



File Handling-2

Prof. K. Adishesha

BE, MSc, M.Tech, NET, (Ph.D.)



Learning objectives

- **Understanding Files Cont.,**
- **Binary files**
- **CSV (Comma separated values) files**
- **Opening and Closing Files**
- **Reading and Writing Files**
- **Using Pickle module**





Types of Files

- The data files are the files that store data pertaining to a specific application, for later use.
- Python allow us to create and manage three types of file

1. TEXT FILE

2. BINARY FILE

3. CSV (Comma separated values) files





BINARY FILE

What is Binary File?

- A binary file contains arbitrary binary data i.e. numbers stored in the file, can be used for numerical operation(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.
- 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.





DIFFERENCE BETWEEN TEXT FILES AND BINARY FILES

Text Files	Binary Files
Text Files are sequential files	A Binary file contain arbitrary binary data
Text files only stores texts	Binary Files are used to store binary data such as image, video, audio, text
There is a delimiter EOL (End of Line \n)	There is no delimiter
Due to delimiter text files takes more time to process. while reading or writing operations are performed on file.	No presence of delimiter makes files to process fast while reading or writing operations are performed on file.
Text files easy to understand because these files are in human readable form	Binary files are difficult to understand
Text files are having extension .txt	Binary files are having .dat extension
Programming on text files are very easy.	Programming on binary files are difficult
Less prone to get corrupt as changes reflect as soon as the file is opened and can easily be undone	Can easily get corrupted, even a single bit change may corrupt the file.





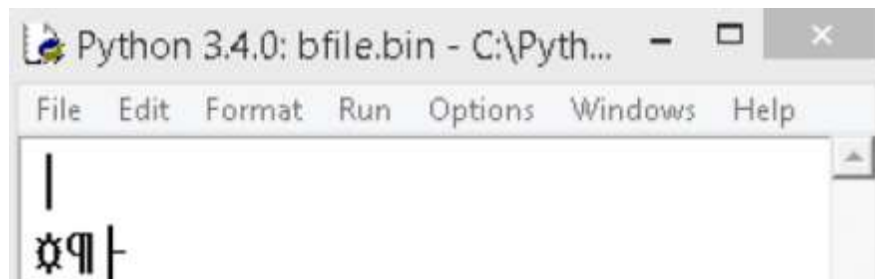
BINARY FILES

CREATING BINARY FILES

```
def binary_file_create():  
    f=open("bfile.bin","wb")  
    num=[5, 10, 15, 20, 25]  
    arr=bytearray(num)  
    f.write(arr)  
    f.close()  
    print("Binary file created")  
    f.close()  
binary_file_create()
```

SEEING CONTENT OF BINARY FILE

Content of binary file which is in codes.





Python File Open

- The key function for working with files in Python is the `open()` function.
- The `open()` function takes two parameters; filename, and mode.
- There are four different methods (modes) for opening a file:
 - **"r"** - Read - Default value. Opens a file for reading, error if the file does not exist
 - **"a"** - Append - Opens a file for appending, creates the file if it does not exist
 - **"w"** - Write - Opens a file for writing, creates the file if it does not exist
 - **"x"** - Create - Creates the specified file, returns an error if the file exists
- In addition you can specify if the file should be handled as binary or text mode
 - **"t"** - Text - Default value. Text mode
 - **"b"** - Binary - Binary mode (e.g. images)
- These are the default modes. The file pointer is placed at the beginning for reading purpose, when we open a file in this mode.





FILE ACCESS MODES

- A file mode governs the type of operations like read/write/append possible methods in the opened file.

MODE *Operations on File Opens in*

- **r+** Text File Read & Write Mode
 - **rb+** Binary File Read Write Mode
 - **w** Text file write mode
 - **wb** Text and Binary File Write Mode
 - **w+** Text File Read and Write Mode
 - **wb+** Text and Binary File Read and Write Mode
 - **a** Appends text file at the end of file, a file is created not exists.
 - **ab** Appends both text and binary files at the end of file
 - **a+** Text file appending and reading.
 - **ab+** Text and Binary file for appending and reading.
- Example: **f=open("tests.dat", 'ab+')**
tests.dat is binary file and is opened in both modes that is reading and appending.





