How to program with python™ programming language (Part 2)

Charlotte HÉRICÉ
Sakata lab - SIPBS

December 2017

Overview

1. Introduction
2. Variables
3. Lists
4. Operations
5. Boolean operators
6. Loops
7. Dictionaries
8. Functions
9. Modules
10. Matplotlib
11. Write/read data files
12. Applications

7. Dictionaries

Creation

A dictionary is a data type similar to the lists, but works with keys and values instead of indexes. Each value stored in the dictionary can be accessed using its key, which can be a string, a number, a list …
A list a represented by `[]` and dictionary by `{}`

```
dico = dict()
print(type(dico))
print(dico)

dico = {}
print(dico)
```

7. Dictionaries

Creation

```
dico = {}
dico["fruit"] = "apple"
dico["vegetable"] = "carrot"
print(dico)
```

Here `fruit` and `vegetable` are keys, `apple` and `carrot` are values.

If you want to access all values of the key `fruit`:

```
print(dico["fruit"])
```

7. Dictionaries

Manipulation

```
myDict = {"rats":4, "mice":6, "monkeys":2}
print(myDict)
del myDict["mice"]
print(myDict)
```

Print all keys of the dictionary :

```
myDict = {"rats":4, "mice":6, "monkeys":2}
for animal in myDict:
    print(animal)

for animal in myDict.keys():
    print(animal)
```

7. Dictionaries

Manipulation

Print all values of the dictionary:

```
myDict = {"rats":4, "mice":6, "monkeys":2}
for number in myDict.values():
     print(number)
```

Print keys and values in the same time (2 ways):

```
for animal, number in myDict.items():
     print("Number of {} : {}".format(animal, number))
     print("Number of %s : %d" % (animal, number))
```

7. Dictionaries

Manipulation

```
myDict = {"rats":4, "mice":6, "monkeys":2}
```

To test if a value/key is in the dictionary :

```
if ("rats" in myDict):
     print("You have rats in your lab")
if ("rats" not in myDict):
     print("You don’t have rats in your lab")
if (6 in myDict.values()):
     print("One group of animals has 6")
```

7. Dictionaries

Application

Do the same kind of program as in Part 6, but with a dictionary:

You have 2 mice and you want their total weight of the last 2 days. Create a program with a dictionary that contains a list with your 2 mouse names, then the program will ask you to give the weight for each mouse, every day. The dictionary will be organised as follow:

```
{'animal1':[weightDay1, weightDay2], 'animal2':
   [weightDay1, weightDay2]}
```

7. Dictionaries

Application

Add another loop to this program in order to print for each day, for each mouse, the weight of this mouse as in the example:

```
"At day 0, the weight of mouse mickey was 15g"
"At day 0, the weight of mouse minie was 14g"
"At day 1, the weight of mouse mickey was 16g"
"At day 2, the weight of mouse minie was 13g"
```

8. Functions

Creation

- `print(), type(), input()` are functions pre-defined by Python, but you can create your own ones.
- Functions are used to group blocks of code that we will use many times. They can have some parameters or not.

```
def functionName():
    instructions

def functionName(parameter1, parameter2):
    instructions
```

To call your function :

```
functionName()
functionName(param1, param2)
```

8. Functions

Example : Let's go with the previous example of the 7 multiplication table

Before :

```
nb = 7
i = 0 # Counter that will be incremented into the loop
while (i < 10):
      print(i+1, "*", nb, "=", (i+1)*nb)
      i += 1
```

With a function :

```
def table_7():
      nb = 7
      i = 0
      while(i < 10):
            print(i+1, "*", nb, "=", (i+1)*nb)
            i += 1

# Main
table_7()
```

8. Functions

Example : Let's go with the previous example of the 7 multiplication table

⇨ Now you can have the multiplication table for any number, just by calling the function with this number for parameter.

```
def table(nb):
      i = 0
      while(i < 10):
            print(i+1, "*", nb, "=", (i+1)*nb)
            i += 1

# Main
table(7)
table(8)
```

8. Functions

Example : Let's go with the previous example of the 7 multiplication table

You can also consider the number of values to be displayed in the table as an other parameter :

```
def table(nb, maxVal):
      i = 0
      while(i < maxVal):
            print(i+1, "*", nb, "=", (i+1)*nb)
            i += 1

# Main
table(7, 10) # Will display 10 first values of 7 mult. table
table(8, 20) # Will display 20 first values of 8 mult. table
```

8. Functions

Other examples

Some functions may return a value to the caller, using the keyword `return`.

```
def multiplication(a, b):
    return (a * b)

def square(a):
    return (a * a)

# Main
c = multiplication(5, 6)
print(c)
d = square(3)
print(d)
```

8. Functions

Exercise

Write a program with a function that prints "yes" is a int is divisible by 10 but not by 3 and prints "no" for the other cases.

You will test this function with the int 30, 33, 37 and 70.

8. Functions

Exercise

Modify this program in oder to print the following example :

```
70 is divisible by 10 but not 3 => yes
33 is not divisible by 10 and but is divisible by 3 => no
30 is divisible by 10 and 3 => no
37 is divisible neither by 10 nor 3 => no
```

9. Modules

In general

So far, we have been working with Python pre-built functions. But you can load modules and packages to be able to use specific librairies, tools developed by other programmers.

A module in Python is simply a piece of code in a file with the `.py` extension. This code will implement a set of functions and you can import the module to use these functions with the keyword `import`. Then to use a function of the module, it is : `moduleName.functionName()`

Example with the Python `math` module :

```
import math

print(math.sqrt(16))
```

→ Here we use the function `sqrt()` of the module `math`.

9. Modules

In general

- All imports have to be written in the first lines of a script
- If you want to know all the available methods of a module : `help("moduleName")` or https://docs.python.org/3/library/

- If the name of the module is too long, or just if you do not want to write it all the time, you can re-name it with the keyword `as`.

```
import math as mt

print(mt.sqrt(16))
```

→ The module is "stored" in the `mt` instead of `math`

9. Modules

Other import method

Other method to import a module : `from ... import ...`

```
from math import sqrt

print(sqrt(16))
```

→ Here we imported only the function `sqrt` from `math`.

If we want to import all the functions :

```
from math import *

print(sqrt(25))
```

10. Matplotlib

Presentation

- One of the most used Python package for 2D graphs
- Fast way to visualise data
- High quality illustrations in various formats

Matplotlib is included in the `pylab` module :

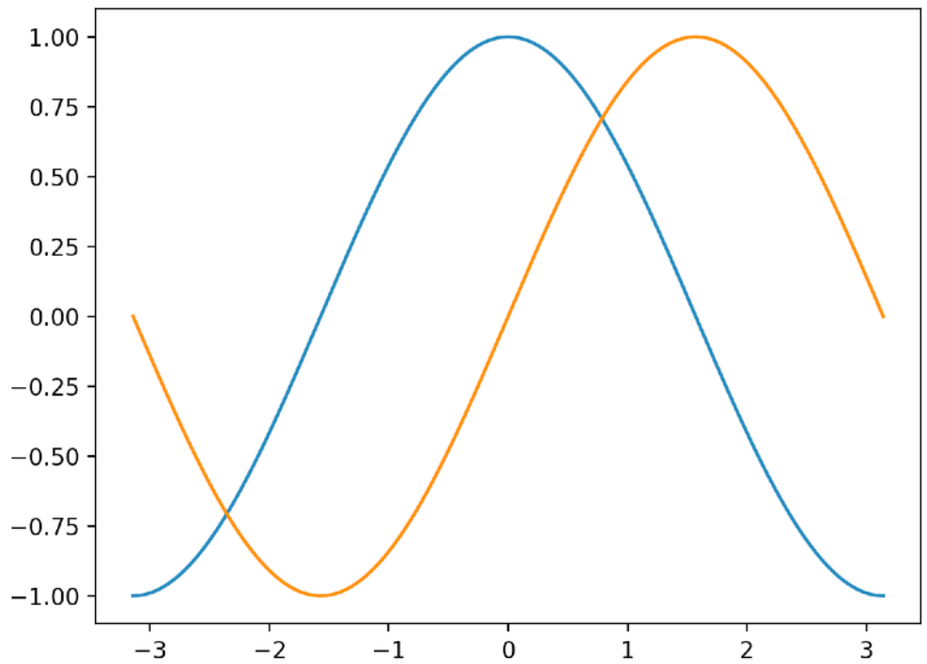
```
from pylab import *
import numpy as np
```

We will also need:
`np.linspace(start, end, nbValues, endpoint)` → Returns `num` evenly spaced samples, calculated over the interval `[start, stop]`. If `endpoint` is True, stop is included in the last sample.

10. Matplotlib

Simple graphs

Example of **cosinus and sinus functions** represented in the same graph :




```
from pylab import *
import numpy as np

X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
cosX = np.cos(X)
sinX = np.sin(X)
```

→ `X` is now a bumpy array with 256 values, from -**π** to **π** (included).

```
plot (X, cosX), plot(X, sinX)
show()
```

10. Matplotlib

Simple graphs

To save the figure : `savefig("figName.extensionOfYourChoice")`

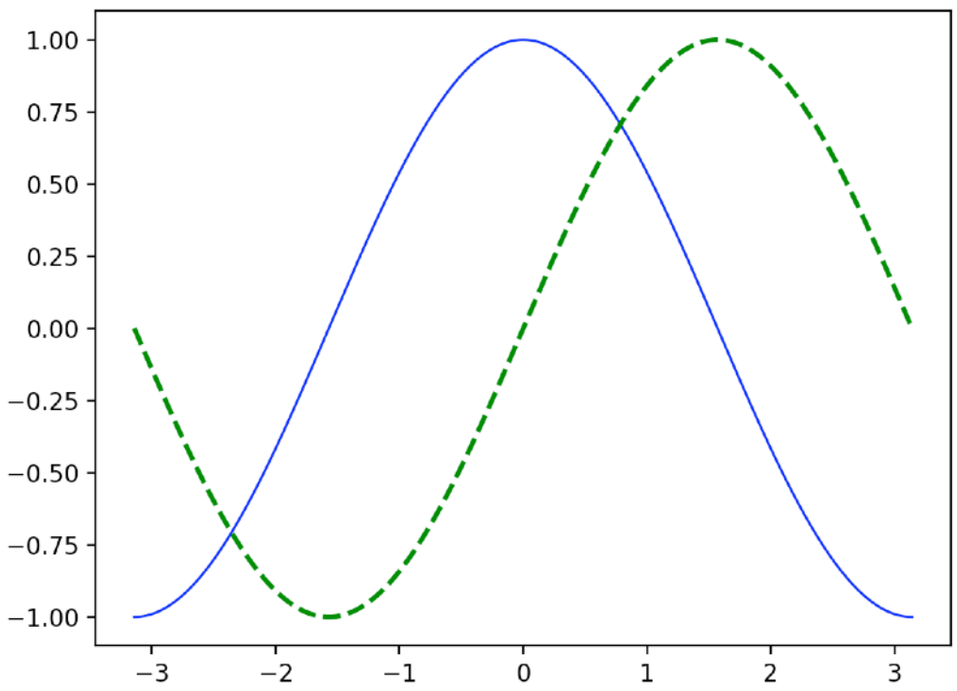
```
from pylab import *
import numpy as np

X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
cosX = np.cos(X)
sinX = np.sin(X)

figure(figsize=(8,6), dpi=80)

plot(X, cosX, color="blue", linewidth=1.0, linestyle="-")
plot(X, sinX, color="green", linewidth=2.0, linestyle="—")

savefig("fig1.png")
show()
```

10. Matplotlib

Simple graphs

Change colour, width and style of the line :

```
plot(X, cosX, color="green", linewidth=1.0, linestyle="-")
plot(X, sinX, color="blue", linewidth=2.0, linestyle="--")
```



