

Unit-4: Python Interaction with text and CSV:

File handling (text and CSV files) using CSV module:

Que) What is CSV?

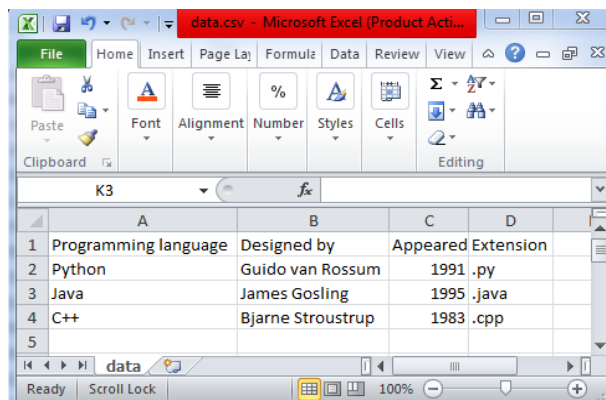
Ans:

- A CSV file (Comma Separated Values file) is a type of plain text file that uses specific structuring to arrange tabular data. Because it's a plain text file, it can contain only actual text data in other words, printable ASCII or Unicode characters.
- A CSV file (Comma Separated Values file) is a delimited text file that uses a comma(,) to separate values. It is used to store tabular data, such as a spreadsheet or database.
- Python's Built-in csv library makes it easy to read, write, and process data from and to CSV files.

Que) Where Do CSV Files Come From?

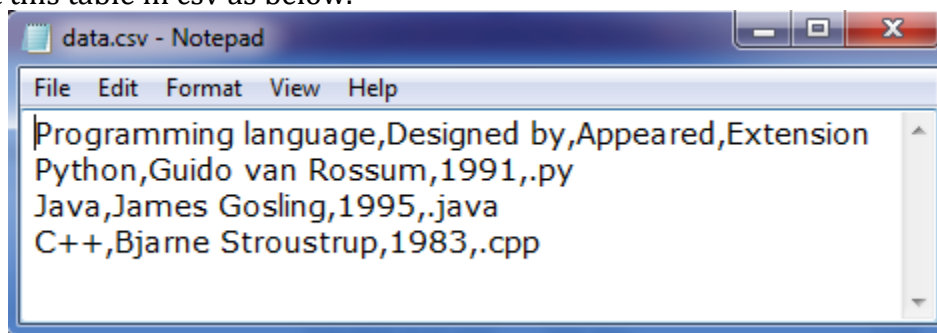
Ans:

- CSV files are normally created by programs that handle large amounts of data. They are a convenient way to export data from spreadsheets and databases as well as import or use it in other programs.
- Data in the form of tables is also called CSV (comma separated values) - literally "comma-separated values." This is a text format intended for the presentation of tabular data. Each line of the file is one line of the table. The values of individual columns are separated by a separator symbol - a comma (,), a semicolon (;) or another symbol.
- Consider the following table,



	A	B	C	D
1	Programming language	Designed by	Appeared	Extension
2	Python	Guido van Rossum	1991	.py
3	Java	James Gosling	1995	.java
4	C++	Bjarne Stroustrup	1983	.cpp

- You can represent this table in csv as below.



```
data.csv - Notepad
File Edit Format View Help
Programming language,Designed by,Appeared,Extension
Python,Guido van Rossum,1991,.py
Java,James Gosling,1995,.java
C++,Bjarne Stroustrup,1983,.cpp
```

- As you can see each row is a newline, and each column is separated with a comma. This is an example of how a CSV file looks like.

Python CSV Module:

Python provides a CSV module to handle CSV files. To read/write data, you need to loop through rows of the CSV. You need to use the split method to get data from specified columns.

The csv module gives the Python programmer the ability to parse CSV (Comma Separated Values) files. A CSV file is a human readable text file where each line has a number of fields, separated by commas or some other delimiter. You can think of each line as a row and each field as a column. The CSV format has no standard, but they are similar enough that the csv module will be able to read the vast majority of CSV files. You can also write CSV files using the csv module.

