File Handling in Python

Files are named locations on disk to store related information. They are used to permanently store data in a non-volatile memory(e.g. hard disk). Since Random Access Memory (RAM) is volatile (which loses its data when the computer is turned off), we use files for future use of the data by permanently storing them.

When we want to read from or write to a file, we need to open it first. When we are done, it needs to be closed so that the resources that are tied with the file are freed.

Hence, in Python, a file operation takes place in the following order:

- Open a file
- Read or write (perform operation)
- Close the file

Types Of File in Python

There are two types of files in Python and each of them are explained below in detail with examples for your easy understanding.

They are:

- Binary file
- Text file

Binary files in Python

All binary files follow a specific format. We can open some binary files in the normal text editor but we can't read the content present inside the file. That's because all the binary files will be encoded in the binary format, which can be understood only by a computer or machine.
For handling such binary files we need a specific type of software to open it.
For Example, You need Microsoft word software to open .doc binary files.
Likewise, you need a pdf reader software to open .pdf binary files and you need a photo editor software to read the image files and so on.

Most of the files that we see in our computer system are called binary files.
Example:
• Document files: .pdf, .doc, .xls etc.
• Image files: .png, .jpg, .gif, .bmp etc.
• Video files: .mp4, .3gp, .mkv, .avi etc.
• Audio files: .mp3, .wav, .mka, .aac etc.
• Database files: .mdb, .accde, .frm, .sqlite etc.
• Archive files: .zip, .rar, .iso, .7z etc.
• Executable files: .exe, .dll, .class etc.

Text files in Python

A text file is usually considered as sequence of lines. Line is a sequence of characters (ASCII), stored on permanent storage media. Although default character coding in python is ASCII but supports Unicode as well.
in text file, each line is terminated by a special character, known as End of Line (EOL). From strings we know that \n is newline character.
at the lowest level, text file is collection of bytes.
Text files are stored in human readable form they can also be created using any text editor
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 etc.

• Source code: c, app, js, py, java etc.

• Documents: txt, tex, RTF etc.

• Tabular data: csv, tsv etc.

• Configuration: ini, cfg, reg etc.

Opening or Creating a New File in Python

The method open() is used to open an existing file or creating a new file. If the complete directory is not given then the file will be created in the directory in which the python file is stored. The syntax for using open() method is given below.

— Syntax:

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

•The open method returns file object which can be stored in the name file_object (file-handle).

File name is a unique name in a directory. The open() function will create the file with the specified name if it is not already exists otherwise it will open the already existing file.

Opening Files in Python

•The access mode it is the string which tells in what mode the file should be opened for operations. There are three different access modes are available in python.

•Reading: Reading mode is crated only for reading the file. The pointer will be at the beginning of the file.

•Writing: Writing mode is used for overwriting the information on existing file.

• Append: Append mode is same as the writing mode. Instead of over writing the information this mode append the information at the end.

•Below is the list of representation of various access modes in python.

Access modes in Text Files

• ‘r' – Read Mode: Read mode is used only to read data from the file.

• ‘w' – Write Mode: This mode is used when you want to write data into the file or modify it. Remember write mode overwrites the data present in the file.

•‘a' – Append Mode: Append mode is used to append data to the file. Remember data will be appended at the end of the file pointer.

• ‘r+' – Read or Write Mode: This mode is used when we want to write or read the data from the same file.

• ‘a+' – Append or Read Mode: This mode is used when we want to read data from the file or append the data into the same file.

Examples Opening a file:

open file in current directory

• f = open("test.txt“, “r”)

specifying full path

• f = open(r“D:\temp\data.txt“, “r”)

– –raw string

• f = open(“D:\\temp\\data.txt“, “r”)

– -absolute path

Closing Files in Python

After processing the content in a file, the file must be saved and closed. To do this we can use another method close() for closing the file. This is an important method to be remembered while handling files in python.

