How to program with python™ programming language

Charlotte HÉRICÉ
Sakata lab - SIPBS
University of Strathclyde - Glasgow

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

1. Introduction

Computer programming

Objectives:

- Learn to « think » from algorithms
- Translate the resulting reasoning into a computer program

Tools:

- Programming languages : Python
- Text editors : NotePad, SublimeText, Emacs, Python IDLE …
- Python environment : Unix console (Terminal), Python console …

1. Introduction

What is a computer program ?

- A source file
- A set of statements/declarations and instructions
- A set of librairies, dynamically or statistically loaded
- An executable machine file (bytecode)

You can generate :
- A program directly executable by a user
- A program requiring a virtual machine / interpreter
- An other program to run yours (such as Python core algorithm in a Java applet, run by a web browser)

1. Introduction

Respect of the lexical and synthetic rules

- The source program must respect the synthetic and medical rules of the programming language you are using :
    - respect of the reserved words (such as « `for` », « `if` », « `print` », « `len` », « `range` » …. in Python)
    - respect of the parenthesis : `()`, `{}`, `[]`
    - respect of the indentation
    - respect of the spaces
    - respect of upper and lower case
    - respect of the language specificities (such as « ; » at the end of each line in Java)

1. Introduction

Python programming language

- Developed since 1991 in Nederland by Guido van Rossum
- OpenSource and free
- Portable
- Python syntax quite simple but allow to build and manipulate complex datasets and evaluated programs
- Automatic memory management thanks to the `Garbage Collector`
- Can be interfaced with many other languages : Perl, Java (JPython), C/C++ (Cython), DataBases systems …
- Many librairies available : TKinter (GUI), NumPy, SciPy …

Note 1: all the following lines of code in these slides are in Python3.
Note 2: there are many ways to write a program, here are just one version among others

1. Introduction

Bibliography

- Official Python website: www.python.org
- https://www.learnpython.org
- https://www.codecademy.com/learn/learn-python
- https://www.datacamp.com
- http://matplotlib.org

1. Introduction

My first Python program

Respect of the tradition : display « Hello World »

→ Open your favorite text editor
→ Write :

```
        print(''Hello Word'')
```

→ Save the current file (`nameOfYourChoice.py`) in your working directory
Note : no spaces, @, #, ^, $, /, *, (, ;, %, é, à .... in a file name
→ Set your working directory in the console or Python interpreter
→ Write `python nameOfYourChoice.py` in the console or Python interpreter
→ See what's happening into the console

2. Variables

The wonderful world of variables

What is a variable ?
- A piece of data from your program, stored on your computer
- An alphanumeric code that you link to a given piece of your program to use it multiple times (ex: store the result of an operation)

In Python :

```
nameOfTheVariable = valueOfYourChoice
```

Basic rules with variables :
- Only letters (upper and/or lower case) and/or numbers and/or underscore « _ » in a variable name
- No spaces, @, #, ^, $, /, *, (, ;, %, é, à …. in a variable name
- The name can not start with a number
- Python is case-sensitive: `AGE`, `aGe`, `age` and `AgE` are 4 different variables

2. Variables

The wonderful world of variables

Syntax conventions (it is up to you but stay coherent) :

```
my_age = 25                    myAge = 25
```

Basic operation with a variable:

Write a program that displays the variable `age` at the number of your choice, then adds 2 to this value and displays the new value.

Do the same but with the ages multiplied by 2 instead.

2. Variables

The wonderful world of variables

Syntax conventions (it is up to you but stay coherent) :

my_age = 25   myAge = 25

Basic operation with a variable:

```
myAge = 25
print(myAge)
myAge = myAge + 2
print(myAge)
```

Do the same but with the ages multiplied by 2 instead.

2. Variables

The wonderful world of variables

Syntax conventions (it is up to you but stay coherent) :

my_age = 25 myAge = 25

Basic operation with a variable:

```
myAge = 25
print(myAge)
myAge = myAge + 2
print(myAge)

myAge_x2 = myAge * 2
print(myAge_x2)
```

2. Variables

The wonderful world of variables

Every programming language has some reserved keywords to manipulate the variables. In Python3 they are:

`and`	`del`	`from`	`none`
`true`	`as`	`elif`	`global`
`nonlocal`	`try`	`assert`	`else`
`if`	`not`	`while`	`break`
`except`	`import`	`or`	`with`
`class`	`false`	`in`	`pass`
`yield`	`continue`	`finally`	`is`
`raise`	`def`	`for`	`lambda`
`return`			

→ That means no variable name with these words.

2. Variables

Types of variables

Python needs to know the type of the variable in order to know which operation can be done. Every data in Python has a type, if you want to verify, just write `type(nameOfYourVariable)`.

Numbers

Python supports 2 types of numbers :
- intergers (`int`) : 5
- floating point numbers (`float`) : 5.0

```
myIntNumber = 5
print(myIntNumber)
print(type(myIntNumber))
```

```
myFloatNumber = 5
print(myFloatNumber)
print(type(myFloatNumber))
```

2. Variables

Types of variables

Strings

For letters, words or sentences. Strings can be defined either with a single quote, a double quotes or a triple quotes.

```
mySingleString = 'hello!'
myDoubleString = "hello!!"
myTripleString = """hello!!!"""

print(mySingleString)
print(myDoubleString)
print(myTripleString)
```

2. Variables

Types of variables

Strings

For letters, words or sentences. Strings can be defined either with a single quote, a double quotes or a triple quotes.

Be careful with apostrophes :

```
myString1 = "Yes I'll do it !"
myString2 = 'Yes I\'ll do it !'
myString3 = 'Yes I'll do it !'        → error

print(myString1)
print(myString2)
print(myString3)
```

2. Variables

Exercise

You want the name and age of a person. Write a script to ask the user to enter his/her name and then his/her age. You will have to store age and name into variables and print them.

You will need the function `input()` that reads a line from input, converts it to a string and returns it.

Help :
- for the name (`string`) → `input()`
- for the age (`integer`) → `int(input())`

2. Variables

Exercise

You want the name and age of a person. Write a script to ask the user to enter his/her name and then his/her age. You will have to store age and name into variables and print them.

You will need the function `input()` that reads a line from input, converts it to a string and returns it.

```
print("Please enter your name")
name = input()
print("Please enter your age (in years)")
age = int(input())
print("Your name is", name, "and you are", age,
"years old.")
```

3. Lists

Basic utilisation

- Similar to arrays to collect ordered elements
- Contain any type of variable (`int`, `float`, `str`, `bool`, other lists …) and as many as you wish (just the limit of your computer memory)

- Declaration of an empty list : `l = []`
- Declaration of a list with values : `l = [val1, val2, val3]`
- Add a value to the list : `l.append(value)`
- Remove a value : `l.remove(value)`
- Length of the list : `len(l)`
- First element of the list : `l[0]` (→ In Matlab, first element at index 1 but in Python and many other programming languages, first element at index 0)
- Fourth element of the list : `l[3]`
- Last element of the list : `l[-1]`

3. Lists

Basic utilisation

Examples :

```
l = []
print(l)
l.append(10)
print(l)
l.append(14)
print(l)
print(l[0])
```

```
li = [4, 6, 7, 2]
print(li)
li.append(3)
print(li)
print(len(li))
print(li[2])
print(li[-1])
li.remove(4)
print(li)
```

3. Lists

Manipulations

```
myList = [[1, 3, 5], [4, 6, 1], [2, 9, 3, 8]]
print(myList)
print(myList[0][1])

myList[0][1]=10
print(myList)
myList.pop(1)
print(myList)
```

→ Here `pop()` suppresses the second element of the list and return it

4. Operations

Operators

- Affectation : `a = 3`
- Arithmetic : `+ - * + % **`
- Comparison : `< > <= >= == !=`
- Logic : `or and not`

Simple examples:

```
number1 = 5
number2 = 6
finalNumber = number1 + number2
print(finalNumber)

hello = "Hello"
world = "world"
finalHello1 = hello + world
finalHello2 = hello + " " + world
print(finalHello1)
print(finalHello2)
```

4. Operations

Operators

- Affectation : `a = 3`
- Arithmetic : `+ - * + % **`
- Comparison : `< > <= >= == !=`
- Logic : `or and not`

But :

```
number1 = 5
hello = "Hello"
helloNumber = hello + number1
print(helloNumber)
```

→ error message

4. Operations

Operators

- Affectation : `a = 3`
- Arithmetic : `+ - * + % **`
- Comparison : `< > <= >= == !=`
- Logic : `or and not`

Correct way :

```
number1 = 5
hello = "Hello"
helloNumber = hello + str(number1)
print(helloNumber)
```

→ `number1` is converted from `int` to `string` with the function `str()`

4. Operations

Operations on strings

- Concatenation (adding) : `"Hello" + " " + "World"`
- Repetition : `"hello" * 2 → "hellohello"`
- Length of the string : `len(myString)`

Access by index :

```
myWord = "anything"
print(myWord[2]) → "y"
print(myWord[:2]) → "an"
print(myWord[2:]) → "ything"
print(myWord[2:4]) → "yt"
print(myWord[-3]) → "i"
```

4. Operations

Operations on strings

```
string1 = "abcdefg"
string2 = "cd"
print(string2 in string1)
```
→ True if `string1` contains `string2`
```
print(string2 not in string1)
```
→ True if `string1` does not contain `string2`

```
myWord1 = "10"
myNumber = int(myWord1)
myWord2 = "11.3"
myFloat = float(myWord2)
myNum = 12
myWord3 = str(myNum)
```

4. Operations

Operations on strings

Comparison

```
s1 = "blue"
s2 = "blue"
s3 = "Blue"
```

```
print(s1 == s2)
```
→ Returns 1 → True
```
print(s1 == s3)
```
→ Returns 0 → False

4. Operations

Operations on strings

Formatting with %

- `%s` for `string`
- `%d` for `int`
- `%f` for `floats`

```
name = "Mary"
age = 22
print("%s is %d years old." %(name, age))
```

5. Boolean operators

If / else / elif

To test a condition in order to control the following instructions to execute : if

Syntax

```
if  (boolean test1):
      instruction if test1 is True
elif (boolean test2):
      instruction if test2 is True
else:
      instruction if test1 and test2 are False
```

Please respect the indentation !

5. Boolean operators

If / else / elif

Example

```
a = 4
 if (a%2 == 0):
    print("a is even")
 else:
    print("a is odd")
```

5. Boolean operators

Exercise 1

- For the purpose of a public health survey, you have a population sample where each person is defined by two variables : `sex` and `height` (in cm)
- The variable sex can be either 0 for male, 1 for female
- The variable height is a full number
- To have someone in this study, there are conditions :
    - Height superior to 180 for a man ("M")
    - Height inferior to 160 for a woman ("F")

5. Boolean operators

Exercise 1

- For the purpose of a public health survey, you have a population sample where each person is defined by two variables : `sex` and `height` (in cm)
- The variable sex can be either 0 for male, 1 for female
- The variable height is a full number
- To have someone in this study, there are conditions :
    - Height superior to 180 for a man ("M")
    - Height inferior to 160 for a woman ("F")

```
print("Please, give a value for the sex (0 for M, 1 for F)")
sex = int(input())
print("Please, give a value for the height")
height = int(input())
```

5. Boolean operators

Exercise 1

```
print("Please, give a value for the sex (0 for M,
  1 for F)")
sex = int(input())
print("Please, give a value for the height")
height = int(input())

if ((sex == 1 and height<160) or (sex == 0 and
  height>180)):
      print("Welcome to the study")
else:
      print("Sorry, you do not fit the profile of
  the study")
```

5. Boolean operators

Exercise 2

You have some mice but you want only the ones that are more than 3 days old. Write a program where the age of the mouse is asked and if this age is more than 3 days, the mouse is selected.

```
print("Please give the age of your mouse in days")
age = int(input())

if …
```

5. Boolean operators

Exercise 2

You have some mice but you want only the ones that are more than 3 days old. Write a program where the age of the mouse is asked and if this age is more than 3 days, the mouse is selected.

```
print("Please give the age of your mouse in days")
age = int(input())
if (age > 3):
  print("This mouse is selected")
else:
  print("This mouse is too young")
```

5. Boolean operators

Exercise 2

You have some mice but you want only the ones that are more than 3 days old. Write a program where the age of the mouse is asked and if this age is more than 3 days, the mouse is selected.

```
print("Please give the age of your mouse in days")
age = int(input())
if (age > 3):
  print("This mouse is selected")
else:
  print("This mouse is too young")
```

Now you want to attribute a code to the selected animals ("F3" for females and "M3" for males) and then display the code of your mouse.

5. Boolean operators

Exercise 2

```
print("Please give the age of your mouse in days")
age = int(input())
print("Please give the sex of this mouse ('M' or 'F')")
sex = input()
if (age > 3):
  if (sex == "M"):
     code = "M3"
  elif (sex == "F"):
     code = "F3"
else:
   print("This mouse is too young")
print("The code of your mouse is", code)
```

5. Boolean operators

Exercise 2

```
print("Please give the age of your mouse in days")
age = int(input())
print("Please give the sex of this mouse ('M' or 'F')")
sex = input()
if (age > 3):
   if (sex == "M"):
      code = "M3"
   elif (sex == "F"):
      code = "F3"
else:
   print("This mouse is too young")
print("The code of your mouse is", code)
```

→ What if someone enters a wrong value ?

5. Boolean operators

Exercise 2 : handle errors

```
print("Please give the age of your mouse in days")
age = int(input())
print("Please give the sex of this mouse ('M' or 'F')")
sex = input()
if (age > 3):
  if (sex == "M"):
     code = "M3"
  elif (sex == "F"):
     code = "F3"
  else:
      print("Sorry, wrong code")
      code = "Undefined"
else:
  print("This mouse is too young")
print("The code of your mouse is", code)
```

5. Boolean operators

Exercise 2 : handle errors (other version)

```
print("Please give the age of your mouse in days")
age = int(input())
print("Please give the sex of this mouse ('M' or 'F')")
sex = input()
if (age > 3):
    if ((sex != "M") and (sex != "F")):
        print("You have to write 'M' or 'F' for the sex")
        code = "Undefined"
    else:
        if (sex == "M"):
            code = "M3"
        elif (sex == "F"):
            code = "A3"
else:
    print("This mouse is too young")
    code = "Undefined"
if (code == "Undefined"):
    print("No code affected")
else:
    print("The code of your mouse is", code)
```

6. Loops

While

Used to repeat an operation as many times as necessary (while the condition is True).

