



15-110 PRINCIPLES OF COMPUTING – F19

LECTURE 26: FILES I/O 4

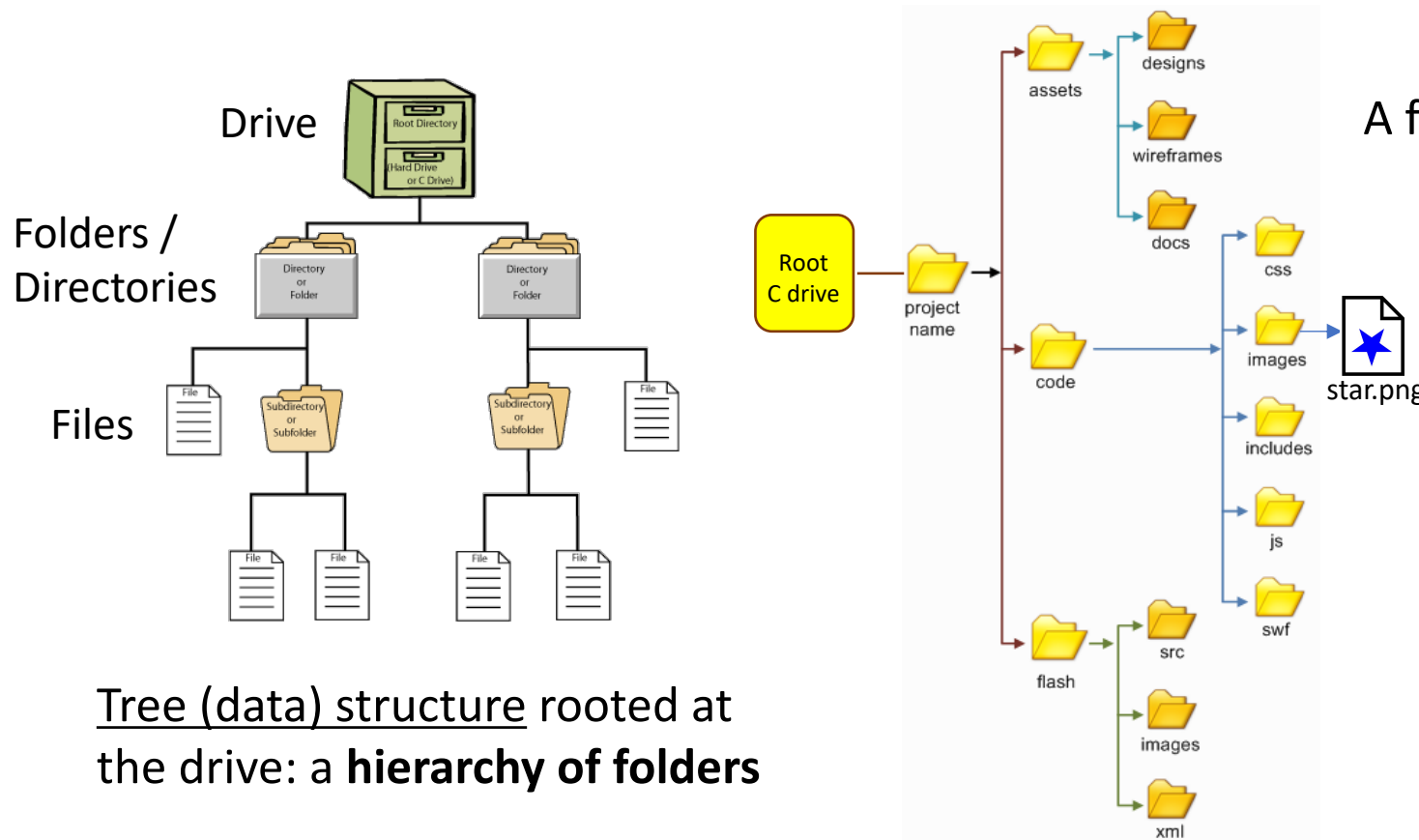
TEACHER:
GIANNI A. DI CARO

Common questions / challenges about files

- Trying to open (with a read flag) a non-existing file results into an error: how do we **check if a file exists** or not in the file system prior any attempt to open it? (to avoid that the program crashes?)
- A file might be **located anywhere in the file system**: how do we refer to such a file?
- How do we **search for a named file in the file system**?
- Sometimes it is appropriate to create a new folder where to put newly created files: how do we **create a new folder** from a python program?
- Opening an existing file with the write flag erases file content: how can we **check file status first**?
- A file might be too large to read, or might be not accessible for read or write to my program, or might have been modified very recently: how do we get **information about a file**?

Operations on the file system

- Module **os** (**O**perating **S**ystem) offers a complete set of functionalities to inspect and manipulate the elements of the **file system of the computer**



A folder / file is uniquely identified by:

- ✓ a **name**
- ✓ the **path** relative to the root of the drive

Windows:

C:\project name\code\images\star.png

Unix:

