Python R training course - Session 4

Van-Duyet LE, Thinh PHAM

August 27, 2017

Lecture list

1. Introduction to Python
2. The Python Standard Library, if/loops statements
3. Vector/matrix structures, `numpy` library
4. **Python data types, File Processing,** `Pandas` **library**
   - Structure of Python files. Read/create a file.
   - Read csv file using `Pandas` library
5. Functions in Python, Debugging.
6. Introduction to R, R for Python programmers.
7. Import data, plot data.
8. Data Mining in Python/R.

1 File Processing

Python provides basic functions and methods necessary to manipulate files by default. You can do most of the file manipulation using a file object.

1.1 Opening and Closing Files

Before you can read or write a file, you have to open it using Python's built-in **open()** function. This function creates a file object, which would be utilized to call other support methods associated with it.

```
f = open(file_name [, access_mode][, buffering])
```

- **file_name**: The file_name argument is a string value that contains the name of the file that you want to access.
- **access_mode**: The access_mode determines the mode in which the file has to be opened, i.e., read, write, append, etc. A complete list of possible values is given below in the table. This is optional parameter and the default file access mode is read (r).

- **buffering**: If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action is performed with the indicated buffer size. If negative, the buffer size is the system default(default behavior).

Here is a list of the different modes of opening a file:

Mode	Description
r	Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
rb	Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
r+	Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
rb+	Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
w	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
wb	Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
w+	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
wb+	Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
a	Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
ab	Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
a+	Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
ab+	Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

Once a file is opened and you have one file object, you can get various information related to that file.

Here is a list of all attributes related to file object:

Attribute	Description
file.closed	Returns true if file is closed, false otherwise.
file.mode	Returns access mode with which file was opened.
file.name	Returns name of the file.
file.softspace	Returns false if space explicitly required with print, true otherwise.

```
In [19]: f = open("text_file.txt", "rw+")
         f.read()

Out[19]: 'Hello!'

In [20]: print "Name of the file: ", f.name
         print "Closed or not : ", f.closed
         print "Opening mode : ", f.mode
         print "Softspace flag : ", f.softspace

Name of the file:  text_file.txt
Closed or not :  False
Opening mode :  rw+
Softspace flag :  0
```

The **close()** method of a file object flushes any unwritten information and closes the file object, after which no more writing can be done.

Python automatically closes a file when the reference object of a file is reassigned to another file. It is a good practice to use the close() method to close a file.

Syntax:

```
fileObject.close()
```

```
In [22]: # Open a file
         fo = open("text_file.txt", "wb")
         print "Name of the file: ", fo.name

         # Close opend file
         fo.close()

Name of the file:  text_file.txt
```

1.2 Reading and Writing Files

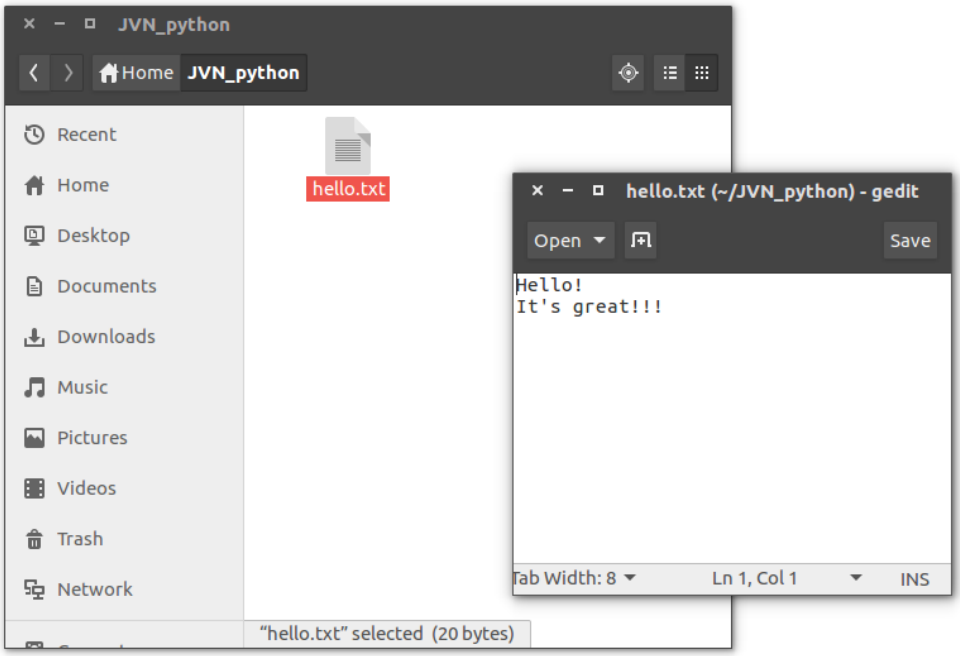
The file object provides a set of access methods to make our lives easier. We would see how to use **read()** and **write()** methods to read and write files.

