

Python Note

Zainab Nazari, and Carl Stermann-Lücke

Contents

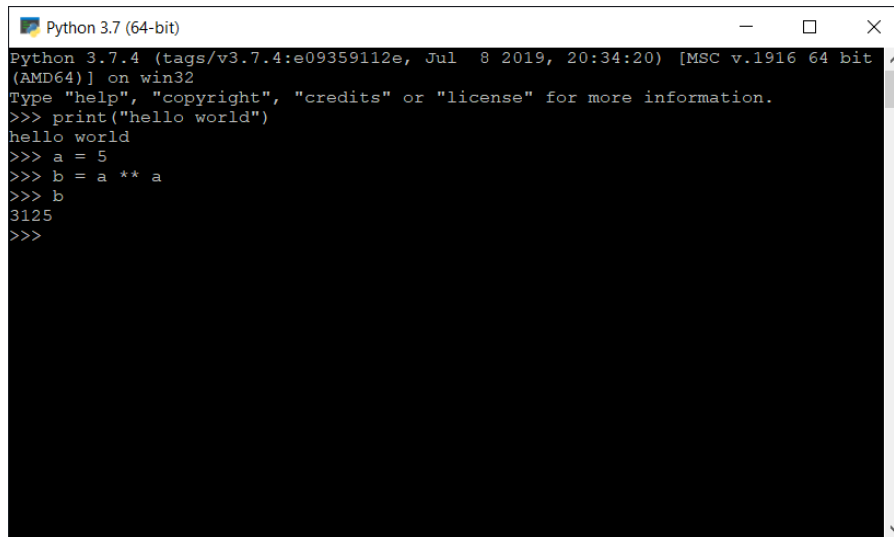
1	Installation, read, write, and run	3
1.1	Mac	3
1.2	Windows	3
1.3	Linux	4
2	Basics	4
2.1	Comments	4
2.2	Help	4
2.3	Data Types	4
2.4	Operators	5
2.5	Lists	5
2.5.1	Copying from a List	6
2.5.2	A counter-intuitive difference between update and reassignment	6
2.6	Indentation	7
2.7	Functions	7
2.7.1	Information About the Function	8
2.8	Method	8
2.8.1	Help on method	8
2.9	Output	10
2.10	Input	10
2.11	Conditional Statements	10
2.12	while-loop	11
2.12.1	while-True	11
2.13	for-loop	11
2.13.1	list comprehension	12
2.13.2	nested for-loop	12
2.14	Sets	12
2.14.1	Use-cases of sets	13
2.15	Dictionaries	13
2.15.1	Iterating through dictionaries	13
2.15.2	Dictionary Comprehension	13
2.15.3	Find key for minimum (maximum) value	14
2.16	Zip	14
2.17	Exception handling	14
2.18	Tuples	15
2.19	Class	16
2.20	Write Data out to Files	16
2.21	Read Data in from Files	17
2.21.1	Read in Several Data files	17
2.22	import	19
2.23	Packages	20
3	Advanced	20
3.1	*args	20
3.2	**kwargs	20

4	NumPy	21
4.1	numpy arrays	21
4.2	slicing	22
4.3	shape and reshape	23
4.4	linspace	23
4.5	arange	24
4.5.1	comparing arange and linspace	25
4.6	meshgrid	25
4.7	random	25
4.8	hstack()	26
4.9	vstack()	26
4.10	Conditional Selection	27
4.11	np.where	27
4.12	Statistical Methods	27
4.13	Error with math versus numpy	28
5	Matplotlib	28
5.1	Plotting List of Data	28
5.1.1	connected data point list plot	29
5.2	Plotting Function	30
5.3	3D plot	31
5.3.1	Extra remark on plotting functions	31
5.4	subplot	33
5.5	Histogram	33
5.6	Customisation	34
5.7	Legends	35
5.8	Colors	36
5.9	Margins	37
6	Pandas	38
6.1	Selection	39
6.1.1	Conditional Selection	40
6.2	Column manipulation	40
7	Scikit-Learn	41
7.1	Supervised Learning	41
7.1.1	kNN in Scikit-Learn	42
7.2	Limitations of supervised learning	42
7.2.1	Insufficient coverage of the attribute space	43
7.2.2	Biased Data	43
7.2.3	Lack of Information	43
7.2.4	Lack of Complexity of the classifier (“underfitting”)	43
7.2.5	Over-complexity of the classifier (“overfitting”)	44
7.3	Why splitting the data is important	44
7.4	Unsupervised Learning	44
7.4.1	Encoding	44
7.4.2	Feature Analysis	44
8	SciPy	44
9	yt-toolkit	44
10	Glossary	44
10.1	Enumerate	44
10.2	Init function	45
10.3	Main condition	45
10.4	Range	46
10.5	ravel()	47
10.6	Basic Git	47
11	Exercise	47

1 Installation, read, write, and run

If you own a Google account you do not need to install anything! Because you can use colaboratory of google. This is the easiest that you can reach to a python notebook and run your code. In the [colaboratory website](#) you can read and write and run the python codes. We generally need to tell apart two different ways of running python

- **Interactive Python**, by using the terminal you can interactively write and run a python code line by line. You just need to call python, and use it interactively in your terminal. For example



```
Python 3.7 (64-bit)
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul 8 2019, 20:34:20) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("hello world")
hello world
>>> a = 5
>>> b = a ** a
>>> b
3125
>>>
```

- **Python Scripts** (.py), which can be written in any editor and run in the terminal. To run a python code, we need to go to the folder/directory where the code is saved, then type `$ python name.py`, and press enter.
- **Interactive Python Notebooks** (.ipynb), which allow for step-by-step evaluation and intermediate output, but needs to be opened by Jupyter Notebook or Google Colab or other compatible notebook environments .

1.1 Mac

There are several ways that you can read and write and run a python code:

- **terminal**: open the terminal on your mac, python by default is installed, so you simply type `$ python`, you would enter the python environment. It also gives you some information that which python version is installed on your mac.
- **atom**, atom is a text editor, which you can write your python with.
- **anaconda navigator** which can be installed from [this website](#). By installing anaconda you can use
 - **Jupyter** which is a web-based, interactive computing notebook environment.
 - **Spyder** which is a scientific Python Development Environment
 - **JupyterLab** which is the next-generation web-based user interface for Project Jupyter.

1.2 Windows

We recommend to install python using an installation file (.exe), downloaded from [the official website](#). It is important to choose the right version: At the time of writing, two different versions of python are officially supported: A Python 2 and a Python 3 release. We exclusively work with Python 3. The differences between the two versions are significant, so choose the link that says "Latest Python 3 Release[...]" . After installing, you can call run python scripts by double-clicking on them. Use an input at the end of the file to keep the terminal window open (explained in section 2.10). You can also install a notebook software called jupyter, which you can use to create and run interactive python notebooks.

1.3 Linux

To install python in linux if you are already using ubuntu 16.10 or newer, open the command line and type: `sudo apt-get update` then `sudo apt-get install python3.8`. For further information maybe look [here](#). If you are using linux you may also use other notebook environment like **Jupyter**, which we explained above. [Here](#) you might find more information on how to install it.

2 Basics

2.1 Comments

There are two types of comments:

1. using dash symbol, `#` in the beginning of any line, will make the line as comment and won't be evaluated during the run.
2. multi-line comments using `'''` or `"""`, we can comment out a part of our code simply by using it before and after the part of the code. example:

```
1 #file name: comment-example.py
2 #This is a line comment.
3 myVariable = 3 #This is also a line comment, but the part before the # can still be
   evaluated.
4 """
5 These
6 lines
7 are
8 a
9 multi-line
10 comment.
11 So the line
12 print(myVariable * 2)
13 is not evaluated.
14 """
15 print(myVariable)
16 '''
17 The output of this code will be:
18 3
19 '''
```

2.2 Help

In order to get information about a particular built-in function or a command in the python shell, we can simply use `help(function)`. This way of calling a function with the help function works for python script, python notebook, and interactive python shell. You can also use the question mark, for example `math.sin?` (having already imported the math module) in the python notebooks, but not in the python shell or script. You can also use the `help(function)` for the function that you have created and made a explanation for that, see [2.7.1](#).

2.3 Data Types

There are several types:

- **string** (str): is a concatenation of characters:
Example: 'text' or "text"
- **integer** (int): is an integer number
Example: 4
- **float** (float): is a rational number.
Example: 3.14
- **list** (list): can be a list of different types of data
Example: [3,1,4,"text",3.14]

Strings and lists, among others, are collectively called sequences. Some operators work differently on sequences than they work on numbers. Additional data types can be created by users. Many of them are already available in libraries, such as numpy arrays explained in [\(4.1\)](#). With the use of different data type function we can alter

2.4 Operators

In python we have all the basic mathematical operators. This includes

=	assignment	$a = 3 \rightarrow a$ is a variable with the value 3
+	addition for numbers, concatenation for sequences	$3 + 2 \rightarrow 5$, <code>"text" + "s" \rightarrow "texts"</code>
-	subtraction	$3 - 2 \rightarrow 1$
*	multiplication for numbers, repetition for sequences	$3 * 2 \rightarrow 6$, <code>"text" * 2 \rightarrow "texttext"</code>
/	division of floating point numbers	$3 / 2 \rightarrow 1.5$
//	division of integers	$3 // 2 \rightarrow 1$
%	modulo-operator, remainder of an integer division	$3 \% 2 \rightarrow 1$
**	power	$3 ** 2 \rightarrow 9$
+=	update (works also with other operators)	$a = 3$, $a += 2 \rightarrow a = 5$
>	greater than for numbers, longer than for sequences	$3 > 2 \rightarrow \text{True}$, <code>"mytext" > "text" \rightarrow True</code>
<	smaller than for numbers, shorter than for sequences	$3 < 2 \rightarrow \text{False}$, <code>"mytext" < "text" \rightarrow False</code>
==	check for identity	$3 == 3 \rightarrow \text{True}$, <code>"text" == "text" \rightarrow True (!)</code>
!=	inverse check of identity	$2 != 3 \rightarrow \text{True}$
not	boolean inversion	<code>not True \rightarrow False</code>
or	boolean or	<code>True or False \rightarrow True</code>
and	boolean and	<code>True and False \rightarrow False</code>
in	containment in a data structure	<code>3 in [1,2,3] \rightarrow True</code>

2.5 Lists

A list is a sequential data type, its elements can be accessed by defining their position that should be returned in the square brackets. In the list the first element has index zero. Elements can also be accessed from the end of the list starting with index -1 . Note that if you delete an element in the list, the indices of the elements of the new list would reduce. Several elements of a list can be deleted at once using the slicing operator. Here is some example of manipulating the data.

```
1 #file name: list.py
2 my_list=[1,2,4.4, 7, ["carl", 27]]
3 #
4 print("my_list: \n", my_list)
5 print("my_list[3]: \n", my_list[3])
6
7 # deleting an element in the list:
8
9 del(my_list[3])
10 print("my_list after deleting my_list[3]:\n ", my_list)
11
12 # accessing the list in the list:
13 print("my_list[-1][:]: \n", my_list[-1][:])
14
15 #replacing element:
16 my_list[0]="zainab"
17 print("my_list after replacing my_list[0]='\n"zainab'\n ", my_list)
18 '''
19 output:
20
21 my_list:
22 [1, 2, 4.4, 7, ['carl', 27]]
23 my_list[3]:
24 7
25 my_list after deleting my_list[3]:
26 [1, 2, 4.4, ['carl', 27]]
27 my_list[-1][:]:
28 ['carl', 27]
29 my_list after replacing my_list[0]="zainab":
30 ['zainab', 2, 4.4, ['carl', 27]]
31
32 '''
```

We should note that there is a slight difference between selecting an element of a sub-list in a list and numpy array. In the list we cannot select the element of sub-list by for example this syntax `x[0,1]=5`, whereas this works in the numpy array. The reason behind this is that unlike the numpy a list is not necessarily an array with specified dimension. For instance a list can have many sub-lists with different lengths, and some of these

sub-lists might contain list, and some might not. The proper way to access element(s) of a list within a list looks something like `x[0][1]`, where `x[0]` is a sub-list where we choose its element at position `[1]`.

2.5.1 Copying from a List

When we copy a list just with the simple equal sign, $y = x$, if the primary list is `x`, then assigning `x` to `y` would copy the reference instead of the elements, and that causes that any changes in `y` will affect the `x` as well (because both point to the same list). There are two ways to copy from a list in such a way that it copies the elements, we elaborate this in the following

```
1 #file name: copy_list.py
2 x=[1,2,3,4,5]
3 print("primary list x= ", x)
4 y=x # copying the refernce x to y!
5 print("y=", y)
6 y[0]=0 # making a change to y
7 print("changed y=", y)
8 print("after changing y, x=", x)
9
10 print("proper way of copying the elements of a list:")
11
12 y=list(x) # proper way of copying the elements of a list (way_1)
13 y=x[:] # proper way of copying the elements of a list (way_2)
14 print("primary list x= ", x)
15 # now let's check!
16 y[0]=-1
17 print("after changing y, x=", x)
18 print("after changing y, y=", y)
19
20 """
21 output:
22 primary list x= [1, 2, 3, 4, 5]
23 y= [1, 2, 3, 4, 5]
24 changed y= [0, 2, 3, 4, 5]
25 after changing y, x= [0, 2, 3, 4, 5]
26 proper way of copying the elements of a list:
27 primary list x= [0, 2, 3, 4, 5]
28 after changing y, x= [0, 2, 3, 4, 5]
29 after changing y, y= [-1, 2, 3, 4, 5]
30
31 """
```

These ways, however, only copy the elements of the list, but not the elements of all the sub-lists. We call this a *shallow copy*. If we want to copy the elements of all lists contained or nested in the list, which we call *deep copy*, we need to apply another technique. The file below explains how it works with using function `deepcopy` from `copy` module.

```
1 #file name: try.py
2 from copy import deepcopy
3 x=[[0,1],2,3]
4 y=deepcopy(x)
5 x[0][1]=9
6
7 print("x=",x)
8 print("y=",y)
9
10 '''
11 output:
12 x= [[0, 9], 2, 3]
13 y= [[0, 1], 2, 3]
14
15 '''
```

2.5.2 A counter-intuitive difference between update and reassignment

You could think that the update operator does the same as computing a new value and reassigning it to the same variable. This is true for numbers, as shown in the following file:

```
1 #file name: UpdateVsReassignment1.py
2 mynumber = 3
3 mynumber += 2#update operator
4 print(mynumber)#result: 5
5
6 mynumber = 3
```

```

7 mynumber = mynumber + 2#reassignment
8 print(mynumber)#result: 5

```

But it's not entirely true for lists. The `+` operator appends a list to a list. It changes the list. The `+` operator, however, makes a new list that consists of the concatenation of the two lists. After assigning it to the same variable, any other variables that point to the original list still point there and not to the concatenation:

```

1 #file name: UpdateVsReassignment2.py
2
3 a = [1,2,3]
4 b = a#b and a now refer to the same list...
5 print("***Update***")
6 a += [4,5]#...so any changes to one of them affect both
7 print("a=",a)#[1, 2, 3, 4, 5]
8 print("b=",b)#[1, 2, 3, 4, 5]
9
10 print("***Assignment***")
11 a = a + [6,7]#This is a reassignment. A new list is assigned to the variable a. So a and b are
    no longer the same list.
12 print("a=",a)#[1, 2, 3, 4, 5, 6, 7]
13 print("b=",b)#[1, 2, 3, 4, 5]
14 """
15 output:
16 ***Update***
17 a= [1, 2, 3, 4, 5]
18 b= [1, 2, 3, 4, 5]
19 ***Assignment***
20 a= [1, 2, 3, 4, 5, 6, 7]
21 b= [1, 2, 3, 4, 5]
22
23 """

```

2.6 Indentation

Spacing matters!

