**Python Assignment 13**

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

Excel spreadsheets have several advantages over CSV (Comma-Separated Values) spreadsheets:

1. Formatting: Excel allows for a wide range of formatting options, such as font styles, colors, borders, and cell alignment. This can make it easier to read and interpret data, especially when dealing with complex datasets.
2. Formulas and Functions: Excel has a large library of built-in formulas and functions that allow for easy calculations and analysis of data. These can be used to perform mathematical operations, statistical analysis, and more.
3. Charts and Graphs: Excel has a variety of chart and graph options that allow for visual representation of data. This can be helpful in identifying trends, patterns, and relationships within the data.
4. Data Validation: Excel has built-in data validation features that allow users to restrict the type of data that can be entered into a cell. This can help ensure data accuracy and consistency.
5. Data Protection: Excel allows users to password protect their spreadsheets and control access to specific cells or ranges of data. This can help ensure data security and prevent unauthorized changes to the data.

CSV files are simple and easy to create, Excel spreadsheets offer a wider range of features and functionality for managing and analyzing data.

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

import csv

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

reader = csv.reader(file)

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

Note that in the second example, we also specify the **newline****=''** argument to the **open()** function, which is required when working with CSV files on Windows systems. This is because Windows uses a different line ending character sequence (**\r\n**) than

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

When working with CSV files in Python using the csv.reader() and csv.writer() functions, you need to open the file objects in different modes depending on whether you are reading from or writing to the file.

import CSV

With open(‘data.csv’,’r’) as file:

reader=csv.reader(file)

With open(‘data2.csv’,’w’) as file:

writer=csv.writer(file)

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

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.

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

The **delimiter** keyword argument specifies the character used to separate fields within each row. By default, the csv module in Python assumes that fields are separated by commas (','). However, you can specify a different delimiter character by passing the **delimiter** argument to the csv.reader() or csv.writer() functions. For example, to use a tab character ('\t') as the delimiter, you would do:

import csv  
with open('data.tsv', 'r') as file:  
 reader = csv.reader(file, delimiter='\t')  
 # Now you can iterate over the reader object to read data from the file

The **line terminator** keyword argument specifies the character(s) used to separate rows within the file. By default, the csv module in Python assumes that rows are separated by newline characters ('\n'). However, you can specify a different line terminator character by passing the **lineterminator** argument to the csv.writer() function. For example, to use a carriage return character ('\r') as the line terminator, you would do:

import csv  
  
data = [  
 ['John', 'Doe'],  
 ['Jane', 'Smith'],  
 ['Bob', 'Johnson']  
]  
  
with open('output.csv', 'w', newline='') as file:  
 writer = csv.writer(file, lineterminator='\r\n')  
 writer.writerows(data)

**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 corresponding Python data structure. The **json** module in Python provides this and other functions for working with JSON data.

import json

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

python\_data = json.loads(json\_data)

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

inpurt

data =

{ "name": "John",

"age": 30,

"city": "New York"

}

The **json.dumps()** function takes a Python data structure and returns a string of JSON data. The **json** module in Python provides this and other functions for working with JSON data.

The **json.dumps()** function converts a Python data structure, typically a dictionary or a list, into a JSON-formatted string.