Intermediate Python Programming – Lesson 2

Facilitated by Kent State University
Topic: Working with Structured Data (CSV and JSON)
Duration: 1 Hour

Learning Objectives

By the end of this lesson, participants will be able to:

- Read and write structured data using the `csv` module.
- Parse and manipulate JSON data using the `json` module.
- Apply best practices for handling structured data efficiently.

Lesson 2: Working with Structured Data (CSV and JSON)

I. Introduction to Structured Data (10 minutes)

Structured data is information that adheres to a consistent model and is organized in a predictable way. It usually appears in formats like tables (rows and columns) or key-value pairs.

- **Structured data** includes data with fixed fields: spreadsheets, databases, and clearly defined records.
- **Unstructured data** includes formats like free-form text, images, audio, and video, where structure must be inferred.

Two common structured formats in Python are:

- **CSV** (Comma-Separated Values): flat, tabular data format ideal for spreadsheets or simple row-based data.
- **JSON** (JavaScript Object Notation): hierarchical data format often used in APIs and web services. Allows nested dictionaries and lists.

Exercise 1: Identify the data format

Question: Which format is best for storing hierarchical data?

A. CSV
B. JSON
C. Both are equally good

Answer: B. JSON – it supports nested structures.

Short Answer Question

What is the key difference between CSV and JSON in terms of structure and usability?

Expected Answer: CSV is tabular and simpler but lacks nesting, while JSON supports hierarchical structures, making it more flexible for web APIs and complex data storage.

II. Working with CSV Data (15 minutes)

The `csv` module provides tools for reading and writing CSV files. These files are commonly used to exchange tabular data between systems.

Reading a CSV file

```
import csv

with open('employees.csv', newline='') as file:
    reader = csv.reader(file)
    for row in reader:
        print(row)
```

This code opens a file named `employees.csv`, reads each row, and prints it as a list of strings.

Using `DictReader`

`csv.DictReader` reads each row as a dictionary using the header row as keys.

```
with open('employees.csv', newline='') as file:
    reader = csv.DictReader(file)
    for row in reader:
        print(row)
```

Writing CSV data

```
with open('output.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(['Name', 'Age', 'Department'])
    writer.writerow(['Alice', 30, 'Engineering'])
```

Exercise 2

Load the following CSV file and print its contents:

```
Name,Age,Department
Alice,30,Engineering
Bob,25,Marketing
Charlie,35,HR
```

Answer:

```
import csv

with open('employees.csv', newline='') as file:
    reader = csv.reader(file)
    for row in reader:
        print(row)
```

Expected Output:

```
['Name', 'Age', 'Department']
['Alice', '30', 'Engineering']
['Bob', '25', 'Marketing']
['Charlie', '35', 'HR']
```

Multiple-Choice Question

Which Python module is used for handling CSV files?

A. json
B. pandas
C. csv
D. os

Answer: C. csv

III. Parsing and Writing JSON Data (15 minutes)

JSON is the format of choice for exchanging hierarchical data, especially in web APIs. Python's `json` module allows reading and writing JSON data.

Python dictionary to JSON string

```
import json

employee = {"name": "Alice", "age": 30, "department": "Engineering"}
json_data = json.dumps(employee, indent=4)
print(json_data)
```

Expected Output:

```
{
    "name": "Alice",
    "age": 30,
    "department": "Engineering"
}
```

Reading JSON from a file

```
with open('data.json') as file:
    data = json.load(file)
```

Reading JSON from a string

```
json_string = '{"name": "Alice", "age": 30}'
data = json.loads(json_string)
```

Writing JSON to a file

```
with open('output.json', 'w') as file:
    json.dump(data, file, indent=4)
```

Exercise 3

Convert the following Python dictionary into a JSON string.

```
employee = {"name": "Alice", "age": 30, "department": "Engineering"}
```

Answer:

```
import json

json_data = json.dumps(employee, indent=4)
print(json_data)
```

Short Answer Question

What is the difference between `json.load()` and `json.loads()`?

Expected Answer: `json.load()` reads from a file, while `json.loads()` parses a JSON string.

IV. Handling Nested JSON Data (10 minutes)

Many real-world JSON documents include nested structures. To work with them, you'll need to access fields within dictionaries inside dictionaries.

Example JSON Structure

```
data = {
    "name": "Alice",
    "address": {
        "street": "123 Main St",
        "city": "New York"
    }
}
```

To extract the city:

```
city = data["address"]["city"]
print(city)
```

Output:

```
New York
```

Exercise 4

Extract the city value from the JSON structure above.

Answer:

```
data = {
    "name": "Alice",
    "address": {
        "street": "123 Main St",
        "city": "New York"
    }
}

city = data["address"]["city"]
print(city)
```

Multiple-Choice Question

How do you access the value of `city` in a nested JSON structure stored in `data`?

A. data["city"]
B. data["address"]["city"]
C. data["street"]["city"]
D. data.get("city")

Answer: B. data["address"]["city"]

V. Best Practices and Common Pitfalls (10 minutes)

Working with CSV

- Always open CSV files with `newline=''` to avoid extra blank lines on Windows.
- Handle missing values explicitly; consider using default values.
- Be aware of delimiters (e.g., tabs vs. commas).

Working with JSON

- Use `dict.get()` when unsure whether a key exists.
- Be cautious of `None` values when working with external APIs.
- Use `indent` and `sort_keys` options in `json.dumps()` for readable output.

When to Use Which Format

- Use **CSV** for flat, spreadsheet-like data.
- Use **JSON** for complex, nested, or hierarchical data such as API responses or configurations.

Final Exercise

Given the following two datasets, determine which format is best suited:

1. Customer records: ID, Name, Email, Phone, Address
2. API response: User details, multiple orders with product details

Expected Answer:

- CSV is better for customer records since it is tabular.
- JSON is better for API responses because it supports nested structures.

Final Multiple-Choice Question

Which of the following is a valid use case for JSON but not CSV?

A. Storing a flat list of users
B. Storing a hierarchical structure with nested relationships
C. Storing a table with a fixed number of columns
D. Importing data into a spreadsheet

Answer: B. JSON is preferred for nested structures.

VI. Recap and Q&A (5 minutes)

Today we:

- Distinguished between structured and unstructured data
- Read and wrote CSV files using the `csv` module
- Parsed and generated JSON using the `json` module
- Accessed nested JSON fields
- Discussed practical guidelines for using structured formats

Optional Practice for Next Time

Write a program that reads a JSON file of users, each with a list of addresses, and prints the city of the first address for each user.
