Python Note

Zainab Nazari, and Carl Stermann-Lücke

Contents

1 Installation, read, write, and run **1**
1.1 Mac . . . 1
1.2 Windows . . . 2
1.3 Linux . . . 2

2 Basics **2**
2.1 Comments . . . 2
2.2 Help . . . 2
2.3 Data Types . . . 2
2.4 Operators . . . 3
2.5 Lists . . . 3
2.5.1 Copying from a List . . . 4
2.5.2 A counter-intuitive difference between update and reassignment . . . 5
2.6 Indentation . . . 5
2.7 Functions . . . 5
2.7.1 Information About the Function . . . 6
2.8 Method . . . 6
2.8.1 Help on method . . . 6
2.9 Output . . . 8
2.10 Input . . . 8
2.11 Conditional Statements . . . 8
2.12 while-loop . . . 9
2.12.1 while-True . . . 9
2.13 for-loop . . . 10
2.13.1 list comprehension . . . 10
2.13.2 nested for-loop . . . 10
2.14 Sets . . . 10
2.14.1 Use-cases of sets . . . 11
2.15 Dictionaries . . . 11
2.15.1 Iterating through dictionaries . . . 11
2.15.2 Dictionary Comprehension . . . 11
2.15.3 Find key for minimum (maximum) value . . . 12
2.16 Zip . . . 12
2.17 Exception handling . . . 12
2.18 Class . . . 13
2.19 Write Data out to Files . . . 14
2.20 Read Data in from Files . . . 14
2.20.1 Read in Several Data files . . . 15
2.21 import . . . 16
2.22 Packages . . . 17

3 Advanced **17**
3.1 ∗args . . . 17
3.2 ∗∗kwargs . . . 18

4 NumPy 18
4.1 numpy arrays 18
4.2 slicing 20
4.3 shape and reshape 20
4.4 linspace 21
4.5 arange 21
4.5.1 comparing arange and linspace 22
4.6 meshgrid 22
4.7 random 23
4.8 hstack() 23
4.9 vstack() 24
4.10 Conditional Selection 24
4.11 np.where 25
4.12 Statistical Methods 25

5 Matplotlib 25
5.1 Plotting List of Data 25
5.1.1 connected data point list plot 26
5.2 Plotting Function 27
5.3 3D plot 27
5.4 subplot 28
5.5 Histogram 29
5.6 Customisation 30
5.7 Legends 30

6 Pandas 31
6.1 Selection 32
6.1.1 Conditional Selection 34
6.2 Column manipulation 34

7 scikit-learn 35

8 SciPy 35

9 yt-toolkit 35

10 Glossary 35
10.1 Enumerate 35
10.2 Init function 35
10.3 Main condition 36
10.4 Range 36
10.5 ravel() 37
10.6 Basic Git 38

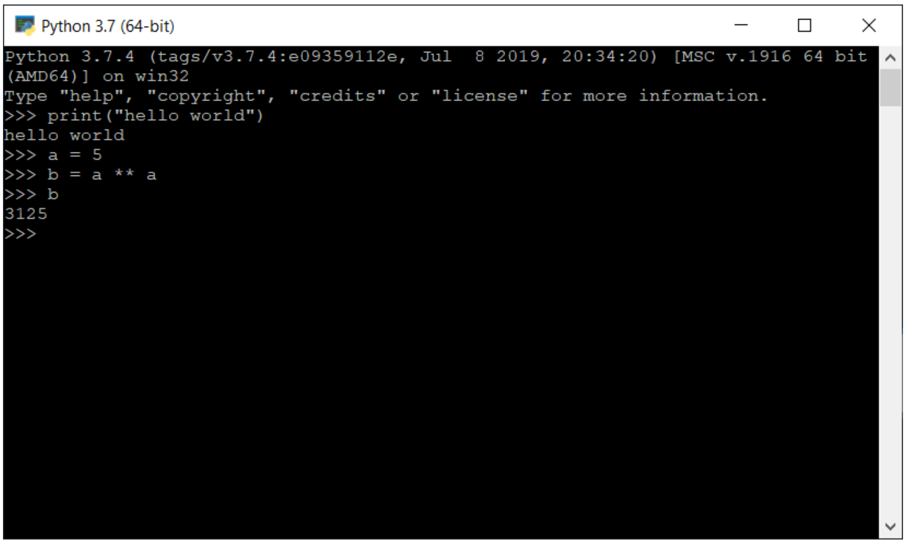
11 Exercise 38

12 Answer 38

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 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:

```
#file name: comment-example.py
#This is a line comment.
myVariable = 3 #This is also a line comment, but the part before the # can still be
    evaluated.
"""
These
lines
are
a
multi-line
comment.
So the line
print(myVariable * 2)
is not evaluated.
"""
print(myVariable)
'''
The output of this code will be:
3
'''
```

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 mathemetical operators. This includes

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 suing the slicing operator. Here is some example of manipulating the data.

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

```python
#file name: list.py
my_list=[1,2,4.4, 7, ["carl", 27]]
#
print("my_list: \n", my_list)
print("my_list[3]: \n", my_list[3])

# deleting an element in the list:

del(my_list[3])
print("my_list after deleting my_list[3]:\n ", my_list)

# accessing the list in the list:
print("my_list[-1][:]: \n", my_list[-1][:])

#replacing element:
my_list[0]="zainab"
print("my_list after replacing my_list[0]=\"zainab\":\n ", my_list)
'''
output:

my_list:
 [1, 2, 4.4, 7, ['carl', 27]]
my_list[3]:
 7
my_list after deleting my_list[3]:
  [1, 2, 4.4, ['carl', 27]]
my_list[-1][:]:
 ['carl', 27]
my_list after replacing my_list[0]="zainab":
  ['zainab', 2, 4.4, ['carl', 27]]

  '''
```

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

```python
#file name: copy_list.py
x=[1,2,3,4,5]
print("primary list x= ", x)
```

```
y=x # copying the refernce x to y!
print("y=", y)
y[0]=0 # making a change to y
print("changed y=", y)
print("after changing y, x=", x)

print("proper way of copying the elements of a list:")

y=list(x) # proper way of copying the elements of a list (way_1)
y=x[:] # proper way of copying the elements of a list (way_2)
print("primary list x= ", x)
# now let's check!
y[0]=-1
print("after changing y, x=", x)
print("after changing y, y=", y)

"""
output:
primary list x=  [1, 2, 3, 4, 5]
y= [1, 2, 3, 4, 5]
changed y= [0, 2, 3, 4, 5]
after changing y, x= [0, 2, 3, 4, 5]
proper way of copying the elements of a list:
primary list x=  [0, 2, 3, 4, 5]
after changing y, x= [0, 2, 3, 4, 5]
after changing y, y= [-1, 2, 3, 4, 5]

"""
```

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.

```
#file name: try.py
from copy import deepcopy
x=[[0,1],2,3]
y=deepcopy(x)
x[0][1]=9

print("x=",x)
print("y=",y)

'''
output:
x= [[0, 9], 2, 3]
y= [[0, 1], 2, 3]

'''
```

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:

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

mynumber = 3
mynumber = mynumber + 2#reassignment
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:

```
#file name: UpdateVsReassignment2.py

a = [1,2,3]
b = a#b and a now refer to the same list...
print("***Update***")
a += [4,5]#...so any changes to one of them affect both
print("a=",a)#[1, 2, 3, 4, 5]
```

```
print("b=",b)#[1, 2, 3, 4, 5]

print("***Assignment***")
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.
print("a=",a)#[1, 2, 3, 4, 5, 6, 7]
print("b=",b)#[1, 2, 3, 4, 5]
"""
output:
***Update***
a= [1, 2, 3, 4, 5]
b= [1, 2, 3, 4, 5]
***Assignment***
a= [1, 2, 3, 4, 5, 6, 7]
b= [1, 2, 3, 4, 5]

"""
```

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:

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

```
# file name: help_func.py
def hi():
  """Ciao is a greeting in Italy"""
  print("Ciao!")

help(hi)
hi()
'''
output:
```

```
Help on function hi in module __main__:

hi()
    Ciao is a greeting in Italy
(END)

'''
```

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 2.18. 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.