File Modes:

Read a CSV Files:

Step-1: To read data from CSV files, you must use the reader function to generate a reader object.

The reader function is developed to take each row of the file and make a list of all columns. Then, you have to choose the column you want the variable data for.

Open()function:

The open() function opens a file, and returns it as a file object.

Syntax: **Open("Filename","mode")**

Parameter Values:

Parameter	Description
<i>file</i>	The path and name of the file
<i>mode</i>	A string, define which mode you want to open the file in: "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 exist 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)

Reader()function:

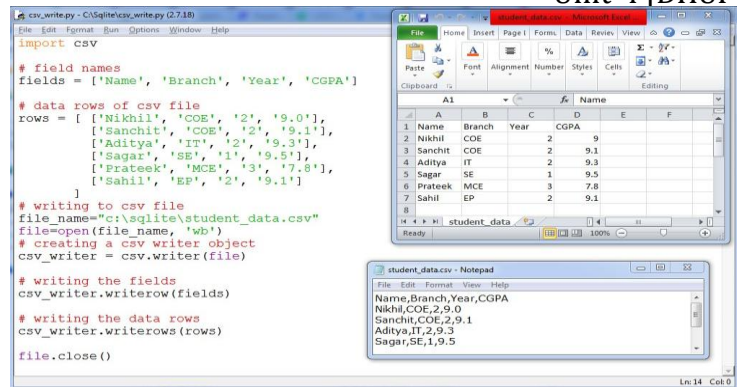
The reader() is used to read the file, which returns an Iterable reader object. The reader object is then iterated using a for loop to print the contents of each row.

Syntax:

reader("Filename")

Example: (Read_csv.py)

```
#import necessary modules
import csv
file=open('C:\sqlite\data.csv','r')
data=csv.reader(file)
next(data)
for row in data:
    print(row)
file.close()
```

**Write Into CSV File:**

- csv.writer class is used to insert data to the CSV file. This class returns a writer object which is responsible for converting the user's data into a delimited string.
- A csv file object should be opened with newline="" otherwise newline characters inside the quoted fields will not be interpreted correctly.
- To write to an existing file, you must add a parameter to the open() function:
 - "a" - Append - will append to the end of the file
 - "w" - Write - will overwrite any existing content

csv.writer class provides two methods for writing to CSV.

They are:

- 1) writerow()
- 2) writerows()

1) writerow(): This method writes a single row at a time. Field row can be written using this method.

Syntax: **writerow(fields)**

2) writerows(): This method is used to write multiple rows at a time. This can be used to write rows list.

Syntax: **writerows(rows)**

Example: (write_csv.py)

```
import csv
# field names
fields = ['Name', 'Branch', 'Year', 'CGPA']
# data rows of csv file
rows = [
    ['Nikhil', 'COE', '2', '9.0'],
    ['Sanchit', 'COE', '2', '9.1'],
    ['Aditya', 'IT', '2', '9.3'],
    ['Sagar', 'SE', '1', '9.5'],
    ['Prateek', 'MCE', '3', '7.8'],
    ['Sahil', 'EP', '2', '9.1']
]
# writing to csv file
file_name = "E:/SEM-3/DHUP/UNIT-4/student_data.csv"
file = open(file_name, 'w', newline="")
# creating a csv writer object
csv_writer = csv.writer(file)
```

```
csv_writer.writerow(fields)
csv_writer.writerows(rows)
print("STUDENT_DATA CSV file created successfully!")
file.close()
```

Specify File Mode

Here are five different modes you can use to open the file:

Character	Mode	Description
'r'	Read (default)	Open a file for read only
'w'	Write	Open a file for write only (overwrite)
'a'	Append	Open a file for write only (append)
'r+'	Read + Write	open a file for both reading and writing
'x'	Create	Create a new file

Python Dictionary:

Dictionary in Python is an ordered collection of data values, used to store data values like a map, which, unlike other Data Types that hold only a single value as an element, Dictionary holds **key: value** pair.

Key-value is provided in the dictionary to make it more optimized. Dictionary holds a pair of values, one being the Key and the other corresponding pair element being its **Key : value**. Values in a dictionary can be of any data type and can be duplicated, whereas keys can't be repeated.

