ITEC 1150 Week 14 Chapter 16

CSV & JSON DATA

Data Interchange Formats

- A couple of weeks ago we learned how to read and write from files and last week we looked at how to interact with web sites to pull down plain text files and scrape data from web pages.
- This has building up to one of the core design concepts in almost all modern software, *data interchange*.
- A computer without access to external data (i.e., internet access) is very limited; so too is a program without access to external data.
- This week, we'll look at two of the most common formats for exchanging data between systems, **CSV** and **JSON**.

CSV Files

"CSV" is short for 'comma-separated values'.

Data is mostly exchanged in CSV format systems in files, which are either sent via File Transfer Protocol (FTP), uploaded to a website, or downloaded from a website.

CSV Use Cases

- CSV is used when:
  - A lot of data needs to be shared, and it's OK to not get a status back right away
  - One or both of the parties sharing the data isn't capable of more complex data formats
  - All the data being shared is *tabular*, meaning it fits into a table-like structure. CSV doesn't support nested data or data with irregular numbers of fields
  - Humans might need to review or edit data—tools like Excel can open and process CSV files, making the data accessible to human users when necessary

CSV Data Example

- The data in a CSV file can be thought of as a table where columns are separated by commas and each line is a new row.
- In this example, the first line is a header which describes the contents of each field.

```
firstName,lastName,address1,address2,city,state,zipCode
Justin,Baker,594 Smith Drive,Apt. 999,South Elizabethfort,ME,81670
Jacqueline,Rice,5772 Rogers Course,,Caldwellfurt,TX,05714
Timothy,Franklin,4962 Lee Fork,Apt. 652,Port Christophertown,WY,98595
John,Lopez,5631 Amy Wall,,New Stephanie,GA,56342
Matthew,Clark,5434 Erika Track,,Port Nancybury,AS,41947
```

CSV Parsing Considerations

- CSV are simple text files, but as with everything, there are special situations that need to be handled, such as:
  - The field separator (a comma) is part of one of your values. This is typically solved by putting values in quotes where necessary.
  - The value contains both a comma and a quotation mark.
  - The value contains characters such as newlines or tabs
  - The field separator is something other than a comma, such as tabs or the pipe character "|"
- Because of these sorts of problems, it's better to use a library than simple string manipulation.
- Python provides the built-in `csv` module.

The CSV Module

- The code below works with an example file from the book. It's in D2L, named `example.csv`. Download and save it your current working directory.

```python
import csv
example_file = open('example.csv', encoding = 'utf-8') # good idea to include encoding
example_reader = csv.reader(example_file)
for row in example_reader:
    print('{:<18} {:^18} {:>5}'.format(*row, end = ''))
```

- What data type is each row?
- What if you want column headings?

Output:

```
4/5/2014 13:34          Apples            73
4/5/2014 3:41          Cherries           85
4/6/2014 12:46          Pears             14
4/8/2014 8:59          O,ranges,          52
4/10/2014 2:07          Apples           152
4/10/2014 18:10        Bananas            23
4/10/2014 2:40       Strawberries         98
```

Reading A File As A List Of Dictionaries

- The `csv.DictReader` will convert each line into a dictionary:

```python
import csv
example_file = open('example.csv', encoding='utf-8')
example_reader = csv.DictReader(example_file, fieldnames=['timestamp','fruit','quantity'])
data_list = list(example_reader)
print(data_list)
print(data_list[0]['timestamp']) # Get by line and key
print(data_list[6]['fruit']) # Get by line and key
```

- You can pass in the field names as shown; if you do not, `DictReader` will automatically use the first line as field names for the rest of the file. This can be handy if your CSV file already has a header line containing the field names.

Reading A File As A List Of Lists

- Use this code to study the structure of a .csv file and methods

```
import csv
example_file = open('example.csv', encoding='utf-8') # open file into a variable
example_reader = csv.reader(example_file) # create Reader object to iterate thru variable data
data_list = list(example_reader) # create a list for viewing the whole file at once
print(data_list) # What data type are the list items (the data rows)?
print(len(data_list)) # this shows length of what?
print(data_list[0][0]) # access values using index for [row] and [column]
print(data_list[6][1]) # which row and column are we printing here?
```

Writing CSV Files

- Use the `csv.writer()` to write data to a CSV file:

```python
import csv
output_file = open('output.csv', 'w', newline='') # open new file in 'w' mode
output_writer = csv.writer(output_file) # create a Writer object
output_writer.writerow(['fname', 'lname', 'phone', 'email']) # give writerow() method a list
# How can you make the next data row dynamic (user-defined)?

output_file.close()
```

- What lab assignments would be easier to do with CSV files than the unstructured text files we've been using?

