VELAMMAL MAT. HR. SEC. SCHOOL, SURAPET, CHENNAI- 66

STD: XII (COMPUTER SCIENCE)

CHAPTER- 13 – PYTHON AND CSV FILES

PART - II

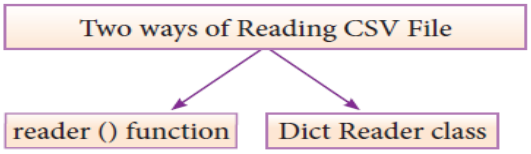
1. What is CSV File?

- CSV stands for Comma Separated Values
- A CSV fi le is a human readable text fi le where each line has a number of fields, separated by commas or some other delimiter.

2. Mention the two ways to read a CSV file using python.

There are two ways to read a CSV file.

1. Use the csv module's reader function
2. Use the DictReader class.



3. Mention the default modes of the File.

Mention the default modes of the File.

- The default mode of csv file in reading and writing is **text mode**

4. What is the use of next() function?

- **"next()" command** is used to avoid or skip the first row or row heading.
- While sorting, the row heading is also get sorted, to avoid that the first row should be skipped.
- This is can be done by using the command "next()".
- The list is sorted and displayed.
- Example: next(reader) # skipping first row(Heading)

5. How will you sort more than one column from a csv file? Give an example statement.

- To sort by more than one column you can use **itemgetter** with multiple indices.

Syntax: **operator.itemgetter(col_no)**

Example: sortedlist = sorted (data, key=operator.itemgetter(1))

6. Write difference between writerow() and writerows() method.

writerow()	writerows()
The **writerow()** method writes one row at a time.	If you need to write all the data at once you can use **writerows()** method.
writerow() takes 1-dimensional data (one row)	writerows takes 2-dimensional data (multiple rows) to write in a file.

PART – III

1. Write a note on open() function of python. What is the difference between the two methods?

- **Python has a built-in function open() to open a file.** This function returns a **file object,** also called a **handle,** as it is used to read or modify the file accordingly.
- **The default is reading in text mode.** In this mode, while reading from the file the data would be in the format of **strings.**
- On the other hand, **binary mode returns bytes and this is the mode to be used when dealing with non-text files like image or exe files.**
- For Example:

```
f = open("test.txt")
f.close()
```

- The above method is **not entirely safe.** If an **exception** occurs when you are performing some operation with the file, the code exits without closing the file.
- The best way to do this is using the **"with"** statement. This ensures that the file is closed when the block inside **with** is exited. You need not to explicitly call the close() method. It is done internally.

```
with open("test.txt",'r') as f:
```

2. Write a Python program to modify an existing file.

```
import csv
row = ['3', 'Meena','Bangalore']
with open('student.csv', 'r') as readFile:
        reader = csv.reader(readFile)
        lines = list(reader)
lines[3] = row
with open('student.csv', 'w') as writeFile:
        writer = csv.writer(writeFile)
        writer.writerows(lines)
readFile.close()
writeFile.close()
```

3. Write a Python program to read a CSV file with default delimiter comma(,)

```
#importing csv
import csv
with open('c:\pyprg\sample1.csv', 'r') as F:
        reader = csv.reader(F)
for row in reader:
        print(row)
F.close()
```

OUTPUT

```
['SNO', 'NAME', 'CITY']
['12101', 'RAM', 'CHENNAI']
['12102', 'LAVANYA', 'TIRUCHY']
['12103', 'LAKSHMAN', 'MADURAI']
```

4. What is the difference between the write mode and append mode

Write Mode	Append Mode
• 'w' - Open a file for writing.	• 'a' - Open for appending at the end of the file without truncating it.

• Creates a new file if it does not exist or truncates the file if it exists.	• Creates a new file if it does not exist.

