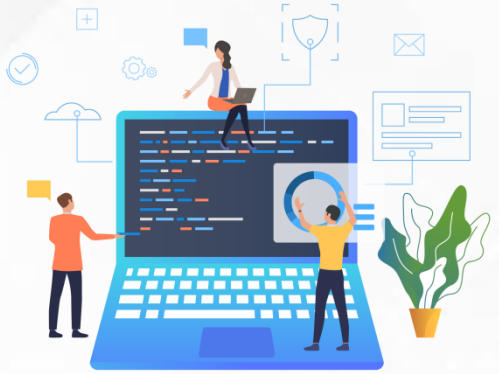
Darshan UNIVERSITY
योग: कर्मसु कौशलम्

Unit-03.1

File IO in Python



Prof. Jayesh D. Vagadiya
Computer Engineering Department
Darshan Institute of Engineering & Technology, Rajkot
Jayesh.vagadiya@darshan.ac.in
9537133260

Outline

- ✓ Introduction
- ✓ Open a file,
- ✓ reading a file
- ✓ writing a file
- ✓ Handling errors using “with” keyword
- ✓ Examples

Introduction

- Sometimes, it is not enough to only display the data on the console.
- The data to be displayed may be very large, and only a limited amount of data can be displayed on the console, and since the memory is volatile, it is impossible to recover the programmatically generated data again and again.
- However, if we need to do so, we may store it onto the local file system which is non-volatile and can be accessed every time.
- File handling in python enables us to create, update, read, the files stored on the local file system through our python program.
- Following are the operation can be performed on files.
  - Creation of the new file
  - Opening an existing file
  - Reading from the file
  - Writing to the file

Darshan UNIVERSITY

Basic IO operations in Python

- Before we can read or write a file, we have to open it using Python's built-in **open()** function.

syntax

```
fileobject = open(filename [, accessmode][, buffering])
```

Parameter Name	Description
Filename	Filename is a name of a file we want to open.
Access mode	Accessmode is determines the mode in which file has to be opened (list of possible values given in next slide)
buffering	If buffering is set to 0, no buffering will happen, if set to 1 line buffering will happen, if grater than 1 then the number of buffer and if negative is given it will follow system default buffering behavior.

open() - Access Mode

M	Description
r	Read only (default)
rb	Read only in binary format
r+	Read and Write both
rb+	Read and Write both in binary format

M	Description (create file if not exist)
w	Write only
wb	Write only in binary format
w+	Read and Write both
wb+	Read and Write both in binary format

M	Description
a	Opens file to append, if file not exist will create it for write
ab	Append in binary format, if file not exist will create it for write
a+	Append, if file not exist it will create for read & write both
ab+	Read and Write both in binary format

Darshan UNIVERSITY
योगः कर्मसु कौशलम्

read()

- The read() method returns the specified number of bytes from the file. Default is -1 which means the whole file.

syntax

```
Fileobject.read([size])
```

Parameter Name	Description
size	Optional. The number of bytes to return. if we don't specify size it will return whole file

readfile.py

```
1 f = open('college.txt')
2 data = f.read()
3 print(data)
```

college.txt

```
Darshan Institute of Engineering and Technology - Rajkot
At Hadala, Rajkot - Morbi Highway,
Gujarat-363650, INDIA
```

OUTPUT

```
Darshan Institute of Engineering and Technology - Rajkot
At Hadala, Rajkot - Morbi Highway,
Gujarat-363650, INDIA
```

Example : Read file in Python

- **readlines()** method will return list of lines from the file.

readlines.py

```
1 f = open('college.txt')
2 lines = f.readlines()
3 print(lines)
```

OUTPUT

```
['Darshan Institute of Engineering and Technology -
Rajkot\n', 'At Hadala, Rajkot - Morbi Highway,\n',
'Gujarat-363650, INDIA']
```

- We can use **for loop** to get each line separately,

readlinesfor.py

```
1 f = open('college.txt')
2 lines = f.readlines()
3 for l in lines :
4     print(l)
```

OUTPUT

```
Darshan Institute of Engineering and Technology - Rajkot

At Hadala, Rajkot - Morbi Highway,

Gujarat-363650, INDIA
```

Darshan UNIVERSITY
योगः कर्मसु कौशलम्

How to write path?