/project name/code/images/star.png

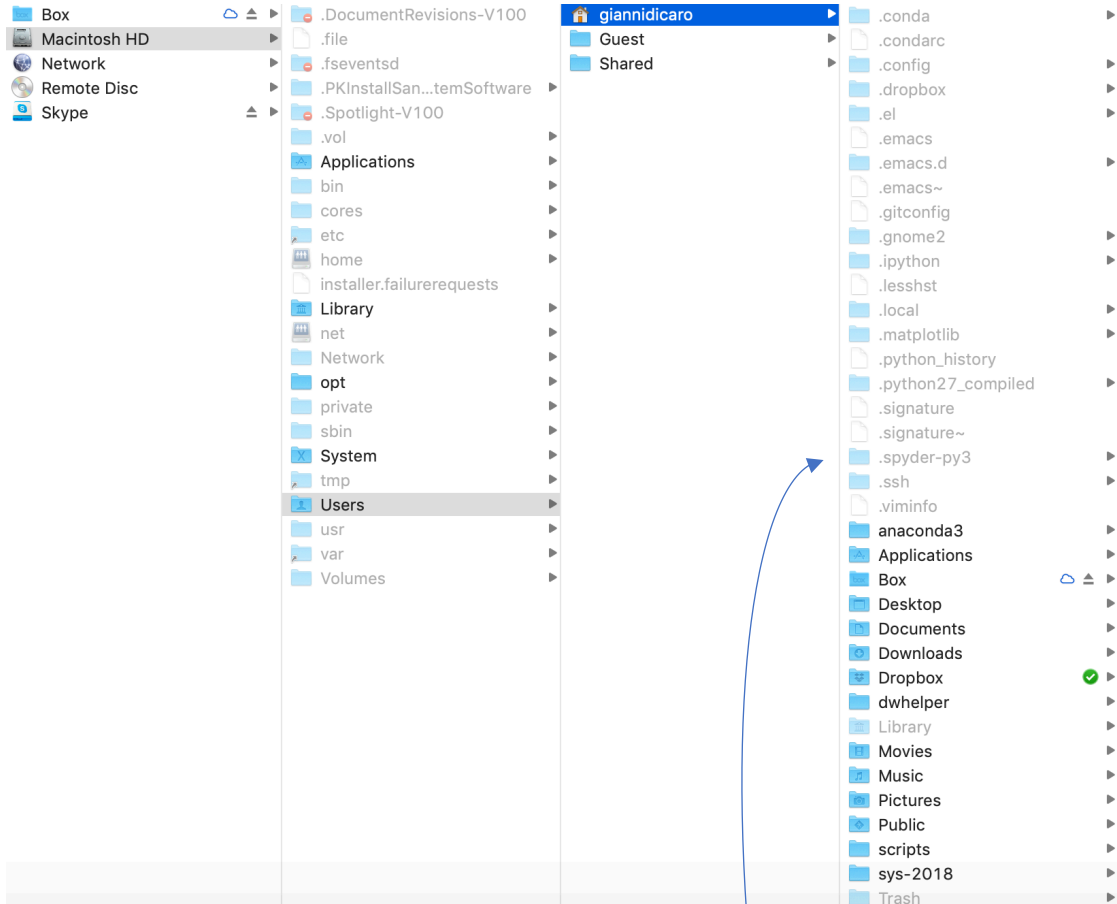
path

name

A folder can contain zero or more files
and/or zero or more sub-folders

Operations on the file system: get the current path / folder

- `os.getcwd()` : Returns a string with the current folder name (current **w**orking **d**irectory), including the prefix path



/Users/giannidicaro/.spyder-py3

```
import os
```

```
full_path = os.getcwd()
```

```
/Users/giannidicaro/.spyder-py3
```

Name of current working folder?

```
folder_start = full_path.rfind('/')  
folder_name = full_path[folder_start+1:]  
print(folder_name)
```

```
.spyder-py3
```

OS-dependent path strings

- Each OS (Windows, Mac, Unix, ...) can (and does) represent a path in different way ☹

Windows:

C:\Users\giannidicaro\anaconda3\bin\spyder.exe

Backslash is the separator used for directory levels

Unix / Mac:

/users/giannidicaro/anaconda3/bin/spyder

Forward slash is the separator used for directory levels

Using `os.getcwd()` is a way to get the path in the correct form for the used OS

OS-independent path strings: `os.path.join()`

- In *Windows* the backslash is the same as the **Escape character**, such that it has to be *protected* with `\` in order to be used 😞

```
file_path = current_folder\\sub_folder\\name.txt
```

- ✓ OS-independent path:

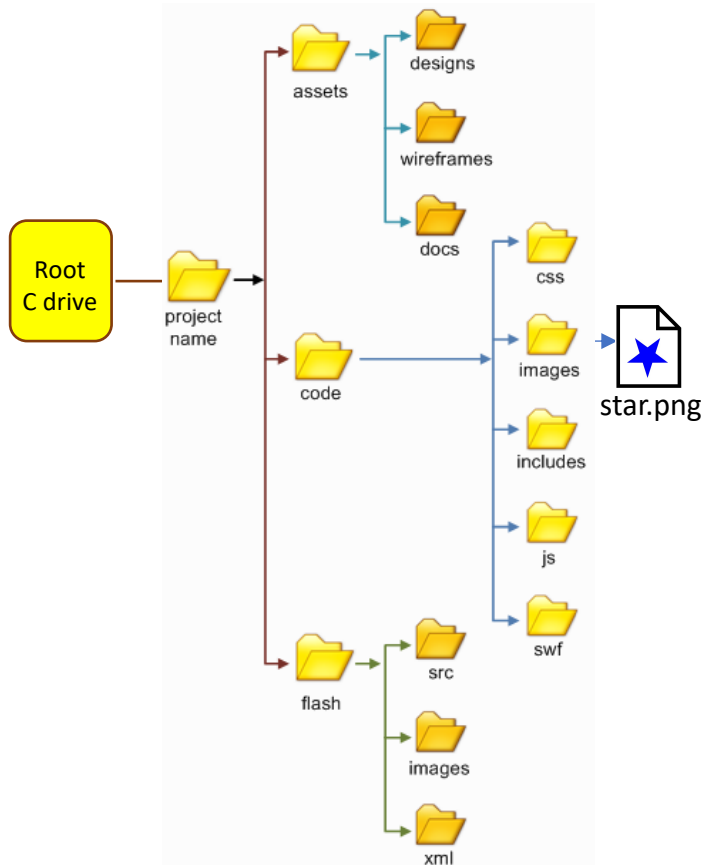
`path = os.path.join(comma_separated list of sub_folders)`

```
data_folder = os.path.join('source_data', 'dna_subject_1', 'experiment_3', 'trial_9')
file_to_open = os.path.join('data_folder', 'data_1_3_9.csv')
print(data_folder, '\n', file_to_open)
```

```
source_data/dna_subject_1/experiment_3/trial_9
data_folder/data_1_3_9.csv
```

Operations on the file system: check if a file exists

- `os.path.isfile(file_name)`: return True if file `file_name` exists, False otherwise; if `file_name` is not specified through a path, the existence of the file is referred to the current folder

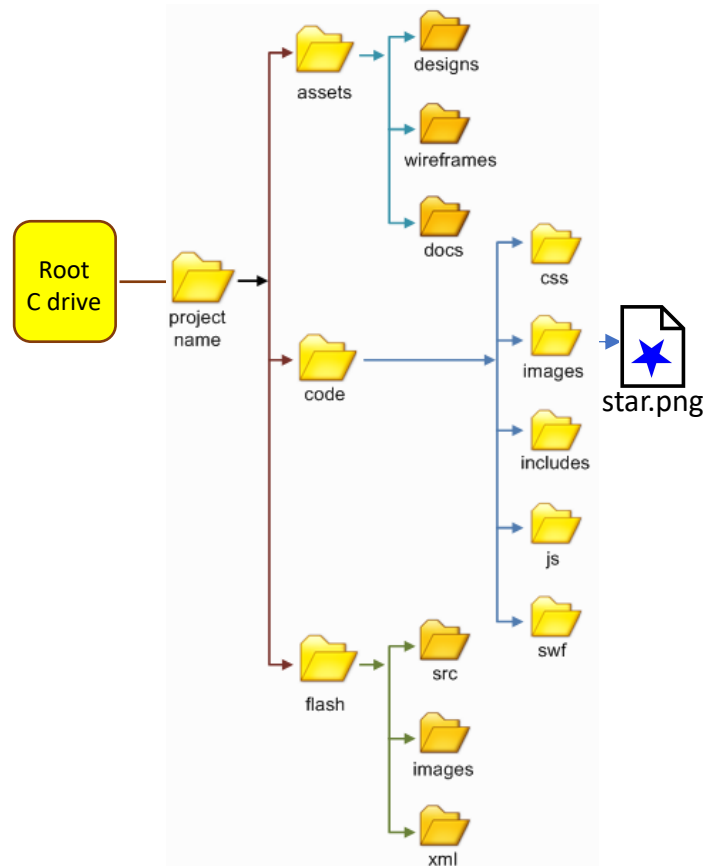


```
import os

if os.path.isfile('star.png'):
    print('File star.png is in the current folder')
elif os.path.isfile('/project name/assets/star.png'):
    print('File star.png is in folder assets')
else:
    print('File star.png is not in the file system')
```

