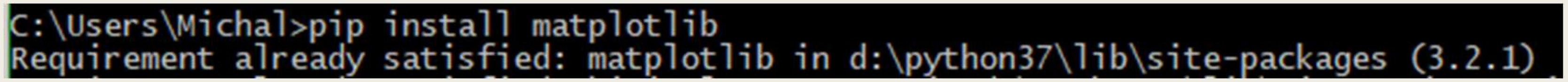
MATPLOTLIB

What is matplotlib

matplotlib
Version 3.3.4

- matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.
- Written in python, C, C#, JS
- It is used along with NumPy and other important libraries
- Author – John D. Hunter
- First version – 2003
- Licension – BSD (liberal) ← open source
- https://github.com/matplotlib/matplotlib

First steps

- 

```
C:\Users\Michal>pip install matplotlib
Requirement already satisfied: matplotlib in d:\python37\lib\site-packages (3.2.1)
```

- Import matplotlib
- Import matplotlib.pyplot
- Import matplotlib.pyplot as plt

Importing whole library…

… however, you often need just pyplot submodule…

… which is by custom imported as „plt"

First chart

```
import matplotlib.pyplot as plt
import numpy as np

xs = np.arange(10)
ys = np.arange(10)

plt.plot(xs, ys)

plt.show()
```

We import libraries, that we need using customary aliases

We can use the numpy library to generate dataset for our chart

This command is responsible for adding feauture to the chart

This command is responsible for displaying chart

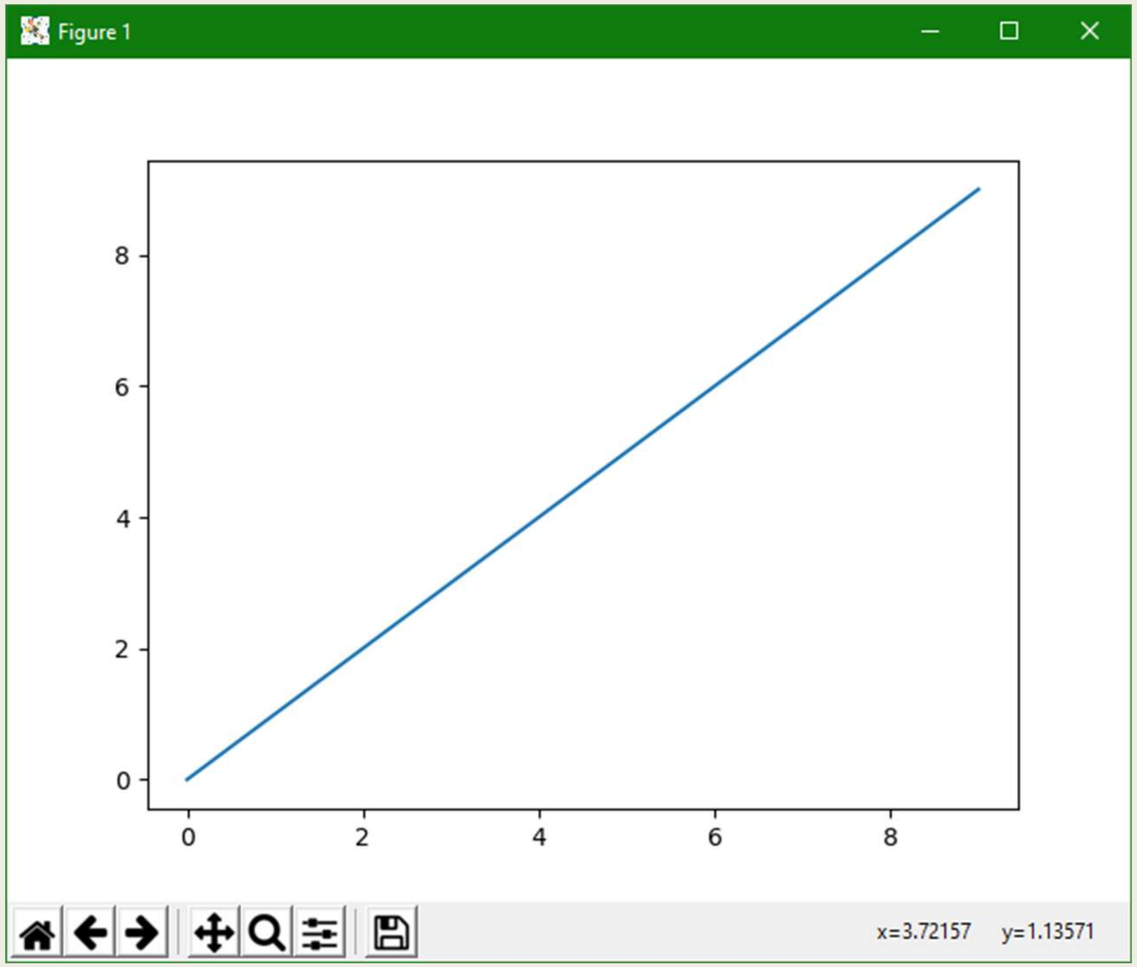
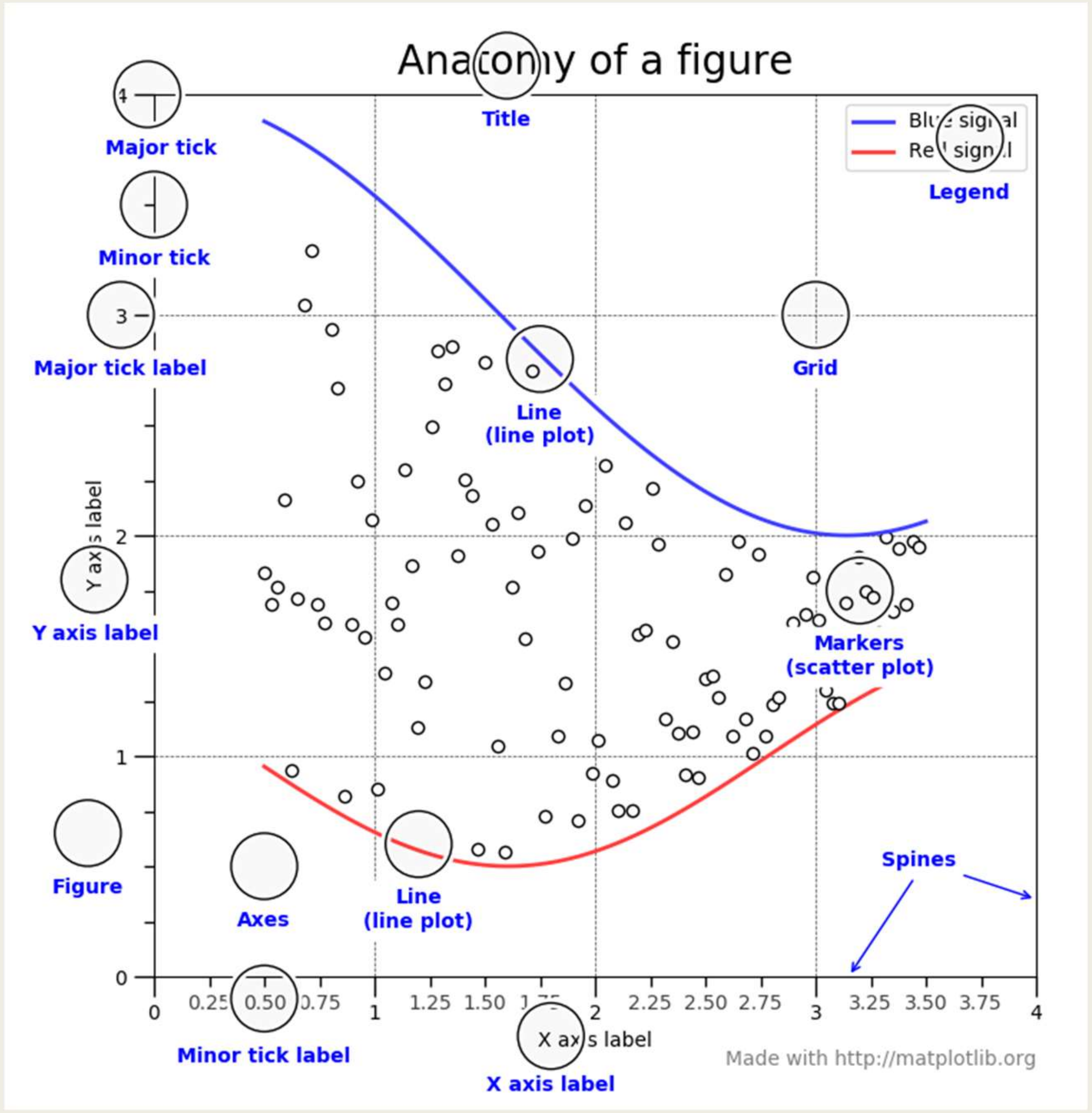


Figure 1
8
6
4
2
0
0
2
4
6
8
x=3.72157 y=1.13571



https://matplotlib.org/2.0.2/examples/showcase/anatomy.html

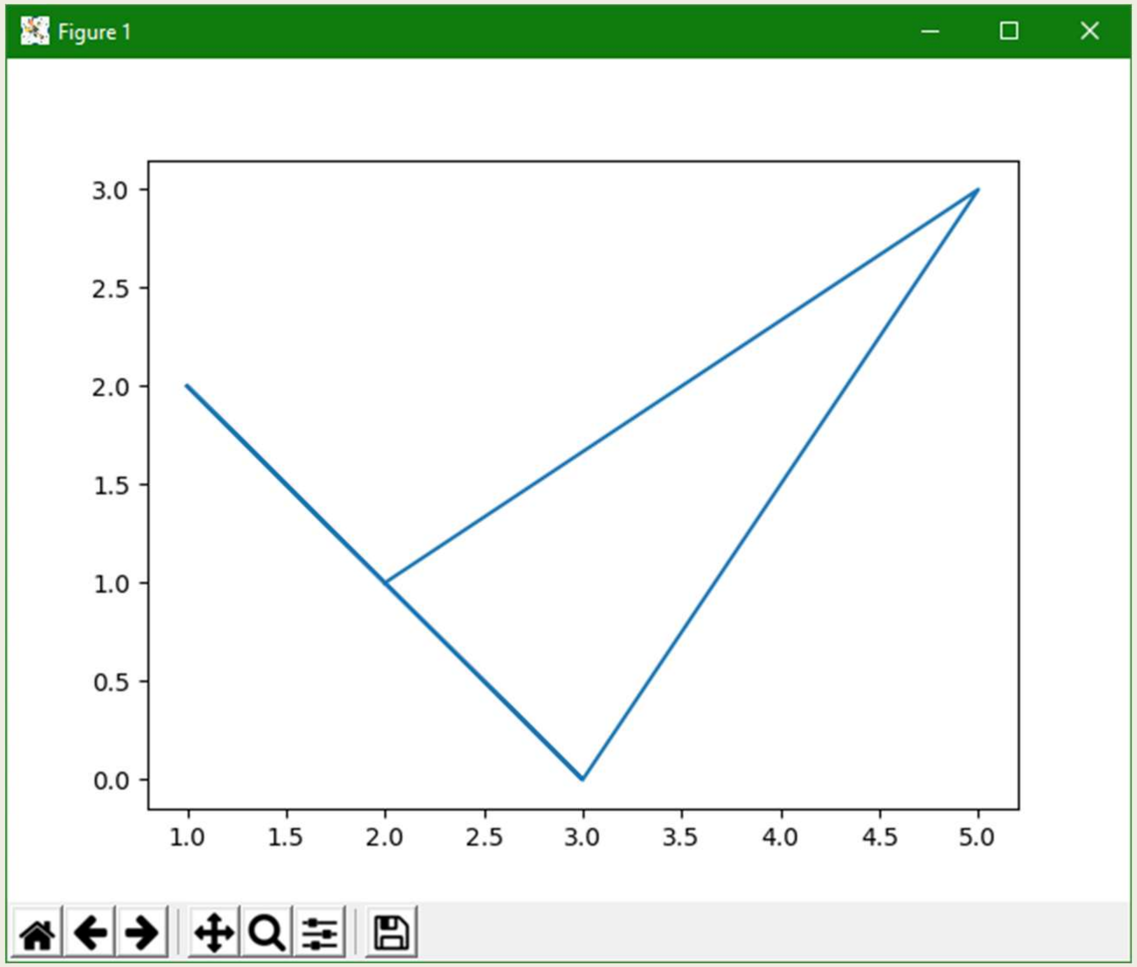


Figure 1
3.0
2.5
2.0
1.5
1.0
0.5
0.0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