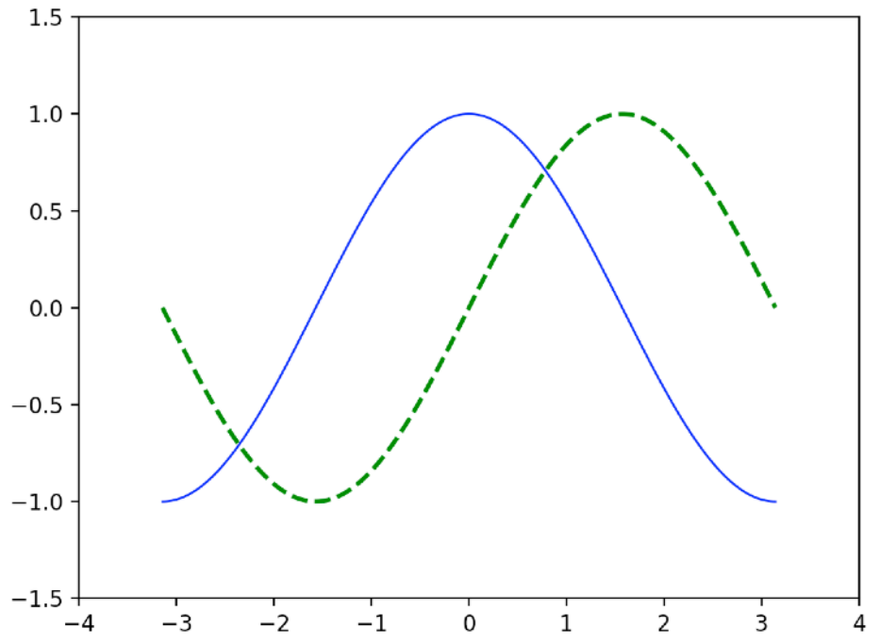
10. Matplotlib

Simple graphs

Change x and y limits :

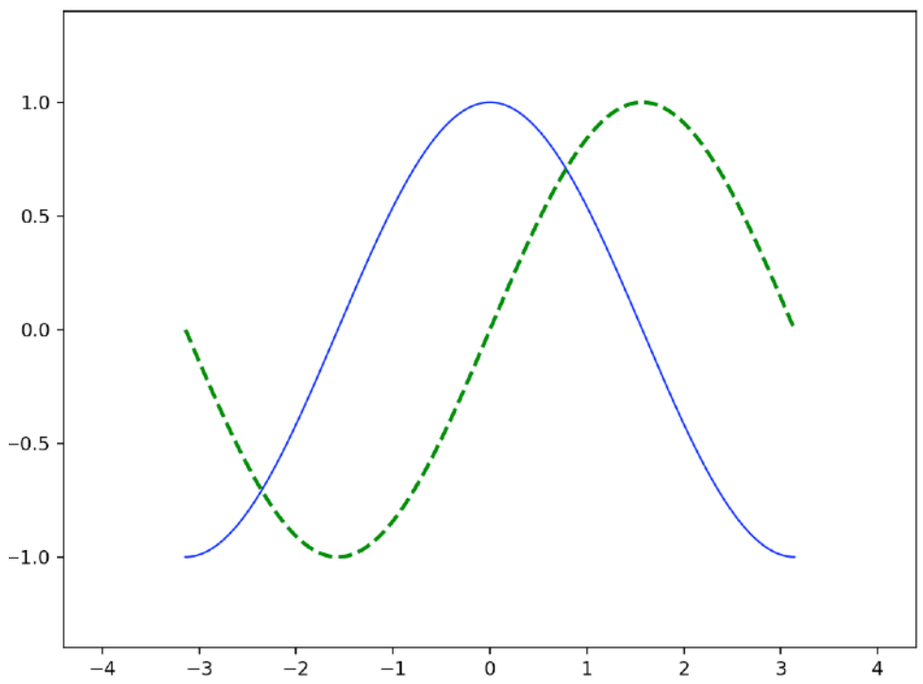
Manually :

```
xlim(-4.0, 4.0)
ylim(-1.5, 1.5)
```



Automatically :

```
xmin, xmax = X.min(), X.max()
ymin, ymax = cosX.min(), cosX.max()
dx = (xmax - xmin) * 0.2
dy = (ymax - ymin) * 0.2
xlim(xmin - dx, xmax + dx)
ylim(ymin - dy, ymax + dy)
```



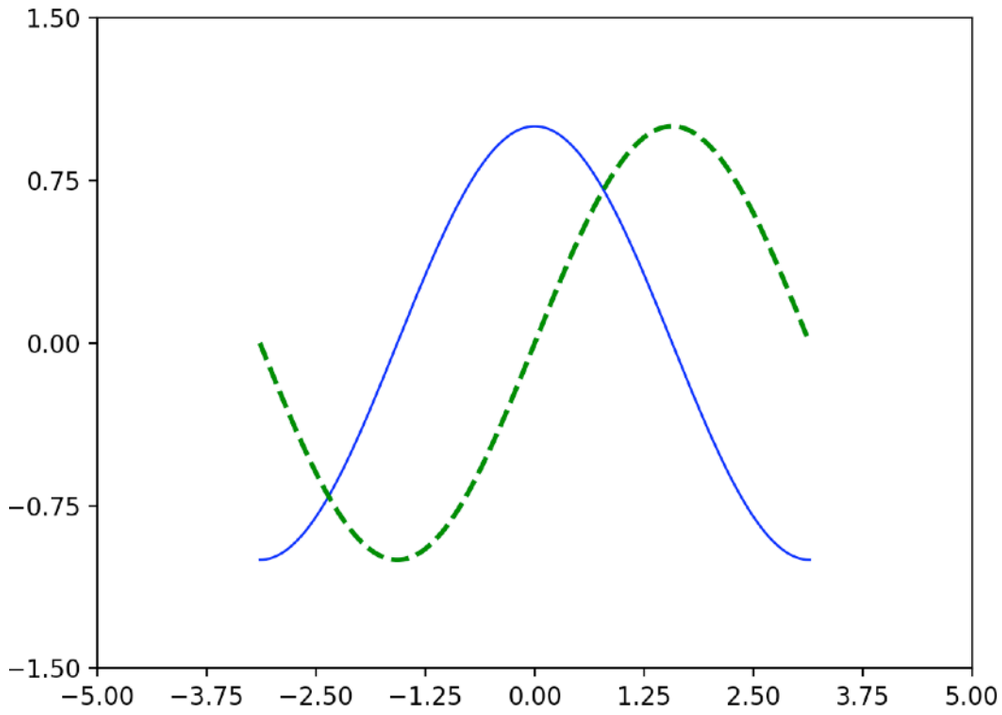
10. Matplotlib

Simple graphs

Change the graduations :

Manually :

```
xticks(np.linspace(-5, 5, 9, endpoint=True))
yticks(np.linspace(-1.5, 1.5, 5, endpoint=True))
```



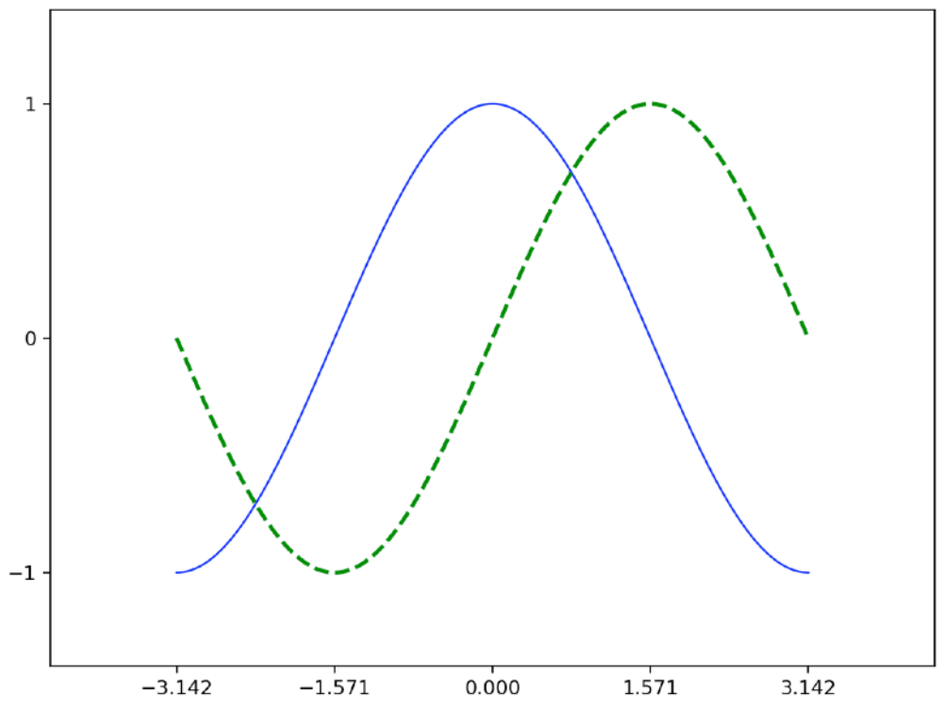
10. Matplotlib

Simple graphs

Change the graduations :

Automatically :

```
xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi])
yticks([-1, 0, +1])
```



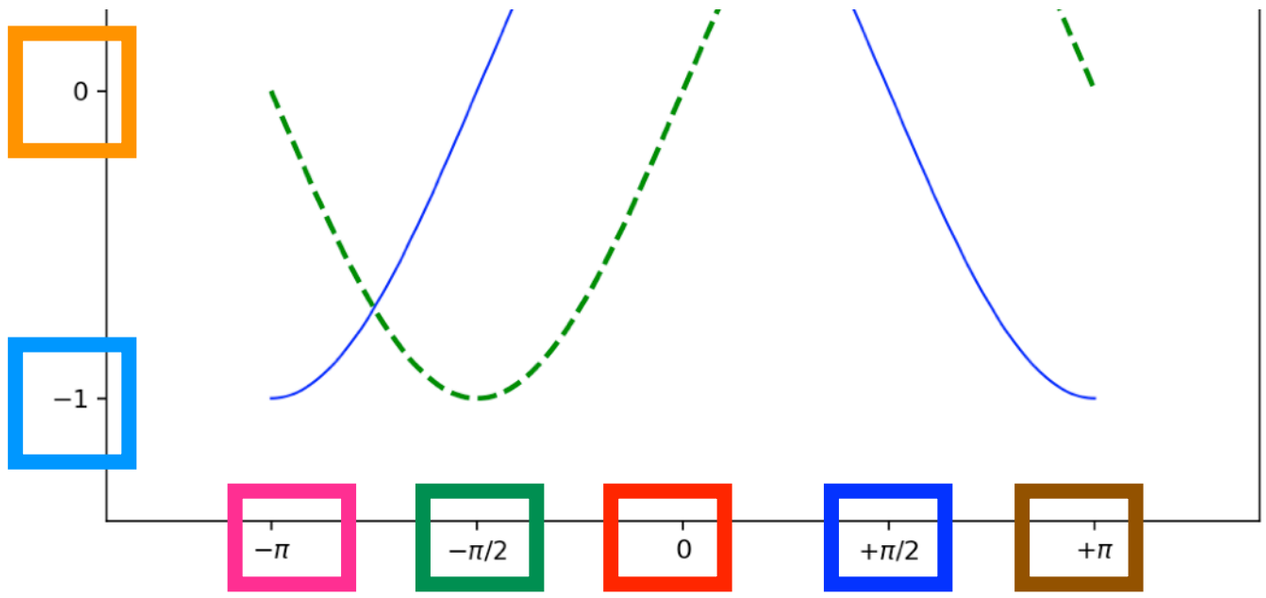
→ Now we want π instead of 3.142

10. Matplotlib

Simple graphs

Change the text of the graduations :

```
xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
       [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$',
        r'$+\pi$'])

yticks([-1, 0, +1],[r'$-1$', r'$0$', r'$+1$'])
```



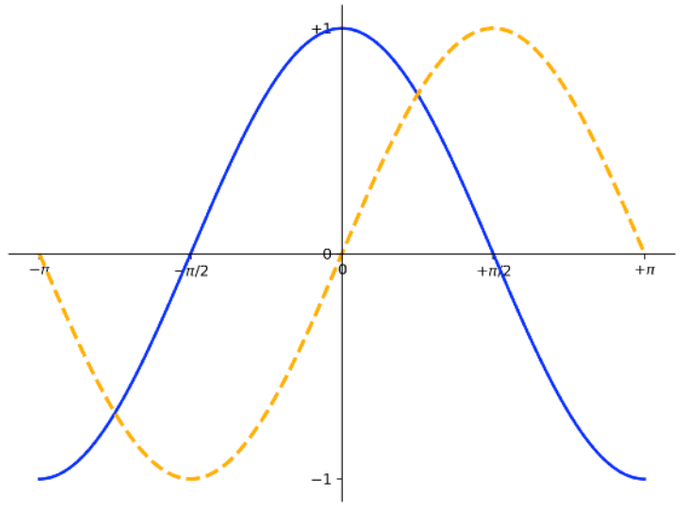
10. Matplotlib

Simple graphs

Change the location of the axis

```
plot(X, cosX, color="blue", linewidth=2.0, linestyle="-")
plot(X, sinX, color="orange", linewidth=2.5, linestyle="--")
```

```
ax = gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data', 0)
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data', 0))
```