Syntax

```
while (condition):
    instruction 1
    instruction 2
    …
```

Note : Pay attention to the condition, you do not want an infinite loop !
Tips : If your program is blocked with an infinite loop, type CTRL+C to stop it (works with Windows, Linux or Mac).

6. Loops

While

Example : you want a program that displays the 7 multiplication table.
Try to write it without any loop.
Without the loop (option 1) :

6. Loops

While

Example : you want a program that displays the 7 multiplication table.
Try to write it without any loop.
Without the loop (option 1) :

```
print(" 1 * 7 =", 1 * 7)
print(" 2 * 7 =", 2 * 7)
print(" 3 * 7 =", 3 * 7)
print(" 4 * 7 =", 4 * 7)
print(" 5 * 7 =", 5 * 7)
print(" 6 * 7 =", 6 * 7)
print(" 7 * 7 =", 7 * 7)
print(" 8 * 7 =", 8 * 7)
print(" 9 * 7 =", 9 * 7)
print("10 * 7 =", 10 * 7)
```

6. Loops

While

Example : you want a program that displays the 7 multiplication table.

Without the loop (option 2) :

```
nb = 7
print(" 1 *", nb, "=", 1 * nb)
print(" 2 *", nb, "=", 2 * nb)
print(" 3 *", nb, "=", 3 * nb)
print(" 4 *", nb, "=", 4 * nb)
print(" 5 *", nb, "=", 5 * nb)
print(" 6 *", nb, "=", 6 * nb)
print(" 7 *", nb, "=", 7 * nb)
print(" 8 *", nb, "=", 8 * nb)
print(" 9 *", nb, "=", 9 * nb)
print("10 *", nb, "=", 10 * nb)
```

6. Loops

While

Example : you want a program that displays the 7 multiplication table.

With the loop :

```
nb = 7
i = 0 # Counter that will be incremented into the loop

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

Note : Do not forget to increment the counter `i` otherwise you will have an infinite loop.

6. Loops

While

You can exit a for/while loop without stopping your program with the keyword `break`.

Example

```
while 1: # 1 always true → infinite loop
    print("Press any key to continue, press 'q'
      to quit")
    letter = input()
    if (letter == "q"):
        print("End of the loop")
        break
```

6. Loops

While

You can skip the current block of instructions and return to the for/while loop with the keyword `continue`.

Example

```
# Print only odd numbers from 1 to 10
x = 0
while (x < 10):
    x += 1
    if (x % 2 == 0): # Check if x is even
        continue
    print(x)
```

Try with even numbers.

6. Loops

While

```
# Print only even numbers from 1 to 10
x = 0
while (x < 10):
    x += 1
    if (x % 2 != 0): # Check if x is odd
        continue
    print(x)
```

6. Loops

For

Used to iterate over a given sequence.

Syntax

```
for element in sequence:
    instruction
```

Here `element` is a variable created by the `for`, you do not have to instantiate it. It will be used only inside the loop, so you can used again the same name for an other loop (is the loops are not nested).

6. Loops

For

```
days = ["Monday", "Tuesday", "Wednesday"]

print("Days:", days)
for day in days:
      print("Day : ", day)

for i in range(len(days)):
      print("i=", i)
      print("days[", i, "]=", days[i])
```

6. Loops

For

Print all values between 0 and 9

```
for i in range(10):
    print(i)
```

The function `range()` creates a list a all full values between 0 and 9.
This function has different syntaxes :

- `range(numberOfValues)`
- `range(startValue, endValue)`
- `range(startValue, endValue, stepValue)`

```
for i in range(2, 10, 3):
    print(i)
```

6. Loops

For

We want to create a list of all squares values of integers between 0 and 20.
Help 1 : i^2 is `i ** 2` in Python
Help 2 : remember how to add a value to a list (section 3)

6. Loops

For

We want to create a list of all squares values of integers between 0 and 20.
Help 1 : i^2 is `i ** 2` in Python
Help 2 : remember how to add a value to a list (section 3)

```
myList = []
for i in range(21):
    tempValue = i ** 2
    print(i, "^ 2 =", tempValue)
    myList.append(tempValue)
print(myList)
```

6. Loops

For

You have 2 mice and you want their total weight of the last 2 days. Create a program with a list with your 2 mouse names, then the program will ask you to give the weight for each mouse, every day, and finally print the total weight for each animal.

6. Loops

For

You have 2 mice and you want their total weight of the last 2 days.

```
animals = ["mickey", "minie"]
days = 2
for animal in animals:
    result = 0
    for i in range(days):
        print("Give a weight for", animal, "at day", i)
        weight = eval(input())
        result = result + weight
    print("Total weight for", animal, "is", result)
print("Done")
```

Applications

Exercise: Basic operations

Write a program that asks the user to enter a float number and :
- if this number is strictly positive, multiply it by 10
- if it is strictly negative, divide it by 2
- otherwise, display an error message.

You will print the outcome result for the positive and negative floats.

Applications

Exercise: Basic operations

```
print("Please enter a float")
nb = float(input())
if (nb > 0):
   res = nb * 10
elif (nb < 0):
   res = nb / 2
else:
   print("Enter any number exept 0")
   res = "Undefined"
if (res != "Undefined"):
   print("The result is", res)
```

Applications

Exercise: Minimum

Write a program that asks the user to enter two numbers, finds the smaller one and displays the result.

Applications

Exercise: Minimum

```
print("Please enter the first number")
nb1 = float(input())
print("Please enter the second number ")
nb2 = float(input())

if (nb1 > nb2):
   smaller = nb2
else:
   smaller = nb1

print("The smaller number between", nb1, "and",
nb2, "is:", smaller)
```

Applications

Exercise: Find the maximum of 3 values

Write a program to ask the user to give 3 numbers and returns the maximum value.

Applications

Exercise: Find the maximum of 3 values

```
print("You will have to give 3 values ...")
print("Please give the first number")
nb1 = float(input())
print("Please give the second number")
nb2 = float(input())
print("Please give the third number")
nb3 = float(input())

maxValue = nb1
if (nb2 > maxValue):
   maxValue = nb2
if (nb3 > maxValue):
   maxValue = nb3

print("The maximal value is ", maxValue)
```

Applications

Exercise: Setup and thresholds

You want a program to help you to set the parameters sound intensity (in dB) and duration (in sec) of your experiment.
You have default values : defaultSound = 20 and defaultDuration = 30.
Write a program that ask you the current values for the sound the time and give the following instructions :
- if the sound intensity is inferior and the duration superior or equal to the default values : ask to stop the experiment
- if the sound is inferior to the default value : ask to increase the sound value
- if the duration is superior to the default value : ask to reduce the duration of the experiment
- if the duration is inferior to the default value and superior to 2 seconds : ask to restart
- otherwise : display that everything is ok

Applications

Exercise: Setup and thresholds

```
defaultSound = 20
defaultDuration = 30
minDuration = 2
print("Current value for the sound intensity (in dB) ?")
sound = float(input())
print("Current value for the duration (in sec) ?")
duration = float(input())

if ((sound < defaultSound) and (duration >= defaultDuration)):
   print("Please stop the experiment")
elif (sound < defaultSound):
   print("Please increase the sound intensity")
elif (duration > defaultDuration):
   print("Please reduce the time of the experiment")
elif (minDuration < duration < defaultDuration):
   print("Please restart the experiment")
else:
   print("Everything is ok")
```

Applications

Exercise: Basic loops operations

Set 2 integers : `i = 0` and `j = 15`
Write a first loop showing and increasing the value of `i` as long as it remains lower than `j`.
Write a second loop that, as long as `j` is not zero, decreases the value of `j` and displays its value if it is even.

Applications

Exercise: Basic loops operations

```
i = 0
j = 15

print("First loop")
while (i < j):
    print("i=", i)
    i += 1

print("Second loop")
while (j != 0):
    j -= 1
    if (j % 2 == 0):
        print("j=", j)
```

Applications

Exercise: Max and sum of a list

Write a program to ask the user to give 5 numbers (floats or integers) and display the maximum value and the sum of all the numbers given by the user.
You will write two possible versions for the maximum and the sum.

Applications

Exercise: Max and sum of a list (version 1)

```
l = []
for i in range(5):
    print("Please give a number")
    nb = float(input())
    l.append(nb)

# Find max (version 1)
print("The maximum value is", max(l))

# Sum all values (version 1)
print("The sum of all your values is", sum(l))
```

Applications

Exercise: Max and sum of a list (version 2)

```
l = []
for i in range(5):
    print("Please give a number")
    nb = float(input())
    l.append(nb)

# Find max (version 2)
tempMax = 0
for i in range(len(l)):
    if (l[i] > tempMax):
        tempMax = l[i]
print("The maximum value is", tempMax)

# Sum all values (version 2)
total = 0
for i in range(len(l)):
    total += l[i]
print("The sum of all your values is", total)
```

Applications

Exercise: Mean of a list

Write a program to ask the user to give 5 numbers and display the mean of these numbers.
You will write two possible versions of this program.

Applications

Exercise: Mean of a list

```
l = []
for i in range(5):
    print("Please give a number")
    nb = float(input())
    l.append(nb)

# Mean
result = sum(l) / len(l)
print("The mean value is", result)
```

Applications

Exercise: How many letters in a sentence?

Write a program that asks you to write a short sentence.
Use a for loop to display each letter one by one.

Then displays the number of letters in the sentence (the spaces are not taken into account).

Applications

Exercise: How many letters in a sentence?

```
print("Please write a sentence")
sentence = input()
cpt = 0

for letter in sentence:
    print(letter)
    if (letter != " "):
        cpt += 1

print("They are", cpt, "letters in this sentence")
```

Applications

Exercise: DNA to RNA

Transform DNA to RNA :

myDNA = "ATGCATGCAGCATGCAT"

Write a code that transform this DNA into RNA. You will need to transform first this string into a list and then apply some conditions for the "RNA conversion": A→U, C→G, G→C, T→A

Applications

Exercise: DNA to RNA

```
myDNA = "ATGCATGCAGCATGCAT"
dna = list(myDNA)
rna = []
    for i in range(len(dna)):
        if (dna[i] == "A"):
            rna.append("U")
        elif (dna[i] == "C"):
            rna.append("G")
        elif (dna[i] == "G"):
            rna.append("C")
        elif (dna[i] == "T"):
            rna.append("A")
        else:
            print("Wrong AA:", dna[i])
print("DNA:", dna)
print("RNA:", rna)
```

Applications

Exercise: Is Methionine in my DNA sample?

You have some DNA sample and you want to know if the codon Methionine (ATG) is inside.
Write a program that helps you find it and if yes, displays the position of the codon in your DNA sample.

dna = "ATGCATGCAGCATAGCAT"

Applications

Exercise: Is Methionine in my DNA sample?

```
dna = "ATGCATGCAGCATAGCAT"
methionine = "ATG"

if (methionine in dna):
    print("Methionine found")

# Version 1
for i in range(len(dna)):
    if (i+2 < len(dna)):
        if ((dna[i]=="A") and (dna[i+1]=="T") and (dna[i+2]=="G")):
            print("Methionine found at position", i)

# Version 2
for i in range(len(dna)):
    if (dna[i:i+3] == methionine):
        print("Methionine found at position", i)
```

Applications

Exercise: Lab schedule manager (version 1)

You will write a program that asks the user to enter 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)

Applications

Exercise: Lab schedule manager (version 1)

```
print("Please give the hour of the experiment beginning")
hour = float(input())
print("Please give the minutes of the experiment beginning")
minutes = float(input())
print("Please give the duration of your experiment (in
minutes)")
duration = float(input())

finalMinutes = minutes + duration
cptHours = 0
while (finalMinutes >= 60):
    finalMinutes = finalMinutes - 60
    cptHours += 1
finalHours = hour + cptHours
print("Your experiment will end at", int(finalHours), ":",
  int(finalMinutes))
```

Applications

Exercise: Lab schedule manager (version 2)

You will write a program that asks the user to enter 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.

Applications

Exercise: Lab schedule manager (version 2)

```
print("Please give the hour of the experiment beginning")
hour = float(input())
while not (0 <= hour <= 23):
    print("Please give a correct time hour (0 to 23)")
    hour = float(input())
print("Please give the minutes of the experiment beginning")
minutes = float(input())
while not (0 <= minutes <= 59):
    print("Please give a correct minutes value (0 to 59)")
    minutes = float(input())
print("Please give the duration of your experiment (in minutes)")
duration = float(input())

finalMinutes = minutes + duration
cptHours = 0
while (finalMinutes >= 60):
    finalMinutes = finalMinutes - 60
    cptHours += 1
finalHours = hour + cptHours
print("Your experiment will end at", int(finalHours), ":", int(finalMinutes))
```

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

```
lab = {"mickey":[], "minie":[]}
days = 2

for animal in lab.keys():
   for i in range(days):
      print("Give a weight (in g) for", animal, "at
day", i)
      weight = float(input())
      lab[animal].append(weight)

print(lab)
```

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"
```

7. Dictionaries

Application

```
lab = {"mickey":[], "minie":[]}
days = 2

for animal in lab.keys():
    for i in range(days):
        print("Give a weight (in g) for", animal, "at day", i)
        weight = float(input())
        lab[animal].append(weight)

print(lab)

for day in range(days):
    for animal in lab.keys():
        weight = lab[animal][day]
        print("At day %d, the weight of %s was %f g" %(day,
            animal, weight))
```

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" if the `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

```
def divisible_By_10_But_Not_3(nb):
    if ((nb % 10 == 0) and not (nb % 3 == 0)):
        print("Yes")
    else:
        print("No")

divisible_By_10_But_Not_3(70)
divisible_By_10_But_Not_3(33)
```

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
```

8. Functions

Exercise