The **write()** method writes any string to an open file. It is important to note that Python strings can have binary data and not just text.

Example:

```
In [25]: # Open a file
         f = open("hello.txt", "w")
         f.write("Hello!\nIt's great!!!");

         # Close opend file
         f.close()
```



The **read()** method reads a string from an open file. It is important to note that Python strings can have binary data. apart from text data.

Syntax

```
fileObject.read([count]);
```

Here, passed parameter is the number of bytes to be read from the opened file. This method starts reading from the beginning of the file and if count is missing, then it tries to read as much as possible, maybe until the end of file.

Example

Let's take a file hello.txt, which we created above.

```
In [35]: f = open("hello.txt", "r+")
         print f.read()
         f.close()

Hello!
It's great!!!
```

1.3 Renaming and Deleting Files

Python **os** module provides methods that help you perform file-processing operations, such as renaming and deleting files.

To use this module you need to import it first and then you can call any related functions.

1.3.1 The rename() Method

The **rename()** method takes two arguments, the current filename and the new filename:

```
os.rename(current_file_name, new_file_name)
```

```
In [36]: import os

         os.rename("hello.txt", "hi.txt")
```

1.3.2 The remove() Method

You can use the **remove()** method to delete files by supplying the name of the file to be deleted as the argument.

Syntax

```
os.remove(file_name)
```

```
In [37]: import os

         os.remove("hi.txt")
```

```
In [38]: # Remove not exsists file

         os.remove("some_file.txt")
```

```
---------------------------------------------------------------------------
OSError                                   Traceback (most recent call last)

<ipython-input-38-6b72632a7e6b> in <module>()
  1 # Remove not exsists file
  2
----> 3 os.remove("some_file.txt")


OSError: [Errno 2] No such file or directory: 'some_file.txt'
```

1.4 Directories in Python

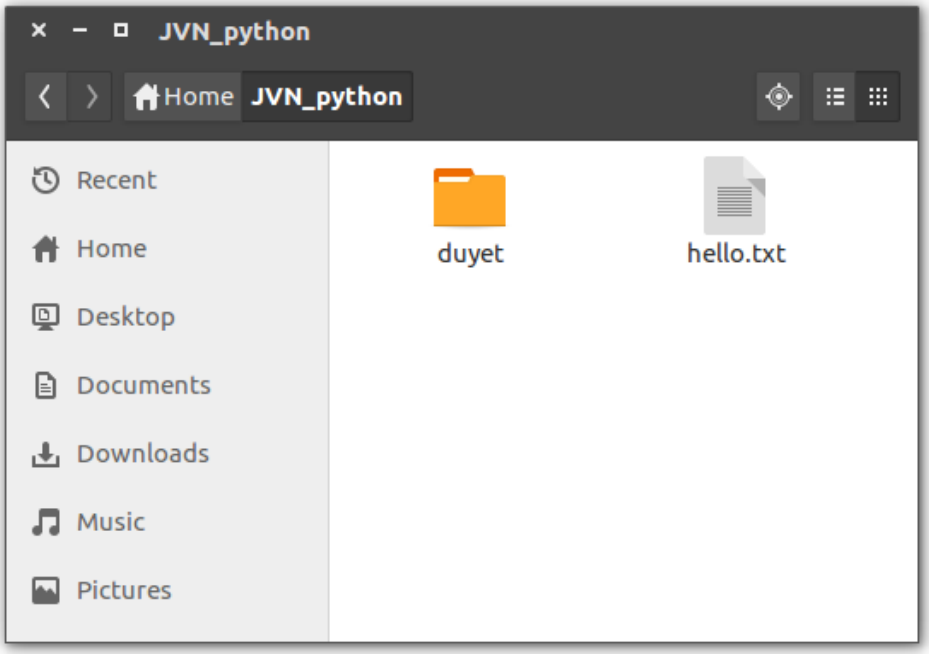
The **os** module has several methods that help you create, remove, and change directories.

