Good morning!

Remaining sessions

- Session 9: Abstract data types
- Session 10: Summary and split projects
- Session 11: Project 1/2
- Session 12: Project 2/2

Agenda

- Recap on files
- Recap on docstrings
- Abstract data types
- Assignment 9

Working with files

- Files are simply just blocks of contents on your harddrive
- They can be **read**, **written to**, **created** and **deleted**

- In Python you can do all of this!

In [2]:

```
important_file = 'some_file.txt'
```

- Another problem is that I'm not doing anything with that pointer. It's not stored in a variable

`open` argument flags

Open can either

- **create** or **write to** a file

```
open(important_file, 'w')
```

- **read** from a file

```
open(important_file, 'r')
```

- **append** to a file

```
open(important_file, 'a')
```

In [7]:

```
with open(important_file, 'w') as file:
    pass
```

- Write that into your code editor

Reading from a file

- Now that we have a **pointer** to the beginning of the file, we can read all the contents

In [ ]:

```
with open(important_file, 'r') as file:
    file.read()
```

Function	Effect
`read()`	Reads **all** the file content into **one** string
`readlines()`	Reads **all** the file content into a **list** of lines

In [19]:

```
with open(important_file, 'a') as file_pointer:
    file_pointer.write('Moomin My')
```

In [25]:

```
with open(important_file, 'r') as file_pointer:
    print(file_pointer.read())
```

```
Moomin MyMoomin MyMoomin MyMoomin MyMoomin My
Moomin My
Moomin My
Moomin My
```

In [24]:

```
with open(important_file, 'a') as file_pointer:
    print('Moomin My', file = file_pointer)
```

In [26]:

```
with open(important_file, 'r') as file_pointer:
    print(file_pointer.read())
```

```
Moomin MyMoomin MyMoomin MyMoomin MyMoomin My
Moomin My
Moomin My
Moomin My
```

Files in summary

- Files can be **read** (`r`), **written to/created** (`w`) or **appended to** (`a`)
- Use the `with open(..) as` syntax
  - It saves changes automatically

In [ ]:

```
# Windows!!!
with open(filename, encoding='utf-8') as file:
    all_the_lines = file.readlines()
```

Your turn!

- Go to GitHub and download the file `the_hound_of_the_baskervilles.txt` from session 9
- Open it with a Python script (using the `with` notation)
- Print the very last line of the story

Remembering state and solving problems

The line `open(important_file, 'r')` only makes sense when you keep the value of `important_file` in your head!

- We're practicing running code line by line
  - Goal: understand what happens in the program
- Make sure you understand that 100% before you solve your problems
  - What is in my variable?!
  - What variables do I have?

In [ ]:

```
1
```

How to use docstrings

- Docstrings are documentation that the future you or future users can read

In [ ]:

```
def is_even(number):
    """Examines whether a positive number is even. False if negative
    :param number: int
        The number to examine
    bool :
        True if the number is even and above 0, False otherwise
    """
    if number <= 0:
        return False
    else:
        return number % 2 == 0
```

Your turn:

Clean up and document this function:

In [27]:

```
def text_with_full_name(first_name, last_name):
    """Puts the first and last name into a string
    ---
    Parameters
    first_name : as input
    last_name : as input

    ---
    Output
    String containing first name and last name as given
    """
    return "Firstname (" + first_name + ") and lastname (" + last_name + ")"
text_with_full_name('Jane', 'Doe')
```

Out[27]:

```
'Firstname (Jane) and lastname (Doe)'
```

Global and local variables

Global variables.

- Are declared outside of functions and objects.
- Can be used inside and outside of functions.

In [28]:

```
def deep_thought():
    """Ask my enything
    """
    print(your_question)  # your_question is a global variable. Everybody knc
    print('The answer to your question is:')
    print(42)


# your_question is a global variable
your_question = 'what is the meaning of life the universe and everything?'
deep_thought()
```

```
what is the meaning of life the universe and everything?
The answer to your question is:
42
```

Local variabels:

- Are declared inside of functinos.
- Can ONLY be used inside the function where it is declared

In [32]:

```
def deep_thought(the_question):
    """Ask my anything
    :param your_question: string
        A string containing your question
    """
    the_answer = 42          # This is a local variable. Only Deep thought kno
    print(the_question)      # the_question is a parameter, which acts as a lo
    print('The answer to your question is:')
    print(the_answer)


your_question = 'what is the meaning of life the universe and everything?'
deep_thought(your_question)

```

```
what is the meaning of life the universe and everything?
The answer to your question is:
42
```

When to print and when to return.

