Python for beginners

A MARUM course

Welcome

- Introduction of trainers
- Introduction of participants

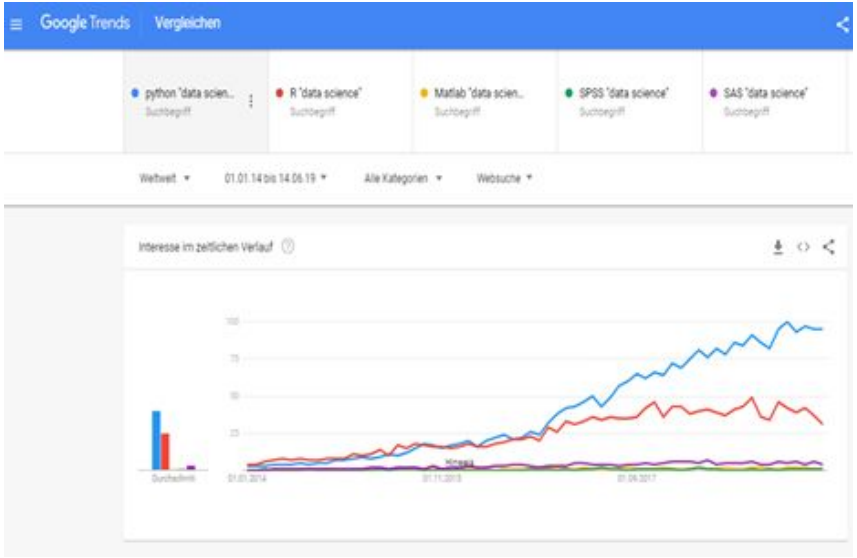
Introduction

What is Python:

- a popular programming language, created by Guido van Rossum, 1991.

Why Python:

- Beginner friendly
- Versatile and flexible
- Very popular + well supported
- **Great for Data Science**



Get started

Schedule:

- **Day 1**: Python synthax, variables, operators, conditioals
- **Day 2**: while-loops, lists, for-loops, dictionaries, tuples and sets
- **Day 3**: functions, classes and objects
- **Day 4**: data science: pandas, pangaeapy, plotting

Course material:

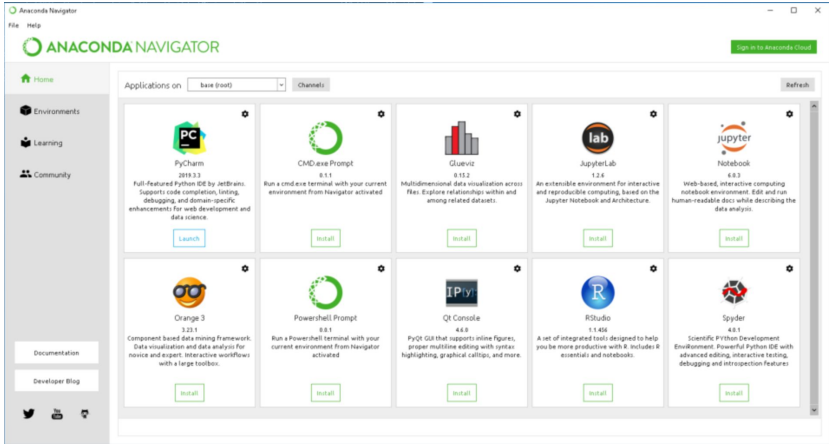
- https://drive.google.com/drive/folders/1Efi2jP1sFUbm3GAd3dwtWDQM_Hq11VQL
- https://github.com/huberrob/Python-for-Beginners-MARUM-

