in which to open the file.

file handler = open("file name", "open mode", "buffering")

Here, the 'file name' represents a name on which the data is stored. We can use any name to reflect the actual data. For example, we can use 'empdata' as file name to represent the employee data. The file 'open mode' represents the purpose of opening the file. Table 17.1 specifies the file open modes and their meanings.

Table 17.1 The file opening Modes

File open mode	Description
w	To write data into file. If any data is already present in the file, it would be deleted and the present data will be stored.
r	To read data from the file. The file pointer is positioned at the beginning of the file.

a	To append data to the file. Appending means adding at the end of existing data. The file pointer at the end of the file. If the file does not exist, it will create a new file for writing data.
w+	To write and read data of a file. The previous data in the file will be deleted.
r+	To read and write data into a file. the previous data in the file will not be deleted. The file pointer is placed at the beginning of the file.
a+	To append and read data of a file. the file pointer will be at the end of the file exists. If the file does not exist, it creates a new file for reading and writing.
x	To open the file in exclusive creation mode. The file creation fails if file already exists.

A buffer represents a temporary block of memory. 'buffering' is an optional integer used to set the size of the buffer for the file. In the binary mode, we can pass 0 as buffering integer to inform not to use any buffering. In text mode, we can use 1 for buffering to retrieve data from the file one line at a time. Apart from these, we can use any positive integer for buffering. Suppose, we use 500, then a buffer of 500 bytes size is used via which the data is read or written. If we do not mention any buffering integer, then the default buffer size used is 4096 or 8192 bytes.

When the open() function is used to open a file, it returns a pointer to the beginning of the file. This is called 'file handler' or 'file object'. As an example, to open a file for storing data into it, we can write the open() function as:

```
f = open("myfile.txt", "w")
```

here, 'f' represents the file handler or file object. It refers to the file with the name "myfile.txt" that is opened in "w" mode. This means, we can write data into the file but we cannot read data from this file. if this file exists already, then its contents are deleted and the present data is stored into the file.

Closing a File

A file which is opened should be closed using the close() method. Once a file is opened but not closed, then the data of the file may be corrupted or deleted in some cases. Also, if the file is not closed, the memory utilized by the file is not freed, leading to problems like insufficient memory. This happens when we are working with several files simultaneously. Hence it is mandatory to close the file.

```
f.close()
```

here, the file represented by the file object t ‘f’ is closed. It means ‘f’ is deleted from the memory. Once the file object is lost, the file data will become inaccessible. If we want to do any work with the file again, we should once again open the file using the open() function.

In program 1, we are creating a file where we want to store some characters. We know that a group of characters represent a string. After entering a string from the keyboard using input() function, we store the string into the file using write() method as:

```
f.write(str)
```

In this way write() can be used to store a character or a group of characters [string] into a file represented by the file object ‘f’

Program 1: A python program to create a text file to store individual characters.

```
# creating a file to store characters
#open the file for writing data
f = open('myfile.txt','w')
#enter characters from keyboard
str  = input('enter text: ')
#write the string into file
f.write(str)
#closing the file
f.close()
```

Output:

```
c:\>python create.py
enter text: This is my first program
```

Program 2: A python program to read characters from a text file.

```
# reading characters from file
#open the file for reading data
f  = open('myfile.txt','r')
#read all characters from file
str = f.read()
#display them in the screen
print(str)
#closing the file
f.close()
```

Output:

```
c:\>python read.py
This is my first program
```

Working with text files containing strings

To store a group of strings into a text file, we have to use the write() method inside a loop. For example, to store strings into the file as long as the user doesnot type **'@'** symbol, we can write while loop as:

```
while str ! = '@'
        #write the string into file
        If(str  !=  '@'):
                f.write(str+"\n")
```

Please observe the '\n' at the end of the string inside write() method. The write() method writes all the strings sequentially in a single line. To write the strings in different lines, we are supposed to add "\n" character at the end of each string.

Program 3: A python program to store a group of strings into a text file.

```
#create a file with strings
#open the file for writing data
f = open('myfile.txt','w')
#enter strings from keyboard
```

```
print('enter text (@ at end): ')
while str!= '@':
    str = input() #accept string into str
    #write the string into file
if(str !='@'):
        f.write(str+"\n")
#closing the file
f.close()
```

Output:

```
c:\>python create1.py
enter text (@ at end):
This is my file line one
This is line two
@
```

Program 4: A python program to read all the strings from the text file and display them.

```
# reading strings from a file
#open the file for reding data
f = open('myfile.txt', 'r')
#read strings from the file
print('The file contents are: ')
str = f.read()
print(str)
#closing the file
f.close()
```

Output:

```
c:\>python read1.py
The file contents are:
This is my file line one
This is line two
```

Program 5: A python program to append data to an existing file and then displaying the entire file.

```
# appending and then reading strings
#open the file for reading data
f = open('myfile.txt','a+')
print('enter text to append(@ at end): ')
while  str!= '@':
    str  = input() #accept string into str
    #write the string into file
if(str!= '@' ):
        f.write(str+"\n")
#put the file pointer to the beginning of file
f.seek(0, 0)
#read strings from the file
print('the file contents are: ')
str = f.read()
print(str)
#closing the file
f.close()
```

Output:

```
c:\>python append.py
enter text to append(@ at end):
This line is added
@
the file contents are:
This is my file line one
This is line two
This line is added
```