5. What is the difference between reader() and DictReader() function?

- csv. reader and csv.writer work with **list/tuple,** while csv.DictReader and csv.DictWriter work with dictionary.
- csv.DictReader and csv.DictWriter take additional argument fieldnames that are used as dictionary keys.
- DictReader() class of csv module creates an object which maps data to a dictionary

PART - IV

1. Difference between EXCEL FILE and CSV file

EXCEL	CSV
Excel is a binary file that holds information about all the worksheets in a file, including both content and formatting	CSV format is a plain text format with a series of values separated by commas
XLS files can only be read by applications that have been especially written to read their format, and can only be written in the same way.	CSV can be opened with any text editor in Windows like notepad, MS Excel, OpenOffice, etc.
Excel is a spreadsheet that saves files into its own proprietary format viz. xls or xlsx	CSV is a format for saving tabular information into a delimited text file with extension .csv
Excel consumes more memory while importing data	Importing CSV files can be much faster, and it also consumes less memory

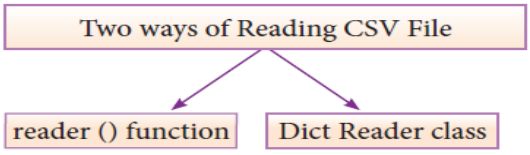
2. Tabulate the different mode with its meaning. (OR) Different mode with its meaning - Python File Modes

Mode	Description
'r'	Open a file for reading. (default)
'w'	Open a file for writing. Creates a new file if it does not exist or truncates the file if it exists.
'x'	Open a file for exclusive creation. If the file already exists, the operation fails.
'a'	Open for appending at the end of the file without truncating it. Creates a new file if it does not exist.
't'	Open in text mode. (default)
'b'	Open in binary mode
'+'	Open a file for updating (reading and writing)

3. Write the different methods to read a File in Python.

There are two ways to read a CSV file.

1. Use the csv module's reader function
2. Use the DictReader class.



CSV Module's Reader Function

- You can read the contents of CSV file with the help of **csv.reader()** method.
- **The reader function is designed to take each line of the file and make a list of all columns.**
- **The syntax for csv.reader() is**

  csv.reader(fileobject,delimiter,fmtparams)

  where

  file object :- passes the path and the mode of the file

  delimiter :- an optional parameter containing the standard dilects like , | etc can be omitted

  fmtparams: optional parameter which help to override the default values of the dialects like skipinitialspace,quoting etc. Can be omitted

  EXAMPLE:

```
import csv
with open('c:\pyprg\sample1.csv', 'r') as F:
        reader = csv.reader(F)
for row in reader:
        print(row)
F.close()
```

  OUTPUT

```
['SNO', 'NAME', 'CITY']
['12101', 'RAM', 'CHENNAI']
['12102', 'LAVANYA', 'TIRUCHY']
['12103', 'LAKSHMAN', 'MADURAI']
```

Reading CSV File Into A Dictionary

- To read a CSV file into a dictionary can be done by using **DictReader** class of csv module which works similar to the reader() class but creates an object which maps data to a dictionary.
- The keys are given by the fieldnames as parameter.
- **DictReader** works by reading the first line of the CSV and using each comma separated value in this line as a **dictionary key.**
- The columns in each subsequent row then behave like dictionary values and can be accessed with the appropriate key (i.e. fieldname).
- Example Program:

```
import csv
filename = 'c:\\pyprg\\sample8.csv'
input_file =csv.DictReader(open(filename,'r'))
for row in input_file:
        print(dict(row))
```

  OUTPUT:

```
{'ItemName ': 'Keyboard ', 'Quantity': '48'}
{'ItemName ': 'Monitor', 'Quantity': '52'}
{'ItemName ': 'Mouse ', 'Quantity': '20'}
```

4. Write a Python program to write a CSV File with custom quotes

```
import csv
csvData = [['SNO','Items'], ['1','Pen'], ['2','Book'], ['3','Pencil']]
```

```
csv.register_dialect('myDialect',delimiter = '|',quotechar = '"',quoting=csv.QUOTE_ALL)
with open('c:\pyprg\ch13\quote.csv', 'w') as csvFile:
        writer = csv.writer(csvFile, dialect='myDialect')
        writer.writerows(csvData)
print("writing completed")
csvFile.close()
```

