Basic Programming

Lesson 9. Working with files. Reading and Writing

Alexandra Ciobotaru

10 dec 2021

Syllabus

- **Lesson 1. Computers, Programming and Cognitive Science. From pseudocode to programming languages.**
- **Lesson 2. Variables in Python. Basic calculus. Using Math library, type() and help() functions.**
- **Lesson 3. Collections: Lists, Tuples.**
- **Lesson 4. Strings. Working with strings.**
- **Lesson 5. Branching and decisions: Logical operators, If-Statements, Nested conditions. Loops: For and While**
- **Lesson 6**. **Lesson 5 continued** Quiz 1 (25%).
- **Lesson 7. Creating functions. Recursive functions. Matrices**.
- **Lesson 8. Collections: Dictionaries. JSON construction.**
- Lesson 9. Working with files. Reading and Writing.
- Lesson 10. Analyzing dataframes. Pandas and Matplotlib Python libraries.
- Lesson 12. Object-Oriented Programming: Encapsulation, Inheritance and Polymorphism. Quiz 2 (25%).
- Lesson 13. Object-Oriented Programming (cont.). Error handling. Best practices when programming.
- Lab (20%) + final exam (30%).

What you'll learn today:

- 1. Reading and writing files
  - 1.1. Reading JSON files
  - 1.2. Looping through JSON elements
  - 1.3. Writing JSON files
  - 1.4. Reading text files
  - 1.5. Writing to a text file
  - 1.6. Looping through a text file
- 2. Working with directories
  - 2.1. os modules
  - 2.2. Recursevly traversing directories
  - 2.2. Paths in Python

1. Reading and writing files
1.1. Reading JSON files

- Remember the JSON file we made last lesson?
- **Deserialization of JSON**
- The Deserialization of JSON means the conversion of JSON objects into their respective Python objects. The load()/loads() method is used for it.
- Method ***json.load*** *reads the string from a file*, parses the JSON data, populates a Python dict with the data and returns it back to you:

```
with open('my_json.json') as json_file:
    data = json.load(json_file)
print(data)
{'title': 'cognitive science rocks', 'number of subjects':
5, 'rocks': True, 'subjects': ['statistics',
'programming', 'AI'], 'my grades': {'statistics': 10,
'programming': 10, 'AI': 10}}
```

- Method ***json.loads*** *creates a JSON object from a string*:

```
my_string = ''' {
"title" : "cognitive science rocks",
"number of subjects" : 5,
"rocks" : true,
"subjects": ["statistics","programming","AI"],
"my_grades": {"statistics":10, "programming":10,
"AI":10}
}'''
new_json = json.loads(my_string)
print(new_json)
{'title': 'cognitive science rocks', 'number of
subjects': 5, 'rocks': True, 'subjects':
['statistics', 'programming', 'AI'], 'my_grades':
{'statistics': 10, 'programming': 10, 'AI': 10}}
```

- → same result, a dictionary

JSON syntax

- **Name/Value pairs:** Represents Data, name is followed by ':'(colon) and the Name/Value pairs are separated by ', ' (comma).
- **Curly braces:** we say that curly brackets hold objects.
- **Square brackets:** we say that square brackets hold arrays with values separated by ', ' (comma).

```
{
  "array": [
    1,
    2,
    3,
    4
  ],
  "boolean": true,
  "color": "#82b92c",
  "null": null,
  "number": 123,
  "object": {
    "a": "b",
    "c": "d",
    "e": "f"
  },
  "string": "Hello World"
}
```

Exercise – acces the nested key ‘salary’ from the following JSON

```
import json
sampleJson = """{
   "company":{
      "employee":{
         "name":"emma",
         "payble":{
            "salary":7000,
            "bonus":800
         }
      }
   }
}"""
data = json.loads(sampleJson)
print(data['company']['employee']['payble']['salary'])
```

1.2. Looping through JSON elements

```
for subject in new_json['subjects']:
    print(subject)
statistics
programming
AI
```

- If I modify my_string a bit so that I have more values for the keys in my dictionary:

```
my_string = ''' {
"title" : "cognitive science rocks",
"number of subjects" : 5,
"rocks" : true,
"subjects": ["statistics","programming","AI"],
"my_grades": {"statistics":[10,9,8], "programming":[10,9],
"AI":10}
}'''
new_json = json.loads(my_string)
print(new_json)
for matter, grade in new_json['my_grades'].items():
    if matter == "statistics":
        print(grade)
```

1.3. Writing JSON files

- **Serialization of JSON** – saving a Python object
  - Saving it to a JSON file - using ***json.dump*** method

```
data = {
    "president": {
        "name": "Zaphod Beeblebrox",
        "species": "Betelgeusian"
    }
}
with open("data_file.json", "w") as write_file:
    json.dump(data, write_file)
```

  - Saving it to a string – using ***json.dumps*** method

```
json_string = json.dumps(data)
```

1.4. Reading text files

- Reading a file requires 3 steps:
  1. Opening the file
  2. Reading the file
  3. Closing the file

```
my_file = open("D:/folder/example.txt", "r")
text = my_file.read()
my_file.close()
print(text)
```

File path
File name
Extension
Mode
Folder path

- The same can be done using 'with open' statement:

```
with open('D:/example.txt', "r") as my_file:
    text = my_file.read()
print(type(text)) -> <class '_io.TextIOWrapper'>
```

The file is closed automatically.