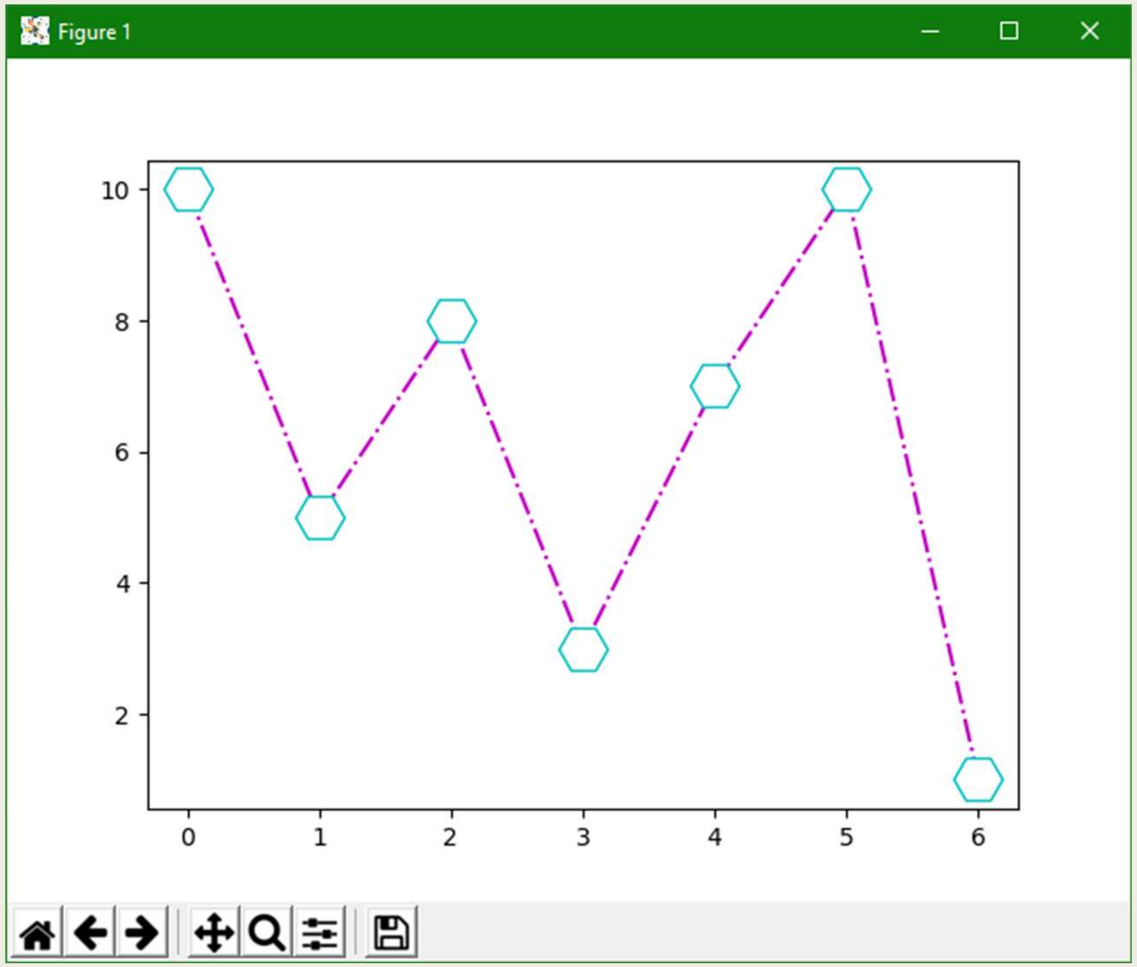


Figure 1
10
8
6
4
2
0
1
2
3
4
5
6

plot()

If you omit this, default is 1, 2, 3, 4, 5…

Default is line chart

- plot(points_on_x_axis, point_on_y_axis)

However, you can define marker

- plot(points_on_x_axis, point_on_y_axis, marker)

- plot(points_on_x_axis, point_on_y_axis, fmt)

Or fmt parameter which has syntax:

marker|line|colour

Markers and lines

- plt.plot(xpoints, ypoints, marker = PLACEHOLDER)

Check the table in the next slide

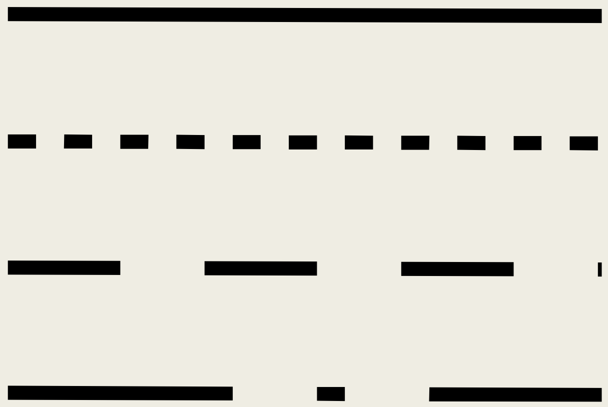
- plt.plot(xpoints, ypoints, marker+line+color)

-

:

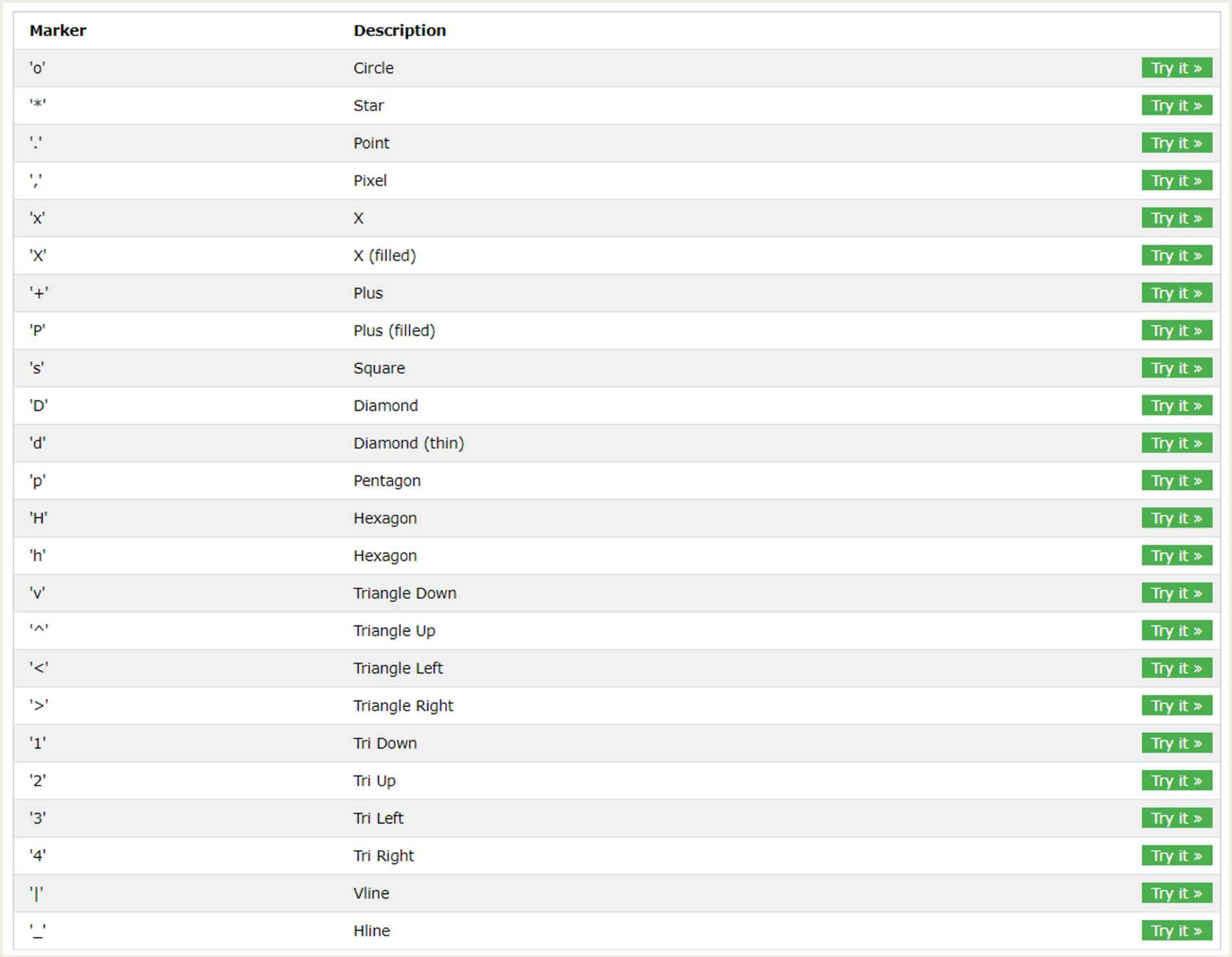
--

-.