Lines of code that are always run after each other are also in the same level of indentation. That means that their first character is the same number of spaces or tabulator-spaces away from the left end of the line.

2.7 Functions

Functions are block of code that only runs when it is called. They also are the most important way of reusing the same piece of code several times. They help to keep your code shorter and readable by giving a name to a combination of commands. They are also of great value when making code available to others. Some of the functions are built in, which that can be used upon calling their names, some requires to call first their related libraries. A function definition starts with the keyword `"def"`, then a name and a (potentially empty) list of parameters in brackets, followed by a colon. The code that is run when the function is called must be *indented* from the line where the function is defined. The function ends where the indentation ends. A function can return a value, but it is not a necessary condition. To return a value, a return statement has to be run. If no return statement is run, then the function returns `"None"`. To call a function, write the name of the function, followed by the list of parameters in brackets. Example:

```

1 # file name: function.py
2 from math import sin, log, pi # this line imports two functions sin(), log() and a constant (
    pi) from the math library.
3 def func(s): # this is a function which is defined by the coder, and it returns a number.
4     return(sin(s)**2+log(s/5)+pi)
5 if __name__=="__main__":
6     print(func(45)) # in this line two functions are called the func(45) and the print().
7 # the output of this code is computation of sin(45)**2+log(45/5)+3.14=6.062854038990598

```

Note, for interactive python shell, if we define a function, then we need to have a line (by pressing enter key once more) before we call the function itself. Otherwise the code will give error, look at the script below

```

>>> def func(s):
...     return s ** 4
...
>>> func(5)
625
>>>

```

2.7.1 Information About the Function

Functions can carry some information and with the use of `help()` function, we can access to them. For example

```
1 # file name: help_func.py
2 def hi():
3     """Ciao is a greeting in Italy"""
4     print("Ciao!")
5
6 help(hi)
7 hi()
8 '''
9 output:
10 Help on function hi in module __main__:
11
12 hi()
13     Ciao is a greeting in Italy
14 (END)
15
16 '''
```

Note that when the information printed it says: Help on function `name_of_function` in module `__main__`: this means that it is a function of the file that is currently run.

2.8 Method

Methods are functions that are defined on an object. Everything that can be saved in a variable is an object. Some classes of objects are already defined in python, such as strings, floats, integers, lists. About the concept of *class*, look at 2.19. The definition of a class tells which methods are defined on the objects. The concept of method have similarity with the concept of function, however, it can be considered as particular functions that works on specific objects. Each object belongs to a class. In other word, objects are instances of a class. Classes are generalisation of objects. They define what information the type of object contains and which functions can be applied to it. The following example elaborates on that.

```
1 #file name: method_call_example.py
2 mylist = [1,2,3]
3 mylist.append(4)#append is a method, defined on objects of the type list.
4 print(mylist)#print is a function
5 """
6 output:
7 [1, 2, 3, 4]
8 """
```

2.8.1 Help on method

To find out the available functions for a particular method, look at the following code.

```
1 #file name: method_help.py
2
3 #Let's find out all the methods defined in the list class:
4 help([1,2,3]) # you can also call it by help(list) or help(mylist) when you defined your list
5               for example: mylist=[1,2,3]
6 """
7 output:
8 class list(object)
9 |   list() -> new empty list
10 |   list(iterable) -> new list initialized from iterable's items
11 |
12 |   Methods defined here:
13 |
14 |   __add__(self, value, /)
15 |       Return self+value.
16 |
17 |   __contains__(self, key, /)
18 |       Return key in self.
19 |
20 |   __delitem__(self, key, /)
21 |       Delete self[key].
22 |
23 |   __eq__(self, value, /)
24 |       Return self==value.
25 |
26 |   __ge__(self, value, /)
27 |       Return self>=value.
```



```

27 |
28 | __getattr__(self, name, /)
29 |     Return getattr(self, name).
30 |
31 | __getitem__(...)
32 |     x.__getitem__(y) <==> x[y]
33 |
34 | __gt__(self, value, /)
35 |     Return self>value.
36 |
37 | __iadd__(self, value, /)
38 |     Implement self+=value.
39 |
40 | __imul__(self, value, /)
41 |     Implement self*=value.
42 |
43 | __init__(self, /, *args, **kwargs)
44 |     Initialize self. See help(type(self)) for accurate signature.
45 |
46 | __iter__(self, /)
47 |     Implement iter(self).
48 |
49 | __le__(self, value, /)
50 |     Return self<=value.
51 |
52 | __len__(self, /)
53 |     Return len(self).
54 |
55 | __lt__(self, value, /)
56 |     Return self<value.
57 |
58 | __mul__(self, value, /)
59 |     Return self*value.n
60 |
61 | __ne__(self, value, /)
62 |     Return self!=value.
63 |
64 | __new__(*args, **kwargs) from builtins.type
65 |     Create and return a new object. See help(type) for accurate signature.
66 |
67 | __repr__(self, /)
68 |     Return repr(self).
69 |
70 | __reversed__(...)
71 |     L.__reversed__() -- return a reverse iterator over the list
72 |
73 | __rmul__(self, value, /)
74 |     Return self*value.
75 |
76 | __setitem__(self, key, value, /)
77 |     Set self[key] to value.
78 |
79 | __sizeof__(...)
80 |     L.__sizeof__() -- size of L in memory, in bytes
81 |
82 | append(...)
83 |     L.append(object) -> None -- append object to end
84 |
85 | clear(...)
86 |     L.clear() -> None -- remove all items from L
87 |
88 | copy(...)
89 |     L.copy() -> list -- a shallow copy of L
90 |
91 | count(...)
92 |     L.count(value) -> integer -- return number of occurrences of value
93 |
94 | extend(...)
95 |     L.extend(iterable) -> None -- extend list by appending elements from the iterable
96 |
97 | index(...)
98 |     L.index(value, [start, [stop]]) -> integer -- return first index of value.
99 |     Raises ValueError if the value is not present.
100 |
101 | insert(...)
102 |     L.insert(index, object) -- insert object before index

```

```

103 |
104 | pop(...)
105 |     L.pop([index]) -> item -- remove and return item at index (default last).
106 |     Raises IndexError if list is empty or index is out of range.
107 |
108 | remove(...)
109 |     L.remove(value) -> None -- remove first occurrence of value.
110 |     Raises ValueError if the value is not present.
111 |
112 | reverse(...)
113 |     L.reverse() -- reverse *IN PLACE*
114 |
115 | sort(...)
116 |     L.sort(key=None, reverse=False) -> None -- stable sort *IN PLACE*
117 |
118 | -----
119 | Data and other attributes defined here:
120 |
121 | __hash__ = None
122 |
123 | """

```

2.9 Output

You can output text to the console (in case you use a script or interactive python) or under your current cell (in case you use a notebook) by typing `print()`. You need to add what you want to output between the brackets. It can be a variable or a string, a number, etc. You can also output multiple things in one line, by separating them by comma.

2.10 Input

You can input the data in your code using the keyboard. To do so, you just need to use input function: `input()`. However, you can specify the type your input with another function. Note that the input function would return string. Example:

```

1 # file name: input.py
2 #We want to ask a number from the user and print the number multiplied by 2
3 a = input("What is your number?") # The console prints "What is your number?" and waits for
4   the user to enter something and press ENTER
5 print(a * 2) # This only prints the input twice, but doesn't do multiplication. The reason is
6   that input returns a string.
7 print(int(a) * 2) # So we need to convert a to an integer. We do that using the function int()
8   . Now it works!

```

2.11 Conditional Statements

This code shows how to write conditional statements:

```

1 # file name: if-statement-example.py
2 number = 24352
3 if(number % 2 == 0):
4     print(number, "is an even number") #This line is only run if the condition in the if-
5   statement is true.
6 else:
7     print(number, "is an odd number") #This line is only run if the condition in the if-
8   statement is false.

```

The indented parts are only run under a certain condition. It is also possible to include more branch options using `elif`:

```

1 # file name: if-statement-example2.py
2 number = 24352
3 if(number == 0):
4     print(number, "is 0") #This line is only run if the condition in the if-statement is true.
5 elif(number % 2 == 0):
6     print(number, "is an even number") #This line is only run if the condition in the elif-
7   statement is true and the condition in the if statement (and any possible elif statements
8   before this) is false.
9 else:
10    print(number, "is an odd number") #This line is only run if the conditions in the if and the
11   elif statements are all false.

```

is

2.12 while-loop

While loop does a task which is repetitive, and that is very useful if you want to avoid repeating the same piece of code over and over again. With while loop you can tell the program how many times the same task should be repeated. For instance to compute the factorial of a number we need to multiply all bunch of the numbers below the number and the number itself. This can be done by the while loop more easily. The while loop will go over the same task as often as we want to. We also use the while loop in the occasion where we do not know how often we are going to repeat the same task until we give an input to the code. While-loops can also be nested, that means the body of a while-loop can contain further while-loops.

```
1 # file name: factorail.py
2 # this code compute the factorail of a given number
3 a=input("what is your number? ")
4 b=1
5 c=1
6 while b<int(a):
7     c=c*(b+1)
8     b=b+1
9 print(c)
```

Here is another example where the input number is going to be repeatedly typed until it reaches the number itself

```
1 # file name: repeat.py
2 '''
3 this code is going to be take a number and then type it something like
4 let's assume the given number is 5
5 then our code is going to print the result as follow
6 5
7 55
8 555
9 5555
10 55555
11 '''
12 b=input("what is your number? ")
13 c=1
14 while c<=int(b):
15     print(b*c)
16     c=c+1
```

2.12.1 while-True

Here is an example which can show how can we have an open loop while code. It basically means that the loop continues non-stop unless we want to break it. For example in the following code we can input as many integer number as we wish and once we enter q, it gives us the sum and the product of the numbers.

```
1 #file name=sum_product.py
2 # this code can take numbers (as much as you give- if not enter the letter 'q' to exit )
3 # and print the sum and product of them.
4
5 s = 0
6 p = 1
7
8 while True: #This is just a regular while loop, but the condition always stays true. The loop
9     will continue indefinitely...
10     num = input('Number or q to quit: ')
11     if num == 'q':
12         break #...unless it is broken using the "break"-command. The "break"-command ends the
13         while loop.
14     num = int(num)
15     s += num
16     p *= num
17 print('Sum:',s,'Product:',p)
```

2.13 for-loop

Unlike a while-loop, a for-loop does not run until a specified condition is met, but rather runs over sequential data types and runs its body for every element in that sequence. Therefore it is particularly useful to traverse sequences. It can also be used in combination with range.

```
1 # file name: for_loop.py
2 a=[1,4,7,9]
3 for x in a:
```

```

4     print(x*2)
5     '''
6     the output is as follow:
7     2
8     8
9     14
10    18
11    '''

```

2.13.1 list comprehension

We can use for loops to convert one list into another. But there is an easier way of doing that, which is called list comprehension. We create a list, where the elements are defined relative to a for loop.

```

1 #file name: list_comprehension.py
2 #old way:
3 newList = []
4 oldList = [1,2,3]
5 for i in oldList:
6     newList += [i/2]
7 #new way:
8 newList2 = [i/2 for i in oldList]
9 print("old way:", newList)
10 print("new way:", newList2)
11 """
12 output:
13 [0.5, 1.0, 1.5]
14 [0.5, 1.0, 1.5]
15 """

```

2.13.2 nested for-loop

```

1 # file name: nested_for_loop.py
2
3 firstList = [2,3]
4 secondList = [4,5,6]
5 for i in firstList:
6     for j in secondList:
7         print("(" ,i ,",", "j ,")")
8
9 """
10 the output:
11 ( 2 , 4 )
12 ( 2 , 5 )
13 ( 2 , 6 )
14 ( 3 , 4 )
15 ( 3 , 5 )
16 ( 3 , 6 )
17 """

```

2.14 Sets

A set is a collection that does not store the order nor the number of appearances of objects. The objects in a set are called members of the set. It stores element, however, it only stores which elements belong to the set. It does not store how often an element belongs to a set or where in the set it is positioned. Sets also implement various operations of set theory (mathematics), such as “difference”, “issubset”, “issuperset”, “isdisjoint”, “union” or “intersection”, etc. The set can find out if an element is there because it can internally sort the elements and can do that simply because it doesn’t need to preserve the order in which the elements are given. Searching in sorted structure is faster than searching in an unsorted structure. Example

```

1 #file name: set.py
2 l = [4,2,6,1,"Carl",2]
3 print(l)
4 s = set(l)
5 print(s)
6 print("Zainab" in s)
7 print(2 in s)
8 print(2 in l)
9 #We can more easily make a set from scratch, without making a list first, using curly brackets
10 s1 = {4,2,5,1,4,3}
11 print(s1)
12 """

```

```

13 output:
14 [4, 2, 6, 1, 'Carl', 2]
15 {1, 2, 4, 6, 'Carl'}
16 False
17 True
18 True
19 {1, 2, 3, 4, 5}
20 """

```

2.14.1 Use-cases of sets

The membership query operator, `in`, is also available for lists. The advantage of using it on sets is that the query runs faster. This is because the set can store the data internally in a way that helps it find the elements faster (for example in some sorted structure), because it does not have to preserve the order of the elements. If the membership of elements is queried relatively often, then it is useful to store the elements in a set. Note that converting a list to a set also takes time.

2.15 Dictionaries

A dictionary is a data-structure that allows to look up values associated to keys. Example:

```

1 #file name: dictionary_example.py
2 ages={"Egypt": 5000, "Carl": 27, "Zainab": 31, "Albert Einstein": 141, "China": 5000}
3 print(ages["China"])
4
5 output:
6 5000
7 """

```

2.15.1 Iterating through dictionaries

We can use a for loop to iterate through the keys of a dictionary, just like through a list. For each element, we can access the associated value. See the following example:

```

1 #file name: dictionary_iteration.py
2 ages={"Egypt": 5000, "Carl": 27, "Zainab": 31, "Albert Einstein": 141, "China": 5000}
3 print("keys:")
4 for item in ages:
5     print(item)
6
7 print("values:")
8 for item in ages:
9     print(ages[item])
10
11 output:
12 keys:
13 Egypt
14 Carl
15 Zainab
16 Albert Einstein
17 China
18 values:
19 5000
20 27
21 31
22 141
23 5000
24 """

```

2.15.2 Dictionary Comprehension

Similar to lists (described in 2.13.1), we can also use a systematic way of constructing dictionaries in one line of code. We call this "dictionary comprehension". Example:

```

1 #file name: dictionary_comprehension.py
2 radius=[1,2,3,4,5]
3 radiusToArea = {r:(r**2)*3.14 for r in radius}#This is a dictionary with radii of circles as
4 keys and areas as values.
5 print(radiusToArea)
6 """
7 output:

```

```

7 {1: 3.14, 2: 12.56, 3: 28.26, 4: 50.24, 5: 78.5}
8 """

```

2.15.3 Find key for minimum (maximum) value

In many applications, it is useful to find the element that has the highest (or lowest) associated value. We can also sort dictionaries by value. Example:

```

1 #file name: dictionary_min_max.py
2 ages={"Egypt": 5000, "Carl": 27, "Zainab": 31, "Albert Einstein": 141, "China": 5000}
3 print(min(ages))#first key in alphabetical order
4 print(max(ages))#last key in alphabetical order
5 print(sorted(ages))#sorts keys alphabetically
6 print(min(ages, key=ages.get))#key where the value is minimum.
7 print(max(ages, key=ages.get))#It returns the first key with the maximum value.
8 print(sorted(ages, key=ages.get))#list of keys, sorted by their values.
9 """
10 output:
11 Albert Einstein
12 Zainab
13 ['Albert Einstein', 'Carl', 'China', 'Egypt', 'Zainab']
14 Carl
15 Egypt
16 ['Carl', 'Zainab', 'Albert Einstein', 'Egypt', 'China']
17 """