- We can specify relative path in argument to **open** method, alternatively we can also specify absolute path.
- To specify absolute path,
  - In windows, f=open('D:\\folder\\subfolder\\filename.txt')
  - In mac & linux, f=open('/user/folder/subfolder/filename.txt')

- We suppose to **close the file** once we are done using the file in the Python using **close()** method.
- Due to buffering, changes made to a file may not show until you close the file.

closefile.py

```
1 f = open('college.txt')
2 data = f.read()
3 print(data)
4 f.close()
```

write()

- **write()** method will write the specified data to the file.

syntax

```
Fileobject.write(text or byte object)
```

Parameter Name	Description
text or byte object	The text or byte object that will be written.

- If we open file with '**w**' mode it will overwrite the data to the existing file or will create new file if file does not exists.
- If we open file with '**a**' mode it will append the data at the end of the existing file or will create new file if file does not exists.

writelines()

- he writelines() function in Python, used to write multiple lines of text or string to a file at once. The lines are in the form of elements of a list.

syntax

```
Fileobject.writelines(list_or_sequence)
```

Example1.py

```
1  file_handler = open("abc.txt", "w")
2  file_handler.writelines(["Darshan", "University"])
3  file_handler.close()
```

abc.txt

```
Darshan
University
```

Example : Write file in Python

WriteData.py

```
1  f = open("demo.txt",'w')
2  f.write("Hello \nDarshan University")
3  f.close()
4
5  f= open("demo.txt",'r')
6  lines = f.readlines()
7  for l in lines:
8     print(l)
```

OUTPUT

```
Hello
Darshan University
```

demo.txt

```
Hello
Darshan University
```

Handling errors using “with” keyword

- It is used for error handling.
- It is possible that we may have typo in the filename or file we specified is moved/deleted, in such cases there will be an error while running the file.
- To handle such situations we can use new syntax of opening the file using **with** keyword.

fileusingwith.py

```
1 with open('college.txt') as f :
2     data = f.read()
3     print(data)
```

- When we open file using with we **need not** to **close** the file.

Examples

Example1.py

```
1  # WAP to count lines, word, and characters within
   a text file.
2  numwords = 0
3  numchars = 0
4  numlines = 0
5
6  f = open("demo.txt","r")
7  for line in f:
8      wordlist = line.split()
9      numlines += 1
10     numwords += len(wordlist)
11     numchars += len(line)
12
13 print ("Lines: ", numlines)
14 print ("Words: ", numwords)
15 print ("Characters: ", numchars)
```

demo.txt

```
Hello
Darshan University
```

OUTPUT

```
Lines:  2
Words:  3
Characters:  24
```

Examples

Example2.py

```
1  # Write an application that reads a file and counts
   the number of occurrences of word.
2
3  numwords = 0
4
5  word = input("Enter word to search in file = ")
6  f = open("demo.txt","r")
7  for line in f:
8      words = line.split()
9      numwords += words.count(word)
10     print(words)
11
12 print("Count = ",numwords)
```

demo.txt

```
Hello Hello Hello
Darshan University
```

OUTPUT

```
Enter word to search in file = Hello
['Hello', 'Hello', 'Hello']
['Darshan', 'University']
Count =  3
```

Darshan UNIVERSITY

योगः कर्मसु कौशलम्

Tell()

- The method tell returns the current stream position, i.e. the position where we will continue, when we use a "read", "readline" or so on.

Example1.py

```
1  f= open("abc.txt","r")
2  print(f.tell())
3  data = f.read(5)
4  print(data)
5  print(f.tell())
```

demo.txt

```
Darshan
University
```

OUTPUT

```
0
Darsh
5
```

Seek()

- we can move the pointer to an arbitrary place in the file. The method seek takes two parameters:

syntax

```
Fileobject.seek(offset, startpoint_for_offset)
```

Parameter Name	Description
offset	The parameter offset specifies how many positions the pointer will be moved.
startpoint_for_offset	• The question is from which position should the pointer be moved. This position is specified by the second parameter startpoint_for_offset. • 0: reference point is the beginning of the file 1: reference point is the current file position 2: reference point is the end of the file • Default value is 0.

Seek()

Example1.py

```
1  f= open("abc.txt","r")
2  f.seek(5)
3  data = f.read(5)
4  print(data)
5  f.seek(2)
6  data = f.read(5)
7  print(data)
```

demo.txt

```
Darshan
University
```

OUTPUT

```
an
Un
rshan
```

Exercise

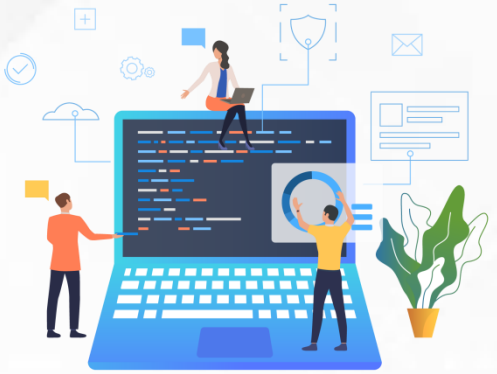
- WAP to print current program it self.
- WAP to copy content of file abc.txt to another file xyz.txt.
- WAP to find the longest word in a file named abc.txt.
- Write a program to replace all “word1” by “word2” from a file1, and output is written to file2 file and display the no. of replacement.

