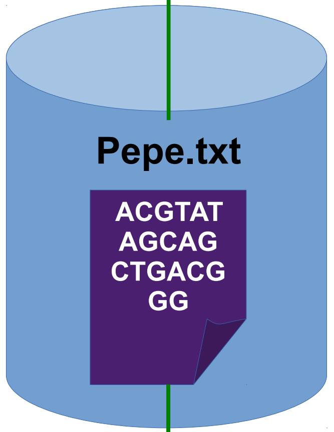
Reading files: all at once



File name (and path)

1 Connects with a file on the disk

```
MyFile = open ( 'pepe.txt' , 'r' )
```

name of a File-object (python object that handles all interactions between program and file ondisk)

Function that creates the file object

Mode: 'r' stands for 'read'

2 Copy data from the file into a variable in the program

```
MyData = MyFile . readlines()
```

Variable (type array) to where file data will be copied. Each element in the array is a text line.

Method associated to file-objects that reads data from file.

3 Close connection

```
MyFile . close()
```

Method associated to file-objects that close a connection to a file.

Reading files

Open the file "aa_frequencies_v1.txt" using a text editor (e.g. gedit)and also using spreadsheet software (e.g. MS Excel or Libreoffice calc).
Then run the program Unit3_OI_example0.py

```
['Ala\t 8.25  \n', 'Arg\t 5.53 \n', 'Asn\t 4.06  \n',
'Asp\t 5.45  \n', 'Cys\t 1.37  \n', 'Gln\t 3.93  \n',
'Glu\t 6.75  \n', 'Gly\t 7.07  \n', 'His\t 2.27  \n',
'Ile\t 5.96  \n', 'Leu\t 9.66  \n', 'Lys\t 5.84  \n',
'Met\t 2.42  \n', 'Phe\t 3.86  \n', 'Pro\t 4.70  \n',
'Ser\t 6.56  \n', 'Thr\t 5.34  \n', 'Trp\t 1.08  \n',
'Tyr\t 2.92  \n', 'Val\t 6.87  \n','\n']
```

Each of the lines in the file MyFile is stored as an element of the MyData array

The escape sequence "\n" means "new line"
Note that all lines end with it

There's a tab (represented by the escape sequence "\t") between aa name and frequency in all the lines. This is a character-delimited text file that uses tab to delimit columns.

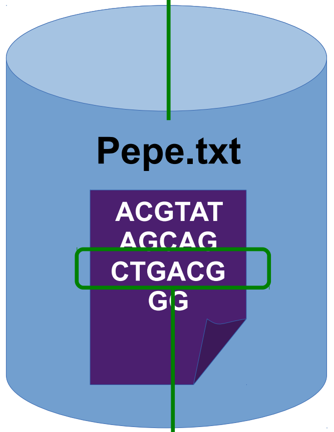
Open the file "aa_frequencies_v2.txt" using a text editor (e.g. gedit)and also using spreadsheet software (e.g. MS Excel or Libreoffice calc).
Then open the program Unit3_OI_example0.py comment line 5, uncomment line 6 and run it

```
['Ala,8.25\n', 'Arg,5.53\n', 'Asn,4.06\n', 'Asp,5.45\n',
'Cys,1.37\n', 'Gln,3.93\n', 'Glu,6.75\n', 'Gly,7.07\n',
'His,2.27\n', 'Ile,5.96\n', 'Leu,9.66\n', 'Lys,5.84\n',
'Met,2.42\n', 'Phe,3.86\n', 'Pro,4.7\n', 'Ser,6.56\n',
'Thr,5.34\n', 'Trp,1.08\n', 'Tyr,2.92\n', 'Val,6.87\n','\n','\n']
```

This is a character-delimited text file that uses comma (",") to delimit columns.

Note that, in this particular file, there's an empty line at the end.

Reading files: One line at a time v1



File name (and path)

1 Connects with a file on the disk

```
MyFile = open ( 'pepe.txt' , 'r' )
```

- MyFile: name of a File-object (python object that handles all interactions between program and file ondisk)
- open: Function that creates the file object
- 'r': Mode: 'r' stands for 'read'

2 Copy data from the file into a variable in the program one line at a time

While not(*end of file*):

```
Line = MyFile . readline()
```

- Line: Variable (type string) that stores the content of the current line
- readline(): Method associated to file-objects that reads data from file.

