Programming fundamentals with Python

Using files with Python

Pepe García jgarciah@faculty.ie.edu

SCHOOL OF SCIENCE & TECHNOLOGY

Plan for today

- Learn about handling files with Python

SCHOOL OF SCIENCE & TECHNOLOGY

Plan for today

- Learn about handling files with Python
- Learn about CSV

SCHOOL OF SCIENCE & TECHNOLOGY

Plan for today

- Learn about handling files with Python
- Learn about CSV
- Learn about JSON

SCHOOL OF SCIENCE & TECHNOLOGY

the **open** function

We can use **open()** to open a file in Python, we only need to pass the path of the file we want to open. Let's say there's a file named hello.txt in my desktop that I want to open and read from Python, I can do it as follows:

SCHOOL OF SCIENCE & TECHNOLOGY

the **open** function

We can use **open()** to open a file in Python, we only need to pass the path of the file we want to open. Let's say there's a file named hello.txt in my desktop that I want to open and read from Python, I can do it as follows:

```
file = open("hello.txt")
```

SCHOOL OF SCIENCE & TECHNOLOGY

Reading the contents of a file

Now that we know how to open and close files, we can read the contents of a file. Let's do that line by line.

Reading the contents of a file

Now that we know how to open and close files, we can read the contents of a file. Let's do that line by line.

```
file = open("hello.txt")

for line in file:
    print(line)

file.close()
```

SCHOOL OF SCIENCE & TECHNOLOGY

Reading the contents of a file

Now that we know how to open and close files, we can read the contents of a file. Let's do that line by line.

```
file = open("hello.txt")

for line in file:
    print(line)

file.close()
```

As you can see, we're treating `file` as a list of lines.

Interlude, **with**

there is a useful Python keyword that one can use to make sure that the file will always be closed, **with**:

Interlude, **with**

there is a useful Python keyword that one can use to make sure that the file will always be closed, **with**:

```
with open("file_path") as file:

    for line in file:
        #do something with line
        print(line)
```

SCHOOL OF SCIENCE & TECHNOLOGY

Handling files. modes

When opening a file, we can choose in which **mode** we open it depending on how we're going to use it.

I/O Mode	Syntax	Behavior
Read	'r'	Opens the contents of a file for reading into the file interface, allowing for lines to be read-in successively.
Write	'w'	Creates a file with the specified name and allows for text to be written to the file; note that specifying a pre-existing filename will overwrite the existing file.
Append	'a'	Opens an existing file and allows for text to be written to it, starting at the conclusion of the original file contents.
Read and Write	'r+'	Opens a file such that its contents can be both read-in and written-to, thus offering great versatility.

Python's available file-access modes are summarized here.

doi:10.1371/journal.pcbi.1004867.t004

Figure 1: file modes

SCHOOL OF SCIENCE & TECHNOLOGY

Writing files

We can write into files in a way similar to the one used for reading them.

Writing files

We can write into files in a way similar to the one used for reading them.

```
with open('goodbye.txt', 'w') as file:
    file.write("goodbye y'all!")
```

SCHOOL OF SCIENCE & TECHNOLOGY

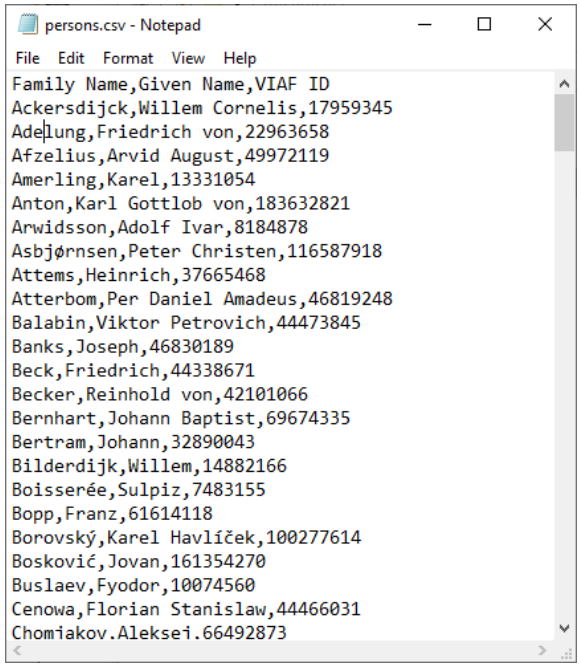
CSV

CSV is a data interchange format used for representing tabular data.