1.4.1 The mkdir() Method

You can use the mkdir() method of the os module to create directories in the current directory

```
os.mkdir("newdir")
```

```
In [39]: os.mkdir("duyet")
```



1.4.2 The chdir() Method

You can use the **chdir()** method to change the current directory.

```
os.chdir("newdir")
```

1.4.3 The getcwd() Method

The **getcwd()** method displays the current working directory.

```
In [41]: print os.getcwd()
```

```
/home/duyetdev/project/Python-R-Basic
```

1.4.4 The rmdir() Method

The rmdir() method deletes the directory, which is passed as an argument in the method.

```
os.rmdir('dirname')
```

```
In [50]: os.rmdir("./duyet")
```

```
In [ ]:
```

```
In [ ]:
```

Ex: Creates the following folder structure

2 Pandas

Pandas is a package of fast, efficient data analysis tools for Python

Just as NumPy provides the basic array data type plus core array operations, pandas

- defines fundamental structures for working with data and
- endows them with methods that facilitate operations such as
  - reading in data
  - adjusting indices
  - working with dates and time series
  - sorting, grouping, re-ordering and general data munging [1]
  - dealing with missing values, etc., etc.

More sophisticated statistical functionality is left to other packages, such as statsmodels and scikit-learn, which are built on top of pandas

```
In [52]: import numpy as np
         import pandas as pd
```

2.1 Series

Two important data types defined by pandas are **Series** and **DataFrame**

You can think of a Series as a "column" of data, such as a collection of observations on a single variable

A DataFrame is an object for storing related columns of data

Let's start with Series

```
In [53]: s = pd.Series(np.random.randn(4), name='daily returns')
         s

Out[53]: 0    0.109632
         1    0.125849
         2    1.593836
         3   -1.826518
         Name: daily returns, dtype: float64
```

Here you can imagine the indices `0, 1, 2, 3` as **indexing** four listed companies, and the values being daily returns on their shares.

Pandas **Series** are built on top of NumPy arrays, and support many similar operations

```
In [54]: s * 100

Out[54]: 0     10.963232
         1     12.584910
         2    159.383605
         3   -182.651820
         Name: daily returns, dtype: float64

In [55]: s.abs()
```

```
Out[55]: 0    0.109632
         1    0.125849
         2    1.593836
         3    1.826518
         Name: daily returns, dtype: float64

In [57]: s[s >= 1]

Out[57]: 2    1.593836
         Name: daily returns, dtype: float64

In [58]: s.describe()

Out[58]: count    4.000000
         mean     0.000700
         std      1.402894
         min     -1.826518
         25%     -0.374405
         50%      0.117741
         75%      0.492846
         max      1.593836
         Name: daily returns, dtype: float64
```

But their indices are more flexible

```
In [59]: s.index = ['AMZN', 'AAPL', 'MSFT', 'GOOG']
         s

Out[59]: AMZN    0.109632
         AAPL    0.125849
         MSFT    1.593836
         GOOG   -1.826518
         Name: daily returns, dtype: float64

In [60]: s['GOOG']

Out[60]: -1.8265181959162498
```

2.2 DataFrames

While a Series is a single column of data, a DataFrame is several columns, one for each variable.