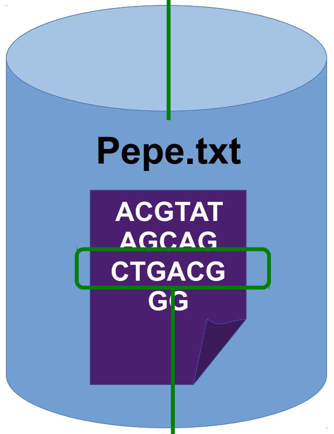
3 Close connection

```
MyFile . close()
```

- close(): Method associated to file-objects that close a connection to a file.

Run the program Unit3_OI_example1.py

Reading files: One line at a time v2



File name (and path)

1 Connects with a file on the disk

```
MyFile = open('pepe.txt','r')
```

name of a File-object (python object that handles all interactions between program and file ondisk)

Function that creates the file object

Mode: 'r' stands for 'read'

2 Copy data from the file into a variable in the program one line at a time

```
for Line in MyFile:
```

MyFile acts as a "collection" of lines so "for type" can go through all the elements (lines) in it

Variable (type string) that stores the content of the current line

3 Close connection

```
MyFile . close()
```

Method associated to file-objects that close a connection to a file.

Run the program Unit3_OI_example2.py

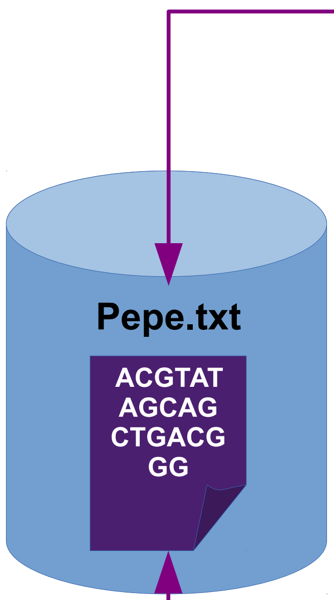
Parsing data

Two common manipulations:

1) Removing end-of line chacters with the .strip() method:
   a) Run again the script “Unit3_OI_example1.py”
   b) Edit the file to add the strip() method (line 10 of code): Line=MyFile.readline().strip()
   c) Save it as “Unit3_OI_example1_v2.py” and run it.
   d) Compare output of both versions
2) Dividing input line into smaller pieces of information (character-delimited files) with the .split() method
   a) Open “Unit3_OI_example2_v2.py” in a text editor study the code
   b) Run “Unit3_OI_example2_v2.py” and observe the output, compare it with the output produced by “Unit3_OI_example2.py”
   c) Edit “Unit3_OI_example2_v2.py” so it reads the file aa_frequencies_v2.csv and prints only the second column. Save it as “Unit3_OI_example2_v3.py” and run it.

Beaware of two common types of data files:
1. Table-like
2. “keyword” based

Writing data to a file



1 Connects with a file on the disk

```
MyFile = open('pepe.txt','w')
```

- File name (and path)
- name of a File-object (python object that handles all interactions between program and file ondisk)
- Function that creates the file object
- Mode: 'w' stands for 'write'

2 Copy data from a variable into a file

```
MyFile.write("text to write")
```

- Method associated to file-objects that writes data to file.