CSV - how does it look like?

CSV - how does it look like?

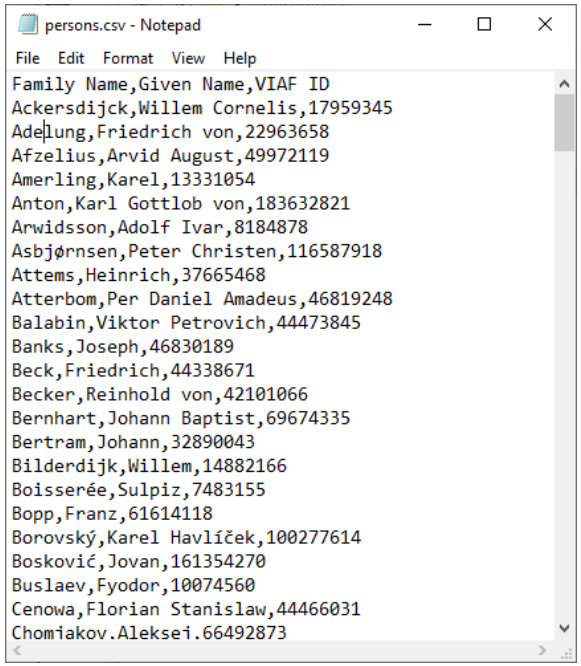
- **syntax** is, just the values separated by commas



SCHOOL OF SCIENCE & TECHNOLOGY

CSV - how does it look like?

- **syntax** is, just the values separated by commas
- We separate entries by adding a new line



CSV files - reading

The **csv** library is based on the idea of readers and writers. One can read all lines in a file like so:

```
import csv

with open("file.csv") as f:
    reader = csv.reader(f)
    for line in reader:
        print(line)  #line is a list
```

SCHOOL OF SCIENCE & TECHNOLOGY

CSV files - reading

The **csv** library is based on the idea of readers and writers. One can read all lines in a file like so:

```
import csv

with open("file.csv") as f:
    reader = csv.reader(f)
    for line in reader:
        print(line)  #line is a list
```

- first we open the file normally

CSV files - reading

The **csv** library is based on the idea of readers and writers. One can read all lines in a file like so:

```
import csv

with open("file.csv") as f:
    reader = csv.reader(f)
    for line in reader:
        print(line)  #line is a list
```

- first we open the file normally
- Then we create a reader using **csv.reader()**

SCHOOL OF SCIENCE & TECHNOLOGY

CSV files - reading

The **csv** library is based on the idea of readers and writers. One can read all lines in a file like so:

```
import csv

with open("file.csv") as f:
    reader = csv.reader(f)
    for line in reader:
        print(line)  #line is a list
```

- first we open the file normally
- Then we create a reader using **csv.reader()**
- Finally, we operate with the reader

SCHOOL OF SCIENCE & TECHNOLOGY

CSV files - writing

writing is not very different from reading:

```
lines = [
   ["asdf", "qwer"],
   ["hello", "world"]
]

with open("file.csv", "a") as f:
    writer = csv.writer(f)
    for line in lines:
        writer.writerow(line)
```

SCHOOL OF SCIENCE & TECHNOLOGY

CSV exercises

Download `all_stocks_5yr.csv` from data/ folder in Blackboard. Then:

1. create a function `find_cheapest_buy_day` that receives a ticker (like 'MSFT' or 'AAPL') and returns which is the day from the dataset to buy that stock the cheapest.
2. create a function that receives:
   - A ticker (example 'GOOG')
   - A buy date (example 2023-01-20)
   - A sell date (example 2023-12-25)
   - and an amount in dollars (example 1000)

And returns the selling amount.

