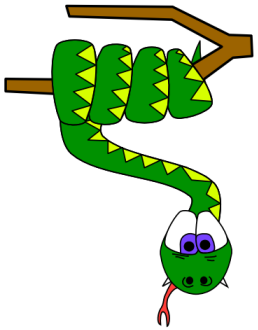

Exception & File Handling

Session-IX

Exception Handling

- Used to handle any unexpected error in Python programs
- Few Standard Exceptions:

Exception Name	Description
Exception	Base class for all exceptions
Arithmetic Error	Base class for all errors that occur for arithmetic calculations
Floating Point Error	Raised when a floating point calculation fails.
Zero Division Error	Raised when division or modulo by zero takes place for all numeric types.
IO Error	Raised when an input / output operation fails, such as print() or open() functions when trying to open a file that does not exist.
Syntax error	Raised when there is a error on Python syntax
Indentation error	Raised when indentation is not specified properly
Value Error	Raised when built-in-function for a data type has a valid type of arguments, but the arguments have invalid values specified
Runtime Error	Raised when a generated error does not fall into any category

Handling Exception

- Exception is an event, which occurs during the execution of program and disrupts the normal flow of program's instructions.
- When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.
- If you have some suspicious code that may raise an exception, you can defend your program by placing the suspicious code in a try: block.
- After the try: block, include an except: statement, followed by a block of code which handles the problem as elegantly as possible.
- Different ways of Exception Handling in Python are:
  - try....except...else
  - try...except
  - try...finally

Handling Exception...

- try…except…else
  - A single try statement can have multiple except statements
  - Useful when we have a try block that may throw different types of exceptions
  - Code in else-block executes if the code in the try: block does not raise an exception

Syntax:

```
try:
   You do your operations here;
   ......................
except ExceptionA:
   If there is ExceptionA, then execute this
block.
except ExceptionB:
   If there is ExceptionB, then execute this
block.
   ......................
else:
   If there is no exception then execute
this block
```

Example:

```
try:
   fh = open("testfile", "w")
   fh.write("This is my test file")
except IOError:
   print ("Error: can\'t find file or read data")
else:
   print ("Written content to file successfully")
   fh.close()
```

Try to open the same file when you do not have write permission, it raises an exception

Output

```
Written content to file successfully
```

Handling Exception...

- try…except..
  - Catches all exceptions that occur
  - It is not considered as good programming practice though it catches all exceptions as it does help the programmer in identifying the root cause of the problem that may occur.

Syntax:

```
try:
You do your operations here;
   ......................
except ExceptionA:
   If there is ExceptionA, then execute this
block.
except ExceptionB:
   If there is ExceptionB, then execute this
block.
   ......................
```

Example:

```
try:
   fh = open("testfile", "w")
   fh.write("This is my test file for exception
handling!!")
except IOError:
   print ("Error: can\'t find file or write data")
```

Try to write to the file when you do not have write permission, it raises an exception

Output

```
Error: can't find file or write data
```

Handling Exception…

Handling an Exception…

- try…finally..
  - finally block is a place to put any code that must execute irrespective of try-block raised an exception or not.
  - else block can be used with finally block

Syntax:

```
try:
   You do your operations here;
   ......................
   Due to any exception, this may be
skipped.
finally:
   This would always be executed.
   ......................
```

Try to write to the file when you do not have write permission

Example:

```
try:
   fh = open("testfile", "w")
   fh.write("This is my test file for exception
handling!!")
finally:
   print("Error: can\'t find file or write data")
```

Output

```
Error: can't find file or write data
```

Opening a File

- Syntax

file object = open(file_name [, access_mode][, buffering])

- file_name – The file_name argument is a string value that contains the name of the file that you want to access.
- access_mode – The access mode determines the mode in which the file has to be opened, i.e., read, write, append, etcThis is an optional parameter and the default file access mode is read (r).
- buffering – If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action is performed with the indicated buffer size. If negative, the buffer size is the system default(default behavior).

File Opening Modes

S.No.	Mode & Description
1	**r** Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
2	**rb** Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
3	**r+** Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
4	**rb+** Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.

Modes...

5	**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.
6	**wb** Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
7	**w+** Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
8	**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, creates a new file for reading and writing.

Modes...

9	**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.
10	**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.
11	**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.
12	**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.

Reading Files

name = open("**filename**","**mode**",**buffering**)

- opens the given file for reading, and returns a file object

name.read() – file's entire contents as a string

name.readline() – next line from file as a string

name.readlines() – file's contents as a list of lines

- the lines from a file object can also be read using a `for` loop

```
>>> f = open("hours.txt")
>>> f.read()
'123 Susan 12.5 8.1 7.6 3.2\n
456 Brad 4.0 11.6 6.5 2.7 12\n
789 Jenn 8.0 8.0 8.0 8.0 7.5\n'
```

File Input Template

- A template for reading files in Python:

```
name = open("filename")
for line in name:
    statements
```

```
>>> input = open("hours.txt")
>>> for line in input:
...     print(line.strip())   # strip()
removes \n
123 Susan 12.5 8.1 7.6 3.2
456 Brad 4.0 11.6 6.5 2.7 12
789 Jenn 8.0 8.0 8.0 8.0 7.5
```

Writing Files

name = open("**filename**", "w")
name = open("**filename**", "a")

- opens file for write (deletes previous contents), or
- opens file for append (new data goes after previous data)

name.write(**str**) – writes the given string to the file

name.close() – saves file once writing is done

```
>>> out = open("output.txt", "w")
>>> out.write("Hello, world!\n")
>>> out.write("How are you?")
>>> out.close()

>>> open("output.txt").read()
'Hello, world!\nHow are you?'
```

File Object Attributes

S.No.	Attribute & Description
1	**file.closed** Returns true if file is closed, false otherwise.
2	**file.mode** Returns access mode with which file was opened.
3	**file.name** Returns name of the file.

File Position

- The *tell()* method tells you the current position within the file; in other words, the next read or write will occur at that many bytes from the beginning of the file.
- The *seek(offset[, from])* method changes the current file position. The **offset** argument indicates the number of bytes to be moved. The **from** argument specifies the reference position from where the bytes are to be moved.
- If *from* is set to 0, the beginning of the file is used as the reference position. If it is set to 1, the current position is used as the reference position. If it is set to 2 then the end of the file would be taken as the reference position.

Renaming & Deleting

```
import os
os.rename('1.txt','hello.txt')
os.remove('hello.txt')
```

File...

```
#opening a File
f=open('demo.txt','w')
f.write('hello world\nhow are you')
f.close()
#reading file
f=open('demo.txt','r')
print(f.read())
f.seek(0)#repositioning cursor
l=f.readlines()
print(l)
f.seek(0)
for line in f.readlines():
    print(line)
f.seek(0)
print(f.readline())
print(f.readline())
print(f.tell())
#file object attribute
print(f.name)
print(f.mode)
print(f.closed)
f.close()
with open('demo.txt','r') as f:
    print(f.read(15))
#no need to close file opened using with
```