Name :- Deepawali . B. Mhaisagar
Assignment no 13

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

ANS

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

1. Structured Data: Excel spreadsheets provide a structured format to organize and store data. They have multiple worksheets within a single file, allowing for the organization of data into different tabs or sheets. This facilitates better organization and management of complex data sets.

2. Formulas and Functions: Excel offers built-in formulas and functions that allow for powerful calculations, data analysis, and manipulation. These formulas enable automatic calculations, conditional formatting, data validation, and various other data processing operations. In CSV files, you would need to handle calculations and transformations manually using external tools or programming.

3. Formatting and Styling: Excel provides a wide range of formatting options, such as cell styling, font styles, colors, borders, and conditional formatting. This makes it easier to present data in a visually appealing and readable manner. CSV files, on the other hand, do not support formatting and styling options as they are plain text files.

4. Data Validation: Excel allows for the application of data validation rules, which ensure data integrity and restrict data entry to specific formats or ranges. Data validation helps maintain consistency and accuracy in the data. CSV files do not have built-in data validation features.

5. Charts and Graphs: Excel enables the creation of charts and graphs directly from the data within the spreadsheet. This visual representation of data provides a quick and intuitive way to understand trends, patterns, and relationships. In CSV files, you would need to import the data into a separate tool to create visualizations.

6. Data Collaboration: Excel spreadsheets support collaboration features, allowing multiple users to work on the same file simultaneously. Users can track changes, leave comments, and protect certain parts of the spreadsheet. This collaborative functionality is not available in CSV files.

7. Macros and Automation: Excel supports macros and automation using Visual Basic for Applications (VBA). Macros allow for the recording and playback of repetitive tasks, making it easier to automate data processing operations. CSV files do not support macros or automation.

Overall, Excel spreadsheets offer a more comprehensive set of features and functionality for data organization, analysis, visualization, and collaboration compared to CSV spreadsheets, which are primarily plain text files for simple data storage.

2. From a PdfFileReader object, how do you get a Page object for page 5?

ANS

To create reader and writer objects using the csv.reader() and csv.writer() functions from the csv module in Python, you need to pass a file object as an argument.

For csv.reader():

- Pass a file object opened in text mode with the CSV data you want to read.
- You can open the file using the open() function before passing it to csv.reader().

Example:

```
import csv



# Open the CSV file in text mode

csv_file = open('data.csv', 'r')



# Create a reader object

reader = csv.reader(csv_file)



# Now you can use the reader object to read the CSV data

for row in reader:

    print(row)
```

```
# Remember to close the file when done

csv_file.close()
```

For csv.writer():

- Pass a file object opened in text mode where you want to write the CSV data.
- You can open the file using the open() function before passing it to csv.writer().

Example:

```
import csv



# Open a file in text mode to write CSV data

csv_file = open('output.csv', 'w', newline='')



# Create a writer object

writer = csv.writer(csv_file)



# Write rows to the CSV file

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

writer.writerow(['John', 25])

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



# Remember to close the file when done

csv_file.close()
```

In both cases, you open a file using open() with the appropriate mode ('r' for reading or 'w' for writing), and then pass the file object to the respective csv.reader() or csv.writer() function to create the reader or writer object.

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

ANS

When working with file objects for reader and writer objects in the csv module, the File objects should be opened in the following modes:

Reader Object:

- The File object used for creating a reader object should be opened in text mode.
- The appropriate mode for the reader object is 'r', which stands for read mode.

Example:

```
import csv



# Open the CSV file in text mode

csv_file = open('data.csv', 'r')



# Create a reader object

reader = csv.reader(csv_file)



# Now you can use the reader object to read the CSV data

for row in reader:

    print(row)
```

```
# Remember to close the file when done

csv_file.close()
```

Writer Object:

- The File object used for creating a writer object should be opened in text mode.
- The appropriate mode for the writer object is 'w', which stands for write mode.
- Additionally, when opening the file in write mode, you should provide the newline='' parameter to ensure proper handling of line endings in CSV files.

Example:

```
import csv


# Open a file in text mode to write CSV data

csv_file = open('output.csv', 'w', newline='')


# Create a writer object

writer = csv.writer(csv_file)


# Write rows to the CSV file

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

writer.writerow(['John', 25])

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


# Remember to close the file when done

csv_file.close()
```

In both cases, it's important to open the file in text mode ('r' for reading or 'w' for writing) to ensure that the data is read or written correctly as text. Additionally, providing newline='' when opening the file in write mode helps to handle line endings properly in CSV files.

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

ANS
The writerow() method is used to write a list argument as a single row to a CSV file using the csv.writer() object.