```
def divisible_By_10_But_Not_3(nb):
    if (nb % 10 == 0):
        if not (nb % 3 == 0):
            print(nb, "is divisible by 10 but not 3 => yes")
        else:
            print(nb, "is divisible by 10 and 3 => no")
    else:
        if not (nb % 3 == 0):
            print(nb, "is divisible neither by 10 nor 3 => no")
        else:
            print(nb, "is not divisible by 10 and but is divisible 
by 3 => no")


divisible_By_10_But_Not_3(30)
divisible_By_10_But_Not_3(33)
divisible_By_10_But_Not_3(37)
divisible_By_10_But_Not_3(70)
```

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 :

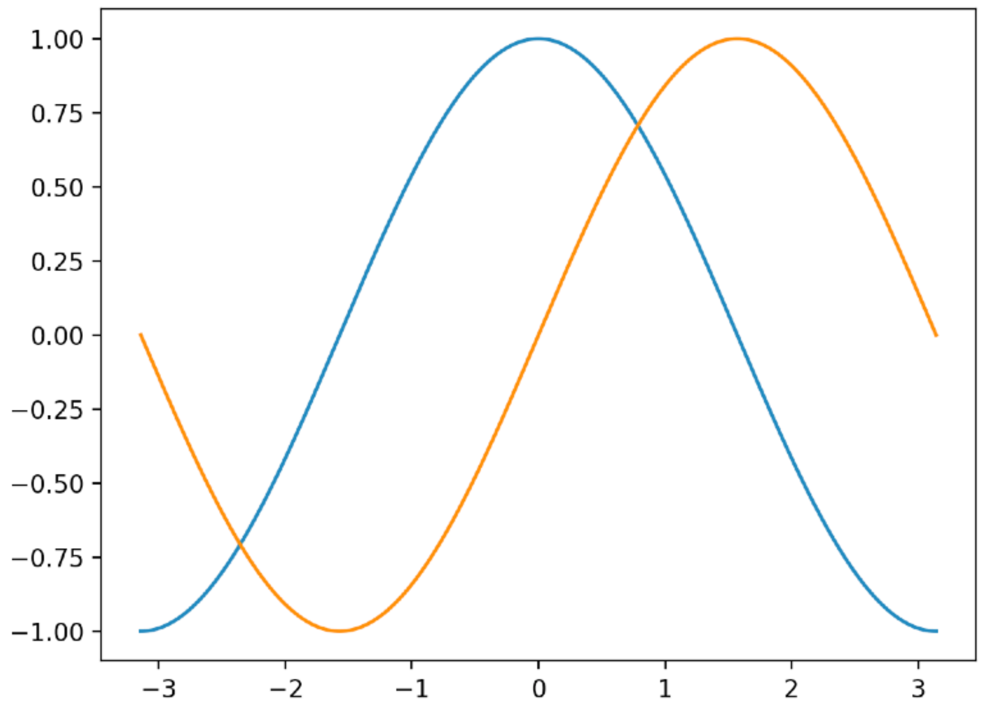
```
from pylab import *
import numpy as np
```

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 numpy 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")`

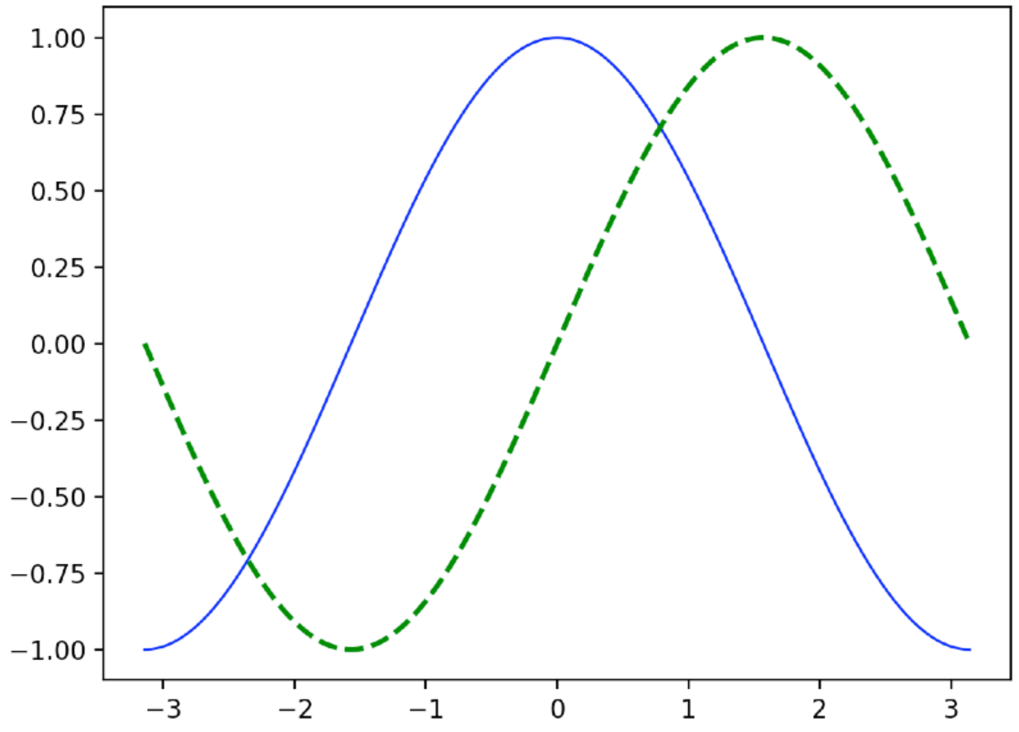
```
from pylab import *
import numpy as np

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="--")
```



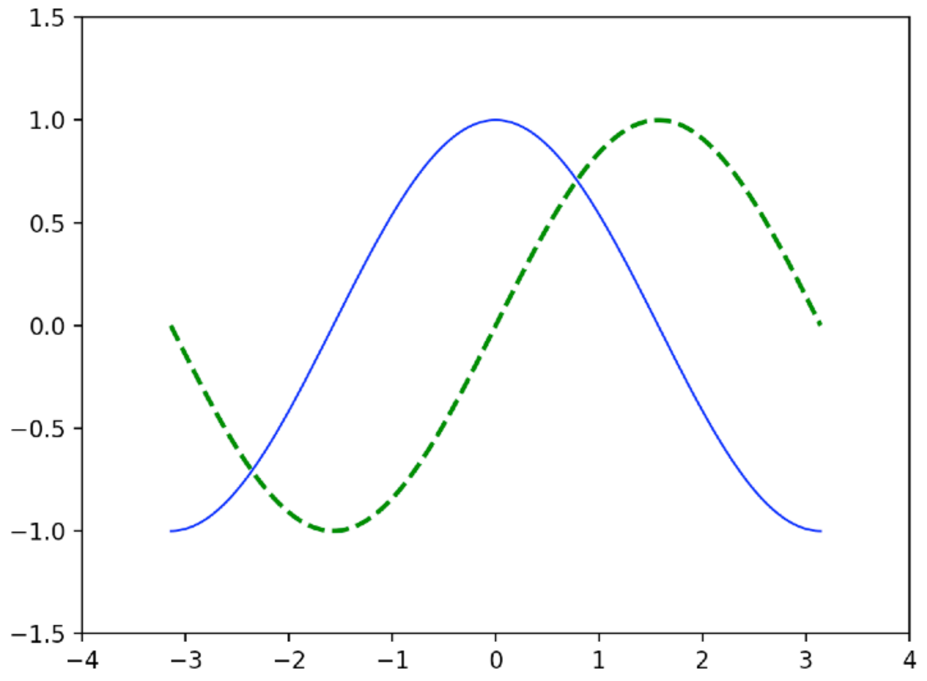
10. Matplotlib

Simple graphs

Change x and y limits :

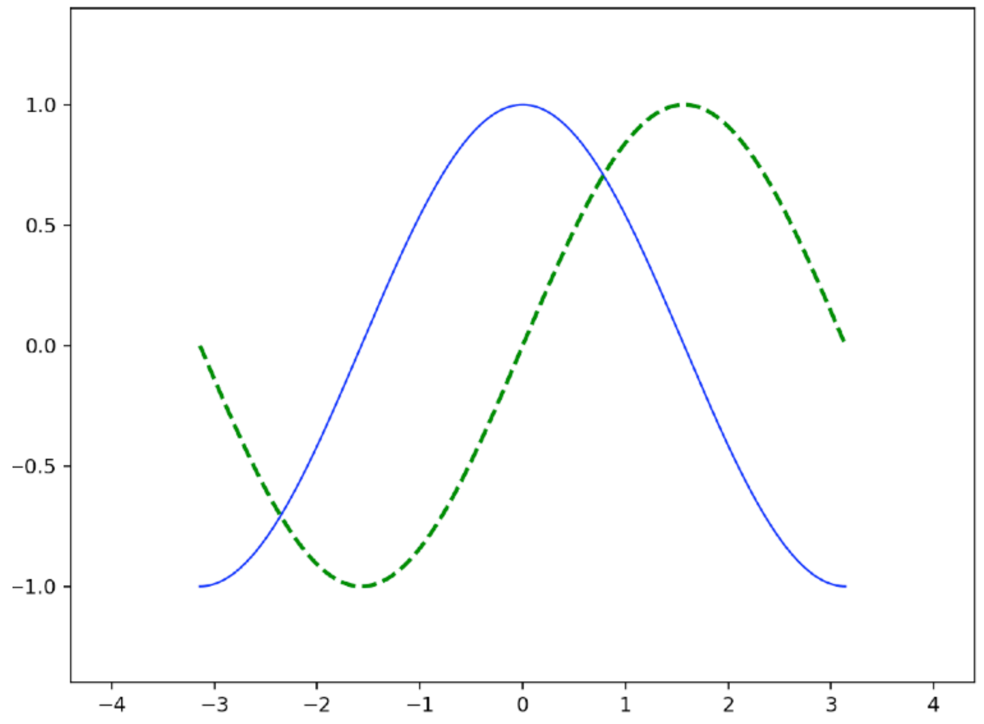
Manually :

```
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)
```

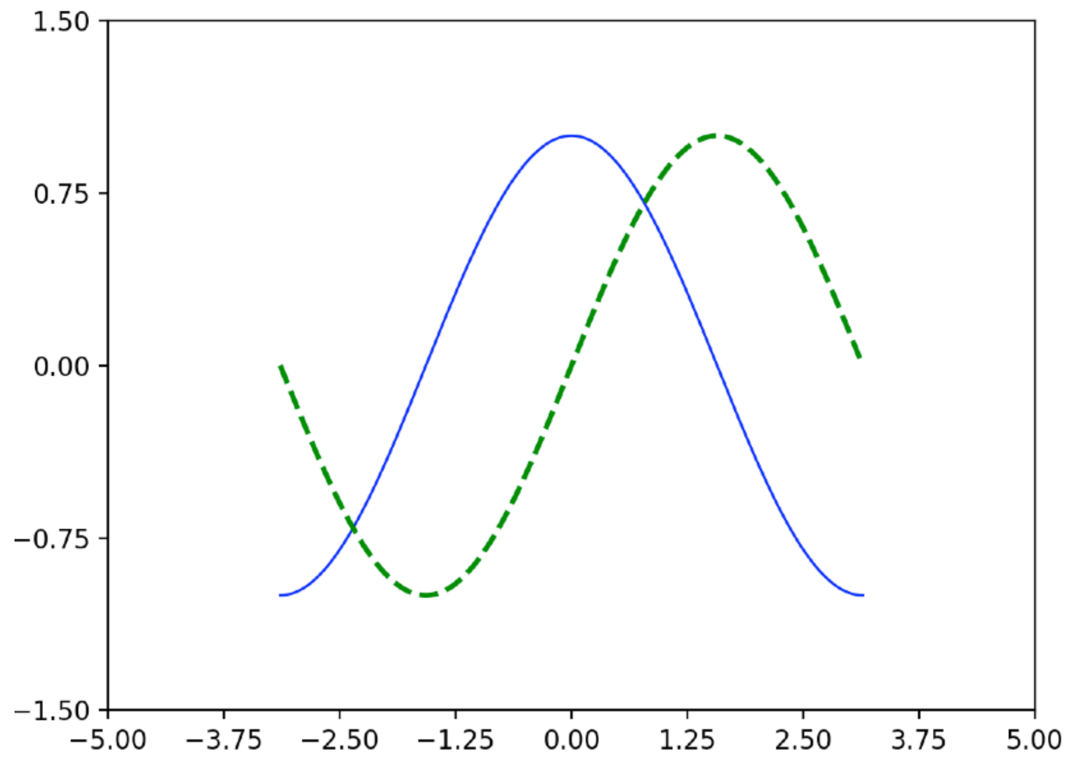


10. Matplotlib

Simple graphs

Change the graduations :

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



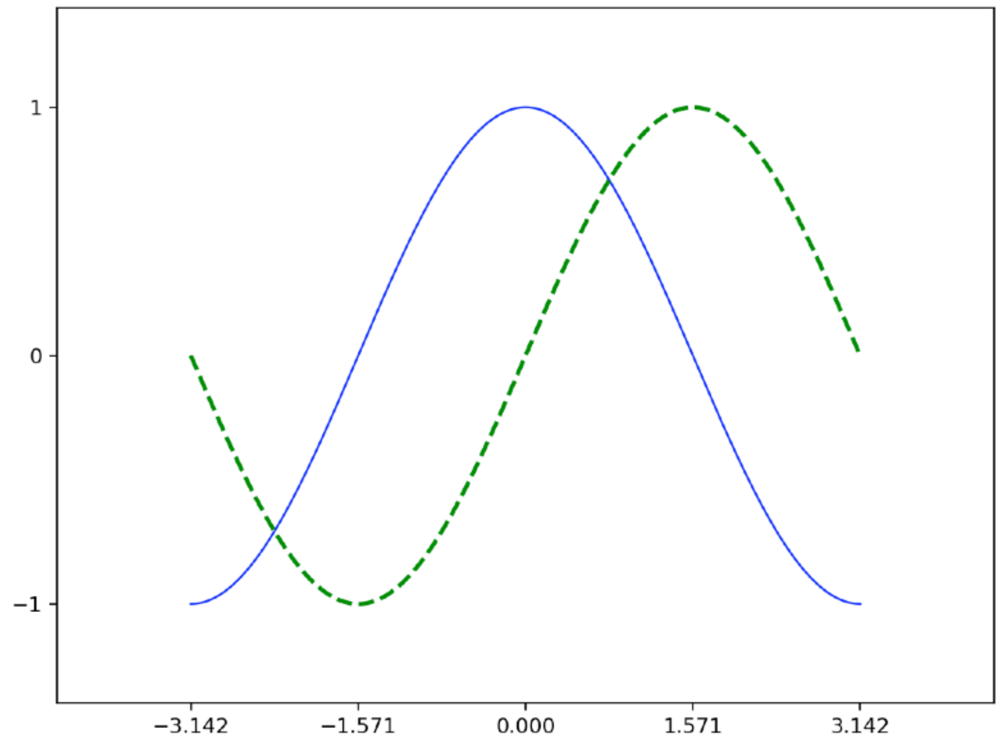
10. Matplotlib

Simple graphs

Change the graduations :

Automatically :

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



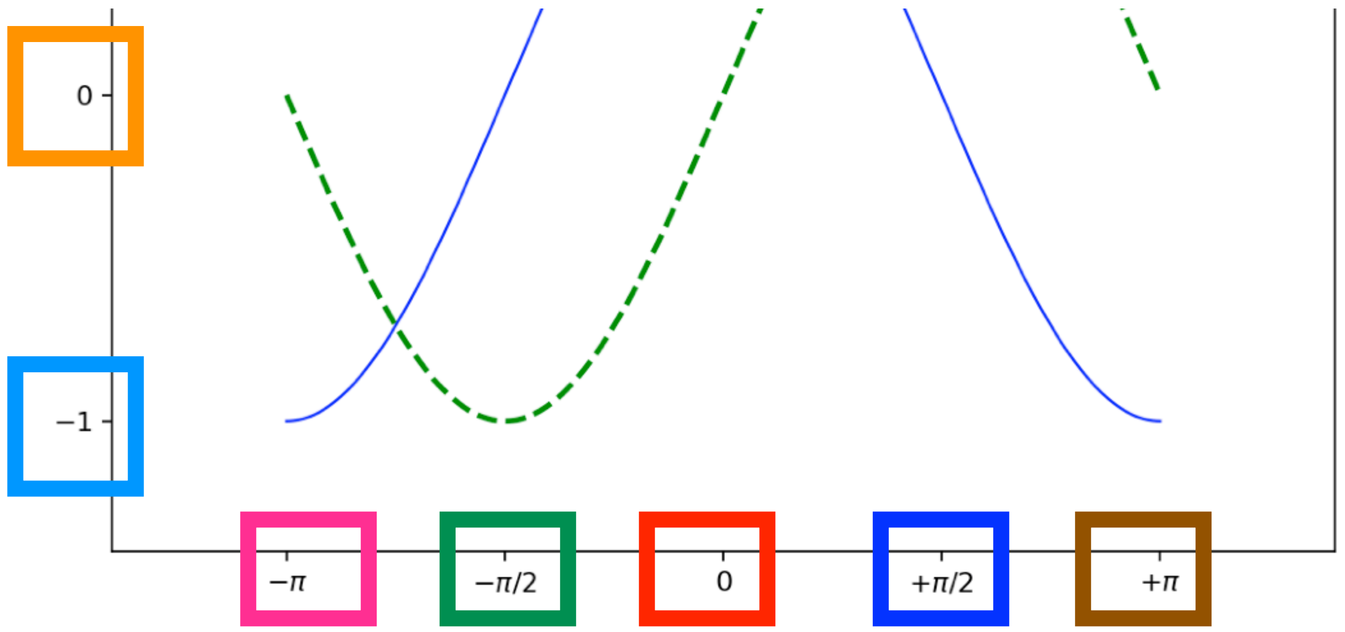
→ Now we want **π** instead of 3.142

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'$+1$'])
```