Darshan UNIVERSITY
योगः कर्मसु कौशलम्

Darshan UNIVERSITY
योग: कर्मसु कौशलम्

Unit-03.2

Exception handling in python



Prof. Jayesh D. Vagadiya
Computer Engineering Department
Darshan Institute of Engineering & Technology, Rajkot
Jayesh.vagadiya@darshan.ac.in
9537133260

Outline

- ✓ Introduction
- ✓ Error vs Exception
- ✓ Try
- ✓ Except
- ✓ Else
- ✓ Finally
- ✓ Raise
- ✓ Custom Exception

Errors and Exceptions

- In Python Programming language there are two distinguishable kinds of errors: syntax errors and exceptions in python.
- **Syntax Errors:**
  - Syntax errors, also known as parsing errors, are perhaps the most common kind of complaint you get while you are still learning Python.
  - As the name suggests this error is caused by the wrong syntax in the code.

Example.py

```
1  while True print('Hello world')
```

SyntaxError: invalid syntax

- **Exceptions:**
  - Even if a statement or expression is syntactically correct, it may cause an error when an attempt is made to execute it.
  - Errors detected during execution are called exceptions.

Darshan UNIVERSITY

Exception

- Exceptions are raised when the program is syntactically correct, but the code resulted in an error.

Example.py

```
1  money = 10000
2  # perform division with 0
3  a = money / 0
4  print(a)
```

ZeroDivisionError: division by zero

- Exceptions come in different types, and the type is printed as part of the message. example are ZeroDivisionError, NameError and TypeError.
- Exception is the base class for all the exceptions in Python.
- An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions.
- In general, when a Python script encounters a situation that it cannot cope with, it raises an exception. An exception is a Python object that represents an error.

Darshan UNIVERSITY

Built-in Exceptions

Parameter Name	Description
AssertionError	Raised when an assert statement fails.
AttributeError	Raised when an attribute reference (see Attribute references) or assignment fails.
ImportError	Raised when the import statement has troubles trying to load a module.
IndexError	Raised when a sequence 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). During execution, a check for interrupts is made regularly.
RecursionError	This exception is derived from RuntimeError. It is raised when the interpreter detects that the maximum recursion depth is exceeded.
EOFError	Raised when the input() method hits an "end of file" condition (EOF)
NameError	Raised when a variable does not exist
ZeroDivisionError	Raised when the second operator in a division is zero

Built-in Exceptions

Parameter Name	Description
ValueError	Raised when there is a wrong value in a specified data type
TypeError	Raised when two different types are combined

Handling Exceptions – try and except

- It is possible to write programs that handle selected exceptions.
- Try and except statements are used to catch and handle exceptions in Python.
- Statements that can raise exceptions are kept inside the try clause and the statements that handle the exception are written inside except clause.

syntax

```
try:
        #You do your operations here;
except ExceptionI:
        #If there is ExceptionI, then execute        this code.
```

Example1.py

```
1  money = 10000
2  try:
3          # perform division with 0
4          a = money / 0
5          print(a)
6  except:
7          print("division by zero")
```

OUTPUT

```
division by zero
```

Example2.py

```
1  #Catching Specific Exception
2  try:
3          x = int(input("Please enter a number: "))
           print(x)
4  except ValueError:
5          print("That was no valid number. Try again...")
6
```

OUTPUT

```
Please enter a number: abc
That was no valid number. Try again...
```

Handling Exceptions – try and except

- First, the try clause (the statement(s) between the try and except keywords) is executed.
- If no exception occurs, the except clause is skipped and execution of the try statement is finished.
- If an exception occurs during execution of the try clause, the rest of the clause is skipped. Then, if its type matches the exception named after the except keyword, the except clause is executed, and then execution continues after the try/except block.

syntax

```
try:
        #You do your operations here;
except ExceptionI:
        #If there is ExceptionI, then        execute this
code.
except ExceptionII:
        #If there is ExceptionII, then       execute this
code.
except ExceptionIII:
        #If there is ExceptionIII, then      execute this
code.
```