R G B
C M Y K
W

mec = ‚marker edge colour’
mfc = ‚marker foreground colour’
ms = int(markersize)
lw = int(linewidth)



Marker	Description	
'o'	Circle	Try it »
'*'	Star	Try it »
'.'	Point	Try it »
','	Pixel	Try it »
'x'	X	Try it »
'X'	X (filled)	Try it »
'+'	Plus	Try it »
'P'	Plus (filled)	Try it »
's'	Square	Try it »
'D'	Diamond	Try it »
'd'	Diamond (thin)	Try it »
'p'	Pentagon	Try it »
'H'	Hexagon	Try it »
'h'	Hexagon	Try it »
'v'	Triangle Down	Try it »
'^'	Triangle Up	Try it »
'<'	Triangle Left	Try it »
'>'	Triangle Right	Try it »
'1'	Tri Down	Try it »
'2'	Tri Up	Try it »
'3'	Tri Left	Try it »
'4'	Tri Right	Try it »
'|'	Vline	Try it »
'_'	Hline	Try it »

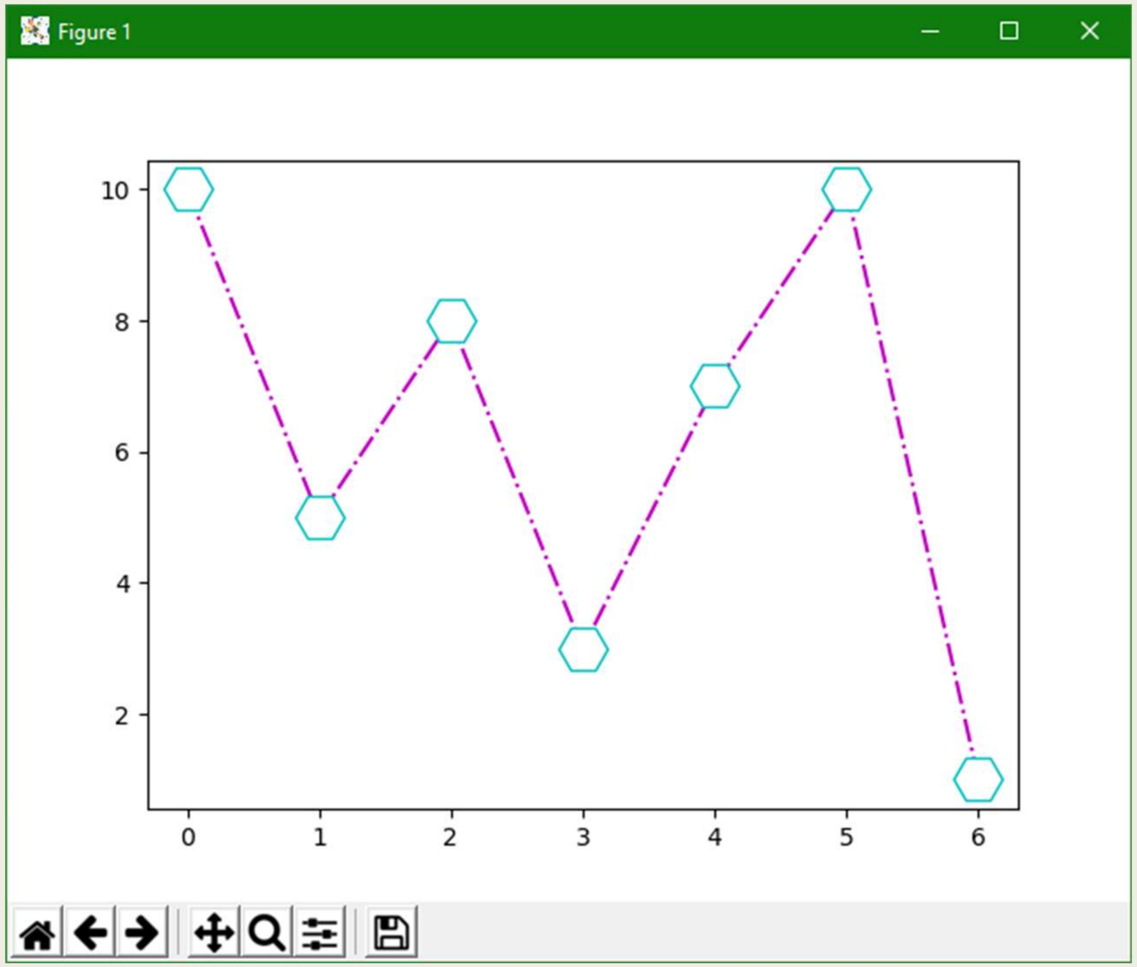


Figure 1
10
8
6
4
2
0
1
2
3
4
5
6

Multiple plots

- You can call plt.plot() multiple Times until you display chart with plt.show()
- If you don't define colours etc. they will be automatically different

Adding titles, labels to axis

- plt.title(„Title text”, loc = „left|center|right| …”)
- plt.xlabel(„Text to display”)
- plt.ylabel(„Text to display”)

Adding grid

- plt.grid()
- plt.grid(axis = „x|y”, color= „r|g|b| …”, linewidth = numer, linestyle= „-|:| … ”)

Ticks and limits

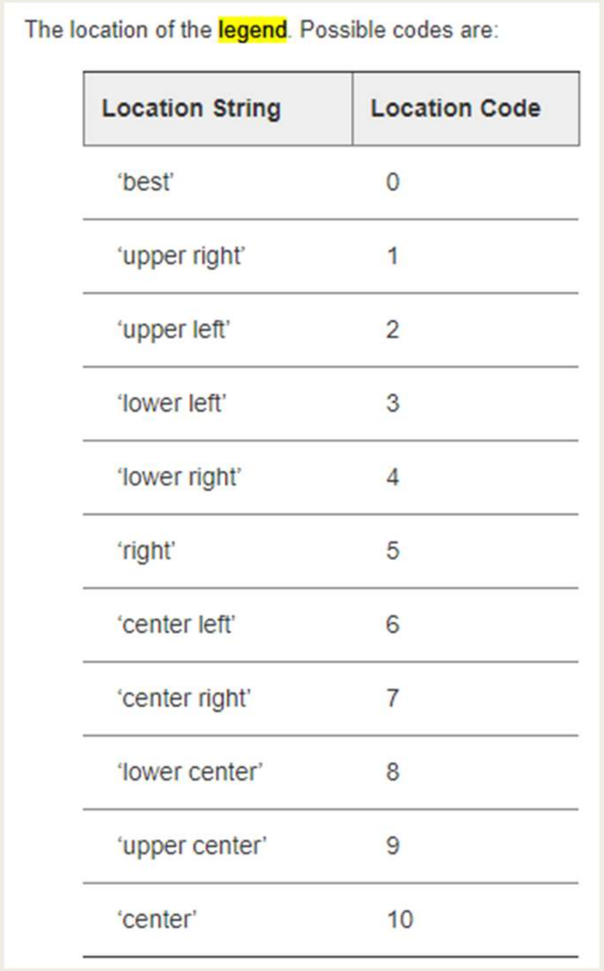
- `plt.xlim(min_x, max_x)`
- `plt.ylim(min_y, max_y)`
- `plt.axis([0.0, 1.0, 0.0, 1.0])`

- `plt.xticks(list of x ticks)`
- `plt.yticks(list of y ticks)`

- `plt.minorticks_on()`

Legend

- plt.legend()
  - *loc = „upper left …”,*
  - *frameon =„True/False”*
  - *fontsize = 20 …*
  - *title = „text”*



The location of the legend. Possible codes are:

Location String	Location Code
'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

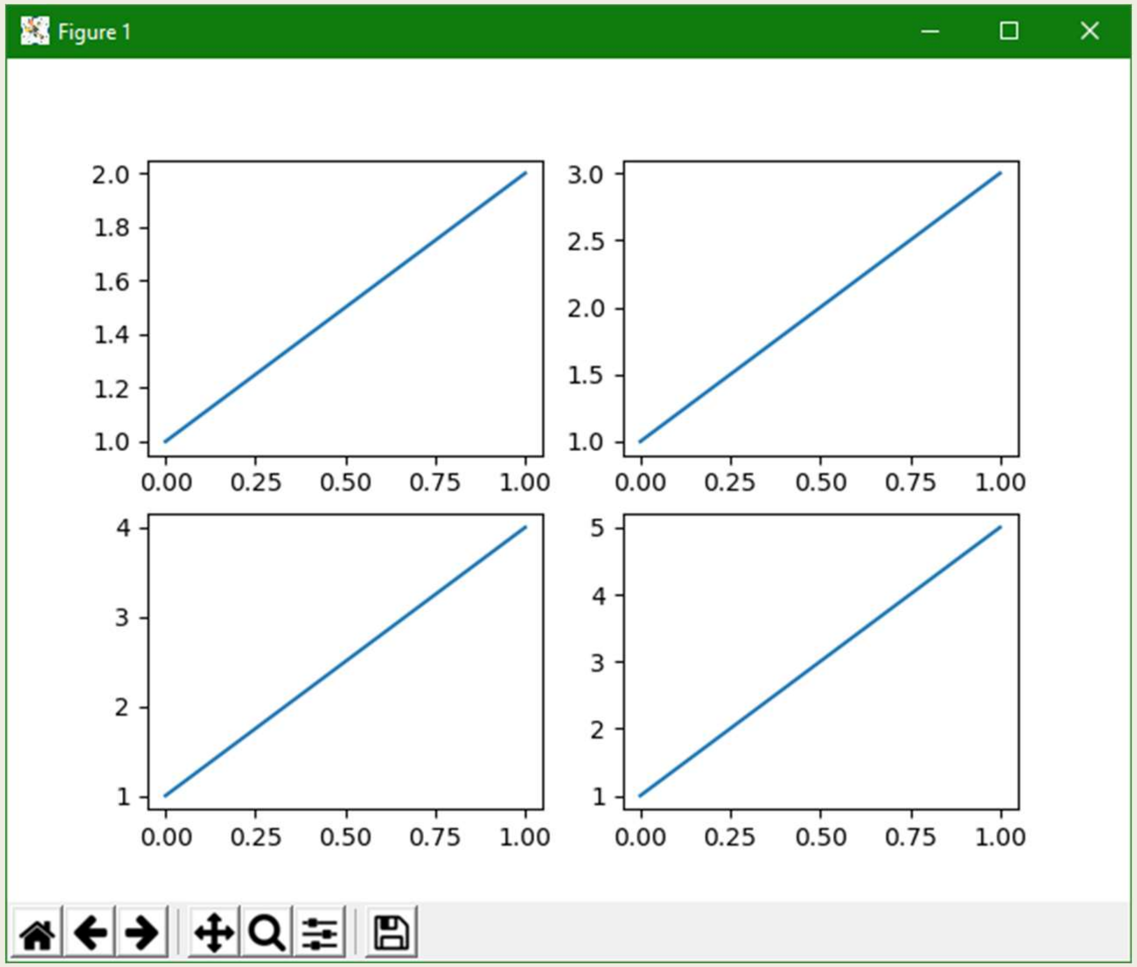
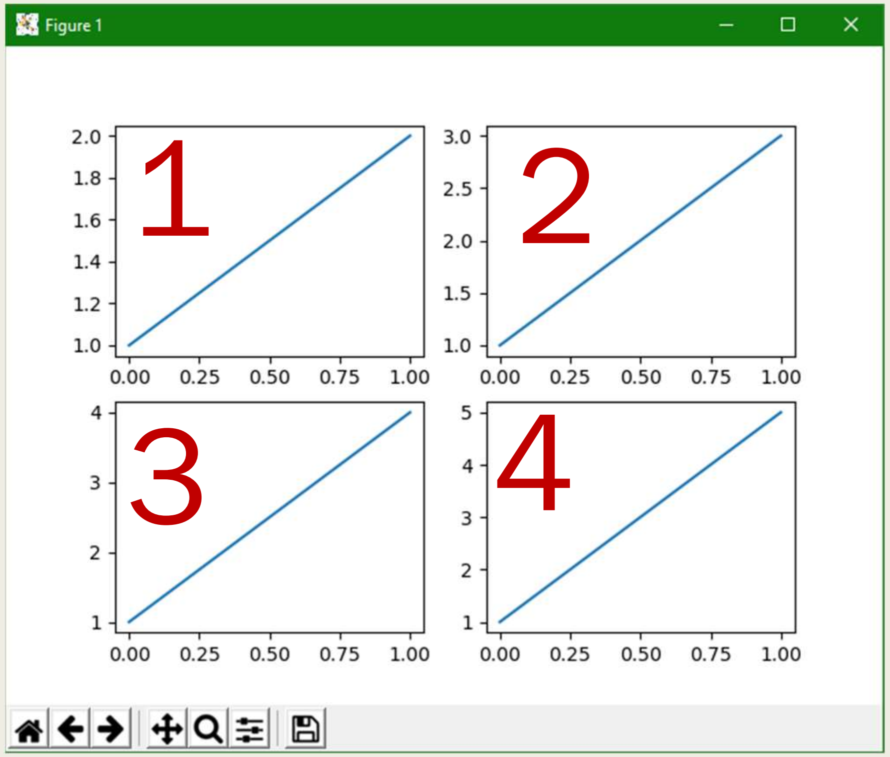