5. Write the Rules to be followed to format the data in a CSV file

- Each record (row of data) is to be located on a separate line, delimited by a line break by pressing enter key. For example:

  xxx,yyy and enter Key to be pressed
- The last record in the file may or may not have an ending line break. For example:

  ppp, qqq
  yyy, xxx
- There may be an optional header line appearing as the first line of the file with the same format as normal record lines. The header will contain names corresponding to the fields in the file and should contain the same number of fields as the records in the rest of the file. For example: field_name1,field_name2,field_name3

  aaa,bbb,ccc
  zzz,yyy,xxx CRLF**(Carriage Return and Line Feed)**
- Within the header and each record, there may be one or more fields, separated by commas. Spaces are considered part of a field and should not be ignored. The last field in the record must not be followed by a comma. For example: Red , Blue
- Each field may or may not be enclosed in double quotes. If fields are not enclosed with double quotes, then double quotes may not appear inside the fields. For example:

  "Red","Blue","Green" #Field data with doule quotes
  Black,White,Yellow #Field data without doule quotes
- Fields containing line breaks (CRLF), double quotes, and commas should be enclosed in double-quotes. For example:

  Red, ",", Blue CRLF # comma itself is a field value.so it is enclosed with double quotes
  Red, Blue , Geen
- If double-quotes are used to enclose fields, then a double-quote appearing inside a field must be preceded with another double quote. For example:

  "Red, " "Blue", "Green", # since double quotes is a field value it is enclosed with another double quotes
  , , White

EXTRA QUESTION ANSWER:

Purpose Of CSV File

- **CSV** is a simple **file format** used to store tabular data, such as a spreadsheet or database.
- Since they're plain text, they're easier to import into a spreadsheet or another storage database, regardless of the specific software you're using.
- You can open CSV files in a spreadsheet program like Microsoft Excel or in a text editor or through a database which make them easier to read.

Creating CSV Normal File

- To create a CSV file in Notepad, First open a new file using
- **File →New or ctrl +N.**
- Then enter the data you want the file to contain, separating each value with a comma and each row with a new line.
- For example consider the following details

  Topic1,Topic2,Topic3
  one,two,three
  Example1,Example2,Example3
- Save this content in a file with the extension .csv.

CSV files- data with Spaces at the beginning

- "sample2.csv" containing the following data when opened through notepad

Topic1,	Topic2,	Topic3,
one,	two,	three
Example1,	Example2,	Example3

```
import csv
csv.register_dialect('myDialect',delimiter = ',',skipinitialspace=True)
F=open('c:\\pyprg\\sample2.csv','r')
reader = csv.reader(F, dialect='myDialect')
for row in reader:
    print(row)
F.close()
```

OUTPUT :

```
['Topic1', 'Topic2', 'Topic3']
['one', 'two', 'three']
['Example1', 'Example2', 'Example3']
```

CSV File-Data With Quotes

Here, we have quotes.csv file with following data.

```
SNO,Quotes
1, "The secret to getting ahead is getting started."
2, "Excellence is a continuous process and not an accident."
3, "Work hard dream big never give up and believe yourself."
4, "Failure is the opportunity to begin again more intelligently."
5, "The successful warrior is the average man, with laser-like focus."
```

- Program read "quotes.csv" file, where delimiter is comma (,) but the quotes are within quotes (" ").

```
import csv
csv.register_dialect('myDialect',delimiter = ',',quoting=csv.QUOTE_ALL, skipinitialspace=True)
f=open('c:\\pyprg\\quotes.csv','r')
reader = csv.reader(f, dialect='myDialect')
for row in reader:
    print(row)
```

OUTPUT:

```
['SNO', 'Quotes']
```

['1', 'The secret to getting ahead is getting started.']
['2', 'Excellence is a continuous process and not an accident.']
['3', 'Work hard dream big never give up and believe yourself.']
['4', 'Failure is the opportunity to begin again more intelligently.']
['5', 'The successful warrior is the average man, with laser-like focus. ']

CSV files with Custom Delimiters

```
import csv
csv.register_dialect('myDialect', delimiter = '|')
with open('c:\\pyprg\\sample4.csv', 'r') as f:
reader = csv.reader(f, dialect='myDialect')
for row in reader:
    print(row)
f.close()
```