RANDOM ACCESS METHODS

- All reading and writing functions discussed till now, work sequentially in the file.
- To access the contents of file randomly –following methods.

1. seek method

2. tell method





RANDOM ACCESS METHODS

Seek() method :

- seek() method can be used to position the file object at particular place in the file.

syntax is :

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.
- Value reference point:
 - 0** -beginning of the file
 - 1** -current position of file
 - 2** -end of file
- Default value of from_what is **0**, i.e. beginning of the file.
Example: **f.seek(7)**
 - keeps file pointer at reads the file content from 8th position onwards to till EOF.





RANDOM ACCESS METHODS

tell method

- 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.

Syntax:

fileobject.tell()

- tell() method returns an integer and assigned to pos variable. It is the current position from the beginning of file.

```
def File_Tell_Fun():  
    f=open("test1.txt",'r')  
    f.seek(7)  
    f_data=f.read(1)  
    pos=f.tell()  
    print("content of f_data is = ", f_data)  
    print("the current position is = ",pos)  
    f.close()  
File_Tell_Fun()
```

```
content of f_data is = P  
the current position is = 8
```





Closing Files

- **close()**- 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.
- `fileobject.close()` will be used to close the file object, once we have finished working on it.
- Syntax
 - `<fileHandle>.close()`
- For example:
 - `fout.close()`

Note: You should always close your files, in some cases, due to buffering, changes made to a file may not show until you close the file.





PYTHON FILE OBJECT ATTRIBUTES

- File attributes give information about the file and file state.

Attribute	Function
name	Returns the name of the file
closed	Returns true if file is closed. False otherwise.
mode	The mode in which file is open.
softspace	Returns a Boolean that indicates whether a space character needs to be printed before another value when using the print statement.





PICKLING AND UNPICKLING USING PICKLE MODULE

- Use the python module pickle for structured data such as list or directory to a file.
- PICKLING refers to the process of converting the structure to a byte stream before writing to a file.
- while reading the contents of the file, a reverse process called UNPICKLING is used to convert the byte stream back to the original structure.
- First we need to import the module, It provides two main methods for the purpose:-

1) dump() method

2) load() method





pickle.dump() Method

- Use pickle.dump() method to write the object in file which is opened in binary access mode.

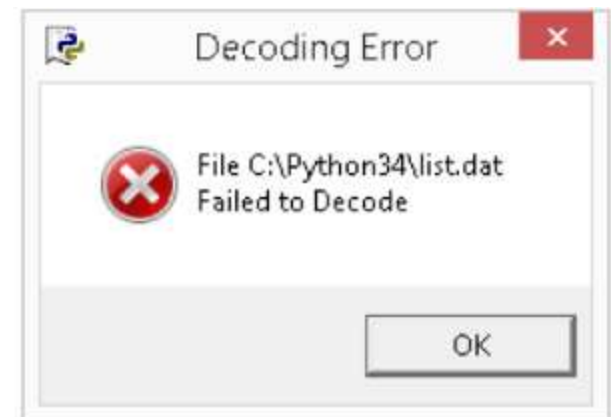
Syntax of dump method is:

dump(object,fileobject)

- A program to write list sequence in a binary file using pickle.dump() method

```
def bin_create():  
    import pickle  
    list1=[40,50,67,39,42,95,897,647,125]  
    f=open("list.dat",'wb')  
    pickle.dump(list1,f)  
    print("Information added to Binary File")  
    f.close()  
bin_create()
```

- Once you try to open list.dat file in python editor to see the content python generates decoding error.



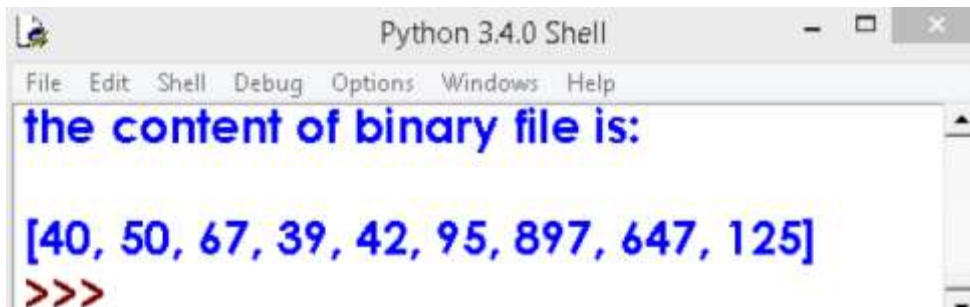


pickle.load() Method

- pickle.load() method is used to read the binary file.

```
def bin_read_file():  
    import pickle  
    f=open("list.dat",'rb')  
    list1=pickle.load(f)  
    print("the content of binary file is:\n")  
    print(list1)  
    f.close()  
bin_read_file()
```