Example1.py

```
1  #try with mutiple except
2  try:
3          x = int(input("Please enter a number: "))
4          print(x/0)
5  except ValueError:
6                  print("That was no valid number. ")
7  except ZeroDivisionError:
8                  print("division by zero")
```

OUTPUT

```
Please enter a number: 4
division by zero
```

Darshan UNIVERSITY

योगः कर्मसु कौशलम्

The except Clause with Multiple Exceptions

- A try statement may have more than one except clause, to specify handlers for different exceptions. At most one handler will be executed.

syntax

```
try:
        #You do your operations here;
Except(Exception1[,Exception2[,...ExceptionN]]       ]):
        #If there is any exception from the       given exception list, then execute       this       block..
```

Example1.py

```
1  # except Clause with Multiple Exceptions
2  try:
3          x = int(input("Please enter a number: "))
4          print(x/0)
5  except (ValueError,ZeroDivisionError):
6          print("Error. Try again...")
```

OUTPUT

```
Please enter a number: 4
Error. Try again...
```

Darshan UNIVERSITY

Exception's arguments.

- When an exception occurs, it may have associated values, also known as the exception's arguments. The presence and types of the arguments depend on the exception type.
- The except clause may specify a variable after the exception name.
- BaseException is the common base class of all exceptions.
- One of its subclasses, Exception, is the base class of all the non-fatal exceptions.
- Exceptions which are not subclasses of Exception are not typically handled,
- because they are used to indicate that the program should terminate.
- They include SystemExit which is raised by sys.exit() and KeyboardInterrupt which is raised when a user wishes to interrupt the program.

Exception's arguments.

Example1.py

```
1  try:
2          a = 5
3          print(a/0)
4  except ZeroDivisionError as e:
5          print(e.args)
6          print(e.args[0])
7          print(e.__str__())
8          print(type(e))
```

OUTPUT

```
('division by zero',)
division by zero
division by zero
<class 'ZeroDivisionError'>
```

Try with Else Clause

- The try … except statement has an optional else clause.
- when present, must follow all except clauses.
- It is useful for code that must be executed if the try clause does not raise an exception.
- The use of the else clause is better than adding additional code to the try clause because it avoids accidentally catching an exception that wasn't raised by the code being protected by the try …except statement.

syntax

```
try:
        #You do your operations here;
except ExceptionI:
        #If there is ExceptionI, then execute this
        code.
else:
        # execute if no exception
```

Example1.py

```
1  try:
2          f = open("abcc.txt", 'r')
3  except FileNotFoundError:
4          print('cannot open')
5  else:
6          print('has', len(f.readlines()), 'lines')
7          f.close()
```

OUTPUT

```
cannot open
```

Darshan UNIVERSITY

Finally Keyword in Python

- You can use a finally: block along with a try: block.
- The finally block is a place to put any code that must execute, whether the try-block raised an exception or not.

syntax

```
try:
        #You do your operations here;
except ExceptionI:
        #If there is ExceptionI, then execute this
        code.
else:
        # execute if no exception
finally:
        # This would always be executed.
```

Example1.py

```
1  money = 10000
2  try:
3          # perform division with 0
4          a = money / 5
5  except:
6     print("division by zero")
7  else:
8     print(a)
9  finally:
10    print('This is always executed')
```

OUTPUT

```
2000.0
This is always executed
```

Darshan UNIVERSITY

Raising Exceptions

- The raise statement allows the programmer to force a specified exception to occur.
- The sole argument to raise indicates the exception to be raised. This must be either an exception instance or an exception class (a class that derives from BaseException, such as Exception or one of its subclasses).
- If an exception class is passed, it will be implicitly instantiated by calling its constructor with no arguments:

Example1.py

```
1  try:
2          raise ZeroDivisionError
3  except ZeroDivisionError:
4          print("division by zero")
```

OUTPUT

```
division by zero
```

Darshan UNIVERSITY

User-Defined Exceptions

- Programs may name their own exceptions by creating a new exception class.
- Exceptions should typically be derived from the Exception class, either directly or indirectly.

Example1.py

```
1  # define Python user-defined exceptions
2  class NegativeNumberException(Exception):
3          "Raised when the input value is less than 0"
4          def __init__(self, arg):
5                  self.arg = arg
6
7  try:
8     a = int(input("Enter Number"))
9          if a > 0:
10                 print(a)
11         else:
12                 raise NegativeNumberException("Number is negative")
13 except NegativeNumberException as e:
14    print(e.arg)
```

OUTPUT

```
Enter Number-9
Number is negative
```