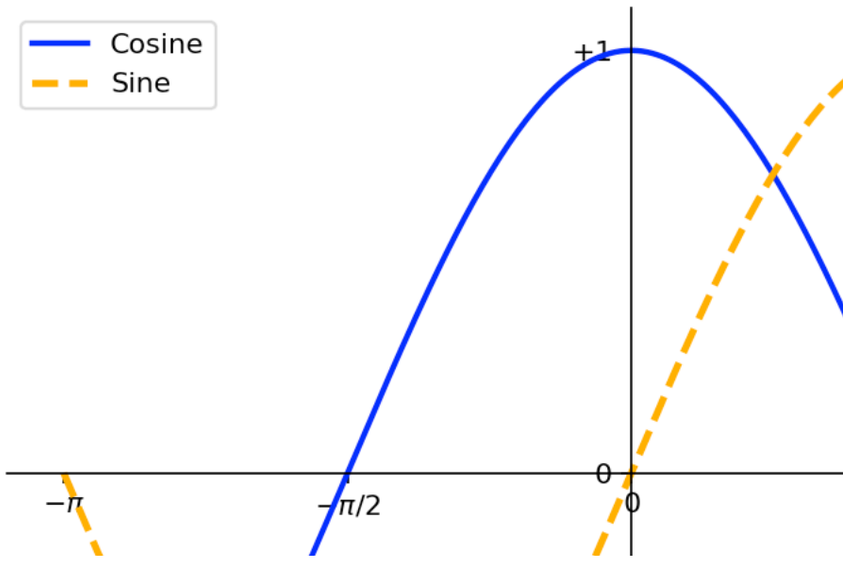

```
xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi], [r'$-\pi$',
r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'])
yticks([-1, 0, +1], [r'$-1$', r'$0$', r'$+1$'])
```

10. Matplotlib

Simple graphs

Add a legend :

```
plot(X, cosX, color="blue", linewidth=2.0,
      linestyle="-", label="Cosine")
plot(X, sinX, color="orange", linewidth=2.5,
      linestyle="--", label="Sine")
...
legend(loc='upper left')
show()
```

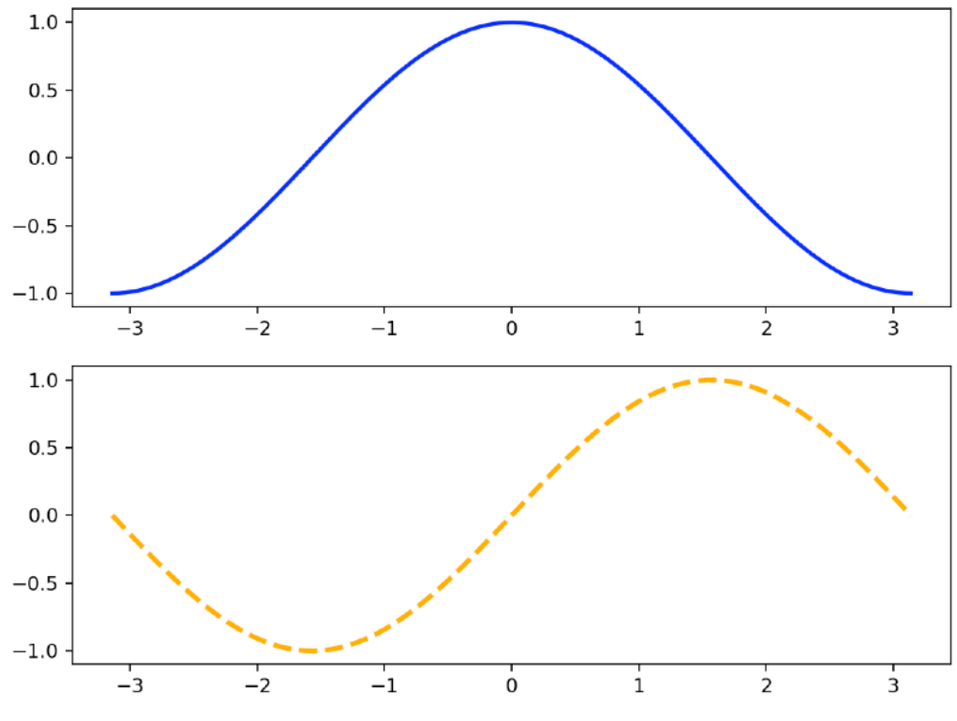


10. Matplotlib

Multi-plots

If you want the cosine and the sine in 2 different plots, but on the same figure :

`subplot(nbRows, nbCol, order)`



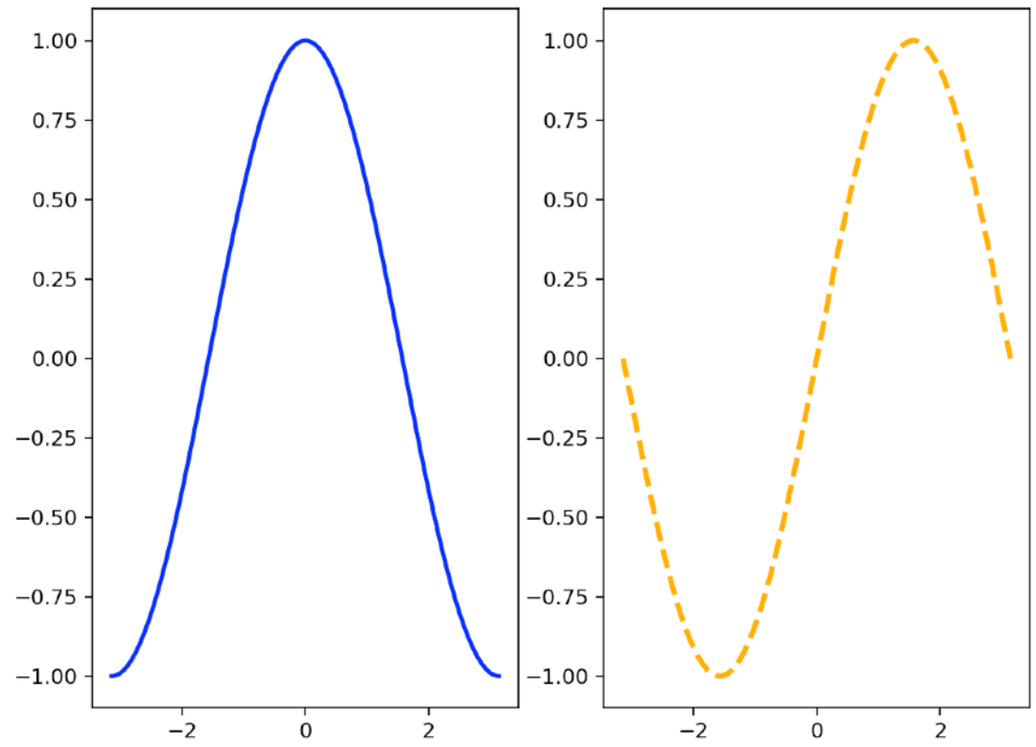

```
subplot(2, 1, 1)
plot(X, cosX, color="blue", linewidth=2.0,
     linestyle="-", label="Cosine")
subplot(2, 1, 2)
plot(X, sinX, color="orange", linewidth=2.5,
     linestyle="--", label="Sine")
```

10. Matplotlib

Multi-plots

If you want the cosine and the sine in 2 different plots, but on the same figure :

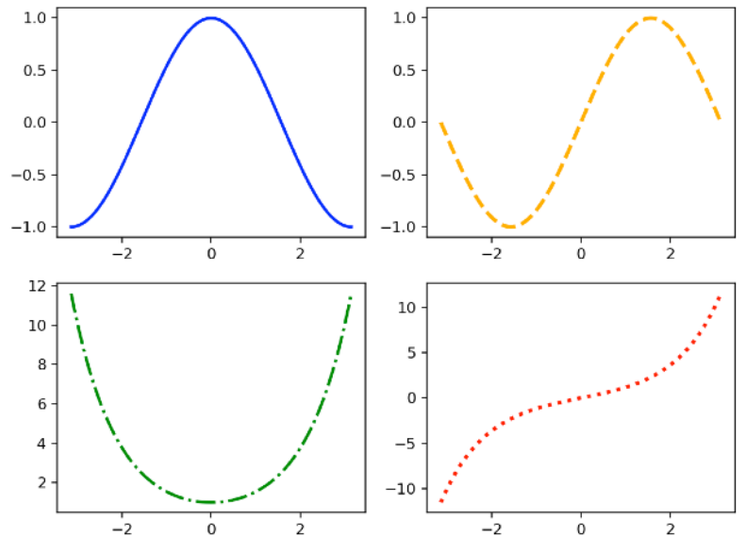
`subplot(nbRows, nbCol, order)`




```
subplot(1, 2, 1)
plot(X, cosX, color="blue", linewidth=2.0,
     linestyle="-", label="Cosine")
subplot(1, 2, 2)
plot(X, sinX, color="orange", linewidth=2.5,
     linestyle="--", label="Sine")
```

10. Matplotlib

Multi-plots



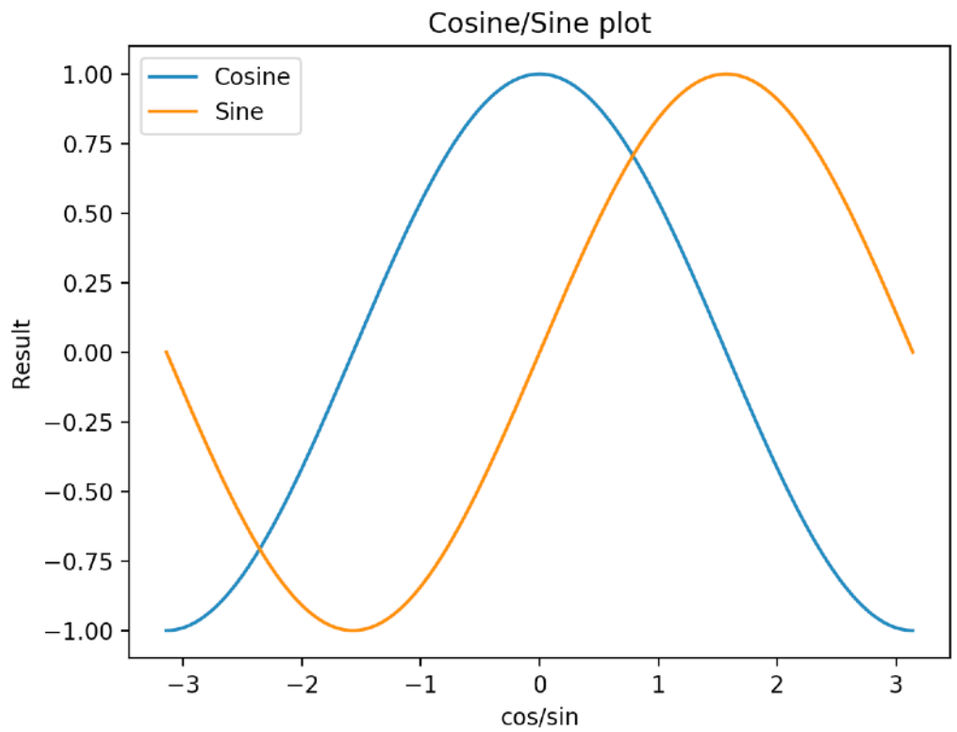

```
subplot(2, 2, 1)
plot(X, cosX, color="blue", linewidth=2.0, linestyle="-",
     label="Cosine")
subplot(2, 2, 2)
plot(X, sinX, color="orange", linewidth=2.5, linestyle="--",
     label="Sine")
subplot(2, 2, 3)
plot(X, coshX, color="green", linewidth=2.0, linestyle="-.",
     label="Hyperbolic cosine")
subplot(2, 2, 4)
plot(X, sinhX, color="red", linewidth=2.5, linestyle=":",
     label="Hyperbolic sine")
```

10. Matplotlib

Multi-plots

With legends, labels and titles :