Write into CSV file using csv.

dictwriterclass: This class returns a writer object which maps dictionaries on to output rows.

Syntax: `csv.DictWriter(csvfile, fieldnames)`

Here,

- **csvfile:** A file object with write() method.
- **field names:** A sequence of keys that identify the order in which values in the dictionary should be passed.

csv.DictWriter provides two methods for writing to CSV.

They are:

1) writeheader(): writeheader() method simply writes the first row of your csv file using the pre-specified fieldnames.

Syntax: `Writeheader()`

2) writerows(): write rows method simply writes all the rows but in each row, it writes only the values (not keys).

Syntax: `writerows(mydict)`

Example:(Write_dict.py)

```
import csv
# field names
fields = ['name', 'branch', 'year', 'cgpa']
# data rows as dictionary objects
mydict = [
```

```
{'name': 'Nikhil', 'branch': 'COE', 'year': 2, 'cgpa': 9.0},
{'name': 'Sanchit', 'branch': 'COE', 'year': 2, 'cgpa': 9.1},
{'name': 'Aditya', 'branch': 'IT', 'year': 2, 'cgpa': 9.3},
{'name': 'Sagar', 'branch': 'SE', 'year': 1, 'cgpa': 9.5},
{'name': 'Prateek', 'branch': 'MCE', 'year': 3, 'cgpa': 7.8},
{'name': 'Sahil', 'branch': 'EP', 'year': 2, 'cgpa': 9.1}
]
# writing to csv file
file_name = "Result.csv"
with open(file_name, "w", newline="") as csvfile:
    writer = csv.DictWriter(csvfile, fieldnames=fields)
    # write header
    writer.writeheader()
    # write data rows
    writer.writerows(mydict)
print("Dictionary data written successfully!")
```

Example: (Write_dict1.py)

Header name (Columnname) taken from Dictionary Key.

```
# importing the csv module
import csv
# my data rows as dictionary objects
mydict = [
    {'branch': 'COE', 'cgpa': '9.0', 'name': 'Nikhil', 'year': '2'},
    {'branch': 'COE', 'cgpa': '9.1', 'name': 'Sanchit', 'year': '2'},
    {'branch': 'IT', 'cgpa': '9.3', 'name': 'Aditya', 'year': '2'},
    {'branch': 'SE', 'cgpa': '9.5', 'name': 'Sagar', 'year': '1'},
    {'branch': 'MCE', 'cgpa': '7.8', 'name': 'Prateek', 'year': '3'},
    {'branch': 'EP', 'cgpa': '9.1', 'name': 'Sahil', 'year': '2'}
]
# field names
fields = list(mydict[0].keys())
print(fields)
# writing to csv file
file = open('E:\SEM-3\DHUP\UNIT-4\Result1.csv', 'w', newline="")
# creating a csv dict writer object
writer = csv.DictWriter(file, fieldnames=fields)
# writing headers (field names)
writer.writeheader()
# writing data rows
writer.writerows(mydict)
file.close()
```

Reading from CSV file using csv.DictReaderclass:

Create an object that operates like a regular reader but maps the information in each row to a dict whose keys are given by the optional fieldnames parameter.

Syntax: csv.DictReader(csvfile, fieldnames)

Example:(Read_dict.py)

```
import csv
file=open('E:/SEM-3/DHUP/UNIT-4/Result.csv','r')
dict = csv.DictReader(file)
for row in dict:
    print(row)
file.close()
```

Append: Opening a file in **append mode ("a")** means new data will be added at the end of the file without erasing existing content.

Example:

```
# importing the csv module
import csv
# new data rows as dictionary objects
newdict = [
    {'branch': 'CSE', 'cgpa': '8.9', 'name': 'Ravi', 'year': '2'},
    {'branch': 'IT', 'cgpa': '9.2', 'name': 'Meena', 'year': '3'}
]
# field names (must match existing CSV)
fields = ['branch', 'cgpa', 'name', 'year']
# opening the file in append mode
File = open('E:/SEM-3/DHUP/UNIT-4/Result1.csv', 'a', newline='')
# creating a csv dict writer object
writer = csv.DictWriter(File, fieldnames=fields)
# writing data rows (no header this time)
writer.writerows(newdict)
File.close()
```

How to install pandas, Numpy and matplotlib packages in python?