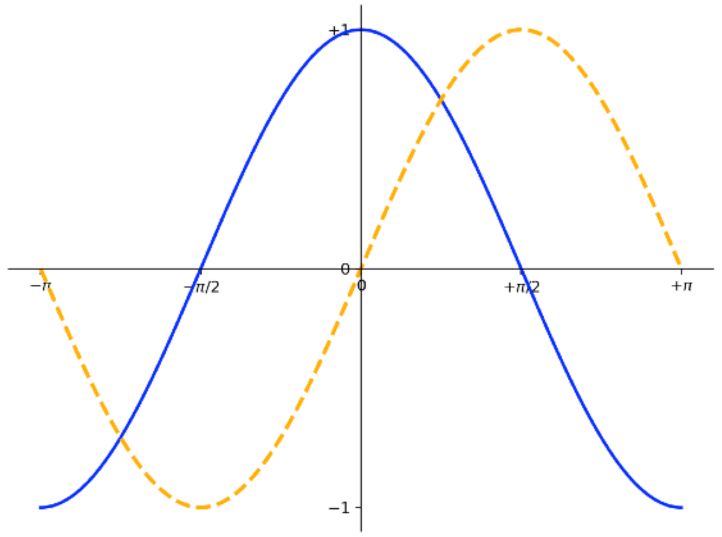


10. Matplotlib

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'$+1$'])
```

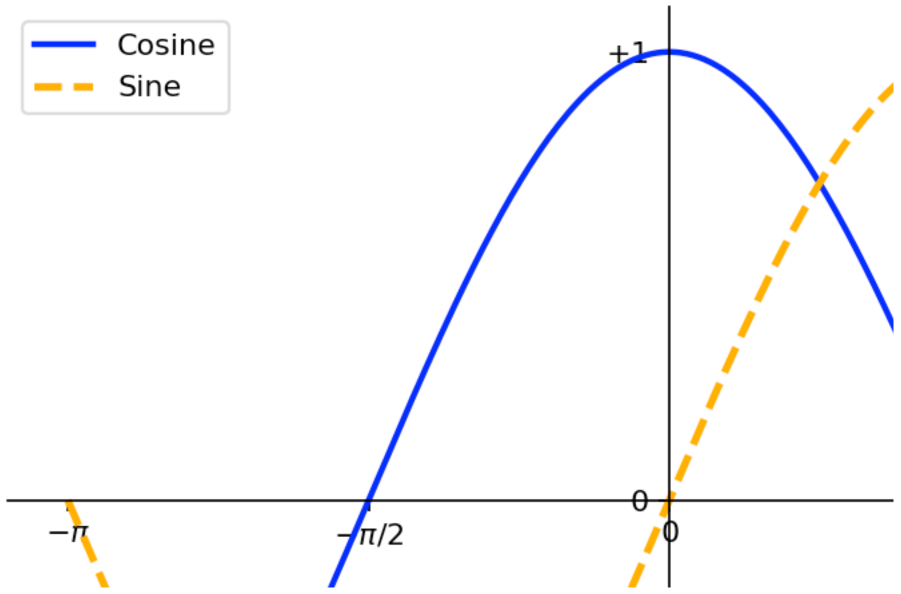


10. Matplotlib

Simple graphs

Add a legend :

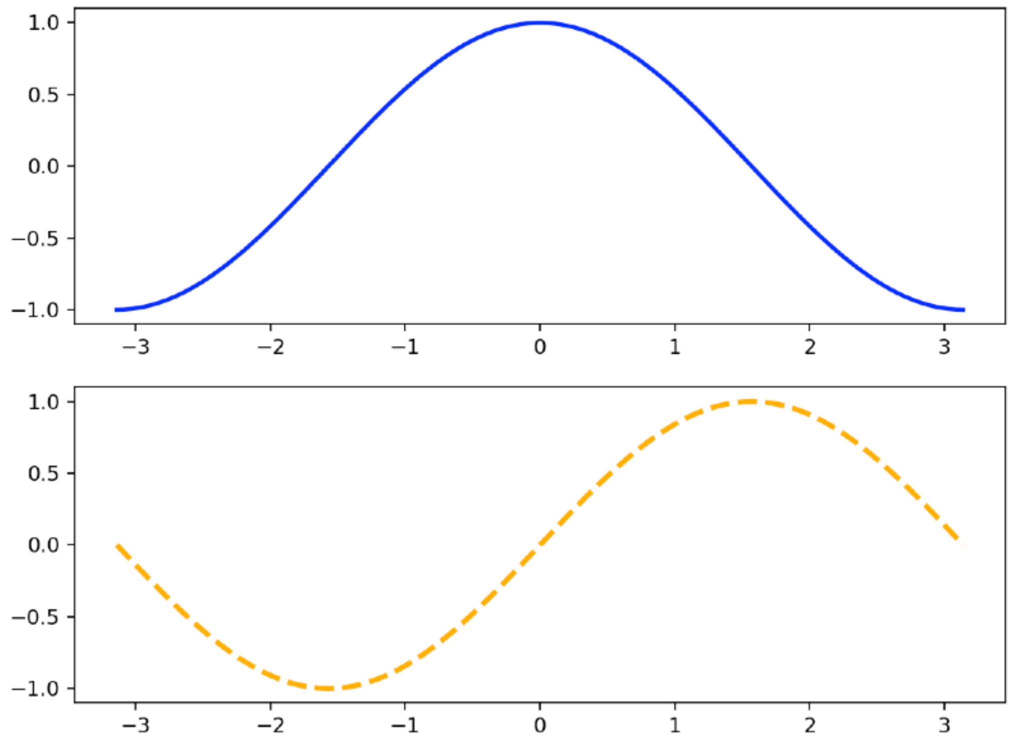
```
plot(X, cosX, color="blue", linewidth=2.0,
      linestyle="-", label="Cosine")
plot(X, sinX, color="orange", linewidth=2.5,
      linestyle="--", label="Sine")
...
legend(loc='upper left')
show()
```



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)`



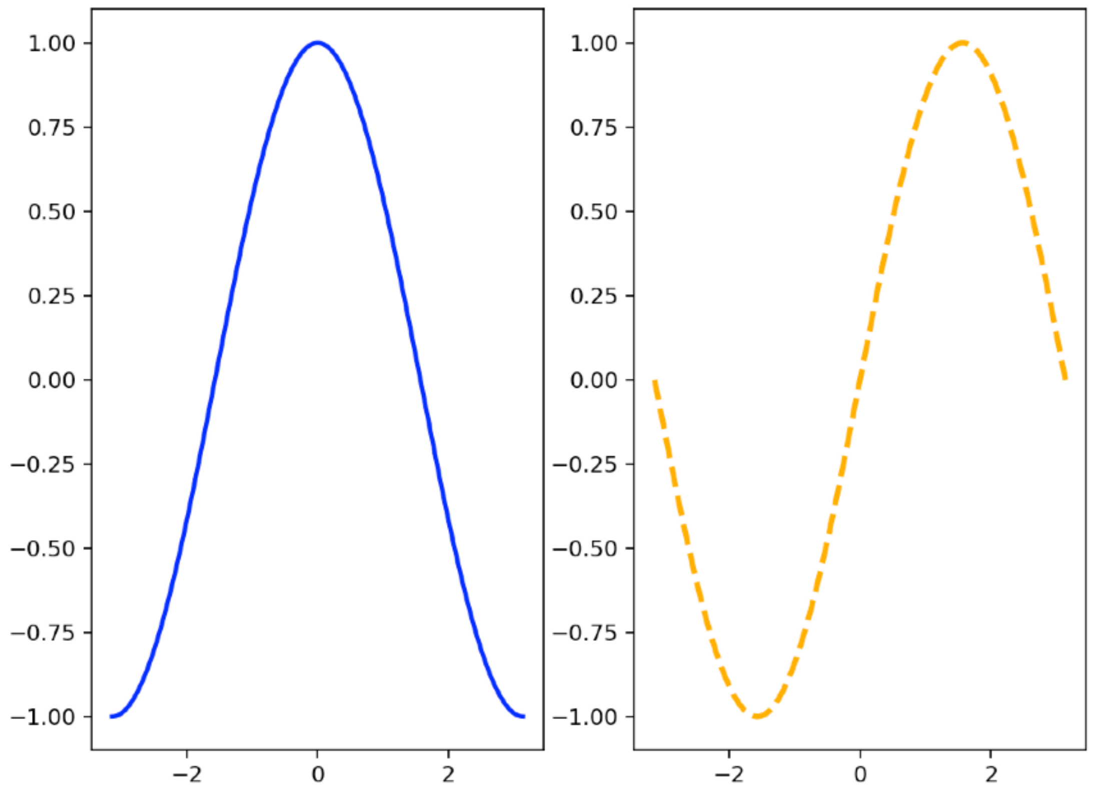
```
subplot(2, 1, 1)
plot(X, cosX, color="blue", linewidth=2.0,
    linestyle="-", label="Cosine")
subplot(2, 1, 2)
plot(X, sinX, color="orange", linewidth=2.5,
    linestyle="--", label="Sine")
```

10. Matplotlib

Multi-plots

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

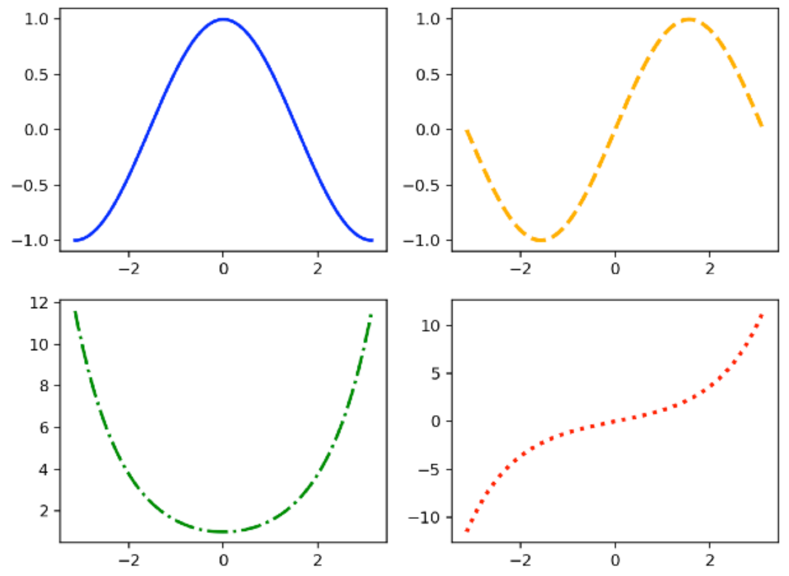
```
subplot(nbRows, nbCol, order)
```