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

```
import csv



# Open a file in text mode to write CSV data

csv_file = open('output.csv', 'w', newline='')



# Create a writer object

writer = csv.writer(csv_file)



# Write a list as a single row to the CSV file

data = ['John', 'Doe', 25, 'john.doe@example.com']

writer.writerow(data)



# Remember to close the file when done

csv_file.close()
```

In the code snippet above, we open a file named 'output.csv' in write mode and create a csv.writer() object using the opened file. We then define a list called data that represents a single row of data.

By calling writer.writerow(data), we write the elements of the data list as a single row to the CSV file. Each element of the list is written as a separate cell in the row.

Remember to close the file (csv_file.close()) once you have finished writing data to it.

Executing the code will create a CSV file named 'output.csv' with a single row containing the elements of the data list.

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

ANS

The keyword arguments `delimiter` and `line_terminator` are used in the `csv.writer()` object to control the formatting and structure of a CSV file.

1. `delimiter`:

- The `delimiter` keyword argument specifies the character used to separate individual fields or values within a row in the CSV file.

- By default, the delimiter is set to a comma (`,`), which is the standard delimiter in CSV files.

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

Example:

```
import csv

# Open a file in text mode to write CSV data

csv_file = open('output.csv', 'w', newline='')



# Create a writer object with a tab delimiter

writer = csv.writer(csv_file, delimiter='\t')
```

```
# Write rows to the CSV file

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

writer.writerow(['John', 25])

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



# Remember to close the file when done

csv_file.close()
```

In the code above, the `csv.writer()` object is created with the `delimiter='\t'` argument, which sets the delimiter to a tab character (`\t`). This means that fields within each row will be separated by tabs instead of commas. The resulting CSV file will use tabs as the delimiter.

2. `line_terminator`:

- The `line_terminator` keyword argument specifies the character(s) used to terminate each line or row in the CSV file.

- By default, the line terminator is set to the system's default line terminator (e.g., `\r\n` for Windows or `\n` for Unix-like systems).

- You can specify a specific line terminator character or sequence, such as `\r\n`, `\n`, or `\r`, depending on your requirements.

Example:

```
import csv

# Open a file in text mode to write CSV data

csv_file = open('output.csv', 'w', newline='')



# Create a writer object with a custom line terminator
```

```
writer = csv.writer(csv_file, lineterminator='\r\n')



# Write rows to the CSV file

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

writer.writerow(['John', 25])

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



# Remember to close the file when done

csv_file.close()

```


In the code above, the `csv.writer()` object is created with the `lineterminator='\r\n'` argument, which sets the line terminator to a carriage return and newline sequence (`\r\n`). This ensures that each row in the resulting CSV file ends with a carriage return and newline, regardless of the system's default line terminator.

By using the `delimiter` and `line_terminator` keyword arguments, you can customize the format and structure of the CSV file to meet your specific requirements.

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

ANS

The function that takes a string of JSON data and returns a Python data structure is `json.loads()`. It is part of the `json` module in Python's standard library and is used to deserialize JSON (JavaScript Object Notation) data.

Here's an example of how to use `json.loads()`:

```
import json
```

```
# JSON data as a string

json_data = '{"name": "John", "age": 25, "city": "New York"}'

# Deserialize the JSON data into a Python data structure

python_data = json.loads(json_data)

# Access the Python data structure

print(python_data['name'])  # Output: John

print(python_data['age'])   # Output: 25

print(python_data['city'])  # Output: New York
```

In the code snippet above, the `json_data` variable stores a string representing JSON data. By calling `json.loads(json_data)`, the JSON data is deserialized, converting it into a Python data structure, which in this case is a dictionary. The resulting Python data structure is assigned to the `python_data` variable.

Once deserialized, you can access the individual elements of the Python data structure as you would with any other Python dictionary. In the example, we print the values associated with the keys `'name'`, `'age'`, and `'city'` to demonstrate accessing the data.

Note that `json.loads()` can handle more complex JSON structures as well, including nested objects, arrays, and various data types supported by JSON.

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

ANS

The function that takes a Python data structure and returns a string of JSON data is `json.dumps()`. It is part of the `json` module in Python's standard library and is used to serialize Python objects into JSON format.

Here's an example of how to use `json.dumps()`:

```
`import json

# Python data structure (dictionary)

python_data = {

    "name": "John",

    "age": 25,

    "city": "New York"

}

# Serialize the Python data structure into a JSON string

json_data = json.dumps(python_data)



# Print the JSON string

print(json_data)

```


In the code snippet above, the `python_data` variable stores a Python dictionary. By calling `json.dumps(python_data)`, the Python data structure is serialized into a JSON string. The resulting JSON string is assigned to the `json_data` variable.

You can use the serialized JSON string for various purposes such as transmitting data, storing it in a file, or sending it over a network. In the example, we simply print the JSON string to demonstrate its creation.

Note that `json.dumps()` supports additional parameters to control the serialization process, such as indent level, sorting, and handling of non-serializable types.