Operations on the file system: check if a folder exists

- `os.path.isdir(folder_name)`: return True if folder `folder_name` exists, False otherwise; `folder_name` needs to include the full path, this holds also for the current path

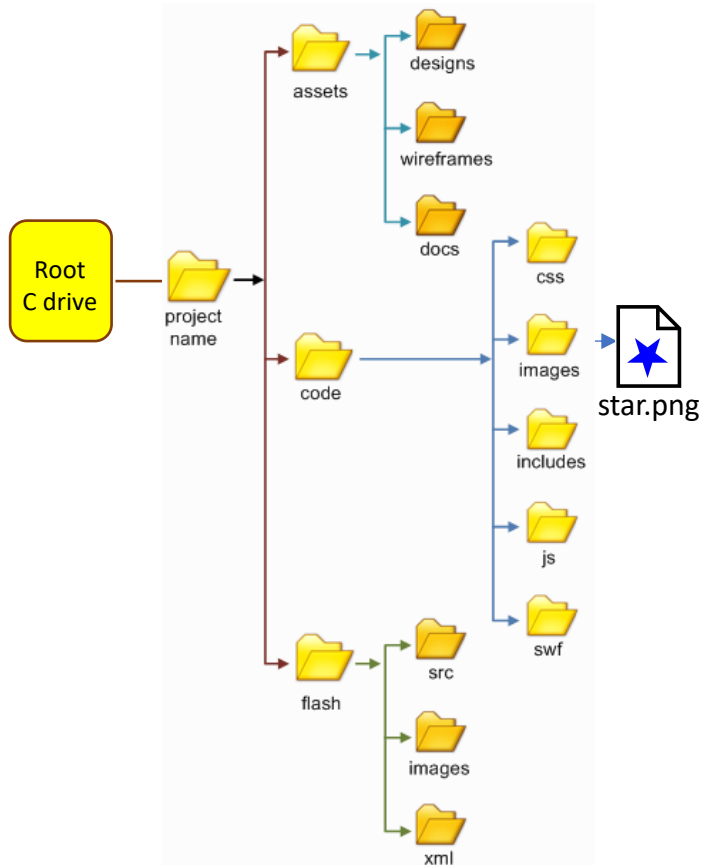


```
import os
```

```
if os.path.isdir('/project name/code/images'):  
    print('Folder exists in the file system')  
else:  
    print('Folder is not in the file system')
```


Operations on the file system: check if a named file or folder exists

- `os.path.exists(name)` : return `True` if name exists, and it can be either a file or a folder, `False` otherwise; no distinction is made between the two types for checking

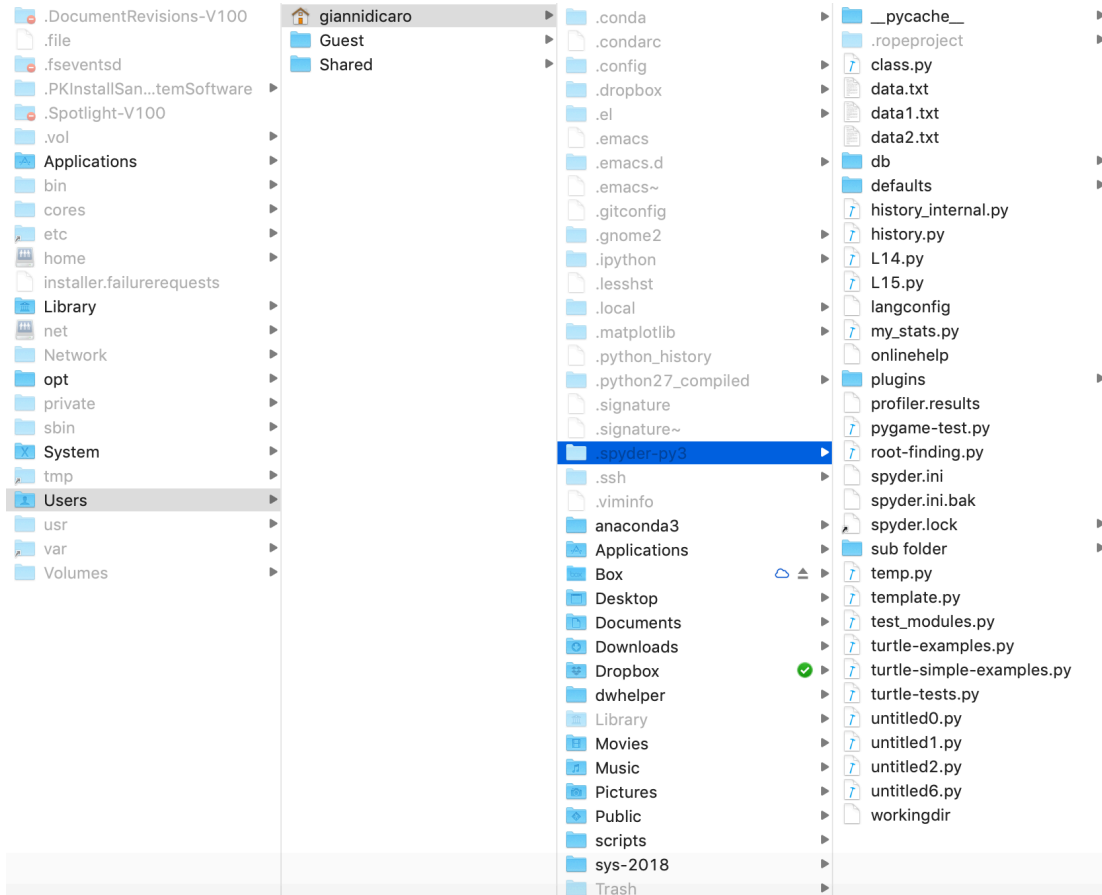


```
import os

if os.path.exists('/project
name/code/images'):
    print('Folder exists in the file system')
else:
    print('Folder is not in the file system')
```