```

2.16 Zip

Sometimes you want to compute a number of values based on two lists of values. In this case, a simple list comprehension is not sufficient, because you can only iterate through one of the lists. An appropriate solution for this problem is the zip function. It takes several lists (you can choose as many as you like) and creates one list of tuples where the ith element is from the ith list and their order is the same as in the original lists. Example:

```

1 #file name: zip_example.py
2 number_list = [1, 2, 3]
3 str_list = ['one', 'two', 'three']
4 l=list(zip(number_list, str_list))
5 print(l)
6 """
7 output:
8 [(1, 'one'), (2, 'two'), (3, 'three')]
9 """
10 #Why is this useful?:
11 #Assume we want to have a dictionary like this:
12 #{1:"one", 2:"two", 3:"three"}
13 print({i[0]:i[1] for i in zip(number_list, str_list)})
14 """
15 output:
16 {1: 'one', 2: 'two', 3: 'three'}
17 """

```

2.17 Exception handling

Sometimes depending in data, a python command can cause errors. In a script you would want to avoid errors if you can foresee them. This is useful, because errors usually stop the program. The following two examples can illustrate the concepts better. First we present a code which cause an error.

```

1 # file name: error.py
2 list=[1,3,55,0,9,88,20,50,0,24,44,76,100]
3 for i in list:
4     print(30/i)
5 '''
6 output:
7 30.0
8 10.0
9 0.5454545454545454
10 Traceback (most recent call last):
11   File "except.py", line 4, in <module>
12     print(30/i)
13 ZeroDivisionError: division by zero
14 '''

```

Now, we foreseen the error.

```
1 #file name: except.py
2 list=[1,3,55,0,9,88,20,50,0,24,44,76,100]
3 for i in list:
4     try:
5         print(30/i)
6     except ZeroDivisionError:
7         print("Oh, that's bad!")
8
9 output:
10
11 30.0
12 10.0
13 0.5454545454545454
14 Oh, that's bad!
15 3.3333333333333335
16 0.3409090909090909
17 1.5
18 0.6
19 Oh, that's bad!
20 1.25
21 0.6818181818181818
22 0.39473684210526316
23 0.3
24
25 ,,,
```

2.18 Tuples

It is often practical to store values that belong together in the same variable. This becomes particularly useful if you want to return multiple values from a function. The function can then simply return a tuple. A tuple can be created by writing variables behind each other, separating them by commas and surrounding all by round brackets. For example: (2,10). The individual values of a tuple can be accessed either by treating the tuple like a list:

first_value = my_tuple[0]

second_value = my_tuple[1]

Or by assigning the tuple to multiple variables: first_value, second_value = my_tuple

In the following code, we use tuples to return two values from a function:

```
1 #file name: multi_return.py
2
3 def smallest_element(list_of_things):
4     """
5     Takes a list of comparable items. Returns the position of the smallest element and its
6     value.
7     """
8     smallest_index = 0
9     smallest_value = list_of_things[0]
10    for index, element in enumerate(list_of_things):
11        if element < smallest_value:
12            smallest_value = element
13            smallest_index = index
14    return (smallest_index, smallest_value)
15
16 if __name__ == "__main__":
17     """The following test code is run only if the file is run, not if it is imported:"""
18     small = smallest_element([9,3,1,4,2,8,4,6,2,6,2,5,72,1,76,8,-1,3,1000])
19     print(small)
20     print(small[0])
21     print(small[1])
22
23 output:
24 (16, -1)
25 16
26 -1
27 """
```

Note that in this code, we are using the concepts of enumerating a list, which itself is a function that returns two values and is described in section 10.1. The test code for this function is only run if the file itself is run, not if it is imported, because we use the main condition (10.3).

2.19 Class

We have learned about data types of Python in section 2.3. Each of these and more data types are implemented by a construct called class. We can also define our own classes for custom data types. In the following example, we write our own data type called time, which is a blueprint of how times work.

```
1 # file name: class.py
2
3 #We define a blueprint of how a time looks like and what it can do:
4 class time:
5     #Here we define which attributes an object of the class time has and how they are
6     #initialized:
7     hour=1
8     minute=1
9     #Here we define which functions the class time offers:
10    #settime sets the time to a given hour and minute.
11    def settime(self,hour,minute):
12        self.hour=hour
13        self.minute=minute
14    #addmin adds a given number of minutes to the time.
15    def addmin(self, adm):
16        self.hour+=(self.minute+adm)//60
17        self.minute=(self.minute+adm)%60
18        self.hour=self.hour%24
19    #Print time prints the time in a nicely formatted way.
20    def printtime(self):
21        print(self.hour,'h', self.minute, 'min')
22
23 #Here we start using the class.
24 mytime=time()#We now have an object of the class time which is called mytime.
25 mytime.settime(19, 20)#This is how we call a function of that object: objectname.functionname(
26     parameters).
27 #The mytime is now 19 hours and 20 minutes.
28 mytime.addmin(40)
29 #The mytime is now 20 hours and 0 minutes - because we added 40 minutes.
30 mytime.printtime()
31 mytime.addmin(400)
32 #The time is now 2 hours and 40 minutes, because we added 400 minutes to the previous state of
33     mytime, which was 20 hours and 0 minutes.
34 mytime.printtime()
35 print(type(mytime))
36
37 '''
38 the output:
39 20 h 0 min
40 2 h 40 min
41 <class '__main__.time'>
42 '''
```

The purpose of a class is to make instances of it. We do that by defining a name of the instance and type assign the name of the class followed by (). Each class can have *methods*. Methods are similar to functions but they are defined in the class definition, and are referring to an instance of that class (that means an object of that class). They can tell us about the content of the instance or modify it. We call them by typing the name of the instance dot the name of the function, as shown in the example. It is also possible and often necessary to query which class an object belongs to. We can do that by using the function “type”.

Find a simpler example in section 10.2.

2.20 Write Data out to Files

We have used function for writing file which belongs to numpy package that we discuss later. We can open a file in python and write into it, here is an example

```
1 # file name: write.py
2 '''
3 here is a simple example to show how we can open a file and write into it the data that in
4     this case, it is an array of 5 digits, however, the write file can only accept strings,
5     therefore we need to convert the integeres to strings.
6 '''
7
8 x=[1,2,3,4,5]
9 file=open('data.csv','w') # here 'w' means write, note that it also it works with .dat, and .
10    txt format
11 file.write(str(x))
12 '''
```



```

10 output in the file:
11 [1 2 3 4 5]
12
13 ',,'
14 # Note, if data.txt already exists, then if you run the code, the content of the data.txt is
    going to be replaced.

```

As an alternative way to write in a file using numpy we can use the following code.

```

1 # file name: save.py
2 # this is a way to save data using numpy
3 import numpy as np
4 x=np.array([[1,2,3,4,5],[1,2,3,4,5]])
5 np.savetxt('ourdata.txt',x) # it also works with .dat files.

1 1.0000000000000000e+00 2.0000000000000000e+00 3.0000000000000000e+00
  4.0000000000000000e+00 5.0000000000000000e+00
2 1.0000000000000000e+00 2.0000000000000000e+00 3.0000000000000000e+00
  4.0000000000000000e+00 5.0000000000000000e+00

```

2.21 Read Data in from Files

For reading data into python we can also use `open()` function for reading, as an example

```

1 # file name: read.py
2 # here is the code to read data into our file
3 f=open("data.csv", "r")
4 contents=f.read()
5 myarray=eval(contents)
6 for n in myarray:
7     print(n*2)
8 #print(myarray*2, type(myarray))

```

There is also other alternative that works with numpy, but it requires file to be formatted just like `saveetxt()` formats it. Here is an example

```

1 # name of file: read2.py
2 # loading data (read data in) with loadtxt
3 import numpy as np
4 newarray=np.loadtxt('ourdata.txt')
5 for x in newarray:
6     print(x**2)

```

Note that, your file that is going to be loaded or be written in it, doesn't necessary have to be in the same folder that the code is. We can also give a path to the directory of the file either absolute path or relative.

2.21.1 Read in Several Data files

Here is an example on how to read several data files using `glob`.

```

1 # file name: several_read.py
2 import glob as gl
3 file_names=gl.glob("dpc-covid19-ita-province-*")
4 print(file_names)
5 for name in file_names:
6     print(name)
7     print(open(name,"r").read())
8 ',,'
9 part of the output:
10 ['dpc-covid19-ita-province-20200312.csv', 'dpc-covid19-ita-province-latest.csv', 'dpc-covid19-
   ita-province-20200311.csv']
11 dpc-covid19-ita-province-20200312.csv
12 data,stato,codice_regione,denominazione_regione,codice_provincia,denominazione_provincia,
   sigla_provincia,lat,long,totale_casi,note_it,note_en
13 2020-03-12T17:00:00,ITA,13,Abruzzo,069,Chieti,CH,42.35103167,14.16754574,20,,
14 2020-03-12T17:00:00,ITA,13,Abruzzo,066,L'Aquila,AQ,42.35122196,13.39843823,8,,
15 2020-03-12T17:00:00,ITA,13,Abruzzo,068,Pescara,PE,42.46458398,14.21364822,48,,
16 2020-03-12T17:00:00,ITA,13,Abruzzo,067,Teramo,TE,42.6589177,13.70439971,8,,
17 .
18 .
19 .
20 dpc-covid19-ita-province-latest.csv
21 data,stato,codice_regione,denominazione_regione,codice_provincia,denominazione_provincia,
   sigla_provincia,lat,long,totale_casi,note_it,note_en
22 2020-05-23T17:00:00,ITA,13,Abruzzo,069,Chieti,CH,42.35103167,14.16754574,817,,
23 2020-05-23T17:00:00,ITA,13,Abruzzo,066,L'Aquila,AQ,42.35122196,13.39843823,246,,

```

```

24 2020-05-23T17:00:00,ITA,13,Abruzzo,068,Pescara,PE,42.46458398,14.21364822,1508,,
25
26 .
27 .
28 .
29 dpc-covid19-ita-province-20200311.csv
30 data,stato,codice_regione,denominazione_regione,codice_provincia,denominazione_provincia,
    sigla_provincia,lat,long,totale_casi,note_it,note_en
31 2020-03-11T17:00:00,ITA,13,Abruzzo,069,Chieti,CH,42.35103167,14.16754574,9,,
32 2020-03-11T17:00:00,ITA,13,Abruzzo,066,L'Aquila,AQ,42.35122196,13.39843823,6,,
33 2020-03-11T17:00:00,ITA,13,Abruzzo,068,Pescara,PE,42.46458398,14.21364822,18,,
34 .
35 .
36 .
37
38 ,,,

```

If we want to manipulate the data, pandas makes it convenient. The pandas is a package explained in detail in section (6). Here we show how we can read in data using pandas. Here is an example on how to read several .csv files using pandas. In this example we read in all the files containing COVID-19 total cases for different dates. In this case there are two of them.

```

1 # file name: read_several_pandas.py
2 import glob
3 import pandas as pd
4
5 for i in glob.glob('dpc-covid19-ita-province-2020*'): # it will take all the files with the
    prefix before *
6     data=pd.read_csv(i)
7     print(data)
8 ,,,
9 output:
10
11      data  stato  codice_regione  denominazione_regione  ...      long
12      totale_casi  note_it  note_en
13 0  2020-03-12T17:00:00  ITA      13      Abruzzo  ...  14.167546
14   20      NaN      NaN
15 1  2020-03-12T17:00:00  ITA      13      Abruzzo  ...  13.398438
16   8      NaN      NaN
17 2  2020-03-12T17:00:00  ITA      13      Abruzzo  ...  14.213648
18  48      NaN      NaN
19 3  2020-03-12T17:00:00  ITA      13      Abruzzo  ...  13.704400
20   8      NaN      NaN
21 4  2020-03-12T17:00:00  ITA      13      Abruzzo  ...  0.000000
22   0      NaN      NaN
23 ..      ...      ...      ...      ...      ...
24 ...      ...      ...
25 123 2020-03-12T17:00:00  ITA      5      Veneto  ...  12.245074
26  279      NaN      NaN
27 124 2020-03-12T17:00:00  ITA      5      Veneto  ...  12.338452
28  205      NaN      NaN
29 125 2020-03-12T17:00:00  ITA      5      Veneto  ...  10.993527
30  150      NaN      NaN
31 126 2020-03-12T17:00:00  ITA      5      Veneto  ...  11.545971
32  122      NaN      NaN
33 127 2020-03-12T17:00:00  ITA      5      Veneto  ...  0.000000
34  128      NaN      NaN
35
36 [128 rows x 12 columns]
37
38      data  stato  codice_regione  denominazione_regione  ...      long
39      totale_casi  note_it  note_en
40 0  2020-03-11T17:00:00  ITA      13      Abruzzo  ...  14.167546
41   9      NaN      NaN
42 1  2020-03-11T17:00:00  ITA      13      Abruzzo  ...  13.398438
43   6      NaN      NaN
44 2  2020-03-11T17:00:00  ITA      13      Abruzzo  ...  14.213648
45  18      NaN      NaN
46 3  2020-03-11T17:00:00  ITA      13      Abruzzo  ...  13.704400
47   5      NaN      NaN
48 4  2020-03-11T17:00:00  ITA      13      Abruzzo  ...  0.000000
49   0      NaN      NaN
50 ..      ...      ...      ...      ...      ...
51 ...      ...      ...
52 123 2020-03-11T17:00:00  ITA      5      Veneto  ...  12.245074
53  185      NaN      NaN
54 124 2020-03-11T17:00:00  ITA      5      Veneto  ...  12.338452
55  179      NaN      NaN

```

```

33 125 2020-03-11T17:00:00 ITA 5 Veneto ... 10.993527
    110 NaN NaN
34 126 2020-03-11T17:00:00 ITA 5 Veneto ... 11.545971
    92 NaN NaN
35 127 2020-03-11T17:00:00 ITA 5 Veneto ... 0.000000
    40 NaN NaN
36
37 [128 rows x 12 columns]
38 '''

```

2.22 import

We can import files as we import libraries into a python code, and we can also import a part of a code like functions that is initially written in an old file. The old file (or the file whose content is being imported) can be called as a module ¹. Let's import a simple example where we defined a function in another python file and now we would like to use that in our new python file so we need to import that. Here is the file that we create for having the function we want to import

```

1 #file name: func2.py
2 # in this file we just define a simple function that can get an integer and return it as a
  power of three.
3 def myfun(s):
4     return s**3
5 print(myfun(2))
6
7 '''
8 output:
9 8
10
11 '''

```

here is the file that imports our function and uses it.

```

1 # file name: import.py
2 # here we import the function that we have defined in another file called: func2.py
3 from func2 import myfun
4 print(myfun(3))
5 '''
6 output:
7 8
8 27
9 '''

```

As you notice, when we run import.py, the output is not only the result of the function that we call for, but also the result of the file which contains that function. To avoid that we need to learn about *main condition*, which we have explained in 10.3. And the appropriate codes for our new file is as follows. Here is the change that we need to make for the main file which contains the function.

```

1 #file name: func3.py
2 def myfun(s):
3     return s**3
4 if __name__=="__main__":
5     print(myfun(2))
6
7 '''
8 output:
9 8
10 '''

```

here is the result that we expect.

```

1 # file name: import2.py
2 # here we import the function that we have defined in another file called: func2.py
3 from func3 import myfun
4 print(myfun(3))
5 '''
6 output:
7 27
8 '''

```

¹a collection of modules can make a library or package

2.23 Packages

There are plenty of useful functions and methods written for python, however, not all of them are installed by default. Packages are collections of modules, which includes functions, methods etc for specific purposes. Based on the requirement of the users they can be installed and being used. The packages are abundant and makes no sense to include them in python initially, and also they are under constant development, therefore, anyone who wants to develop or use a package, they need to install them first. Here is a simple way of installation (a package called NumPy) using `pip`:

1. go to this link <https://pip.readthedocs.io/en/stable/installing/>
2. download `get-pip.py`
3. go to the terminal:
 - type: `python3 get-pip.py`
 - type: `pip3 install numpy`

If you want to figure out the defined functions, etc inside the packages, you can use `dir()`. For instance, if you want to get into NumPy package, you can type `print(dir(numpy))`, provided that you already imported NumPy.

