ASSIGNMENT - 13 (PYTHON)

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

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

Rich Formatting: Excel allows for rich formatting options such as bold, italic, underline, font styles, colors, borders, and cell merging. CSV files only contain plain text data without any formatting.

Multiple Sheets: Excel workbooks can contain multiple sheets, allowing you to organize data into separate tabs within a single file. CSV files represent a single table of data and do not support multiple sheets.

Formulas and Functions: Excel supports the use of formulas and functions for calculations, data manipulation, and analysis. These formulas can be complex and can reference data from different sheets within the same workbook. CSV files contain raw data only and do not support formulas or functions.

Charts and Graphs: Excel allows you to create visual representations of data using charts and graphs. These visualizations can help in understanding data patterns and trends. CSV files do not support the creation of charts or graphs directly.

Data Validation: Excel provides features for data validation, allowing you to define rules and restrictions for data entry. This helps ensure data integrity and accuracy. CSV files do not support data validation.

Macros and VBA: Excel supports macros and Visual Basic for Applications (VBA), which allow you to automate tasks, create custom functions, and extend Excel's functionality. CSV files do not support macros or VBA.

Cell Comments and Notes: Excel allows you to add comments and notes to individual cells, providing additional context or explanations for the data. CSV files do not support cell comments.

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 from the `csv` module in Python, you pass file-like objects as arguments. Specifically:

1. csv.reader(): You pass a file-like object opened in text mode (with `'r'`) as the first argument to create a reader object. This can be a file object, a file-like object (such as StringIO), or any iterable object that returns lines of text.

Example:

import csv

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

reader = csv.reader(file)

2. csv.writer(): You pass a file-like object opened in text mode (with `'w'`) as the first argument to create a writer object. This can be a file object, a file-like object (such as StringIO), or any writable object where the CSV data will be written.

Example:

import csv

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

writer = csv.writer(file)

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

For File objects used with reader and writer objects in the csv module, the following modes are typically used:

For reader objects (csv.reader()):

File objects need to be opened in text mode ('t' or 'r') to read CSV data.

The recommended mode is 'r', which is used to open files for reading text data.

Example: open('data.csv', 'r')

For writer objects (csv.writer()):

File objects need to be opened in text mode ('t' or 'w') to write CSV data.

The recommended mode is 'w', which is used to open files for writing text data.

It's important to use 'w' to ensure that if the file already exists, it's truncated (emptied) before writing new data.

Example: open('output.csv', 'w', newline=")

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

The method that takes a list argument and writes it to a CSV file is the writerow() method of a writer object created using csv.writer().

Create a writer object using csv.writer() and pass a file object opened in write mode.

Call the writerow() method on the writer object and pass a list containing the data you want to write to the CSV file.

Example:

```
import csv

# Open a CSV file in write mode

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

writer = csv.writer(file)

# Write a list to the CSV file
```

```
writer.writerow(['Name', 'Age', 'City'])

writer.writerow(['Alice', 30, 'New York'])

writer.writerow(['Bob', 25, 'Los Angeles'])
```

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 characters used to delimit fields and terminate lines in a CSV file, respectively.

delimiter:

The delimiter keyword argument specifies the character used to separate fields in a CSV file. By default, the comma (,) is used as the delimiter.

You can specify a different delimiter character, such as a tab (\t), a semicolon (;), or a pipe (|), depending on your requirements.

Example: csv.writer(file, delimiter='\t') will use a tab character as the delimiter.

lineterminator:

The lineterminator keyword argument specifies the character or characters used to terminate lines in a CSV file.

By default, the newline character (\n) is used as the line terminator.

You can specify a different line terminator, such as \r\n for Windows-style line endings or \r for old-style Mac line endings.

Example: csv.writer(file, lineterminator='\r\n') will use Windows-style line endings.

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

The function that takes a string of JSON data and returns a Python data structure is json.loads().

Example:

```
import json

# JSON data as a string

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

# Convert JSON string to Python data structure

python_data = json.loads(json_string)

print(python_data)
```

In this example, json.loads() parses the JSON string json_string and returns a Python dictionary representing the JSON data.

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

The function that takes a Python data structure and returns a string of JSON data is json.dumps().

Example:

```
import json

# Python data structure (dictionary)

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

# Convert Python data structure to JSON string

json_string = json.dumps(python_data)

print(json_string)
```

In this example, json.dumps() serializes the Python dictionary python_data into a JSON string json_string.