Get started

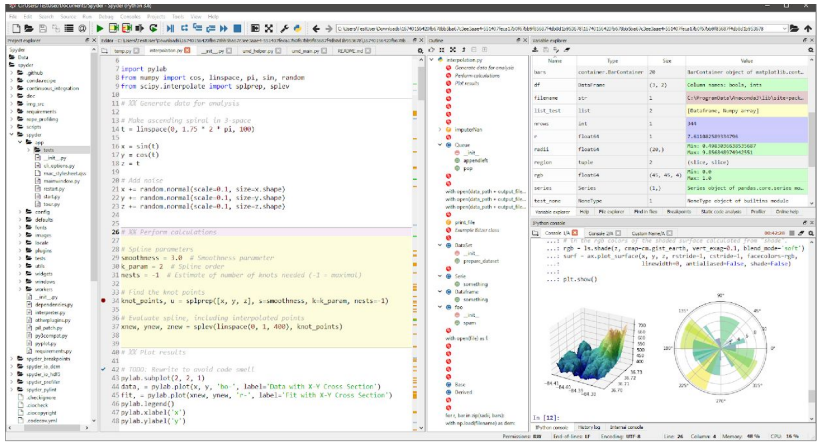
Option1:

Install Python (Anaconda) locally

- Easy to install
- https://docs.anaconda.com/anaconda/install/
- Installs Python and Tools (e.g. Editor)
- Contains most important Data Science libraries



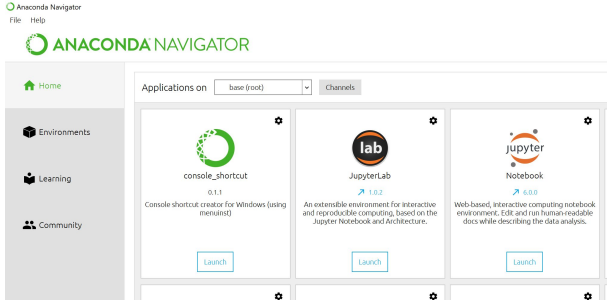
Start the Editor (e.g. Spyder) or Jupyter

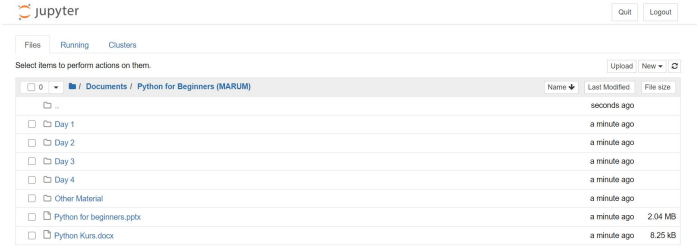


Start programming

Setting up your local environment

- Install Anaconda
- Download course material from
  - https://drive.google.com/drive/folders/1Efi2jP1sFUbm3GAd3dwtWDQM_Hq11VQL
- Unzip and copy material to ‘Documents’ folder
- Open ‘Anaconda Navigator’
- Launch jupyter Notebook
  -



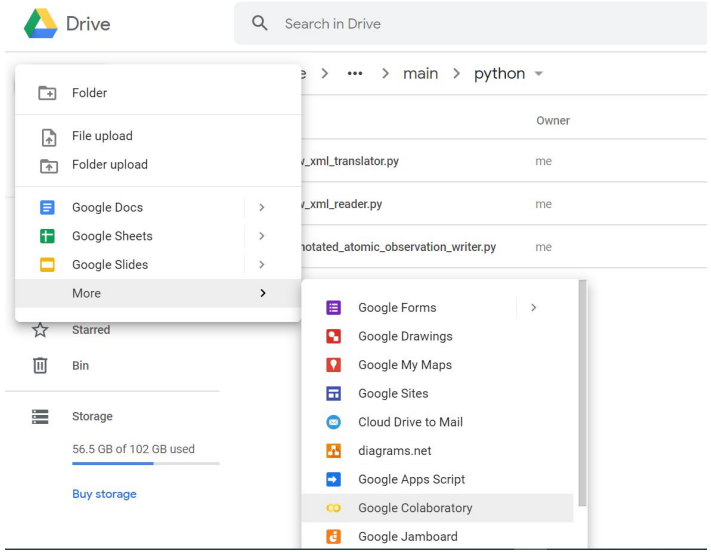


Get started

Option 2:

Use online (Jupyter) Notebooks

1. Open Google Colaboratory:
   - Create a Google account
   - Visit https://drive.google.com/
   - Open or create Notebook
2. Visit the Github page
   - https://github.com/huberrob/Python-for-Beginners-MARUM-
   - Click on the icon launch binder



Lets code!