JSON Files

JSON is short for 'JavaScript object notation'

It has become a *de facto* standard for exchanging data on the internet.

It's a very common standard for data exchange, even when no JavaScript is involved.

JSON Use Cases

- Unlike CSV, JSON works very well with nested data and allows much more complex data structures than the table-like format of CSV.
- Every value in a JSON document has a key, so programs can understand JSON without knowing the format beforehand (although this requires clever and complicated programming—it's still easiest to know what the data structure is beforehand).
- JSON is widely used for web services. Web services are like web pages designed to be read by programs—they have no HTML or styling and are pure data.
  - Web services can produce many different data formats (even CSV), but JSON is the most common.

JSON Data

- Look at the example on the right. Does it look at all like data you've added to a Python program?
- There's what looks like a dictionary with the keys `store_id`, `report_date`, and `products`.
- The value of `products` is a list (see the square brackets?)
- Each entry in the `products` list is another dictionary with the keys `name` and `price`.

```
{
    "store_id": 123,
    "report_date": "2024-04-15",
    "products": [
        {
            "name": "apples",
            "price": 3.99
        },
        {
            "name": "bananas",
            "price": 0.89
        },
        {
            "name": "oranges",
            "price": 4.99
        },
        {
            "name": "strawberries",
            "price": 6.99
        }
    ]
}
```

Working With JSON Data

If we put our JSON data into a variable named report, we can work with it just like we do with dictionaries and lists, because that's exactly what it is!

```python
report = {…} # data goes here; doesn't fit on the slide
print(f"Price report for store {report['store_id']} as of {report['report_date']}") # simple values
for product in report["products"]: # products is a list of dictionaries
    print(f"Product: {product['name']:<15} ${product['price']:>7.2f}")
```

```
Price report for store 123 as of date 2024-04-15
Product: apples          $   3.99
Product: bananas         $   0.89
Product: oranges         $   4.99
Product: strawberries    $   6.99
```

Tips For Understanding A JSON File

```
{
    "store_id": 123,
    "report_date": "2024-04-15",
    "products": [
        {
            "name": "apples",
            "price": 3.99
        },
        {
            "name": "bananas",
            "price": 0.89
        },
        {
            "name": "oranges",
            "price": 4.99
        },
        {
            "name": "strawberries",
            "price": 6.99
        }
    ]
}
```

- JSON files can be confusing at first but remember that they are just made up of dictionaries and lists.
- There is always one dictionary or list that makes up the base of the JSON string, and it is always in braces {}.
- Look at the data, figure out what the dictionaries are and what the lists are, and you should be able to work with it easily.

Reading JSON Data

- Import the built-in `json` and then call `json.loads()` with our data to convert it from a string to a data structure:

```python
import json
with open('example.json') as store_file:
    data = store_file.read() # this could be requests.get() for a web service!
    store_data = json.loads(data)
    print(store_data) # the parsed data, which can be helpful for debugging
    print("{:15}{:>6}".format('Store ID: ', store_data['store_id']))
    for product in store_data['products']:
        print("{:14}${:>6.2f}".format(product['name'], product['price']))
```

Writing JSON Data

- Converting data structures to JSON is simple; simply pass your data structure to the `json.dumps()` method to turn it into a plain string.
- That string can then be printed or written to a file.
- We could even write it to our own web service response! (You may do so in later courses.)

```python
import json
products = [{'name': 'apple', 'price': 1.69},
            {'name': 'banana', 'price': 0.49},
            {'name': 'orange', 'price': 1.19},
            {'name': 'strawberries', 'price': 2.49}]
store_report = {
             'store_id': 123,
             'report_date': '2024-04-15',
             'products': products
}
json_data = json.dumps(products)
print(json_data)
```