Opening modes

Character	Meaning
'r'	Opens a file for reading, error if the file does not exist (**default**)
'w'	Open for writing, truncating (overwriting) the file first
'a'	Opens a file for appending, creates the file if it does not exist
'x'	Creates the specified file, returns an error if the file exists

In addition, you can specify if the file should be handled as binary or text mode:

"t" - Text - Text mode (**default**)
"b" - Binary - Binary mode (e.g. images)

Reading line by line

```
f = open("demofile.txt", "r")
print(f.readline()) - prints first line
print(f.readline()) - prints second line
```

- Loop through the file line by line:

```
f = open("demofile.txt", "r")
for x in f:
  print(x)
```

- Don’t forget to close the file.

1.5. Writing to a text file

- **Write method** writes string content referenced by file object: *file_name.write(content)*

If the file does not exist, it is created:

```
with open('new_file.txt', 'w') as f:
    f.write('Hello world!')
Appending lines at the end of a file:
with open('new_file.txt', 'a') as f:
    f.write('Hello hello world!')
```

- See that it appended with no new line? To add a new line write /n:

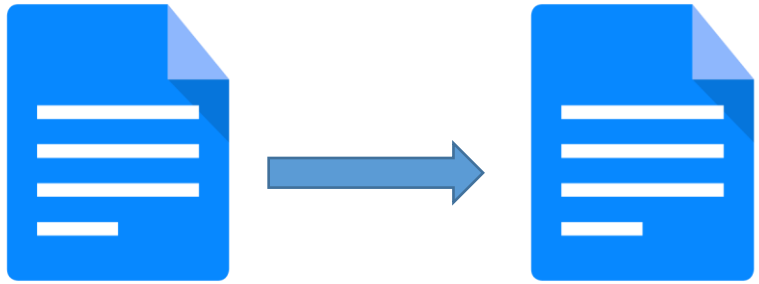
```
with open('new_file.txt', 'a') as f:
    f.write('Hello /n hello /n world!')
```

- **Writelines method** writes all the strings present in the list "list_of_lines" referenced by file object: *file_name.writelines(list_of_lines)*

```
f = open("demofile3.txt", "a")
f.writelines(["\nSee you soon!", "\nOver and out."])
f.close()
```

1.6. Looping through a text file

- Suppose you have a text file containing some numbers, and you want to write in another text file only those numbers that start with a specific value:



```
outf1 = open('output1.txt', 'w')
with open('data.txt') as inf:
    for line in inf:
        #print(line.strip())
        if int(line.strip().startswith('2')):
            outf1.write('this line startswith 2 : {}
\n'.format(line.strip()) )
outf1.close()
```

- Closing a file is important because a closed file reduces the risk of being unwarrantedly modified or read.

List comprehension

- You can write any accumulating pattern using list comprehension – the most elegant way of creating lists in python.

Do this	For this collection	In this situation
`[x**2`	`for x in range(0, 50)`	`if x % 3 == 0]`

- Our example:

```
outf2 = open('output2.txt', 'w')
with open('data.txt') as inf:
    acc = [ 'this line startswith 2
(with list
    comprehension): {}
\n'.format(line.strip()) for
    line in inf if
int(line.strip().startswith('2'))]
    outf2.writelines(acc)
outf2.close()
```

```
In [5]: n = 0
        for x in range(10):
            S[n] = x**2
            n +=1
        S

Out[5]: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
In [1]: S =[x**2 for x in range(10)]
        S

Out[1]: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
numbers = []
for i in range(100):
    numbers.append(i)
```

```
numbers = [i for i in range(100)]
```

2. Working with directories
2.1. os module

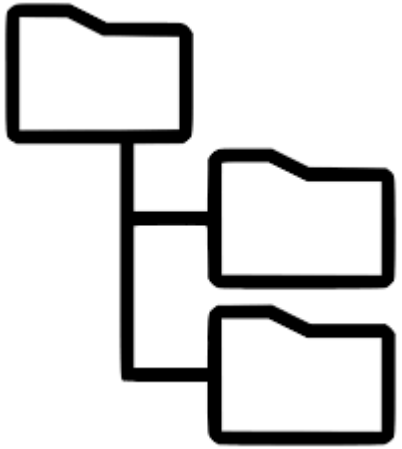
- Getting the **current working directory**
  import os
  current_dir = os.getcwd()
- Change the current working directory
  path = “path_you_wish”
  os.chdir(path)
- Printing all directories in the path:
  print(os.listdir()) - default is working path
  print(os.listdir(“some_other_path”))
- Printing the contents of a directory called my_folder inside the path:
  print(os.listdir(“my_folder”))
- Create a new directory in the current path:
  os.mkdir(“new_dir”)
- Rename a directory
  os.rename(“new_dir”, “new_dir_updated”)
- Remove a file:
  os.remove(“path_to_file”)
- To remove a directory, that directory must be empty:
  os.rmdir(“path_todirectory”)

2.2. Recursively traversing directories

- Create directory tree image here

```
for root, dirs, files in os.walk(path):
    print(len(root), root, dirs, files)
    for file in files:
        print(file)


for i in os.walk(path):
    print(i)
```



2.3. Working with paths

- Common way of writing a path:
- path = ‘D:/pycharmProjects/snscrape_tweets/env/wassa_ro’
- But when you copy a path from folder and paste it you will get an error:
- path = ‘D:\pycharmProjects\snscrape_tweets\env\wassa_ro’
- Solution:
  - path = r‘D:\pycharmProjects\snscrape_tweets\env\wassa_ro’ (raw string)
  - path = ‘D:\\pycharmProjects\\snscrape_tweets\\env\\wassa_ro’ (escape backslashes)
- By default, reading and writing files will be done in the current working directory.
- with open('new_file.txt') as f:
- ... print(f) → Tries to find the file in the mother directory
- with open('../new_file.txt') as f:
- ... print(f) → Tries to find the file in the mother’s mother directory
- with open('../../new_file.txt') as f:
- ... print(f)

Thank you