- print will ONLY print something, afterwards it will be forgotten forever.

In [33]:

```
def greetings(name):
    print('Hi ' + name)


greetings('Viktor')
```

```
Hi Viktor
```

When to print and when to return.

- return will return something, which you then can store and use forever and ever.

In [42]:

```python
def remove_strings(data_list):
    """This function removes strings from a list
    :param data_list: list
        A list containing strings and other types.
    """
    new_list = []
    for element in data_list:
        if type(element) != str:
            new_list.append(element)
    return new_list


some_list = ['This', 1, 'is', 3, 'an', 12, 'odd', 536, 'list.']

new_list = remove_strings(some_list)  # The new list is now stored as a varia
print('This is the cleaned new list:', new_list)  # Therefore we can print it
```

```
This is the cleaned new list: [1, 3, 12, 536]
```

In [43]:

```python
# We can also perform new calculations on this varible, e.g., `sum`
print('This new list have a sum of:', sum(new_list))
```

```
This new list have a sum of: 552
```

List `sorted` vs. `sort`

What do you expect is printed by the following program?

In [44]:

```python
my_data = [9, 7, 5, 1, 8, 2]

sorted_list = sorted(my_data)
print(sorted_list)
print(my_data)
```

```
[1, 2, 5, 7, 8, 9]
[9, 7, 5, 1, 8, 2]
```

What do you expect is printed by this one?

In [50]:

```
my_data = [9, 7, 5, 1, 8, 2]

copy = my_data[:]
sorted_list = my_data.sort()
print(copy)
print(my_data)
```

```
[9, 7, 5, 1, 8, 2]
[1, 2, 5, 7, 8, 9]
```

In [46]:

```
print(help(sorted))
```

```
Help on built-in function sorted in module builtins:

sorted(iterable, /, *, key=None, reverse=False)
    Return a new list containing all items from the iterable in asce
nding order.

    A custom key function can be supplied to customize the sort orde
r, and the
    reverse flag can be set to request the result in descending orde
r.

None
```

In [47]:

```
print(help([].sort))
```

```
Help on built-in function sort:

sort(*, key=None, reverse=False) method of builtins.list instance
    Stable sort *IN PLACE*.

None
```

Homework 7 solutions

Part A

> Two functions but they do the same. Read Excel file, take 'Sheet1', turn columns into a list, choose single column, slice column to exclude first entry, extract values from cell objects to list. Return list of values.

```
filename = 'country_codes.xlsx'
wb = openpyxl.load_workbook(filename)
sheet = wb.get_sheet_by_name("Sheet1")

list_of_columns = list(sheet.columns)
column_of_interest = list_of_columns[0]
cells_of_interest = column_of_interest[1:]

country_codes = []

for cell_object in cells_of_interest:
    country_codes.append(cell_object.value)

return country_codes
```

Part B

```
wb = openpyxl.load_workbook(filename)
sheet = wb["Sheet1"]

list_of_columns = list(sheet.columns)
column_of_interest = list_of_columns[column_number]
cells_of_interest = column_of_interest[1:]
```

Part C

Modify the function `country_codes_data(filename, column_number)` from the earlier `country_codes_stats.py` so that it returns a **sorted** list of **unique** country codes. That is, it does not contain duplicates!

In [ ]:

```
...
country_codes = set(country_codes)
country_codes = sorted(country_codes)

return country_codes
```

Part E

Implement the function `translate_code_to_text` in your `country_codes_stats.py` module. The goal is to allow you to translate a numerical country code into an explanatory string. For example, the following code should print `'Belgien'`.

```
descriptive_string = translate_code_to_text(5126)
print(descriptive_string)
```

In [ ]:

```
def translate_code_to_text(code):
    """Finds the country name for a given country code.

    :param code: str or int
        The country code
    :return: str
        The country name
    """
    return stat_codes.COUNTRY_CODES[str(code)]
```