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

Excel spreadsheets have several advantages over CSV spreadsheets:

* **Formatting and Styling**
* **Formulas and Functions**
* **Charts and Graphs**
* **Multiple Sheets**
* **Data Validation**

CSV spreadsheets are generally simpler and more lightweight than Excel spreadsheets, and can be easily imported and exported to other applications. However, they lack many of the advanced features and formatting options that Excel offers.

Top of Form

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

**To create reader** and **writer objects** using the **csv module** in Python, you pass a **file object** and **optionally other parameters** to the csv.reader() and csv.writer() functions.

For **example**, **to create a reader object** for a CSV file named "example.csv", you can do:

**import csv**

**with open('example.csv', 'r') as csv\_file:**

**reader = csv.reader(csv\_file)**

Here**, csv\_file is the file object representing the CSV file**, and csv.reader() is called with csv\_file as the first argument. Additional parameters can be passed to csv.reader() to customize the parsing behavior.

**Similarly, to create a writer object** for a CSV file named "output.csv", you can do:

**import csv**

**with open('output.csv', 'w') as csv\_file:**

**writer = csv.writer(csv\_file)**

Here, **csv\_file is the file object representing the output CSV file**, and csv.writer() is called with csv\_file as the first argument. Additional parameters can be passed to csv.writer() to customize the formatting behavior.

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

For a **reader object**, the corresponding file object needs to be opened in **"read" mode ("r").**

For a **writer object**, the corresponding file object needs to be opened in **"write" mode ("w")** or **"append" mode ("a").**

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

The **writerows() method** of a **csv.writer object** takes a list of rows as an argument and writes them to a CSV file. Each row in the list should itself be a list or tuple containing the values to be written to the CSV file. For **example**:

**import csv**

**rows = [**

**['name', 'age', 'gender'],**

**['Alice', '25', 'F'],**

**['Bob', '30', 'M'],**

**['Charlie', '35', 'M']**

**]**

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

**writer = csv.writer(csvfile)**

**writer.writerows(rows)**

In this example, writerows() writes a list of rows to a file named example.csv. Each row is a list containing three values, representing the name, age, and gender of a person. The `**newline=''** `argument is passed to the open() function to **ensure** that the CSV file is written with the **correct line endings** on all platforms.{otherwise default is `\n`}

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

In the context of the `**csv**` module, the `**delimiter**` keyword argument is used to **specify the character used to separate the fields in a CSV file**. By **default**, a **comma (,)** is used as the delimiter, but it can be changed to any character that is appropriate for the data being processed.

The `**lineterminator**` keyword argument is used to **specify the string used to terminate lines** when writing a CSV file. By **default**, the **newline character (\n)** is used as the line terminator. However, it can be changed to any other string or character that is appropriate for the data being processed.

For **example**, if you want to use a **tab** character as the **delimiter** and a **semicolon** as the line **terminator**, you could use the **following code** to write to a CSV file:

**import csv**

**with open('example.csv', 'w', newline='', encoding='utf-8') as file:**

**writer = csv.writer(file, delimiter='\t', lineterminator=';')**

**writer.writerow(['Name', 'Age', 'Gender']) writer.writerow(['John', '35', 'Male']) writer.writerow(['Jane', '28', 'Female'])**

This **would create a CSV file** with **tab-separated** values and **semicolon-separated lines**.

{{{both newline and lineterminator can be used to control line endings in a CSV file. However, they work in slightly different ways.

newline is a parameter used when opening a file for reading or writing, which determines how the file handles line endings. When writing to a CSV file, for example, newline can be set to an empty string to indicate that the file should use the system's default line endings. Alternatively, newline can be set to "\n" to force the file to use Unix-style line endings, or "\r\n" to use Windows-style line endings.

lineterminator, on the other hand, is an optional parameter that can be passed to a csv.writer object to specify the character used to terminate each row of data. By default, lineterminator is set to "\r\n", which corresponds to Windows-style line endings. However, you can set it to any other string, such as "\n" or "\r", depending on your needs.

In summary, newline controls how the file handles line endings globally, while lineterminator specifies the line ending character on a row-by-row basis.}}}

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

The `**json.loads()`** function takes a string of JSON data and returns a Python data structure. The function is part of the built-in `**json**` module in Python.

Here's an **example**:

**import json**

**json\_string = '{"name": "John", "age": 30, "city": "New York"}'**

**python\_data = json.loads(json\_string) print(python\_data)** # Output: {'name': 'John', 'age': 30, 'city': 'New York'}

**print(type(python\_data))** # Output: <class 'dict'>

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

The **`json.dumps()`** function is used to take a Python data structure and return a string of JSON data. The dumps() function stands for "dump string" and is part of the `**json**` module in Python's standard library.

Here's an **example**:

**import json**

**data = { "name": "John", "age": 30, "city": "New York" }**

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

**print(json\_data)**

Output:

**{"name": "John", "age": 30, "city": "New York"}**

In this example, the `json.dumps()` function is used to convert the Python dictionary data to a string of JSON data `json\_data`.