Figure 1
2.0
1.8
1.6
1.4
1.2
1.0
0.00
0.25
0.50
0.75
1.00
3.0
2.5
2.0
1.5
1.0
4
3
2
1
5

Subplots

- You can plot more than one chart in one plt.show()
- plt.subplot(how_many_rows, how_many_columns, which_subplot_is_this)



Bar chart

- plt.bar(category_list, values_list)
- plt.barh(category_list, values_list)
- Parameters:
  - *color*
  - *Width/height*

Vertical bars

horizontal

By default 0.8

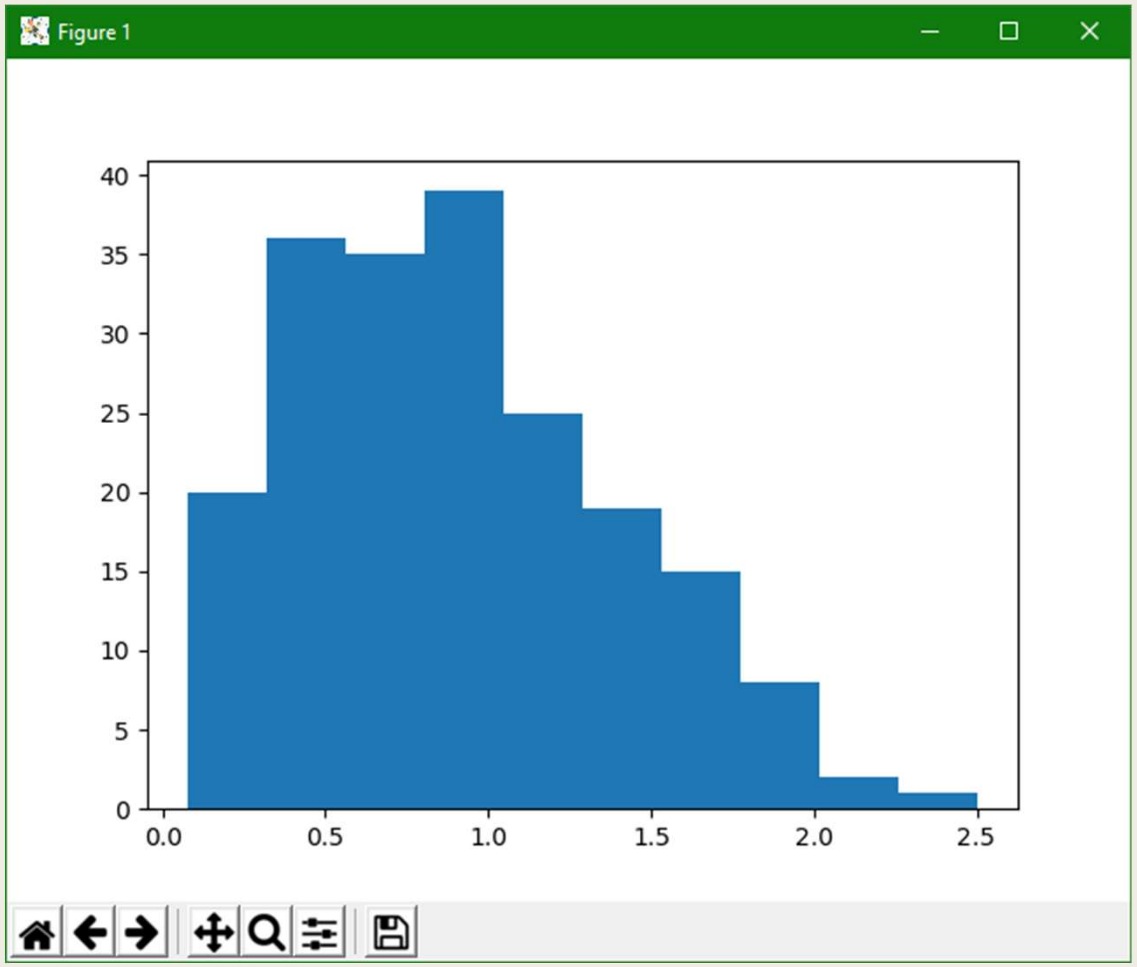
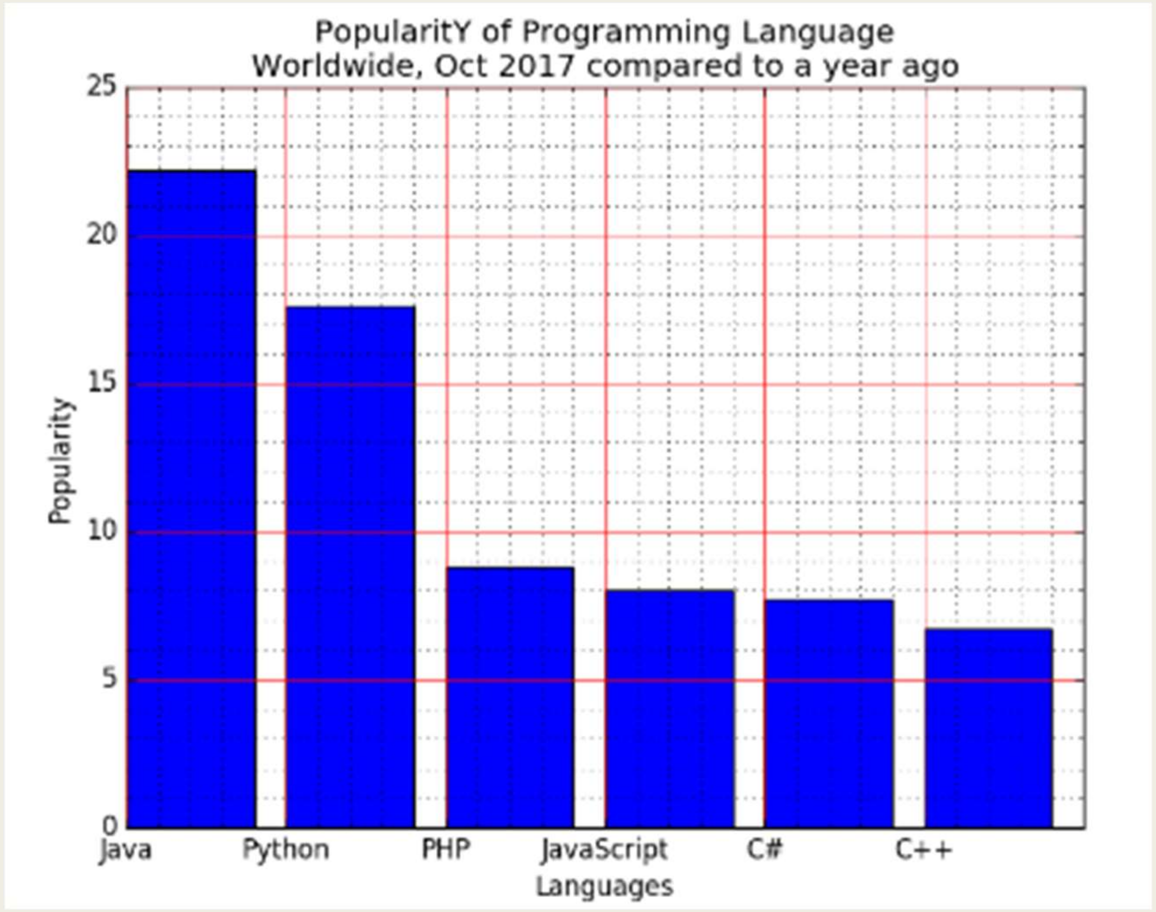
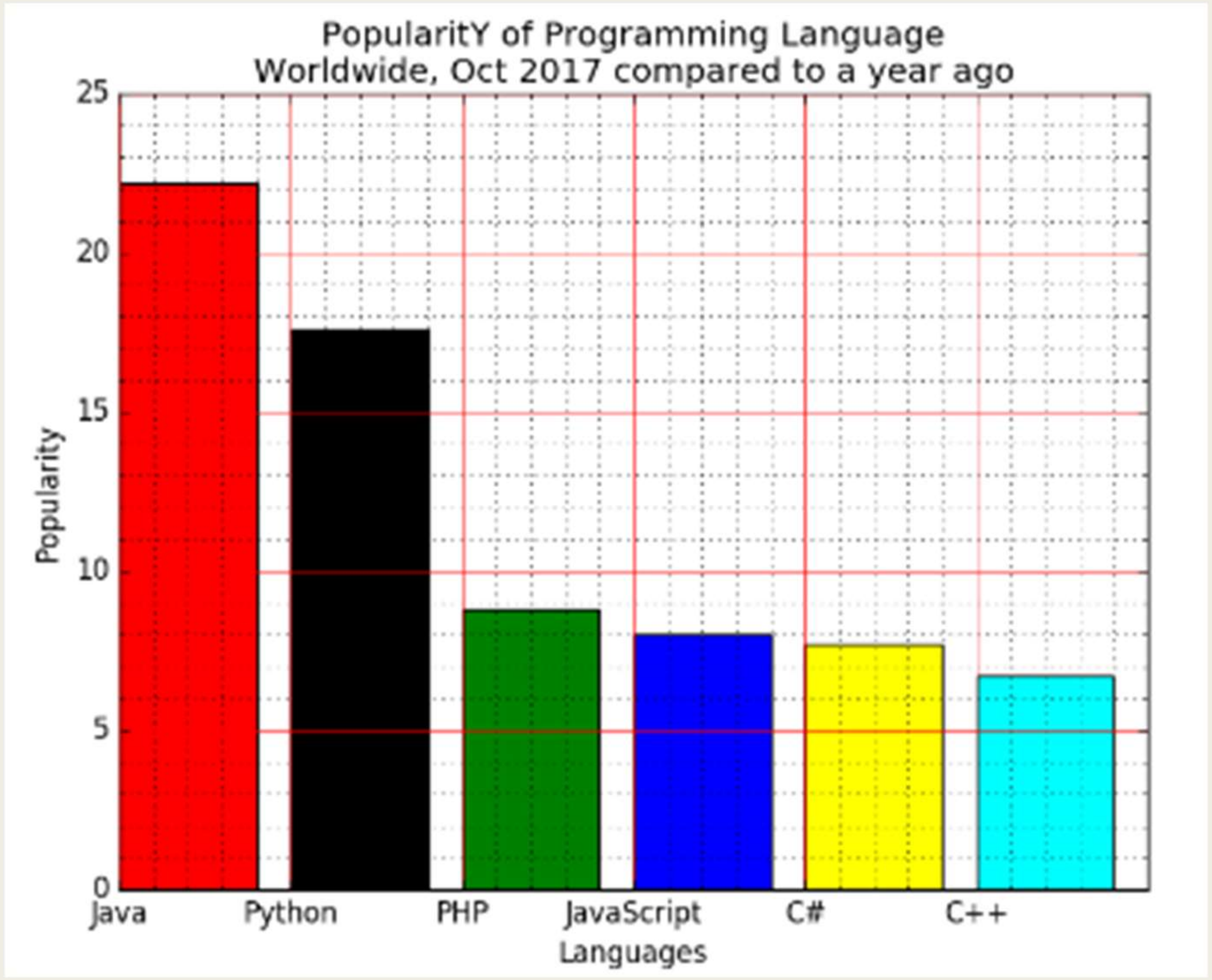
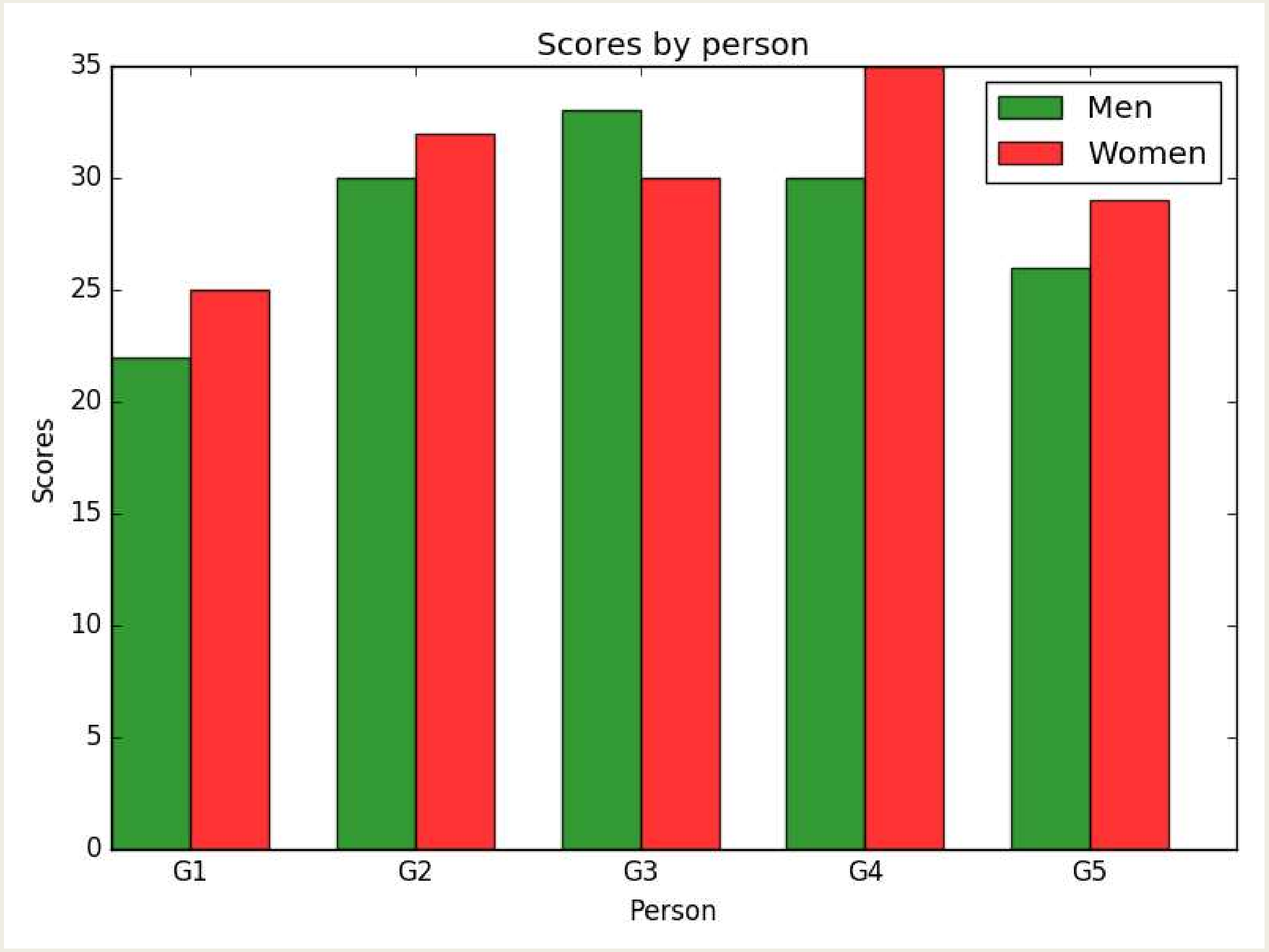