3 Close connection

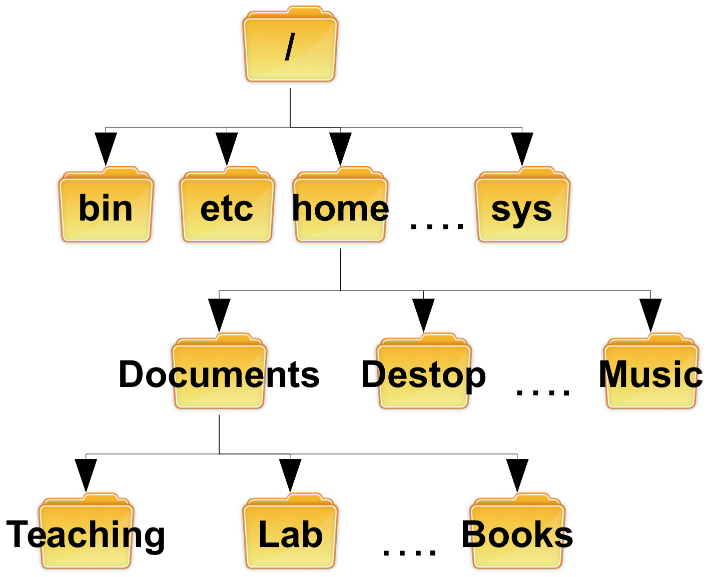
```
MyFile.close()
```

- Method associated to file-objects that close a connection to a file.

Run the program Unit3_OI_example3.py

Handling the filesystem

Absolute/Relative paths



Current dir: .
Parent dir: ..

Absolute path to Lab: /home/Documents/Lab
Relative from home to Lab: ./Documents/Lab
Relative from Lab to Documents: ../
Relative from Lab to home: ../../
Relative from Lab to Destop: ../../Destop

Bash/DOS Shell (Terminal)

```
$ pwd (1)
$ ls (2)
```

```
$ cd (1)
$ dir (2)
```

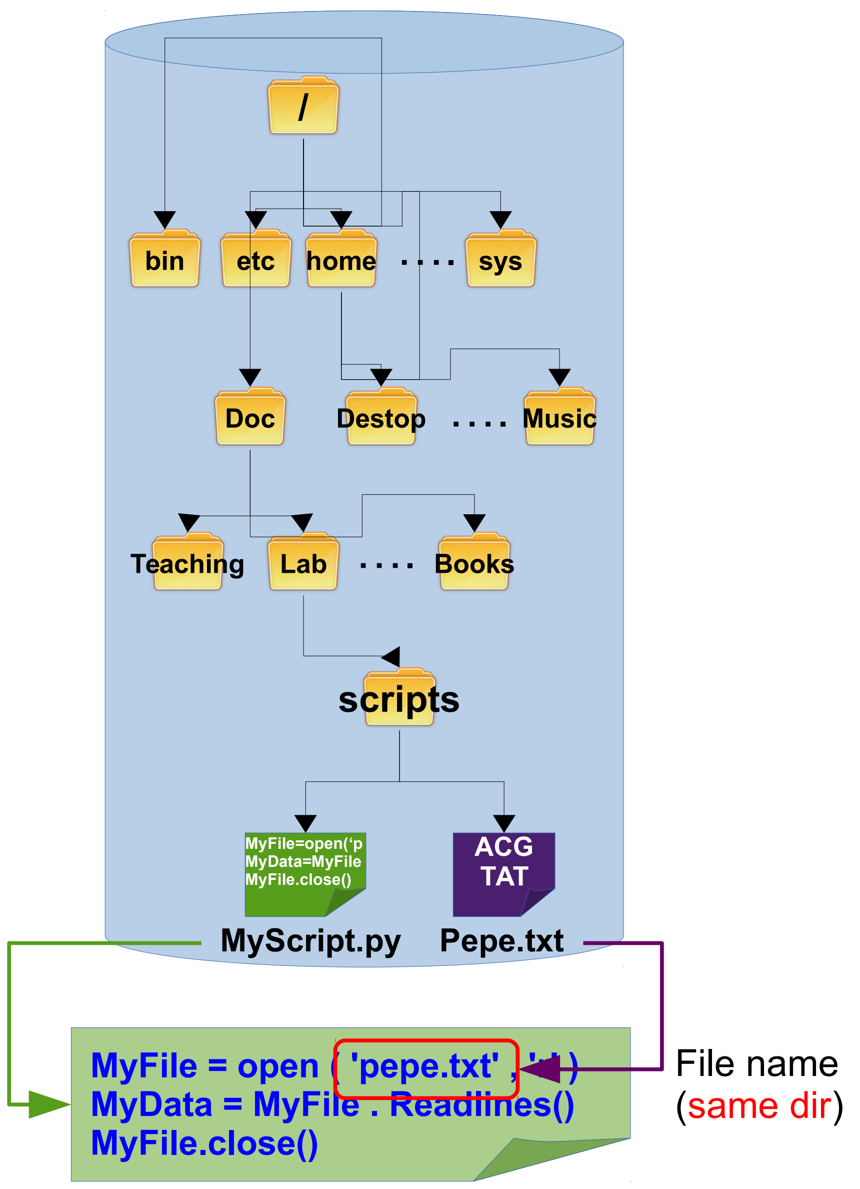
(1) Shows current directory
(2) Shows current directory