Operations on the file system: get the list of files in current folder

- `os.listdir(path=None)` : Returns a list containing the names (as strings) of the files in the directory path. If path is None, the current directory, ' . / ', is listed



```
import os
```

```
file_list = os.listdir()  
print(file_list)
```

```
['untitled0.py', 'untitled1.py', 'turtle-examples.py', 'plugins', 'workingdir',  
'onlinehelp', '__pycache__', 'defaults', 'pygame-test.py', 'spyder.ini', 'turtle-  
tests.py', '.ropeproject', 'temp.py', 'class.py', 'profiler.results', 'template.py', 'db',  
'my_stats.py', 'turtle-simple-examples.py', 'data1.txt', 'untitled2.py', 'spyder.lock',  
'langconfig', 'L15.py', 'data2.txt', 'history_internal.py', 'untitled6.py',  
'test_modules.py', 'spyder.ini.bak', 'L14.py', 'data.txt', 'sub folder', 'root-  
finding.py', 'history.py']
```

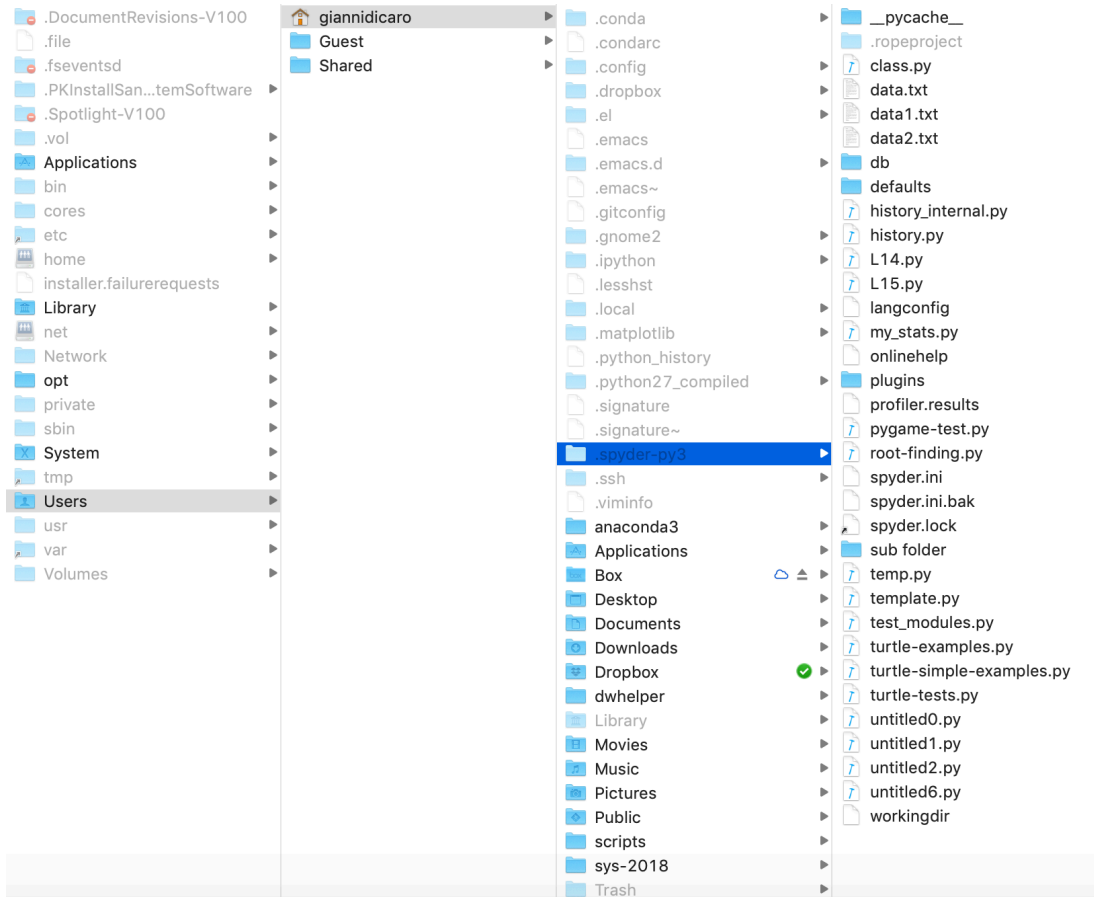
Is a specific file in the folder?

```
if 'L15.py' in file_list:  
    print('File L15.py is in current folder')
```

File L15.py is in the current folder

Operations on the file system: change the working folder

➤ `os.chdir(path)` : Changes the current path to path

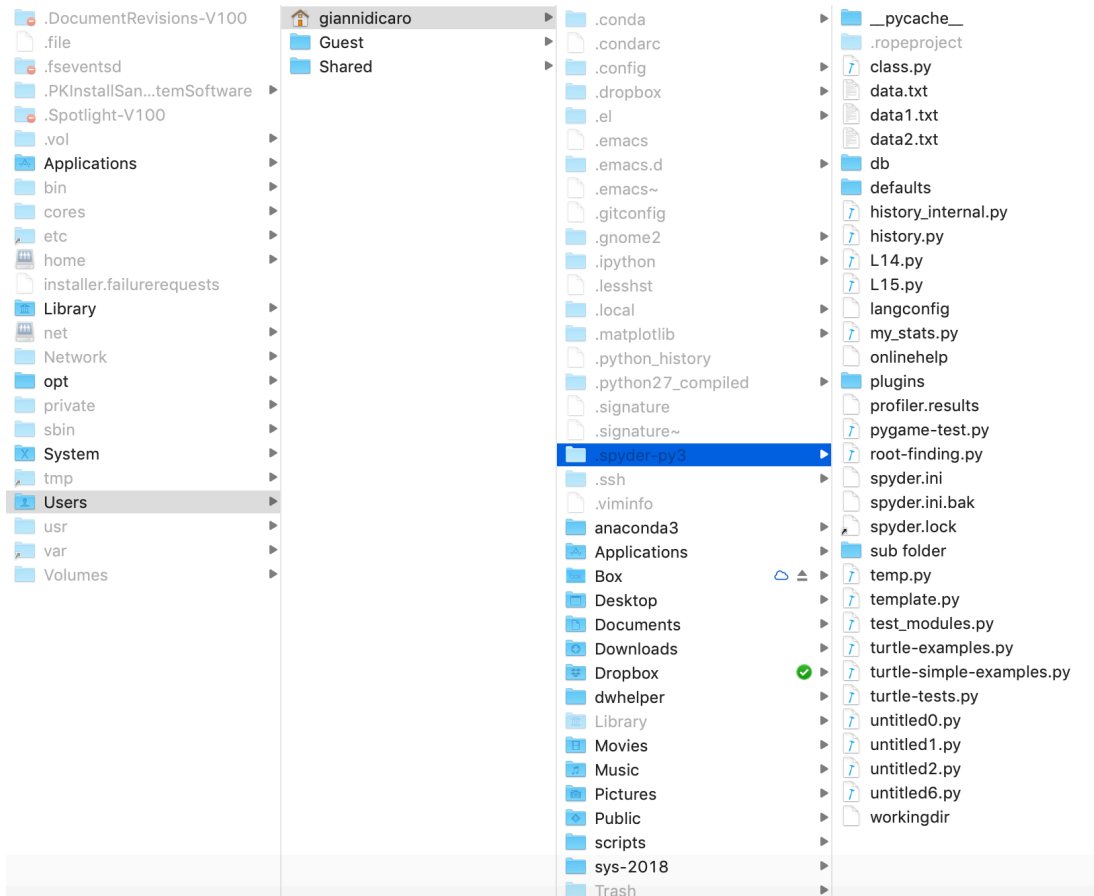


```
import os
print(os.getcwd())
os.chdir('/Applications')    #move anywhere in the fs
print(os.getcwd())
os.chdir('/Users/giannidicaro') #go back to the home folder
print(os.getcwd())
#print(os.listdir())
os.chdir('./anaconda3') #move down to a specific sub-folder
print(os.getcwd())
os.chdir('../')          #move to a folder up in the hierarchy
print(os.getcwd())
os.chdir('../.spyder-py3')
print(os.getcwd())
```

```
/Users/giannidicaro/.spyder-py3
/Applications
/Users/giannidicaro
/Users/giannidicaro/anaconda3
/Users/giannidicaro
/Users/giannidicaro/.spyder-py3
```