Here's the contents of *test_pwt.csv*

```
"country","country isocode","year","POP","XRAT","tcgdp","cc","cg"
"Argentina","ARG","2000","37335.653","0.9995","295072.21869","75.716805379","5.5788042896"
"Australia","AUS","2000","19053.186","1.72483","541804.6521","67.759025993","6.7200975332"
"India","IND","2000","1006300.297","44.9416","1728144.3748","64.575551328","14.072205773"
"Israel","ISR","2000","6114.57","4.07733","129253.89423","64.436450847","10.266688415"
"Malawi","MWI","2000","11801.505","59.543808333","5026.2217836","74.707624181","11.658954494"
"South Africa","ZAF","2000","45064.098","6.93983","227242.36949","72.718710427","5.7265463933"
"United States","USA","2000","282171.957","1","9898700","72.347054303","6.0324539789"
"Uruguay","URY","2000","3219.793","12.099591667","25255.961693","78.978740282","5.108067988"
```

```
In [61]: df = pd.read_csv('https://github.com/QuantEcon/QuantEcon.lectures.code/raw/master/panda

In [62]: df.head()

Out[62]:      country country isocode  year          POP       XRAT         tcgdp  \
         0  Argentina             ARG  2000    37335.653   0.999500  2.950722e+05
         1  Australia             AUS  2000    19053.186   1.724830  5.418047e+05
         2      India             IND  2000  1006300.297  44.941600  1.728144e+06
         3     Israel             ISR  2000     6114.570   4.077330  1.292539e+05
         4     Malawi             MWI  2000    11801.505  59.543808  5.026222e+03

                   cc         cg
         0  75.716805   5.578804
         1  67.759026   6.720098
         2  64.575551  14.072206
         3  64.436451  10.266688
         4  74.707624  11.658954

In [63]: df[2:4]

Out[63]:   country country isocode  year          POP      XRAT         tcgdp  \
         2   India             IND  2000  1006300.297  44.94160  1.728144e+06
         3  Israel             ISR  2000     6114.570   4.07733  1.292539e+05

                   cc         cg
         2  64.575551  14.072206
         3  64.436451  10.266688
```

To select columns, we can pass a list containing the names of the desired columns represented as strings

```
In [64]: df[['country', 'tcgdp']]

Out[64]:          country         tcgdp
         0      Argentina  2.950722e+05
         1      Australia  5.418047e+05
         2          India  1.728144e+06
         3         Israel  1.292539e+05
         4         Malawi  5.026222e+03
         5   South Africa  2.272424e+05
         6  United States  9.898700e+06
         7        Uruguay  2.525596e+04
```

To select both rows and columns using integers, the iloc attribute should be used with the format `.iloc[rows,columns]`

```
In [65]: df.iloc[2:5,0:4]

Out[65]:   country country isocode  year          POP
         2   India             IND  2000  1006300.297
         3  Israel             ISR  2000     6114.570
         4  Malawi             MWI  2000    11801.505
```

To select rows and columns using a mixture of integers and labels, the loc attribute can be used in a similar way

```
In [70]: df.loc[df.index[2:5], ['year', 'country']]
```

```
Out[70]:    year country
         2  2000   India
         3  2000  Israel
         4  2000  Malawi
```

Let's imagine that we're only interested in population and total GDP (tcgdp).

One way to strip the data frame df down to only these variables is to overwrite the dataframe using the selection method described above

```
In [71]: df = df[['country','POP','tcgdp']]
         df
```

```
Out[71]:          country          POP         tcgdp
         0      Argentina    37335.653  2.950722e+05
         1      Australia    19053.186  5.418047e+05
         2          India  1006300.297  1.728144e+06
         3         Israel     6114.570  1.292539e+05
         4         Malawi    11801.505  5.026222e+03
         5   South Africa    45064.098  2.272424e+05
         6  United States   282171.957  9.898700e+06
         7        Uruguay     3219.793  2.525596e+04
```

Here the index 0, 1,..., 7 is redundant, because we can use the country names as an index