OUTPUT:

```
['RollNo', 'Name', 'City']
['12101', 'Arun', 'Chennai']
['12102', 'Meena', 'Kovai']
['12103', 'Ram', 'Nellai']
```

Read a specific column In a File

```
import csv
f=open("c:\\pyprg\\sample5.csv",'r')
readFile=csv.reader(f)
for col in readFile :
    print col[0],col[3]
f.close()
```

OUTPUT:

```
Item Name Profit
Keyboard 1152
Monitor 10400
Mouse 2000
```

Read A CSV File And Store It In A List

```
import csv
inFile= 'c:\\pyprg\\sample.csv'
F=open(inFile,'r')
reader = csv.reader(F)
arrayValue = []
for row in reader:
    arrayValue.append(row)
    print(row)
F.close()
```

OUTPUT:

```
['Topic1', 'Topic2', 'Topic3']
```

[' one', 'two', 'three']
['Example1', 'Example2', 'Example3']

Read A CSV File And Store A Column Value In A List For Sorting

- Since the row heading is also get sorted, to avoid that the first row should be skipped. This is can be done by using the command “next()”. The list is sorted and displayed.

```
import csv
inFile= ‘c:\\pyprg\\sample6.csv’
F=open(inFile,’r’)
reader = csv.reader(F)
next(reader)
arrayValue = []
a = int(input (“Enter the column number 1 to 3:-“))
for row in reader:
arrayValue.append(row[a])
arrayValue.sort()
for row in arrayValue:
    print (row)
F.close()
```

OUTPUT:

```
Enter the column number 1 to 3:-
50
12
10
```

Sorting A CSV File With A Specified Column

- The list of rows is sorted and displayed in ascending order of quantity. To sort by more than one column you can use itemgetter with multiple indices: operator .itemgetter (1,2).

```
import csv ,operator
data = csv.reader(open(‘c:\\PYPRG\\sample8.csv’))
next(data)
sortedlist = sorted (data, key=operator.itemgetter(1))
for row in sortedlist:
    print(row)
```

OUTPUT:

```
[‘Mouse ‘, ‘20’]
[‘Keyboard ‘, ‘48’]
[‘Monitor’, ‘52’]
```

Reading CSV File With User Defined Delimiter Into A Dictionary

```
import csv
csv.register_dialect(‘myDialect’,delimiter = ‘|’,skipinitialspace=True)
filename = ‘c:\\pyprg\\ch13\\sample8.csv’
with open(filename, ‘r’) as csvfile:
```

```
    reader = csv.DictReader(csvfile, dialect='myDialect')
    for row in reader:
        print(dict(row))
csvfile.close()
```

OUTPUT:

```
{'ItemName ,Quantity': 'Keyboard ,48'}
{'ItemName ,Quantity': 'Monitor,52'}
{'ItemName ,Quantity': 'Mouse ,20'}
```

Writing Data Into Different Types in CSV Files

- Creating A New Normal CSV File
- Modifying An Existing File
- Writing On A CSV File with Quotes
- Writing On A CSV File with Custom Delimiters
- Writing On A CSV File with Lineterminator
- Writing On A CSV File with Quotechars
- Writing CSV File Into A Dictionary
- Getting Data At Runtime And Writing In a File

Creating A New Normal CSV File

- The csv.writer() method returns a writer object which converts the user's data into delimited strings on the given file-like object.
- The writerow() method writes a row of data into the specified file.
- **The syntax for csv.writer() is**
  csv.writer(fileobject,delimiter,fmtparams)
  where

  fileobject: Passes the path and the mode of the file

  delimiter: an optional parameter containing the standard dilects like , | etc can be omitted

  fmtparams: optional parameter which help to override the default values of the dialects like skipinitialspace, quoting etc. can be omitted.