Operations on the file system: create a new folder

- `os.mkdir(path)` : Create a new folder at the given path, if path only contains a name (and not a full path) the folder is created as a sub-folder of the current one



```
import os
os.mkdir('temp')
```

How do we check that the new folder is there?

```
for ff in os.listdir():
    if os.path.isdir(ff):
        print(ff)
```

```
temp
plugins
__pycache__
defaults
.ropeproject
db
sub folder
```

Operations on the file system: remove a folder

➤ `os.remove(path)`: Remove the folder at the given path, the folder must be empty, otherwise a `PermissionError` exception is generated

- If the folder is empty (and exists) this works with no errors

```
import os
if os.path.isdir('temp'):
    os.rmdir('temp')
```

- Check first whether the folder is empty or not, remove all folder files

```
import os
if os.path.isdir('temp'):
    os.chdir('temp')
    for file in os.listdir():
        os.remove(file)
    os.chdir('../')
    os.rmdir('temp')
```

Operations on the file system: get status / stats of a file

- `os.stat(file_name)`: Returns a data structure with various information about `file_name`, information is stored in separate fields of the structure, that can be accessed individually with the dot notation

- **st_mode** – protection bits.
- **st_ino** – inode number.
- **st_dev** – device.
- **st_nlink** – number of hard links.
- **st_uid** – user id of owner.
- **st_gid** – group id of owner.
- **st_size** – size of file, in bytes.
- **st_atime** – time of most recent access.
- **st_mtime** – time of most recent content modification.
- **st_ctime** – time of most recent metadata change.

```
import os
stat_info = os.stat('L15.py')
print(stat_info)
print('Size:', stat_info.st_size)
print('Last modification time:', stat_info.st_mtime)
```

```
os.stat_result(st_mode=33188, st_ino=5499079, st_dev=16777220,
st_nlink=1, st_uid=501, st_gid=20, st_size=2810,
st_atime=1551870605, st_mtime=1551870605,
st_ctime=1551870605)
Size: 2810
Last modification time: 1551872218.209256
```

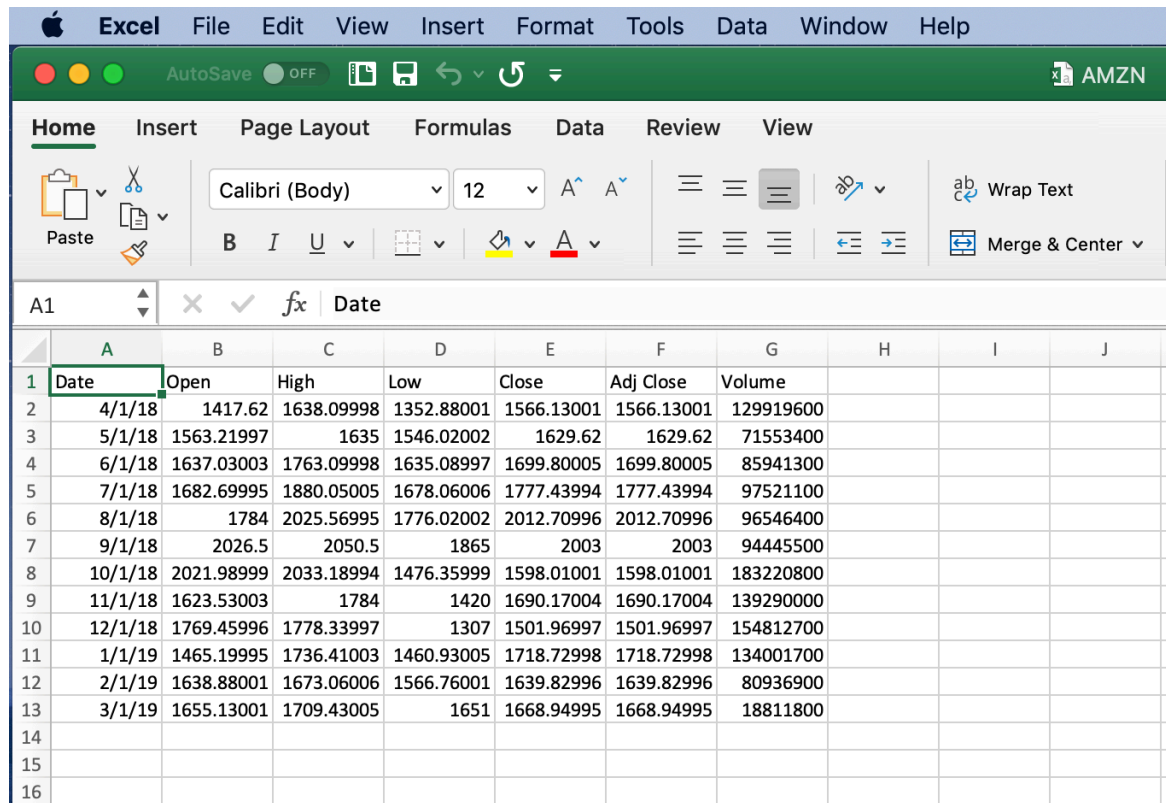
```
[~/spyder-py3$ ls -al L15.py
-rw-r--r--@ 1 giannidicaro  staff  3007 Mar  6 14:22 L15.py
~/spyder-py3$
```

```
from datetime import datetime
print('Last modification date: ', end='')
print(datetime.fromtimestamp(stat_info.st_mtime).strftime('%Y-%m-%d %H:%M:%S'))
```

Last modification date: 2019-03-06 14:36:58

Special data files: Comma Separated Values (CSV)