Note that we can also import a sub-package from a package, one example is when we import `pyplot` sub-package from `matplotlib` package, `matplotlib.pyplot`. See 5.1. The official documentation on how to make packages and modules of your own can be found [here](#). Here is also an interesting blog to learn more about

3 Advanced

3.1 *args

So far we have written functions and methods that receive specified number of parameters. Now, we learn how to make functions receiving arbitrary numbers of arguments. Example:

```
1 #file_name: args.py
2 def test(parameter):
3     print(parameter)
4
5 test(4)
6 test([1,2,3])
7
8 def test1(*args):
9     print(args)
10
11 test1(3,8, "Carl", [4,2,3,1])
12 """
13 output:
14 4
15 [1, 2, 3]
16 (3, 8, 'Carl', [4, 2, 3, 1])
17 """
```

3.2 **kwargs

There is a second option to pass an arbitrary number of arguments: If we add two `*` in front of the parameter, it becomes a dictionary. We are then supposed to provide a name for every one of the parameters that we give - we get an error otherwise. Example:

```
1 #file_name: kwargs.py
2 def test(parameter):
3     print(parameter)
4
5 test(4)
6 test([1,2,3])
7
8 def test1(**kwargs):
9     print(kwargs)
10
11 test1(blue=3,green=8, red="Carl", pink=[4,2,3,1])
12 """
13 output:
14 4
```

```

15 [1, 2, 3]
16 {'blue': 3, 'green': 8, 'red': 'Carl', 'pink': [4, 2, 3, 1]}
17 """

```

4 NumPy

NumPy is a library that brings many functions and data structures that are useful for scientific computing in general and matrix computing in particular.

4.1 numpy arrays

we can arbitrary make an array of numbers, see the following

```

1 # file name: array.py
2 import numpy as np
3 x=np.array([[1,2],[3,4]])
4 print(x)
5 '''
6 output:
7 [[1 2]
8  [3 4]]
9
10 '''

```

NumPy deals with arrays and matrices. In the following example you can see how an operation on a list differs from the numpy array.

```

1 # file name: numpy_ex1.py
2 list1=[1,2,3,4]
3 list2=[1,2,3,4]
4 list3=[[1,2,3,4],[1,2,3,4]]
5 #print("list1*list2= ",list1*list2) # this will give error, the operation of multiplication on
   lists is not defined!
6 print("list1+list2= ",list1+list2)
7 print("list3+list1= ",list3+list1)
8 import numpy as np
9 numpyarray1=np.array([1,2,3,4])
10 numpyarray2=np.array([1,2,3,4])
11 numpyarray3=np.array([[1,2,3,4],[1,2,3,4]])
12 print("numpyarray1*numpyarray2= ", numpyarray1*numpyarray2)
13 print("numpyarray1+numpyarray2= ", numpyarray1+numpyarray2)
14 print("numpyarray3+numpyarray1= ", numpyarray3+numpyarray1)
15 print("numpyarray3*numpyarray1= ", numpyarray3*numpyarray1)
16
17 '''
18 output:
19
20 list1+list2=  [1, 2, 3, 4, 1, 2, 3, 4]
21 list3+list1=  [[1, 2, 3, 4], [1, 2, 3, 4], 1, 2, 3, 4]
22 numpyarray1*numpyarray2=  [ 1  4  9 16]
23 numpyarray1+numpyarray2=  [2  4  6  8]
24 numpyarray3+numpyarray1=  [[2  4  6  8]
25  [2  4  6  8]]
26 numpyarray3*numpyarray1=  [[ 1  4  9 16]
27  [ 1  4  9 16]]
28
29 '''

```

Elements in the numpy array can be any types, however if the types are different from one another, then one type would be the type of the array we set, that means all the elements are converted to the same type. Python decide which type will that be. For instance, if you use mix of integers and strings, the string would be the final type of all elements. Also note that when you initially decide on which type you want to create your array, and later you if you insert an element within the array with different type, the type of the new element would be converted to the original type of the array, if possible. Look to the following examples.

```

1 # file name: numpy_ex2.py
2 import numpy as np
3 x1=np.array([1,2,3,4])
4 x2=np.array([1,2,3,"C"])
5 print(type(x1[0]),type(x2[0]))
6 '''
7 output:
8

```

```

9 <class 'numpy.int64'> <class 'numpy.str_'>
10
11 '''
12 # now we make a change in x2
13 x2[3]=4
14 print(x2)
15 print(type(x2[3]))
16
17 '''
18 output:
19 ['1' '2' '3' '4']
20 <class 'numpy.str_'>
21 '''
22 # you see even though we have changed the last element in x2 to integer,
23 # the type still remains as string.
24
25 x1[3]=7.7
26 print(x1)
27 print(type(x1[3]))
28
29 '''
30 output:
31 [1 2 3 7]
32 <class 'numpy.int64'>
33 '''
34 # you see above that the type of element at position 3 is still integer,
35 # despite the fact that we input a float number.
36 x1[3]="C"
37 print(x1)
38 print(type(x1[3]))
39
40 '''
41 output:
42 ValueError: invalid literal for int() with base 10: 'C'
43 '''
44 # as you see, the string cannot be converted to integer.

```

A side note: If you insert integers, strings, and booleans in a numpy array, you get to have a single type of string. Note that the property of the NumPy array which requires it to hold elements of a single type makes the NumPy faster in calculation compared with list. Also note that if you have a numpy array with booleans and number types (float, integer), numpy will convert the boolean True to 1 and False to 0.

4.2 slicing

Slicing means accessing a subsection of a numpy array. The following examples can represent how it works

```

1 # file name: slicing.py
2
3 import numpy as np
4
5 simpleArray = np.array([19,8,7,1,5,4])
6 firstElement = simpleArray[0]#Numpy arrays start indexing with 0.
7 lastElement = simpleArray[-1]#The second-to-last element would be simpleArray[-2].
8 withoutFirstAndLastElement = simpleArray[1:-1]# x:y means every element from position x (
    including) to position y (excluding). If x is not given, it takes elements from the start.
    If y is not given, it takes elements to the end.
9 myArray = np.array([[1,2,3],[4,5,6],[7,8,9]])
10 firstRow = myArray[0,:]
11 firstcolumn=myArray[:,0]
12 secondColumn = myArray[:,1]
13 oddRowsEvenColumns = myArray[1::2,0::2] # x:y:z means every element from x (including) to y (
    excluding) in steps of z. Like before, if x is not given, it takes elements from the start
    . If y is not given, it takes elements to the end.
14 print("Simple Array: ", simpleArray)
15 print("First Element: ", firstElement)
16 print("Last Element: ", lastElement)
17 print("The simple array without its first and last element: ", withoutFirstAndLastElement)
18 print("A two-dimensional array: ", myArray)
19 print("First Row: ", firstRow)
20 print("First Column: ",firstcolumn)
21 print("Second Column: ", secondColumn)
22 print("All values in odd rows and even columns: ", oddRowsEvenColumns)
23 """
24 Output:
25 Simple Array:  [19  8  7  1  5  4]
26 First Element:  19

```

```

27 Last Element: 4
28 The simple array without its first and last element: [8 7 1 5]
29 A two-dimensional array: [[1 2 3]
30 [4 5 6]
31 [7 8 9]]
32 First Row: [1 2 3]
33 First Column: [1 4 7]
34 Second Column: [2 5 8]
35 All values in odd rows and even columns: [[4 6]]
36
37 """

```

4.3 shape and reshape

Here is we introduce the reshape function, which can change the dimensions of the numpy array. Here is some examples

```

1 #file name: reshape.py
2 import numpy as np
3
4 myarray = np.array([[2,5,6],[3,4,7]])
5 print("myarray: \n",myarray)
6 print("sizeofmyarray:", myarray.shape) # this shows: (gives the number of rows and columns of
   myarray) the dimensionality and the size of each dimension.
7 onecolumnsixrows = myarray.reshape(-1,1)#The size of the new array in each dimension is given.
   If you type -1, that dimension is going to be computed based on the number of elements in
   the original array.
8 print("onecolumnsixrows: \n",onecolumnsixrows)
9 onedimarray = myarray.reshape(-1)#It's also possible to change the dimensionality. This now is
   a one-dimensional array
10 print("onedimarray: \n",onedimarray)
11 threedimarray = myarray.reshape(2,1,-1)#The dimensionality can also be increased. Only one
   parameter can be -1, because otherwise it could not be inferred.
12 # the number of numbers inside parentheses determines the dimensionality of the array, the -1
   will automatically provide the adequate number,
13 # and the multiplication of the numbers (excluding -1, means including any proper
   multiplication) should match to the numbers of the elements of the array.
14 print("threedimarray: \n",threedimarray)
15 """
16 output:
17 myarray:
18 [[2 5 6]
19 [3 4 7]]
20 sizeofmyarray: (2, 3)
21 onecolumnsixrows:
22 [[2]
23 [5]
24 [6]
25 [3]
26 [4]
27 [7]]
28 onedimarray:
29 [2 5 6 3 4 7]
30 threedimarray:
31 [[[2 5 6]]
32
33 [[3 4 7]]]
34 """

```

4.4 linspace

Linspace function is being used to create a line with the amount of discretization that we would like to have. For instance, if we want to have a 20 meters stick and we want to chop it 100 times, each piece would have 0.2 meter length, this can be useful in some problem. This example can be seen in the following code

```

1 # file name: linspace.py
2 # here is an example of linspace
3 import numpy as np
4 x=np.linspace(0,20,100)
5 print(x)
6
7 '''
8 output:
9

```

```

10 [ 0.          0.2020202  0.4040404  0.60606061  0.80808081  1.01010101
11    1.21212121  1.41414141  1.61616162  1.81818182  2.02020202  2.22222222
12    2.42424242  2.62626263  2.82828283  3.03030303  3.23232323  3.43434343
13    3.63636364  3.83838384  4.04040404  4.24242424  4.44444444  4.64646465
14    4.84848485  5.05050505  5.25252525  5.45454545  5.65656566  5.85858586
15    6.06060606  6.26262626  6.46464646  6.66666667  6.86868687  7.07070707
16    7.27272727  7.47474747  7.67676768  7.87878788  8.08080808  8.28282828
17    8.48484848  8.68686869  8.88888889  9.09090909  9.29292929  9.49494949
18    9.6969697   9.8989899   10.1010101  10.3030303  10.50505051  10.70707071
19   10.90909091  11.11111111  11.31313131  11.51515152  11.71717172  11.91919192
20   12.12121212  12.32323232  12.52525253  12.72727273  12.92929293  13.13131313
21   13.33333333  13.53535354  13.73737374  13.93939394  14.14141414  14.34343434
22   14.54545455  14.74747475  14.94949495  15.15151515  15.35353535  15.55555556
23   15.75757576  15.95959596  16.16161616  16.36363636  16.56565657  16.76767677
24   16.96969697  17.17171717  17.37373737  17.57575758  17.77777778  17.97979798
25   18.18181818  18.38383838  18.58585859  18.78787879  18.98989899  19.19191919
26   19.39393939  19.5959596   19.7979798   20.          ]
27
28 '''

```

4.5 arange

Arange is a function that generate arrays with desired space between the elements. It is similar to linspace with the difference that the last number would indicate the size of the steps rather than the number of the steps. It works exactly like range, except that it makes numpy array. Look at [10.4](#) The following examples would elaborate on it better.

```

1 # file name: arange.py
2 # This file contains the same commands as range.py, but using numpy.arange rather than range.
   The output is the same.
3 import numpy as np
4 print('this is the range of a given number 10, the numpy style')
5 x=np.arange(10)
6 for n in x:
7     print(n)
8 print('this is the range of given starting number 1, and ending number 10')
9 y=np.arange(1,10)
10 for n in y:
11     print(n)
12 print('this is the range of given starting number 1, and ending number 10, with defined step
   =2')
13 z=np.arange(1,10,2)
14 for n in z:
15     print(n)
16
17 #However, np.arange has an advantage over range: It produces an array, which we can print and
   work on further (for example slicing, concatenating, adding a value, etc.):
18 print(range(1,10,2))#output: range(1, 10, 2)
19 print(np.arange(1,10,2))#output: [1 3 5 7 9]
20 '''
21 output:
22 this is the range of a given number 10, the numpy style
23 0
24 1
25 2
26 3
27 4
28 5
29 6
30 7
31 8
32 9
33 this is the range of given starting number 1, and ending number 10
34 1
35 2
36 3
37 4
38 5
39 6
40 7
41 8
42 9
43 this is the range of given starting number 1, and ending number 10, with defined step=2
44 1
45 3

```



```

46 5
47 7
48 9
49 range(1, 10, 2)
50 [1 3 5 7 9]
51
52 '''

```

4.5.1 comparing arange and linspace

In this example we can see the difference between arange and linspace more clearly.

```

1 #file name: linspace_and_arange_comparision.py
2
3 #This file makes the difference between linspace and arange clear. These two commands are
  often mixed up.
4 import numpy as np
5
6 print(np.linspace(4,20,3))#The last number indicates the number of substeps.
7 print(np.arange(4,20,3))#The last number indicates the step size between substeps.
8
9 """
10 output:
11 [ 4. 12. 20.]
12 [ 4  7 10 13 16 19]
13 """

```

4.6 meshgrid

Meshgrid can help us to find the coordinates of a certain point in a multi-dimensional space. It takes input arrays that can be seen as coordinate axes. Let's assume we give two arrays. These form the coordinate axes of a two-dimensional space. Meshgrid tells us the coordinates for each position in the resulting two-dimensional array. Each position has (in the two-dimensional case) two coordinates. So meshgrid returns two two-dimensional arrays, one with all the coordinates in the first and the other in the second dimension. A good visualisation is shown under the following [link](#): Here is how it works in a two dimensional arrays which is representing the link's example.

```

1 # file name: meshgrid.py
2 import numpy as np
3 x=[1,2,3,4]
4 y=[5,6,7]
5 XX, YY=np.meshgrid(x,y)
6 print("This is XX values:\n", XX) # \n will take us to the next line
7 print("This is YY values:\n", YY)
8 '''
9 output:
10 This is XX values:
11 [[1 2 3 4]
12  [1 2 3 4]
13  [1 2 3 4]]
14 This is YY values:
15 [[5 5 5 5]
16  [6 6 6 6]
17  [7 7 7 7]]
18
19 '''

```

4.7 random

Here we illustrate how to generate random numbers using numpy

```

1 # file name: random1.py
2
3 # To generate a 1D random number between [0,1):
4 import numpy as np
5 x=np.random.rand(5)
6 print(x)
7 '''
8 output:
9 [0.90907376 0.13799189 0.58644767 0.9362912  0.22169574]
10 '''
11 # To generate a 2D random number:

```

```

12 y=np.random.rand(2,2)
13 print(y)
14
15 '''
16 output:
17 [[0.63012486 0.50949039]
18  [0.83741808 0.63140463]]
19 '''
20 # To generate a nD random number:
21 # myrandom=np.random.rand(1D,2D,3D,...,nD)
22
23
24 # To generate random number with Gaussian distribution
25 # np.random.normal(mean, standard deviation, shape)
26 # one dimension: np.random.normal(mean, standard deviation, 1d)
27 xg=np.random.normal(0.0,1.0,5)
28 print(xg)
29
30 '''
31 output:
32 [-0.16594284  1.50995672 -0.93375937 -0.01684865  0.13507735]
33
34 '''
35 # To generate random Gaussian distribution with shape 2D:
36
37 yg=np.random.normal(0.0,1.0,(2,2))
38 print(yg)
39
40 '''
41 output:
42
43 [[ 0.89220523  0.98818578]
44  [ 0.40821246 -0.7114725 ]]
45
46 '''
47 # To generate Guasssian random number for nD
48 # yg=np.random.normal(mean,standard deviation,(1D,2D,3D,...nD))

```

4.8 hstack()

hstack function can be used to stack data horizontally, for example

```

1 #file name: hstack.py
2 import numpy as np
3 x=np.array([1,2,3])
4 y=np.array([4,5,6])
5 print(np.hstack((x,y)))
6
7 '''
8 output:
9
10 [1 2 3 4 5 6]
11
12 '''

```

4.9 vstack()

```

1 #file name: vstack.py
2 import numpy as np
3 x=np.array([1,2,3])
4 y=np.array([4,5,6])
5 print(np.vstack((x,y)))
6
7 '''
8 output:
9
10 [[1 2 3]
11  [4 5 6]]
12
13 '''
14 x1=np.array([[11],[12],[13]])
15 y1=np.array([[14],[15],[16]])
16 print(np.vstack((x1,y1)))
17
18 '''
19 output:
20
21 [[11 14]
22  [12 15]
23  [13 16]]
24
25 '''

```

```

19 [[11]
20 [12]
21 [13]
22 [14]
23 [15]
24 [16]]
25
26 '''

```

4.10 Conditional Selection

The following example will illustrate how we can select elements of a numpy array that meet a certain condition.

```

1 #file name: numpy_condition
2 import numpy as np
3 x=[1,7,4, 90, 12.4, 56, 10, 2, 0, -5, 22, 34, 65, 10, -4, 17, 2]
4 y=np.array(x)
5 print(y<12)
6 print(y[y<12])
7
8 '''
9 output:
10 [ True  True  True False False False  True  True  True  True False False
11  False  True  True False  True]
12 [ 1.  7.  4. 10.  2.  0. -5. 10. -4.  2.]
13 '''

```

Here you can find another useful code on picking the right data from the second numpy array.

```

1 #file name: statistic1.py
2 list1=["Carl", "Hanna", "Piter", "Ali", "Hassan", "Paul", "Zainab", "Zahra", "Catalina", "Anna",
3        "Julia", "Dina", "Sara", "Lina", "Albert"]
4 list2=[1.80,1.67,1.87,1.55,1.77,1.56,1.78,1.69,1.80,1.58, 1.87, 1.59, 1.65, 1.90, 1.79]
5 import numpy as np
6 height_np=np.array(list2)
7 name_np=np.array(list1)
8 # let's figure out the height of Zahra and Julia:
9 print("Zahra height= ", height_np[name_np=="Zahra"], "Julia height= ", height_np[name_np=="
10      Julia"])
11
12 '''
13 output
14 Zahra height= [1.69] Julia height= [1.87]
15 '''

```

4.11 np.where

this function can be used to apply an operation on elements of numpy array. Here you can find an example where it returns the absolute value of a list of numbers.

```

1 #file name: where.py
2
3 import numpy as np
4 mylist = np.array([9,1,4,2,-4,1,-6,-1,4])
5 mylistAbsolute = np.where(mylist < 0, -mylist, mylist)#all the negative values are turned
6               positive
7 print(mylistAbsolute)
8
9 """
10 output:
11 [9 1 4 2 4 1 6 1 4]
12 """

```

4.12 Statistical Methods

Here is some Numpy function that can be useful dealing with data.

Functions like: mean, median, min, max.

```

1 #file name: statistic.py
2 # here is a list of 15 student's height in cm which includes mistakenly some wrong values, let
3   's see how we can deal with that.
4 student_h=[1.40, 1.42,1.50, 1.67, 1.44, 16.5, 1.38,1.77, 1.44,166, 1.54, 1.30, 1.83, 1.47,
5            1.44]
6 # let's see the mean of their hights:
7 import numpy as np
8 np_student_h=np.array(student_h)
9 print("mean: ", np.mean(np_student_h))

```

```

8 # the output is: 13.473333333333334, it seems something went wrong! but not with the mean,
   maybe a value inserted wrongly.
9 # let's see the median
10 print("median: ", np.median(np_student_h))
11 # output is: 1.47 which makes sense
12
13 # let's see the maximum, and minimum height of this list:
14
15 print("maximum: ", np.max(np_student_h)) # output: maximum: 166.0
16 print("minimum: ", np.min(np_student_h)) # output: minimum: 1.3
17
18 # As you see it seems that the maximum height of the students has been inserted wrongly in cm
   instead of meter, which affected the mean but not the median.

```

4.13 Error with math versus numpy

We have seen that basic arithmetic operations, such as addition or multiplication, can be performed on numpy arrays and apply to every element. Some mathematical operations are implemented as functions in the math package, such as `sqrt()`, `sin()`, `cos()`, and etc. These do not work on numpy arrays. When you give numpy ad parameter to them it throws the following error. Look at the file

```

1 #file name: numpy_and_math_error.py
2 import numpy as np
3 import math
4 myarray = np.array([1,2,3])
5 root = math.sqrt(myarray)
6 print(root)
7 """
8 Output:
9 Traceback (most recent call last):
10   File "numpy_and_math_error.py", line 5, in <module>
11     root = math.sqrt(myarray)
12 TypeError: only size-1 arrays can be converted to Python scalars
13 """

```

We have found one solution for it, and that is using the functions from numpy package not from math package.

```

1 #file name: numpy_math_fix.py
2 import numpy as np
3 import math
4 myarray = np.array([1,2,3])
5 root = np.sqrt(myarray)
6 print(root)
7 """
8 Output:
9 [1.         1.41421356  1.73205081]
10 """

```

5 Matplotlib

Matplotlib is a library that allows to represent data graphically. The plenty of examples provided in [this link](#) can help a lot while trying to plot your favourite one. Also the tutorial [here](#) can be useful. Here we check some of them. Along with simple examples we try to show some of the features that you can add to a plot.

5.1 Plotting List of Data

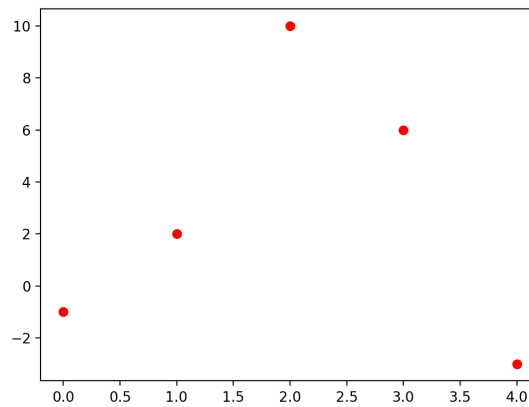
here is a simple example for a one-dimensional list

```

1 # file name: plot_list_data_1.py
2 import matplotlib.pyplot as plt
3 import numpy as np
4 a=np.array([-1,2,10,6,-3])
5 #a=[-1,2,10,6,-3] or you can simply use the list.
6 plt.plot(a, "or") # "o" here stands for point and "r" here stands for the red color in the
   plot.
7 plt.show()
8 plt.clf() # you can use clf() function to clear the previous plot and go for the next (once
   you close the plot)!
9 plt.plot(a, "ob") # 'b' stands for blue.
10 plt.show()

```

Figure 1: one-dimensional plot

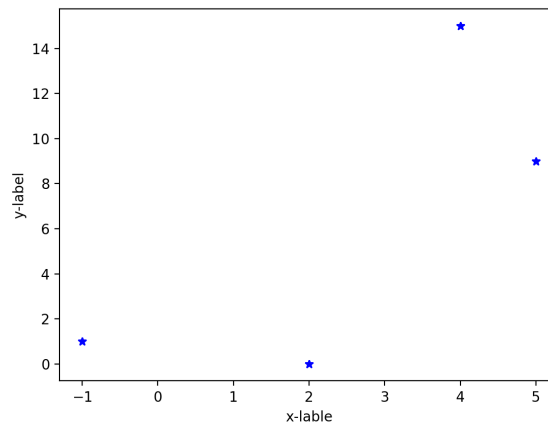


The result looks as follows. As you see the vertical axis is the range of our data and the horizontal axis varies with step one by default. Here is the two-dimensional version of the plot

```
1 # file name: plot_list_data_2.py
2 import matplotlib.pyplot as plt
3 plt.plot([-1, 2, 5, 4], [1, 0, 9, 15], 'b*')
4 plt.xlabel('x-label') # here is how to label the plot
5 plt.ylabel('y-label')
6 plt.show()
```

The result looks as follows. You can obtain the two dimensional plot also with the scattering option. Note that,

Figure 2: two-dimensional plot



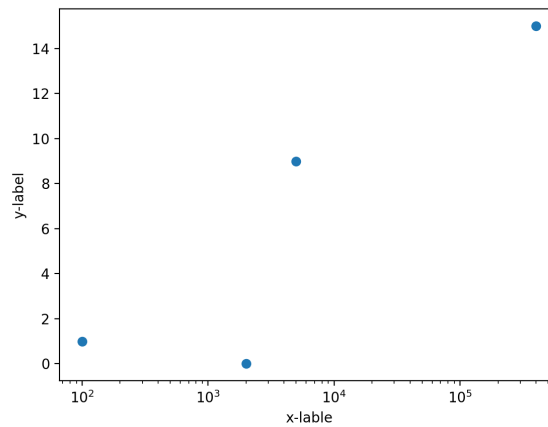
scatter plot is useful to assess the correlation between horizontal and vertical axes's.

```
1 # file name: plot_list_data_3.py
2 import matplotlib.pyplot as plt
3 plt.scatter([100, 2000, 5000, 400000], [1, 0, 9, 15])
4 plt.xscale('log') # if the range of the values is really big, the logarithmic scale is a good
5                   # option to represent the data.
6 plt.xlabel('x-label') # here is how to label the plot
7 plt.ylabel('y-label')
8 plt.show()
```

5.1.1 connected data point list plot

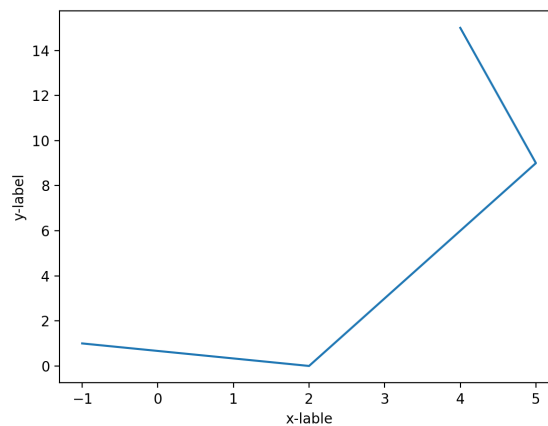
```
1 #file name: plot_list_data_4
2 import matplotlib.pyplot as plt
3 plt.plot([-1, 2, 5, 4], [1, 0, 9, 15]) #if we use plt.plot([-1, 2, 5, 4], [1, 0, 9, 15], 'b*')
4                                     # instead, it will not connect the points in the list.
```

Figure 3: scatter plot



```
4 plt.xlabel('x-label') # here is how to lable the plot
5 plt.ylabel('y-label')
6 plt.show()
```

Figure 4: connected points (line plot)



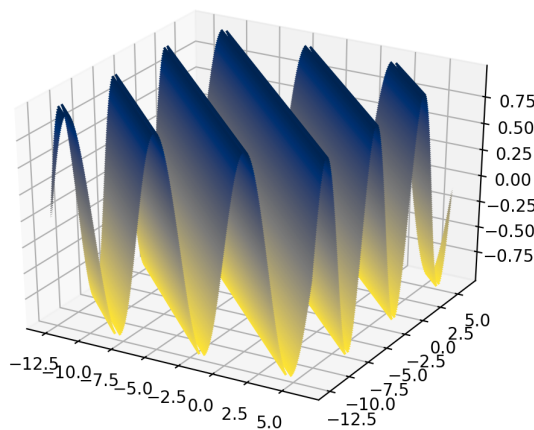
5.2 Plotting Function

Here is a simple example how to plot a $\sin(x)$ function.

```
1 #file name: sin.py
2 from matplotlib import rc # this line is for naming the plot on top of the frame.
3 rc('text', usetex=True)
4 import numpy as np
5 import math
6 import matplotlib.pyplot as plt
7
8
9 x = np.linspace(0.,17.,200) # here is to define the number of points, and the size of the x-
   axis
10 fx = np.sin(x) # here goes the function itself
11
12 plt.plot(x,fx)
13 # in the following line you see how we can write text as well as mathematiacal equations with
   the latex formating.
14 plt.title(
15     r'plot of  $\sin(x)$ ', fontsize=25
16 )
17
18
19 plt.show()
```

5.3 3D plot

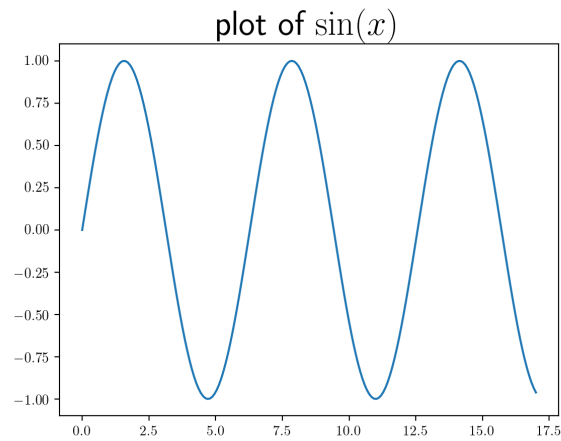
```
1 #file name: 3dplot.py
2 import matplotlib.pyplot as plt
3 import math
4 import numpy as np
5 from mpl_toolkits import mplot3d
6 ax = plt.axes(projection='3d')
7 x=np.arange(-4*np.pi, 2*np.pi,0.1)
8 y=np.arange(-4*np.pi, 2*np.pi,0.1)
9 x, y = np.meshgrid(x, y)
10 z=np.sin(x+y)
11 ax.contour3D(x, y, z,100,cmap='cividis_r' ) # possible other options for "cmap" are listed
12         below.
13 plt.show()
14 """
15 possible other colors:
16 Possible values are: Accent, Accent_r, Blues, Blues_r, BrBG, BrBG_r, BuGn, BuGn_r, BuPu,
17         BuPu_r, CMRmap, CMRmap_r, Dark2, Dark2_r, GnBu, GnBu_r, Greens,
18         Greens_r, Greys, Greys_r, OrRd, OrRd_r, Oranges, Oranges_r, PRGn, PRGn_r, Paired, Paired_r,
19         Pastel1, Pastel1_r, Pastel2, Pastel2_r, PiYG, PiYG_r, PuBu,
20         PuBuGn, PuBuGn_r, PuBu_r, PuOr, PuOr_r, PuRd, PuRd_r, Purples, Purples_r, RdBu, RdBu_r, RdGy,
21         RdGy_r, RdPu, RdPu_r, RdYlBu, RdYlBu_r, RdYlGn, RdYlGn_r,
22         Reds, Reds_r, Set1, Set1_r, Set2, Set2_r, Set3, Set3_r, Spectral, Spectral_r, Wistia, Wistia_r,
23         YlGn, YlGnBu, YlGnBu_r, YlGn_r, YlOrBr, YlOrBr_r, YlOrRd,
24         YlOrRd_r, afmhot, afmhot_r, autumn, autumn_r, binary, binary_r, bone, bone_r, brg, brg_r, bwr,
25         bwr_r, cividis, cividis_r, cool, cool_r, coolwarm,
26         coolwarm_r, copper, copper_r, cubehelix, cubehelix_r, flag, flag_r, gist_earth, gist_earth_r,
27         gist_gray, gist_gray_r, gist_heat, gist_heat_r,
28         gist_ncar, gist_ncar_r, gist_rainbow, gist_rainbow_r, gist_stern, gist_stern_r, gist_yarg,
29         gist_yarg_r, gnuplot, gnuplot2, gnuplot2_r, gnuplot_r,
30         gray, gray_r, hot, hot_r, hsv, hsv_r, inferno, inferno_r, jet, jet_r, magma, magma_r,
31         nipy_spectral, nipy_spectral_r, ocean, ocean_r, pink, pink_r,
32         plasma, plasma_r, prism, prism_r, rainbow, rainbow_r, seismic, seismic_r, spring, spring_r,
33         summer, summer_r, tab10, tab10_r, tab20, tab20_r, tab20b,
34         tab20b_r, tab20c, tab20c_r, terrain, terrain_r, twilight, twilight_r, twilight_shifted,
35         twilight_shifted_r, viridis, viridis_r, winter, winter_r
36 """
```