Example:

```
import csv
csvData = [['Student', 'Age'], ['Dhanush', '17'], ['Kalyani', '18'], ['Ram', '15']]
with open('c:\\pyprg\\Pupil.csv', 'w') as CF:
    writer = csv.writer(CF)
    writer.writerows(csvData)
CF.close()
```

ADDING NEW ROW:

```
import csv
row = ['6', 'Sajini ', 'Madurai']
with open('student.csv', 'a') as CF:
writer = csv.writer(CF)
```

```
writer.writerow(row)
CF.close()
```

Roll No	Name	City
1	Harshini,	Chennai
2	Adhith,	Mumbai
3	Meena	Bengaluru
4	egiste,	Tiruchy
5	Venkat,	Madurai
6	Sajini ,	Madurai

CSV Files With Quotes

```
import csv
info = [['SNO', 'Person', 'DOB'],
['1', 'Madhu', '18/12/2001'],
['2', 'Sowmya','19/2/1998'],
['3', 'Sangeetha','20/3/1999'],
['4', 'Eshwar', '21/4/2000'],
['5', 'Anand', '22/5/2001']]
csv.register_dialect('myDialect',quoting=csv.QUOTE_ALL)
with open('c:\\pyprg\\ch13\\person.csv', 'w') as f:
    writer = csv.writer(f, dialect='myDialect')
    for row in info:
        writer.writerow(row)
f.close()
```

OUTPUT:

```
"SNO","Person","DOB" "1","Madhu","18/12/2001" "2","Sowmya","19/2/1998"
    "3","Sangeetha","20/3/1999" "4","Eshwar","21/4/2000"
"5","Anand","22/5/2001"
```

CSV Files With Custom Delimiters:

```
import csv
info = [['SNO', 'Person', 'DOB'],
['1', 'Madhu', '18/12/2001'],
['2', 'Sowmya','19/2/1998'],
['3', 'Sangeetha','20/3/1999'],
['4', 'Eshwar', '21/4/2000'],
['5', 'Anand', '22/5/2001']]
csv.register_dialect('myDialect',delimiter = '|')
with open('c:\pyprg\ch13\dob.csv', 'w') as f:
    writer = csv.writer(f, dialect='myDialect')
    for row in info:
        writer.writerow(row)
f.close()
```

OUTPUT:

Roll No	Name	City
1	Madhu	18/12/2001
2	Sowmya	19/2/1998
3	Sangeetha	20/3/1999
4	Eshwar	21/4/2000
5	Anand,	22/5/2001

CSV File With A Line Terminator:

- **A Line Terminator is a string used to terminate lines produced by writer.** The default value is \r or \n.

```
import csv
Data = [[‘Fruit’, ‘Quantity’], [‘Apple’, ‘5’], [‘Banana’, ‘7’], [‘Mango’, ‘8’]]
csv.register_dialect(‘myDialect’, delimiter = ‘|’, lineterminator = ‘\n’)
with open(‘c:\\pyprg\\ch13\\line.csv’, ‘w’) as f:
        writer = csv.writer(f, dialect=’myDialect’)
        writer.writerows(Data)
f.close()
```

OUTPUT:

Fruit	Quantity
Apple	5
Banana	7
Mango	8

CSV Files With Custom Delimiters:

```
import csv
info = [[‘SNO’, ‘Person’, ‘DOB’],
[‘1’, ‘Madhu’, ‘18/12/2001’],
[‘2’, ‘Sowmya’,’19/2/1998’],
[‘3’, ‘Sangeetha’,’20/3/1999’],
[‘4’, ‘Eshwar’, ‘21/4/2000’],
[‘5’, ‘Anand’, ‘22/5/2001’]]
csv.register_dialect(‘myDialect’,delimiter = ‘|’)
with open(‘c:\pyprg\ch13\dob.csv’, ‘w’) as f:
    writer = csv.writer(f, dialect=’myDialect’)
    for row in info:
        writer.writerow(row)
f.close()
```

OUTPUT:

Roll No	Name	City
1	Madhu	18/12/2001
2	Sowmya	19/2/1998
3	Sangeetha	20/3/1999
4	Eshwar	21/4/2000
5	Anand,	22/5/2001

CSV File With A Line Terminator:

```
import csv
Data = [['Fruit', 'Quantity'], ['Apple', '5'], ['Banana', '7'], ['Mango', '8']]
csv.register_dialect('myDialect', delimiter = '|', lineterminator = '\n')
with open('c:\\pyprg\\ch13\\line.csv', 'w') as f:
    writer = csv.writer(f, dialect='myDialect')
    writer.writerows(Data)
f.close()
```