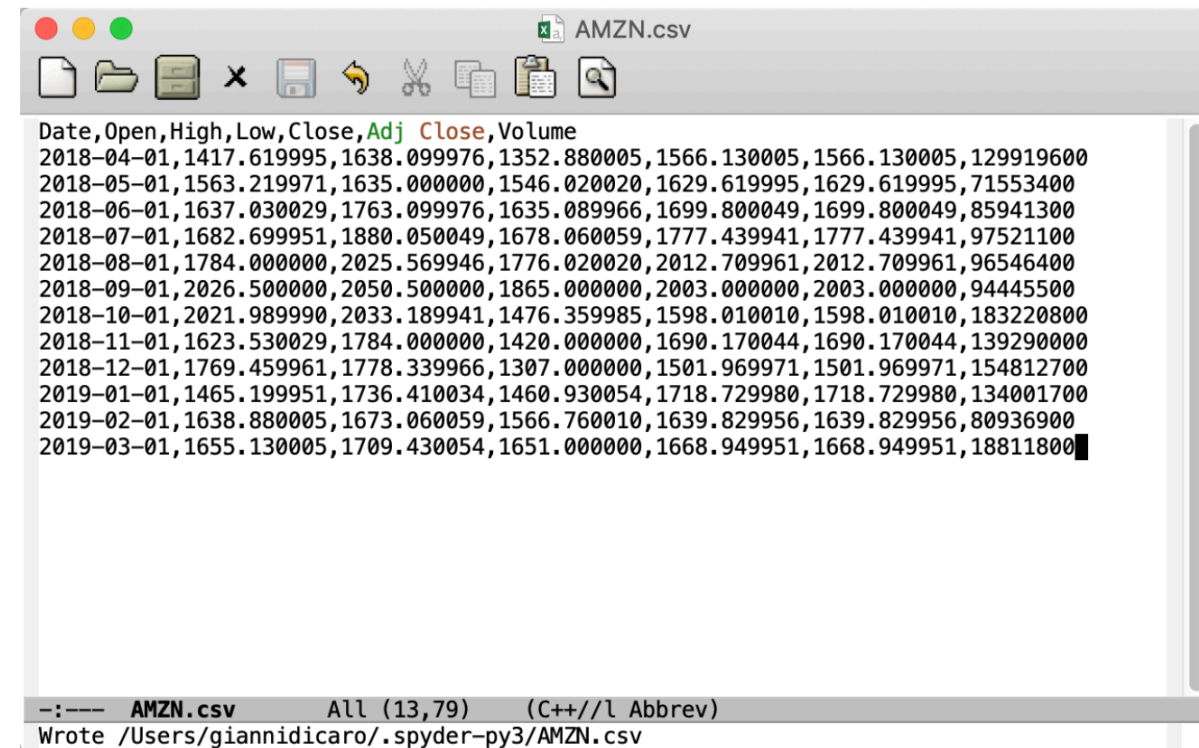
- ❖ **CSV:** A quite flexible and compact format which is around since long time, and it's the main format used by popular spreadsheet programs such as Excel.
- Many data repositories make use of CSV as one their standard formats for data.



The screenshot shows the Microsoft Excel interface with a CSV file named 'AMZN' imported. The data is displayed in a table with columns: Date, Open, High, Low, Close, Adj Close, and Volume. The data covers the period from April 2018 to March 2019.

	A	B	C	D	E	F	G	H	I	J
1	Date	Open	High	Low	Close	Adj Close	Volume			
2	4/1/18	1417.62	1638.09998	1352.88001	1566.13001	1566.13001	129919600			
3	5/1/18	1563.21997	1635	1546.02002	1629.62	1629.62	71553400			
4	6/1/18	1637.03003	1763.09998	1635.08997	1699.80005	1699.80005	85941300			
5	7/1/18	1682.69995	1880.05005	1678.06006	1777.43994	1777.43994	97521100			
6	8/1/18	1784	2025.56995	1776.02002	2012.70996	2012.70996	96546400			
7	9/1/18	2026.5	2050.5	1865	2003	2003	94445500			
8	10/1/18	2021.98999	2033.18994	1476.35999	1598.01001	1598.01001	183220800			
9	11/1/18	1623.53003	1784	1420	1690.17004	1690.17004	139290000			
10	12/1/18	1769.45996	1778.33997	1307	1501.96997	1501.96997	154812700			
11	1/1/19	1465.19995	1736.41003	1460.93005	1718.72998	1718.72998	134001700			
12	2/1/19	1638.88001	1673.06006	1566.76001	1639.82996	1639.82996	80936900			
13	3/1/19	1655.13001	1709.43005	1651	1668.94995	1668.94995	18811800			
14										
15										
16										

With Excel

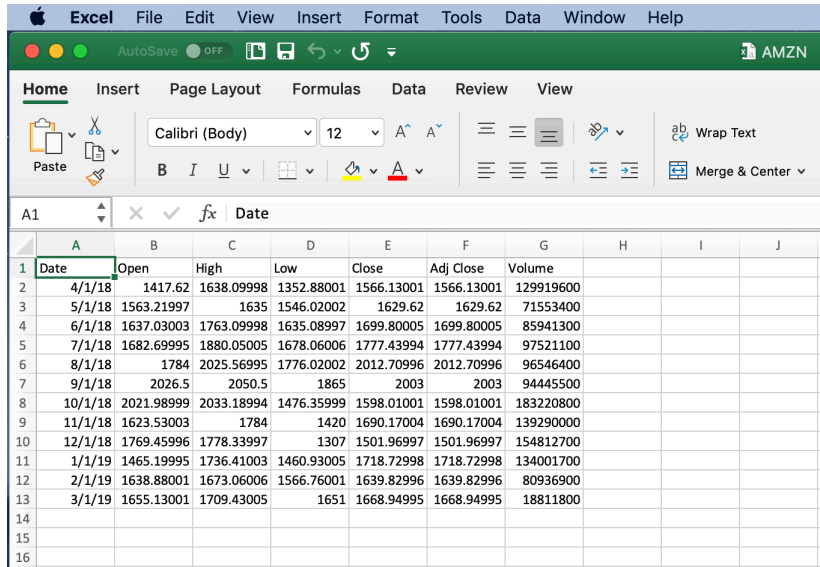


The screenshot shows a text editor window titled 'AMZN.csv' displaying the raw CSV data. The data is a single line of text with columns separated by commas. The columns are: Date, Open, High, Low, Close, Adj Close, and Volume. The data covers the period from April 2018 to March 2019.

```
Date,Open,High,Low,Close,Adj Close,Volume
2018-04-01,1417.619995,1638.099976,1352.880005,1566.130005,1566.130005,129919600
2018-05-01,1563.219971,1635.000000,1546.020020,1629.619995,1629.619995,71553400
2018-06-01,1637.030029,1763.099976,1635.089966,1699.800049,1699.800049,85941300
2018-07-01,1682.699951,1880.050049,1678.060059,1777.439941,1777.439941,97521100
2018-08-01,1784.000000,2025.569946,1776.020020,2012.709961,2012.709961,96546400
2018-09-01,2026.500000,2050.500000,1865.000000,2003.000000,2003.000000,94445500
2018-10-01,2021.989990,2033.189941,1476.359985,1598.010010,1598.010010,183220800
2018-11-01,1623.530029,1784.000000,1420.000000,1690.170044,1690.170044,139290000
2018-12-01,1769.459961,1778.339966,1307.000000,1501.969971,1501.969971,154812700
2019-01-01,1465.199951,1736.410034,1460.930054,1718.729980,1718.729980,134001700
2019-02-01,1638.880005,1673.060059,1566.760010,1639.829956,1639.829956,80936900
2019-03-01,1655.130005,1709.430054,1651.000000,1668.949951,1668.949951,18811800
```

