**FSDS MAY BATCH 2022(Python Basics 13)**

**Submitted by: Shubham Tiwari**

Q1. What advantages do Excel spreadsheets have over CSV spreadsheets?

Ans: Excel spreadsheets have several advantages over CSV spreadsheets, including:

1. **Graphical User Interface:** Excel provides a graphical user interface (GUI) that makes it easier to work with data and perform tasks such as data entry, sorting, and formatting.
2. **Advanced Functions and Formulas**: Excel offers a wide range of advanced functions and formulas for data analysis and manipulation, which are not available in CSV spreadsheets.
3. **Built-in Charts and Graphs**: Excel provides built-in charts and graphs, allowing you to quickly visualize data and draw insights from it.
4. **Data Validation:** Excel has data validation features that allow you to ensure that data entered into a spreadsheet is accurate and consistent.
5. **Collaboration:** Excel spreadsheets can be easily shared and collaborated on, making it easier to work on projects with others.
6. **Formatting Options:** Excel provides a range of formatting options, including conditional formatting, cell borders, and font styles, making it easier to present data in a clear and visually appealing way.

Q2. What do you pass to csv.reader() and csv.writer() to create reader and writer objects?

Ans: In the Python Standard Library's **csv** module, we pass a file object to both **csv.reader()** and **csv.writer()** to create reader and writer objects, respectively.

Example:

**import csv**

**with open("file.csv", "r") as file:**

**reader = csv.reader(file)**

**for row in reader:**

**print(row)**

The example of writer object is as follows:

**import csv**

**data = [["Name", "Age"], ["Shubham", "30"], ["Shubham", "28"]]**

**with open("file.csv", "w", newline="") as file:**

**writer = csv.writer(file)**

**writer.writerows(data)**

When creating the writer object, it's a good practice to open the file in write mode (**"w"**), and to specify the newline parameter as an empty string (**newline=""**) to avoid issues with newline characters on different platforms.

Q3. What modes do File objects for reader and writer objects need to be opened in?

Ans: File objects for reading need to be opened in "r" (read) mode, and file objects for writing need to be opened in "w" (write) mode. In read mode, the file pointer is positioned at the beginning of the file, and you can only read from the file. In write mode, the file is truncated to 0 length or created if it does not exist, and you can only write to the file. There are other modes available like "a" (append) mode, "r+" (read and write) mode, etc. depending on the specific requirements of your use case.

Q4. What method takes a list argument and writes it to a CSV file?

Ans: In the Python Standard Library's **csv** module, the **writerows()** method takes a list of lists (rows) as an argument and writes it to a CSV file. The **writerows()** method is part of the **csv.writer** object.

Example:

**import csv**

**data = [["Name", "Age"], ["Shubham", "32"], ["Shubham", "29"]]**

**with open("file.csv", "w", newline="") as file:**

**writer = csv.writer(file)**

**writer.writerows(data)**

In this example, the **writerows()** method takes the **data** list as an argument and writes it to the file **file.csv**. Each sublist in **data** represents a row in the CSV file.

Q5. What do the keyword arguments delimiter and line terminator do?

Ans: In the Python Standard Library's **csv** module, the **delimiter** and **line terminator** keyword arguments are used when creating a **csv.writer** object.

The **delimiter** argument specifies the character used to separate the fields (columns) in each row of the CSV file. The default delimiter is a comma (**,**).

The **line terminator** argument specifies the character or string used to separate rows in the CSV file. The default line terminator is the newline character (**\n**) for most platforms.

An example of using the **delimiter** and **line terminator** arguments:

**import csv**

**data = [["Name", "Age"], ["John Doe", "32"], ["Shubham", "29"]]**

**with open("file.csv", "w") as file:**

**writer = csv.writer(file, delimiter="\t", lineterminator="\r\n")**

**writer.writerows(data)**

In this example, the delimiter is set to a tab character (‘**\t’**) and the line terminator is set to a carriage return followed by a newline character (‘**\r\n’**). These characters will be used to separate fields and rows in the output CSV file.

Q6. What function takes a string of JSON data and returns a Python data structure?

Ans: In Python, the **json** module provides a function called **json.loads()** that takes a string of JSON data and returns a Python data structure (such as a dictionary, list, or string).

An example of using **json.loads()**:

**import json**

**json\_data = '{"name": "Shubham", "age": 32, "city": "Jhansi"}'**

**data = json.loads(json\_data)**

**print(type(data))**

**# Output: <class 'dict'>**

**print(data)**

# Output: {'name': 'Shubham, 'age': 32, 'city':’Jhansi '}

In this example, **json.loads()** takes the **json\_data** string as an argument and returns a dictionary, which can be accessed like any other dictionary in Python.

Q7. What function takes a Python data structure and returns a string of JSON data?

Ans: In Python, the **json** module provides a function called **json.dumps()** that takes a Python data structure (such as a dictionary, list, or string) and returns a string of JSON data.