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

2.8.1 Help on method

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

```
#file name: method_help.py

#Let's find out all the methods defined in the list class:
help([1,2,3]) # you can also call it by help(list) or help(mylist) when you defined your list
    for example: mylist=[1,2,3]
"""
output:
class list(object)
 |  list() -> new empty list
 |  list(iterable) -> new list initialized from iterable's items
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __delitem__(self, key, /)
 |      Delete self[key].
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __getitem__(...)
 |      x.__getitem__(y) <==> x[y]
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __iadd__(self, value, /)
 |      Implement self+=value.
 |
```

```
 |  __imul__(self, value, /)
 |      Implement self*=value.
 |
 |  __init__(self, /, *args, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __mul__(self, value, /)
 |      Return self*value.n
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  __reversed__(...)
 |      L.__reversed__() -- return a reverse iterator over the list
 |
 |  __rmul__(self, value, /)
 |      Return self*value.
 |
 |  __setitem__(self, key, value, /)
 |      Set self[key] to value.
 |
 |  __sizeof__(...)
 |      L.__sizeof__() -- size of L in memory, in bytes
 |
 |  append(...)
 |      L.append(object) -> None -- append object to end
 |
 |  clear(...)
 |      L.clear() -> None -- remove all items from L
 |
 |  copy(...)
 |      L.copy() -> list -- a shallow copy of L
 |
 |  count(...)
 |      L.count(value) -> integer -- return number of occurrences of value
 |
 |  extend(...)
 |      L.extend(iterable) -> None -- extend list by appending elements from the iterable
 |
 |  index(...)
 |      L.index(value, [start, [stop]]) -> integer -- return first index of value.
 |      Raises ValueError if the value is not present.
 |
 |  insert(...)
 |      L.insert(index, object) -- insert object before index
 |
 |  pop(...)
 |      L.pop([index]) -> item -- remove and return item at index (default last).
 |      Raises IndexError if list is empty or index is out of range.
 |
 |  remove(...)
 |      L.remove(value) -> None -- remove first occurrence of value.
 |      Raises ValueError if the value is not present.
 |
 |  reverse(...)
 |      L.reverse() -- reverse *IN PLACE*
 |
 |  sort(...)
```

```
 |      L.sort(key=None, reverse=False) -> None -- stable sort *IN PLACE*
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __hash__ = None

"""
```

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:

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

2.11 Conditional Statements

This code shows how to write conditional statements:

```python
# file name: if-statement-example.py
number = 24352
if(number % 2 == 0):
  print(number, "is an even number") #This line is only run if the condition in the if-statement is true.
else:
  print(number, "is an odd number") #This line is only run if the condition in the if-statement is false.
```

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

```python
# file name: if-statement-example2.py
number = 24352
if(number == 0):
  print(number, "is 0") #This line is only run if the condition in the if-statement is true.
elif(number % 2 == 0):
  print(number, "is an even number") #This line is only run if the condition in the elif-statement is true and the condition in the if statement (and any possible elif statements before this) is false.
else:
  print(number, "is an odd number") #This line is only run if the conditions in the if and the 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.

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

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

```
# file name: repeat.py
'''
this code is going to be take a number and then type it something like
let's assume the given number is 5
then our code is going to print the result as follow
5
55
555
5555
55555
'''
b=input("what is your number? ")
c=1
while c<=int(b):
    print(b*c)
    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.

```
#file name=sum_product.py
# this code can take numbers (as mush as you give- if not enter the letter 'q' to exit )
# and print the sum and product of them.

s = 0
p = 1

while True: #This is just a regular while loop, but the condition always stays true. The loop
    will continue indefinitely...
    num = input('Number or q to quit: ')
    if num == 'q':
        break #...unless it is broken using the "break"-command. The "break"-command ends the
    while loop.
    num = int(num)
    s += num
    p *= num
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.

```
# file name: for_loop.py
a=[1,4,7,9]
for x in a:
    print(x*2)
'''
the output is as follow:
2
8
14
18
'''
```

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.

```
#file name: list_comprehension.py
#old way:
newList = []
oldList = [1,2,3]
for i in oldList:
  newList += [i/2]
#new way:
newList2 = [i/2 for i in oldList]
print("old way:", newList)
print("new way:", newList2)
"""
output:
[0.5, 1.0, 1.5]
[0.5, 1.0, 1.5]
"""
```

2.13.2 nested for-loop

```
# file name: nested_for_loop.py

firstList = [2,3]
secondList = [4,5,6]
for i in firstList:
    for j in secondList:
        print("(",i,",",j,")")

"""
the output:
( 2 , 4 )
( 2 , 5 )
( 2 , 6 )
( 3 , 4 )
( 3 , 5 )
( 3 , 6 )
"""
```

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

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

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:

```
#file name: dictionary_example.py
ages={"Egypt": 5000, "Carl": 27, "Zainab": 31, "Albert Einstein": 141, "China": 5000}
print(ages["China"])
"""
output:
5000
"""
```

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:

```
#file name: dictionary_iteration.py
ages={"Egypt": 5000, "Carl": 27, "Zainab": 31, "Albert Einstein": 141, "China": 5000}
print("keys:")
for item in ages:
  print(item)

print("values:")
for item in ages:
  print(ages[item])
"""
output:
keys:
Egypt
Carl
Zainab
Albert Einstein
China
values:
5000
27
31
141
5000
"""
```

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:

```
#file name: dictionary_comprehension.py
radius=[1,2,3,4,5]
radiusToArea = {r:(r**2)*3.14 for r in radius}#This is a dictionary with radii of circles as
    keys and areas as values.
print(radiusToArea)
"""
output:
{1: 3.14, 2: 12.56, 3: 28.26, 4: 50.24, 5: 78.5}
"""
```

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:

```
#file name: dictionary_min_max.py
ages={"Egypt": 5000, "Carl": 27, "Zainab": 31, "Albert Einstein": 141, "China": 5000}
print(min(ages))#first key in alphabetical order
print(max(ages))#last key in alphabetical order
print(sorted(ages))#sorts keys alphabetically
print(min(ages, key=ages.get))#key where the value is minimum.
print(max(ages, key=ages.get))#It returns the first key with the maximum value.
print(sorted(ages, key=ages.get))#list of keys, sorted by their values.
"""
output:
Albert Einstein
Zainab
['Albert Einstein', 'Carl', 'China', 'Egypt', 'Zainab']
Carl
Egypt
['Carl', 'Zainab', 'Albert Einstein', 'Egypt', 'China']
"""
```

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:

```
#file name: zip_example.py
number_list = [1, 2, 3]
str_list = ['one', 'two', 'three']
l=list(zip(number_list, str_list))
print(l)
"""
output:
[(1, 'one'), (2, 'two'), (3, 'three')]
"""
#Why is this useful?:
#Assume we want to have a dictionary like this:
#{1:"one", 2:"two", 3:"three"}
print({i[0]:i[1] for i in zip(number_list, str_list)})
"""
output:
{1: 'one', 2: 'two', 3: 'three'}
"""
```

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.

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

Now, we foreseen the error.

```
#file name:except.py
list=[1,3,55,0,9,88,20,50,0,24,44,76,100]
for i in list:
    try:
       print(30/i)
    except ZeroDivisionError:
```

```
        print("Oh, that's bad!")
'''
output:

30.0
10.0
0.5454545454545454
Oh, that's bad!
3.3333333333333335
0.3409090909090909
1.5
0.6
Oh, that's bad!
1.25
0.6818181818181818
0.39473684210526316
0.3

'''
```

2.18 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.