```
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



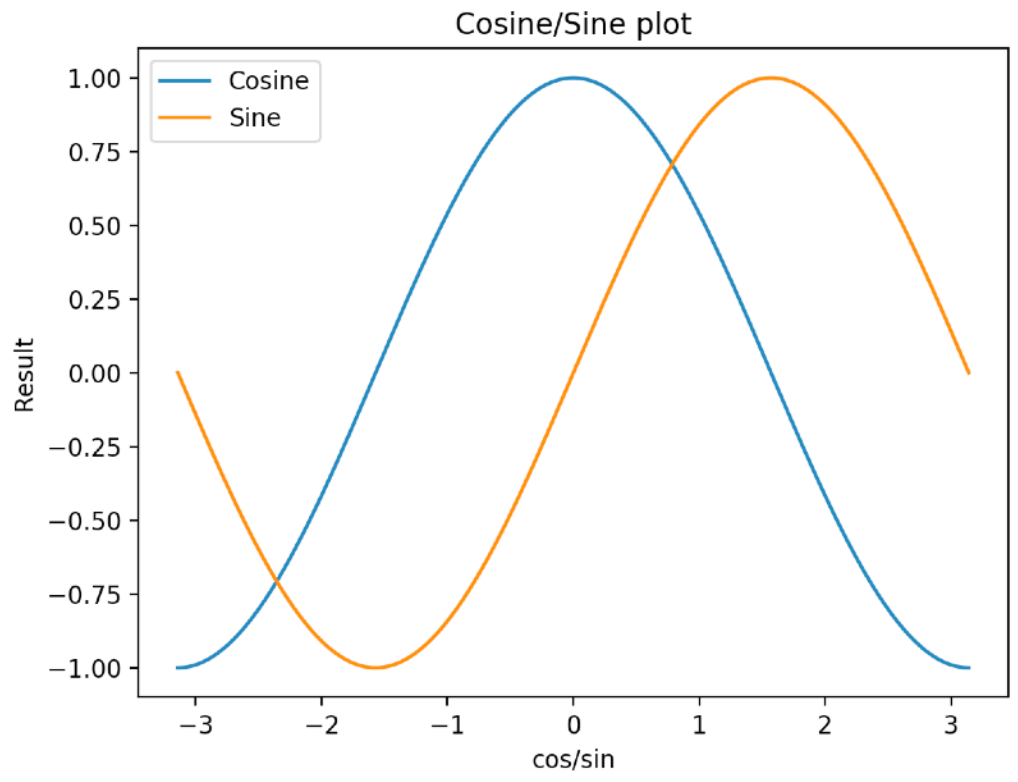
```
subplot(2, 2, 1)
plot(X, cosX, color="blue", linewidth=2.0, linestyle="-",
     label="Cosine")
subplot(2, 2, 2)
plot(X, sinX, color="orange", linewidth=2.5, linestyle="--",
     label="Sine")
subplot(2, 2, 3)
plot(X, coshX, color="green", linewidth=2.0, linestyle="-.",
     label="Hyperbolic cosine")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

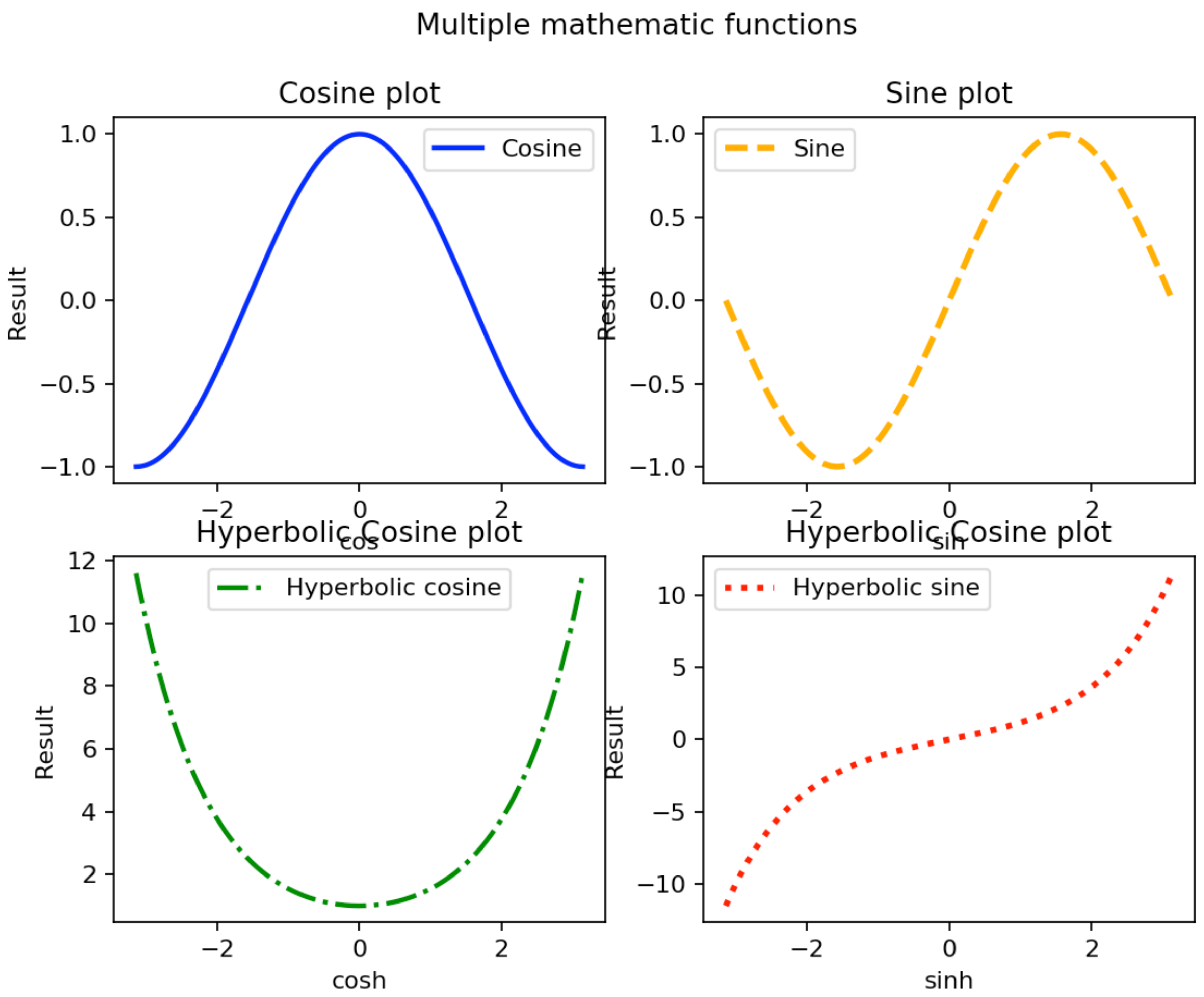
```
from pylab import *
import numpy as np

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()
```



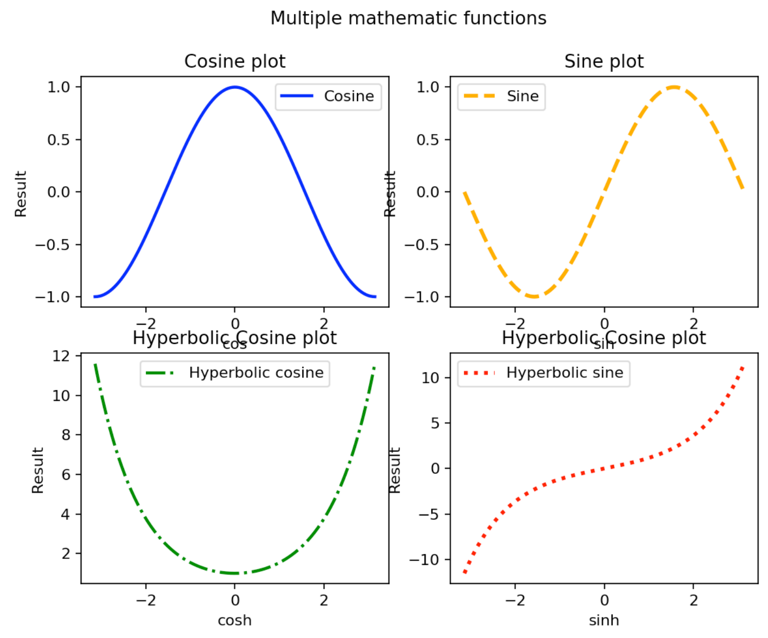
10. Matplotlib

Multi-plots

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



10. Matplotlib



Multi-plots

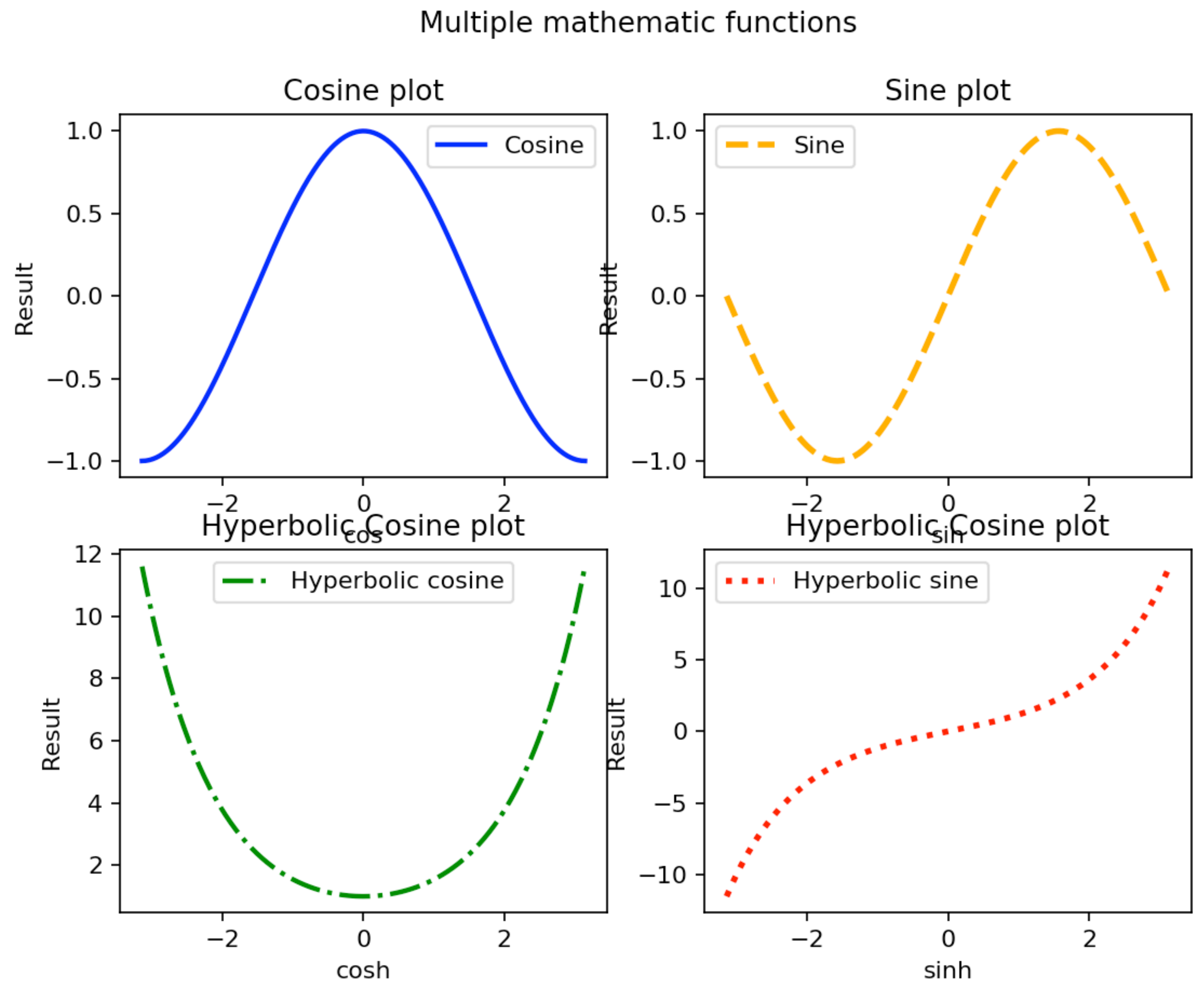
With legends, labels and titles :

```
figure(figsize=(8,6), dpi=80)
suptitle("Multiple mathematic functions")