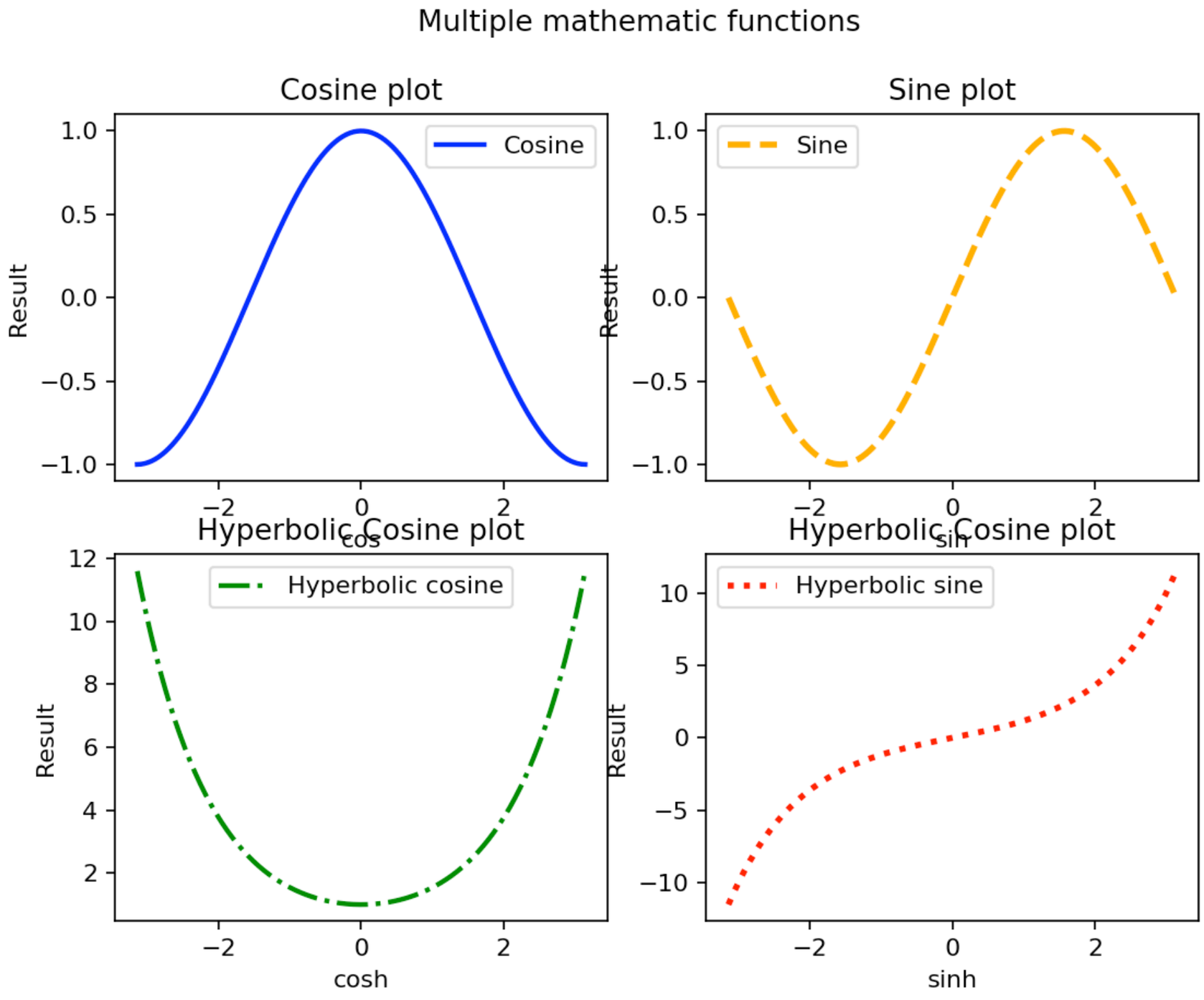
```
from pylab import *
import numpy as np

X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
cosX = np.cos(X)
sinX = np.sin(X)

plot (X, cosX, label="Cosine")
plot(X, sinX, label="Sine")
legend()
title("Cosine/Sine plot")
xlabel("cos/sin")
ylabel("Result")
show()
```



10. Matplotlib

Multi-plots

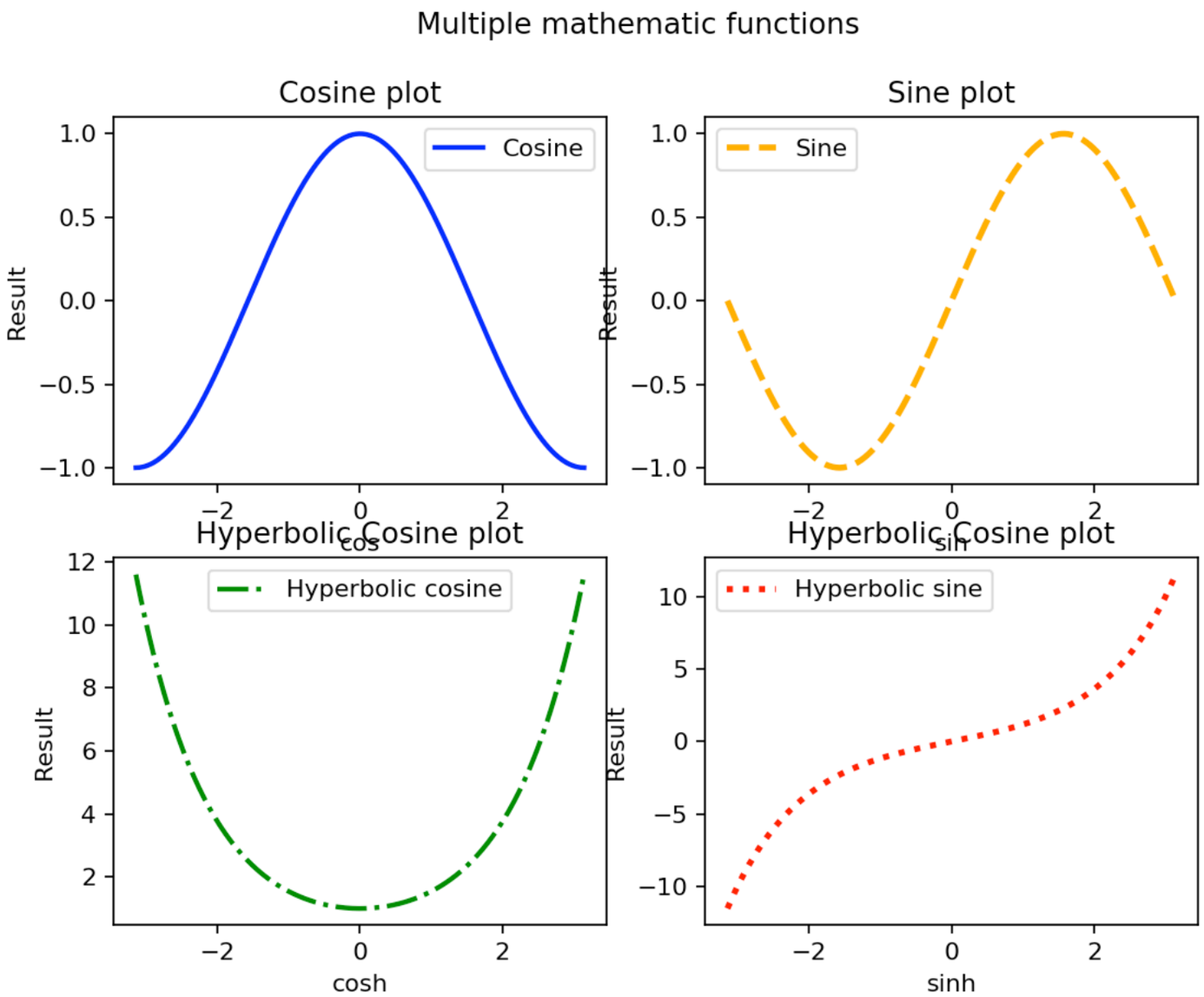
Try to adapt the 2 previous scripts to have this figure with all labels, legends and titles.



10. Matplotlib

Multi-plots

With legends, labels and titles :



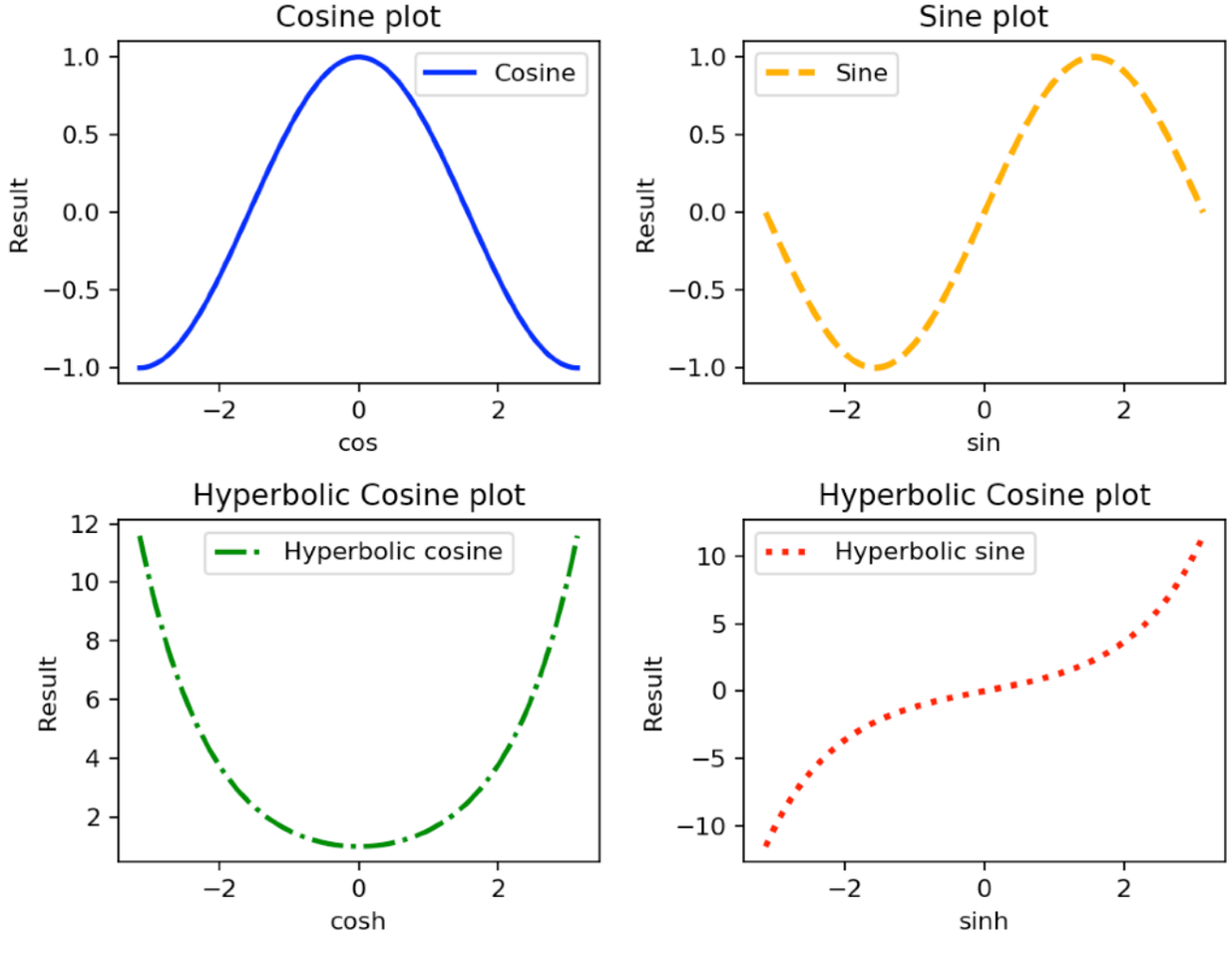
→ Problems of superposition

10. Matplotlib

Multi-plots

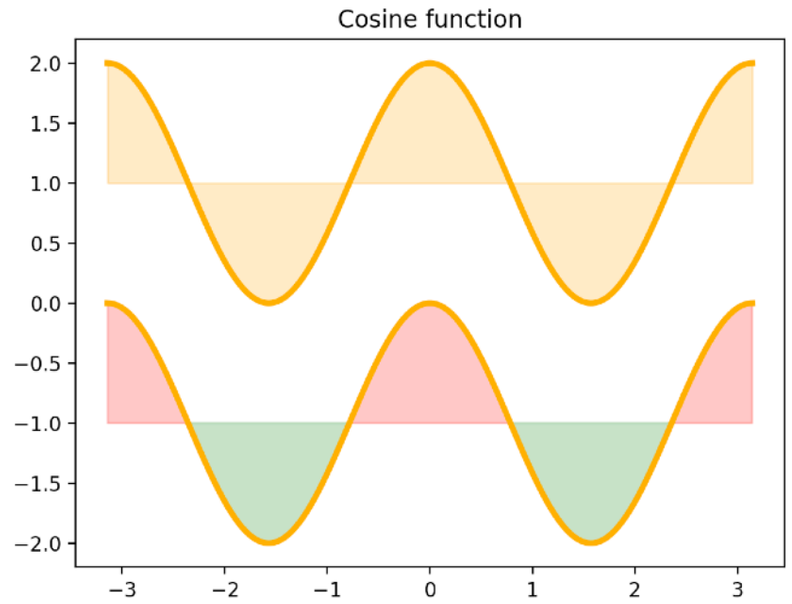
```
subplots_adjust(left=0.12, bottom=0.11, right=0.90,
     top=0.88, wspace=0.30, hspace=0.40)
```