- Once you try to open list.dat file in python editor to see the content python



```
Python 3.4.0 Shell  
File Edit Shell Debug Options Windows Help  
the content of binary file is:  
  
[40, 50, 67, 39, 42, 95, 897, 647, 125]  
>>>
```

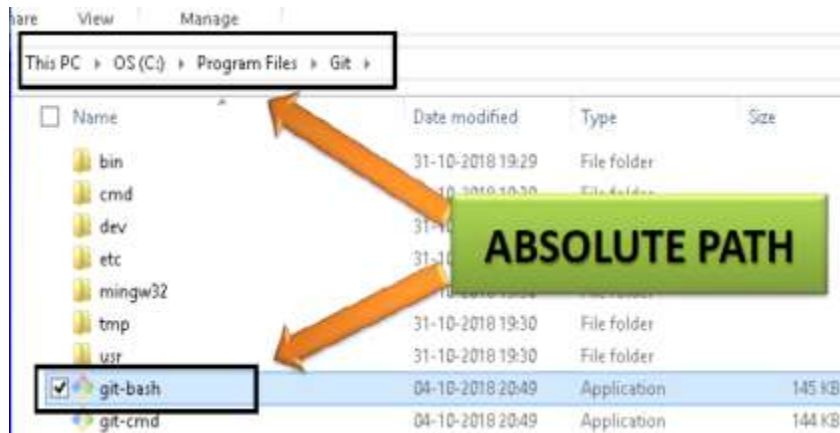




HANDLING FILES THROUGH OS MODULE

- The os module of Python allows you to perform Operating System dependent operations such as making a folder, listing contents of a folder, know about a process, end a process etc..
- Let's see some useful os module methods that can help you to handle files and folders in your program.
 - **ABSOLUTE PATH**
 - **RELATIVE PATH**

Absolute path of file is file location, where in it starts from the top most directory



Relative Path of file is file location, where in it starts from the current working directory





HANDLING FILES THROUGH OS MODULE

Method	Function
os.makedirs()	Create a new folder
os.listdir()	List the contents of a folder
os.getcwd()	Show current working directory
os.path.getsize()	show file size in bytes of file passed in parameter
os.path.isfile()	Is passed parameter a file
os.path.isdir()	Is passed parameter a folder
os.chdir	Change directory/folder
os.rename(current,new)	Rename a file
os.remove(file_name)	Delete a file





CSV files

What is CSV File?

- A comma-separated values (CSV) file is a delimited text file that uses a comma to separate values.
- Each line of the file is a data record.
- Each record consists of one or more fields, separated by commas.
- The use of the comma as a field separator is the source of the name for this file format.
- CSV file is used to transfer data from one application to another.
- CSV file stores data, both numbers and text in a plain text.



CSV File Reading and Writing

- CSV (Comma Separated Values) format is the most common import and export format for spreadsheets and databases.
- The lack of a well-defined standard means that subtle differences often exist in the data produced and consumed by different applications.
- These differences can make it annoying to process CSV files from multiple sources.
- CSV module implements classes to read and write tabular data in CSV format.



CSV File Reading and Writing

- The CSV module's reader and writer objects read and write sequences.
 - `csv.reader(csvfile, dialect='excel', **fmtparams)`
 - `csv.writer(csvfile, dialect='excel', **fmtparams)`
- Programmers can also read and write data in dictionary form using the DictReader and DictWriter classes.

```
>>> import csv
>>> with open('names.csv', newline='') as csvfile:
...     reader = csv.DictReader(csvfile)
...     for row in reader:
...         print(row['first_name'], row['last_name'])
```





Conclusion!

Thank you