With a text editor

CSV: Tabular data with a predefined separator



	A	B	C	D	E	F	G	H	I	J
1	Date	Open	High	Low	Close	Adj Close	Volume			
2	4/1/18	1417.62	1638.09998	1352.88001	1566.13001	1566.13001	129919600			
3	5/1/18	1563.21997	1635	1546.02002	1629.62	1629.62	71553400			
4	6/1/18	1637.03003	1763.09998	1635.08997	1699.80005	1699.80005	85941300			
5	7/1/18	1682.69995	1880.05005	1678.06006	1777.43994	1777.43994	97521100			
6	8/1/18	1784	2025.56995	1776.02002	2012.70996	2012.70996	96546400			
7	9/1/18	2026.5	2050.5	1865	2003	2003	94445500			
8	10/1/18	2021.98999	2033.18994	1476.35999	1598.01001	1598.01001	183220800			
9	11/1/18	1623.53003	1784	1420	1690.17004	1690.17004	139290000			
10	12/1/18	1769.45996	1778.33997	1307	1501.96997	1501.96997	154812700			
11	1/1/19	1465.19995	1736.41003	1460.93005	1718.72998	1718.72998	134001700			
12	2/1/19	1638.88001	1673.06006	1566.76001	1639.82996	1639.82996	80936900			
13	3/1/19	1655.13001	1709.43005	1651	1668.94995	1668.94995	18811800			
14										
15										
16										

Rows: **M** data record,
each with N (labeled) fields

- ✓ As a default, fields are separated by commas
- ✓ Other custom separators can be defined and employed

Columns: **N** labels

row 1: column 1, column 2, column 3, , column N

row 2: column 1, column 2, column 3, , column N

row 3: column 1, column 2, column 3, , column N

.....

row M: column 1, column 2, column 3, , column N

Matrix / Dictionary

CSV: Tabular data with a predefined separator

- ❖ First row may / should include the labels for the column data (however, it's optional)

```
name, address, id, age, sex
J. Smith, Falcon Tower West-Bay, 532720 , 38, M
A. White, Tower 99 The Pearl, 33145, 29, F
```

- ❖ This would work but we need to know the meaning of the fields in advance

```
J. Smith, Falcon Tower West-Bay, 532720 , 38, M
A. White, Tower 99 The Pearl, 33145, 29, F
```

Note: there are **spaces**, we'll see that they are irrelevant, **the (comma) separator matters!**

CSV example file: Mall customer statistics

```
CustomerID,Gender,Age,Annual Income (k$),Spending Score (1-100)
1, Male, 19, 15, 39
2, Male, 21, 15, 81
3, Female, 20, 16, 6
4, Female, 23, 16, 77
5, Female, 31, 17, 40
6, Female, 22, 17, 76
7, Female, 35, 18, 6
8, Female, 23, 18, 94
9, Male, 64, 19, 3
10, Female, 30, 19, 72
```

... 200 records

CSV module: opening and getting the iterator

- ✓ The **csv** module provides a number of methods to effectively and efficiently deal with the basic reading and writing operations on CSV files

```
import csv

file_path = '/Users/giannidicaro/Box/110-Fall19/csv/Mall_Customers.csv'
file_name = file_path.split('/')[-1]

f_csv = open(file_path)

csv_data = csv.reader(f_csv, delimiter=',')
```

- **csv.reader(file_handler, <delimiter>)** returns a **csv file iterator**, the variable `csv_data`
- The csv file iterator is an object that at each call **reads the next line** in the file and returns it into a **list of strings**, where each list element is a string with a field value, identified based on the given delimiter

CSV module: reading raw

- ✓ We can use the `csv_data` iterator to **loop over all the records** in the file, reading record by record in the variable `row`, which is a list of strings

```
line_count = 0
for row in csv_data:
    print('Row {:d}: {} (length: {})'.format(line_count, row, len(row)))
    line_count += 1
```

Row 0: ['CustomerID', 'Gender', 'Age', 'Annual Income (k\$)', 'Spending Score (1-100)'] (length: 5)

Row 1: ['1', ' Male', ' 19', ' 15', ' 39'] (length: 5)

Row 2: ['2', ' Male', ' 21', ' 15', ' 81'] (length: 5)

Row 3: ['3', ' Female ', '20', '16', '6'] (length: 5)

Row 4: ['4', 'Female', '23 ', '16 ', '77'] (length: 5)

Row 5: ['5', 'Female', '31', '17', '40'] (length: 5)

Row 6: ['6', 'Female', '22', '17', '76'] (length: 5)

Row 7: ['7', 'Female', '35', '18', '6'] (length: 5)

Row 8: ['8', 'Female', '23', '18', '94'] (length: 5)

Row 9: ['9', 'Male', '64', '19', '3'] (length: 5)

Row 10: ['10', 'Female', '30', '19', '72'] (length: 5)

Note: white spaces do not harm, but stay there because they are part of the fields

Reading/printing using fields

- ✓ Let's make a nicer reading / printing by exploiting the known organization in fields

```
f_csv.seek(0) # rewind the file
line_count = 0
for row in csv_data:
    if line_count == 0: # header with labels
        columns = len(row)
        print('File {} contains {:d} columns: {:s}'.format(file_name,
            columns, ' - '.join(row)))
    else:
        print('ID {} is a {:>6s} of {:2d} years making {:3d}$ / year \
            and has a spending score of {:3d}'.format(int(row[0]),
            row[1], int(row[2]), int(row[3]), int(row[4])))
    line_count += 1
f_csv.close()
```

Formatted result

File Mall_Customers.csv contains 5 columns: CustomerID - Gender - Age - Annual Income (k\$) - Spending Score (1-100)

ID 1 is a Male of 19 years making 15\$/year and has a spending score of 39

ID 2 is a Male of 21 years making 15\$/year and has a spending score of 81

ID 3 is a Female of 20 years making 16\$/year and has a spending score of 6

ID 4 is a Female of 23 years making 16\$/year and has a spending score of 77

ID 5 is a Female of 31 years making 17\$/year and has a spending score of 40

ID 6 is a Female of 22 years making 17\$/year and has a spending score of 76

ID 7 is a Female of 35 years making 18\$/year and has a spending score of 6

ID 8 is a Female of 23 years making 18\$/year and has a spending score of 94

ID 9 is a Male of 64 years making 19\$/year and has a spending score of 3

ID 10 is a Female of 30 years making 19\$/year and has a spending score of 72

One-shot reading

```
file_path = '/Users/giannidicaro/ Box/110-Fall19/csv/Mall_Customers.csv'
f_csv = open(file_path)
csv_data = csv.reader(f_csv, delimiter=',')

all_data = list(f_csv)

print(all_data)
f_csv.close()
```

