1. What advantages do Excel spreadsheets have over CSV spreadsheets?

Ans-:**CSV file can**'t perform operations **on** data while **Excel can** perform operations **on** the data. **CSV files** are faster and also consumes less memory whereas **Excel** consumes more memory while importing data. **CSV files can** be opened with any text editor **in** windows while **Excel files can**'t be opened with text editors.

The **difference between CSV** and XLS **file** formats is that **CSV** format is a plain text format in which values are separated by commas (Comma Separated Values), while XLS **file** format is an **Excel Sheets** binary **file** format which holds information about all the **worksheets in a file**, including both content and formatting.

2. To construct reader and writer artefacts, what do you pass to csv.reader() and csv.writer()?

The [csv](https://docs.python.org/3/library/csv.html#module-csv) module’s [reader](https://docs.python.org/3/library/csv.html#csv.reader) and [writer](https://docs.python.org/3/library/csv.html#csv.writer) objects read and write sequences. Programmers can also read and write data in dictionary form using the [DictReader](https://docs.python.org/3/library/csv.html#csv.DictReader) and [DictWriter](https://docs.python.org/3/library/csv.html#csv.DictWriter) classes.

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

>>> import csv

>>> with open('eggs.csv', newline='') as csvfile:

... spamreader = csv.reader(csvfile, delimiter=' ', quotechar='|')

... for row in spamreader:

... print(', '.join(row))

Spam, Spam, Spam, Spam, Spam, Baked Beans

Spam, Lovely Spam, Wonderful Spam

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

import csv

with open('eggs.csv', 'w', newline='') as csvfile:

spamwriter = csv.writer(csvfile, delimiter=' ',

quotechar='|', quoting=csv.QUOTE\_MINIMAL)

spamwriter.writerow(['Spam'] \* 5 + ['Baked Beans'])

spamwriter.writerow(['Spam', 'Lovely Spam', 'Wonderful Spam'])

1. In what modes can File artefacts for readers and writers be opened?

Ans-:**There are many modes for opening a file:**

* r - **open** a **file** in read **mode**.
* w - opens or create a text **file** in write **mode**.
* a - opens a **file** in append **mode**.
* r+ - opens a **file** in both read and write **mode**.
* a+ - opens a **file** in both read and write **mode**.
* w+ - opens a **file** in both read and write **mod**

|  |
| --- |
| # Append vs write mode  file1 = open("myfile.txt","w")  L = ["This is Delhi \n","This is Paris \n","This is London \n"]  file1.close()    # Append-adds at last  file1 = open("myfile.txt","a")#append mode  file1.write("Today \n")  file1.close()    file1 = open("myfile.txt","r")  print "Output of Readlines after appending"  print file1.readlines()  print  file1.close()    # Write-Overwrites  file1 = open("myfile.txt","w")#write mode  file1.write("Tomorrow \n")  file1.close()    file1 = open("myfile.txt","r")  print "Output of Readlines after writing"  print file1.readlines()  print  file1.close() |

* Output:
* Output of Readlines after appending
* ['This is Delhi \n', 'This is Paris \n', 'This is London \n', 'Today \n']
* Output of Readlines after writing
* ['Tomorrow \n']

1. Which method outputs a CSV file from a list argument?

Ans-:Python provides an in-built module called csv to work with CSV files. There are various classes provided by this module for writing to CSV:

* Using csv.writer class
* Using csv.DictWriter class

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:**

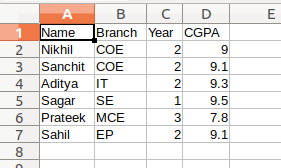
Writing CSV files in Python

writerows(rows)

**Ex-:**

|  |
| --- |
| 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']]    # name of csv file  filename = "university\_records.csv"    # writing to csv file  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:**



1. What do the keyword arguments delimiter and lineterminator do?

Ans-: You **can** change characters to different values by using the **delimiter and lineterminator keyword arguments** with **csv**. ... Passing delimeter='\t' and **lineterminator**='\n\n' ❶ changes the character between cells to a tab and the character between rows to two newlines.

>>> **import csv**

>>> **csvFile = open('example.tsv', 'w', newline='')**

❶ >>> **csvWriter = csv.writer(csvFile, delimiter='\t', lineterminator='\n\n')**

>>> **csvWriter.writerow(['apples', 'oranges', 'grapes'])**

24

>>> **csvWriter.writerow(['eggs', 'bacon', 'ham'])**

17

>>> **csvWriter.writerow(['spam', 'spam', 'spam', 'spam', 'spam', 'spam'])**

32

>>> **csvFile.close()**

1. What function returns a Python data structure from a string of JSON data?

Ans-: loads() **method return Python data structure** of **JSON string** or **data**. To translate a string containing JSON data into a Python value, pass it to the json.loads() function. (The name means “load string,” not “loads.”)

>>> **stringOfJsonData = '{"name": "Zophie", "isCat": true, "miceCaught": 0,**

**"felineIQ": null}'**

>>> **import json**

>>> **jsonDataAsPythonValue = json.loads(stringOfJsonData)**

>>> **jsonDataAsPythonValue**

{'isCat': True, 'miceCaught': 0, 'name': 'Zophie', 'felineIQ': None}

1. What function returns a string of JSON data from a Python data structure?

Ans-: After you import the json module, you can call loads() and pass it a string of JSON data. Note that JSON strings always use double quotes. It will return that data as a Python dictionary. Python dictionaries are not ordered, so the key-value pairs may appear in a different order when you print .