Do not worry about this now, will see it in the next unit

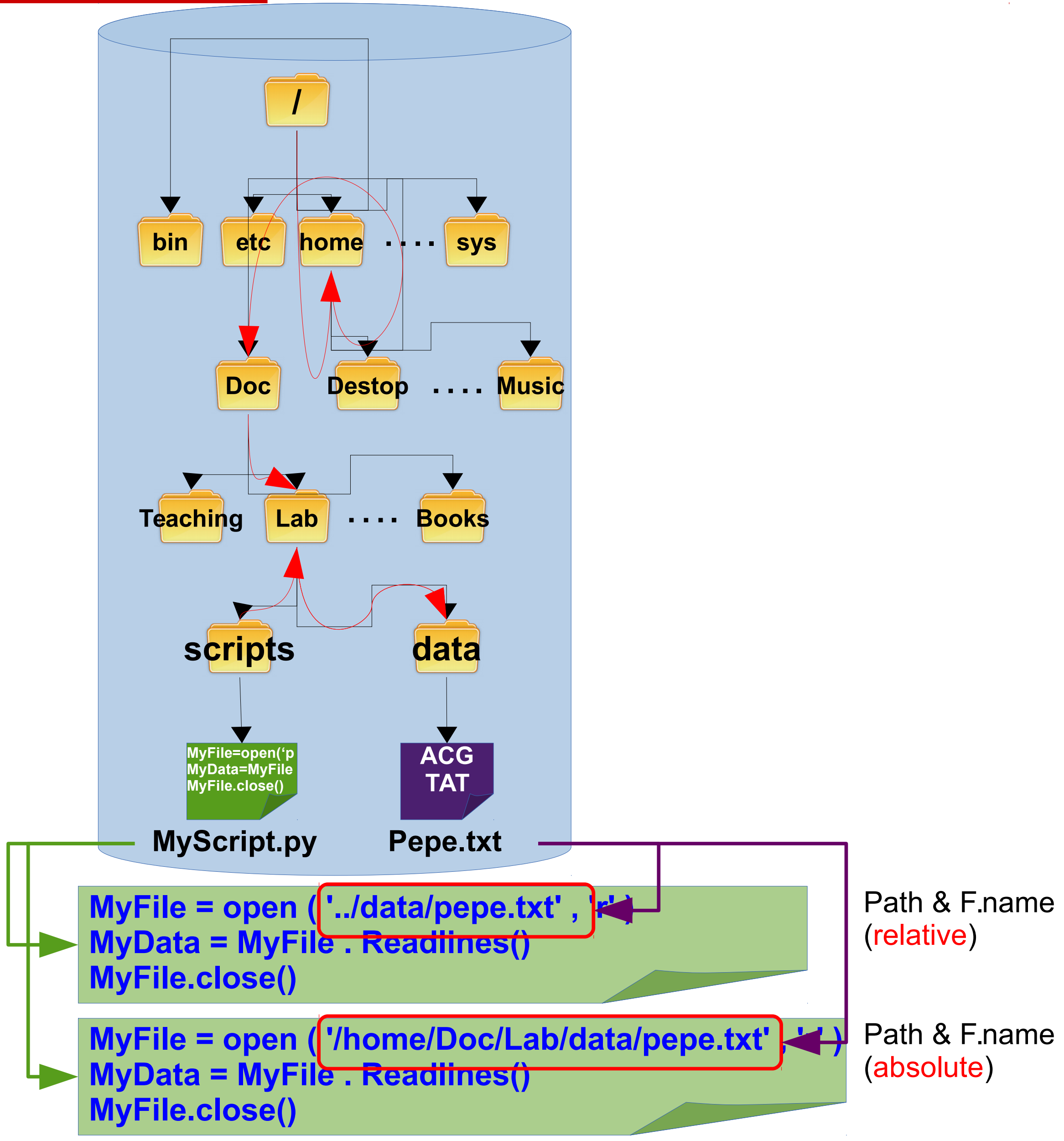
Python interactive shell

```
>>> import os
>>> os.getcwd() (1)
'/home/luis'
>>> os.listdir() (2)
Lots of stuff here
>>> os.chdir('/home/luis/Documents')
>>> os.getcwd()
'/home/luis/Documents'
>>> os.chdir('../')
>>> os.getcwd()
'/home/luis'
>>> os.chdir('./Desktop')
['sq.24737.aln', 'tmp']
>>> os.path.exists('./tmp')
True
>>> os.path.exists('./pepe.py')
False
```

