

Engineering programming

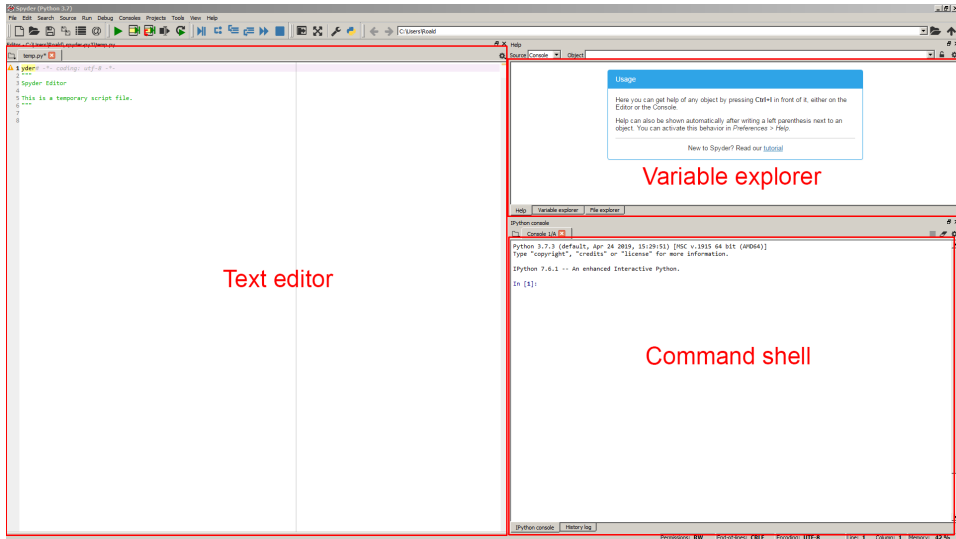
Files

Thibault Thétier - thibault.thetier@vub.be

Tom Godden - tgodden@vub.be

1ste semester 2021

SPYDER IDE



FILES

Problem: we need to work with data

- ▶ We need to be able to save it
- ▶ We need to be able to access it
- ▶ We need to be able to share it

Solution: Files!

- ▶ Very often the datas you will have to use are stored in files

PATH

- ▶ **Absolute path**: describes the location of a file or folder regardless of the current working directory
 - ▶ contains the complete location of a file or directory
 - ▶ example: `D:\documents \mydocument .doc`
- ▶ **Relative path**: describes the location of a file or folder in relative to the current working directory
 - ▶ If you're in the same directory: `mydocument .doc`

BASICS

To open a file for reading or writing, use the built-in open function with either a relative or absolute file path:

```
1 path = 'example.txt'
2 f = open(path)
```

By default, the file is opened in read-only mode 'r'.

`f` is a file handle and can be treated like a list.

```
1 for line in f:
2     #do something
```

When you use open to create file objects, never forget to close it.

```
1 f.close()
```

BASICS

To make it easier: the `with` statement

```
1 with open(path) as f:  
2     for line in f:  
3         #do something
```

This will automatically close the file `f` when exiting the `with` block.
You have multiple file modes:

- ▶ `r`: Read-only mode
- ▶ `w`: Write-only mode; creates a new file (erase the existing)
- ▶ `a`: Append to existing file (create file if not existing)
- ▶ `r+`: Read and write

```
1 f = open(path, 'a')
```

READING & WRITING

You can read and write a file using python's built-in methods

```
1 with open('demo.txt', 'w') as f:  
2     f.writelines("one text line %d\n" %i for i in range(4))
```

Then you can access what's inside the file:

```
1 with open('demo.txt', 'r') as f:  
2     lines = f.readlines()  
3 lines  
4 ['line 0\n', 'line 1\n', 'line 2\n', 'line 3\n']
```

You have multiple methods

- ▶ read([size]) – Return data from file as a string
- ▶ readlines([size]) – Return list of lines in the file
- ▶ write(str) – Write string to file
- ▶ writelines(strings) – Write sequence of strings to the file

One of the most popular formats for exchanging data is the CSV format

- ▶ Comma-Separated Values
- ▶ Plain text file using specific structuring to arrange tabular data
- ▶ Multiple libraries existing: csv, pandas, etc.

An example of a CSV file:

```
1 column 1 name,column 2 name,column 3 name
2 first row data 1,first row data 2,first row data 3
3 second row data 1,second row data 2,second row data 3
```


CSV - DICTREADER

The csv module implements classes to read and write tabular data in CSV format. Dictreader maps the information read from the file into a dictionary.

```
1 import csv
2 with open('exp.csv') as f:
3     reader = csv.DictReader(f)
4     for row in reader:
5         print(row)
```

```
6         #also possible: list(reader)
1  {'min': '1', 'avg': '5.5', 'max': '10'}
2  {'min': '2', 'avg': '3.5', 'max': '5'}
3  {'min': '3', 'avg': '5.5', 'max': '8'}
4  {'min': '4', 'avg': '5.5', 'max': '7'}
```

CSV - DICTWRITER

DictWriter maps Python dictionaries into CSV rows.