5.3.1 Extra remark on plotting functions

Note that there are cases that `math` functions would not work properly with `numpy`. In this case we can either use the `numpy` functions or vectorization technique. Let's take a look at this example:

```
1 #plot name: plot_error.py
2 import numpy as np
3 import math
4 import matplotlib.pyplot as plt
5 x = np.linspace(-5.0,5.,1000.0)
```



```

6 f = -x * (2.0 - 8.0 * math.exp(-(x**2)/6.0))
7 plt.plot(x, f)
8 plt.show()
9
10 '''
11 output:
12 TypeError: only size-1 arrays can be converted to Python scalars
13
14 '''

```

One way to simply resolve this is by altering math to numpy.

```

1 #plot name: plot_fix.py
2 import numpy as np
3 import matplotlib.pyplot as plt
4 x = np.linspace(-5.0,5.,1000.0)
5 f = -x * (2.0 - 8.0 * np.exp(-(x**2)/6.0))
6 plt.plot(x, f)
7 plt.show()

```

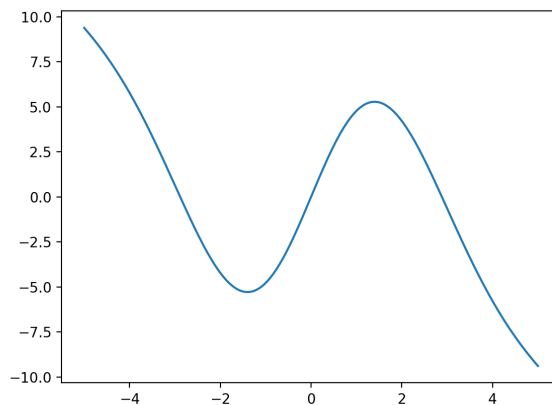
Another way is to use the vectorize function, see example

```

1 #file name: vectorisation.py
2 import numpy as np
3 import matplotlib.pyplot as plt
4 x = np.linspace(-5.0,5.,1000.0)
5 def f(x):
6     return -x * (2.0 - 8.0 * np.exp(-(x**2)/6.0))
7 f2= np.vectorize(f)
8 plt.plot(x, f2(x))
9 plt.show()

```

Figure 5: vectorisation

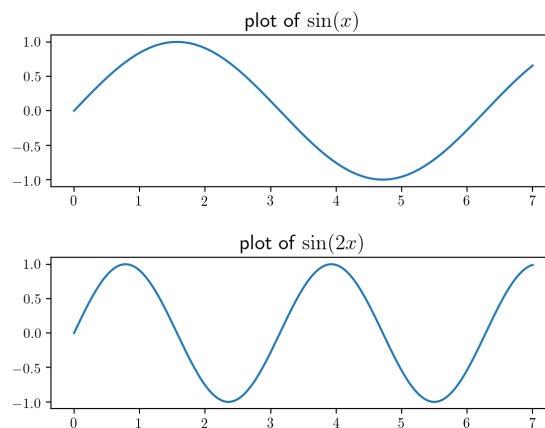


5.4 subplot

here is an example where we can learn how to make several plot in the same figure.

```
1 #file name: subplot.py
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from matplotlib import rc
5 rc('text', usetex=True)
6
7 x = np.linspace(0.,7.,10000)
8
9 fx = np.sin(x)
10 fy = np.sin(2*x)
11
12 # the first subplot:
13
14 plt.subplot(2,1,1) # this line would tell how our subplot should look like: in this case it
15 # has two rows and one coloum.
16 # the last parameter refers to the plot number which in principle starts from one, meaning
17 # that it counts subplots from left to right
18 # and from top to the bottom.
19 plt.plot(x,fx)
20 # the plot title of each subplot needs to be below subplot function and before the next one.
21 plt.title(
22     r'plot of  $\sin(x)$ ', fontsize=15
23 )
24
25 # the second subplot:
26
27 plt.subplot(2,1,2)
28 plt.plot(x,fy)
29 plt.title(
30     r'plot of  $\sin(2x)$ ', fontsize=15
31 )
32
33 plt.show() # this line is needed to show the plot at the end.
```

Figure 6: here is an example of subplot



5.5 Histogram

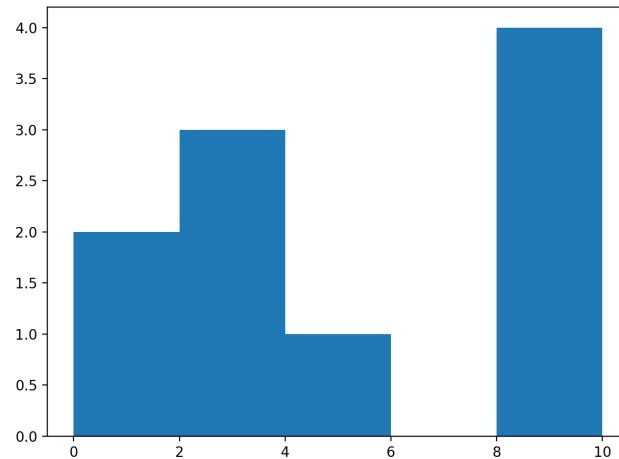
Here you can see, one of the simplest example for making a histogram of the data. A histogram shows how data is distributed - that means how many data points appear with values in certain ranges. The hist-function divides the difference between the highest and the lowest value into bins of equal size. You can specify the number of bins or set it to 'auto' for an automatic decision of how many bins should be made. To estimate the distribution of the data from the histogram, it is recommended to have roughly \sqrt{n} bins, where n is the number of data points. In the exceptional case that a data value is exactly on the boundary of a bin, it will be counted for the next, not the previous bin, except if it is the largest value, which is counted towards the previous bin. In the example given in this part, you can check how the values are distributed in the boundaries of ranges. In our example, there are 5 bins, each of them having the size 2. So there is a bin for the range 6 to 8 and another one for the range 8 to 10.

```

1 #name of file: histogram.py
2 import matplotlib.pyplot as plt
3 x=[1,3,3,8,2,9,10,0,8,5]
4 fig, axs = plt.subplots(1, 1, sharey=True, tight_layout=True)
5 #axs.hist(x, bins=5)
6 axs.hist(x, bins='auto')
7 plt.show()

```

Figure 7: here is an example of histogram



5.6 Customisation

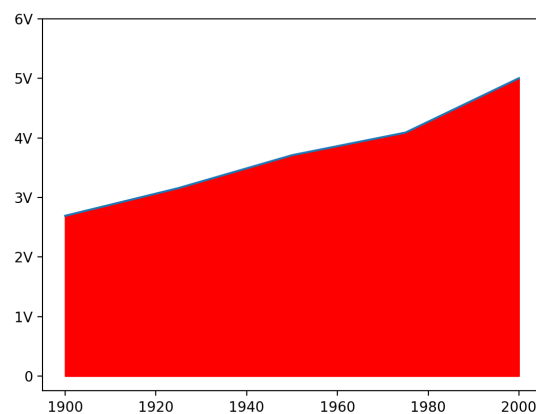
In the following plot we see, how fill_between, and yticks/xticks functions work.

```

1 # plot name: fill.py
2 import matplotlib.pyplot as plt
3 x=[1900,1925, 1950, 1975, 2000]
4 y=[13.46, 15.79, 18.55, 20.45, 25.02]
5 plt.plot(x,y)
6 plt.fill_between(x,y,0,color="red")
7 plt.yticks([0, 5, 10, 15, 20, 25, 30], ['0', '1V', '2V', '3V', '4V', '5V', '6V'])
8 # similarly you can use xticks() function to rescale and rename the steps on the x axes.
9 plt.show()

```

Figure 8: fill_between, and yticks



Here is an example for the scatter plot with the size of the data points.

```

1 # file name: scatter_size.py
2 import matplotlib.pyplot as plt
3 import numpy as np
4 x=np.array([4.1,5,10,6,-3])
5 y=np.array([4,5.1,14,7,3])

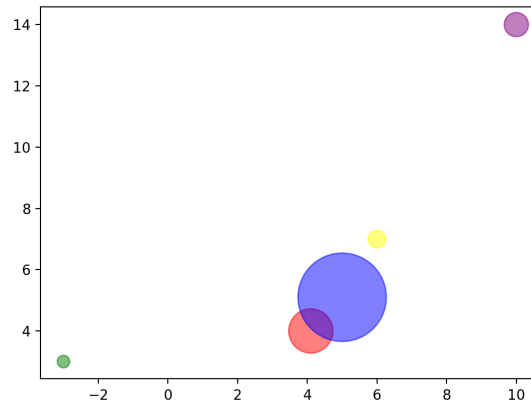
```

```

6  oursize=np.array([100,400,30,16,8])
7  ourcolor=['red','blue', 'purple','yellow','green']
8  plt.scatter(x,y,s=10*oursize,c=ourcolor, alpha=0.5) # alpha changes the opacity of the colors,
9  #it varies from zero to one. # "s" referes to the size, and can also be written "size", "c"
   refers to color and
10 # can be written "color" as well.
11 plt.show()

```

Figure 9: scatter plot with size and colors



5.7 Legends

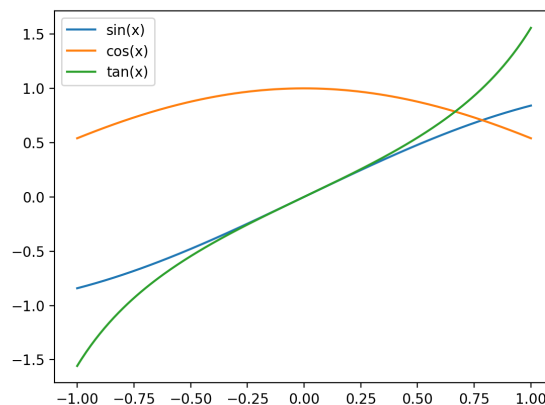
Here is an example on how to make a legend for a plot.

```

1  # file name: legend.py
2  import numpy as np
3  import math
4  import matplotlib.pyplot as plt
5  x = np.linspace(-1,1,200)
6  fig, ax = plt.subplots()
7  y1=np.sin(x)
8  y2=np.cos(x)
9  y3=np.tan(x)
10 ax.plot(x,y1,label='sin(x)')
11 ax.plot(x,y2,label='cos(x)')
12 ax.plot(x,y3,label='tan(x)')
13
14 ax.legend()
15 plt.show()

```

Figure 10: Legend



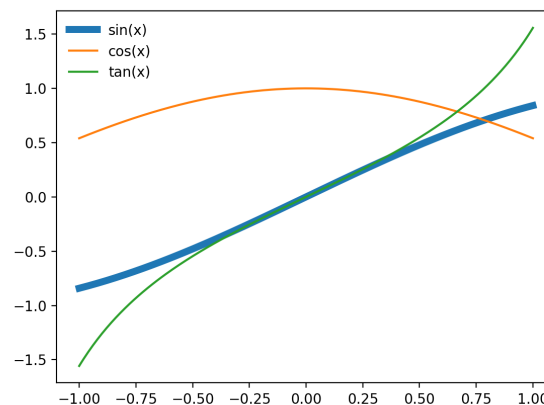
If we like to have no frame for the legend bar we can use the command `plt.legend(frameon=False)`.

```

1 # file name: legendframeless.py
2 import numpy as np
3 import math
4 import matplotlib.pyplot as plt
5 x = np.linspace(-1,1,200)
6 fig, ax = plt.subplots()
7 y1=np.sin(x)
8 y2=np.cos(x)
9 y3=np.tan(x)
10 ax.plot(x,y1,label='sin(x)', linewidth=5) # linewidth determines the width of a line.
11 ax.plot(x,y2,label='cos(x)')
12 ax.plot(x,y3,label='tan(x)')
13
14 ax.legend(frameon=False)
15 plt.show()

```

Figure 11: Frameless Legend



Matplotlib is not the only package for data visualisation in python. For further information, you might take a look at [ggplot](#) package.

5.8 Colors

There are several ways to pick colours for a plot. There are some default colours that we can use right away. They are: ['blue', 'green', 'red', 'cyan', 'magenta', 'yellow', 'black', 'white'], and they are referred as ['b', 'g', 'r', 'c', 'm', 'y', 'k', 'w']. One other option is to use the [xkcd](#) package. These are 954 most common RGB monitor colours. To see the list of colours available see for instance [here](#).

Another option is to specify the colour using RGB code. In the RGB code, the colours Red, Green and Blue are specified each on a scale between black and white. Matplotlib allows to specify their brightness between 0 and 1, or between 0 and 255 in the hexadecimal system ². Option 1: color=(0.5,1.0,0.0) Option 2: color='#48a7f9'

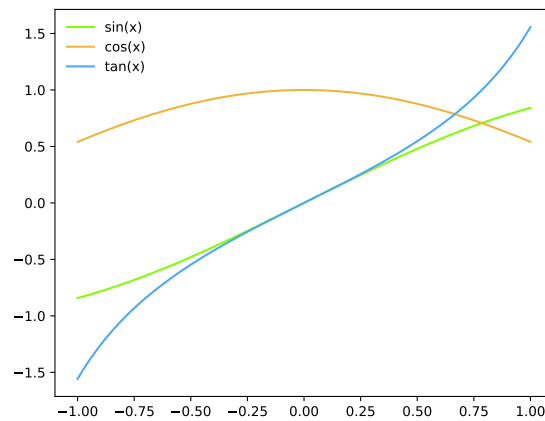
```

1 # file name: all_colour.py
2 import numpy as np
3 import math
4 import matplotlib.pyplot as plt
5 x = np.linspace(-1,1,200)
6 fig, ax = plt.subplots()
7 y1=np.sin(x)
8 y2=np.cos(x)
9 y3=np.tan(x)
10 ax.plot(x,y1,label='sin(x)',color=(0.5,1.0,0.0))
11 ax.plot(x,y2,label='cos(x)',color='xkcd:macaroni and cheese')
12 ax.plot(x,y3,label='tan(x)',color='#48a7f9')
13
14 ax.legend(frameon=False)
15 plt.show()

```

²for instance: '#48a7f9' corresponds to (48, a7, f9)=($4 \times 16^1 + 8 \times 16^0$, $10 \times 16^1 + 7 \times 16^0$, $15 \times 16^1 + 9 \times 16^0$)=(72, 167, 249).