Multiple mathematic functions



10. Matplotlib

Exercise : "Filled" plot



From this code, try to obtain this figure :

```
from pylab import *

n = 256
X = np.linspace(-np.pi, np.pi, n, endpoint=True)
Y = np.sin(2*X)

plot (X, Y+1, color='blue', linewidth=3, alpha=1.00)
fill_between(X, 1, Y+1, color='orange', alpha=.25)
plot (X, Y-1, color='orange', linewidth=3, alpha=1.00)
show()
```

→ You will need the `conditionToFill` of the function :

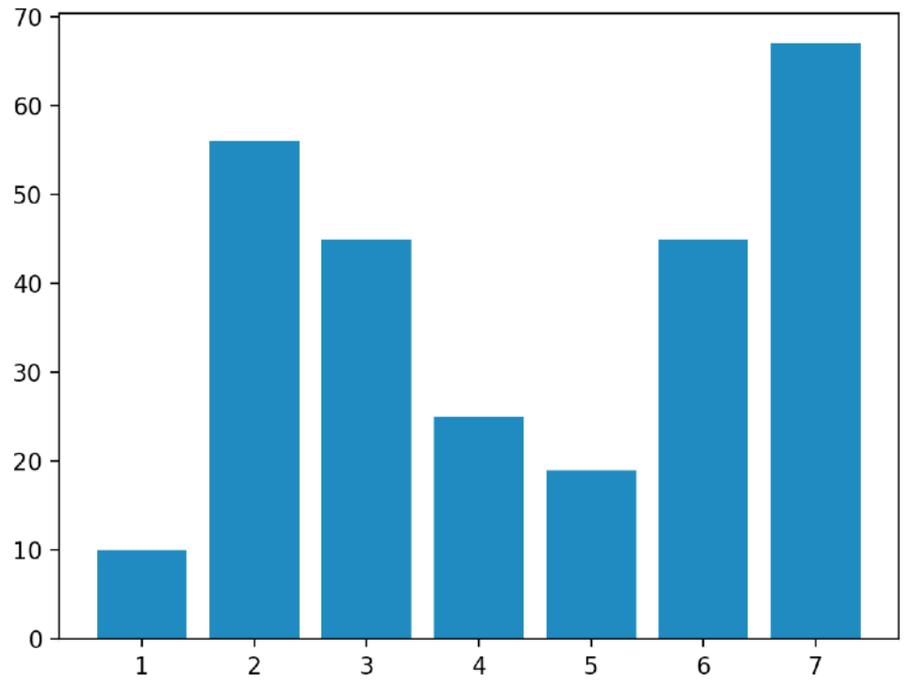
```
fill_between(xValues, yValues1, yValues2, conditionToFill, color, alpha)
```

10. Matplotlib

Barplot

Obtained with : `bar(xValues, yValues)`

```
from pylab import *

x = [1, 2, 3, 4, 5, 6, 7]
y = [10, 56, 45, 25, 19, 45, 67]

bar(x, y)

show()
```

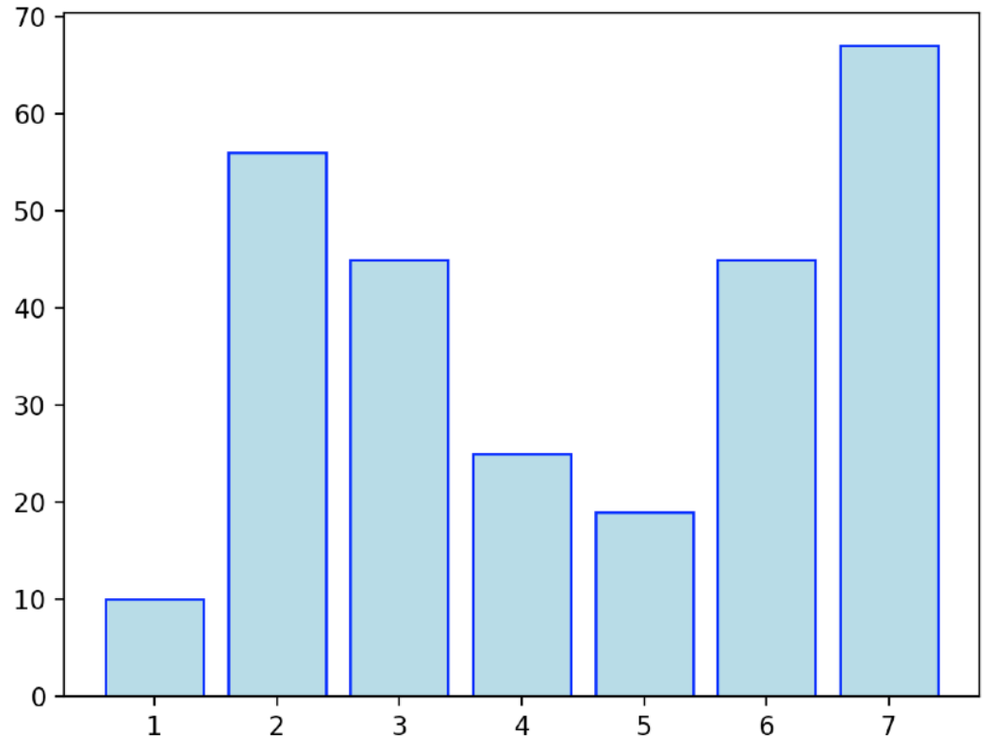


10. Matplotlib

Barplot

Colors can be customised :

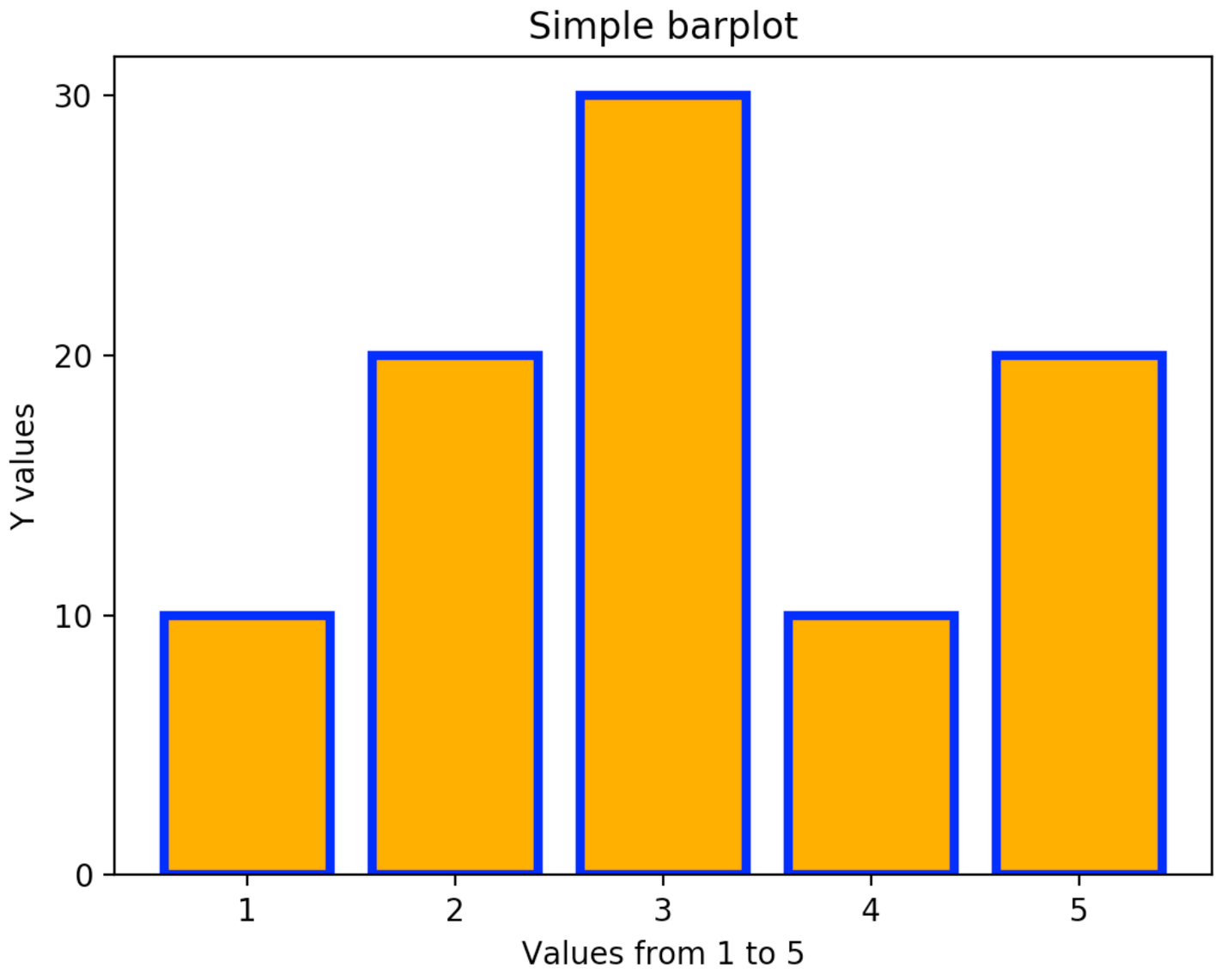
```
from pylab import *

x = [1, 2, 3, 4, 5, 6, 7]
y = [10, 56, 45, 25, 19, 45, 67]

bar(x, y, facecolor='lightblue', edgecolor='blue')

show()
```



10. Matplotlib

Barplot

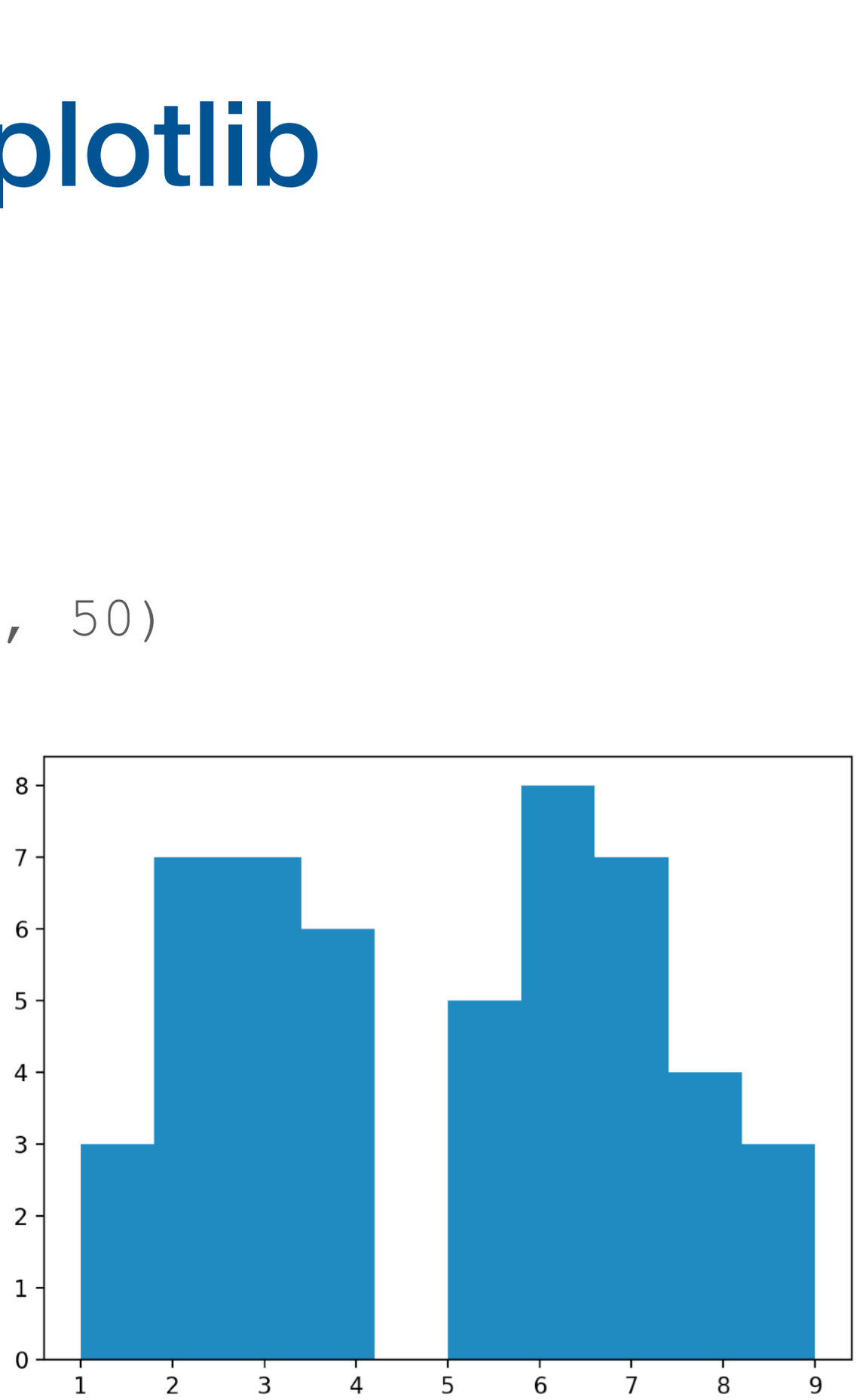
Exercise : try to obtain the exact same barplot