The fieldnames parameter is a sequence of keys that identify the order in which values are written to the CSV file.

```
1  lst_rows = [                                1  fnames = ['min', 'max', 'avg']
2  {'min': '1', 'avg': '5.5', 'max': '10'},      with open('exp2.csv', 'w') as f:
3  {'min': '2', 'avg': '3.5', 'max': '5'},      writer = csv.DictWriter(f,
4  {'min': '3', 'avg': '5.5', 'max': '8'},      ↪ fieldnames=fnames)
5  {'min': '4', 'avg': '5.5', 'max': '7'}      writer.writerow()
6  ]                                             5  writer.writerows(lst_rows)
```

WITH PANDAS - READING

Pandas has its own tools for reading tabular data as a DataFrame object

```
1 df = pd.read_csv('files/ex1.csv')
2     a    b    c    d  message
3 0  1    2    3    4   hello
4 1  5    6    7    8   world
5 2  9   10   11   12    foo
```



```
1 pd.read_table('files/ex1.csv', sep=',')
2     a    b    c    d  message
3 0  1    2    3    4   hello
4 1  5    6    7    8   world
5 2  9   10   11   12    foo
```

WITH PANDAS - READING

A file will not always have a header row, you might also want to specify the header yourself

```
1 pd.read_csv('files/ex2.csv', header=None)
```

```
2      0    1    2    3    4
3  0    1    2    3    4  hello
4  1    5    6    7    8  world
5  2    9   10   11   12   foo
```

```
1 pd.read_csv('files/ex2.csv', names=['a', 'b', 'c', 'd', 'message'])
```

```
2      a    b    c    d  message
3  0    1    2    3    4  hello
4  1    5    6    7    8  world
5  2    9   10   11   12   foo
```

WITH PANDAS - WRITING

- ▶ Data can also be exported to a delimited format

```
1 data = pd.read_csv('files/ex3.csv')
2     something  a    b    c    d message
3 0         one  1    2  3.0  4      NaN
4 1         two  5    6  NaN  8    world
5 2        three  9   10 11.0 12      foo
6 data.to_csv('files/out.csv')
```

- ▶ Missing values appear as empty strings. You can decide of the sentinel value:

```
1 data.to_csv(sys.stdout, na_rep='NULL')
2 ,something,a,b,c,d,message
3 0,one,1,2,3.0,4,NULL
4 1,two,5,6,NULL,8,world
5 2,three,9,10,11.0,12,foo
```

WITH PANDAS - WRITING

- ▶ You can disable row and column labels

```
1 data.to_csv(sys.stdout, index=False, header=False)
2 one,1,2,3.0,4,
3 two,5,6,,8,world
4 three,9,10,11.0,12,foo
```

- ▶ You can also write only a subset of the columns, in the desired order

```
1 data.to_csv(sys.stdout, index=False, columns=['a', 'b', 'c'])
2 a,b,c
3 1,2,3.0
4 5,6,
5 9,10,11.0
```

JSON

- ▶ Internet standard for exchanging data
- ▶ Same structure as a Python Dict
- ▶ Values can be numbers, text, null/None, lists or JSON objects

```
1 {'name': 'Wes',  
2   'pet': null,  
3   'places_lived': ['United States', 'Spain', 'Germany'],  
4   'siblings': [{  
5     ↪ 'age': 30, 'name': 'Scott', 'pets': ['Zeus',  
6     ↪ 'Zuko']},  
7     {  
8     ↪ 'age': 38, 'name': 'Katie', 'pets': ['Sixes',  
9     ↪ 'Stache', 'Cisco']}]  
10  }
```

JSON

```
1 import json
2 with open('example.json') as jsonfile:
3     data = json.load(jsonfile)
```

- ▶ JSON Objects become Python dicts
- ▶ JSON Arrays become Python lists

JSON WITH PANDAS

```
1 import pandas as pd
2 df = pd.read_json('path/to/json/database.json')
```

JSON is less structured than csv

- ▶ maps poorly to tables and dataframes
- ▶ Pandas cannot deal well with nested objects
- ▶ Nested objects become Python dicts

PD.JSON_NORMALIZE

```
>>> data = [{'id': 1, 'name': {'first': 'Coleen', 'last': 'Volk'}},  
...         {'name': {'given': 'Mose', 'family': 'Regner'}},  
...         {'id': 2, 'name': 'Faye Raker'}]  
>>> pandas.json_normalize(data)
```

	id	name	name.family	name.first	name.given	name.last
0	1.0	NaN	NaN	Coleen	NaN	Volk
1	NaN	NaN	Regner	NaN	Mose	NaN
2	2.0	Faye Raker	NaN	NaN	NaN	NaN

- ▶ Adds values of nested dicts to dataframe
- ▶ Does not work with lists!