subplot(2, 2, 1)
plot(X, cosX, color="blue", linewidth=2.0, linestyle="-", label="Cosine")
legend()
title("Cosine plot")
xlabel("cos")
ylabel("Result")
subplot(2, 2, 2)
plot(X, sinX, color="orange", linewidth=2.5, linestyle="--", label="Sine")
legend()
title("Sine plot")
xlabel("sin")
ylabel("Result")
subplot(2, 2, 3)
plot(X, coshX, color="green", linewidth=2.0, linestyle="-.", label="Hyperbolic cosine")
legend()
title("Hyperbolic Cosine plot")
xlabel("cosh")
ylabel("Result")
subplot(2, 2, 4)
plot(X, sinhX, color="red", linewidth=2.5, linestyle=":", label="Hyperbolic sine")
legend()
title("Hyperbolic Cosine plot")
xlabel("sinh")
ylabel("Result")
```

10. Matplotlib

Multi-plots

With legends, labels and titles :

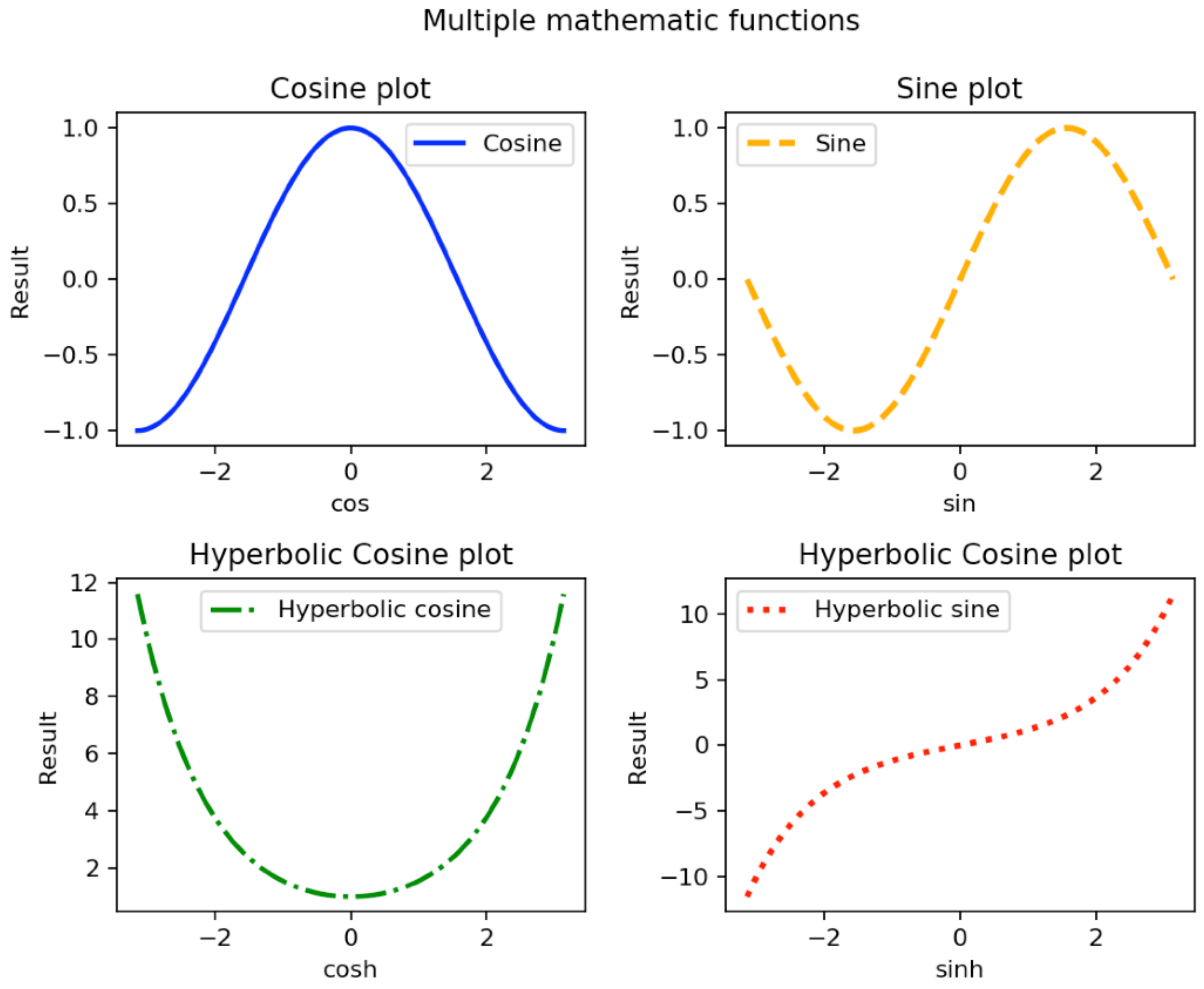


→ Problems of superposition

10. Matplotlib

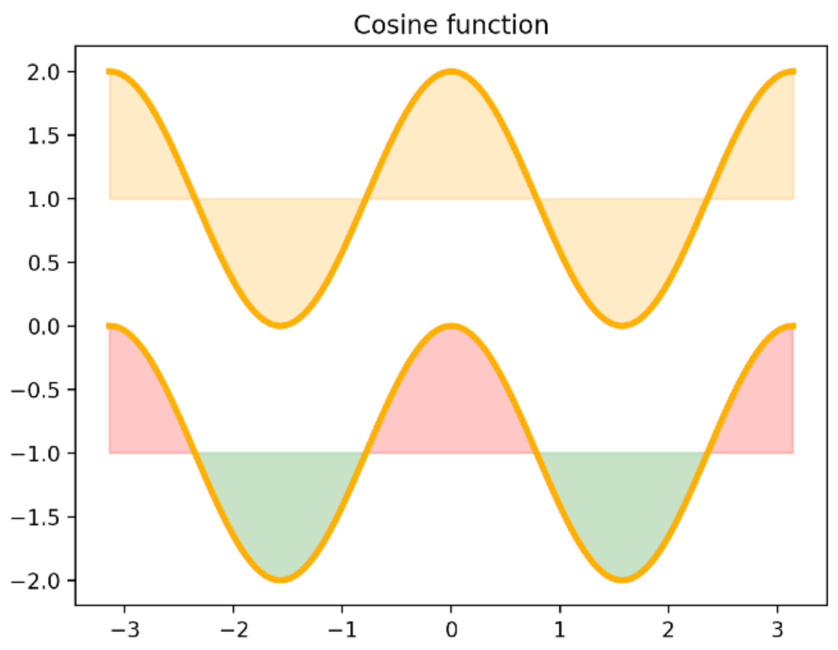
Multi-plots

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



10. Matplotlib

Exercise : "Filled" plot



From this code, try to obtain this figure :

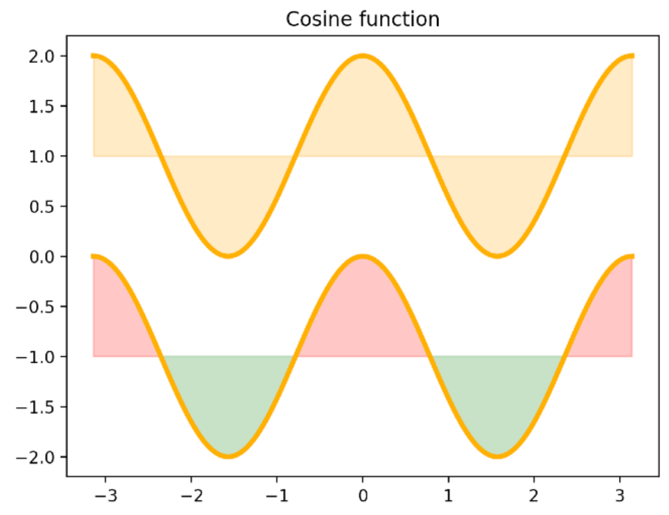
```
from pylab import *

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

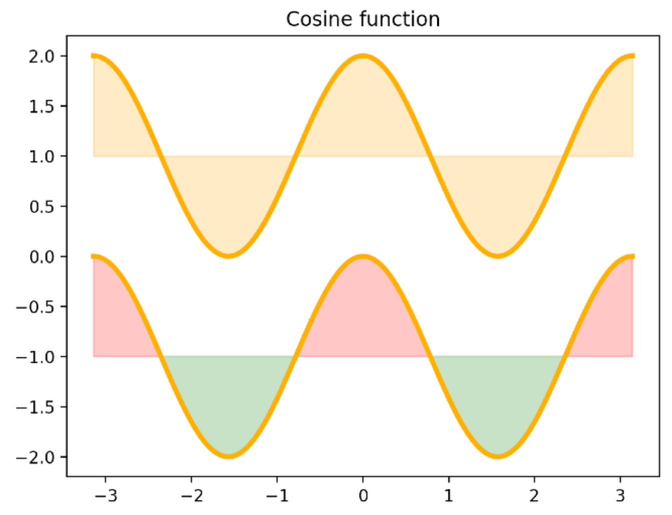
Exercise : "Filled" plot



```
from pylab import *

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

plot (X, Y+1, color='orange', 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)
fill_between(X, -1, Y-1, (Y-1) > -1, color='red', alpha=.25)
fill_between(X, -1, Y-1, (Y-1) < -1, color='green',  alpha=.25)

title("Cosine function")
show()
```

10. Matplotlib

Exercise : "Filled" plot



```
from pylab import *

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

plot (X, Y+1, color='orange', 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)
fill_between(X, -1, Y-1, (Y-1) > -1, color='red', alpha=.25)
fill_between(X, -1, Y-1, (Y-1) < -1, color='green',  alpha=.25)

title("Cosine function")
show()
```

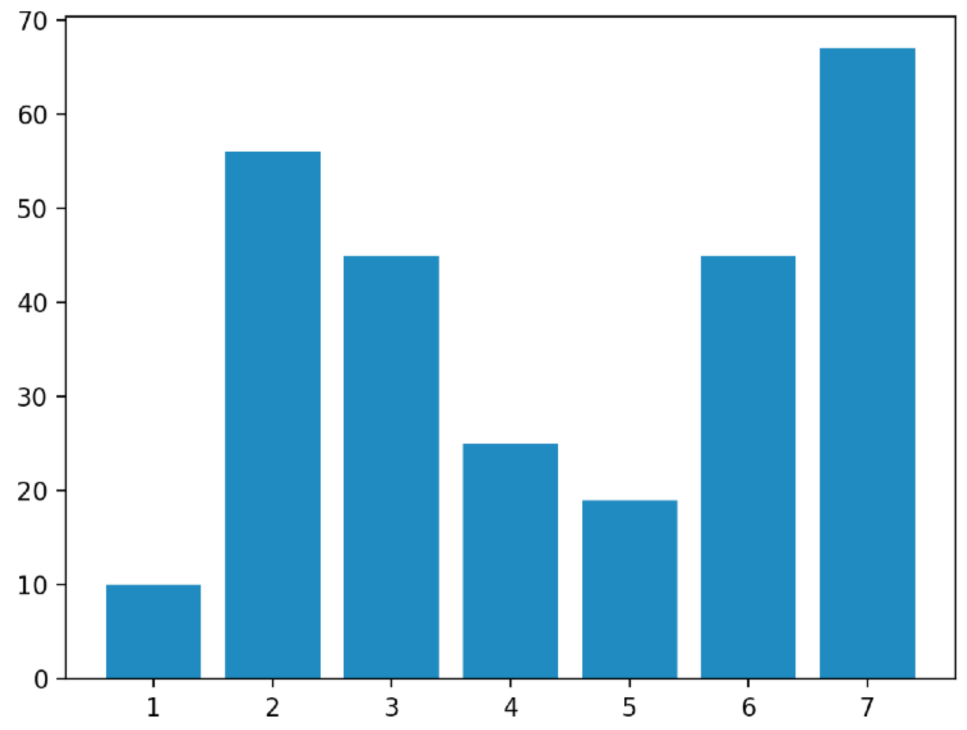
→ For more details : http://matplotlib.org

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()
```

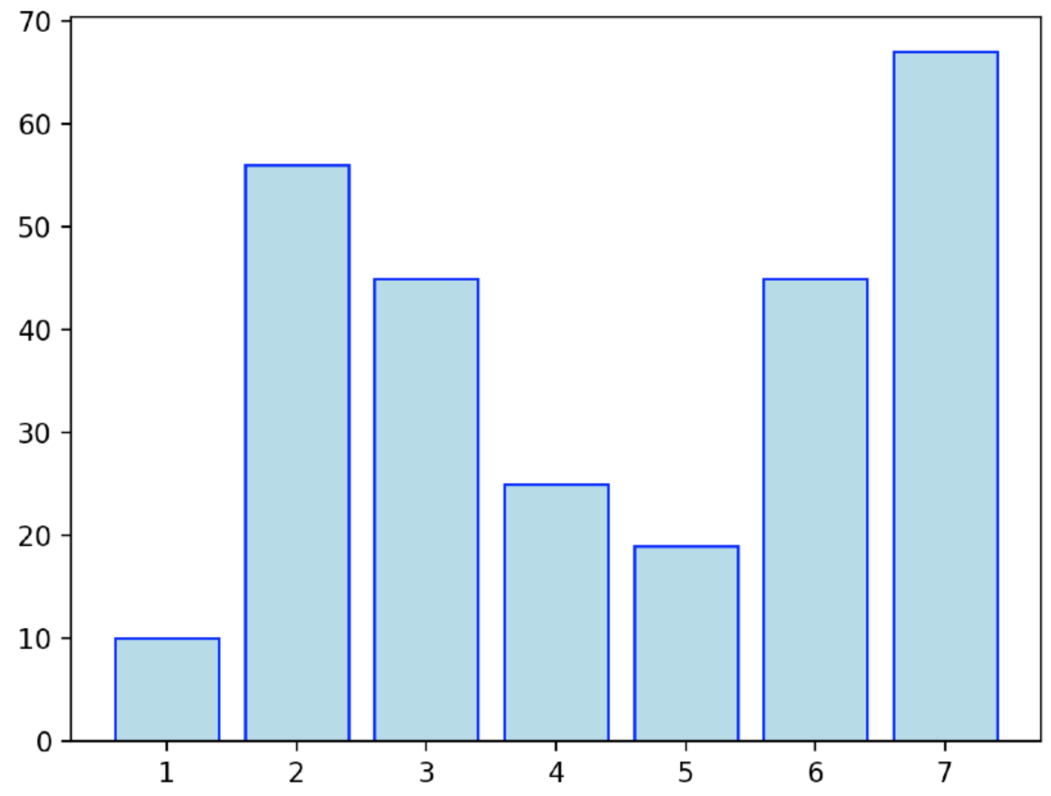


10. Matplotlib

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()
```

