**Python Assignment 13**

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

Excel spreadsheets (in the XLSX format) have several advantages over CSV (Comma-Separated Values)

spreadsheets:

Structured Data and Formatting: Excel spreadsheets support structured data with multiple sheets, formulas, cell formatting (such as fonts, colors, and borders), cell merging, conditional formatting, and more. This allows for more complex and visually appealing layouts.

Rich Data Types: Excel supports a wide range of data types, including numbers, dates, times, text, and even specialized types like currency, percentages, and scientific notation. This can be helpful for maintaining data integrity and performing calculations.

Formulas and Functions: Excel provides a powerful set of built-in functions and formulas for performing calculations, aggregations, and data transformations. Formulas can reference cells across different sheets and perform complex calculations.

Data Validation: Excel allows you to define data validation rules for cells, ensuring that data entered meets specific criteria, such as ranges, lists, or custom formulas.

Charts and Graphs: Excel supports the creation of various types of charts and graphs, enabling visual representation of data for analysis and reporting.

Filtering and Sorting: Excel provides built-in tools for filtering and sorting data, making it easy to analyze and explore datasets.

Data Collaboration: Excel allows multiple users to collaborate on a single spreadsheet, tracking changes, adding comments, and sharing through cloud services.

Data Protection: Excel offers password protection and encryption for sensitive data, helping to secure your spreadsheets.

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 in Python's csv module, you need to pass file objects that are opened in the appropriate modes. Specifically:

csv.reader():

To create a reader object, you pass an open file object that is being read from. This object should be opened in text mode ('rt' or 'r') to read CSV content.

import csv

with open('data.csv', 'rt') as csv\_file:

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

csv.writer():

To create a writer object, you pass an open file object that is being written to. This object should be opened in text mode ('wt' or 'w') to write CSV content.

import csv

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

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

Both csv.reader() and csv.writer() return iterable objects that allow you to read or write CSV data from or to the provided file object. You can then iterate over the reader object to read rows from the CSV file, and use the writer object to write rows to the CSV file.

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

File objects for reader and writer objects in the csv module need to be opened in specific text modes to correctly read from and write to CSV files:

Reader Object:

When working with a reader object, the file object should be opened in read text mode ('rt' or 'r') to read data from the CSV file.

import csv

with open('data.csv', 'rt') as csv\_file:

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

for row in csv\_reader:

print(row)

Writer Object:

For a writer object, the file object should be opened in write text mode ('wt' or 'w') to write data to the CSV file.

import csv

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

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

csv\_writer.writerow(['Name', 'Age', 'City'])

csv\_writer.writerow(['Alice', 30, 'New York'])

csv\_writer.writerow(['Bob', 25, 'Los Angeles'])

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

The writerow() method is used to write a list of values to a CSV file using a writer object in the csv module. This method takes a single argument, which should be a list of values representing a row of data. Each value in the list corresponds to a column in the CSV file.

Here's an example of how to use the writerow() method to write a list to a CSV file:

import csv

data = [['Name', 'Age', 'City'],

['Alice', 30, 'New York'],

['Bob', 25, 'Los Angeles']]

with open('output.csv', 'wt', newline='') as csv\_file:

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

# Write the header row

csv\_writer.writerow(data[0])

# Write the data rows

for row in data[1:]:

csv\_writer.writerow(row)

In this example, the writerow() method is used to write both the header row and the data rows to the CSV file. The newline='' parameter is used to ensure that newlines are handled consistently across different platforms.

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

In the context of the csv module in Python, the delimiter and line terminator are keyword arguments that can be used to customize how CSV data is formatted and written.

delimiter:

The delimiter keyword argument specifies the character that separates fields (columns) within a CSV row. By default, the delimiter is a comma (,), but you can change it to a different character if needed. For example, you might use a semicolon (;) as the delimiter to create semicolon-separated values (SSV) instead of comma-separated values (CSV).

import csv

data = [['Name', 'Age', 'City'],

['Alice', 30, 'New York'],

['Bob', 25, 'Los Angeles']]

with open('output.csv', 'wt', newline='') as csv\_file:

csv\_writer = csv.writer(csv\_file, delimiter=';')

for row in data:

csv\_writer.writerow(row)

line terminator:

The line terminator keyword argument specifies the character(s) used to terminate each CSV row. By default, a newline character (\n) is used as the line terminator. However, you can customize this to use a different sequence, such as a carriage return and newline (\r\n) or just a carriage return (\r).

import csv

data = [['Name', 'Age', 'City'],

['Alice', 30, 'New York'],

['Bob', 25, 'Los Angeles']]

with open('output.csv', 'wt', newline='') as csv\_file:

csv\_writer = csv.writer(csv\_file, lineterminator='\r\n')

for row in data:

csv\_writer.writerow(row)

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

The json.loads() function in Python takes a string of JSON data and returns a Python data structure. The loads stands for "load string." It is part of the json module, which provides methods for working with JSON data.

Here's how you can use json.loads() to convert a JSON string into a Python data structure:

import json

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

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

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

print(python\_data) # Output: {'name': 'Alice', 'age': 30, 'city': 'New York'}

In this example, the json.loads() function is used to parse the JSON string into a Python dictionary. The resulting python\_data variable contains the parsed data as a Python dictionary.

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

The json.dumps() function in Python takes a Python data structure and returns a string of JSON data. The dumps stands for "dump string." It is part of the json module, which provides methods for working with JSON data.

Here's how you can use json.dumps() to convert a Python data structure into a JSON string:

import json

python\_data = {'name': 'Alice', 'age': 30, 'city': 'New York'}

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

print(type(json\_string)) # Output: <class 'str'>

print(json\_string) # Output: {"name": "Alice", "age": 30, "city": "New York"}

In this example, the json.dumps() function is used to serialize the Python dictionary python\_data into a JSON-formatted string. The resulting json\_string variable contains the JSON representation of the data.