•Syntax: file_object.close()

```
string = "This is a String in Python“
my_file = open(my_file_name.txt,"w+",1)
my_file.write(string)
my_file.close()
print(my_file.closed)
```

Reading Information in the File

•In order to read a file in python, we must open the file in read mode.

•There are three ways in which we can read the files in python.

- read([n])
- readline([n])
- readlines()
- all lines returned to a list

Here, n is the number of bytes to be read

Example 1:

```
my_file = open("C:/Documents/Python/test.txt", "r") print(my_file.read(5))
```

Output:

Hello

Here we are opening the file test.txt in a read-only mode and are reading only the first 5 characters of the file using the my_file.read(5) method.

Example 2:

```
my_file = open("C:/Documents/Python/test.txt", "r")
print(my_file.read())
```

Output:

Hello World

Hello Python

Good Morning

Here we have not provided any argument inside the read() function.Hence it will read all the content present inside the file.

Example 3:

```
my_file = open(“C:/Documents/Python/test.txt”, “r”)
print(my_file.readline(2))
```

Output:

He

This function returns the first 2 characters of the next line.

Example 4:

```
my_file = open(“C:/Documents/Python/test.txt”, “r”)
print(my_file.readline())
```

Output:

Hello World

Using this function we can read the content of the file on a line by line basis.

Example 5:

```
my_file = open(“C:/Documents/Python/test.txt”, “r”)
print(my_file.readlines())
```

Output:

[‘Hello World\n’, ‘Hello Python\n’, ‘Good Morning’]

Here we are reading all the lines present inside the text file including the newline characters.

Reading a specific line from a File

```
line_number = 4
fo = open("C:/Documents/Python/test.txt", 'r')
currentline = 1
for line in fo:
        if(currentline == line_number):
            print(line)
            break
        currentline = currentline +1
```

Output:

How are You

In the above example, we are trying to read only the 4 th line from the 'test.txt' file using a "for loop".

Reading the entire file at once

```
filename = "C:/Documents/Python/test.txt"
filehandle = open(filename, 'r')
filedata = filehandle.read()
print(filedata)
```

Output:

Hello World

Hello Python

Good Morning

How are You

Write to a Python File

In order to write data into a file, we must open the file in write mode.

• We need to be very careful while writing data into the file as it overwrites the content present inside the file that you are writing, and all the previous data will be erased.

• We have two methods for writing data into a file as shown below.

– write(string)

– writelines(list)

• Example 1:

```
my_file = open("C:/Documents/Python/test.txt", "w")
my_file.write("Hello World")
```

The above code writes the String 'Hello World' into the 'test.txt' file.

Example 2:

```
my_file = open("C:/Documents/Python/test.txt", "w")
my_file.write("Hello World\n")
my_file.write("Hello Python")
```

•The first line will be 'Hello World' and as we have mentioned \n character,the cursor will move to the next line of the file and then write 'Hello Python'.

•Remember if we don't mention \n character, then the data will be written continuously in the text file like 'Hello WorldHelloPython'

Example 3:

```
fruits = ["Apple\n", "Orange\n", "Grapes\n", "Watermelon"]
my_file = open("C:/Documents/Python/test.txt", "w")
my_file.writelines(fruits)
```

The above code writes a list of data into the 'test.txt' file simultaneously

Append in a Python File

To append data into a file we must open the file in 'a+' mode so that we will have access to both the append as well as write modes.

Example 1:

```
my_file = open("C:/Documents/Python/test.txt", "a+")
my_file.write ("Strawberry")
```

The above code appends the string 'Strawberry' at the end of the 'test.txt' file

Example 2:

```
my_file = open("C:/Documents/Python/test.txt", "a+")
my_file.write ("\nGuava")
```

The above code appends the string ‘Apple’ at the end of the ‘test.txt’ file in a new line.

flush()function

•When we write any data to file, python hold everything in buffer (temporary memory) and pushes it onto actual file later. If you want to force Python to write the content of buffer onto storage, you can useflush() function.
•Python automatically flushes the files when closing them i.e. it will be implicitly called by the close(), BUT if you want to flush before closing any file you can useflush()

Removing WhiteSpaces after Reading From File

•read() and readline() reads data from file and return it in the form of string and readlines() returns data in the form of list.
• All these read function also read leading and trailing whitespaces, new line characters. If you want to remove these characters you can use functions

– strip() : removes the given character from both ends.