Please Open

Python for Beginners (MARUM)\Day 1\Exercises\Lesson 1\Hello World.ipynb

Python syntax and structure

Expressions and Statements:

- Typically a line of code
- (we ignore the differences)
- Example: `print("Hello, World!")`

Identation:

- Blocks of code are denoted by **line indentation**, which is rigidly enforced.
- More about this during exercising..
- Example see ->

```
try:
   # open file stream
   file = open(file_name, "w")
except IOError:
   print "There was an error writing
to", file name
   sys.exit()
print "Enter '", file finish,
print "' When finished"
while file text != file finish:
   file text = raw_input("Enter
text: ")
   if file text == file_finish:
      # close the file
      file.close
      break
   file.write(file text)
   file.write("\n")
file.close()
```

Variables, types and values

Values and types:

a. You need data **value** to ‘feed’ a program e.g. a letter or a number: 1, 2, and 'Hello, World!'
b. Values have **types** such as number type (float, integer) or string type (character, string)

Variables and keywords

c. **Variables** store values
d. Variable names can defined by the programmer but:
    i. Have to begin with a letter
    ii. May not contain some special characters like !, @, #, $, %
    iii. Some **keywords** (e.g. if, True, in...) are reserved by the Python language

Operators and operands I

Operators are special symbols that represent computations like + - * /. The values the operator is applied to are called **operands**.

Arithmetic operators

+ addition; **-** subtraction; ***** multiplication; **/** division; ****** exponentiation; **%** modulo division; **//** floor division

Assignment operators

= simple assignment ; **+=** **-=** ***=** compound assignments

Assignments

- To assign a value to a variable use the **=** operator
  - Example:

    a = 1
- To change the value, repeat
  - A = 1
  - B = 2
  - A = 3 => Now variable A has the value 3
- To update you can also:
  - A = 1
  - A = A + 1
  - Which is useful in loops (later…)

Type casting and checking

Sometimes you need to change the type of a variable e.g. to print numbers together with text

You can set the type in python explicitly (type casting):

- **int(“10”)**
- **str(100)**

To check the type of a variable use:

type(0)

Lets code!

Please Open

Python for Beginners (MARUM)\Day 1\Exercises\Lesson 2\Variables and Operators.ipynb

Conditionals

Used to change program behaviour based on conditions

Comparison operator

- equal : **==** ; not equal: **!=**, less than **<**, more than: **>**; equal or less (more) **<= >=**

Condition

- **A == B**

Boolean values

- Can be **True** or **False**: e.g. **isWrong=False**

Result of a condition is always either True or False

Conditionals II

If statement:

- Do something if condition becomes true…
-
- Sythax: **if condition :**
- Must be followed by identation (code block)

```
a = 33
b = 200
if b > a:
  print("b is greater")
```

Logical Operators:

- Allow to combine conditions
- **And**, **or**, **not**

```
if b > a and a > 1:
  print("b is greater than a
and a > 1")
```

Conditionals III

If ... else:

- Do something if condition becomes true...
- Do something else if the condition is false
- Use elif for multipe conditions

```
a = 33
b = 200
if b > a:
  print("b is greater")
elif a == 1:
  print("b is greater")
else:
  print("b is smaller")
```

User input

Python has a built in function which allows to collect user input:

Synthax: **input(*prompt*)**

This prints a message (defined by the prompt variable) and an input prompt

Example: **input('Enter your name:')** will produce:

Enter your name: |

You can assign the input to another variable

name = input('name:')

Lets code!