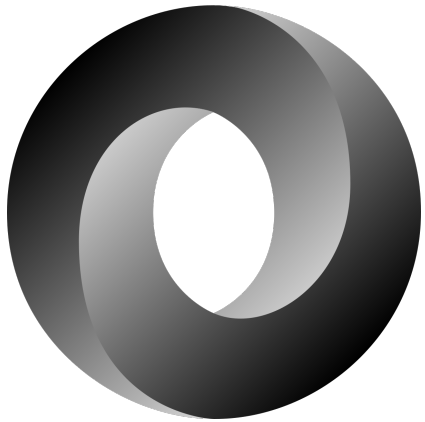
SCHOOL OF SCIENCE & TECHNOLOGY

Plan for today

- Files refresher
- CSV refresher
- Learn about JSON

JSON

JSON (http://json.org) is a data interchange format, like CSV. The name JSON stands for **Javascript Object Notation**, because the way of writing it is very similar to Javascript.
The main difference is that **JSON can represent arbitrary data, not only tabular data**.



SCHOOL OF SCIENCE & TECHNOLOGY

JSON - how does it look like?


```
{
    "orders": [
        {
            "orderno": "748745375",
            "date": "June 30, 2088 1:54:23 AM",
            "trackingno": "TN0039291",
            "custid": "11045",
            "customer": [
                {
                    "custid": "11045",
                    "fname": "Sue",
                    "lname": "Hatfield",
                    "address": "1409 Silver Street",
                    "city": "Ashland",
                    "state": "NE",
                    "zip": "68003"
                }
            ]
        }
    ]
}
```


SCHOOL OF SCIENCE & TECHNOLOGY

JSON - how does it look like?


```
{
    "orders": [
        {
            "orderno": "748745375",
            "date": "June 30, 2088 1:54:23 AM",
            "trackingno": "TN0039291",
            "custid": "11045",
            "customer": [
                {
                    "custid": "11045",
                    "fname": "Sue",
                    "lname": "Hatfield",
                    "address": "1409 Silver Street",
                    "city": "Ashland",
                    "state": "NE",
                    "zip": "68003"
                }
            ]
        }
    ]
}
```


- **syntax** similar to Python data structures

SCHOOL OF SCIENCE & TECHNOLOGY

JSON - how does it look like?

```
{
    "orders": [
        {
            "orderno": "748745375",
            "date": "June 30, 2088 1:54:23 AM",
            "trackingno": "TN0039291",
            "custid": "11045",
            "customer": [
                {
                    "custid": "11045",
                    "fname": "Sue",
                    "lname": "Hatfield",
                    "address": "1409 Silver Street",
                    "city": "Ashland",
                    "state": "NE",
                    "zip": "68003"
                }
            ]
        }
    ]
}
```

- **syntax** similar to Python data structures
- supports **primitive** datatypes (**int, str, bool, float**).

SCHOOL OF SCIENCE & TECHNOLOGY

JSON - how does it look like?


```
{
    "orders": [
        {
            "orderno": "748745375",
            "date": "June 30, 2088 1:54:23 AM",
            "trackingno": "TN0039291",
            "custid": "11045",
            "customer": [
                {
                    "custid": "11045",
                    "fname": "Sue",
                    "lname": "Hatfield",
                    "address": "1409 Silver Street",
                    "city": "Ashland",
                    "state": "NE",
                    "zip": "68003"
                }
            ]
        }
    ]
}
```


- **syntax** similar to Python data structures
- supports **primitive** datatypes (**int, str, bool, float**).
- supports collections of elements with **lists**