OUTPUT:

Fruit	Quantity
Apple	5
Banana	7
Mango	8

Writing CSV File Into A Dictionary:

```
import csv
data = [{'MOUNTAIN' : 'Everest', 'HEIGHT': '8848'},
{'MOUNTAIN' : 'Anamudi ', 'HEIGHT': '2695'},
{'MOUNTAIN' : 'Kanchenjunga', 'HEIGHT': '8586'}]
with open('c:\\pyprg\\ch13\\peak.csv', 'w') as CF:
    fields = ['MOUNTAIN', 'HEIGHT']
    w = csv.DictWriter(CF, fieldnames=fields)
    w.writeheader()
    w.writerows(data)
print("writing completed")
CF.close()
```

OUTPUT:

MOUNTAIN,	HEIGHT
Everest,	8848
Anamudi ,	2695
Kanchenjunga,	8586

Writing Dictionary Into CSV File With Custom Dialects:

```
import csv
```

```
csv.register_dialect('myDialect', delimiter = '|', quoting=csv.QUOTE_ALL)
with open('c:\\pyprg\\ch13\\grade.csv', 'w') as csvfile:
    fieldnames = ['Name', 'Grade']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames, dialect="myDialect")
    writer.writeheader()
    writer.writerows([{'Grade': 'B', 'Name': 'Anu'},
                      {'Grade': 'A', 'Name': 'Beena'},
                      {'Grade': 'C', 'Name': 'Tarun'}])
print("writing completed")
```

OUTPUT:

"Name"	"Grade"
"Anu"	"B"
"Beena"	"A"
"Tarun"	"C"

Getting Data At Runtime And Writing It In a CSV File:

```
import csv
with open('c:\\pyprg\\ch13\\dynamicfile.csv', 'w') as f:
    w = csv.writer(f)
    ans='y'
    while (ans=='y'):
        name = input("Name?: ")
        date = input("Date of birth: ")
        place = input("Place: ")
        w.writerow([name, date, place])
        ans=input("Do you want to enter more y/n?: ")
F=open('c:\\pyprg\\ch13\\dynamicfile.csv','r')
reader = csv.reader(F)
for row in reader:
    print(row)
F.close()
```

OUTPUT:

```
Name?: Nivethitha
Date of birth: 12/12/2001
Place: Chennai
Do you want to enter more y/n?: y
Name?: Leena
Date of birth: 15/10/2001
Place: Nagercoil
Do you want to enter more y/n?: y
Name?: Padma
Date of birth: 18/08/2001
```

Place: Kumbakonam
Do you want to enter more y/n?: n
['Nivethitha', '12/12/2001', 'Chennai']
[]
['Leena', '15/10/2001', 'Nagercoil']
[]
['Padma', '18/08/2001', 'Kumbakonam']

H8 ^ fx

	A	B	C
1	Nivethitha	12/12/2001	Chennai
2			
3	Leena	15/10/2001	Nagercoil
4			
5	Padma	18/08/2001	Kumbakonam
6			

Choose the best answer (1 Marks)

1. A CSV file is also known as a ….

(A) Flat File

(B) 3D File

(C) String File

(D) Random File

2. The expansion of CRLF is

(A) Control Return and Line Feed

(B) Carriage Return and Form Feed

(C) Control Router and Line Feed

(D) Carriage Return and Line Feed

3. Which of the following module is provided by Python to do several operations on the CSV files?

(A) py

(B) xls

(C) csv

(D) os

4. Which of the following mode is used when dealing with non-text files like image or exe files?

(A) Text mode

(B) Binary mode

(C) xls mode

(D) csv mode

5. The command used to skip a row in a CSV file is

(A) next()

(B) skip()

(C) omit()

(D) bounce()

6. Which of the following is a string used to terminate lines produced by writer() method of csv module?

(A) Line Terminator

(B) Enter key

(C) Form feed

(D) Data Terminator

7. What is the output of the following program?

```
import csv
d=csv.reader(open('c:\PYPRG\ch13\city.csv'))
next(d)
for row in d:
    print(row)
```

if the file called “city.csv” contain the following details

```
chennai,mylapore

mumbai,andheri
```