10. Matplotlib

Histogram

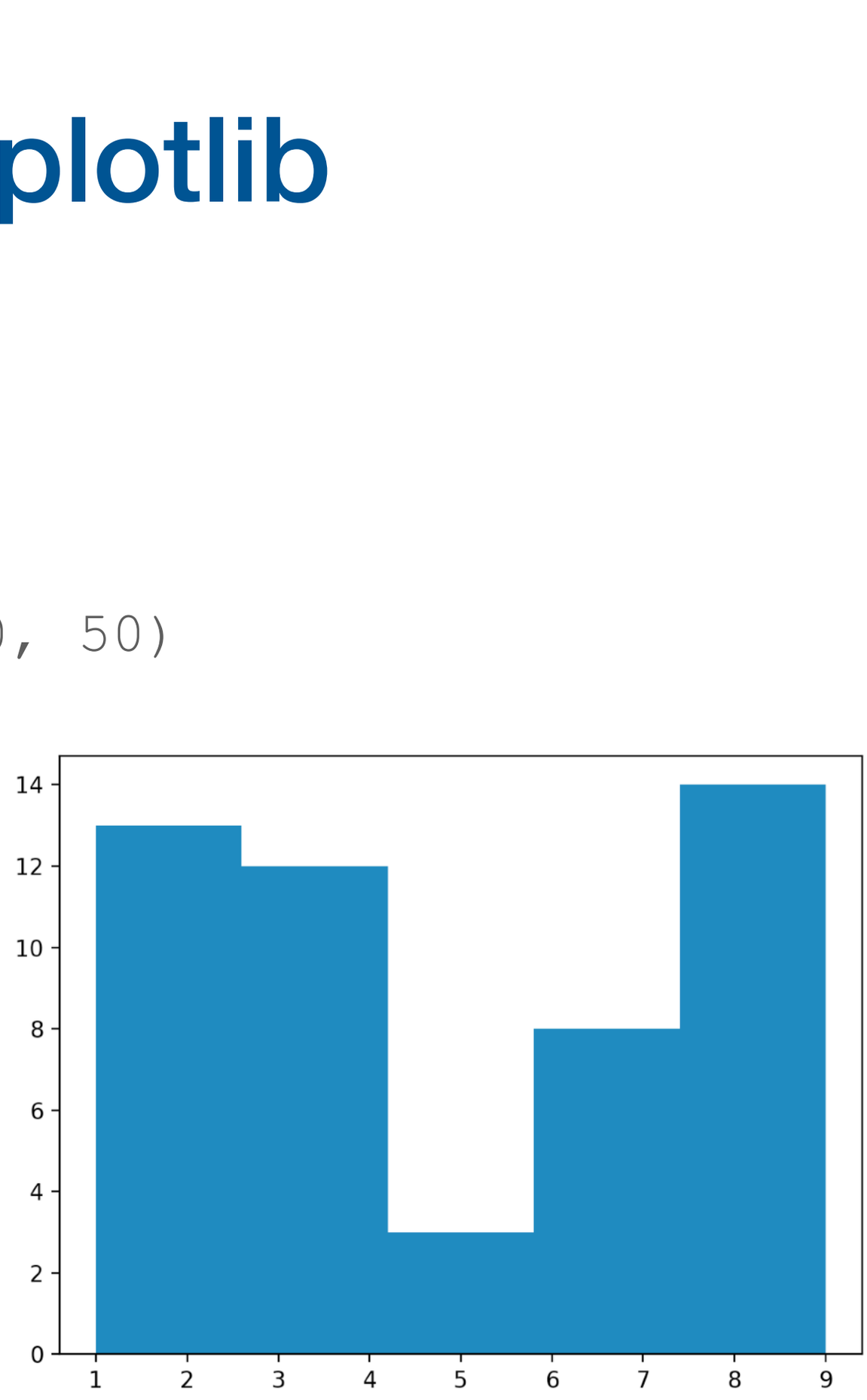
```
from pylab import *

x = np.random.randint(1, 10, 50)
print(x)

hist(x)

show()
```



10. Matplotlib

Histogram

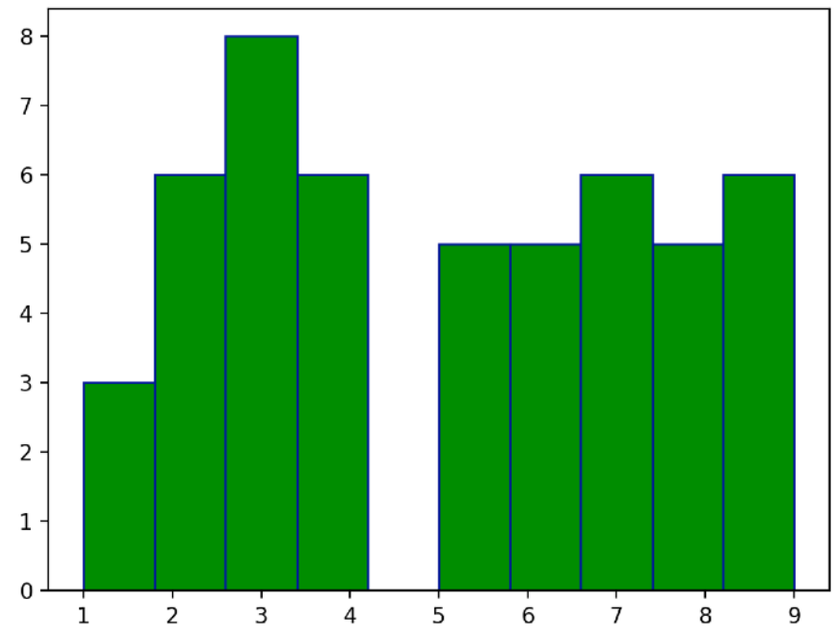
```
from pylab import *

x = np.random.randint(1, 10, 50)
print(x)

hist(x, bins=5)

show()
```



10. Matplotlib

Histogram

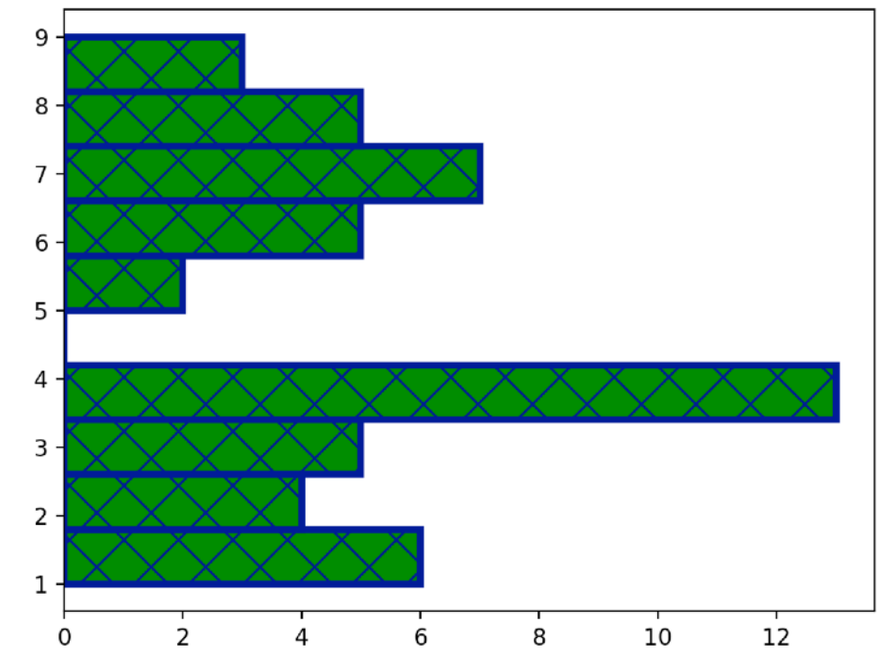
```
from pylab import *

x = np.random.randint(1, 10, 50)
print(x)

hist(x, color="green", edgecolor="darkblue")

show()
```



10. Matplotlib

Histogram

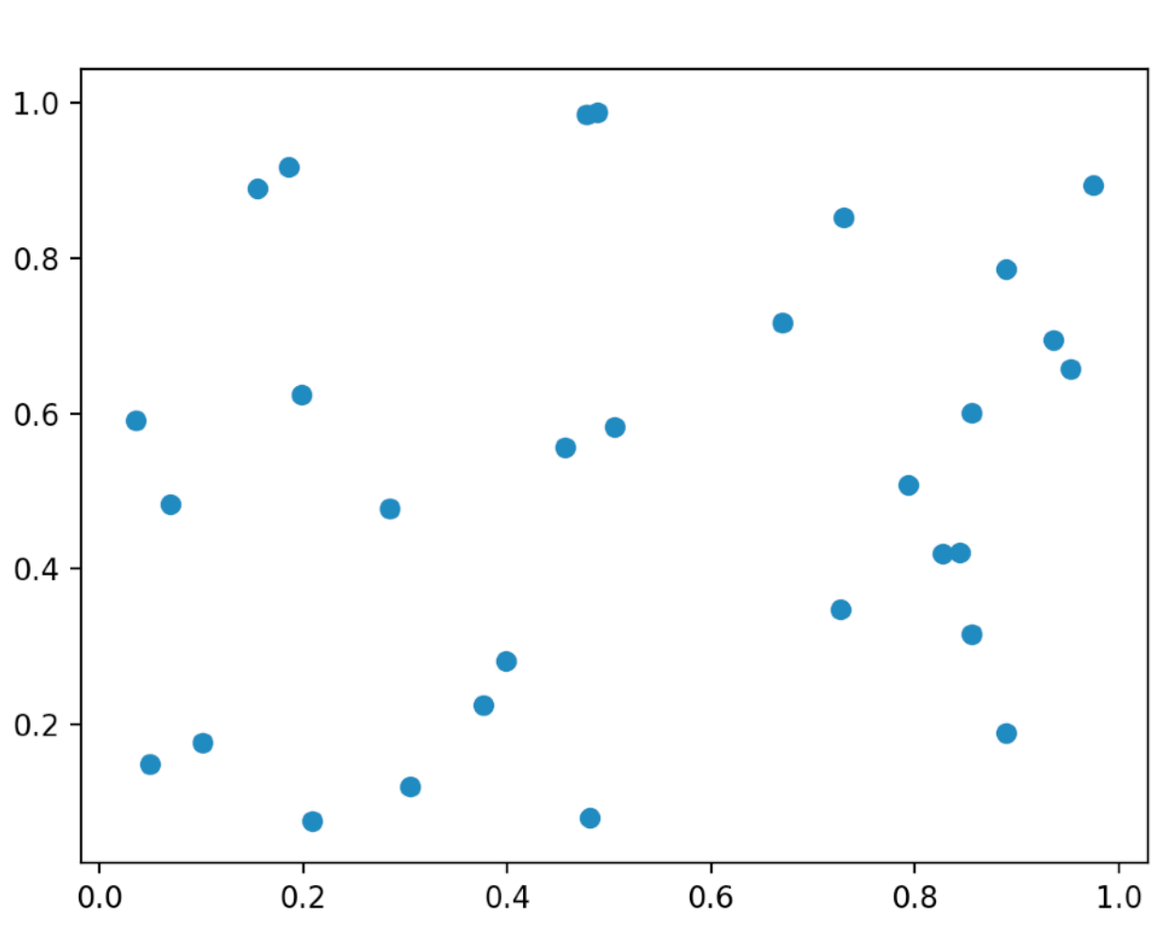
```
from pylab import *

x = np.random.randint(1, 10, 50)
print(x)

hist(x, color="green", edgecolor="darkblue",
   linewidth=3, hatch='x', orientation='horizontal')

show()
```