Figure 12: Colours

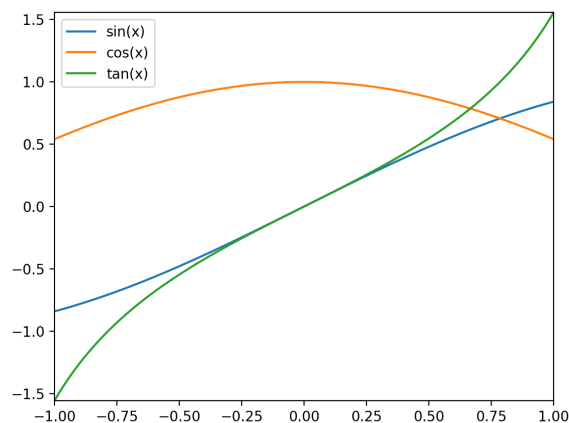


5.9 Margins

If we want to make the frame fit to the data, then we can use the command `plt.margins(0)`. This is the example of the legend 5.7.

```
1 # file name: margin.py
2 import numpy as np
3 import math
4 import matplotlib.pyplot as plt
5 x = np.linspace(-2,2,400)
6 fig, ax = plt.subplots()
7 y1=np.sin(x)
8 y2=np.cos(x)
9 y3=np.tan(x)
10 ax.plot(x,y1,label='sin(x)')
11 ax.plot(x,y2,label='cos(x)')
12 ax.plot(x,y3,label='tan(x)')
13 plt.margins(0) # this makes the frame fit right to the data.
14 ax.legend()
15 plt.show()
```

Figure 13: Margin

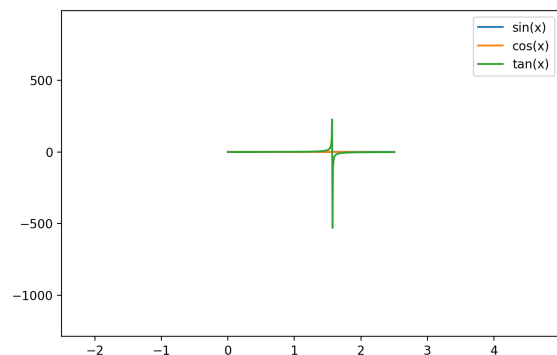


The parameter given to the `plt.margin` function tells the size of the margin between the extreme points (the left-most, right-most, lowest and highest points) and the border of the plot. The value is relative to the distance between opposite extreme points. That means: If our data starts at the x-position of 0 and ends at the x-position of 2.5, then margin parameter of 1 would lead to the plot starting at $x_{min} - margin_x * (x_{max} - x_{min}) = -2.5$ and ending at $x_{max} - margin_x * (x_{max} - x_{min}) = 5$. The same principle applies for the y-axis. The function can be called with one parameter that applies to both the x- and y-axis, or with two parameters that may differ

for x- and y-axis. Example:

```
1 # file name: margin1.py
2 import numpy as np
3 import math
4 import matplotlib.pyplot as plt
5 x = np.linspace(0,2.5,400)
6 fig, ax = plt.subplots()
7 y1=np.sin(x)
8 y2=np.cos(x)
9 y3=np.tan(x)
10 ax.plot(x,y1,label='sin(x)')
11 ax.plot(x,y2,label='cos(x)')
12 ax.plot(x,y3,label='tan(x)')
13 plt.margins(1)
14 ax.legend()
15 plt.show()
```

Figure 14: Margin



6 Pandas

Pandas is a high level data manipulation tool. Pandas has two notable data structure ³, the data frame and series. Unlike NumPy array, Pandas data frames can hold data of different types. Here we give an example to read the data from a file and store it in a pandas data frame. Here is the data for coronavirus report of 2020-05-15 of some countries.

```
1 ,"Country","Cases","Dead","Recovered"
2 "IR","Iran",116635,6902,91836
3 "CN","China",82933,4633,78209
4 "IT","Italy",223885,31610,120205
5 "DE","Germany",175223,7933,151700
```

and here is the code for reading the data with pandas.

```
1 #file name: corona_cases.py
2 import pandas as pd
3 data = pd.read_csv("2020-05-15_Corona_Cases.csv")
4 print(data)
5 '''
6 output:
7
8   Unnamed: 0  Country  Cases  Dead  Recovered
9  0         IR    Iran  116635  6902    91836
10 1         CN    China  82933  4633    78209
11 2         IT    Italy  223885  31610   120205
12 3         DE  Germany  175223   7933   151700
13
14 '''
```

As you noticed the row labels, the country codes that we want to use as row indices, are seen as a column in their own right. To solve this, we use `index_col=0`. This will make the zeroth column to be considered as the indexing column. Alternatively one can set the name of the column as string.

³some data type that stores data in structured way, for example a NumPy array

```

1 #file name: corona_cases_2.py
2 import pandas as pd
3 data = pd.read_csv("2020-05-15_Corona_Cases.csv", index_col=0)
4 print(data)
5 '''
6 output:
7
8      Country  Cases  Dead  Recovered
9 IR      Iran  116635   6902     91836
10 CN     China   82933   4633     78209
11 IT     Italy  223885  31610    120205
12 DE  Germany  175223   7933    151700
13
14 '''

```

6.1 Selection

One of the powerful thing that we can do with pandas is accessing the column easily. Here is an example, we use the previous example, and we want to get the column for **Cases**. Alternatively, We can also use the dot notation: `data.Recovered`

```

1 # file name: pandas_select.py
2 import pandas as pd
3 data = pd.read_csv("2020-05-15_Corona_Cases.csv", index_col=0)
4 print(data["Cases"])
5 '''
6 output
7
8 IR      116635
9 CN      82933
10 IT     223885
11 DE     175223
12 Name: Cases, dtype: int64
13
14 '''
15 print(data.Cases) #alternative way to access a column
16
17 '''
18 output
19
20 IR      116635
21 CN      82933
22 IT     223885
23 DE     175223
24 Name: Cases, dtype: int64
25
26 '''

```

This way we get a pandas series. If we want to select multiple columns we need to use pandas dataframe structure, example follows

```

1 #file name: pandas_multiple_select.py
2 import pandas as pd
3 data = pd.read_csv("2020-05-15_Corona_Cases.csv", index_col=0)
4
5 # select single column with Pandas dataframe
6
7 print(data[["Cases"]])
8 '''
9 output:
10      Cases
11 IR  116635
12 CN   82933
13 IT  223885
14 DE  175223
15
16 '''
17 # Select multiple columns with Pandas dataframe
18 print(data[["Cases", "Recovered"]])
19
20 '''
21 output
22
23      Cases  Recovered
24 IR  116635     91836

```

```

25 CN      82933      78209
26 IT     223885     120205
27 DE     175223     151700
28
29 '''

```

To access the row we need to use `loc` in the way described here.

```

1 # file name: row_pandas.py
2 import pandas as pd
3 data = pd.read_csv("2020-05-15_Corona_Cases.csv", index_col=0)
4 print(data.loc["DE"])
5 '''
6 output
7
8 Country      Germany
9 Cases        175223
10 Dead         7933
11 Recovered    151700
12 Name: DE, dtype: object
13
14 '''

```

To access one value you can use `loc` again as follows.

```

1 #file name: element_pandas.py
2 import pandas as pd
3 data = pd.read_csv("2020-05-15_Corona_Cases.csv", index_col=0)
4 print("One way:", data.loc["DE", "Recovered"])
5 # alternatively:
6 print("The other way: ", data["Recovered"].loc["DE"])
7 # or
8 print("Or: ", data.loc["DE"]["Recovered"])
9 '''
10 One way: 151700
11 The other way: 151700
12 Or: 151700
13 '''

```

6.1.1 Conditional Selection

In order to figure out the index of a row, we can select it by a condition. The following example can illustrate that. There we want to know the **Trieste** province row index. To do so we need to look up in the column which has the name of the provinces.

```

1 #file name: find_triESTE.py
2 import pandas as pd
3 data=pd.read_csv("dpc-covid19-ita-province-latest.csv")
4 trieste_row=data[data.denominazione_provincia=="Trieste"]
5 print(trieste_row)
6 '''
7 output:
8
9      data stato  codice_regione  denominazione_regione  ...      long
10     totale_casi note_it  note_en
11 34  2020-05-23T17:00:00  ITA          6  Friuli Venezia Giulia  ...  13.768136
12     1372      NaN      NaN
13
14 [1 rows x 12 columns]
15 '''
16 # to find out the totale_casi of the trieste we can do the following:
17 print(trieste_row[["totale_casi"]])
18 '''
19 output:
20
21     totale_casi
22 34          1372
23 '''

```

6.2 Column manipulation

We can also add a column as well as adding a column based on the other columns. Here is an example:

```

1 # file name: pandas_column.py
2 import pandas as pd
3 data = pd.read_csv("2020-05-15_Corona_Cases.csv", index_col=0)

```



```

4 data["Serious critical"]=[2294, 8, 762, 1166]
5 print(data)
6 '''
7 output:
8
9      Country    Cases    Dead    Recovered    Serious critical
10 IR      Iran    116635    6902      91836          2294
11 CN      China    82933    4633      78209           8
12 IT      Italy    223885    31610    120205          762
13 DE    Germany    175223    7933     151700         1166
14
15 '''
16 data["Active cases"]=data["Cases"]-(data["Dead"]+data["Recovered"])
17 print(data)
18
19 '''
20 output:
21
22      Country    Cases    Dead    Recovered    Serious critical
23 IR      Iran    116635    6902      91836          2294
24 CN      China    82933    4633      78209           8
25 IT      Italy    223885    31610    120205          762
26 DE    Germany    175223    7933     151700         1166
27      Country    Cases    Dead    Recovered    Serious critical    Active cases
28 IR      Iran    116635    6902      91836          2294          17897
29 CN      China    82933    4633      78209           8           91
30 IT      Italy    223885    31610    120205          762         72070
31 DE    Germany    175223    7933     151700         1166         15590
32 '''

```

Since pandas is based on numpy, you can treat column and rows as a numpy array and we can apply arithmetic operations on them.

7 Scikit-Learn

Machine Learning is the study of algorithms that improve their estimate of structures in data by learning from data. It includes three major branches:

- Supervised Learning, solving the task of classification
- Unsupervised Learning, solving the task of clustering
- Reinforcement Learning, solving a step-wise optimisation task

There are also sub-branches and hybrid approaches such as Semi-Supervised Learning or Self-Supervised Learning.

Scikit-Learn is a python library that implements important steps in the machine learning workflow. This includes not only the machine learning algorithms themselves, but also other important steps, such as the preprocessing of the data.

Let us have a closer look at Supervised Learning. A simple example consists of three steps:

- Splitting the data into training and test sets
- Fitting the classifier to the data
- Evaluating the quality of the classifier for the data we are dealing with

7.1 Supervised Learning

In supervised learning the machine tries to learn from the given data. Let's imagine we are the machine and we want to learn from a simple dataset. For that let's consider some data from weather forecast. Note that this example is to simplify the concept of supervised learning and has nothing to do with reality. We are given data for temperature and wind speed for three consecutive days. Temperature and wind speed are the *attributes* of the day. Each day belongs to one of two *classes*: Rain or no rain.

Day	Temperature	Wind speed	Rain
Monday	20	5	Yes
Tuesday	25	4	No
Wednesday	27	3	?

We want to know whether it will rain on Wednesday. One idea is that we find the most similar day and predict based on that. To find out the closest case to Wednesday is to simply find out the distance in sort of vector space, where temperature is one axis and wind is another axis. Therefore we have

$$D(Wed - Tue) = \sqrt{(27 - 25)^2 + (3 - 4)^2} = \sqrt{5} = 2.2$$

$$D(Wed - Mon) = \sqrt{(27 - 20)^2 + (3 - 5)^2} = \sqrt{53} = 7.3 \quad (1)$$

As you see the closest condition to Wednesday is on Tuesday. Therefore, just a naive prediction is that on Wednesday it might not rain. This form of classification is called “k-nearest-neighbours”-classifier, or “kNN

” for short. k is a parameter that tells how many nearest neighbours the algorithm should look at. What we have described is a 1-NN (for 1-nearest-neighbour) classifier, so it only looks at the single nearest neighbouring data point. It is also possible to look at, for example, the 15 nearest data points and predict the class that most of the 15 neighbours belong to. This would be a 15-NN classifier.

7.1.1 kNN in Scikit-Learn

Scikit-learn has implemented the kNN classifier. In order to see how well the kNN classifier classifies the data, we would make an experiment. We take the data in which we already know their appropriate classes. We then split the data into two different distinct sets. One called training, and the other called testing. In general the fraction of the data which goes to the training part is larger than the testing. In the training part, we give both the data for attribute space and their classes to the machine. We aim to teach the machine how attribute space is associated to their classes. In the testing part, we try to see if the machine actually learnt enough from the training part. We give the data of attribute values of the testing set, but not their classes. Then we ask the machine to predict their classes. As we already know their appropriate classes, we can find out how close the prediction was with their actual class. This way, we can learn how much our kNN classifier is reliable. In the following code we first split the data into training and test set, then make the classifier learn on the training set, and then compare the prediction of the classifier on the test set with the actual classes in the test set. We used example data which was generated in a supervised learning approach for finding appropriate behaviour of robot swarms. (swarm behaviour)

```

1 #file name: classifier.py
2 import pandas as pd
3 from sklearn.model_selection import train_test_split
4 import numpy as np
5 from sklearn.neighbors import KNeighborsClassifier
6 # in the following we read the data in:
7 df=pd.read_csv("Tunnel_1_4Linear8Sensors9ClassesCappedRange8Fast_10.txt", sep='\t') # sep='\t'
8   recognises the tab between each data value as separator (rather than a comma).
9 # below the split between training and testing data happens:
10 #(train_X is the attributes of the training set, val_X is the attributes of the test set,
11   train_y is the classes of the training set, and val_y is the classes of test set)
12 train_X, val_X, train_y, val_y = train_test_split(df.loc[:,df.columns != "Class"], df.Class,
13   random_state = 0)
14 # creating the classifier objects and considers 5 neighboring data for each data point.
15 neigh = KNeighborsClassifier(n_neighbors=5)
16 # the actual training happens here as it teaches to the machine the classes associated to the
17   attributes.
18 neigh.fit(train_X, train_y)
19 # here machine predicts the classes of those data points which it was not trained on.
20 predictions = neigh.predict(val_X)
21 # here is the accuracy of the machine's prediction.
22 accuracy = sum(predictions == val_y)/len(predictions)
23 print("accuracy: ", accuracy)
24 '''
25 output:
26 [[1 2]
27  [3 4]]
28 [[1 2]
29  [3 4]]
30 accuracy:  0.9329923273657289
31 '''
32 importing the following package
33 from sklearn.model_selection import train_test_split
34 leads to the following output:
35 [[1 2]
36  [3 4]]
37 [[1 2]
38  [3 4]]
39 we are not sure about this output but maybe they haven't used the "main condition", so it
40   might be the output of the test code of train_test_split.

```

7.2 Limitations of supervised learning

In general, the more data we have the more reliable the prediction becomes. However, there are a few issues with both the data and the classifier that can cause the learning to underperform. We mention some of them in the following.

7.2.1 Insufficient coverage of the attribute space

It is important that the data contains samples for most of the attribute space. For example, if the classifier has never seen a day with more than 35 degrees and more than 75 km/h wind speed, then it cannot learn whether it is more likely to rain or not on such a day. If it needs to classify such a day, it may well not be accurate, even if it has seen a lot of colder or less windy days.

7.2.2 Biased Data

In the process of training the machine, there might be some occasion that the machine gets biased based on the data given. For example, let's consider the case where a satellite takes a lot of photos from giraffes in savanna and train the machine that they belongs to them. However, it turns out that machine couldn't distinguish them later on. The problem was that the photos that previously shown to machine as giraffe were taken in a cloudy day. The machine has identified giraffes by the weather not the features of the giraffes. When they took other photos of the giraffes in a sunny day, machine was not able to recognise them well. We call this kind of issues as biased data.

