**Python Assignment 13**

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

Ans. Excel spreadsheets have several advantages over CSV spreadsheets:

**Formatting**: - Excel allows you to format your data and make it more readable, whereas CSV files only contain raw data. You can change the font, color, size, and other formatting options in Excel, which can make it easier to interpret the data.

**Calculation**: - Excel allows you to perform calculations and data analysis using built-in functions, charts, and graphs. You can easily calculate totals, averages, and other statistics with Excel, which can save you time and effort.

**Multiple** **sheets**: - Excel allows you to have multiple sheets in a single workbook, making it easier to organize and manage your data. You can also link data between sheets and perform calculations across multiple sheets.

**Macros**: - Excel allows you to automate repetitive tasks using macros. Macros can save you time and effort by performing tasks automatically, such as formatting, sorting, and analyzing data.

**Protection**: - Excel allows you to protect your data by setting passwords, restricting editing, and controlling access to your workbook. This can help keep your data secure and prevent unauthorized access or changes.

In summary, Excel offers a range of features and functionality that can make it easier to work with and analyze data compared to CSV files. However, CSV files are a simpler, more lightweight format that can be useful in certain situations where you just need to quickly import or export data.

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

Ans. To create reader and writer objects in Python's csv module, you need to pass a file object to the csv.reader() and csv.writer() functions.

For example, to create a reader object, you can pass a file object that contains CSV data to the csv.reader() function:

import csv

with open('data.csv', 'r') as file:

reader = csv.reader(file)

# Perform operations with the reader object

In the example above, we opened the file data.csv in read mode and passed it to the csv.reader() function, which created a reader object named reader.

Similarly, to create a writer object, you can pass a file object that you want to write CSV data to:

import csv

with open('output.csv', 'w', newline='') as file:

writer = csv.writer(file)

# Perform operations with the writer object

In the example above, we opened the file output.csv in write mode and passed it to the csv.writer() function, which created a writer object named writer. We also included the newline='' parameter to avoid issues with line endings when writing CSV data to the file.

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

Ans. When working with reader and writer objects in Python's csv module, you need to open the file objects in the appropriate mode.

To create a reader object, you should open the file object in read mode ('r') to read the CSV data from the file:

import csv

with open('data.csv', 'r') as file:

reader = csv.reader(file)

# Perform operations with the reader object

Similarly, to create a writer object, you should open the file object in write mode ('w') to write the CSV data to the file:

import csv

with open('output.csv', 'w', newline='') as file:

writer = csv.writer(file)

# Perform operations with the writer object

Note that we also included the newline='' parameter when opening the file in write mode to ensure that the correct line endings are used when writing CSV data.

It's important to use the correct mode when opening file objects for reader and writer objects in the csv module. If you open the file in the wrong mode, you may encounter errors or unexpected behavior when working with the CSV data.

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

Ans. To write a list of data to a CSV file in Python's csv module, you can use the writerow() method of a csv.writer object.

The writerow() method takes a list argument containing the values you want to write to the CSV file, where each element of the list corresponds to a column in the CSV file.

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

import csv

data = [['Name', 'Age', 'City'], ['John', '30', 'New York'], ['Jane', '25', 'Los Angeles'], ['Bob', '45', 'Chicago']]

with open('output.csv', 'w', newline='') as file:

writer = csv.writer(file)

for row in data:

writer.writerow(row)

In this example, we first defined a list data containing the data we want to write to the CSV file. We then opened a file object named file in write mode and passed it to the csv.writer() function to create a writer object named writer. Finally, we used a for loop to iterate over each row in the data list and wrote each row to the CSV file using the writerow() method.

After running this code, the data in the data list will be written to the output.csv file in CSV format.

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

Ans. In Python's csv module, you can use the delimiter and lineterminator keyword arguments to customize the formatting of the CSV output. The delimiter argument specifies the character to use as the delimiter between fields in the CSV file. By default, the delimiter is a comma (,), but you can change it to any other character, such as a semicolon (;) or a tab (\t). Here's an example that uses a semicolon as the delimiter:

import csv

data = [['Name', 'Age', 'City'], ['John', '30', 'New York'], ['Jane', '25', 'Los Angeles'], ['Bob', '45', 'Chicago']]

with open('output.csv', 'w', newline='') as file:

writer = csv.writer(file, delimiter=';')

for row in data:

writer.writerow(row)

In this example, we passed the delimiter=';' argument to the csv.writer() function to specify that we want to use a semicolon as the delimiter between fields in the CSV file.

The lineterminator argument specifies the character to use as the line terminator between rows in the CSV file. By default, the line terminator is a carriage return (\r) followed by a line feed (\n), but you can change it to any other character, such as a newline (\n) or a space ( ). Here's an example that uses a newline as the line terminator:

import csv

data = [['Name', 'Age', 'City'], ['John', '30', 'New York'], ['Jane', '25', 'Los Angeles'], ['Bob', '45', 'Chicago']]

with open('output.csv', 'w', newline='') as file:

writer = csv.writer(file, lineterminator='\n')

for row in data:

writer.writerow(row)

In this example, we passed the lineterminator='\n' argument to the csv.writer() function to specify that we want to use a newline as the line terminator between rows in the CSV file.

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

Ans. To convert a string of JSON data into a Python data structure, you can use the json.loads() function from Python's built-in json module.

The json.loads() function takes a single argument, which is the JSON string to be converted into a Python data structure. The function returns a Python data structure such as a dictionary, list, or string, depending on the contents of the JSON data.

Here's an example that shows how to use json.loads() to convert a JSON string into a Python dictionary:

import json

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

python\_dict = json.loads(json\_string)

print(python\_dict)

In this example, we first defined a JSON string containing some data in the form of a dictionary. We then passed the JSON string to the json.loads() function to convert it into a Python dictionary named python\_dict. Finally, we printed the contents of the python\_dict dictionary to verify that it was converted correctly.

The output of this code will be:

{'name': 'John', 'age': 30, 'city': 'New York'}

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

Ans. To convert a Python data structure into a string of JSON data, you can use the json.dumps() function from Python's built-in json module.

The json.dumps() function takes a single argument, which is the Python data structure to be converted into a JSON string. The function returns a JSON string that represents the contents of the Python data structure.

Here's an example that shows how to use json.dumps() to convert a Python dictionary into a JSON string:

import json

python\_dict = {'name': 'John', 'age': 30, 'city': 'New York'}

json\_string = json.dumps(python\_dict)

print(json\_string)

In this example, we first defined a Python dictionary containing some data. We then passed the dictionary to the json.dumps() function to convert it into a JSON string named json\_string. Finally, we printed the contents of the json\_string variable to verify that it was converted correctly.

The output of this code will be:

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