```
# file name: class.py

#We define a blueprint of how a time looks like and what it can do:
class time:
  #Here we define which attributes an object of the class time has and how they are
    initialized:
  hour=1
  minute=1
  #Here we define which functions the class time offers:
  #settime sets the time to a given hour and minute.
  def settime(self,hour,minute):
    self.hour=hour
    self.minute=minute
  #addmin adds a given number of minutes to the time.
  def addmin(self, adm):
    self.hour+=(self.minute+adm)//60
    self.minute=(self.minute+adm)%60
    self.hour=self.hour%24
  #Print time prints the time in a nicely formatted way.
  def printtime(self):
    print(self.hour,'h', self.minute, 'min')

#Here we start using the class.
mytime=time()#We now have an object of the class time which is called mytime.
mytime.settime(19, 20)#This is how we call a function of that object: objectname.functionname(
    parameters).
#The mytime is now 19 hours and 20 minutes.
mytime.addmin(40)
#The mytime is now 20 hours and 0 minutes - because we added 40 minutes.
mytime.printtime()
mytime.addmin(400)
#The time is now 2 hours and 40 minutes, because we added 400 minutes to the previous state of
     mytime, which was 20 hours and 0 minutes.
mytime.printtime()
print(type(mytime))

'''
the output:
20 h 0 min
2 h 40 min
<class '__main__.time'>
'''
```

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.

2.19 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

```
# file name: write.py
'''
here is a simple example to show how we can open a file and write into it the data  that in
    this case, it is an array of 5 digits,  however, the write file can only accept strings,
    therfore we need to convert the integeres to strings.
'''

x=[1,2,3,4,5]
file=open('data.csv','w') # here 'w' means write, note that it also it works with .dat, and .
    txt format
file.write(str(x))
'''
output in the file:
[1 2 3 4 5]

'''
# 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.

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

```
1.000000000000000000e+00 2.000000000000000000e+00 3.000000000000000000e+00
    4.000000000000000000e+00 5.000000000000000000e+00
1.000000000000000000e+00 2.000000000000000000e+00 3.000000000000000000e+00
    4.000000000000000000e+00 5.000000000000000000e+00
```

2.20 Read Data in from Files

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

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

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

```
# name of file: read2.py
# loading data (read data in) with loadtxt
import numpy as np
newarray=np.loadtxt('ourdata.txt')
for x in newarray:
    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.

2.20.1 Read in Several Data files

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

```
# file name: several_read.py
import glob as gl
file_names=gl.glob("dpc-covid19-ita-province-*")
print(file_names)
for name in file_names:
```

```
    print(name)
    print(open(name,"r").read())
'''
part of the output:
['dpc-covid19-ita-province-20200312.csv', 'dpc-covid19-ita-province-latest.csv', 'dpc-covid19-ita-province-20200311.csv']
dpc-covid19-ita-province-20200312.csv
data,stato,codice_regione,denominazione_regione,codice_provincia,denominazione_provincia,sigla_provincia,lat,long,totale_casi,note_it,note_en
2020-03-12T17:00:00,ITA,13,Abruzzo,069,Chieti,CH,42.35103167,14.16754574,20,,
2020-03-12T17:00:00,ITA,13,Abruzzo,066,L'Aquila,AQ,42.35122196,13.39843823,8,,
2020-03-12T17:00:00,ITA,13,Abruzzo,068,Pescara,PE,42.46458398,14.21364822,48,,
2020-03-12T17:00:00,ITA,13,Abruzzo,067,Teramo,TE,42.6589177,13.70439971,8,,
.
.
.
dpc-covid19-ita-province-latest.csv
data,stato,codice_regione,denominazione_regione,codice_provincia,denominazione_provincia,sigla_provincia,lat,long,totale_casi,note_it,note_en
2020-05-23T17:00:00,ITA,13,Abruzzo,069,Chieti,CH,42.35103167,14.16754574,817,,
2020-05-23T17:00:00,ITA,13,Abruzzo,066,L'Aquila,AQ,42.35122196,13.39843823,246,,
2020-05-23T17:00:00,ITA,13,Abruzzo,068,Pescara,PE,42.46458398,14.21364822,1508,,

.
.
.
dpc-covid19-ita-province-20200311.csv
data,stato,codice_regione,denominazione_regione,codice_provincia,denominazione_provincia,sigla_provincia,lat,long,totale_casi,note_it,note_en
2020-03-11T17:00:00,ITA,13,Abruzzo,069,Chieti,CH,42.35103167,14.16754574,9,,
2020-03-11T17:00:00,ITA,13,Abruzzo,066,L'Aquila,AQ,42.35122196,13.39843823,6,,
2020-03-11T17:00:00,ITA,13,Abruzzo,068,Pescara,PE,42.46458398,14.21364822,18,,
.
.
.

'''
```

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.

```
# file name: read_several_pandas.py
import glob
import pandas as pd

for i in glob.glob('dpc-covid19-ita-province-20200*'): # it will take all the files with the prefix before *
    data=pd.read_csv(i)
    print(data)
'''
output:
                    data stato  codice_regione denominazione_regione  ...       long  totale_casi note_it  note_en
0    2020-03-12T17:00:00   ITA              13               Abruzzo  ...  14.167546  20      NaN      NaN
1    2020-03-12T17:00:00   ITA              13               Abruzzo  ...  13.398438  8      NaN      NaN
2    2020-03-12T17:00:00   ITA              13               Abruzzo  ...  14.213648  48      NaN      NaN
3    2020-03-12T17:00:00   ITA              13               Abruzzo  ...  13.704400  8      NaN      NaN
4    2020-03-12T17:00:00   ITA              13               Abruzzo  ...   0.000000  0      NaN      NaN
..                   ...   ...             ...                   ...  ...        ...  ...      ...      ...
123  2020-03-12T17:00:00   ITA               5                Veneto  ...  12.245074  279      NaN      NaN
124  2020-03-12T17:00:00   ITA               5                Veneto  ...  12.338452  205      NaN      NaN
125  2020-03-12T17:00:00   ITA               5                Veneto  ...  10.993527  150      NaN      NaN
126  2020-03-12T17:00:00   ITA               5                Veneto  ...  11.545971  122      NaN      NaN
127  2020-03-12T17:00:00   ITA               5                Veneto  ...   0.000000  128      NaN      NaN
```

```

[128 rows x 12 columns]
                   data stato  codice_regione denominazione_regione  ...       long  totale_casi note_it  note_en
0    2020-03-11T17:00:00   ITA              13               Abruzzo  ...  14.167546            9      NaN      NaN
1    2020-03-11T17:00:00   ITA              13               Abruzzo  ...  13.398438            6      NaN      NaN
2    2020-03-11T17:00:00   ITA              13               Abruzzo  ...  14.213648           18      NaN      NaN
3    2020-03-11T17:00:00   ITA              13               Abruzzo  ...  13.704400            5      NaN      NaN
4    2020-03-11T17:00:00   ITA              13               Abruzzo  ...   0.000000            0      NaN      NaN
..                   ...   ...             ...                   ...  ...        ...          ...      ...      ...
123  2020-03-11T17:00:00   ITA               5                Veneto  ...  12.245074          185      NaN      NaN
124  2020-03-11T17:00:00   ITA               5                Veneto  ...  12.338452          179      NaN      NaN
125  2020-03-11T17:00:00   ITA               5                Veneto  ...  10.993527          110      NaN      NaN
126  2020-03-11T17:00:00   ITA               5                Veneto  ...  11.545971           92      NaN      NaN
127  2020-03-11T17:00:00   ITA               5                Veneto  ...   0.000000           40      NaN      NaN

[128 rows x 12 columns]
'''
```

2.21 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

```python
#file name: func2.py
# in this file we just define a simple function that can get an integer and return it as a power of three.
def  myfun(s):
    return s**3
print(myfun(2))

'''
output:
8

'''
```

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

```python
# file name: import.py
# here we import the function that we have defined in another file called: func2.py
from func2 import myfun
print(myfun(3))
'''
output:
8
27
'''
```

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.