Figure 1
0
5
10
15
20
25
30
35
40
0.0
0.5
1.0
1.5
2.0
2.5


PopularitY of Programming Language
Worldwide, Oct 2017 compared to a year ago
Popularity
0
5
10
15
20
25
Java
Python
PHP
JavaScript
C#
C++
Languages


PopularitY of Programming Language
Worldwide, Oct 2017 compared to a year ago
Popularity
Languages
0
5
10
15
20
25
Java
Python
PHP
JavaScript
C#
C++


Scores by person
Men
Women
Scores
Person
G1
G2
G3
G4
G5
0
5
10
15
20
25
30
35

Histograms

- plt.hist(list_of_values)
- https://numpy.org/doc/1.16/reference/routines.random.html

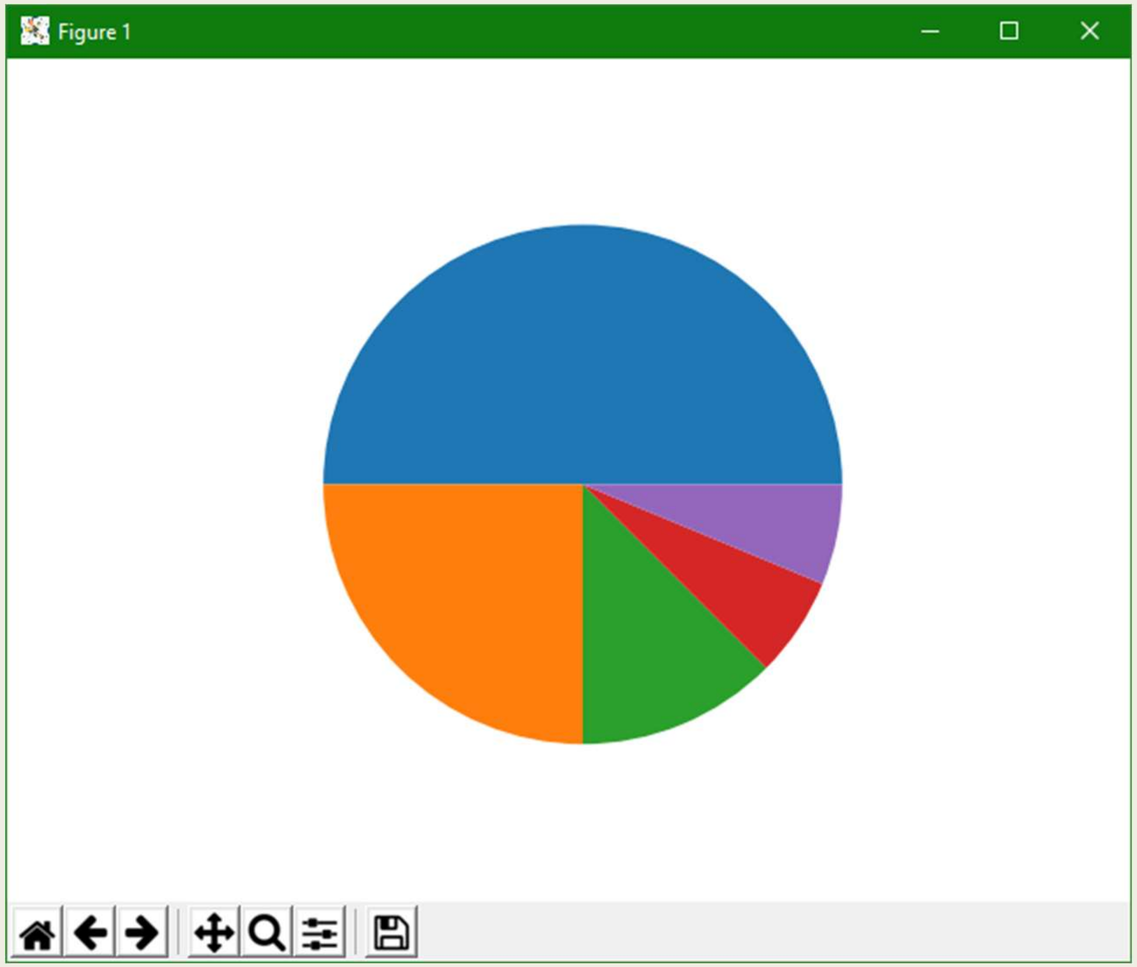
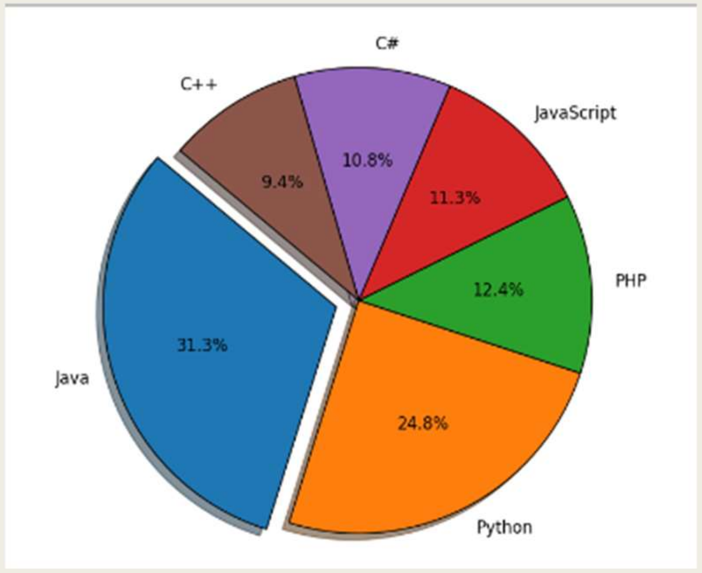


