Lab 10: Files and Formats

In this lab, you'll get practice with files and formats, in preperation for P9.

File Vocabulary

For P9, you'll need to be familiar with the following file-related terms to know what we're asking you to do.

Before we get started with the assignment, let's talk about the distinction between these three terms, which will become important as we go along.

- **Directory:** a collection of files. "Folder" is a less-technical synonym you've doubtless heard frequently.
- **File Name:** a name you can use for a file if you know what directory you're in. For example, `movies.csv`, `test.py`, and `main.ipynb` are examples of file names. Note that different files can have the same name, as long as those files are in different directories.
- **Path:** a more-complete name that tells you the file name AND what directory it is in. For example, `p8/main.ipynb` and `p9/main.ipynb` are examples of path names on a Mac, referring to a file named `main.ipynb` in the `p8` directory and a second file with the same name in the `p9` directory, respectively. Windows uses back-slashes instead of forward slashes, so on a Windows laptop the paths would be `p8\main.ipynb` and `p9\main.ipynb`. There may be more levels in a path to represent more levels of directories. For example, `courses\cs301\p8\test.py` refers to the `test.py` file in the `p8` directory, which is in the `cs301` directory, which is in the `courses` directory.

In Python, there's not a special type for file names or paths; we just use regular strings instead.

Practice

Create a new directory named `lab10` and create a `main.ipynb` file there.

Let's start by doing some imports we'll need:

```
import os, json, csv
```

Files and Directories

Try running this cell to see the files and directories available alongside your notebook (remember that "." is a shorthand referring to the current directory):

```
# cell 1
os.listdir(".")
```

Let's try creating a new directory to experiment in by running this cell:

```
# cell 2
os.mkdir("fruit")
```

Now go back and manually rerun `cell 1` (when you called `listdir`).
Do you see the `fruit` directory this time?

Now click `Restart & Run All` from the `Kernel` menu. Do you notice that there's an exception in the cell where you created the `fruit` directory? This is because the directory already exists, and it is not possible to create another with the same name.

There are two options for doing the `mkdir` in a way that won't cause your notebook to fail in the case that the directory already exists. To get familiar with them, replace the code with option 1 below, then do a "Restart & Run All".

Netx, try option 2 as well.

Option 1: try/except

```
try:
    os.mkdir("fruit")
except FileExistsError:
    print("tried to create fruit, but it already existed")
```

Option 2: check beforehand

```
if not os.path.exists("fruit"):
    os.mkdir("fruit")
else:
    print("did not try to create fruit because it already existed")
```

Let's try creating a couple files in the directory. We'll need to specify a path to the file. Run this cell to get a path:

```
path = os.path.join("fruit", "apple.txt")
path
```

If you're on a Mac, you'll see `fruit/apple.txt`; on Windows, you'll see `fruit\apple.txt`. Be careful! Use this way to create paths. Never use the regular string join method we've learned, because that will not work on everbody's computer.

Now let's create a file with that path:

```
f = open(path, "w", encoding="utf-8")
f.write("apples are red\n")
f.close()
```

Did it work? Let's check:

```
os.listdir("fruit")
```

Also, try using `idle` to find and open the `apple.txt` file.

Now copy and adapt the above code to create a `banana.txt` and `orange.txt` file. You can decide what to write to these files.

Paste this code to a cell:

```
def fruit_message(name):
    f = open(os.path.join("fruit", name+".txt"), encoding="utf-8")
    msg = f.read()
    f.close()
    return msg
```

What does `fruit_message("apple")` return? (try it!)

Try the other fruits too. What if you try getting the message for a fruit that doesn't exist? Modify `fruit_message` so it returns "bad fruit" in that scenario. Use the `mkdir` example from earlier for inspiration.

JSON

JSON allows us to represent various Python structures (e.g., dicts) as strings. It is possible to save a string containing JSON data to a file (one might call such a file a JSON file, even though there is nothing special about the file except for its contents).

Saving Python data to a JSON file is a two step process (we'll soon see how to make this a one-step process):

1. convert the dict (or other structure) to a string
2. write that string to a file

Let's try it:

```
# Python structures
fruits = [
    {"name": "apple", "count": 50, "tasty": True},
    {"name": "watermelon", "count": 60, "tasty": False},
    {"name": "kiwi", "count": 55, "tasty": True},
]
print("Python structs:", fruits)

# JSON string
json_str = json.dumps(fruits)
print("JSON string:", json_str)

# save to file
f = open(os.path.join("fruit", "summary.json"), "w", encoding="utf-8")
f.write(json_str)
f.close()
```

Open `summary.json` in `idle`. How many differences do you see between JSON and the Python we wrote to create the structures?

Notice we had to call both `json_str = json.dumps(fruits)` and `f.write(json_str)`. The `json.dump` function combines these two. Try it! Replace `????` below to save some fruits of your choosing to a file of your choosing.

```
# Python structures
fruits = [
    ????
]
print("Python structs:", fruits)

# save to file
f = open(os.path.join("fruit", ????), "w", encoding="utf-8")
json.dump(fruits, f)
f.close()
```

Reading data back is also a two step process:

1. read from file to string
2. convert that string to JSON structures

Try it:

```
f = open(os.path.join("fruit", "summary.json"), encoding="utf-8")
json_str = f.read()
f.close()

data = json.loads(json_str)
print(data)
```

Just like `json.dump(data, f)` is a shortcut for `json.dumps` and `f.write`, `data = json.load(f)` is a shortcut for `f.read` and `json.loads`. Try simplifying the above code by using this shortcut.

CSV

Create a couple CSV files by running the following:

```
f = open(os.path.join("fruit", "good.csv"), "w", encoding="utf-8")
f.write("fruit,count\n")
f.write("apple,10\n")
f.write("banana,3\n")
f.write("orange,0\n")
f.close()

f = open(os.path.join("fruit", "rotten.csv"), "w", encoding="utf-8")
f.write("fruit,count\n")
f.write("apple,10\n")
f.write("banana,3\n")
f.write("orange\n")
f.close()
```

There are different ways to read CSV files. Perhaps one of the easiest is with a `csv.DictReader` object. A DictReader is created based on a file object. A DictReader is an iterator object; it produces a dictionary for each row of a CSV file, automatically using the header of the CSV to determine the keys for the dicts.

Try it:

```
f = open(os.path.join("fruit", "good.csv"), encoding="utf-8")
reader = csv.DictReader(f)
for row in reader:
    print(row)
f.close()
```

You should see something like this:

```
OrderedDict([('fruit', 'apple'), ('count', '10')])
OrderedDict([('fruit', 'banana'), ('count', '3')])
OrderedDict([('fruit', 'orange'), ('count', '0')])
```

What is an `OrderedDict`? It behaves just like the normal `dict` with which you are familar, but it keeps keys in a fixed order. The important thing for now is that you can use it like a regular dictionary:

For example, try looking specific cells and printing them:

```
f = open(os.path.join("fruit", "good.csv"), encoding="utf-8")
reader = csv.DictReader(f)
for row in reader:
    print(row["fruit"], row["count"])
f.close()
```

Try changing the above code to read "rotten.csv" instead of "good.csv". In "rotten.csv", there is a missing value for the count in the orange row. How does `DictReader` handle this? For the project, you'll need to write some code to skip CSV rows with missing values.