Jurre Hageman
Hanze *University of Applied Sciences*
Groningen

Bioinformatics data processing and –analysis using the scripting language Python

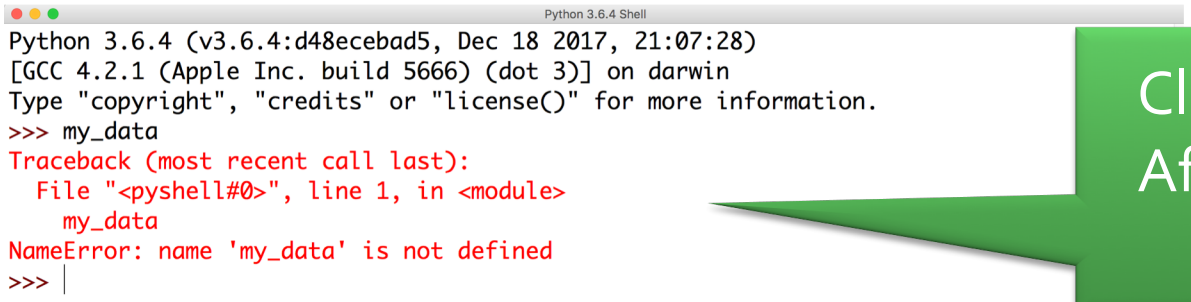
Introduction to programming with Python

Input and Output

Data

Often, you want to read data in your program. Your data will not be present in lists, tuples and dictionaries. These data structures only live in **memory**. Not in a **persistent** state.

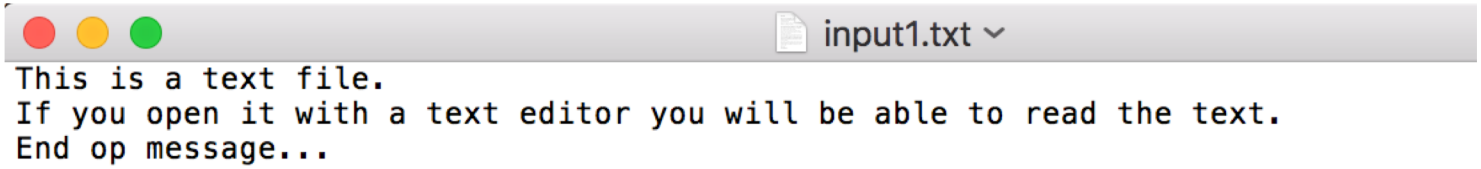
```
Python 3.6.4 (v3.6.4:d48ecebad5, Dec 18 2017, 21:07:28)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> my_data = "This data is not persistant and only lives in memory".split()
>>> my_data
['This', 'data', 'is', 'not', 'persistant', 'and', 'only', 'lives', 'in', 'memory']
>>>
```



Closed the IDLE session. After opening again, the data is lost.

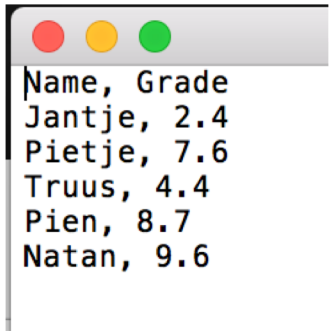
Text Files

- Python can read many file types.
- The most basic file type is the text file or ASCII file.
- This is a file that you can open with a text-editor and yields readable text:

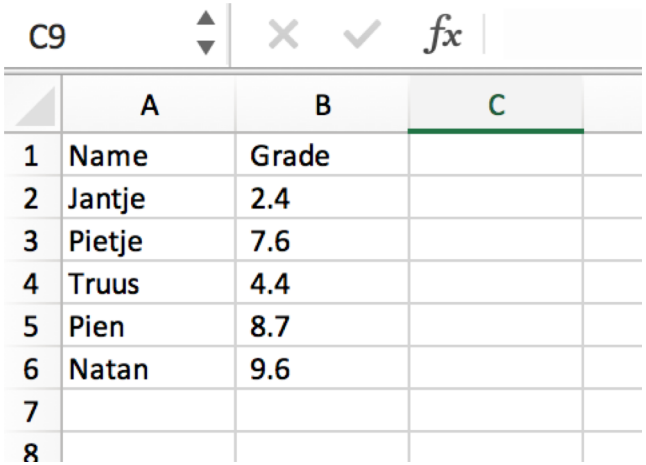


CSV Files

- Often, researchers use text files to store data and store it in a table. If comma's are used, such a file is a comma-separated text file or csv file. Instead of comma's, other column separators can be used.