Figure 1

Pie charts

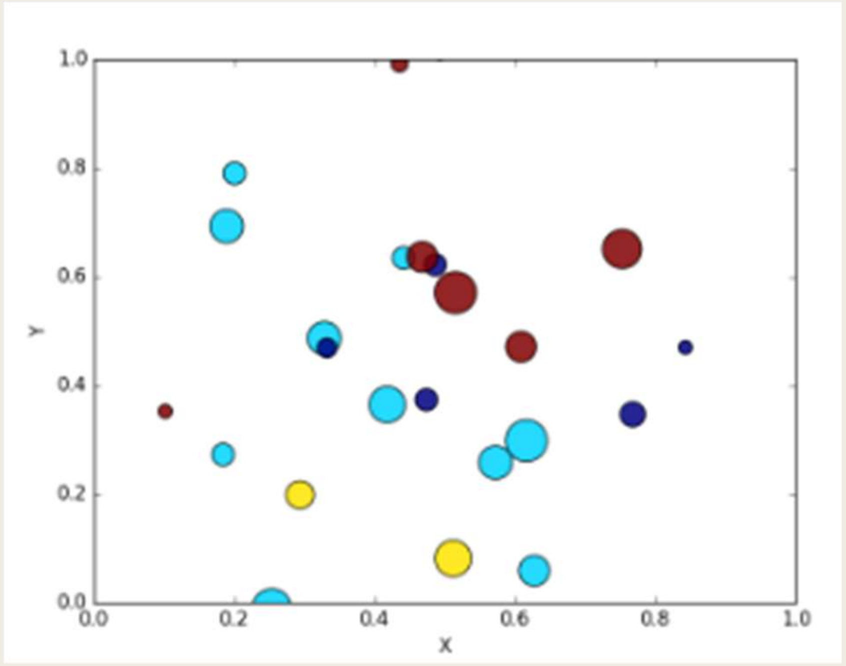
- plt.pie(values, explode, labels, colors)



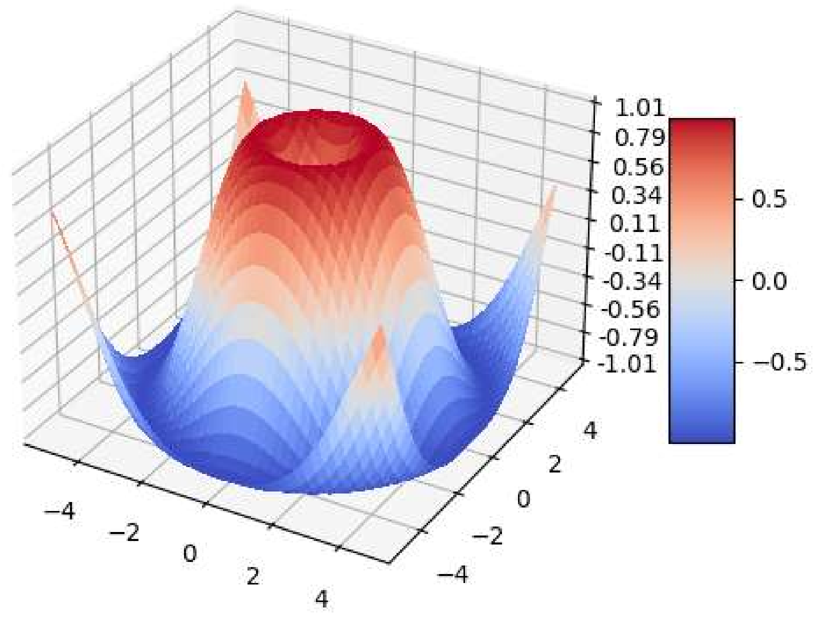
Scatter

- `plt.scatter(x, y, s=areas, c=colors, alpha=0.85)`

https://numpy.org/doc/1.16/reference/routines.random.html



3d plotting



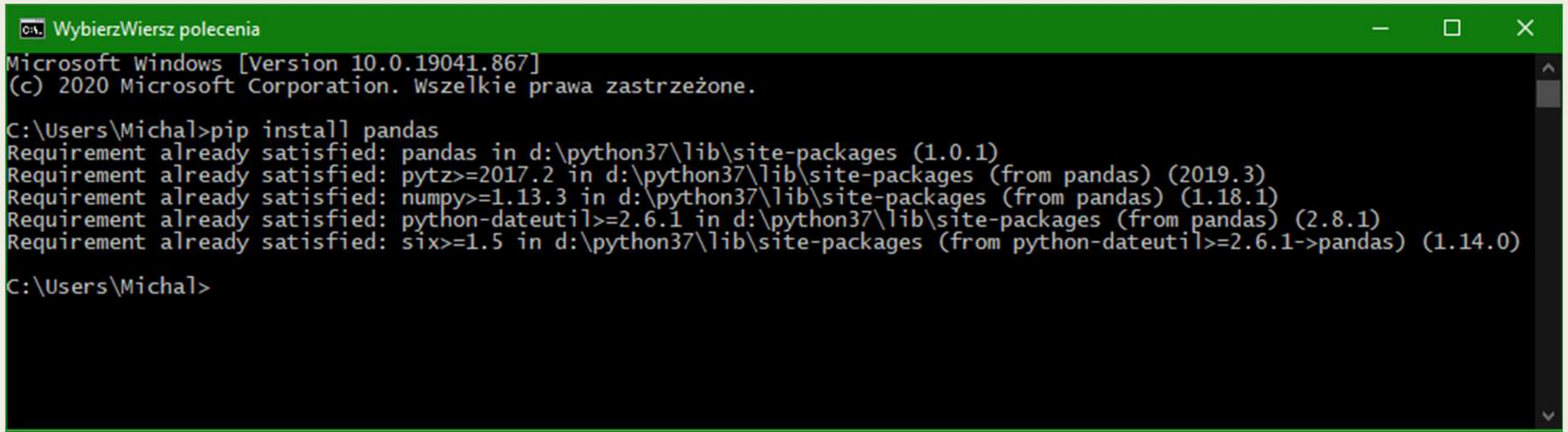
- https://www.w3resource.com/graphics/matplotlib/barchart/matplotlib-barchart-exercise-1.php

PANDAS

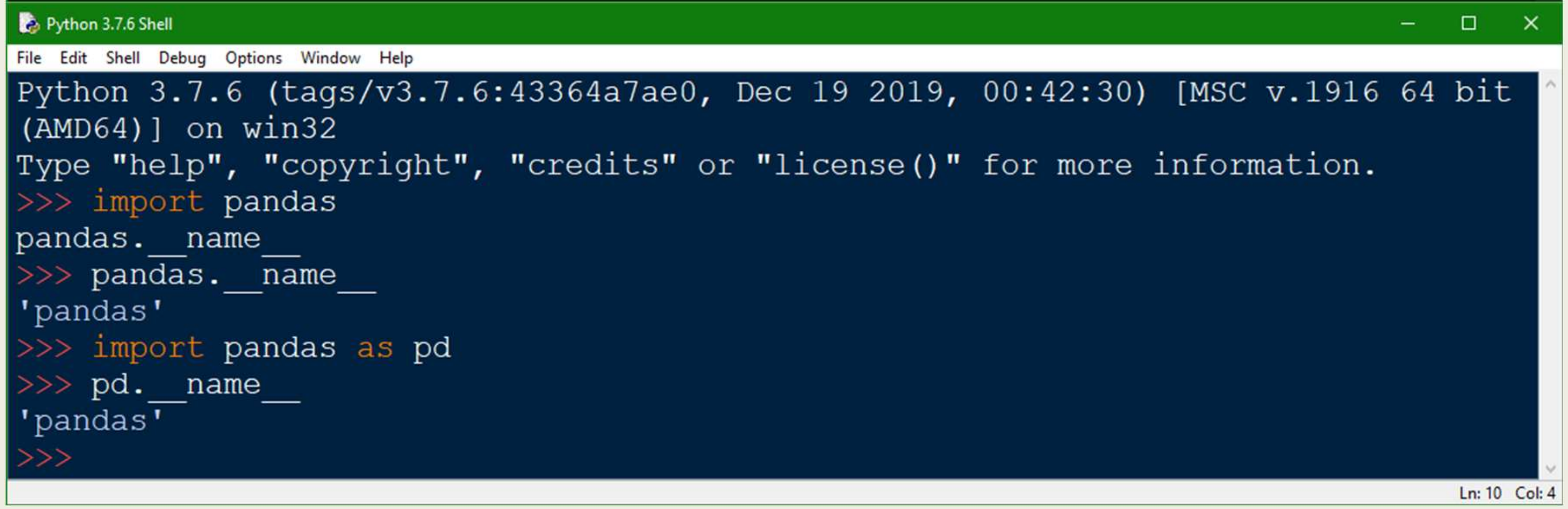
What is Panda?

- Library for Python to manage data sets
- It is possible to make analysis, do some cleaning, change of data etc.
- Pandas name is based on:
  - *„Panel data”*
  - *„Python Data Analysis”*
- The creater of the Pandas is Wes McKinney
- First release was in 2008
- https://github.com/pandas-dev/pandas

First steps… as always:



```
Microsoft Windows [Version 10.0.19041.867]
(c) 2020 Microsoft Corporation. Wszelkie prawa zastrzeżone.

C:\Users\Michal>pip install pandas
Requirement already satisfied: pandas in d:\python37\lib\site-packages (1.0.1)
Requirement already satisfied: pytz>=2017.2 in d:\python37\lib\site-packages (from pandas) (2019.3)
Requirement already satisfied: numpy>=1.13.3 in d:\python37\lib\site-packages (from pandas) (1.18.1)
Requirement already satisfied: python-dateutil>=2.6.1 in d:\python37\lib\site-packages (from pandas) (2.8.1)
Requirement already satisfied: six>=1.5 in d:\python37\lib\site-packages (from python-dateutil>=2.6.1->pandas) (1.14.0)

C:\Users\Michal>
```



```
Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import pandas
pandas.__name__
>>> pandas.__name__
'pandas'
>>> import pandas as pd
>>> pd.__name__
'pandas'
>>>
```

Pandas Series – basic data structure

- Pandas series can be understand as a column
- It is also similar in some way to Python Standard Library list – it can hold data of any type. It can be created from the list by using Series() function
- Pandas series is indexed
  - *By deflaut the indexes are consectutive numbers*

pd.Series()

- pd.Series(list)
- pd.Series(list, index = [list])
- pd.Series(dictionary)