Step-1: First, make sure pip has been installed on your OS. If it is not installed, then refer following command. You can download file using get-pip.py.

Step-2: Run pip install command to install related packages pip

install numpy

pip install pandas

pip install matplotlib

Numpy:

Numpy is a Python library used for working with arrays. In Python we have lists that serve the purpose of arrays, but they are slow to process. Numpy aims to provide an array object that is up to 50x faster than traditional Python lists. The array object in Numpy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.

Numpy is used to work with arrays. The array object in Numpy is called ndarray. We can create a Numpy ndarray object by using the array() function.

Importing numpy:

Once Numpy is installed, import it in your applications by adding the import keyword:

```
import numpy
```

Example:

```
#import numpy library
import numpy
#creating array under numpy using array()
a=numpy.array([10,20,30,40])
print(a)
```

Numpy as np

Numpy is usually imported under the np alias. Create an alias with the as key word while importing:

Syntax: `import numpy as np`

Now the Numpy package can be referred to as np instead of numpy.

Example: `import numpy as np`
 `arr=np.array([1,2,3,4,5])`
 `print(arr)`

Creating 2-D array in NumPy:**Example:**

```
import numpy as np
arr=np.array([[1,2,3],[4,5,6]])
print(arr)
```

Check the number of dimension:

Numpy Arrays provides then dim attribute that returns an integer that tells us how many dimensions the array have.

Example:

```
import numpy as np
a= np.array(42)
b=np.array([1,2,3,4,5])
c=np.array([[1,2,3],[4,5, 6]])
print(a.ndim)
print(b.ndim)
print(c.ndim)
```

Pandas in Python

- A panda is a Python library used for working with datasets.
- It has functions for analyzing, cleaning, exploring, and manipulating data.
- A panda is a Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive.
- It aims to be the fundamental high-level building block for doing practical, real-world data analysis in Python.

What Can Pandas Do?

Pandas gives you answers about the data Like:

- 1) Is there a correlation between two or more columns?
- 2) What is average value?
- 3) Max value?
- 4) Min value?

Import Pandas

Once Pandas is installed, import it in your applications by adding the import keyword:

Syntax: **import pandas**

Pandas as pd

A panda is usually imported under the pd alias. Create an alias with the as keyword while importing:

import pandas as pd

Now the Pandas package can be referred to as pd instead of pandas.

Data Frames:

Pandas DataFrame is two-dimensional size tabular data structure with labeled axes (rows and columns).

A Dataframe is a two-dimensional data structure ,i.e. ,data is aligned in a tabular fashion in rows and columns.

Pandas Data Frame consists of three principal components, the data, rows, and columns.

	Name	Team	Number	Position	Age
0	Avery Bradley	Boston Celtics	0.0	PG	25.0
1	John Holland	Boston Celtics	30.0	SG	27.0
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN
6	Evan Turner	Boston Celtics	11.0	SG	27.0

Creating a data frame

Example:

```
import pandas as pd
lst=['Kia','Piya','Diya','Jiyai','Niya']
df = pd.DataFrame(lst)
print(df)
```

Pandas Read CSV File

A simple way to store big data sets is to use CSV files. (comma separated files).

CSV files contains plain text and is a well know format that can be read by everyone including Pandas.

Load the CSV into Data Frame:**Example:**

```
import pandas as pd
df=pd.read_csv('c:\sqlite\data.csv')
print(df)
```

If you want to print all the records from CSV file then use **to_string()** to print the entire DataFrame.