```python
#file name: func3.py
def  myfun(s):
    return s**3
if __name__=="__main__":
    print(myfun(2))
```

[1]a collection of modules can make a library or package

```

'''
output:
8
'''
```

here is the result that we expect.

```
# file name: import2.py
# here we import the function that we have defined in another file called: func2.py
from func3 import myfun
print(myfun(3))
'''
output:
27
'''
```

2.22 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-packeage 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:

```
#file_name: args.py
def test(parameter):
  print(parameter)

test(4)
test([1,2,3])

def test1(*args):
  print(args)

test1(3,8, "Carl", [4,2,3,1])
"""
output:
4
[1, 2, 3]
(3, 8, 'Carl', [4, 2, 3, 1])
"""
```

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:

```
#file_name: kwargs.py
def test(parameter):
  print(parameter)

test(4)
test([1,2,3])

def test1(**kwargs):
  print(kwargs)

test1(blue=3,green=8, red="Carl", pink=[4,2,3,1])
"""
output:
4
[1, 2, 3]
{'blue': 3, 'green': 8, 'red': 'Carl', 'pink': [4, 2, 3, 1]}
"""
```

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

```
# file name: array.py
import numpy as np
x=np.array([[1,2],[3,4]])
print(x)
'''
output:
[[1 2]
 [3 4]]

 '''
```

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

```
# file name: numpy_ex1.py
list1=[1,2,3,4]
list2=[1,2,3,4]
list3=[[1,2,3,4],[1,2,3,4]]
#print("list1*list2= ",list1*list2) # this will give error, the operation of multiplication on
     lists is not defined!
print("list1+list2= ",list1+list2)
print("list3+list1= ",list3+list1)
import numpy as np
numpyarray1=np.array([1,2,3,4])
numpyarray2=np.array([1,2,3,4])
numpyarray3=np.array([[1,2,3,4],[1,2,3,4]])
print("numpyarray1*numpyarray2= ", numpyarray1*numpyarray2)
print("numpyarray1+numpyarray2= ", numpyarray1+numpyarray2)
print("numpyarray3+numpyarray1= ", numpyarray3+numpyarray1)
print("numpyarray3*numpyarray1= ", numpyarray3*numpyarray1)

'''
output:

list1+list2=  [1, 2, 3, 4, 1, 2, 3, 4]
list3+list1=  [[1, 2, 3, 4], [1, 2, 3, 4], 1, 2, 3, 4]
numpyarray1*numpyarray2=  [ 1  4  9 16]
numpyarray1+numpyarray2=  [2 4 6 8]
numpyarray3+numpyarray1=  [[2 4 6 8]
 [2 4 6 8]]
numpyarray3*numpyarray1=  [[ 1  4  9 16]
```

```
 [ 1  4  9 16]]

'''
```

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.

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

<class 'numpy.int64'> <class 'numpy.str_'>

'''
# now we make a change in x2
x2[3]=4
print(x2)
print(type(x2[3]))

'''
output:
['1' '2' '3' '4']
<class 'numpy.str_'>
'''
# you see even though we have changed the last element in x2 to integer,
# the type still remains as string.

x1[3]=7.7
print(x1)
print(type(x1[3]))

'''
output:
[1 2 3 7]
<class 'numpy.int64'>
'''
# you see above that the type of element at position 3 is still integer,
# despite the fact that we input a float number.
x1[3]="C"
print(x1)
print(type(x1[3]))

'''
output:
ValueError: invalid literal for int() with base 10: 'C'
'''
# 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

```python
# file name: slicing.py

import numpy as np

simpleArray = np.array([19,8,7,1,5,4])
firstElement = simpleArray[0]#Numpy arrays start indexing with 0.
lastElement = simpleArray[-1]#The second-to-last element would be simpleArray[-2].
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.
```

```
myArray = np.array([[1,2,3],[4,5,6],[7,8,9]])
firstRow = myArray[0,:]
firstcolumn=myArray[:,0]
secondColumn = myArray[:,1]
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.
print("Simple Array: ", simpleArray)
print("First Element: ", firstElement)
print("Last Element: ", lastElement)
print("The simple array without its first and last element: ", withoutFirstAndLastElement)
print("A two-dimensional array: ", myArray)
print("First Row: ", firstRow)
print("First Column: ",firstcolumn)
print("Second Column: ", secondColumn)
print("All values in odd rows and even columns: ", oddRowsEvenColumns)
"""
Output:
Simple Array:  [19  8  7  1  5  4]
First Element:  19
Last Element:  4
The simple array without its first and last element:  [8 7 1 5]
A two-dimensional array:  [[1 2 3]
 [4 5 6]
 [7 8 9]]
First Row:  [1 2 3]
First Column:  [1 4 7]
Second Column:  [2 5 8]
All values in odd rows and even columns:  [[4 6]]

"""
```

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

```
#file name: reshape.py
import numpy as np

myarray = np.array([[2,5,6],[3,4,7]])
print("myarray: \n",myarray)
print("sizeofmyarray:", myarray.shape) # this shows: (gives the number of rows and columns of
    myarray) the dimensionality and the size of each dimension.
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.
print("onecolumnsixrows: \n",onecolumnsixrows)
onedimarray = myarray.reshape(-1)#It's also possible to change the dimensionality. This now is
     a one-dimensional array
print("onedimarray: \n",onedimarray)
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.
# the number of numbers inside parentheses determines the dimensionallity of the array, the -1
     will automatically provide the adequate number,
# and the multiplication of the numbers (excluding -1, means including any proper
    multiplication) should match to the numbers of the elements of the array.
print("threedimarray: \n",threedimarray)
"""
output:
myarray:
 [[2 5 6]
 [3 4 7]]
sizeofmyarray: (2, 3)
onecolumnsixrows:
 [[2]
 [5]
 [6]
 [3]
 [4]
 [7]]
onedimarray:
 [2 5 6 3 4 7]
threedimarray:
 [[[2 5 6]]
```

```

 [[3 4 7]]]
"""
```

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

```python
# file name: linspace.py
# here is an example of linspace
import numpy as np
x=np.linspace(0,20,100)
print(x)

'''
 output:

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

 '''
```

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.

```python
# file name: arange.py
# This file contains the same commands as range.py, but using numpy.arange rather than range.
    The output is the same.
import numpy as np
print('this is the range of a given number 10, the numpy style')
x=np.arange(10)
for n in x:
    print(n)
print('this is the range of given starting number 1, and ending number 10')
y=np.arange(1,10)
for n in y:
    print(n)
print('this is the range of given starting number 1, and ending number 10,  with defined step
    =2')
z=np.arange(1,10,2)
for n in z:
    print(n)

#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.):
print(range(1,10,2))#output: range(1, 10, 2)
print(np.arange(1,10,2))#output: [1 3 5 7 9]
'''
output:
this is the range of a given number 10, the numpy style
0
1
```

```
2
3
4
5
6
7
8
9
this is the range of given starting number 1, and ending number 10
1
2
3
4
5
6
7
8
9
this is the range of given starting number 1, and ending number 10,  with defined step=2
1
3
5
7
9
range(1, 10, 2)
[1 3 5 7 9]

'''
```

4.5.1 comparing arange and linspace

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

```
#file name: linspace_and_arange_comparision.py

#This file makes the difference between linspace and arange clear. These two commands are
    often mixed up.
import numpy as np

print(np.linspace(4,20,3))#The last number indicates the number of substeps.
print(np.arange(4,20,3))#The last number indicates the step size between substeps.

"""
output:
[ 4. 12. 20.]
[ 4  7 10 13 16 19]
"""
```

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 retuns 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.

