**Assignment 13**

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

Excel spreadsheets have several advantages over CSV spreadsheets, including:

1. Formatting: Excel allows users to apply formatting to cells, such as font styles, colors, and conditional formatting. This can make it easier to read and interpret data.
2. Formulas and Functions: Excel allows users to perform calculations and analysis using formulas and functions. These can be used to automate repetitive tasks, perform complex calculations, and generate reports.
3. Charts and Graphs: Excel has built-in charting tools that allow users to create visual representations of their data. This can help to identify trends and patterns in the data.
4. Macros: Excel allows users to create macros, which are sets of instructions that automate tasks. This can help to save time and reduce errors.
5. Security: Excel has built-in security features that allow users to control who can access and modify the data in the spreadsheet. This can help to protect sensitive information.

CSV, on the other hand, is a simpler format that only stores data values separated by commas. While it is more lightweight and easier to share between different software applications, it does not have the advanced functionality of Excel.

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.reader()** and **csv.writer()** functions, you need to pass a file object as the first argument.

For example, to create a reader object, you can pass a file object containing CSV data to **csv.reader()**, like this:

pythonCopy code

import csv with open('example.csv', 'r') as file: reader = csv.reader(file) for row in reader: print(row)

In this example, **example.csv** is the name of the file containing CSV data. The **csv.reader()** function creates a reader object, which can be used to read the rows in the CSV file.

To create a writer object, you can pass a file object and a list of field names as arguments to **csv.writer()**, like this:

pythonCopy code

import csv with open('example.csv', 'w', newline='') as file: writer = csv.writer(file) writer.writerow(['Name', 'Age', 'Gender']) writer.writerow(['John', '25', 'Male']) writer.writerow(['Jane', '30', 'Female'])

In this example, **example.csv** is the name of the file to be created. The **csv.writer()** function creates a writer object, which can be used to write rows to the CSV file. The **writerow()** method is used to write a single row to the file. The **newline=''** argument is used to ensure that the file uses the correct line endings for the operating system being used.

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

For reader and writer objects in **csv** module, File objects need to be opened in different modes.

When creating a reader object using the **csv.reader()** function, the file object should be opened in **'r'** mode to indicate that it is being opened for reading:

pythonCopy code

import csv with open('example.csv', 'r') as file: reader = csv.reader(file) for row in reader: print(row)

On the other hand, when creating a writer object using the **csv.writer()** function, the file object should be opened in **'w'** mode to indicate that it is being opened for writing:

pythonCopy code

import csv with open('example.csv', 'w', newline='') as file: writer = csv.writer(file) writer.writerow(['Name', 'Age', 'Gender']) writer.writerow(['John', '25', 'Male']) writer.writerow(['Jane', '30', 'Female'])

In this example, **'w'** mode is used to open the file for writing, and the **newline=''** argument is used to ensure that the file uses the correct line endings for the operating system being used.

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

The **csv.writer()** method's **writerow()** is used to write a list argument to a CSV file.

The **writerow()** method takes an iterable (e.g., list, tuple, etc.) as an argument and writes it to the CSV file as a single row. Each element in the iterable is separated by a delimiter, which is a comma by default but can be changed by passing the **delimiter** argument to **csv.writer()**.

Here's an example that writes a list to a CSV file using the **writerow()** method:

kotlinCopy code

import csv data = [['Name', 'Age', 'Gender'], ['John', '25', 'Male'], ['Jane', '30', 'Female']] with open('example.csv', 'w', newline='') as file: writer = csv.writer(file) for row in data: writer.writerow(row)

In this example, the **data** variable contains a list of lists, where each inner list represents a row of data. The **csv.writer()** method is used to create a writer object, and the **writerow()** method is called on each row of data to write it to the CSV file.

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

The keyword arguments **delimiter** and **lineterminator** are used in the **csv.writer()** function to specify the character that should be used to separate the fields and to terminate the lines in the output file, respectively.

* **delimiter**: The **delimiter** argument is used to specify the character that separates the fields in the CSV file. By default, it is a comma (**,**), but it can be changed to any other character, such as a tab (**\t**) or a semicolon (**;**). For example, to use a tab as the delimiter, you can specify the **delimiter** argument as follows:

kotlinCopy code

import csv data = [['Name', 'Age', 'Gender'], ['John', '25', 'Male'], ['Jane', '30', 'Female']] with open('example.csv', 'w', newline='') as file: writer = csv.writer(file, delimiter='\t') for row in data: writer.writerow(row)

In this example, the **csv.writer()** function is used to create a writer object with a tab delimiter, and the **writerow()** method is called on each row of data to write it to the CSV file using the tab as the field separator.

* **lineterminator**: The **lineterminator** argument is used to specify the character that terminates each line in the CSV file. By default, it is the operating system's newline character, but it can be changed to any other character, such as a space () or a semicolon (**;**). For example, to use a semicolon as the line terminator, you can specify the **lineterminator** argument as follows:

kotlinCopy code

import csv data = [['Name', 'Age', 'Gender'], ['John', '25', 'Male'], ['Jane', '30', 'Female']] with open('example.csv', 'w', newline='') as file: writer = csv.writer(file, lineterminator=';') for row in data: writer.writerow(row)

In this example, the **csv.writer()** function is used to create a writer object with a semicolon line terminator, and the **writerow()** method is called on each row of data to write it to the CSV file using the semicolon as the line terminator.

6. 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 **loads()** function is part of the **json** module, which provides methods for encoding and decoding JSON data.

Here's an example that uses the **json.loads()** function to convert a JSON string to a Python data structure:

makefileCopy code

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

In this example, the **json\_string** variable contains a string of JSON data. The **json.loads()** function is used to parse the JSON string and return a Python dictionary object. The resulting **python\_data** variable contains the Python data structure equivalent of the original JSON string.

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

The **json.dumps()** function takes a Python data structure and returns a string of JSON data. The **dumps()** function is part of the **json** module, which provides methods for encoding and decoding JSON data.

Here's an example that uses the **json.dumps()** function to convert a Python dictionary object to a string of JSON data:

goCopy code

import json python\_data = {"name": "John", "age": 30, "city": "New York"} json\_string = json.dumps(python\_data) print(json\_string)

In this example, the **python\_data** variable contains a Python dictionary object. The **json.dumps()** function is used to encode the dictionary object as a JSON string. The resulting **json\_string** variable contains the JSON data equivalent of the original Python dictionary.