To do this, we set the index to be the country variable in the dataframe

```
In [72]: df = df.set_index('country')
         df
```

```
Out[72]:                        POP         tcgdp
         country
         Argentina        37335.653  2.950722e+05
         Australia        19053.186  5.418047e+05
         India          1006300.297  1.728144e+06
         Israel            6114.570  1.292539e+05
         Malawi           11801.505  5.026222e+03
         South Africa     45064.098  2.272424e+05
         United States   282171.957  9.898700e+06
         Uruguay           3219.793  2.525596e+04
```

```
In [73]: df.columns = 'population', 'total GDP'
         df
```

```
Out[73]:                 population     total GDP
         country
```

```
Argentina        37335.653  2.950722e+05
Australia        19053.186  5.418047e+05
India          1006300.297  1.728144e+06
Israel            6114.570  1.292539e+05
Malawi           11801.505  5.026222e+03
South Africa     45064.098  2.272424e+05
United States   282171.957  9.898700e+06
Uruguay           3219.793  2.525596e+04
```

Population is in thousands, let's revert to single units

```
In [74]: df['population'] = df['population'] * 1e3
         df
```

```
Out[74]:                  population     total GDP
         country
         Argentina      3.733565e+07  2.950722e+05
         Australia      1.905319e+07  5.418047e+05
         India          1.006300e+09  1.728144e+06
         Israel         6.114570e+06  1.292539e+05
         Malawi         1.180150e+07  5.026222e+03
         South Africa   4.506410e+07  2.272424e+05
         United States  2.821720e+08  9.898700e+06
         Uruguay        3.219793e+06  2.525596e+04
```

Next we're going to add a column showing real GDP per capita, multiplying by 1,000,000 as we go because total GDP is in millions

```
In [75]: df['GDP percap'] = df['total GDP'] * 1e6 / df['population']
         df
```

```
Out[75]:                  population     total GDP    GDP percap
         country
         Argentina      3.733565e+07  2.950722e+05   7903.229085
         Australia      1.905319e+07  5.418047e+05  28436.433261
         India          1.006300e+09  1.728144e+06   1717.324719
         Israel         6.114570e+06  1.292539e+05  21138.672749
         Malawi         1.180150e+07  5.026222e+03    425.896679
         South Africa   4.506410e+07  2.272424e+05   5042.647686
         United States  2.821720e+08  9.898700e+06  35080.381854
         Uruguay        3.219793e+06  2.525596e+04   7843.970620
```

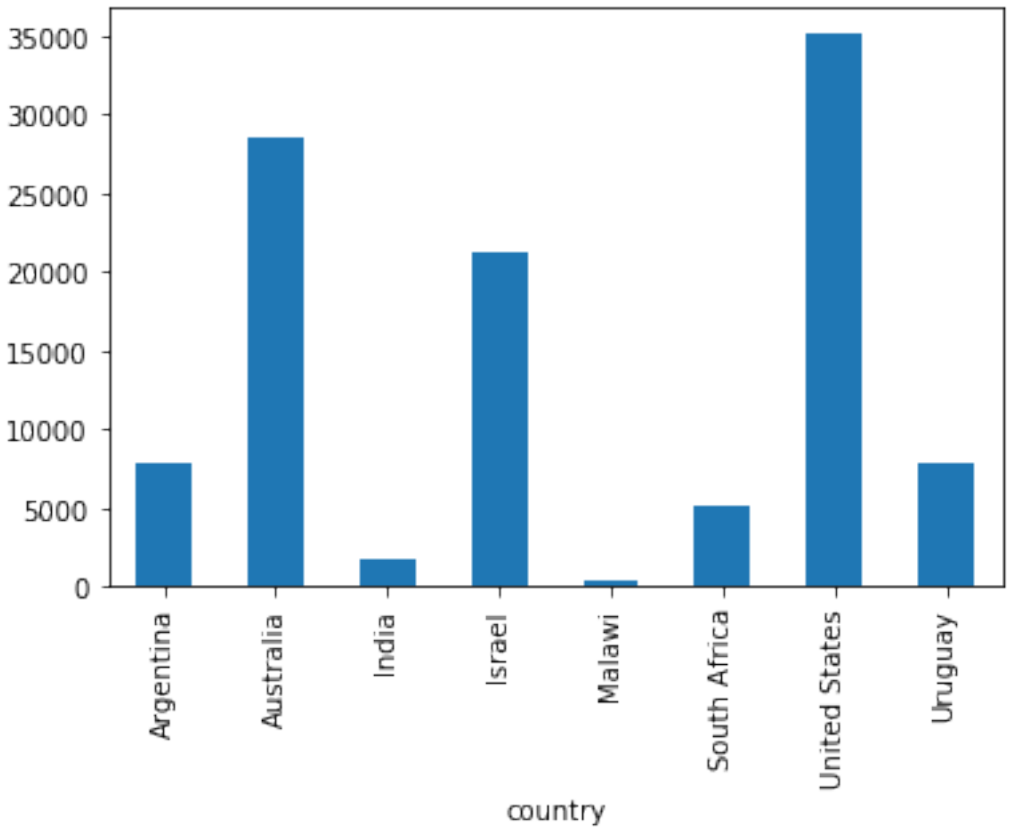
One of the nice things about pandas DataFrame and Series objects is that they have methods for plotting and visualization that work through Matplotlib

```
In [77]: df['GDP percap'].plot(kind='bar')
```

```
Out[77]: <matplotlib.axes._subplots.AxesSubplot at 0x7f230ff42650>
```

```
In [78]: import matplotlib.pyplot as plt

         plt.show()
```