```
# file name: meshgrid.py
import numpy as np
x=[1,2,3,4]
y=[5,6,7]
XX, YY=np.meshgrid(x,y)
print("This is XX values:\n", XX) # \n will take us to the next line
print("This is YY values:\n", YY)
'''
output:
This is XX values:
 [[1 2 3 4]
 [1 2 3 4]
 [1 2 3 4]]
This is YY values:
 [[5 5 5 5]
```

```
 [6 6 6 6]
 [7 7 7 7]]

'''
```

4.7 random

Here we illustrate how to generate random numbers using numpy

```
# file name: random1.py

# To generate a 1D random number between [0,1):
import numpy as np
x=np.random.rand(5)
print(x)
'''
output:
[0.90907376 0.13799189 0.58644767 0.9362912  0.22169574]
'''
# To generate a 2D random number:
y=np.random.rand(2,2)
print(y)

'''
output:
[[0.63012486 0.50949039]
 [0.83741808 0.63140463]]
'''
# To generate a nD random number:
# myrandom=np.random.rand(1D,2D,3D,..,nD)


# To generate random number with Gaussian distribution
# np.random.normal(mean, standard deviation, shape)
# one dimension: np.random.normal(mean, standard deviation, 1d)
xg=np.random.normal(0.0,1.0,5)
print(xg)

'''
output:
[-0.16594284  1.50995672 -0.93375937 -0.01684865  0.13507735]

'''
# To generate random Gaussian distribution with shape 2D:

yg=np.random.normal(0.0,1.0,(2,2))
print(yg)

'''
output:

[[ 0.89220523  0.98818578]
 [ 0.40821246 -0.7114725 ]]

'''
# To generate Guasssian random number for nD
# 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

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

[1 2 3 4 5 6]

'''
```

4.9 vstack()

```python
#file name: vstack.py
import numpy as np
x=np.array([1,2,3])
y=np.array([4,5,6])
print(np.vstack((x,y)))
'''
output:

[[1 2 3]
 [4 5 6]]

'''
x1=np.array([[11],[12],[13]])
y1=np.array([[14],[15],[16]])
print(np.vstack((x1,y1)))
'''
output:

[[11]
 [12]
 [13]
 [14]
 [15]
 [16]]

'''
```

4.10 Conditional Selection

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

```python
#file name: numpy_condition
import numpy as np
x=[1,7,4, 90, 12.4, 56, 10, 2, 0, -5, 22, 34, 65, 10, -4, 17, 2]
y=np.array(x)
print(y<12)
print(y[y<12])

'''
output:
[ True  True  True False False False  True  True  True  True False False
 False  True  True False  True]
[ 1.  7.  4. 10.  2.  0. -5. 10. -4.  2.]
'''
```

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

```python
#file name: statistic1.py
list1=["Carl", "Hanna", "Piter", "Ali", "Hassan", "Paul", "Zainab", "Zahra", "Catalina", "Anna
    ", "Julia", "Dina", "Sara", "Lina", "Albert"]
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]
import numpy as np
height_np=np.array(list2)
name_np=np.array(list1)
# let's figure out the height of Zahra and Julia:
print("Zahra height= ", height_np[name_np=="Zahra"], "Julia height= ", height_np[name_np=="
    Julia"])
'''
output
Zahra height=  [1.69] Julia height=  [1.87]
'''
```

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.

```python
#file name: where.py

import numpy as np
mylist = np.array([9,1,4,2,-4,1,-6,-1,4])
mylistAbsolute = np.where(mylist < 0, -mylist, mylist)#all the negative values are turned
    positive
print(mylistAbsolute)
```

```
"""
output:
[9 1 4 2 4 1 6 1 4]
"""
```

4.12 Statistical Methods

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

Functions like: mean, median, min, max.

```
#file name: statistic.py
# here is a list of 15 student's height in cm which includes mistakenly some wrong values, let
    's see how we can deal with that.
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,
    1.44]
# let's see the mean of their hights:
import numpy as np
np_student_h=np.array(student_h)
print("mean: ", np.mean(np_student_h))
# the output is: 13.473333333333334, it seems something went wrong! but not with the mean,
    maybe a value inserted wrongly.
# let's see the median
print("median: ", np.median(np_student_h))
# output is:  1.47  which makes sense

# let's see the maximum, and minimum height of this list:

print("maximum: ", np.max(np_student_h)) # output: maximum:  166.0
print("minimum: ", np.min(np_student_h)) # output: minimum:  1.3

# 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.
```

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

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

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

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

The result looks as follows. You can obtain the two dimensional plot also with the scattering option. Note that, scatter plot is useful to assess the correlation between horizontal and vertical axes's.

```
# file name: plot_list_data_3.py
import matplotlib.pyplot as plt
plt.scatter([100, 2000, 5000, 400000], [1, 0, 9, 15])
```

Figure 1: one-dimensional plot

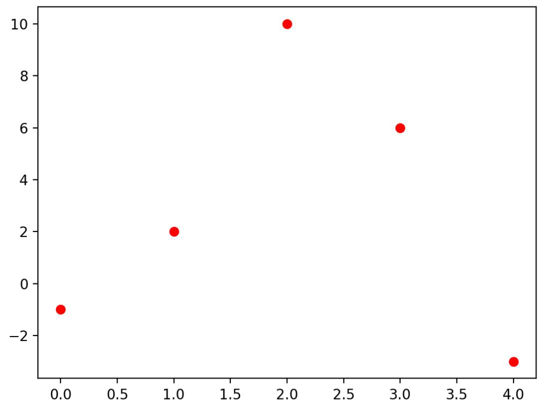
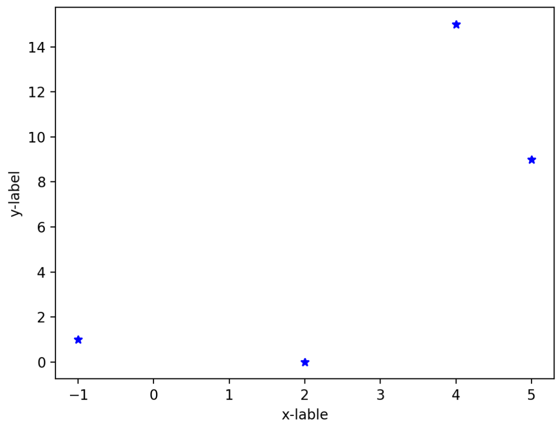


Figure 2: two-dimensional plot




```
plt.xscale('log') # if the range of the values is really big, the logarithmic scale is a good option to represent the data.
plt.xlabel('x-lable') # here is how to lable the plot
plt.ylabel('y-label')
plt.show()
```

5.1.1 connected data point list plot

```
#file name: plot_list_data_4
import matplotlib.pyplot as plt
plt.plot([-1, 2, 5, 4], [1, 0, 9, 15]) #if we use plt.plot([-1, 2, 5, 4], [1, 0, 9, 15], 'b*') instead, it will not connect the points in the list.
plt.xlabel('x-lable') # here is how to lable the plot
plt.ylabel('y-label')
plt.show()
```

5.2 Plotting Function

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