10. Matplotlib

Scatter plot

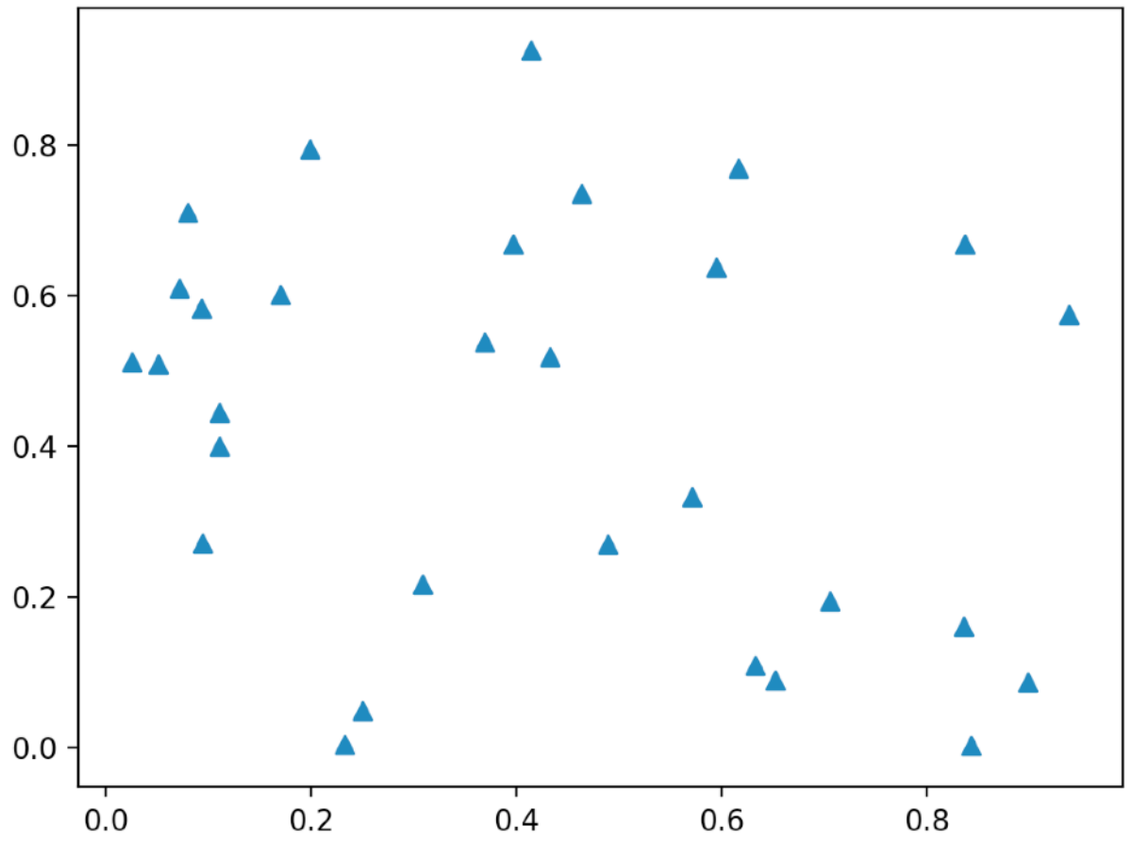
```
import matplotlib as np
from pylab import *

nbVal = 30
x = np.random.rand(nbVal)
y = np.random.rand(nbVal)

scatter(x, y)
show()
```



10. Matplotlib

Scatter plot

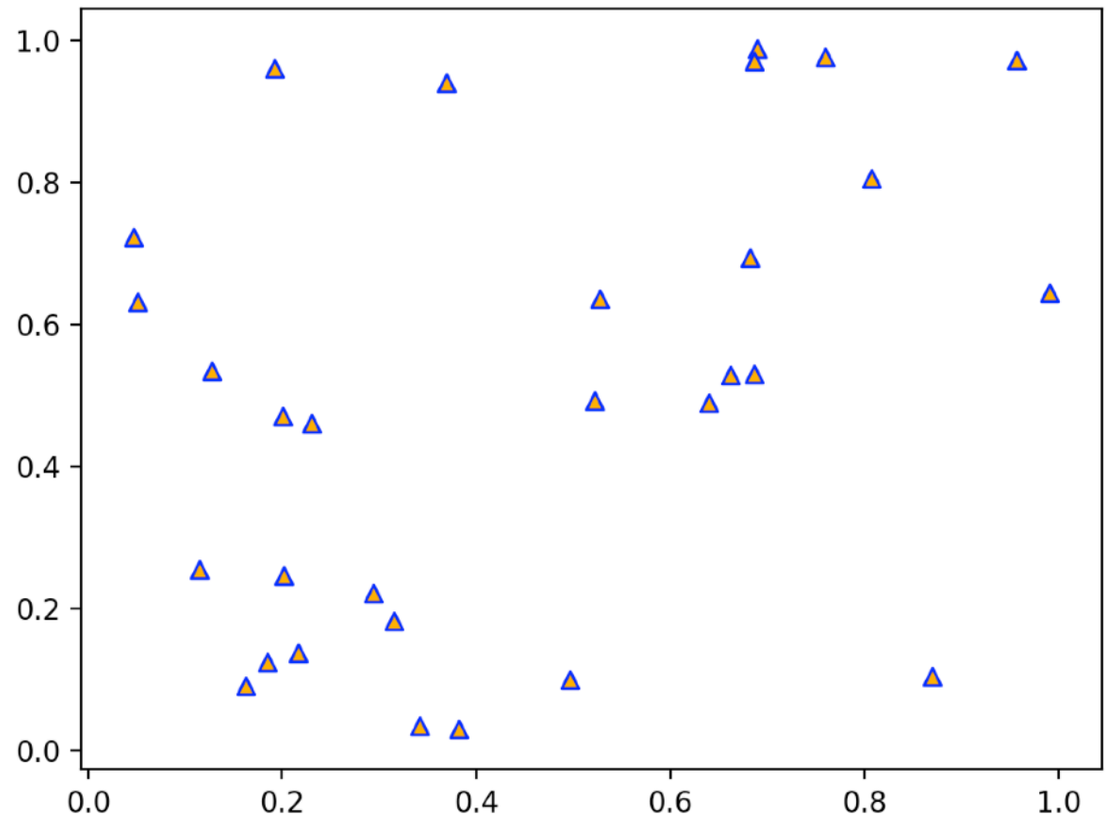
```
import matplotlib as np
from pylab import *

nbVal = 30
x = np.random.rand(nbVal)
y = np.random.rand(nbVal)

scatter(x, y, marker='^')
show()
```



10. Matplotlib

Scatter plot

```
import matplotlib as np
from pylab import *

nbVal = 30
x = np.random.rand(nbVal)
y = np.random.rand(nbVal)

scatter(x, y, marker='^', color='orange',
edgecolor='blue')
show()
```

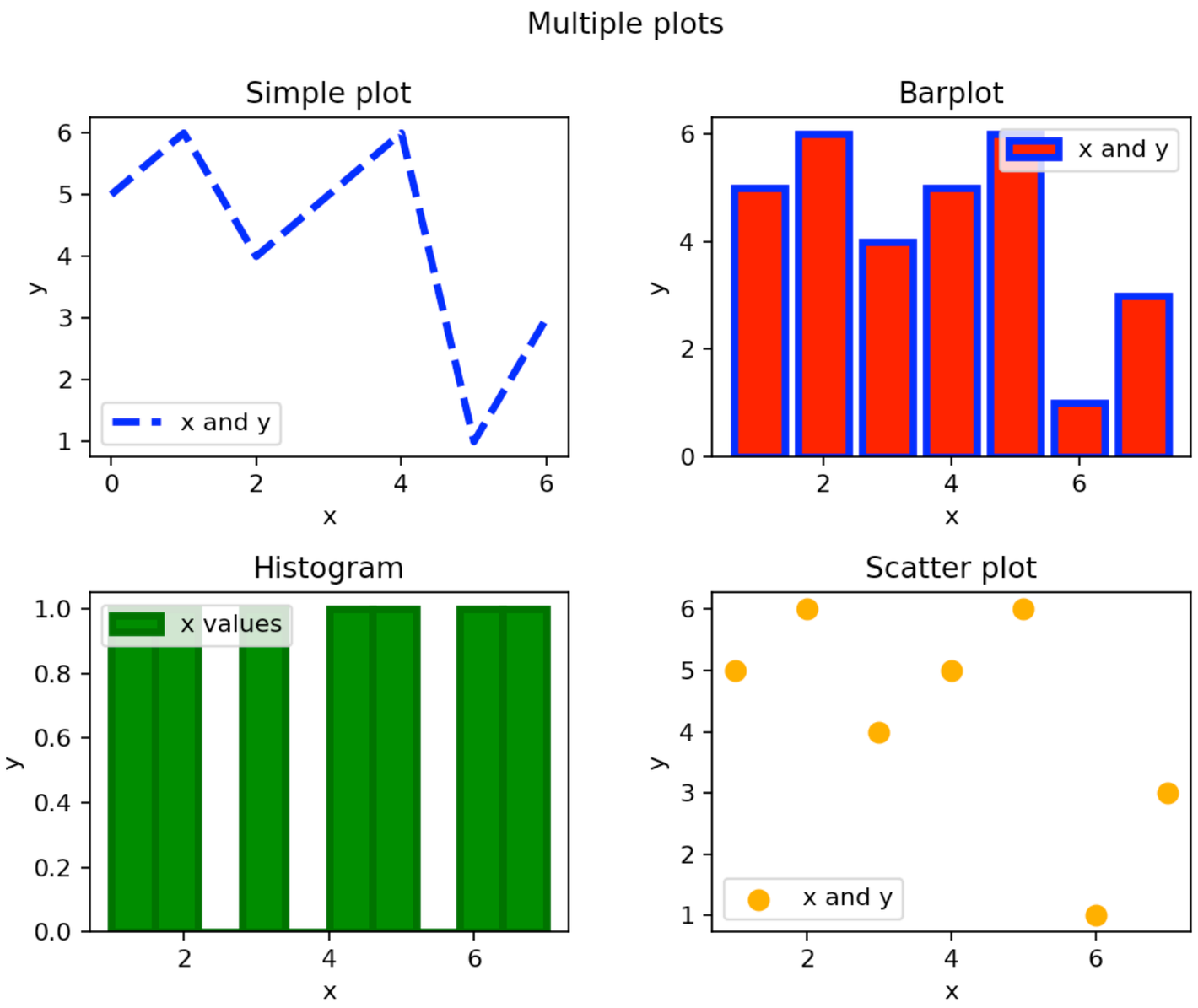


10. Matplotlib

Scatter plot

Try to generate this figure

```
x = [1, 2, 3, 4, 5, 6, 7]
y = [5, 6, 4, 5, 6, 1, 3]
```



Applications

Exercise: Find max/min/sum/mean

You will write a program that asks the user to enter 5 numbers and displays the maximum, minimum, sum and mean values.
You will create 4 functions for this (one for the max, one for the min, one for the sum and one for the mean).

Note: You can not use the functions `max()`, `min()` ans `sum()`.

Applications

Exercise: Lab schedule manager (version 3)

3 researchers are working in the lab and you want to follow the evolution of durations of their experiments.
You will write a program that asks the user his/her name, the time he/she starts (hour and minutes), the duration of the experiment (in minutes) and then displays the time he/she will end the experiment.
(Note: "24 hours" based system and no "AM/PM" for this exercise)

You will pay attention that the numbers entered for hours are from 0 to 23 and from 0 to 59 for the minutes.

You will use a dictionary to store the names (key) and all the others values (arriving/departure time and duration of the experiment), and then prints all the details.