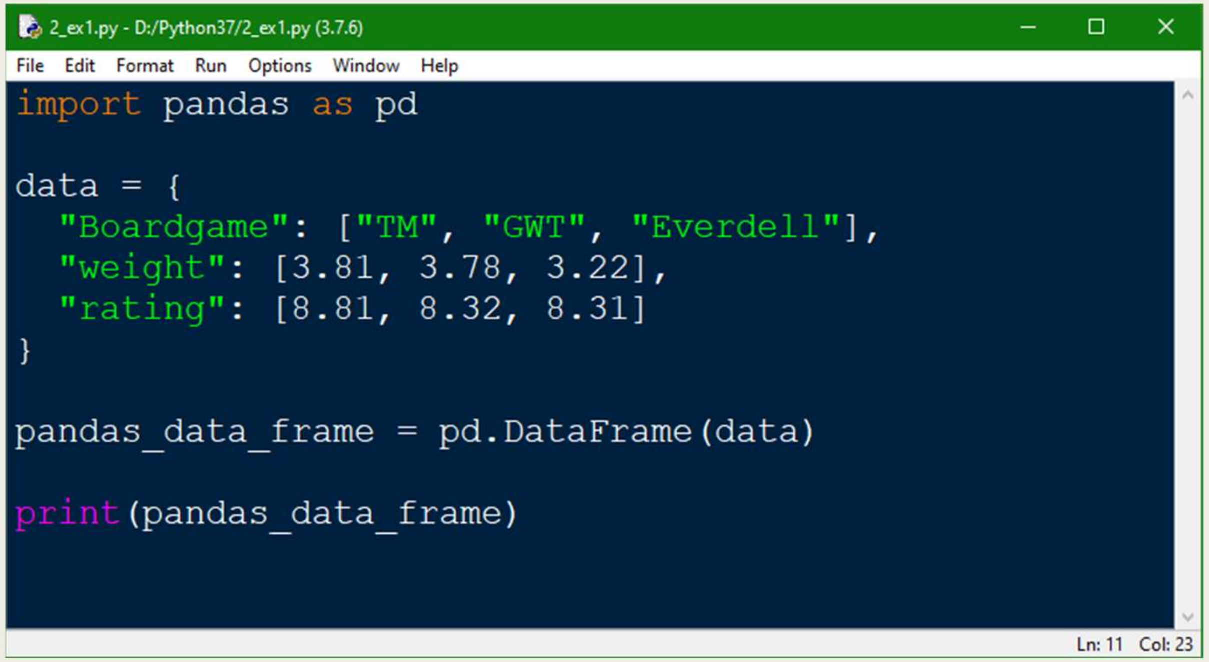
Default labels are consecutive numbers

If you label your data you can access data cel by these indices

Keys from dictionary will become labels

pd.DataFrame

- In practice main data structure during usage of Pandas. It represents multidimensional tables. It is build from series


2_ex1.py - D:/Python37/2_ex1.py (3.7.6)
File Edit Format Run Options Window Help
Ln: 11 Col: 23

```
import pandas as pd

data = {
  "Boardgame": ["TM", "GWT", "Everdell"],
  "weight": [3.81, 3.78, 3.22],
  "rating": [8.81, 8.32, 8.31]
}

pandas_data_frame = pd.DataFrame(data)

print(pandas_data_frame)
```

Define data from csv or json

- pd.read_csv(‚filename.csv’)

- pd.read_json(‚filename.json’)

JSON have sama structure
as Python dictionary

- You can use print(dataframe.to_string()) to print whole DataFrame

What we can do with DataFrame

- loc[]🡐 return one or more rows
  - *loc[number]*
  - *loc[[list of rows]]*
- dataframe.rename(columns={„old” : „new”})

You can modify the data in this way, however it is not efficient way to do this

Repair the data

You can modify current dataframe if you use inplace = True parameter

- df.dropna() <- removes the empty cells – the effect is new dataframe

- df.fillna() <- fills rows with data – the effect is new dataframe

- for x in dataframe.index:

  do something

  dataframe.loc[x, „column”] = True

  dataframe.drop(x) if …

Analyze the data

- dataframe.mean()
- dataframe.median()
- dataframe.mode()

Can be call for whole dataframe or just for one column

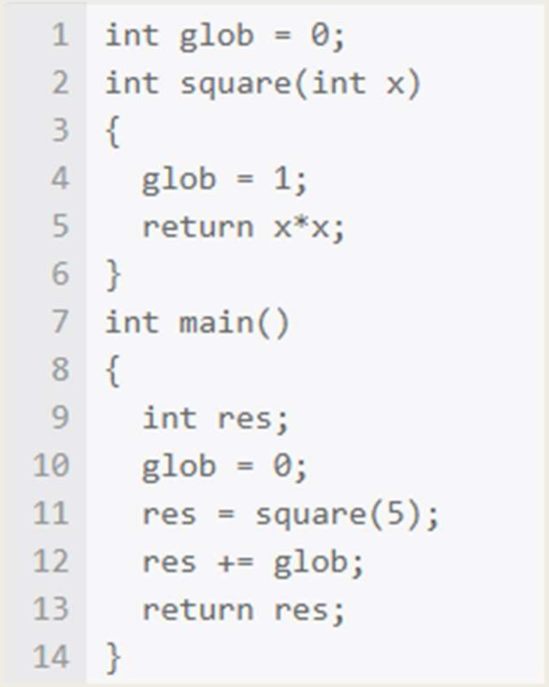
Warm – up

- 2 ** 3 ** 2
- 33 * (5 // 6)

Side effects

- Any operation that modify the state of the computer or which interacts with the outside world
- Examples to think about:
  - *Sleep()*
  - *int dbl(int x) {return 2*x}*
  - *Str wrt(str sth) {print(sth) return true}*

Why side-effects can be bad?



```
int glob = 0;
int square(int x)
{
  glob = 1;
  return x*x;
}
int main()
{
  int res;
  glob = 0;
  res = square(5);
  res += glob;
  return res;
}
```

Scope of variables

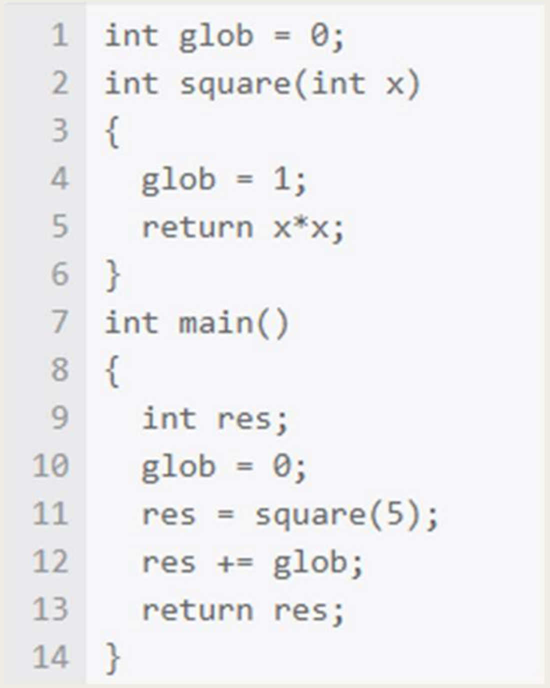
- Local scope – the variable is restricted to the function where it is created
- Global scope – the variable created out of the any function can be accessed from any point of code

You can have two variables with the same name – one global and one local – the local one is shadowing global one

- You can make global variable within function if you use *global* keyword
  - *global x = 2*

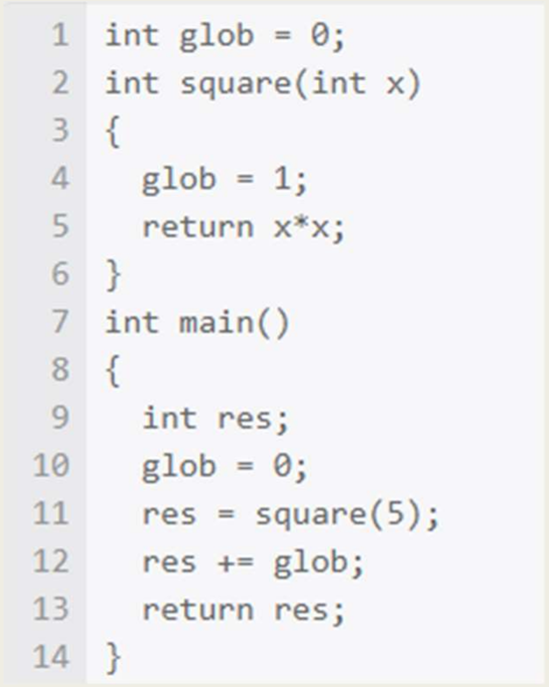
You can use the same approach to changing global variable inside function

Why side-effects can be bad?

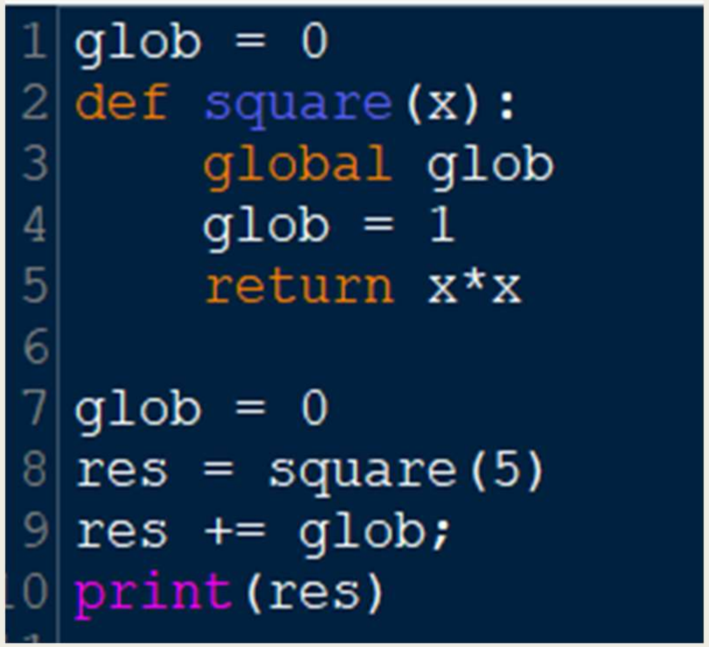


```
int glob = 0;
int square(int x)
{
  glob = 1;
  return x*x;
}
int main()
{
  int res;
  glob = 0;
  res = square(5);
  res += glob;
  return res;
}
```

Why side-effects can be bad?