```
#file name: sin.py
from matplotlib import rc # this line is for naming the plot on top of the frame.
rc('text', usetex=True)
import numpy as np
import math
import matplotlib.pyplot as plt


x = np.linspace(0.,17.,200) # here is to define the number of points, and the size of the x-axix
```

Figure 3: scatter plot

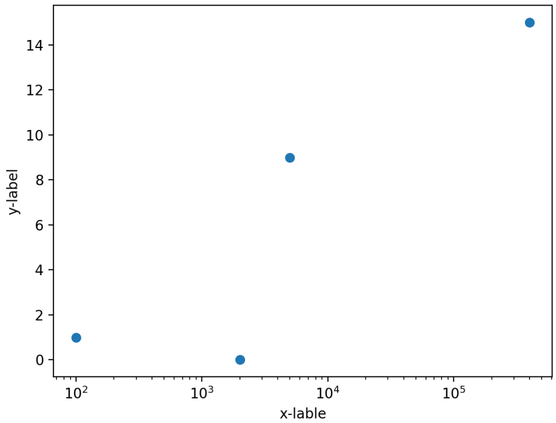
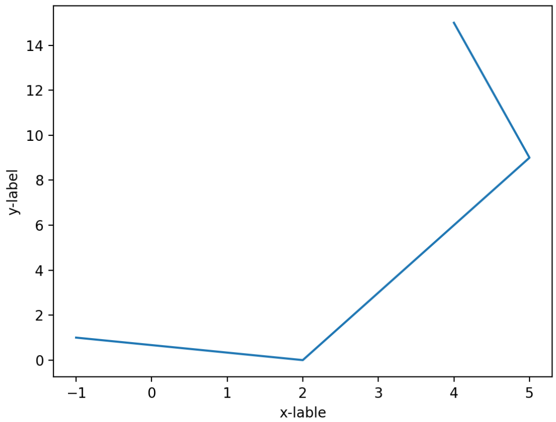


Figure 4: connected points (line plot)

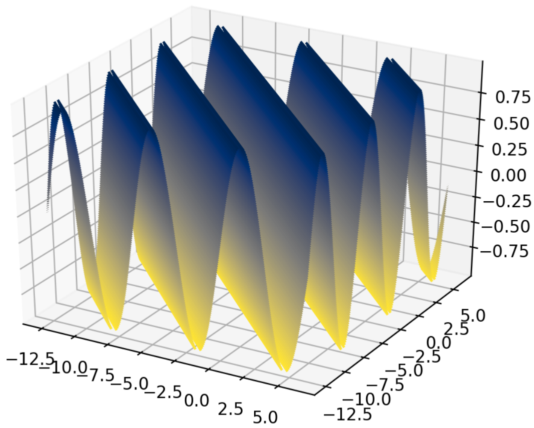


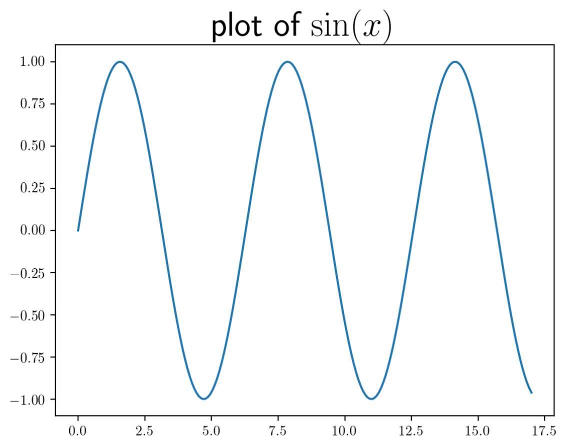

```
fx = np.sin(x) # here goes the function itself

plt.plot(x,fx)
# in the following line you see how we can write text as well as mathematiacl equations with the latex formating.
plt.title(
  r'plot of $\sin(x)$', fontsize=25
)


plt.show()
```

5.3 3D plot

```
#file name: 3dplot.py
import matplotlib.pyplot as plt
import math
import numpy as np
from mpl_toolkits import mplot3d
ax = plt.axes(projection='3d')
x=np.arange(-4*np.pi, 2*np.pi,0.1)
y=np.arange(-4*np.pi, 2*np.pi,0.1)
x, y = np.meshgrid(x, y)
z=np.sin(x+y)
ax.contour3D(x, y, z,100,cmap='cividis_r' ) # possible other options for "cmap" are listed bellow.
plt.show()
"""
possible other colors:
```

```
Possible values are: Accent, Accent_r, Blues, Blues_r, BrBG, BrBG_r, BuGn, BuGn_r, BuPu, BuPu_r, CMRmap, CMRmap_r, Dark2, Dark2_r, GnBu, GnBu_r, Greens,
Greens_r, Greys, Greys_r, OrRd, OrRd_r, Oranges, Oranges_r, PRGn, PRGn_r, Paired, Paired_r, Pastel1, Pastel1_r, Pastel2, Pastel2_r, PiYG, PiYG_r, PuBu,
PuBuGn, PuBuGn_r, PuBu_r, PuOr, PuOr_r, PuRd, PuRd_r, Purples, Purples_r, RdBu, RdBu_r, RdGy, RdGy_r, RdPu, RdPu_r, RdYlBu, RdYlBu_r, RdYlGn, RdYlGn_r,
Reds, Reds_r, Set1, Set1_r, Set2, Set2_r, Set3, Set3_r, Spectral, Spectral_r, Wistia, Wistia_r, YlGn, YlGnBu, YlGnBu_r, YlGn_r, YlOrBr, YlOrBr_r, YlOrRd,
YlOrRd_r, afmhot, afmhot_r, autumn, autumn_r, binary, binary_r, bone, bone_r, brg, brg_r, bwr, bwr_r, cividis, cividis_r, cool, cool_r, coolwarm,
coolwarm_r, copper, copper_r, cubehelix, cubehelix_r, flag, flag_r, gist_earth, gist_earth_r, gist_gray, gist_gray_r, gist_heat, gist_heat_r,
gist_ncar, gist_ncar_r, gist_rainbow, gist_rainbow_r, gist_stern, gist_stern_r, gist_yarg, gist_yarg_r, gnuplot, gnuplot2, gnuplot2_r, gnuplot_r,
gray, gray_r, hot, hot_r, hsv, hsv_r, inferno, inferno_r, jet, jet_r, magma, magma_r, nipy_spectral, nipy_spectral_r, ocean, ocean_r, pink, pink_r,
plasma, plasma_r, prism, prism_r, rainbow, rainbow_r, seismic, seismic_r, spring, spring_r, summer, summer_r, tab10, tab10_r, tab20, tab20_r, tab20b,
 tab20b_r, tab20c, tab20c_r, terrain, terrain_r, twilight, twilight_r, twilight_shifted, twilight_shifted_r, viridis, viridis_r, winter, winter_r

 """
```





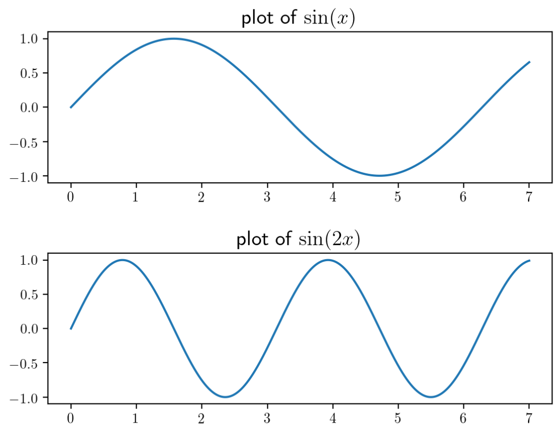
5.4 subplot

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

```
#file name: subplot.py
import numpy as np
```

```python
import matplotlib.pyplot as plt
from matplotlib import rc
rc('text', usetex=True)

x = np.linspace(0.,7.,10000)

fx = np.sin(x)
fy = np.sin(2*x)

# the first subplot:

plt.subplot(2,1,1) # this line would tell how our subplot should look like: in this case it has two rows and one coloum.
# the last parameter refers to the plot number which in principle starts from one, meaning that it counts subplots from left to right
# and from top to the bottom.
plt.plot(x,fx)
# the plot title of each subplot needs to be below subplot function and before the next one.
plt.title(
  r'plot of $\sin(x)$', fontsize=15
)