Reading files: path to file



```
MyFile = open ( 'pepe.txt' , 'r' )
MyData = MyFile . Readlines()
MyFile.close()
```



```
MyFile = open ( '../data/pepe.txt' , 'r' )
MyData = MyFile . Readlines()
MyFile.close()
```

```
MyFile = open ( '/home/Doc/Lab/data/pepe.txt' , 'r' )
MyData = MyFile . Readlines()
MyFile.close()
```

Redirecting output

Bash/DOS Shell (Terminal)

```
$ python3 myScript_name.py
```

Output of print to stdout (screen)

```
$ python3 myScript_name.py > myfile
```

Output of print to file (creates/overwrites myfile)

```
$ python3 myScript_name.py >> myfile
```

Output of print to file (appends to myfile)

Shell (linux) redirect operator

```
$ python3 Unit3_OI_example2_v2.py
Ala
Arg
Asn
...
Val
$ python3 Unit3_OI_example2_v2.py > res_file
$
```

Redirecting input

Bash/DOS Shell (Terminal)

```
$ python3 Unit3_OI_explample5.py
Type here any text you want.
The text can expand several lines.
Type Crt+D when you are done

You gave me:
Type here any text you want.
The text can expand several lines.
Type Crt+D when you are done
```

This is what you type

python output

Unit3_OI_explample5.py

```
import sys

UserInput=sys.stdin.read()

print("You gave me:\n",UserInput)
```

Do not worry about this now, will see it in the next unit

Tells python to get data from the standard input (by default keyboard)

```
$ python3 Unit3_OI_explample5.py < Unit3_OI_explample5.py
You gave me:
#! /usr/bin/python3
# HPBBM, Unit 3, example 5: redirect input
# reads data from the standard input (stdin)
# data can be feed by keyboard: python3 Unit3_OI_explample5.py
# or redirected from a file: python3 Unit3_OI_explample5.py < anyfile
import sys
UserInput=sys.stdin.read()
print("You gave me:\n",UserInput)
```

Getting user input from command line arguments (sys.argv)

In the last week script we had to read a sequence from a file. The name of the file was “hard-coded” in the script so we'd have to edit the script if we wanted to read a different sequence. To avoid that we could have written the program to accept the name of the file as a command line argument.

Command line arguments: values that are passed to the program when it is firts launched

Note that we've been already passing arguments to programs. For example:

```
$ cd /home/luis/Documents
```

program argument

There're several ways to get access to the command line arguments from within the script. The simpler way is to use the ***sys.argv*** variable which contains a list of all the arguments.

In order to use sys.argv variable we need to import the ***sys*** module first.
A module is a small program that contains definitions and statements.
Importing a module can be thought of as importing a function that was not accessible before.

Imports the module *sys* so it can be used within the script

TestArgs.py

```
import sys
for argumento in sys.argv:
    print(argumento)
```

List of all the arguments passed to the script from the command line

Run TestArgs.py providing several arguments. For example:

```
$ python3 TestArgs.py 1 2 pepe
$ python3 TestArgs.py pepe y luis
$ python3 TestArgs.py “pepe y luis”
```

What is the value of sys.argv[0]?
What is the type of each of the elemnts in sys.argv?