SCHOOL OF SCIENCE & TECHNOLOGY

JSON - how does it look like?

```
{
    "orders": [
        {
            "orderno": "748745375",
            "date": "June 30, 2088 1:54:23 AM",
            "trackingno": "TN0039291",
            "custid": "11045",
            "customer": [
                {
                    "custid": "11045",
                    "fname": "Sue",
                    "lname": "Hatfield",
                    "address": "1409 Silver Street",
                    "city": "Ashland",
                    "state": "NE",
                    "zip": "68003"
                }
            ]
        }
    ]
}
```

- **syntax** similar to Python data structures
- supports **primitive** datatypes (**int, str, bool, float**).
- supports collections of elements with **lists**
- supports mapping of elements with **dictionaries**

JSON

JSON can contain

```
[1, 2, 3]
1
true
"potatoes"
4.77
null
{"first":"Pepe","last":"Garcia"}
```

SCHOOL OF SCIENCE & TECHNOLOGY

JSON

JSON can contain

```
[1, 2, 3]
1
true
"potatoes"
4.77
null
{"first":"Pepe","last":"Garcia"}
```

- lists

SCHOOL OF SCIENCE & TECHNOLOGY

JSON

JSON can contain

```
[1, 2, 3]
1
true
"potatoes"
4.77
null
{"first":"Pepe","last":"Garcia"}
```

- lists
- integers

SCHOOL OF SCIENCE & TECHNOLOGY

JSON

JSON can contain

```
[1, 2, 3]
1
true
"potatoes"
4.77
null
{"first":"Pepe","last":"Garcia"}
```

- lists
- integers
- booleans

SCHOOL OF SCIENCE & TECHNOLOGY

JSON

JSON can contain

```
[1, 2, 3]
1
true
"potatoes"
4.77
null
{"first":"Pepe","last":"Garcia"}
```

- lists
- integers
- booleans
- strings

SCHOOL OF SCIENCE & TECHNOLOGY

JSON

JSON can contain

```
[1, 2, 3]
1
true
"potatoes"
4.77
null
{"first":"Pepe","last":"Garcia"}
```

- lists
- integers
- booleans
- strings
- floats

SCHOOL OF SCIENCE & TECHNOLOGY

JSON

JSON can contain

```
[1, 2, 3]
1
true
"potatoes"
4.77
null
{"first":"Pepe","last":"Garcia"}
```

- lists
- integers
- booleans
- strings
- floats
- null (an empty value)

SCHOOL OF SCIENCE & TECHNOLOGY

JSON

JSON can contain

```
[1, 2, 3]
1
true
"potatoes"
4.77
null
{"first":"Pepe","last":"Garcia"}
```

- lists
- integers
- booleans
- strings
- floats
- null (an empty value)
- dictionaries

SCHOOL OF SCIENCE & TECHNOLOGY

JSON

JSON is very similar to how we declare our data in Python but the cool thing about it is that it can be used **from any language**. In Python we will be able to use JSON using the `json` module

```
import json
```

SCHOOL OF SCIENCE & TECHNOLOGY

JSON - reading JSON data in Python

As with other formats we've seen so far, in order to operate with json files we will first **open()** the file.

SCHOOL OF SCIENCE & TECHNOLOGY

JSON - reading JSON data in Python

As with other formats we've seen so far, in order to operate with json files we will first **open()** the file.

```
import json

with open("data.json") as file:
    json_data = json.load(file)

    for key in json_data:
        print(key)
```

`json.load` is a function from the json module that takes a **file object** as parameter and returns the contents of that file **parsed as JSON**.

JSON - writing JSON files

The process of writing JSON files is similar to what we know already.

```
import json

data = {
  "name": "Pepe",
  "last_name": "Garcia"
}

with open("data.json", "w") as file:
    json.dump(data, file)
```

As you can see, we're calling **`json.dump`** from the **`json`** library, and passing first the data we want to write to the file and then the file object as parameters.

Exercises

You will find the data files for these exercises in the repository

- Let's get personal data from the person represented in `luke.json`. Print the `name`, `height`, `eye_color`, and `mass`.
- Let's create a **format conversor**. Our function `convert_format` will read all the data from `data/data.csv` and write it to a new **JSON** file named `converted.json`