# the second subplot:

plt.subplot(2,1,2)
plt.plot(x,fy)
plt.title(
  r'plot of $\sin(2x)$', fontsize=15
)

plt.show() # this line is needed to show the plot at the end.
```

Figure 5: here is an example of subplot



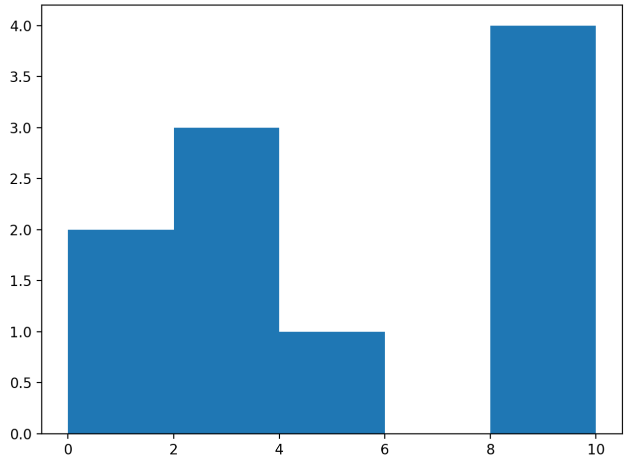
5.5 Histogram

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.

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

```
axs.hist(x, bins='auto')
plt.show()
```

Figure 6: here is an example of histogram

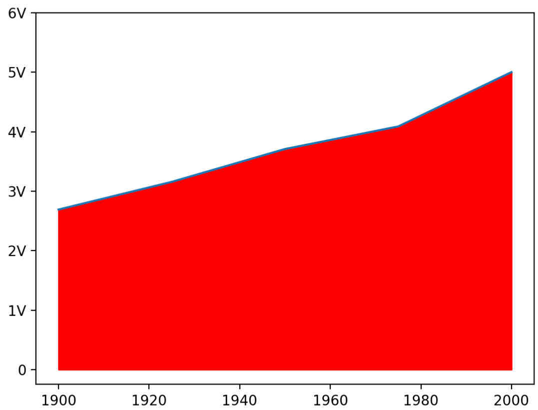


5.6 Customisation

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

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

Figure 7: fill_between, and yticks

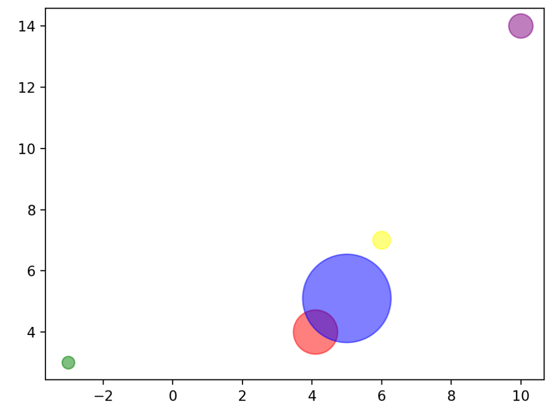


Here is an example for the scatter plot with the size of the data points.

```
# file name: scatter_size.py
import matplotlib.pyplot as plt
import numpy as np
x=np.array([4.1,5,10,6,-3])
y=np.array([4,5.1,14,7,3])
oursize=np.array([100,400,30,16,8])
ourcolor=['red','blue', 'purple','yellow','green']
plt.scatter(x,y,s=10*oursize,c=ourcolor, alpha=0.5) # alpha changes the opacity of the colors,
#it varies from zero to one. # "s" referes to the size, and can also be written "size", "c"
    refers to color and
```

```
# can be written "color" as well.
plt.show()
```

Figure 8: scatter plot with size and colors

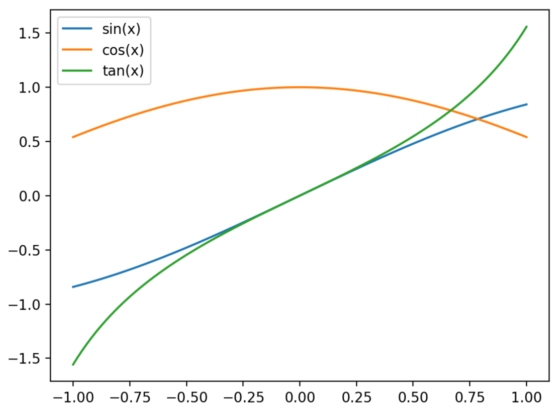


5.7 Legends

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

```
# file name: legend.py
import numpy as np
import math
import matplotlib.pyplot as plt
x = np.linspace(-1,1,200)
fig, ax = plt.subplots()
y1=np.sin(x)
y2=np.cos(x)
y3=np.tan(x)
ax.plot(x,y1,label='sin(x)')
ax.plot(x,y2,label='cos(x)')
ax.plot(x,y3,label='tan(x)')

ax.legend()
plt.show()
```

Figure 9: Legend



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

6 Pandas

Pandas is a high level data manipulation tool. Pandas has two notable data structure [2], 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.

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

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

```
#file name: corona_cases.py
import pandas as pd
data = pd.read_csv("2020-05-15_Corona_Cases.csv")
print(data)
'''
output:

  Unnamed: 0  Country   Cases   Dead  Recovered
0          IR     Iran  116635   6902      91836
1          CN    China   82933   4633      78209
2          IT    Italy  223885  31610     120205
3          DE  Germany  175223   7933     151700

'''
```

As you noticed the row labels, the country codes that we want to use as row indices, are seen 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.

```
#file name: corona_cases_2.py
import pandas as pd
data = pd.read_csv("2020-05-15_Corona_Cases.csv", index_col=0)
print(data)
'''
output:

    Country   Cases   Dead  Recovered
IR     Iran  116635   6902      91836
CN    China   82933   4633      78209
IT    Italy  223885  31610     120205
DE  Germany  175223   7933     151700

'''
```

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`

```
# file name: pandas_select.py
import pandas as pd
data = pd.read_csv("2020-05-15_Corona_Cases.csv", index_col=0)
print(data["Cases"])
'''
output

IR    116635
CN     82933
IT    223885
DE    175223
Name: Cases, dtype: int64

'''
print(data.Cases) #alternative way to access a column

'''
output
```

[2]some data type that stores data in structured way, for example a NumPy array

```

IR     116635
CN      82933
IT     223885
DE     175223
Name: Cases, dtype: int64

'''
```

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

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

# select single column with Pandas dataframe

print(data[["Cases"]])
'''
output:
     Cases
IR  116635
CN   82933
IT  223885
DE  175223

'''
# Select multiple columns with Pandas dataframe
print(data[["Cases", "Recovered"]])

'''
output

     Cases  Recovered
IR  116635      91836
CN   82933      78209
IT  223885     120205
DE  175223     151700

'''
```

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

```
# file name: row_pandas.py
import pandas as pd
data = pd.read_csv("2020-05-15_Corona_Cases.csv", index_col=0)
print(data.loc["DE"])
'''
output

Country      Germany
Cases         175223
Dead            7933
Recovered     151700
Name: DE, dtype: object

'''
```

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

```
#file name: element_pandas.py
import pandas as pd
data = pd.read_csv("2020-05-15_Corona_Cases.csv", index_col=0)
print("One way:", data.loc["DE", "Recovered"])
# alternativly:
print("The other way: ", data["Recovered"].loc["DE"])
# or
print("Or: ", data.loc["DE"]["Recovered"])
'''
One way: 151700
The other way:  151700
Or:  151700
'''
```

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.

```
#file name: find_trieste.py
import pandas as pd
data=pd.read_csv("dpc-covid19-ita-province-latest.csv")
trieste_row=data[data.denominazione_provincia=="Trieste"]
print(trieste_row)
'''
output:
                    data stato  codice_regione  denominazione_regione  ...       long
    totale_casi note_it  note_en
34  2020-05-23T17:00:00   ITA               6  Friuli Venezia Giulia  ...  13.768136
    1372      NaN      NaN

[1 rows x 12 columns]
'''
# to find out the totale_casi of the trieste we can do the following:
print(trieste_row[["totale_casi"]])
'''
output:

    totale_casi
34         1372
'''
```