– lstrip(): removes given character from left end

– rstrip(): removes given character from right end

FilePointer

•Every file maintains a file pointer which tells the current position in the file where reading and writing operation will take.

•When we perform any read/write operation two things happens:

– The operation at the current position of file pointer

– File pointer advances by the specified number of bytes

Binary file Operations

•If we want to write a structure such as list or dictionary to a file and read it subsequently we need to use the Python module pickle.

•Pickling is the process of converting structure to a byte stream before writing to a file and while reading the content of file a reverse process called Unpickling is used to convert the byte stream back to the original format.

First we need to import the module called pickle.

• This module provides 2 main functions:

- dump() : to write the object in file which is loaded in binary mode

  Syntax : dump(object_to_write, filehandle)

- – load() : dumped data can be read from file using load() i.e. it is used to read object from pickle file.

  Syntax: object = load(filehandle)

The four major operations performed using a binary file are—

1.Inserting/Appending a record in a binary file

2.Reading records from a binary file

3.Searching a record in a binary file

4.Updating a record in a binary file

1.Inserting/Appending a record in a binary file

•Inserting or adding (appending) a record into a binary file requires importing pickle module into a program followed by dump() method to write onto the file.

2.Reading a record from a binary file

•deals with reading the contents from binary file student using load() method of pickle module. It is used to read the object from the opened file. The syntax for this is given by the statement—

– object = pickle.load(file)

3.Searching a record in a binary file

Searching the binary file ("student")is carried out on the basis of the roll number entered by the user. The file is opened in the read-binary mode and gets stored in the file object, f. load() method is used to read the object from the opened file. A variable ‘found’ is used which will tell the status of the search operation being successful or unsuccessful. Each record from the file is read and the content of the field, roll no, is compared with the roll number to be searched. Upon the search being successful, appropriate message is displayed to the user.

4.Updating a record in a binary file

•Updating a record in the file requires roll number (search field) to be fetched from the user whose name (Record) is to be updated

•Once the record is found, the file pointer is moved to the beginning of the file using seek(0) statement, and then the changed values are

written to the file and the record is updated. seek() method is used for random access to the file. (more about seek() in later sessions)

RANDOM ACCESS IN FILES USING TELL() AND SEEK()

•Till now, in all our programs we laid stress on the sequential processing of data in a text and binary file.
•But files in Python allow random access of the data as well using built-in methods seek() and tell().

seek()—seek() function is used to change the position of the file handle (file pointer) to a given specific position. File pointer is like a cursor, which defines from where the data has to be read or written in the file.
• Python file method seek() sets the file's current position at the offset. This argument is optional and defaults to 0, which means absolute file positioning. Other values are: 1, which signifies seek is relative (may change) to the current position, and 2, which means seek is relative to the end of file. There is no return value.
• The reference point is defined by the "from_what" argument. It can have any of the three values:

• 0: sets the reference point at the beginning of the file, which is by default.

• 1: sets the reference point at the current file position.

• 2: sets the reference point at the end of the file.

- seek() can be done in two ways:

– Absolute Positioning

– Relative Positioning

•Absolute referencing using seek() gives the file number on which the file pointer has to position itself. The syntax for seek() is—

– f.seek(file_location) #where f is the file pointer

•For example, f.seek(20) will give the position or file number where the file pointer has been placed. This statement shall move the file pointer to 20th byte in the file no matter where you are.