Applications

Exercise: Lab schedule manager (version 4)

3 researchers are working in the lab and you want to follow the evolution of durations of their experiments.
You will write a program that asks the user his/her name, the time he/she starts (hour and minutes), the duration of the experiment (in minutes) and then displays the time he/she will end the experiment.
(Note: "24 hours" based system and no "AM/PM" for this exercise)

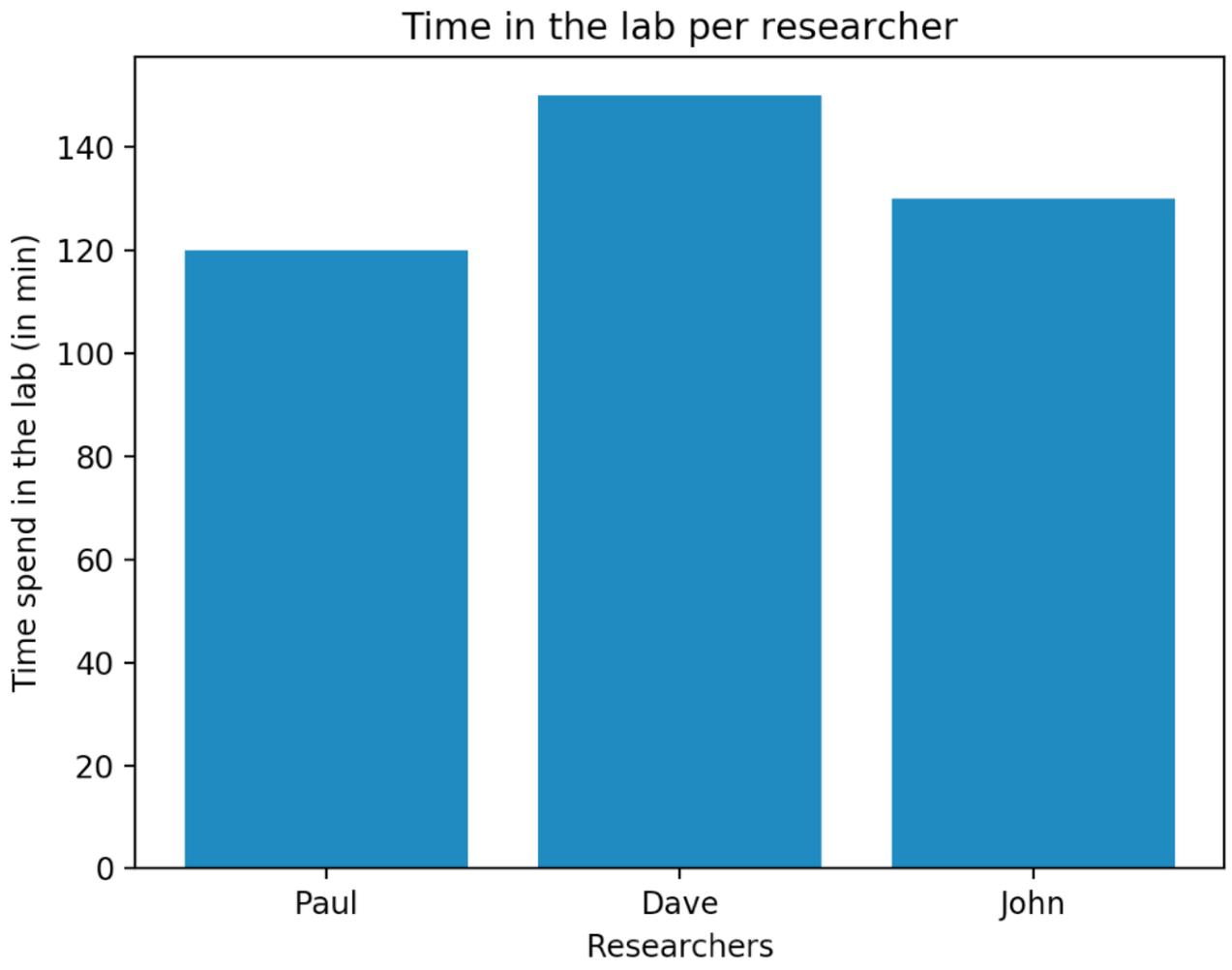
You will pay attention that the numbers entered for hours are from 0 to 23 and from 0 to 59 for the minutes.

Now you will have to make a plot to know who stays more in the lab.

Applications

Exercise: Lab schedule manager (version 4)

Now you will have to make a plot to know who stays more in the lab as in this example :




```
Paul arrives at 14:0 and stays 120 minutes in the lab
Dave arrives at 13:45 and stays 150 minutes in the lab
John arrives at 15:15 and stays 130 minutes in the lab
```

11. Write / Read data files

First step : directories

When you close your script, none of your variables is saved. With Python you can write and read data files (most of the time .txt, .dat or .xls files).

But first you need to set your working directory:

- **Windows** : if you work with the Python console, the working directory by default is `C:\Python3X` (`X`=Python version). You can change it with the command `chdir()` (`ch`ange `dir`ectory) :

```
>>> import os
>>> os.chdir("C:/Documents/MyPythonScripts")
```

If you want to check, use `getcwd()` (`get c`urrent `w`orking `d`irectory) :

```
>>> os.getcwd()
C:/Documents/MyPythonScripts
```

11. Write / Read data files

First step : directories

When you close your script, none of your variables is saved. With Python you can write and read data files (most of the time .txt, .dat or .xls files).

But first you need to set your working directory:

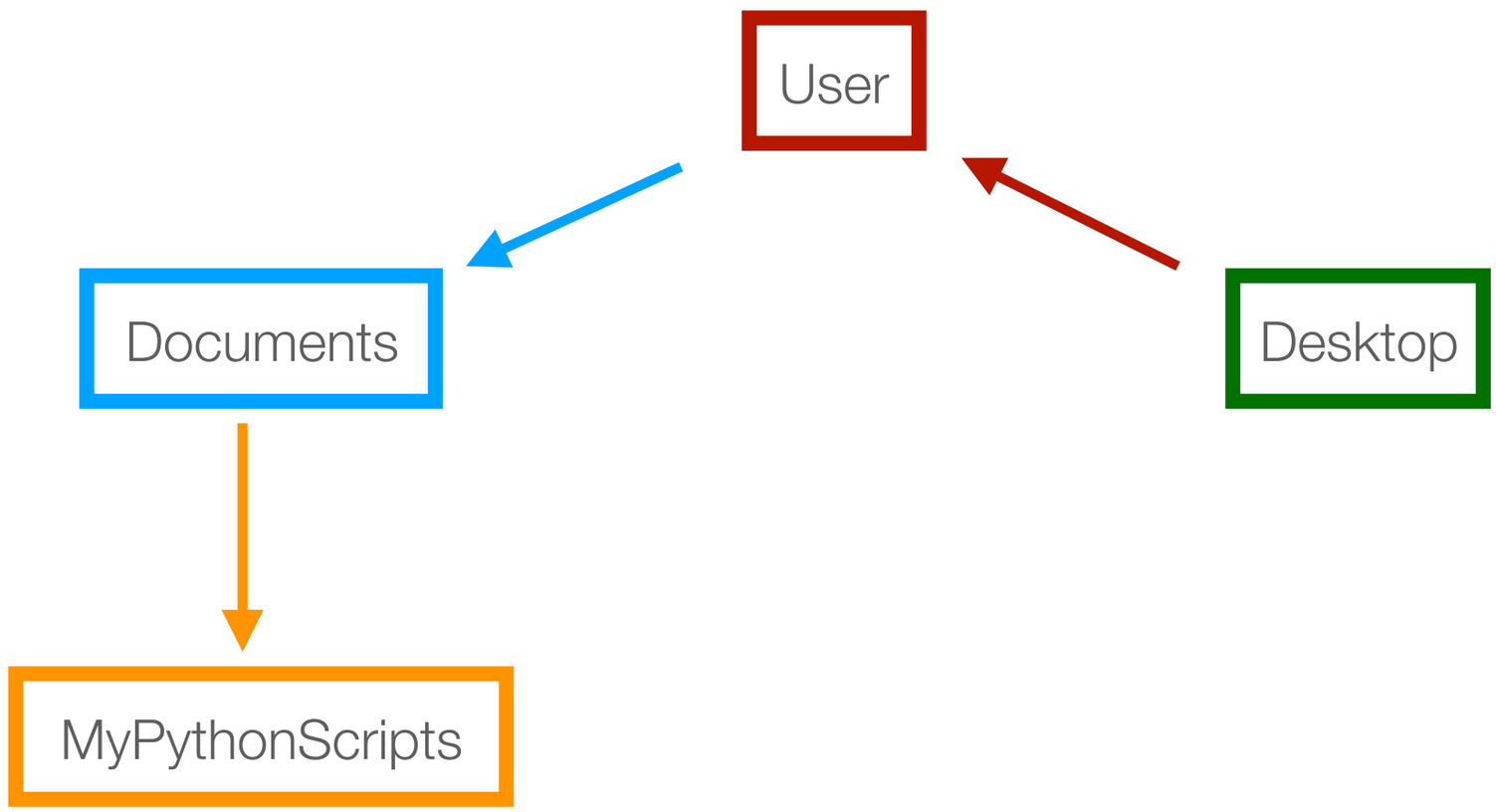
- **Linux/Mac** : you can use the Unix command line with the Terminal to set your working directory. The command `pwd` (`print w`orking `d`irectory) is to check your current directory and `cd` (`c`hange `d`irectory) to set it :

```
>>> pwd
/User/Desktop
>>> cd ../Documents/MyPythonScripts
```

11. Write / Read data files

First step : directories

```
>>> pwd
/User/Desktop
>>> cd ../Documents/MyPythonScripts
```



11. Write / Read data files

Opening/closing a file

Open your text editor, create a new file and write some random text. Save it as "`test.txt`".

To open a file : `open("path/fileName.extension", "mode")`

If your Python script and the txt file are in the same directory, you do not need the path.
For the mode, you have 3 main options :
- "`r`" : reading mode (when the file is only being read)
- "`w`" : writing mode (edit and write new informations in the file). Careful, if there is an existing file with the same name, this file will be erased and replaced by the new one. If the file does not exist, il will be created.
- "`a`" : appending mode (writing new data to the end of the existing file without replacing the existing content. If the file does not exist, il will be created.)

Every time you open a file, you have to close it when you are done working : `close()`

```
myFile = open("test.txt", "r")
print(myFile)
myFile.close()
```

```
<_io.TextIOWrapper name='test.txt' mode='r' encoding='UTF-8'>
```
→ Prints the class, name and encoding of your file

11. Write / Read data files

Reading a file

Read all the file : `read()`
This function "saves" the entire content of the file in a string.

```
myFile = open("test.txt", "r")
myText = myFile.read()
print(myText)
myFile.close()
```

Note : the line breaks are encoded by "`\n`", the tabulations by "`\t`"

11. Write / Read data files

Reading a file

Read a line of the file :

```
myFile = open("test.txt", "r")
firstLine = myFile.readline()
first2letters = myFile.readline(2)
print("First word of the file:", firstLine)
print("First 2 letters:", first2letters)
myFile.close()
```

Read all lines :

```
myFile = open("test.txt", "r")
allLines = myFile.readlines()
print("All lines of the file:", allLines)
myFile.close()
```

11. Write / Read data files

Looping over a file

You can use a for loop to read every line of your file :

```
myFile = open("test.txt", "r")
for line in myFile:
   print(line)
myFile.close()
```

Using a table, you can save the content of your file once it is closed :

```
myWords = []
myFile = open("test.txt", "r")
for line in myFile:
   myWords.append(line)
myFile.close()
print(myWords)
```

11. Write / Read data files

Looping over a file

You can split the lines taken from a file with the function : split()
It splits the string contained in variable data whenever the interpreter encounters a space character.

```
myFile = open("test.txt", "r")
allLines = myFile.readlines()
for line in allLines:
   words = line.split()
   print(words)
myFile.close()
```

You can also split your text using other character, such as "`\n`", "`;`", "`\t`" depending of what is inside your text file and how you want your data split.

11. Write / Read data files

Writing to a file

After opening the file and choosing the mode ("`w`" or "`a`"), you can write to your text file with the function : `write()`

```
myWords = []
myFile = open("test.txt", "a")
myFile.write("\n")
myFile.write("Fine, thanks")
myFile.close()
```

→ Check in `test.txt` to see what just happened or print it.

11. Write / Read data files

Working with Excel files

You will need the file "mice.xlsx" and the library Panda.

```
import pandas as pd

# Excel file
myFile = 'mice.xlsx'

# Excel spreadsheet
sp = pd.ExcelFile(myFile)

# Print the sheet names
print(sp.sheet_names)

# Load a sheet into a DataFrame named myData
myData = sp.parse('Feuil1')
```

11. Write / Read data files

Working with Excel files

```
print(myData.head())
```

```
  Weights  Mouse1  Mouse2  Mouse3  Mouse4  Mouse5
0    Day1      20      18      22      14      21
1    Day2      21      16      21      15      21
2    Day3      20      19      23      13      20
3    Day4      19      17      22      16      23
4    Day5      18      20      20      16      23
```

11. Write / Read data files

Working with Excel files

You will need the file "mice.xlsx" and the library Panda.

print(myData["Mouse1"])

```
0    20
1    21
2    20
3    19
4    18
5    22
6    21
7    20
Name: Mouse1, dtype: int64
```

→ You can use myData with the dictionary methods.

11. Write / Read data files

Working with Excel files

With the file "mice.xlsx", print all the weights for each mouse and then make a plot as in the example.