Please Open

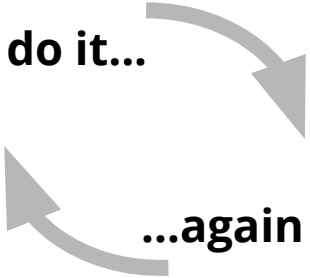
Python for Beginners (MARUM)\Day 1\Exercises\Lesson 3\Conditionals.ipynb

Day 2

Iteration

What it is good for:

- Repeats a statements e.g. in loops
- Are necessary to avoid writing code again and again



The while loop:

- Repeats a **statement** until a **condition** becomes true
-
- Sythax: **while condition :**
- Eg. `while b > a:`

Iteration - while loop

The while loop an example:

```
i = 1
while i < 6:
    print(i)
    i += 1
```

Initial assignment and iteration rule/condition

Identation of code block

Statements to repeat

Reads as: print the value of variable i while i is smaller than 6

Iteration - while loop

The break statement:

Stops execution within a loop even if while condition is true:

```
i = 1
while i < 6:
  if i == 2:
    break
  print(i)
  i += 1
```

Iteration - while loop

The continue statement:

Continues to next iteration (e.g. leave one out):

```
i = 1
while i < 6:
  if i == 2:
    continue
  print(i)
  i += 1
```

Iteration - while loop

The else statement:

Executed once the condition is no longer true:

```
i = 1
while i < 6:
  if i == 2:
  print(i)
  i += 1
else:
  print(“End of loop”)
```

Lets code!

Please Open

Python for Beginners (MARUM)\Day 2\Exercises\Lesson 1\Iteration_while.ipynb

Lists

- A list is a ordered, mutable, **sequence** of values
- It is a Python data type which holds this sequence
- Lists are created using square brackets []

Example:**mylist = [1,3,5,7,4]**

value	1	3	5	7	4
index	0	1	2	3	4

- Lists are mutable
- Each list value has an index which indicates its position
- Lists can contain any data type including other lists
- **mylist = []** creates an empty list
- The **len()** method gives you the number of entries

List access and manipulation

- You can access list entries by using its index (starting with 0)
  - Positive indices start at the beginning of the list
  - Negative indices start at the end of the list
- To get the first entry use e.g. mylist[0]:

```
print(mylist[0])
```

- You can also use the index to manipulate a list entry:

```
mylist[0]  = 2
```

List methods

`list.`**`append`**`(`*`x`*`)`
Add an item to the end of the list.

`list.`**`extend`**`(`*`iterable`*`)`
Extend the list by appending all the items from the e.g. another list

`list.`**`insert`**`(`*`i`*`, `*`x`*`)`
Insert an item at a given position.

`list.`**`remove`**`(`*`x`*`)`
Remove the first item from the list whose value is equal to *x*.

`list.`**`pop`**`([`*`i`*`])`
Remove the item at the given position in the list, and return it.

And there are some more:

`list.`**`clear`**`()`, `list.`**`index`**`(`*`x`*`[, `*`start`*`[, `*`end`*`]]), list.`**`count`**`(`*`x`*`), list.`**`sort`**`(`*`key=None`*`, `*`reverse=False`*`),list.`**`reverse`**`(), list.`**`copy`**`()`

Lists and string access: Slicing

Gives access to a specified range of **sequence**'s (e.g. **list, string**) elements.

Synthax: **sequence** *[start:stop[:step]]*

start: Optional. Starting index of the slice. Defaults to 0.
stop: Optional. The last index of the slice or the number of items to get. Defaults to *len(sequence)*.
step: Optional. Extended slice syntax. Step value of the slice. Defaults to 1.

Example:

```
mystring = 'thestring'  #or mylist = ['t','h','e','s','t','r','i','n','g']
print(mystring[2:5])  Returns: 'est'
print(mystring[:5])  Returns: 'theest'
print(mystring[:5:2])  Returns: 'tet'
```

Lets code !