Relative referencing/positioning has two arguments, offset and the position from which it has to traverse. The syntax for relative referencing is:

– f.seek(offset, from_what) #where f is file pointer, For example,

– f.seek(–10,1) from current position, move 10 bytes backward

– f.seek(10,1) from current position, move 10 bytes forward

– f.seek(–20,1) from current position, move 20 bytes backward

– f.seek(10,0) from beginning of file, move 10 bytes forward

tell()—tell() returns the current position of the file read/write pointer within the file. Its syntax is:

• f.tell() #where f is file pointer

• When we open a file in reading/writing mode, the file pointer rests at 0th byte.

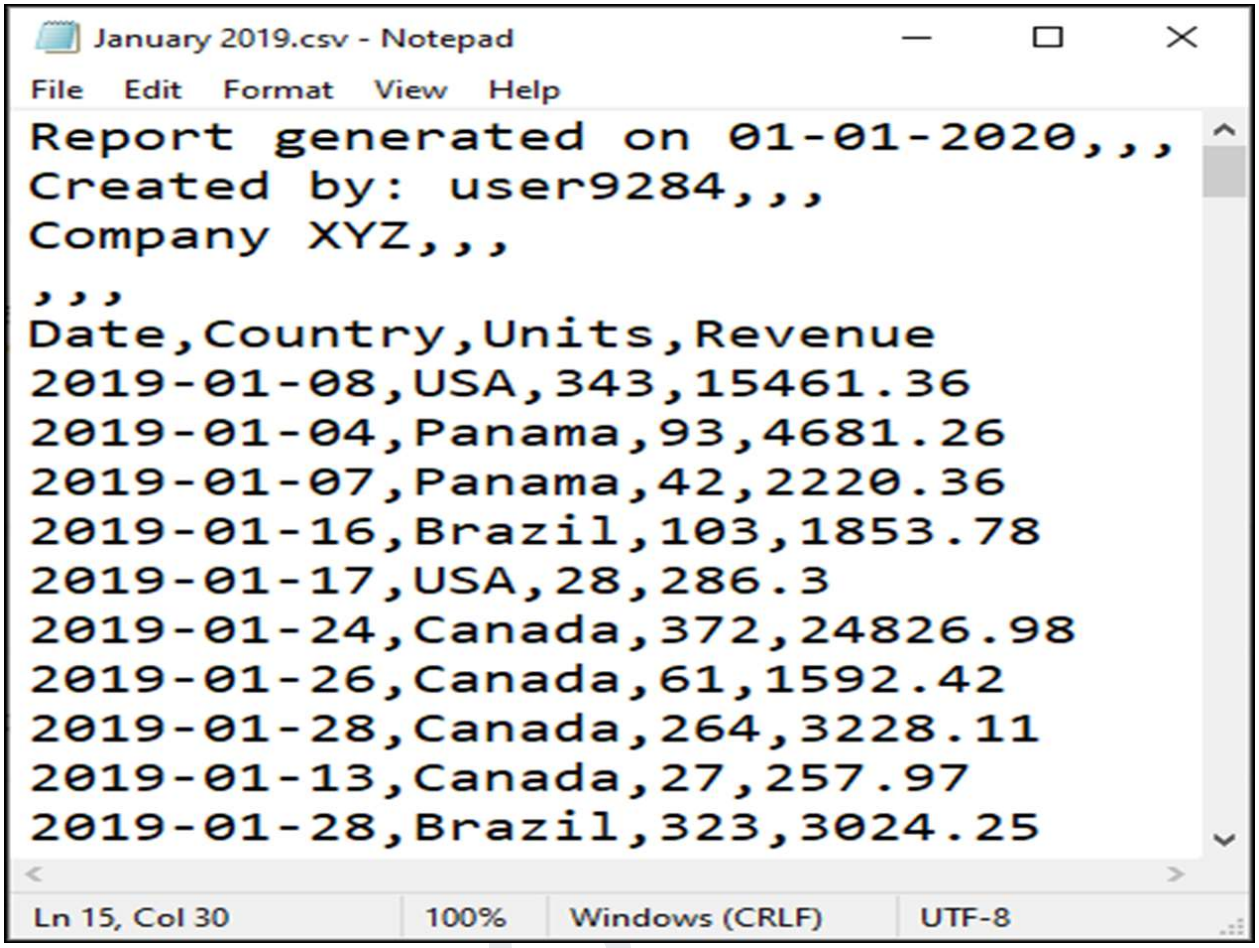
• When we open a file in append mode, the file pointer rests at the last byte.

