Unit 1

Python File Input-Output

Opening and closing files, various types of file modes, reading and writing to files, manipulating directories. Iterables, iterators and their problem solving applications

Exception handling

What is an exception, various keywords to handle exceptions such try, catch, except, else, finally, raise. Regular Expressions: Concept of regular expression, various types of regular expressions, using match function.

Unit 2

(using Tkinter/wxPython/Qt) What is GUI, Advantages of GUI and Introduction to GUI library. Layout management, events and bindings, fonts, colours, drawing on canvas (line, oval, rectangle, etc.) Widgets such as : frame, label, button, checkbutton, entry, listbox, message, radiobutton, text, spinbox etc

Unit 3

Database connectivity in Python

Installing mysql connector, accessing connector module, using connect, cursor, execute & close functions, reading single & multiple results of query execution, executing different types of statements, executing transactions, understanding exceptions in database connectivity.

Network connectivity

Socket module, creating server-client programs, sending email, reading from URL

Text books

Paul Gries , Jennifer Campbell, Jason Montojo, Practical Programming: An Introduction to Computer Science Using Python 3, Pragmatic Bookshelf, 2/E 2014

Additional References

James Payne , Beginning Python: Using Python2.6 and Python 3, Wiley India, 2010

A. Lukaszewski, MySQL for Python: Database Access Made Easy, Pact Publisher, 2010

File is a named location on the system storage which records data for later access. It enables persistent storage in a non-volatile memory i.e. Hard disk.

- ✓ The directory can be considered as the address of the file.
- ✓ A file can be stored in different **folders in a drive.**
- ✓ Each folder is separated by a back slash (\). A file name will contain two parts.

  They are file name and extension.
- ✓ File name is a **unique string** used to name the file.
- ✓ The extension is the suffix used after the file name to denote the type of file.

Types Of File in Python:

There are two types of files in Python.

They are:

- Binary file
- Text file

Binary files in Python

Example:

1. **Document files:** .pdf, .doc, .xls
2. **Image files:** .png, .jpg, .gif, .bmp
3. **Video files:** .mp4, .3gp, .mkv, .avi
4. **Audio files:** .mp3, .wav, .mka, .aac
5. **Database files:** .mdb, .accde, .frm, .sqlite
6. **Archive files:** .zip, .rar, .iso, .7z
7. **Executable files:** .exe, .dll, .class

Binary files can be opened in the normal text editor but can't read the content inside the file, because binary files are encoded in the binary format, which can be **understood only by a machine.**

For handling such binary files we need a specific type of software to open it.

Text files in Python

Text files don't have any specific encoding and it can be opened in normal text editor itself.

Example:

- **Web standards:** html, XML, CSS, JSON
- **Source code:** c, app, js, py, java
- **Documents:** txt, tex, RTF
- **Tabular data:** csv, tsv
- **Configuration:** ini, cfg, reg

In Python, file processing takes place in the following order.

- **Open a file that returns a fileobject.**
- **Use the object to perform read or write action.**
- **Close the fileobject**

What is Buffering ?

Buffering is the process of storing a chunk of a file in a temporary memory until the file loads completely. In python there are different values can be given. If the buffering is set to 0 , then the buffering is off. The buffering will be set to 1 when we need to buffer the file.

File open modes in Python

MODES DESCRIPTION

<r> It opens a file in read-only mode while the file offset stays at the root.

<rb> It opens a file in (binary + read-only) modes. And the offset remains at the root level.

<r+> It opens the file in both (read + write) modes while the file offset is again at the root level.

<rb+> It opens the file in (read + write + binary) modes. The file offset is again at the root level.

<w> It allows write-level access to a file. If the file already exists, then it'll get overwritten. It'll create a new file if the same doesn't exist.

<wb> Use it to open a file for writing in binary format. Same behavior as for write-only mode.

<w+> It opens a file in both (read + write) modes. Same behavior as for write-only mode.

<wb+> It opens a file in (read + write + binary) modes. Same behavior as for write-only mode.

<a> It opens the file in append mode. The offset goes to the end of the file. If the file doesn't exist, then it gets created.

<ab> It opens a file in (append + binary) modes. Same behavior as for append mode.

<a+> It opens a file in (append + read) modes. Same behavior as for append mode.

<ab+> It opens a file in (append + read + binary) modes. Same behavior as for append mode. Div B 20th Jan

Operations on File

The operations performed on a file is called the file operation. Following is the list of operations can be applied on a file in python.