Python for Beginners (MARUM)\Day 2\Exercises\Lesson 1\Lists.ipynb

Iteration II - the for loop

A **for loop** is used for iterating over a **iterables** such as **sequence** (**list, string etc.)**

- Repeats a statement until the end of the sequence is reached
- Iteratively assigns values of the sequence to a variable used in the loop

Synthax: **for variable in sequence :**

```
for i in [1,2,3,4]:
  print(i)
```

As in a while loop you can use the **break** and **continue** statements

The for loop - range()

The **range()** function returns a **sequence** of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

```
for i in range(4):
    print(i)
```

Will return:

```
0
1
2
3
```

Nested Lists

Can be used to define multi-dimensional data structures..

E.g. to create a 2D - matrix... something like a data table:

```
matrix = [[1, 2, 3, 4],[5, 6, 7, 8],[9, 10, 11, 12]]
```

Access:

`matrix[0][0]` will select the value first row and column

Nested Loops

You can also nest for loops, which is useful to iterate through nested lists:

```
for i in [[1,2,3,4],[5,6,7,8]]:
  print(i)
  for j in i:
    print(j)
```

Needed: correct indentation!

Lets code !

Python for Beginners (MARUM)\Day 2\Exercises\Lesson 3\Iteration_for_loop.ipynb

Dictionaries

- A dictionary is a unordered, changeable collection of items (values).
- Each item in a dictionary has a key and a value.
- Are created with curly brackets {}
  - **mydict= {}** creates an empty dict
  - **mydict ={‘first’: 1, ‘second’:2}**
- Can be very useful to create more complex data structures:
  - `person{'first_name':’John’,'last_name':’Doe’}`
  - `data{'month':[1,2,3,4,5,6],'income':[100,200,1200,400,1300,600]}`

Dictionaries

Accessing dictionary values:

```
person['first_name']
person.get('first_name')
```

Assigning or changing dictionary values:

```
person['first_name']=’John’
person['friends']=[‘Jane’,’Ben’]
```

Dictionaries

For Loops:

- Return the keys not the values!
- Use **.values()** to return the values or **.items()** to return the entry
-

```
for i in {‘number’:1}:
    print(i)
for i in {‘number’:1}.values():
    print(i)
```

Sets and Tuples

Sets and tuples are other **sequence** types such as lists

- **Tuples:**
  - Are created with normal brackets () eg: **mytuple =(1,2,3,4,5)**
  - Tuples are **immutable** but **ordered**!

- **Sets:**
  - Are created with curly brackets {} eg: **myset ={1,2,3,4,5}**
  - Tuples are **immutable**, **unordered** and have unique entries !
  - **myset = {1,2,3,2} will just save {1,2,3}**

- **Access to values:**
  - **Sets:** Access by index not possible
  - **Tuples:** Access by index: **e.g. myset[1]**
  - **Both:** **Using a for loop..**

Lets code !

Python for Beginners (MARUM)\Day 2\Exercises\Lesson 3\Dictionaries.ipynb

Day 3

Functions

- A function is a block of code which only runs when it is called.
- Arguments are specified after the function name, inside the parentheses.
- You can have zero or more arguments defined for a function.

```
def my_function(name):

  print(“Hello ” + name)
```

To let a function return a value, use the **return** statement.

Keyword Argument

- The order of the arguments does not matter as you can send the arguments with the key=value syntax.

```
def my_city(city2, city1, city3):

  print("The second city name is " + city2)



my_city(city3 = "Munich", city2 = "Bremen", city1= "Bielefeld")
```

Global vs. Local Variables

- Variables that are defined inside a function body have a **local** scope, and those defined outside have a **global** scope.

```
product = 0; # This is global variable.

def multiply(arg1, arg2):

   product = arg1 * arg2; # Here total is local variable.

   print ("Inside the function local product is : ", product)

   return product;
```