• This is illustrated in the practical implementation that follows:

CSV File operations in Python

• 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.

• The structure of a CSV file is given away by its name. Normally, CSV files use a comma to separate each specific data value.



Normally, the first line identifies each piece of data—in other words, the name of a data column. Every subsequent line after that is actual data and is limited only by file size constraints.

• In general, the separator character is called a delimiter, and the comma is not the only one used. Other popular delimiters include the tab (\t), colon (:) and semi-colon (;) characters. 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 data (numbers and text) in plain text.

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

• CSV stands for “comma separated values”. Thus, we can say that a comma separated file is a delimited text file that uses a comma to separate values.

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 of the records is also a part of this file. Tabular data is stored as text in a CSV file. The use of comma as a field separator is the source of the name for this file format. It stores our data into a spreadsheet or a database.

WHY USE CSV?

•The extensive use of social networking sites and their various associated applications requires the handling of huge data. But the problem arises as to how to handle and organize this large unstructured data?

•The solution to the above problem is CSV. Thus, CSV organizes data into a structured form and, hence, the proper and systematic organization of this large amount of data is done by CSV. Since CSV file formats are of plain text format, it makes it very easy for website developers to create applications that implement CSV.

•the several advantages that are offered by CSV files are as follows:

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

•To perform read and write operations with CSV file, we must import CSV module. CSV module can handle CSV files correctly regardless of the operating system on which the files were created.

• Along with this module, open() function is used to open a CSV file and return file object. We load the module in the usual way using import:

– >>> import csv

Like other files (text and binary) in Python, there are two basic operations that can be carried out on a CSV file:

– 1. Reading from a CSV file

– 2. Writing to a CSV file

Reading from a CSV File:

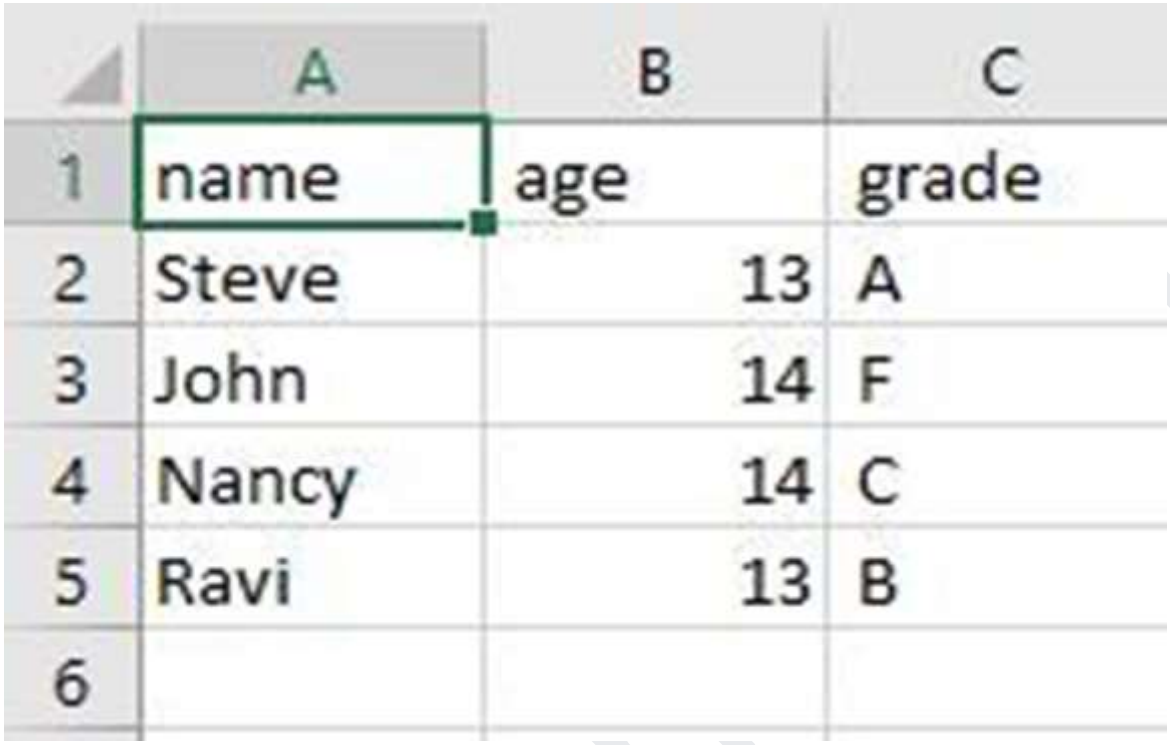
Python contains a module called csv for the handling of CSV files. The reader class from the module is used for reading data from a CSV file. At first, the CSV file is opened using the open() method in ‘r’ mode(specifies read mode while opening a file) which returns the file object then it is read by using the reader() method of CSV module that returns the reader object that iterates throughout the lines in the specified CSV document.