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:

```
# file name: pandas_column.py
import pandas as pd
data = pd.read_csv("2020-05-15_Corona_Cases.csv", index_col=0)
data["Serious critical"]=[2294, 8, 762, 1166]
print(data)
'''
output:

    Country   Cases   Dead  Recovered  Serious critical
IR     Iran  116635   6902      91836              2294
CN    China   82933   4633      78209                 8
IT    Italy  223885  31610     120205               762
DE  Germany  175223   7933     151700              1166

'''
data["Active cases"]=data["Cases"]-(data["Dead"]+data["Recovered"])

print(data)

'''
output:
    Country   Cases   Dead  Recovered  Serious critical
IR     Iran  116635   6902      91836              2294
CN    China   82933   4633      78209                 8
IT    Italy  223885  31610     120205               762
DE  Germany  175223   7933     151700              1166
    Country   Cases   Dead  Recovered  Serious critical  Active cases
IR     Iran  116635   6902      91836              2294         17897
CN    China   82933   4633      78209                 8            91
IT    Italy  223885  31610     120205               762         72070
DE  Germany  175223   7933     151700              1166         15590
'''
```

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 a field of computer science that 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

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.

```python
# file name: enumerate.py
# this is an example on how enumerate function works
mylist=['rose', 'orchid', 'tulip', 'sunflower']
for counter, value in enumerate(mylist, 1): # this 1 is to enumerate from one instead of zero.
    print(counter, value)

'''
output:
1 rose
2 orchid
3 tulip
4 sunflower

'''
```

10.2 Init function

here is an example of init function:

```python
# file name: Init.py
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)#The __init__ function allows us to specify parameters for the
    instantiation of p1.
print(p1.name)
print(p1.age)
```

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

```python
# file name: NoInit.py
class Person:
  def function(self, name, age):
    self.name = name
    self.age = age


```

```
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.
p1.function("John", 36)#If we want to specify attributes, we need to call an extra function.
print(p1.name)
print(p1.age)
'''
* 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.
'''
```

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 on is the file that call the first file to use the class which is defined in the first file.

```
# file name: main_file.py
# this file is called main_file and contains the main condition

class Person:
  def function(self, name, age):
    self.name = name
    self.age = age


if __name__ == "__main__":
  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.
  p1.function("John", 36)#If we want to specify attributes, we need to call an extra function.
  print(p1.name)
  print(p1.age)
  '''
  * 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.
  '''

"""
When we run this file, the output is:
John
36
"""
```

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

```
# file name: main_example.py
# this is the second file which is importing the main_file, the first file
from main_file import Person

professor = Person()
professor.function("Mehran", 45)

print(professor.name)
"""
When we run this file, the output is:
"Mehran"
"""
```

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

```
# file name: range.py
print('this is the range of a given number 10')
x=range(10)
for n in x:
    print(n)
print('this is the range of given starting number 1, and ending number 10')
y=range(1,10)
for n in y:
    print(n)
print('this is the range of given starting number 1, and ending number 10,  with defined step
    =2')
z=range(1,10,2)
for n in z:
    print(n)
'''
output:
this is the range of a given number 10
0
1
2
3
4
5
6
7
8
9
this is the range of given starting number 1, and ending number 10
1
2
3
4
5
6
7
8
9
this is the range of given starting number 1, and ending number 10,  with defined step=2
1
3
5
7
9
'''
```

10.5 ravel()

It is a function to flatten the data.

```
# file name: ravel.py
import numpy as np
x=[[1,2],[3,4]]
print("Initially it looks:", x)
# when it gets flattened:
y=np.ravel(x)
print("Flattened version looks:", y)
'''
output:
Initially it looks: [[1, 2], [3, 4]]
Flattened version looks: [1 2 3 4]
'''
```

Another example:

```
# file name: ravel2.py
import numpy as np
x=np.array([[1,2],[3,4]])
print("Initially looks: ", x)
print("Flattened version looks: ", x.ravel())

'''
output:
Initially looks:  [[1 2]
 [3 4]]
Flattened version looks:  [1 2 3 4]
'''
```

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:

```python
#file name: mean.py
import numpy as np # a code which finds the mean of a list:
p=np.asarray(eval(input("numbers?")))
y=0
z=0
for x in p:
   z=z+1
   y=y+x
   mean=y/z
print(mean)
```

2. A sample answer:

```python
#file name: match.py

import numpy as np
from numpy.random import choice
def tour(b,r):
    c=b.size//2
    pair=choice(b, size=(c,2), replace=False)
    print("************************")
    print("This is the",r,"round")
    print(pair)
    if b.size % 2 ==1:
     notchosen=b[np.isin(b, pair)==False]
     print("Number", notchosen, "needs to wait.")
    winners=np.asarray(eval(input("Who (are) is the winner(s)? ")))
    if b.size %2==1:
     if winners.size>1:
       lucky=np.asarray(choice(winners, size=(notchosen.size)))
     else:
      lucky=winners
     print("Number", notchosen, "needs to play against", lucky)
     who=np.asarray(eval(input("Who won? ")))
     winners=np.asarray(np.where(winners==lucky, who, winners))
    return winners
p=input("Number of players: ")
a=eval(p)
r=1
people=np.arange(a)
while people.size>=2:
```

```
      people=np.asarray(tour(people,r))
      r=r+1
print("Cheers!")

'''
output for 9 participants:

Number of players: 9
************************
This is the 1 round
[[6 3]
 [7 1]
 [0 8]
 [2 4]]
Number [5] needs to wait.
Who (are) is the winner(s)? 6,7,8,2
Number [5] needs to play against [6]
Who won? 5
************************
This is the 2 round
[[2 8]
 [5 7]]
Who (are) is the winner(s)? 2,7
************************
This is the 3 round
[[7 2]]
Who (are) is the winner(s)? 7
Cheers!


'''
```

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)

```
# daily_cases_trieste.py
import matplotlib.pyplot as plt
import glob
import pandas as pd
import git

g = git.cmd.Git("./COVID-19") # this is to tell that ./COVID-19 is a git repository, and that the git commands that we apply on "g"
 #should be applied on that repository.
#This only works because we have already git cloned the repository to our local directory.
g.pull() # this is how to git pull the new data from the remote repository to our local repository.
list1=[]
# to read the csv files
for i in glob.glob('./COVID-19/dati-province/dpc-covid19-ita-province-20200*'):
   try: # this "try" and "except" is a check whether the files are all doing well!
   # (if some file is broken it won't be included in our list and the program continues )
       data=pd.read_csv(i, index_col="denominazione_provincia", parse_dates=True)
       store=data.loc["Trieste"][["totale_casi","data"]]
       list1+=[store]
   except:
       print("some error in file", i, "please fix it!")
list1=sorted(list1,key=lambda item:item['data'])
dlist=[]
clist=[]
for item in list1:
    dlist+=[item["data"]]
    clist+=[item['totale_casi']]
daily_new_cases=[]
for x in range(len(clist)-1):
    daily_new_cases+=[clist[x+1]-clist[x]]
# below we get rid of negative values for the new daily cases by setting it to the previous value of the previous day.
# This should have not been done as it basically ruin the actual data. We made it just to make the data look good for the moment.
for n, v in enumerate(daily_new_cases):
    if v<0:
        daily_new_cases[n]=daily_new_cases[n-1]
nlist=dlist[::10]
date=[]
```

```
for x in nlist:
    y=x.split("T")
    s=y[0].split("2020-")
    date+=[s[1]]
plt.bar(x=range(len(dlist)-1), height=daily_new_cases, color='grey')
dindex=list(range(len(dlist)))[::10]
plt.title("COVID-19, Trieste-2020", fontsize=15
)
plt.xlabel('Date: Month-Day')
plt.xticks(dindex,date, rotation=25)
plt.ylabel("New Cases")
plt.show()
```