Csv file handling:

A csv file (comma separated value file) is a type of plain text file that uses specific structuring tabular data. Because it’s a plain text, it can contain only actual text data- in other word, printable ASCII or Unicode characters.

The structure of a CSV file is given away by its name. normally, CSV files use a comma to separate each specific data value. Here what that structure looks like

In general, the separator character is called a delimiter, and the comma is not only one used other popular delimiter include the tab(\t), colon(:) and semi-colon(;) character.

Properly parsing a CSV file requires us to know which delimiter is being used.

CSV is a simple flat file in a human readable format which is extensively used to store tabular data in a spreadsheet or database. A CSV file stores tabular in plain text.

Each line in a file is known as data/record. Each record consists of one or more fields, separated by commas(also known as delimiters), i.e., each records is also part of this file. Tabular data is stored as text in a CSV file. The use of comma as a field separator is source of the name for this file format. It stores our data into a spreadsheet or a database.

There are several advantages.

* CSV is faster to handle
* CSV is smaller in size
* CSV is easy to generate and import onto a spreadsheet or database.
* CSV is human readable and easy to edit manually.
* CSV is simple to implement and parse.
* CSV is processed by almost all existing applications.

For working with CSV files in Python, there is an inbuilt module named CSV.

It is used to read and write tabular data in CSV format.

To perform read and write tabular data in CSV file, we must import the CSV module.

Import CSV

Along with this module, open() function is used to open a CSV file and return a file object.

Fileobject=open(CSVfile,mode,newline=’\n’)

There are two basic operations that can be carried out on a csv file:-

* Reading from a csv file
  + Csv.reader(fileobject,dialect=’excel’,\*\*fmparams)
* Writing to csv File
  + Csv.writer(fileobject,dialect=’excel’,\*\*fmparams)

Handling CSV files in python is a common task for data manipulation and analysis. Python provides built-in libraries and modules to handle CSV files

1. Writing csv files:
   1. To write data to a csv file you can use the csv.writer class from the ‘csv’ module. You can pass the data as a list of list, where each inner list represents a row in the CSV file.
      1. Example
         1. Import csv
         2. Data =[[‘name’,’age’,’country’],[‘john’,30,’USA’],[‘alice’,25,’IND]]
         3. File=open(‘output.csv’,’w’,newline=’\n’)
         4. Writer=csv.writer(file)
         5. Writer.writerows(data)
2. Writer CSV files with Header:
   1. To write data with header, you can use the CSV.dictwriter class which allows you to write data using dictionaries

Generator: it is a phenomenon of new value one space to another space by the help of yield keyword.

* Generator object will be generated by the generator function.
* It holds the set values from yield keyword.
* Generator object can be utilized only one time.
* If we want to fetch the values from generator object, we have to apply typecasting or generator object given to the iterator (static or dynamic).
* When we utilize the generator object then only generator will perform the task

Yield: it is a predefined keyword

* It will take the value and it return to the generator object.
* Yield will pause the execution of the program and it return the value to the generator object after the return the it will resume the execution of the program.
* Syntax:-
  1. Def func\_name(args,…):
     + Statements
     + Yield value
     + Statements
  2. Vn=func\_name(args)

Filter:

* It is a predefined function used to filter the value from the give collection and that condition is satisfied those results are stored in the filter object.
* Inside the filter function we provide the function address and collection.
* Filter function will not accept the none value. It will avoid the none type value.
* If the condition is satisfied it store the actual value otherwise it will store the none value, but final result will avoid the none value.
* Filter will also return the memory address called filter object, where the object where the values will be stored.
* Filter() will return the a filter object that can be either given to static iterator or for loop or typecasting.
* Which stores the address returned by the filter function is called filter object.
* When the condition is satisfied it stores the satisfied value inside the filter object based on the result function.
* To reduce the size of the code by using the filter function.

Syntax:

Var=filter(func\_address,collection)

List comprehension:

* In python list comprehensive is a simple and compact syntax for making a list out of a list
* It’s a quick technique to make a new list by performing an operation on each item in the old one
* list comprehension is much faster than using a for loop to handle a list
* A list comprehension is made up of brackets that hold the expression that is run for each element as well as for loop that iterates each element

syntax

var= [return value for var in collection]

var= [return value for var in collection if<condition>]

var= [return value if<condition> else return valve for var in collection]

var= [return value for var in collection for var in collection if<condition>]

var= [return value if<condition> else return valve for var in collection for var in collection]

var= [[fsb tsb][condition] for I in collection]

tuple comprehension:

* It returns the generator object

Syntax:

var= (return value for var in collection)

var= (return value for var in collection if<condition>)

var= (return value if<condition> else return valve for var in collection)

var= (return value for var in collection for var in collection if<condition>)

var= (return value if<condition> else return valve for var in collection for var in collection)

set comprehensive:

Syntax:

var= {return value for var in collection}

var= {return value for var in collection if<condition>}

var= {return value if<condition> else return valve for var in collection}

var= {return value for var in collection for var in collection if<condition>}

var= {return value if<condition> else return valve for var in collection for var in collection}

dict comprehensive:

Syntax:

var= {return key:return value for var in collection}

var= { return key:return value for var in collection if<condition>}

var= { return key:return value if<condition> else return valve for var in collection}

var= {return key: return value for var in collection for var in collection if<condition>}

var= {return key: return value if<condition> else return valve for var in collection for var in collection}