Syntax: DataFrame.to_string()

Write a Data frame into CSV File:**Example:**

```
import pandas as pd
name=["Kajal","Bhumika","Ruchika","Pinal"] #list of name, degree, score
deg=["MBA","BCA","M.Tech","MBA"]
scr = [90, 40, 80, 98] #dictionary of lists
dict={'name':name,'degree':deg,'score':scr}
df = pd.DataFrame(dict)
df.to_csv('file1.csv')
```

Write Data Frame into CSV file without Header and Index:**Example:**

```
import pandas as pd
#list of name, degree, score
name=["Kajal","Bhumika","Ruchika","Pinal"]
deg=["MBA","BCA","M.Tech","MBA"]
scr = [90, 40, 80, 98]
#dictionary of lists
dict={'name':name,'degree':deg,'score':scr}
df = pd.DataFrame(dict)
# saving the dataframe
df.to_csv('file1.csv',header=False, index=False)
```

How to change row Index number as per Choice:

For this purpose Use range function

Syntax: range(start_value,End_value)

Example:

```
import pandas as pd
#list of name, degree, score
name=["Kajal","Bhumika","Ruchika","Pinal"]
deg=["MBA","BCA","M.Tech","MBA"]
scr = [90, 40, 80, 98]
```

```
#dictionary of lists
dict={'name':nme,'degree':deg,'score':scr}
df=pd.DataFrame(dict,index=range(1,5))
#saving the dataframe
df.to_csv('file1.csv')
```

Example:

```
import pandas as pd
#list of name, degree,score
name=["Kinjal","Bhumi","Archana","Pinky"]
deg=["MBA","BCA","M.Tech","MBA"]
scr = [90, 40, 80, 98]
#dictionaryoflists
dict={'name':name,'degree':deg,'score':scr}
#df=pd.DataFrame(dict,index=range(1,5))
# You can provide index value like this also
df=pd.DataFrame(dict,index=['s101','s102','s103','s104'])
#savingthedataframe
df.to_csv('file1.csv')
```

How to Append a CSV file Using Data Frame?**Example:**

```
import pandas as pd
# list of name, degree, score
nme = ["Tisha", "Anandi", "Tanvi", "Khushali"]
deg = ["B.TECH", "BBA", "BCOM", "BCA"]
scr = [29, 30, 40, 38]
# dictionary of lists
dict = {'name': nme, 'degree': deg, 'score': scr}
# You can provide index value like this also
df = pd.DataFrame(dict, index=['s105', 's106', 's107', 's108'])
# saving the dataframe
df.to_csv('file1.csv', mode='ab', header=False)
```

How to write data into Excel file using Data Frame?

Note: Before Working With Excel install following packages:

```
pip install openpyxl
pip install xlrd==1.2.0
```

Example:

```
import pandas as pd
import openpyxl
# employee data
```

```
data = [
    ['Heta', 25000],
    ['Rekha', 30000],
    ['Roma', 35000]
]
# create DataFrame with custom index and column names
df = pd.DataFrame(data, index=['E101', 'E102', 'E103'], columns=['Name', 'Salary'])
# write DataFrame to Excel file
w = pd.ExcelWriter('Emp.xlsx')
df.to_excel(w)
w.save()
```

How to write data into Specific Excel Sheet in an Excel using DataFrame?

Example:

```
import pandas as pd
# student data
stud_data = [
    ['Isha', 'MCA', 80.7],
    ['Karan', 'Bcom', 67.78],
    ['Priya', 'BBA', 57],
    ['Swati', 'CA', 78]
]
# create DataFrame with custom index and column names
df = pd.DataFrame(stud_data, index=['s101', 's102', 's103', 's104'], columns=['Name', 'Degree', 'Per'])
# write DataFrame to Excel file with sheet name
w = pd.ExcelWriter("Stud.xlsx")
df.to_excel(w, 'SVP_student')
w.save()
```

How to read data from Excel file using DataFrame?

Example:

```
import pandas as pd
print("=====Reading Excel Sheet=====")
df = pd.read_excel('Stud.xlsx')
print(df)
print("=====read without Index=====")
df = pd.read_excel('Stud.xlsx', index_col=0)
print(df)
print("=====Read specific Sheet=====")
df = pd.read_excel('Stud.xlsx', sheet_name='Division1')
print(df)
```

Data Frame Functions:

1. DataFrame.head():

- The head() function is used to get the first n rows. This function returns the first n rows for the object based on position.
- It is beneficial when we have massive datasets, and it is not possible to see the entire dataset at once.