Let's give another example for biased data: Assume we have only the body height of people and we want to predict whether the people are men or women. Our training data might be biased if, for example, the men happened to be Japanese and the women South Sudanese, or if the men were measured in the evening (when people are smaller) and the women in the morning. In those cases, the bias is probably big enough to completely distort our predictions, so that they do not generalize to people from different countries, or to measurements taken over the course of the day.

7.2.3 Lack of Information

The data might not contain the information that is necessary to predict the class. For example, the relation from temperature and wind to rainfall is weak. Imagine there are many days where the temperature is 20 degrees and the wind speed is 5 km/h and it rained, and many others where the temperature and wind speed are the same, but it did not rain. If this is the case, then the classifier cannot achieve good accuracy of prediction, because the data does not contain enough information. Rain does not only depend on temperature and wind speed, but also on the level of clouds, on relative humidity, on air pressure, on records of rain in surrounding areas, etc. All of this information is missing in our data, so the classifier cannot be accurate.

Just as a comparison among these three limitations, i.e insufficient coverage of the attribute space, biased data, and lack of information, we give extra explanation as follows. Let's explain the insufficient coverage of the attribute space 7.2.1 with another simple weather forecast scenario; in general if the wind is more than 200 km/h, it rains. However, we only give training data of wind which is just below 100 km/h. If we ask the machine to classify whether it rains if the wind is 215 km/h, then the machine simply cannot predict this. This kind of limitation is related to the distribution of the attributes. As a comparison, if we have a lack of information (7.2.3), the problem is about the class. Let's say we have multiple days where the wind was "exactly" 30 km/h. In some of them it rained, in others not. This means that from the wind speed alone, we cannot decide whether it will rain or not, at least not surely. We need other attributes in connection with the wind, so we can make more reliable predictions. A case of biased data would be if, in general, it rains when the wind is strong, and it doesn't rain when the wind is weak, but for some reason, we happen to have measured the exceptions to this rule for our training data. For example, we took all the measurements of strong winds in the desert, and all the measurements of weak wind in the rain-forest. If a classifier gets such data only, it will learn that strong winds lead to dry weather and weak winds to rain. So the problem with biased data is an untypical relation between the attributes and the classes in the training data, which is not representative of the actual concept to be learned.

7.2.4 Lack of Complexity of the classifier ("underfitting")

The data might contain the necessary information, but the relation between the attributes and the class might be complex. For example, a simple relation would be that if the temperature is below 25 degrees, then it rains. A complex relation would be that it rains if and only if the temperature is between 19 and 21 degrees and the

wind speed is between 5 and 8 km/h, or if the temperature is between 23 and 25 degrees and the wind speed is not between 2 and 4 km/h, and so on. We know all the information that we need to predict whether it rains or not. Imagine you draw a diagram with temperature on the x-axis and wind-speed on the y-axis, and mark the area where it rains. We say, the class “rain” occupies a certain area in the configuration space (spanned by temperature and wind-speed). In this case, there are three unconnected areas where it rains: One between 19 and 21 degrees and between 5 and 8 km/h. One between 23 and 25 degrees and between 0 and 2 km/h. And finally, one between 23 and 25 degrees and wind-speed above 4 km/h.

Some simple classifiers are unable to identify multiple unconnected regions of the same class in the configuration space. They are only able to draw one line that is supposed to separate the two classes (in case of a two-class-problem). Such classifiers are therefore not complex enough to perform well on our complex weather problem. We say, they are underfitting the data. It means that the decision boundary that the classifier draws and that is supposed to separate the classes is not taking the data into account sufficiently to provide good results.

7.2.5 Over-complexity of the classifier (“overfitting”)

Data conveys a general relationship between attributes and classes. However, the individual data points are often subject to noise, or the classes have some overlap. In our example of the weather forecast, perhaps the measurement equipment has occasional error, or there are certain weather conditions where both rain and not rain is possible. An easier example is when we try to find out whether a person is a man or a woman by only knowing their body height. The body heights are normally distributed. The distributions have, of course, some overlap. That means, if a person is 1.95m, it is more likely to be a man than a woman, but there are occasionally women of that height. A classifier could over-adapt to the data it has seen (we call this “overfitting the data”). If, for example, due to randomness, there is a woman of height 1.95 and there is no man of that exact height, then an overfitting classifier would predict any person of that height to be a woman. Since in the height range around 1.95 there are more men, so a meaningful classifier should generalize and learn that within that range, a person is more likely to be a man.

7.3 Why splitting the data is important

In our example code [7.1.1](#), we split the data into two sets before learning: training and test (or validation). This is very important to detect overfitting. The classifier learns from the training data (we say it “fits the training data”). After learning, we use the test dataset in order to find out how well it learned. We should not do that only on the training dataset, because the classifier might have overfitted to the training data. If a classifier performs well on the training data, but poor on the test data, it means it has over-adapted to the training data and not learned generally enough to also classify data of the same distribution well.

7.4 Unsupervised Learning

Another major branch of machine learning does not involve class labels. Instead, it searches for connections of attributes, patterns and correlations in the data. For instance, considering the previous example of weather forecast, in the unsupervised learning, machine would try to collect the data points in groups that have similar temperatures and wind speeds. It organises them in sets of similar properties. This technique is called clustering. However, there are other techniques that use unsupervised learning such as Encoding and Feature Analysis. They all have a common point that there is no ground truth given.

7.4.1 Encoding

7.4.2 Feature Analysis

8 SciPy

SciPy is the collections of scientific packages, which is applicable for data analysis. It uses NumPy, but has more feature.

9 yt-toolkit

10 Glossary

10.1 Enumerate

This is a built-in function in python which counts or get the indices of items.

```
1 # file name: enumerate.py
2 # this is an example on how enumerate function works
3 mylist=['rose', 'orchid', 'tulip', 'sunflower']
4 for counter, value in enumerate(mylist, 1): # this 1 is to enumerate from one instead of zero.
5     print(counter, value)
6
7 '''
8 output:
9 1 rose
10 2 orchid
11 3 tulip
12 4 sunflower
13
14 '''
```

10.2 Init function

here is an example of init function:

```
1 # file name: Init.py
2 class Person:
3     def __init__(self, name, age):
4         self.name = name
5         self.age = age
6
7 p1 = Person("John", 36)#The __init__ function allows us to specify parameters for the
8     instantiation of p1.
9 print(p1.name)
10 print(p1.age)
```

you can alternatively have the following code which does the same job:

```
1 # file name: NoInit.py
2 class Person:
3     def function(self, name, age):
4         self.name = name
5         self.age = age
6
7
8 p1 = Person()#Since we don't have an __init__-function specified, we cannot give parameters on
9     instantiation*. p1 does not have any attributes (name or age) yet.
10 p1.function("John", 36)#If we want to specify attributes, we need to call an extra function.
11 print(p1.name)
12 print(p1.age)
13 '''
14 * instantiation: In programming, instantiation is the creation of a real instance or
15     particular realization of an abstraction or template such as a class of objects or a
16     computer process.
```

10.3 Main condition

Sometimes we want to run certain codes to test some functions or classes or else in the file, however, we prefer when we import the file which contains the code to some other files that test function doesn't run again. In this case we use the main condition. In the following you can see an example of two files the first one is the file which contains the main condition, and the other one is the file that call the first file to use the class which is defined in the first file.

```
1 # file name: main_file.py
2 # this file is called main_file and contains the main condition
3
4 class Person:
5     def function(self, name, age):
6         self.name = name
7         self.age = age
```

```

8
9
10 if __name__ == "__main__":
11     p1 = Person()#Since we don't have an __init__-function specified, we cannot give parameters
        on instantiation*. p1 does not have any attributes (name or age) yet.
12     p1.function("John", 36)#If we want to specify attributes, we need to call an extra function.
13     print(p1.name)
14     print(p1.age)
15     '''
16     * instantiation: In programming, instantiation is the creation of a real instance or
        particular realization of an abstraction or template such as a class of objects or a
        computer process.
17     '''
18
19     """
20     When we run this file, the output is:
21     John
22     36
23     """

```

the following is the second file where we import the class Person into it, considering that it does not print the test functions.

```

1 # file name: main_example.py
2 # this is the second file which is importing the main_file, the first file
3 from main_file import Person
4
5 professor = Person()
6 professor.function("Mehran", 45)
7
8 print(professor.name)
9 """
10 When we run this file, the output is:
11 "Mehran"
12 """

```

10.4 Range

we can use range function to generate a sequence of numbers, similar to 4.5. There are three ways to do so.

1. given a number
2. given starting and ending number
3. given starting and ending number with defined step

here is the example

```

1 # file name: range.py
2 print('this is the range of a given number 10')
3 x=range(10)
4 for n in x:
5     print(n)
6 print('this is the range of given starting number 1, and ending number 10')
7 y=range(1,10)
8 for n in y:
9     print(n)
10 print('this is the range of given starting number 1, and ending number 10, with defined step
    =2')
11 z=range(1,10,2)
12 for n in z:
13     print(n)
14 '''
15 output:
16 this is the range of a given number 10
17 0
18 1
19 2
20 3
21 4
22 5
23 6
24 7
25 8
26 9
27 this is the range of given starting number 1, and ending number 10

```

```

28 1
29 2
30 3
31 4
32 5
33 6
34 7
35 8
36 9
37 this is the range of given starting number 1, and ending number 10, with defined step=2
38 1
39 3
40 5
41 7
42 9
43 ', '

```

10.5 ravel()

It is a function to flatten the data.

```

1 # file name: ravel.py
2 import numpy as np
3 x=[[1,2],[3,4]]
4 print("Initially it looks:", x)
5 # when it gets flattened:
6 y=np.ravel(x)
7 print("Flattened version looks:", y)
8 '''
9 output:
10 Initially it looks: [[1, 2], [3, 4]]
11 Flattened version looks: [1 2 3 4]
12 '''

```

Another example:

```

1 # file name: ravel2.py
2 import numpy as np
3 x=np.array([[1,2],[3,4]])
4 print("Initially looks: ", x)
5 print("Flattened version looks: ", x.ravel())
6
7 '''
8 output:
9 Initially looks:  [[1 2]
10 [3 4]]
11 Flattened version looks:  [1 2 3 4]
12 '''

```

10.6 Basic Git

It is often important to work in a code with other people. However, working together sometimes cause conflicts. As changes of someone might not be compatible with the changes of the other person. **Git** is a version control system that facilitate group working. Git is particularly suitable for raw text files not for compiled files. Git repository is also useful to store data in a machine readable form. To learn more about it you may check [this link](#). In order to use git command from within a python script, we need to have a package called **git python** installed. First if your operating system doesn't have **git** installed already by default, you need to install **git** first. Find out how to install it [here](#). After that to install **git python** you can use the following command: **pip3 install gitpython**. In order to use it in a code you need to **import git**. The documentation of gitpython can be found [here](#).

11 Exercise

1. Write a code that computes the mean value of a given numpy array.
2. Write a code that takes the number of participants who want to make a ping pong tournament and generates a simple match among them. This requires to make a random matching among participants and makes rounds of winners. You may use the knock-out algorithm for simplicity.
3. Write a code that takes the data of COVID-19 from a github resource, for example [here](#). Plot the daily new cases for a chosen city, like: Trieste, as well as daily new cases.

12 Answer

1. An answer:

```
1 #file name: mean.py
2 import numpy as np # a code which finds the mean of a list:
3 p=np.asarray(eval(input("numbers?")))
4 y=0
5 z=0
6 for x in p:
7     z=z+1
8     y=y+x
9     mean=y/z
10 print(mean)
```

2. A sample answer:

```
1 #file name: match.py
2
3 import numpy as np
4 from numpy.random import choice
5 def tour(b,r):
6     c=b.size//2
7     pair=choice(b, size=(c,2), replace=False)
8     print("*****")
9     print("This is the",r,"round")
10    print(pair)
11    if b.size % 2 ==1:
12        notchosen=b[np.isin(b, pair)==False]
13        print("Number", notchosen, "needs to wait.")
14    winners=np.asarray(eval(input("Who (are) is the winner(s)? ")))
15    if b.size %2==1:
16        if winners.size>1:
17            lucky=np.asarray(choice(winners, size=(notchosen.size)))
18        else:
19            lucky=winners
20        print("Number", notchosen, "needs to play against", lucky)
21        who=np.asarray(eval(input("Who won? ")))
22        winners=np.asarray(np.where(winners==lucky, who, winners))
23    return winners
24 p=input("Number of players: ")
25 a=eval(p)
26 r=1
27 people=np.arange(a)
28 while people.size>=2:
29     people=np.asarray(tour(people,r))
30     r=r+1
31 print("Cheers!")
32
33 '''
34 output for 9 participants:
35
36 Number of players: 9
37 *****
38 This is the 1 round
39 [[6 3]
40  [7 1]
41  [0 8]
42  [2 4]]
43 Number [5] needs to wait.
44 Who (are) is the winner(s)? 6,7,8,2
45 Number [5] needs to play against [6]
46 Who won? 5
47 *****
48 This is the 2 round
49 [[2 8]
50  [5 7]]
51 Who (are) is the winner(s)? 2,7
52 *****
53 This is the 3 round
54 [[7 2]]
55 Who (are) is the winner(s)? 7
56 Cheers!
57
```



```
58
59 ' , ,
```

3. A proposed answer (this answer has been manipulated slightly by removing the peculiar negative values for the daily new cases reported as to represent the data in aesthetic way)

```
1 # daily_cases_triESTE.py
2 import matplotlib.pyplot as plt
3 import glob
4 import pandas as pd
5 import git
6
7 g = git.cmd.Git("./COVID-19") # this is to tell that ./COVID-19 is a git repository, and
8   that the git commands that we apply on "g"
9 #should be applied on that repository.
10 #This only works because we have already git cloned the repository to our local directory
11 .
12 g.pull() # this is how to git pull the new data from the remote repository to our local
13 repository.
14 list1=[]
15 # to read the csv files
16 for i in glob.glob('./COVID-19/dati-province/dpc-covid19-ita-province-2020*'):
17     try: # this "try" and "except" is a check whether the files are all doing well!
18         # (if some file is broken it won't be included in our list and the program continues )
19         data=pd.read_csv(i, index_col="denominazione_provincia", parse_dates=True)
20         store=data.loc["Trieste"][["totale_casi", "data"]]
21         list1+=[store]
22     except:
23         print("some error in file", i, "please fix it!")
24 list1=sorted(list1, key=lambda item: item['data'])
25 dlist=[]
26 clist=[]
27 for item in list1:
28     dlist+=[item["data"]]
29     clist+=[item['totale_casi']]
30 daily_new_cases=[]
31 for x in range(len(clist)-1):
32     daily_new_cases+=[clist[x+1]-clist[x]]
33 # below we get rid of negative values for the new daily cases by setting it to the
34 # previous value of the previous day.
35 # This should have not been done as it basically ruin the actual data. We made it just to
36 # make the data look good for the moment.
37 for n, v in enumerate(daily_new_cases):
38     if v<0:
39         daily_new_cases[n]=daily_new_cases[n-1]
40 nlist=dlist[:10]
41 date=[]
42 for x in nlist:
43     y=x.split("T")
44     s=y[0].split("2020-")
45     date+=[s[1]]
46 plt.bar(x=range(len(dlist)-1), height=daily_new_cases, color='grey')
47 dindex=list(range(len(dlist)))[10:]
48 plt.title("COVID-19, Trieste-2020", fontsize=15)
49 )
50 plt.xlabel('Date: Month-Day')
51 plt.xticks(dindex, date, rotation=25)
52 plt.ylabel("New Cases")
53 plt.show()
```

References