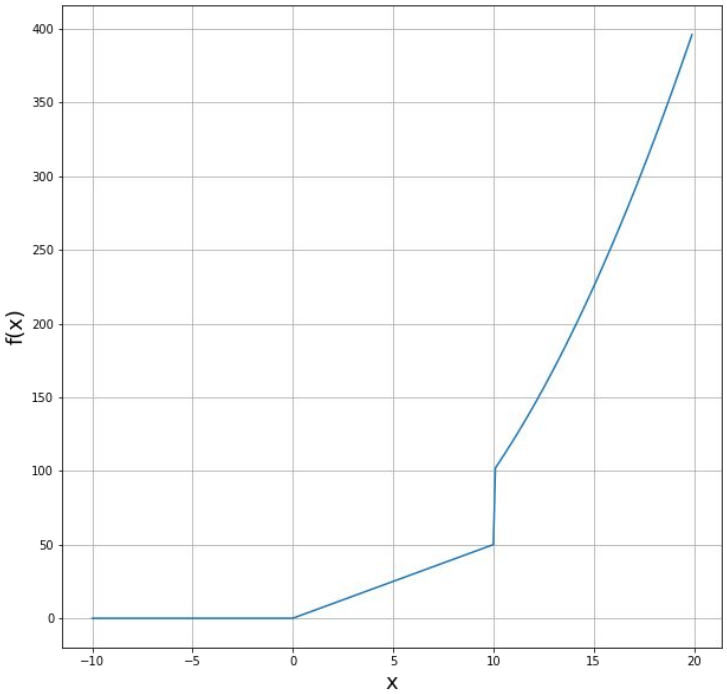
What will be the output of the following statement?

```
multiply( 10, 5 )
```

Lets code !

https://colab.research.google.com/drive/1HfZgdJT3-dMD0JFw4f4gxVrlyg_zuMxY?usp=sharing

Exercise 3. Expected plot



Classes and Objects

- A Class is like an object constructor, or a "template" for creating objects.

```
Color = Green
Manufacturer = Ford
Series = Mustang
Fuel Type = Gasoline
```

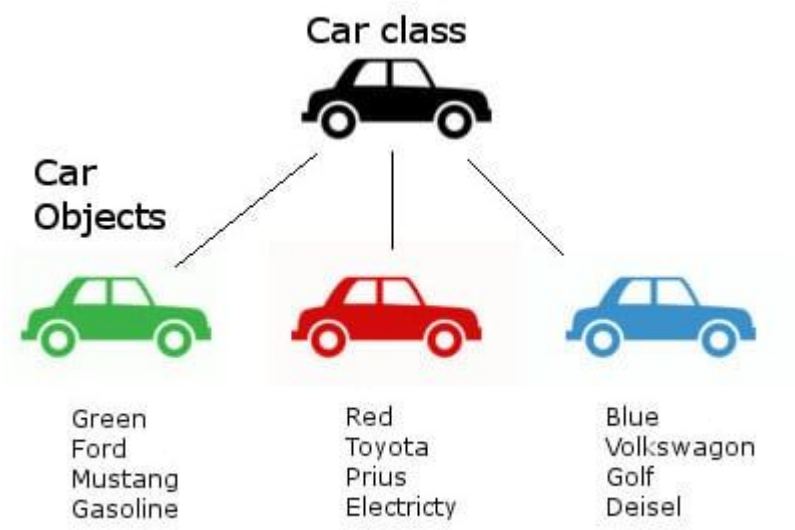


Image source: https://github.com/andrejrs/Object-Oriented-PHP
Slides Content: https://www.w3schools.com/python

Classes and Objects

- Use the keyword `class` to create a class.

```
class MyClass:

    x = 5
```

- Create an object called m1 and print the value of variable x.

```
m1 = MyClass()

print(m1.x)
```

Classes and Objects

- All classes have a function called __init__(), which is always executed when the class is being initiated.

```
class Student:

  def __init__(self, name, age):

    self.name = name

    self.age = age
```

```
p1 = Student("Tanya", 40)

print(p1.name)

print(p1.age)

p2 = Student("Christian", 50)

print(p2.name)

print(p2.age)
```

The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

Object Method

- Objects can also contain methods (functions).

```
class Student:

  def __init__(self, name, age):

    self.name = name

    self.age = age

  def greet(self):

    print("My name is " + self.name)
```