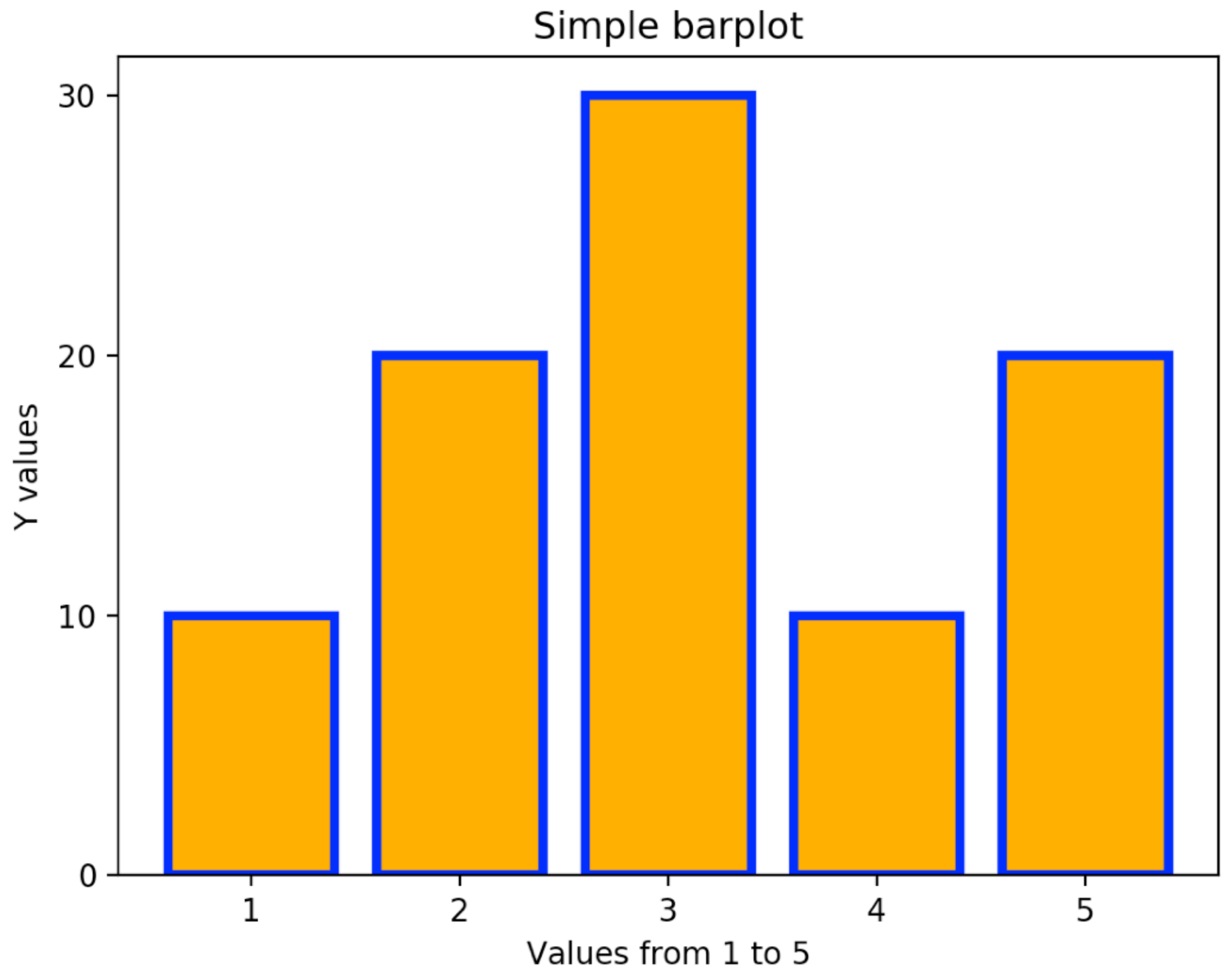


10. Matplotlib

Barplot

Exercise : try to obtain the exact same barplot

```
x = range(1, 6)
y = [10, 20, 30, 10, 20]
```



10. Matplotlib

Barplot

Exercise : try to obtain the exact same barplot

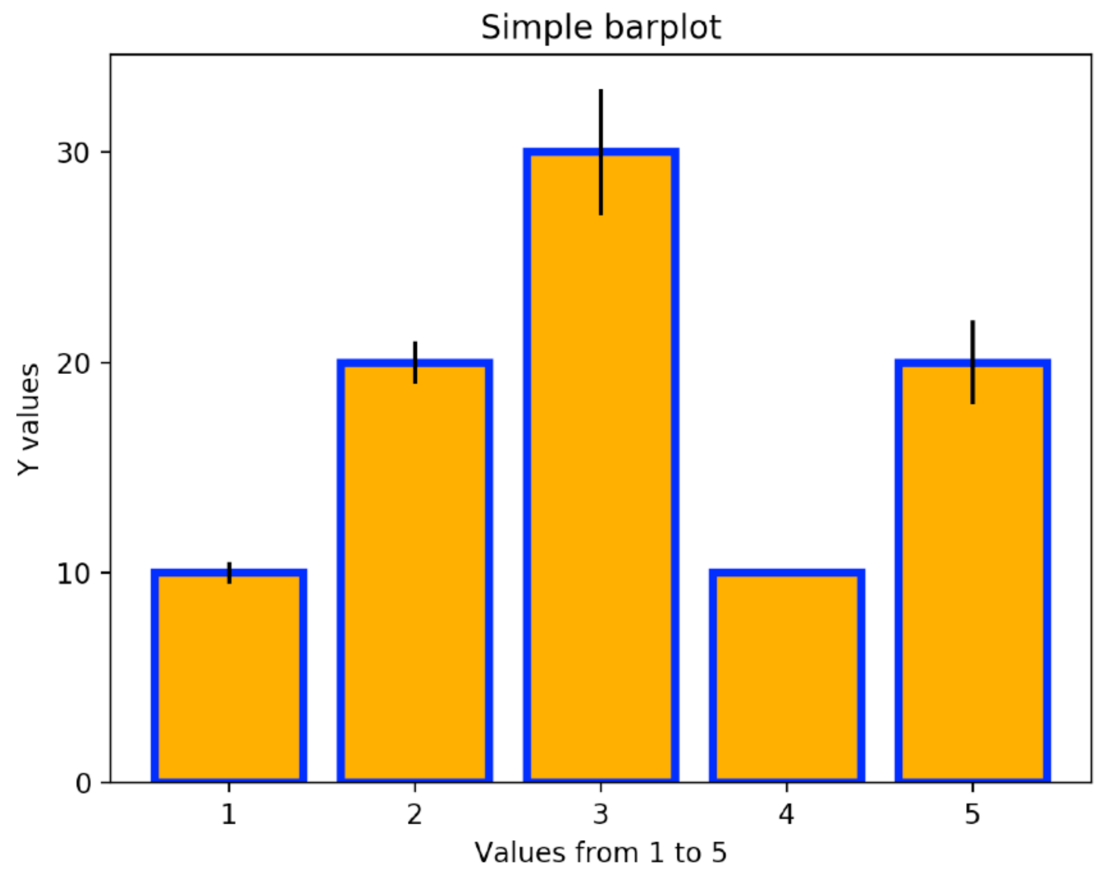
```
from pylab import *

x = range(1, 6)
y = [10, 20, 30, 10, 20]

bar(x, y, facecolor='orange', edgecolor='blue', linewidth=3)
title("Simple barplot")
xlabel("Values from 1 to 5")
ylabel("Y values")
yticks([0, 10, 20, 30])

show()
```

10. Matplotlib

Barplot

Error bars :

```
from pylab import *

x = range(1, 6)
y = [10, 20, 30, 10, 20]
y_error = [0.5, 1, 3, 0, 2]
```

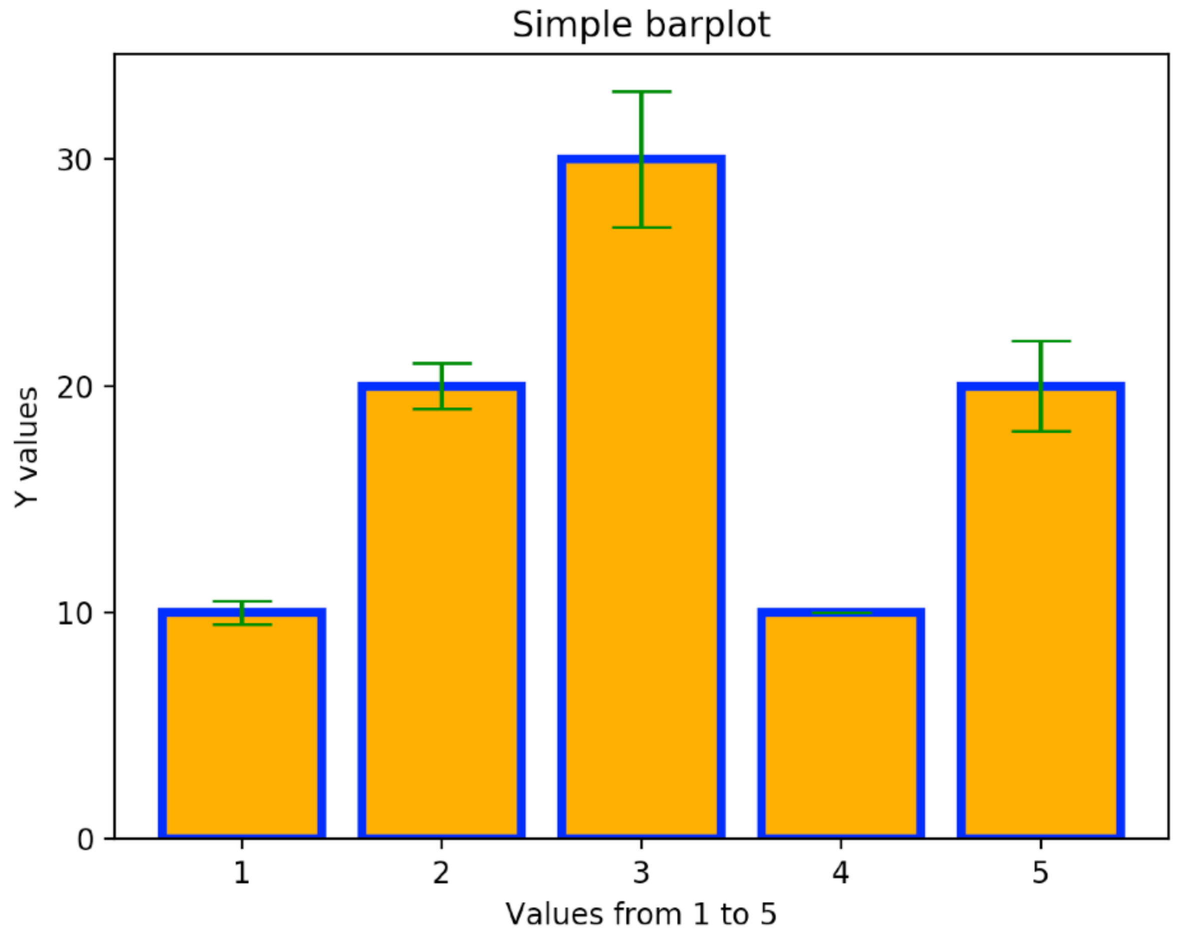


```
bar(x, y, facecolor='orange', edgecolor='blue', linewidth=3,
yerr=y_error)
title("Simple barplot")
xlabel("Values from 1 to 5")
ylabel("Y values")
yticks([0, 10, 20, 30])

show()
```

10. Matplotlib

Barplot

Error bars … can be customised too !

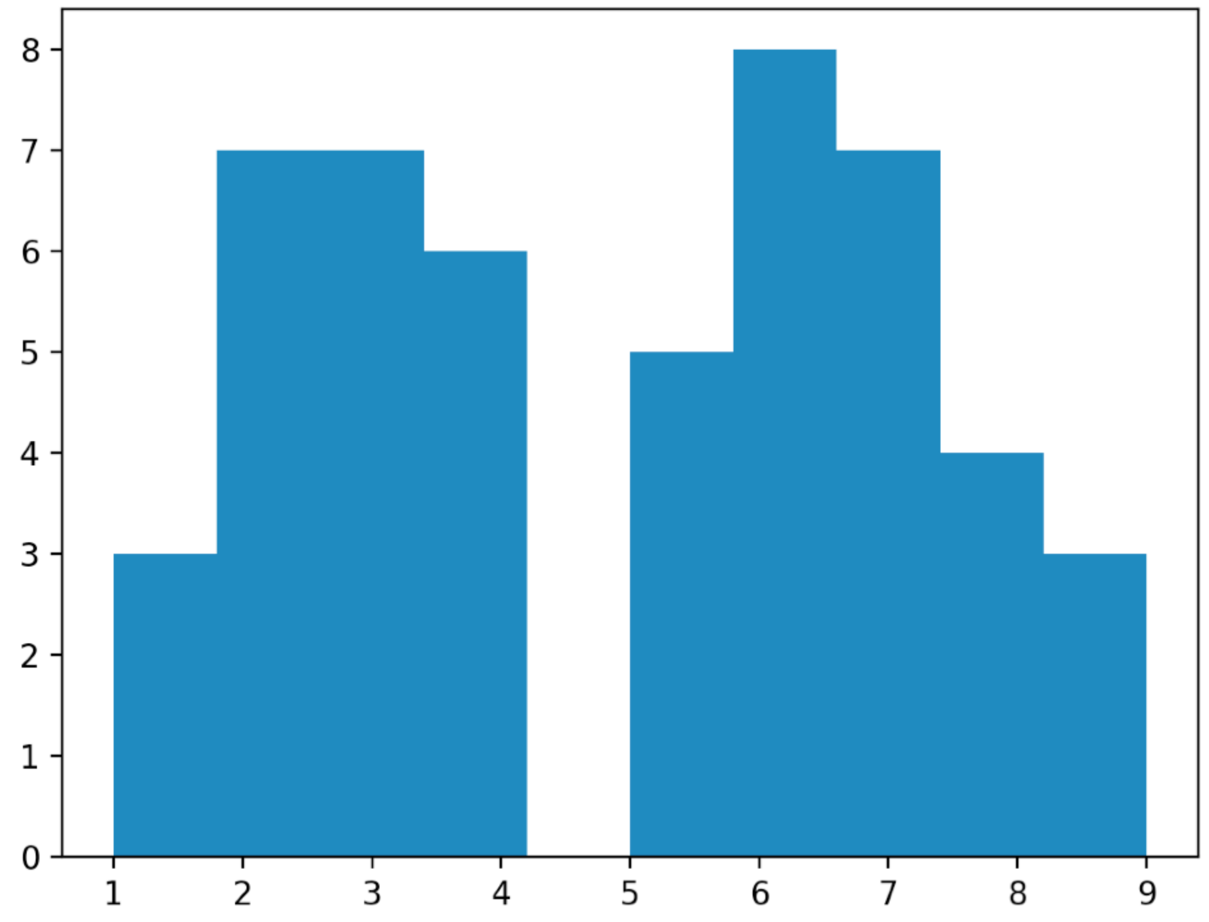


```
from pylab import *

x = range(1, 6)
y = [10, 20, 30, 10, 20]
y_error = [0.5, 1, 3, 0, 2]

bar(x, y, facecolor='orange', edgecolor='blue', linewidth=3,
   yerr=y_error, ecolor='green', capsize=10)
title("Simple barplot")
xlabel("Values from 1 to 5")
ylabel("Y values")
yticks([0, 10, 20, 30])

show()
```