3 References

- 10 Minutes to pandas - (https://pandas.pydata.org/pandas-docs/stable/10min.html)[https://pandas.pydata.org/pandas-docs/stable/10min.html]
- Pandas Tutorial: DataFrames in Python https://www.datacamp.com/community/tutorials/pandas-tutorial-dataframe-python
- https://www.tutorialspoint.com/python/python_files_io.htm

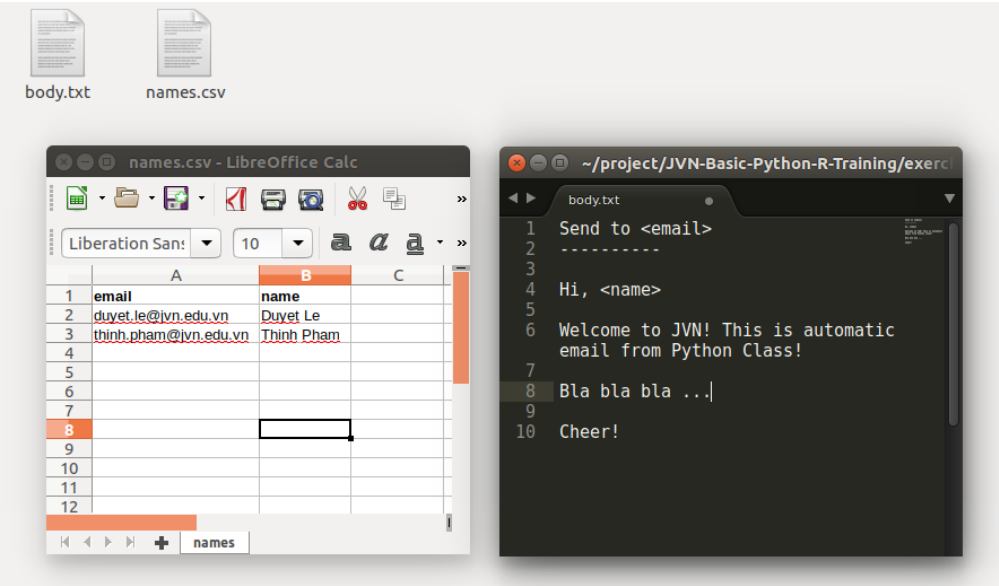
4 Excercises

Python Program to Merge Mails

When we want to send the same invitations to many people, the body of the mail does not change. Only the name (and maybe address) needs to be changed.

Mail merge is a process of doing this. Instead of writing each mail separately, we have a template for body of the mail and a list of names that we merge together to form all the mails.

For this program, we have written all the names and emails in separate lines in the file "names.csv". The body is in the "body.txt" file.



figs/session_4/merge_mails.png

We open both the files in reading mode and iterate over each name. A new file with the name "[name].txt" is created, where name is the name of that person. Replace the and in body email.

```
In [ ]:
```