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

Answer :

Excel spreadsheets (e.g., .xlsx, .xls files) offer several advantages compared to CSV (Comma-Separated Values) files:

* **Multiple Worksheets:** Excel files can contain multiple sheets within a single workbook, allowing for better organization of related data. CSV files are limited to a single sheet.
* **Data Formatting:** Excel supports rich formatting options, including fonts, colors, cell styles, conditional formatting, and more. CSV files store plain text without any formatting.
* **Formulas and Functions:** Excel allows the use of formulas, functions, and calculations directly within the spreadsheet. CSV files can only store raw data without any computational capabilities.
* **Data Types and Validation:** Excel can handle various data types (numbers, dates, text) with specific formatting and can enforce data validation rules. CSV treats all data as text without inherent type distinctions.
* **Charts and Graphs:** Excel provides tools to create charts, graphs, and other visual representations of data. CSV files do not support embedded visual elements.
* **Macros and Automation:** Excel supports macros and VBA scripting for automating tasks. CSV files lack this capability.
* **File Size and Efficiency:** For large datasets with complex features, Excel can be more efficient in terms of storage and data management compared to multiple CSV files.

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

Answer :

In Python's csv module:

* **csv.reader():** You pass a file-like object (typically obtained by opening a CSV file in text mode) to create a reader object. This object can then be iterated over to read rows from the CSV file.

python

Copy code

import csv

with open('data.csv', 'r', newline='') as csvfile:

reader = csv.reader(csvfile)

for row in reader:

print(row)

* **csv.writer():** Similarly, you pass a file-like object (opened in write mode) to create a writer object. This object can be used to write rows to a CSV file.

python

Copy code

import csv

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

writer = csv.writer(csvfile)

writer.writerow(['Column1', 'Column2', 'Column3'])

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

Answer :

When working with csv.reader() and csv.writer():

* **Reader Objects (csv.reader()):** The file should be opened in **read text mode**. It's recommended to specify newline='' to ensure that newlines are handled correctly across different operating systems.

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

* **Writer Objects (csv.writer()):** The file should be opened in **write text mode**. Similarly, newline='' should be specified to prevent issues with newline characters, especially on Windows.

python

Copy code

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

**Note:** Always use text mode ('r' or 'w') rather than binary mode ('rb' or 'wb') when working with the csv module in Python 3.

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

Answer :

The csv.writer object provides the writerow() method, which takes a single list (or any iterable) as an argument and writes it as a single row in the CSV file.

import csv

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

writer = csv.writer(csvfile)

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

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

Additionally, the writerows() method can take an iterable of lists and write multiple rows at once:

data = [

['Name', 'Age', 'City'],

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

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

]

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

writer = csv.writer(csvfile)

writer.writerows(data)

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

Answer :

When creating a csv.reader or csv.writer object, you can specify several keyword arguments to customize how the CSV data is parsed or written. Two important arguments are delimiter and lineterminator:

* **delimiter:** This argument specifies the character that separates fields (columns) in the CSV file. By default, it is a comma (','), but it can be changed to other characters like a tab ('\t'), semicolon (';'), pipe ('|'), etc.

**Summary:**

* **delimiter:** Defines the character that separates fields in the CSV.
* **lineterminator:** Defines the string that indicates the end of a line in the CSV.

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

Answer :

In Python, the json module provides functions to work with JSON data. The function that parses a JSON-formatted string and converts it into the corresponding Python data structure is json.loads().

* **json.loads(s):** Parses the JSON string s and returns the corresponding Python object (e.g., dict, list).

import json

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

data = json.loads(json\_string)

print(data)

# Output: {'name': 'Alice', 'age': 30, 'city': 'New York'}

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

Answer :

The function that serializes a Python data structure into a JSON-formatted string is json.dumps().

* **json.dumps(obj):** Converts the Python object obj into a JSON-formatted string.

import json

data = {

'name': 'Alice',

'age': 30,

'city': 'New York',

'hobbies': ['reading', 'traveling', 'swimming']

}

json\_string = json.dumps(data)

print(json\_string)

# Output: '{"name": "Alice", "age": 30, "city": "New York", "hobbies": ["reading", "traveling", "swimming"]}'