Note that the comma acts as column separator. You can also open the file in Excel.



Open a file in Python

- Python can generate a file object with the "open" function.
- You can use the file name as an argument. Put your file in the same folder as your Python script.

script1.py - /Users/jurrehageman/OneDrive - Hanzehogeschool Groningen/2018-2019/kwartaal_1/Informatica_1/informatica1/HC/HC0

```
file_name = "input1.txt" #note that this is just a string.
file_object = open(file_name) #now a file object will be created.
#The file is not read yet.
print(file_object)


 RESTART: /Users/jurrehageman/OneDrive - Hanz      geschool Groningen/2018-2
_1/Informatica_1/informatica1/HC/HC03/scri      1.py
<_io.TextIOWrapper name='input1.txt' mode='r' encoding='US-ASCII'>
>>>
```

Note that the file content is not yet opened. Only a file object is constructed

Open a file in Python

- The file object is an itterable. This means that you can loop through it:

line is the placeholder in the for loop. You can name it whatever you want.

```
>>> for line in file_object:
        print(line)


This is a text file.

If you open it with a text editor you will be able to read the text.

End op message...
```

Note that there is a newline after each line of text. This because the text file has newlines after each line

Open a file in Python

- Now the new lines are stripped off.

script1.py - /Users/jurrehageman/OneDrive - Hanzehogeschool Groningen/2018-2019/kwartaal_1/Informatica_1/informatica1/HC

```
file_name = "input1.txt" #note that this is just a string.
file_object = open(file_name) #now a file object will be created
#The file is not read yet.
print(file_object)

for line in file_object:
    line_cleaned = line.strip() #new lines stripped off
    print(line_cleaned)
```

```
<_io.TextIOWrapper name='input1.txt' mode='r' encoding='US-ASCII'>
This is a text file.
If you open it with a text editor you will be able to read the text.
End op message...
>>>
```

Open a file in Python

- A file object is itterable, but it is itterable only once. You can not go back. You need to create a new object to iterate through the lines again:

```
>>> file_object = open(file_name)
>>> for line in file_object:
        print(line.strip())


This is a text file.
If you open it with a text editor you will be able to read the text.
End op message...
>>> for line in file_object:
        print(line.strip())


>>>
```

Note that the file object is empty now. You need to create a new file object to iterate through it again.

File modi in Python

- There are 3 modi a file object can be in:
  - Read "r"
  - Write "w"
  - Append "a"
- Read is default so you do not have to explicitly define it:

```
>>> file_object = open(file_name, "r")
>>> for line in file_object:
        print(line.strip())


This is a text file.
If you open it with a text editor you will be able to read the text.
End op message...
>>>
```

File object in read modus. You can omit this argument for reading files.

Write File in Python

- You can write data to a file by generating a file object in write modus.
- The print function can write data using
- Close the file object. This is especially important when writing data!

```
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.

>>>
```

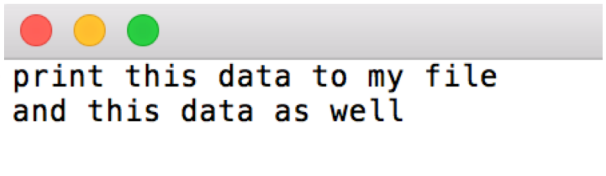
Print can write data to a file.