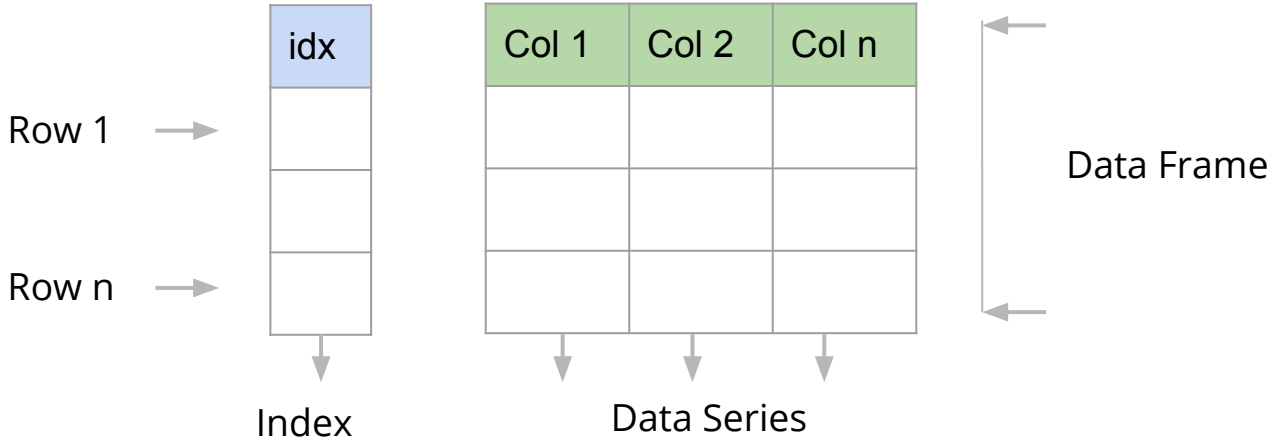
```
p1 = Student("Tanya", 40)

p1.greet()
```

Lets code !

Day 4

Pandas - Introduction

- Pandas is a software library for data manipulation and analysis
- It's key data structure is **DataFrame** which allow to store and manipulate tabular data.
- A DataFrame consists of **DataSeries** which represent data columns



Pandas - use and create

To use pandas you need to import the module

```
import pandas as pd
```

Create an empty DataFrame:

```
df = pd.DataFrame()
```

Or fill the frame with dictionary data:

```
df = pd.DataFrame({'X':[1,2,3],'Y':[4,5,6]})
```

Pandas - load data from file

```
pd.read_csv(pathtofile)
```

Reads a file from the given location and assumes values are separated by comma and the first row represents column name

Example:

```
df = pd.read_csv("data/data.csv")
```

Pandas - basics

Basics:

- `df.head(n)` shows the top n rows
- `df.tail(n)` shows the last n rows
- `df.columns` shows the column names (print required)
- `df.shape` rows and columns (print required)
- `df.info()` quick overview of content and datatypes

Lets code !

Python for Beginners (MARUM)\Day 2\Exercises\Lesson 1\pandas.ipynb

Pandas - selecting data

Selecting columns:

- Data from a distinct column:

  `df.X` or **`df['X']`**

- Data from more columns in given order:

  `df[['Y','X']]`

Pandas - selecting data

Selecting a subset of data:

- A single value form a distinct column:

  ```
  df['X'][pos]
  ```

- A range of rows in a distinct column:

  ```
  df['X'][start:stop] same as df['X'].iloc[start:stop]
  ```

- Rows and columns range:

  ```
  df.iloc[start:stop,start:stop]
  ```

Lets code !

Pandas - basic data analysis

Pandas has some built in functions to analyse data:

- `df[column].min()`
- `df[column].max()`
- `df[column].mean()`
- `df[column].median()`
- `df[column].std()`

- `df.describe()` shows overview statistics
- `df.corr()` shows a correlation matrix (pearson -> r)

Pandas - filtering data

To select rows based on a conditional expression, use a condition inside the selection brackets [].