(A) chennai,mylapore

(B) mumbai,andheri

(C) chennai

mumbai

(D) chennai,mylapore

mumbai,andheri

8. Which of the following creates an object which maps data to a dictionary?

(A) listreader()

(B) reader()

(C) tuplereader()

(D) DicReader ()

9. Making some changes in the data of the existing file or adding more data is called

(A)Editing

(B) Appending

(C)Modification

(D) Alteration

10. What will be written inside the file test.csv using the following program

```
import csv
D = [['Exam'],['Quarterly'],['Halfyearly']]
csv.register_dialect('M',lineterminator = '\n')
with open('c:\pyprg\ch13\line2.csv', 'w') as f:
    wr = csv.writer(f,dialect='M')
    wr.writerows(D)
f.close()
```

(A) Exam Quarterly Halfyearly

(B) Exam Halfyearly Quarterly

(C) E

Q

H

(D) Exam,

Quarterly,

Halfyearly

11. CSV - Comma Separated Values

12. Files saved in **excel** cannot be opened or edited by text editors

13. **CSV File** cannot store charts or graphs. It stores data but does not contain formatting, formulas, macros, etc.

14. A CSV file is also known as a **Flat File**.

15. Files in the CSV format can be imported to and exported from programs that store data in tables, such as ***Microsoft Excel*** or ***OpenOfficeCalc***

16. The last row begins with two commas because the first two fields of that row were empty in our spread sheet. They cannot be omitted.

17. If both MS Excel and Open Office calc is installed in the computer, by default the CSV file will be opened in **MS Excel**.

18. The CSV library contains objects and other code to **read, write, and process** data from and to CSV files.

19. **CSV files** have been used extensively in e-commerce applications because they are considered very easy to process.

20. There are **two** ways to read a CSV file.

21. File name or the complete path name can be represented either with in **" " or in ' '** in the open command.

22. Python has a built-in function **open()** to open a file.

23. open() function returns a **file object,** also called a **handle,** as it is used to read or modify the file accordingly.

24. The default is reading in **text mode.** In this mode, while reading from the file the data would be in the format of **strings.**

25. Binary mode returns bytes and this is the mode to be used when dealing with non-text files like image or exe files.

26. These whitespaces can be removed, by registering new dialects using **csv.register_dialect()** class of csv module.
27. **A dialect** describes the format of the csv file that is to be read.
28. In dialects the parameter **"skipinitialspace"** is used for removing whitespaces after the delimiter.
29. By default "skipinitialspace" has a value **false.**
30. A **dialect** is a class of csv module which helps to define parameters for reading and writing CSV
31. A **list** is a data structure in Python that is a mutable, or changeable, ordered sequence of elements.
32. To avoid that the first row should be skipped can be done by using the command **"next()"**.
33. **list_name.sort()** command arranges a list value in ascending order.
34. **list_name. sort(reverse)** is used to arrange a list in descending order.
35. To sort by more than one column you can use **itemgetter** with multiple indices: operator .itemgetter (col_no)
36. **DictReader()** gives OrderedDict by default in its output.
37. An **OrderedDict** is a dictionary subclass which saves the order in which its contents are added.
38. To remove the OrderedDict use **dict().**
39. The **writerow()** method writes one row at a time.
40. If you need to write all the data at once you can use **writerows()** method.
41. A **delimiter** is a string used to separate fields. The default value is **comma(,).**
42. **A Line Terminator** is a string used to terminate lines produced by writer**.** The default value is **\r or \n.**
43. Using **DictWriter()** class of csv module, we can write a csv file into a dictionary. It creates an **object** which maps data into a dictionary.
45. The **keys** are given by the fieldnames parameter.
46. **Excel** is a binary file whereas CSV format is a plain text format
47. Python has a **garbage collector** to clean up unreferenced objects
48. **close() method** will free up the resources that were tied with the file
49. The function **dict()** is used to print the data in dictionary format without order.
50. Adding a new row at the end of the file is called **appending a row**.

***** Best of Luck *****