Write File in Python

- You can write data to a file by generating a file object in write modus.
- The print function can write data using
- Close the file object. This is especially important when writing data!

```
>>> my_file = "output.txt"
>>> my_file_obj = open(my_file, "w")
>>> print(my_file_obj)
<_io.TextIOWrapper name='output.txt' mode='w' encoding='US-ASCII'>
>>> print("print this data to my file", file=my_file_obj)
>>> print("and this data as well", file=my_file_obj)
>>> my_file_obj.close()
```

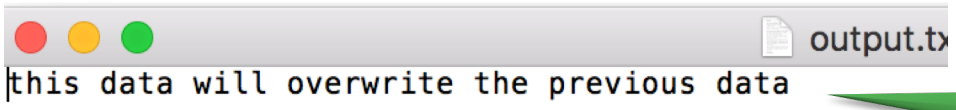
Yes, it worked!



Append File in Python

- Write mode will create a new file and overwrite your data:

```
>>> my_new_file = "output.txt"
>>> my_file_obj = open(my_new_file, "w")
>>> print("this data will overwrite the previous data", file=my_file_obj)
>>> my_file_obj.close()
>>>
```

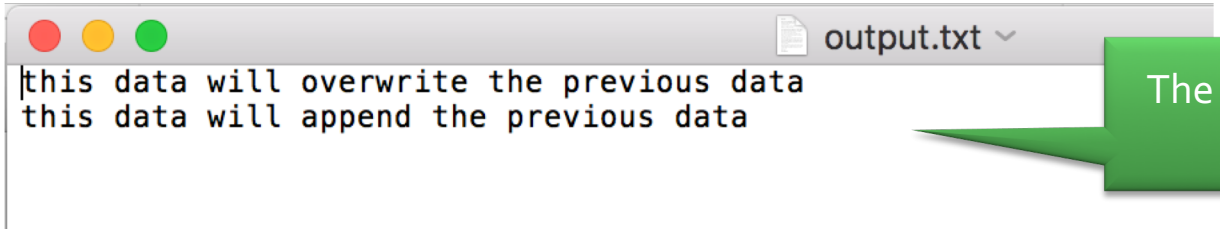


The data is overwritten!

Append File in Python

- Use the "a" argument to append to the file:

```
>>> my_new_file = "output.txt"
>>> my_file_obj = open(my_new_file, "a")
>>> print("this data will append the previous data", file=my_file_obj)
>>> my_file_obj.close()
```



The data is appended to the previous data!

A script to calculate GC content

- A script that calculates the GC content
- Reports the highest GC percentage
- Reports the sequence with the highest GC percentage
- Writes data to file

A script to calculate GC content

highest_GC.py - /Users/jurrehageman/OneDrive - Hanzehogeschool Groningen/2018-2019/kwartaal_1/Informatica_1/informatica1/HC/HC03/highes

```
file_name = "input_sequence.txt"
file_name_output = "output_gc_content.txt"
file_obj = open(file_name)
GC_max = 0 #set variable GC max to 0
highest_GC_sequence = "" #this is set to an empty string

for line in file_obj:
    line = line.strip()
    G = line.count("G")
    C = line.count("C")
    GC = G + C
    perc_GC = GC / len(line) * 100
    perc_GC = round(perc_GC)
    print("processing", line, ", GC percentage:", perc_GC)
    if perc_GC > GC_max: #if the percentage is the highest
        GC_max = perc_GC #overwrite the highscore
        highest_GC_sequence = line #overwrite the sequence with the highest GC
        print("New record, highest GC sequence is:", highest_GC_sequence)
        print("New record, highest GC percentage:", GC_max)
file_obj.close() #close the file
print() #print empty line
print("The sequence with the highest GC value is:", highest_GC_sequence)
print("The Highest GC percentage is:", GC_max)

file_obj_out = open(file_name_output, "w")
print("The sequence with the highest GC value is:", highest_GC_sequence, file=file_obj_out)
print("The Highest GC percentage is:", GC_max, file=file_obj_out)
file_obj_out.close() #do not forget to close the file object!
print("Data written to:", file_name_output)
```