Syntax:

csv.reader(csvfile, dialect='excel', **fmtparams)

Example:

Consider the below CSV file –



	A	B	C
1	name	age	grade
2	Steve	13	A
3	John	14	F
4	Nancy	14	C
5	Ravi	13	B
6			

Source Code:

```
import csv
# opening the CSV file
with open('Giants.csv', mode ='r')as file:
# reading the CSV file
csvFile = csv.reader(file)
# displaying the contents of the CSV file
for lines in csvFile:
  print(lines)
```

Output:

```
[['Steve', 13, 'A'],
['John', 14, 'F'],
['Nancy', 14, 'C'],
```

['Ravi', 13, 'B']]

Writing to 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.

Syntax:

csv.writer(csvfile, dialect='excel', **fmtparams)

csv.writer class provides two methods for writing to CSV. They are writerow() and writerows().

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

•

Syntax:

writerow(fields)

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

Syntax:

writerows(rows)

Example:

```
import csv
fields = ['Name', 'Branch', 'Year', 'CGPA']
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']]
filename = "university_records.csv"
with open(filename, 'w') as csvfile:
  # creating a csv writer object
  csvwriter = csv.writer(csvfile)
  # writing the fields
```

```
csvwriter.writerow(fields)
# writing the data rows
csvwriter.writerows(rows)
```

Output:

	A	B	C	D	E
1	Name	Branch	Year	CGPA	
2	Nikhil	COE	2	9	
3	Sanchit	COE	2	9.1	
4	Aditya	IT	2	9.3	
5	Sagar	SE	1	9.5	
6	Prateek	MCE	3	7.8	
7	Sahil	EP	2	9.1	
8					

We can also write dictionary to the CSV file. For this the CSV module provides the csv.DictWriter class. This class returns a writer object which maps dictionaries onto output rows.

Syntax:

csv.DictWriter(csvfile, fieldnames, restval=”, extrasaction=’raise’, dialect=’excel’, *args, **kwds)

csv.DictWriter provides two methods for writing to CSV. They are:

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

Syntax:

writeheader()

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

Syntax:

writerows(mydict)

Example:

```
import csv
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'}]

fields = ['name', 'branch', 'year', 'cgpa']
filename = "university_records.csv"
with open(filename, 'w') as csvfile:
  writer = csv.DictWriter(csvfile, fieldnames = fields)
  writer.writeheader()
  writer.writerows(mydict)
```

Output:

	A	B	C	D
1	Name	Branch	Year	CGPA
2	Nikhil	COE	2	9
3	Sanchit	COE	2	9.1
4	Aditya	IT	2	9.3
5	Sagar	SE	1	9.5
6	Prateek	MCE	3	7.8
7	Sahil	EP	2	9.1
8				