Example: Find all entries of column ‘age’ in dataframe df which are > 2

```
df[df[‘age’] > 2]
```

Example: Find all entries of column ‘age’ in dataframe df which are between 2 and 4

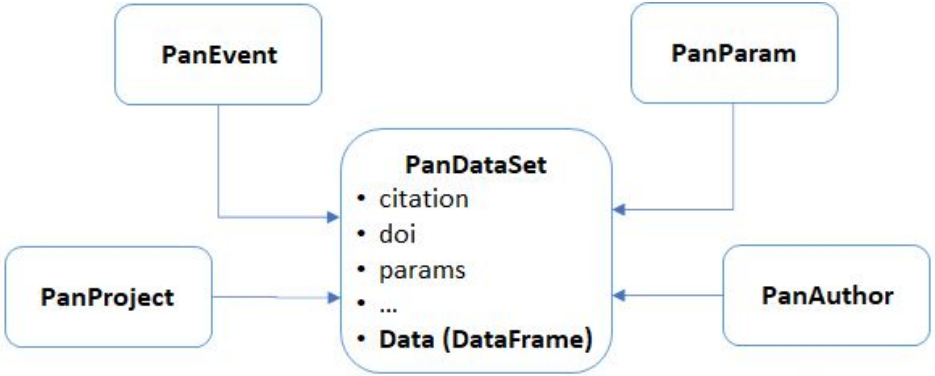
```
df[(df[‘age’] > 2) & (df[‘age’]<4)]
```

Lets code !

https://colab.research.google.com/drive/1Dq6oAtiTesrlCJ8VG5WqdiHu_Ddu7ris

pangaeapy

https://github.com/pangaea-data-publisher/pangaeapy
https://pypi.org/project/pangaeapy/



SciPy pandas matplotlib

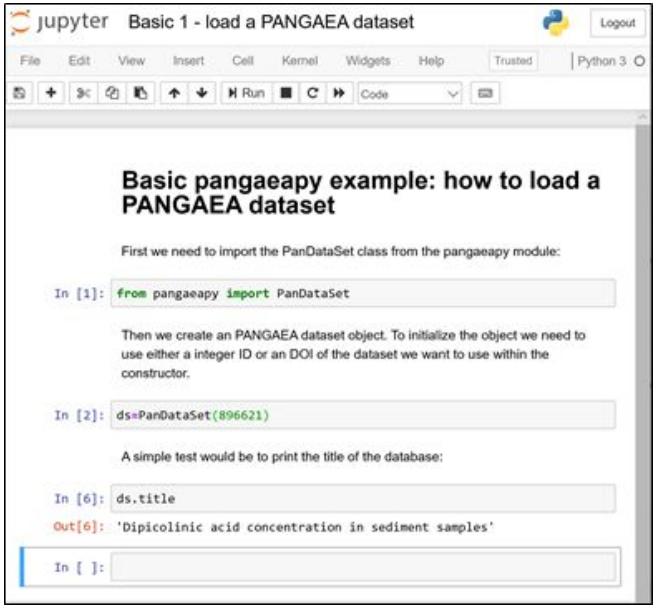
Pangaepy - usage

```
pip install pangaeapy
```

```
from pangaeapy import PanDataSet
ds=PanDataSet(doi or id)
```

Example:

```
ds = PanDataSet(896621)
mydataframe = ds.data
```



Lets code !

https://colab.research.google.com/drive/1Dq6oAtiTesrlCJ8VG5WqdiHu_Ddu7ris

Pandas - grouping

Allows to perform e.g. statistics on groups of data

Example: show mean of all **y** values for

which **x** has the same value.

```
groupedframe = df.groupby('x')
groupedframe.sum()
```

x	y	z
A	12	2
B	2	3
A	1	7
A	13	4
B	8	9

Lets code !

https://colab.research.google.com/drive/1Dq6oAtiTesrlCJ8VG5WqdiHu_Ddu7ris

Pandas - plotting data

Most simple but basic:

```
df.plot()
```

More options:

```
plot(x,y,kind='bar', figsize=(50,100))
```

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.plot.html#pandas.DataFrame.plot