One big list of strings, one per data record

```
['CustomerID,Gender,Age,Annual Income (k$),Spending Score (1-100)\n', '1, Male, 19, 15, 39\n', '2, Male, 21, 15, 81\n', '3, Female ,20,16,6\n', '4,Female,23 ,16 ,77\n', '5,Female,31,17,40\n', '6,Female,22,17,76\n', '7,Female,35,18,6\n', '8,Female,23,18,94\n', '9,Male,64,19,3\n', '10,Female,30,19,72\n', '11,Male,67,19,14\n', '12,Female,35,19,99\n', '13,Female,58,20,15\n', '14,Female,24,20,77\n', '15,Male,37,20,13\n', '16,Male,22,20,79\n', '17,Female,35,21,35\n', '18,Male,20,21,66\n', '19,Male,52,23,29\n', ...]
```

What about a different delimiter?

- The new file has the same contents but it's using `;` as a field delimiter
- ✓ Just let csv module to know about it in the `csv.reader()` call

```
file_path = '/Users/giannidicaro/Box/110-Fall19/csv/Mall_Customers-d2.csv'
f2_csv = open(file_path)
csv_data = csv.reader(f2_csv, delimiter=';')
line_count = 0
for row in csv_data:
    print('Line: {}'.format(' '.join(row)))
    line_count += 1
f2_csv.close()
```


What about a different delimiter with more than one character?

```
file_path = '/Users/giannidicaro/Box/110-Fall19/csv/Mall_Customers-d2.csv'
f3_csv = open(file_path)
try:
    csv_data = csv.reader(f3_csv, delimiter='--')
except:
    print("Delimiter must be 1-character string!")
else:
    line_count = 0
    for row in csv_data:
        print('Line: {}'.format(' '.join(row)))
        line_count += 1
finally:
    f3_csv.close()
```

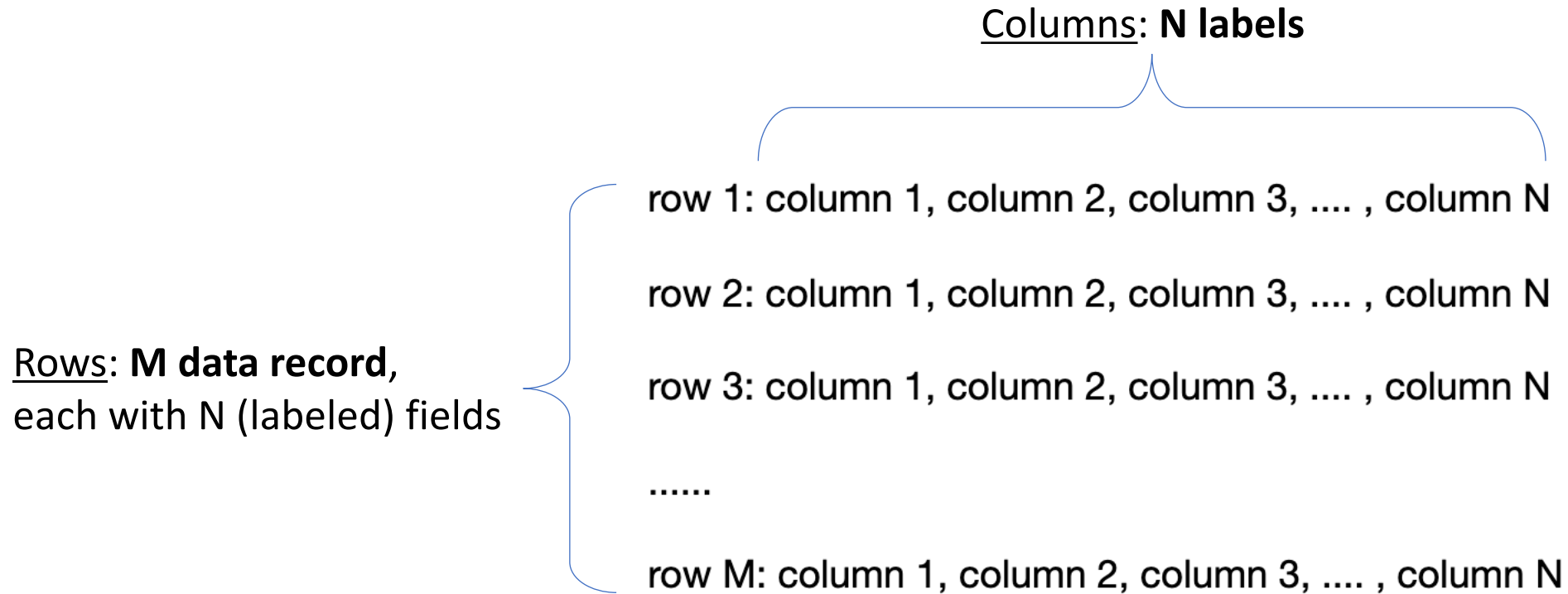
TypeError! delimiter must be 1-character string!

What about data that contains commas in the fields?

Three possible strategies, **all requiring modifying the original csv file**:

1. Use a **different delimiter** in the csv file (e.g., ';')
2. Wrap the data containing commas in **quotes**: the string between the quotes is not evaluated for the delimiter. The character used for quoting needs to be specified by the **quotechar** optional parameter if different from `"`, which is the default
3. **Escape the delimiter character** in the data: adding `\` *protects* the character from being evaluated as a delimiter. If an escape character is used, it must be specified using the **escapechar** optional parameter.

CSV files are tabular data / dictionaries



- ❖ Column names are the **keys**
- ❖ Each row is a dictionary with N (key, value) pairs
- ❖ `tabular_data[i][key]` access field `key` in the `i`-th record

CSV files are tabular data / dictionaries

```
file_path = '/Users/giannidicarò/ Box/110-Fall19/csv/Mall_Customers.csv'
f_csv = open(file_path)
csv_data = csv.reader(f_csv, delimiter=',')

num_of_keys = len(csv_data.fieldnames)
keys = csv_data.fieldnames
print('File {} has size {} bytes and contains {:d} keys: {:s}\n'.format(file_name,
    size, num_of_keys, ' - '.join(keys)))

tabular_csv = list(csv_data)
# tabular_csv is a list of records in the form of ordered dictionaries

print(tabular_csv[1]['name'])

for i in range(len(tabular_csv)):
    for k in range(num_of_keys):
        print('{:9s} '.format(tabular_csv[i][keys[k]]), end='')
    print('\n')

f_csv.close()
```