Syntax: DataFrame.head(n)

Here, n is the number of rows to be selected. Here default value is

Example:

```
import pandas as pd
df=pd.read_csv('data.csv')
print("\n",df.head())
print("=====")
print("\n",df.head(2))
print("=====")
```

Note:

We can use Negative Number with DataFrame.head() function. we want to see all the rows except for the last n rows, we can pass the negative value as a parameter to DataFrame.head().

Example: if we want to display all the rows except the bottom two rows, we can use DataFrame.head(2) function.

1. DataFrame.tail():

We can use the DataFrame.tail() function to display the last n rows of the DataFrame. Like the head function, this function is used when we want to view a smaller section of the entire DataFrame.

It takes input as the number of rows to be displayed from the bottom. The default value is 5.

Syntax: DataFrame.tail(n)

Here, n is number to be printed from the bottom.

Example:

```
import pandas as pd
df=pd.read_csv('data.csv')
print("\n",df.tail())
print("=====")
print(df.tail(2))
```

Note:

We can use negative number with DataFrame.tail() function. When we want to see our entire dataset except for the first few rows, we can use DataFrame.tail() function and pass the negative value as a parameter to it.

Example: if we display all the rows except the top 2 rows using DataFrame.tail (2).

1. DataFrame.describe():

The describe() method is used for calculating some statistical data like percentile, mean and std of the numerical values of the Series or DataFrame.

Syntax: DataFrame.describe(percentiles=None)

Parameter:

Percentiles: The percentiles to include in the output. All should fall between 0 and 1. The default is [0.25,0.5,0.75], which returns the 25th, 50th, and 75th percentiles.

How to calculate Central tendency-

1. Mean
2. Median
3. Mode
4. variance using describe()

Example:

```
import pandas as pd
df = pd.read_csv('file1.csv')
print("=====Reading Score=====")
print(df['score'].to_string(index=False))
print("=====Calculation=====")
print("Sum:\t", df['score'].sum())
print("Min:\t", df['score'].min())
print("Max:\t", df['score'].max())
print("Median:\t", df['score'].median())
print("=====Mode=====")
print(df['score'].mode())
print("=====Standard Deviation=====")
print(df['score'].std())
print("=====Variance (actually skewness here)=====")
print(df['score'].skew())
print("=====Describe Data=====")
print(df['score'].describe())
```

- 1) loc():** loc is label-based, which means that you have to specify rows and columns based on their row and column labels.

Syntax: loc[row_label,column_label]

- 2) iloc():** i loc is integer position-based, so you have to specify rows and columns by their integer position values (0-based integer position).

Syntax: iloc[row_position,column_position]**Example:**

```
import pandas as pd
df = pd.read_csv('file1.csv')
print("=====Fetch name=====")
print(df.loc[:, 'name'])
print("=====Fetch Name and Score=====")
print(df.loc[:, ['name', 'score']])
print("=====Fetch detail of 2nd index=====")
print(df.iloc[2])
print("=====Fetch detail of 2nd column=====")
print(df.iloc[:, 2])
print("=====Fetch column 1st and 2nd=====")
print(df.iloc[:, [1, 2]])
print("=====Fetch value of column name=====")
print(df['name'])
```

3) To_numpy:

Pandas DataFrame is two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). This data structure can be converted to NumPy ndarray with the help of DataFrame.to_numpy() method.

Syntax: DataFrame.to_numpy(Data_set)

Example:

```
import pandas as pd
data = [
    ['Heta', 15000],
    ['Pankaj', 16000],
    ['Minal', 20000]
]
df = pd.DataFrame(data)
print("List converted into numpy array:\n")
df_numpy = df.to_numpy()
print(df_numpy)
```

4) DataFrame.Values():

This function will display the value of the data frame.

Syntax: DataFrame.values()

Example:

```
import pandas as pd
df = pd.read_csv('file1.csv')
print(df)
print("=====Values of file1=====")
print(df.values)
```