

# Programming fundamentals with Python

## Session 6

Pepe García [jgarciah@faculty.ie.edu](mailto:jgarciah@faculty.ie.edu)

2020-10-16

# Plan for today

- Files refresher

# Plan for today

- Files refresher
- Learn about JSON

# 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:

# 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("/Users/pepe/Desktop/hello.txt")
```

# 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("/Users/pepe/Desktop/hello.txt")  
  
for line in file:  
    print(line)  
  
file.close()
```

# 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("/Users/pepe/Desktop/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)
```

# 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

# 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.

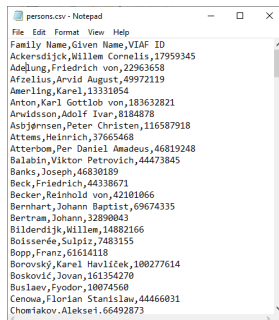
```
with open('/Users/pepe/Desktop/goodbye.txt', 'w') as file:  
    file.write("goodbye y'all!")
```

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



```
persons.csv - Notepad
File Edit Format View Help
Family Name,Given Name,VIAF ID
Ackersdijck,Willem Cornelis,17959345
Adelung,Friedrich von,22963658
Afzelius,Arvid August,49972119
Amerling,Karel,13331054
Anton,Karl Gottlob von,183632821
Arwidsson,Adolf Ivar,8184878
Asbjørnsen,Peter Christen,116587918
Attems,Heinrich,37665468
Atterbom,Per Daniel Amadeus,46819248
Balabin,Viktor Petrovich,44473845
Banks,Joseph,46830189
Beck,Friedrich,44338671
Becker,Reinhold von,42101066
Bernhart,Johann Baptist,69674335
Bertram,Johann,32890043
Bilderdijk,Willem,14882166
Boisserée,Sulpiz,7483155
Bopp,Franz,61614118
Borovský,Karel Havlíček,100277614
Bosković,Jovan,161354270
Buslaev,Fyodor,10074560
Cenowa,Florian Stanislaw,44466031
Chomiakov,Aleksei,66492873
```



# CSV - how does it look like?

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

```
persons.csv - Notepad
File Edit Format View Help
Family Name, Given Name, VIAF ID
Ackersdijck, Willem Cornelis, 17959345
Adellung, Friedrich von, 22963658
Afzelius, Arvid August, 49972119
Amerling, Karel, 13331054
Anton, Karl Gottlob von, 183632821
Arwidsson, Adolf Ivar, 8184878
Asbjørnsen, Peter Christen, 116587918
Attems, Heinrich, 37665468
Atterbom, Per Daniel Amadeus, 46819248
Balabin, Viktor Petrovich, 44473845
Banks, Joseph, 46830189
Beck, Friedrich, 44338671
Becker, Reinhold von, 42101066
Bernhart, Johann Baptist, 69674335
Bertram, Johann, 32890043
Bilderdijk, Willem, 14882166
Boisseree, Sulpiz, 7483155
Bopp, Franz, 61614118
Borovský, Karel Havlíček, 100277614
Bosković, Jovan, 161354270
Buslaev, Fyodor, 10074560
Cenowa, Florian Stanislaw, 44466031
Chomiakov, Aleksei, 66492873
```

# 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
```

# 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()**

# 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

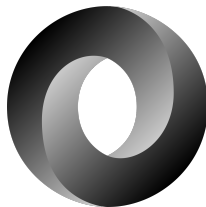
# CSV files - writing

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)
```

JSON (<http://json.org>) is a data interchange format, like CSV

The main difference is that with JSON we can represent arbitrary data, not only tabular data.



# 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"
        }
      ]
    }
  ]
}
```



# 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

# 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**).

# 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**

# 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

Some valid JSON values are:

```
[1, 2, 3]
```

```
1
```

```
true
```

```
"potatoes"
```

```
{"name": "Pepe", "surname": "Garcia"}
```

# JSON

Some valid JSON values are:

```
[1, 2, 3]
```

```
1
```

```
true
```

```
"potatoes"
```

```
{"name": "Pepe", "surname": "Garcia"}
```

- lists

# JSON

Some valid JSON values are:

```
[1, 2, 3]
1
true
"potatoes"
{"name": "Pepe", "surname": "Garcia"}
```

- lists
- integers

# JSON

Some valid JSON values are:

```
[1, 2, 3]
1
true
"potatoes"
{"name": "Pepe", "surname": "Garcia"}
```

- lists
- integers
- booleans



# JSON

Some valid JSON values are:

```
[1, 2, 3]
1
true
"potatoes"
{"name": "Pepe", "surname": "Garcia"}
```

- lists
- integers
- booleans
- strings

# JSON

Some valid JSON values are:

```
[1, 2, 3]
1
true
"potatoes"
{"name": "Pepe", "surname": "Garcia"}
```

- lists
- integers
- booleans
- strings
- dictionaries

# 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**.

# Homework

You will find the data files for these exercises in this repository:  
<https://github.com/pfp-2020/session-6>