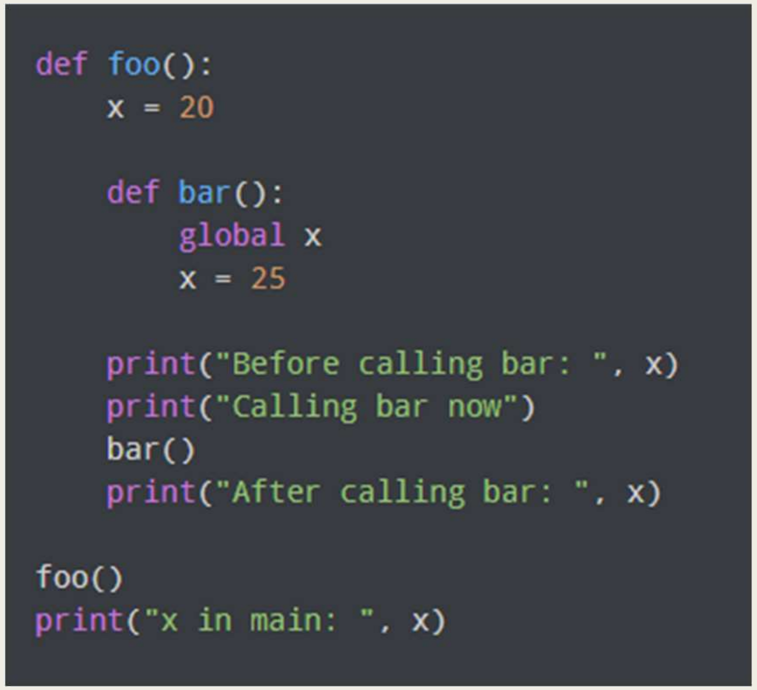


```c
int glob = 0;
int square(int x)
{
  glob = 1;
  return x*x;
}
int main()
{
  int res;
  glob = 0;
  res = square(5);
  res += glob;
  return res;
}
```



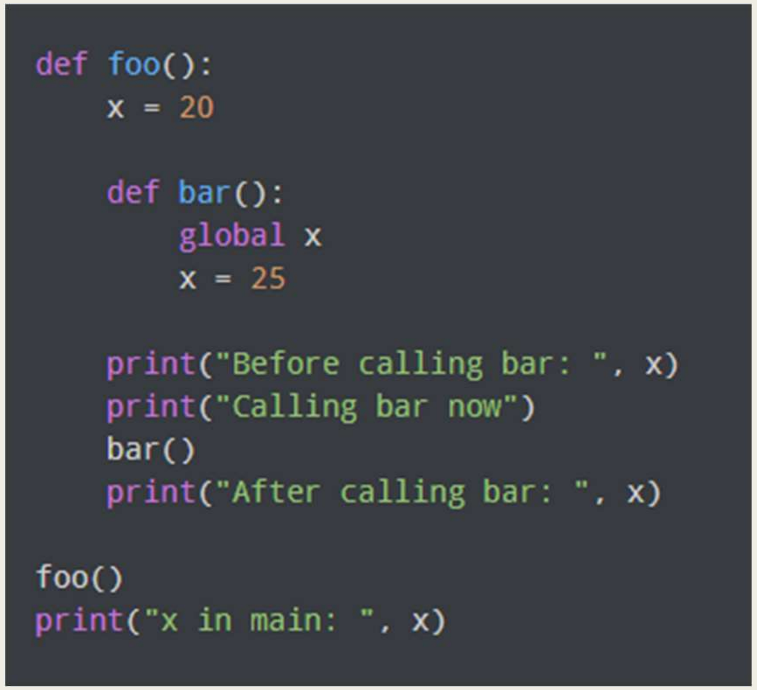
```python
glob = 0
def square(x):
    global glob
    glob = 1
    return x*x

glob = 0
res = square(5)
res += glob;
print(res)
```

More about scopes and global

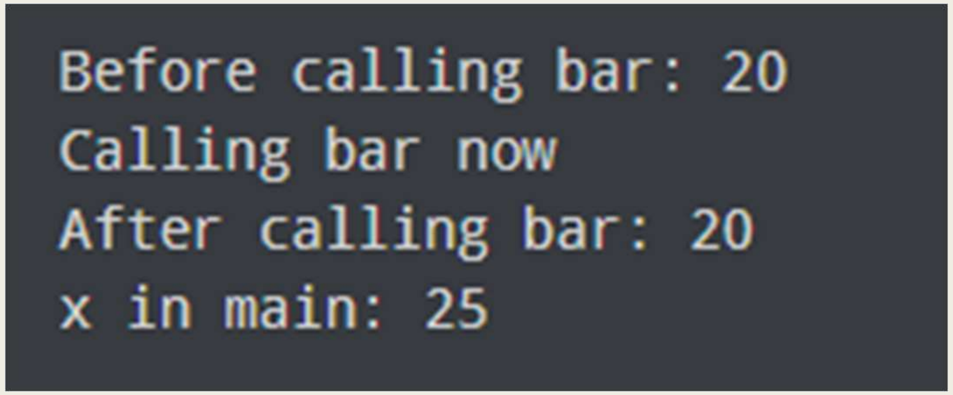


```python
def foo():
    x = 20

    def bar():
        global x
        x = 25

    print("Before calling bar: ", x)
    print("Calling bar now")
    bar()
    print("After calling bar: ", x)

foo()
print("x in main: ", x)
```

More about scopes and global



```python
def foo():
    x = 20

    def bar():
        global x
        x = 25

    print("Before calling bar: ", x)
    print("Calling bar now")
    bar()
    print("After calling bar: ", x)

foo()
print("x in main: ", x)
```



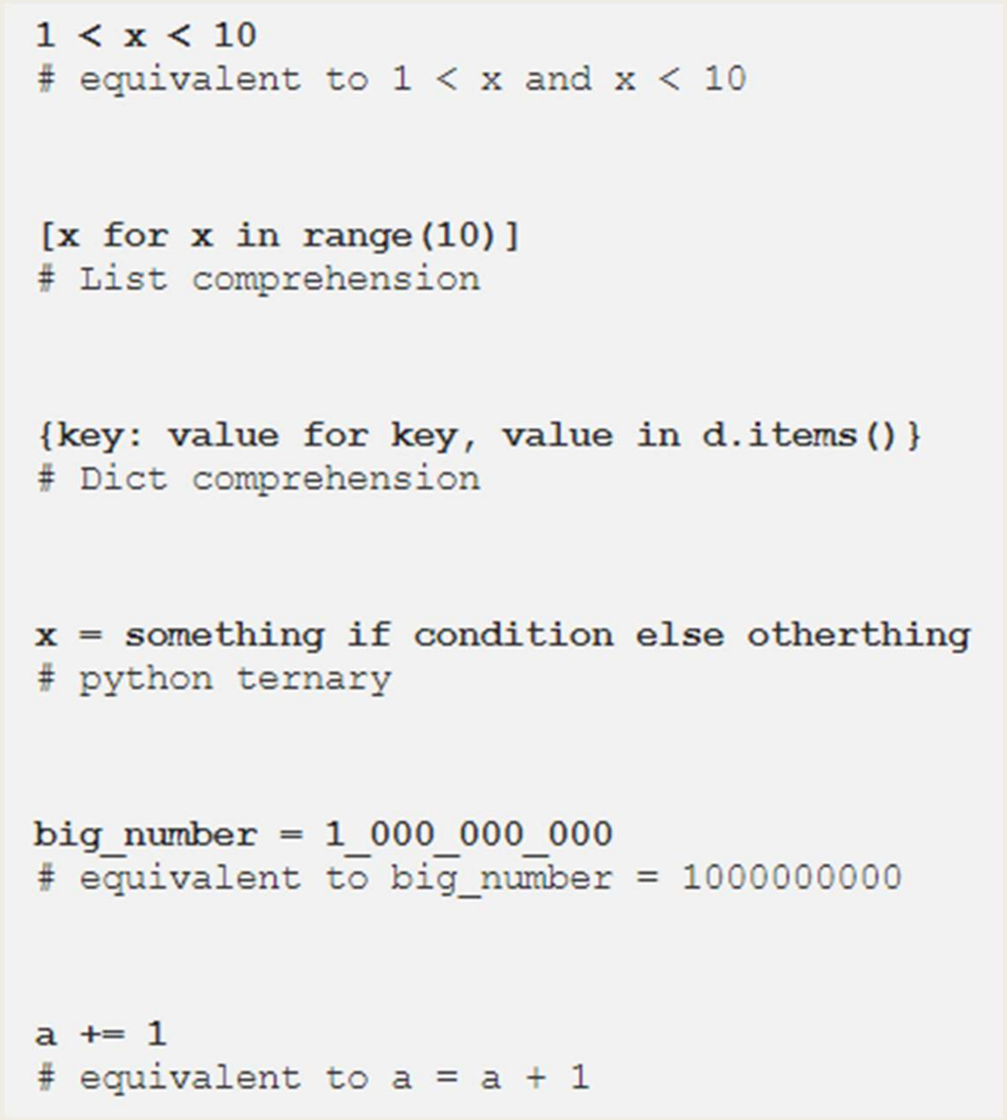
```
Before calling bar: 20
Calling bar now
After calling bar: 20
x in main: 25
```

Syntactic sugar

- Syntactic sugar, or syntax sugar, is a visually or logically-appealing "shortcut" provided by the language, which reduces the amount of code that must be written in some common situation.

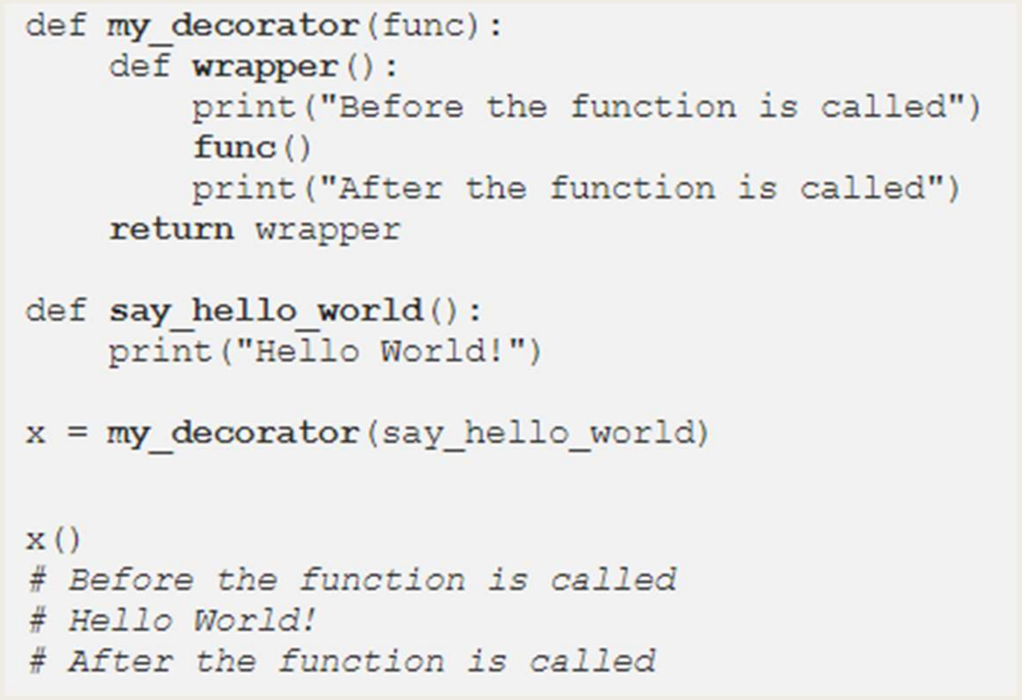
- # class newClass = WhateverClass()
  - *# call __new__ and __init__ method*
- num_list = [1,2,3,4]
  - *print(1 in num_list)*
  - *print(num_list.__contains__(1))*
  - *print(len(num_list))*
  - *print(num_list.__len__())*

Syntactic sugar



```
1 < x < 10
# equivalent to 1 < x and x < 10


[x for x in range(10)]
# List comprehension


{key: value for key, value in d.items()}
# Dict comprehension


x = something if condition else otherthing
# python ternary


big_number = 1_000_000_000
# equivalent to big_number = 1000000000


a += 1
# equivalent to a = a + 1
```

Decorators



```python
def my_decorator(func):
    def wrapper():
        print("Before the function is called")
        func()
        print("After the function is called")
    return wrapper

def say_hello_world():
    print("Hello World!")

x = my_decorator(say_hello_world)


x()
# Before the function is called
# Hello World!
# After the function is called
```