1. Opening or Creating a File
2. Writing a File
3. Reading a File
4. Closing a File
5. Renaming a File
6. Deleting a File

Syntax:

file_object = *open*(file_name, “Access Mode”, Buffering)

file Object Attributes

Once a file is opened , a *file* object is created and then various information related to that file can be obtained .

Attribute	Description
file.closed	Returns true if file is closed, false otherwise.
file.mode	Returns access mode with which file was opened.
file.name	Returns name of the file.
file.softspace	Returns false if space explicitly required with print, true otherwise.

Methods for finding File Positions

seek() method

In Python, seek() function is used to change the position of the File Handle to a given specific position.

File handle is like a cursor, which defines **from where the data has to be read or written in the file.**

Syntax: f.seek(offset, from_what), where f is file pointer

Parameters:

Offset: Number of postions to move forward

from_what: It defines point of reference.

The reference point is selected by the **from_what** argument. It accepts three values:

- **0:** sets the reference point at the beginning of the file
- **1:** sets the reference point at the current file position
- **2:** sets the reference point at the end of the file

By default from_what argument is set to 0.

Note: Reference point at current position / end of file cannot be set in text mode except when offset is equal to 0.

Python file method tell() returns the current position of the file read/write pointer within the file.

Syntax:

fileObject.tell()

Parameters:

NA

Return Value:

This method returns the current position of the file read/write pointer within the file.

Renaming and Deleting Files:

- Python **os** module provides methods perform file-processing operations, like renaming and deleting files.
- To use this module we need to import it first

The rename() Method:

- The *rename()* method takes two arguments, the current filename and the new filename.

Syntax:

```
os.rename(current_file_name, new_file_name)
```

Example:

```
import os
os.rename( "abc.txt", "xyz.txt" )
```

The *delete()* Method:

delete() method is used to delete files by supplying the name of the file to be deleted as the argument.

Syntax:

```
os.remove(file_name)
```

Example:

```
import os
os.remove("xyz.txt")
```

Directories in Python:

All files are contained within various directories, and **os** module has several methods to create, remove, and change directories.

The *mkdir()* Method:

mkdir() method of the os module is used to create directories in the current directory.

Syntax:

```
os.mkdir("dirname")
```

Example:

```
import os # Create a directory "CSB"
os.mkdir("CSB")
```

The *getcwd()* Method:

The *getcwd()* method displays the current working directory.

Syntax:

```
os.getcwd()
```

Example:

```
import os
os.getcwd()
```

Iterator

- ✓ An iterator is an object which consist of two methods : __iter__() and __next__() .
- ✓ An iterator is an object representing a stream of data and it returns the data one at a time.
- ✓ A Python iterator supports a method called **__next__() that takes no arguments and always returns the next element** of the stream.
- ✓ If there are no more elements in the stream,__next__() raises the StopIteration exception.

Iterable:

- ✓ Iterable allows looping using for loop.
- ✓ An object is called an iterable if iterator can be derived out of it.
- ✓ Calling **iter()** function on an iterable yields an iterator.
- ✓ Calling **next()** function on iterator gives us the next element.

And at end raises StopIteration exception.

Eg:

```
it_rable=["Python",".NET","Perl","Java","Ruby"]
it_rator=iter(it_rable)
print (it_rator)
#Iterating through iterable using for loop.
for i in it_rable:
    print (i)
print ("\n\n")
#Iterating through iterator using for loop.
for i in it_rator:
    print (i,end=" ")
```

Data types supporting Iterators:

- Text sequence: str
- Sequence: list, tuple, range()
- Mapping- dict
- set

```
#how to convert string to iterable object
str="How are you"
iter_str=iter(str)
print(iter_str)
print(next(iter_str))
print(next(iter_str))
print(type(iter_str))

#similarly generate iterable object from list, tuple, range,
set,dictionary
```

#Convert range(iterable) into iterator

object using iter()

#convert dictionary into iterator object

using iter()

```
#dictionary
student_details={'Manish':1111,'Rajesh':2222,'Rakesh':3333}
#Convert dictionary(iterable) into iterator object using iter()method.
iter_object=iter(student_details)
#display object location and type



#Use the next method for looping  over the keys in the dictionary
print (next(iter_object))
print (next(iter_object))

value_ele=iter(student_details.values())
item_ele=iter(student_details.items())

print(next(value_ele))
print(next(value_ele))

print(next(item_ele))
print(next(item_ele))
```

converting an iterator to list/tuple/dict

- Define list element
- Convert into an iterator object
- Convert this iterator object into tuple using tuple constructor
- Similarly convert into dict using constructor