10. Matplotlib

Histogram

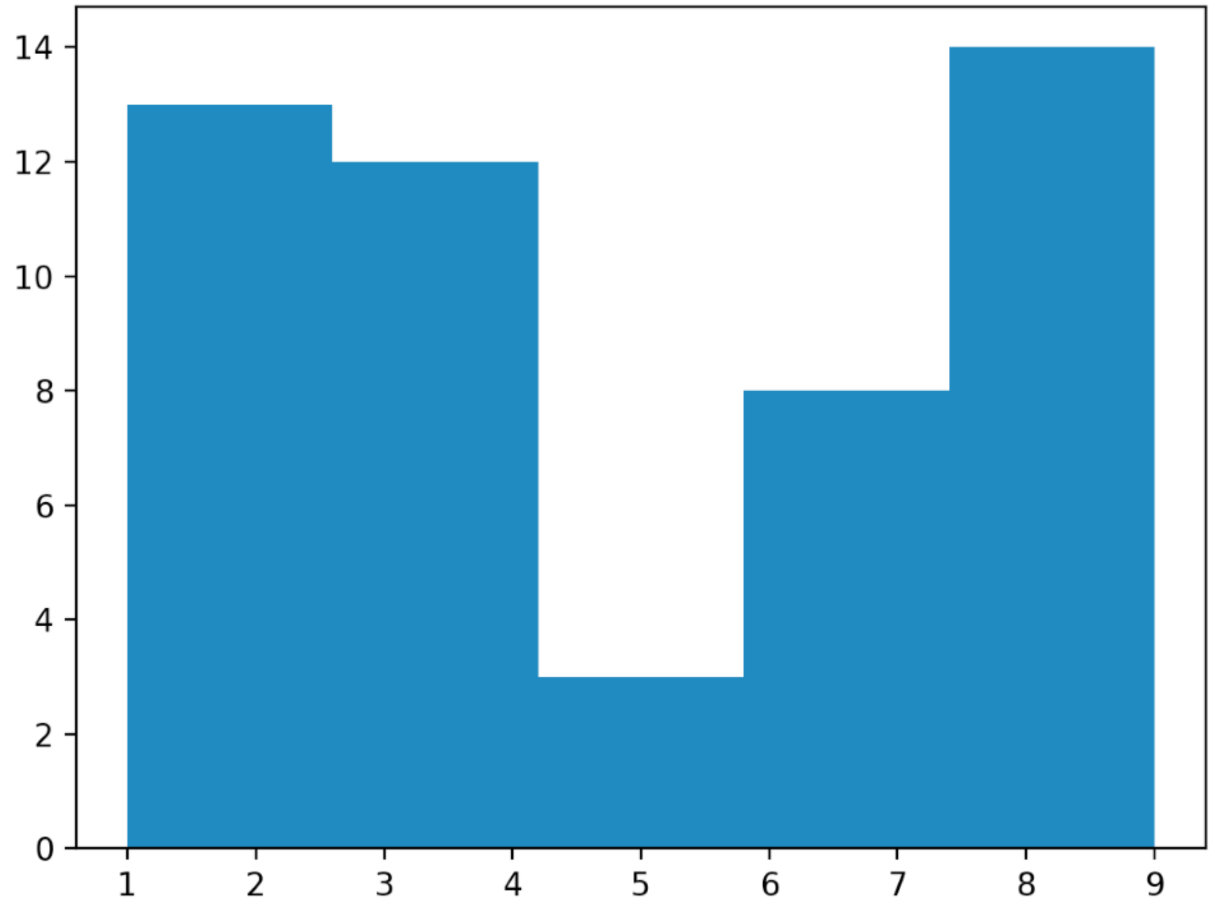
```
from pylab import *

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

hist(x)

show()
```



10. Matplotlib

Histogram

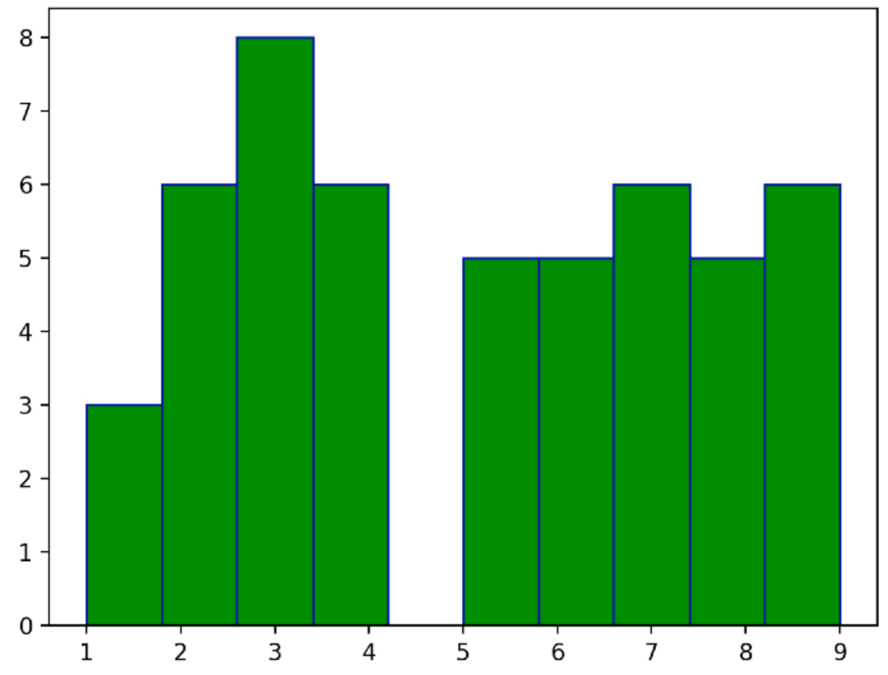
```
from pylab import *

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

hist(x, bins=5)

show()
```



10. Matplotlib

Histogram

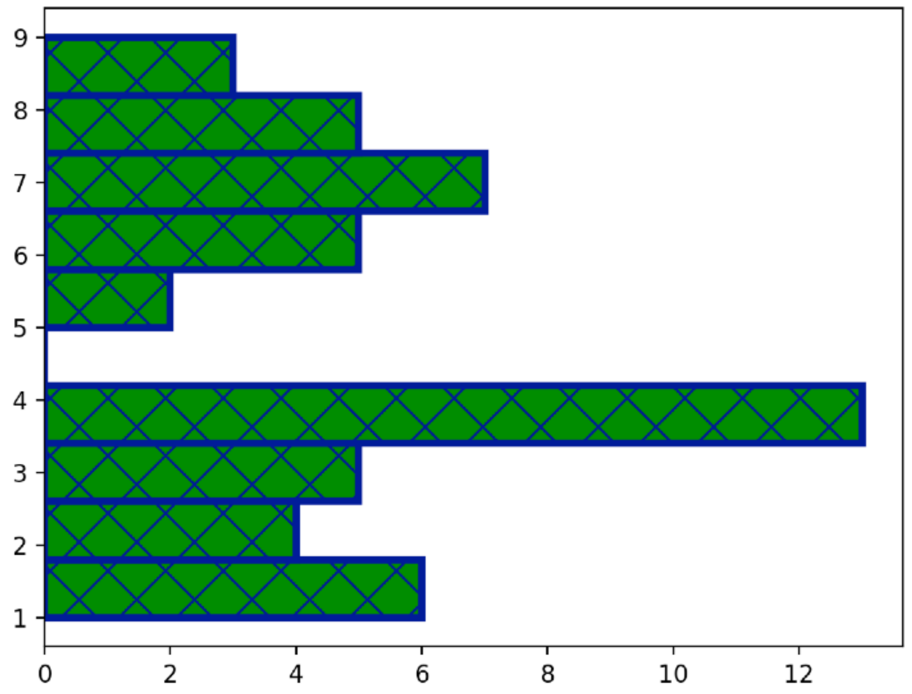
```
from pylab import *

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

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

show()
```



10. Matplotlib

Histogram

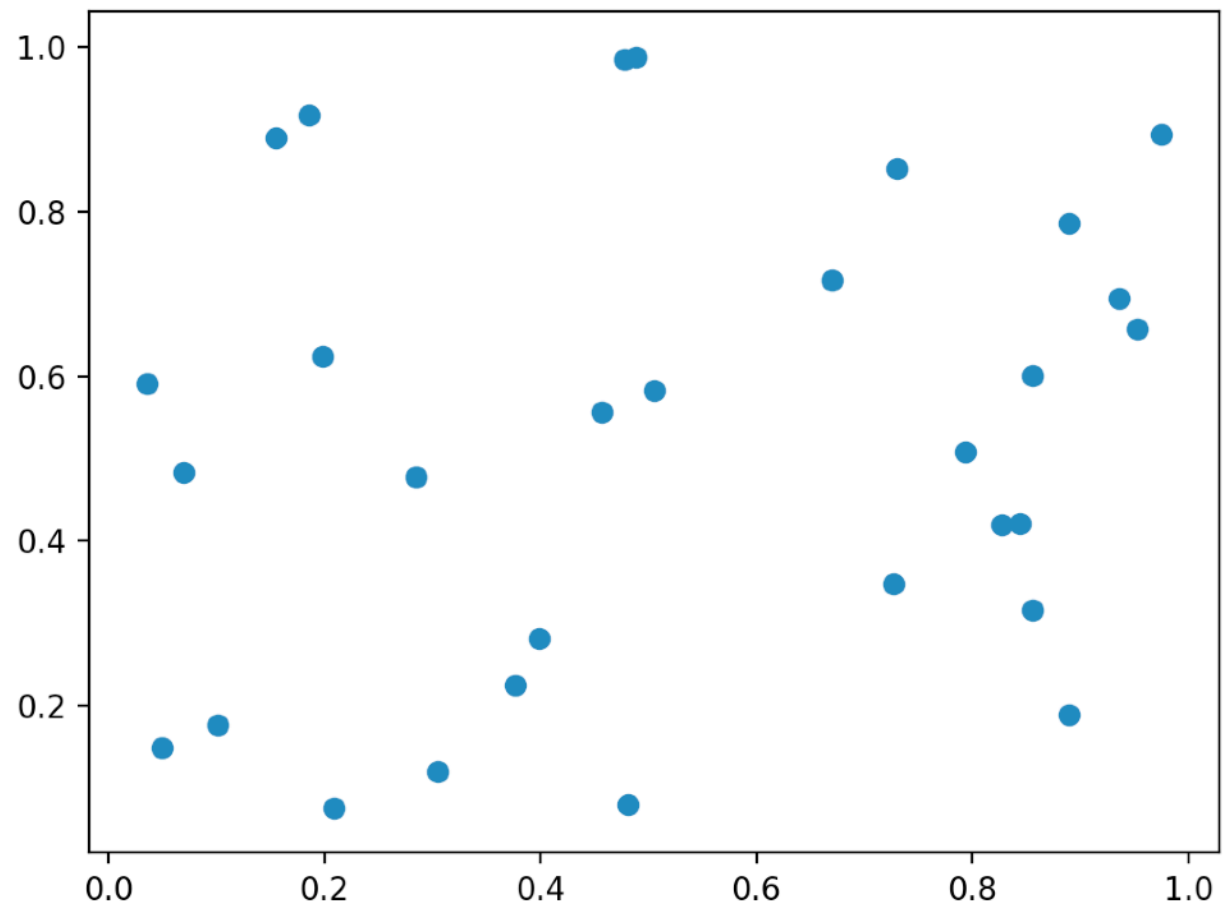
```
from pylab import *

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

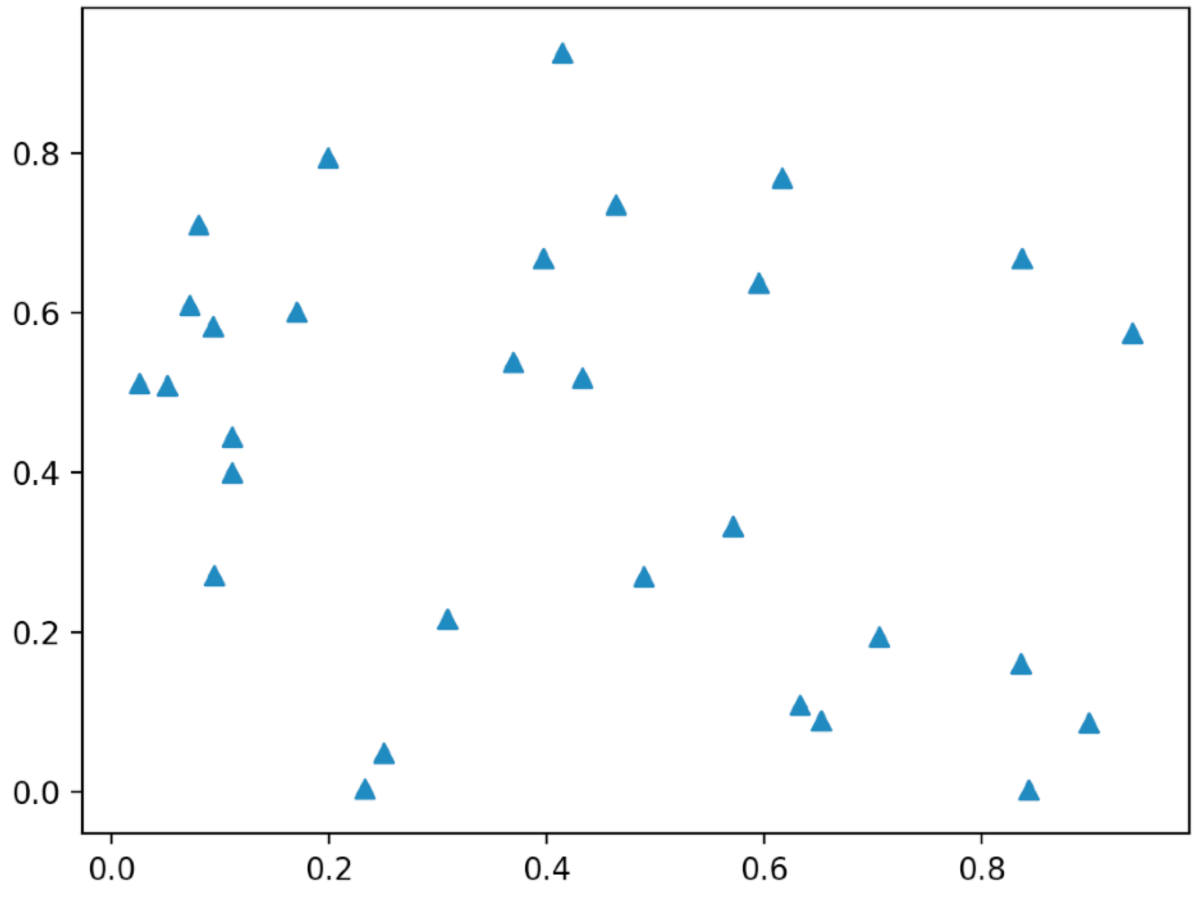
```
import matplotlib as np
from pylab import *

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

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



10. Matplotlib

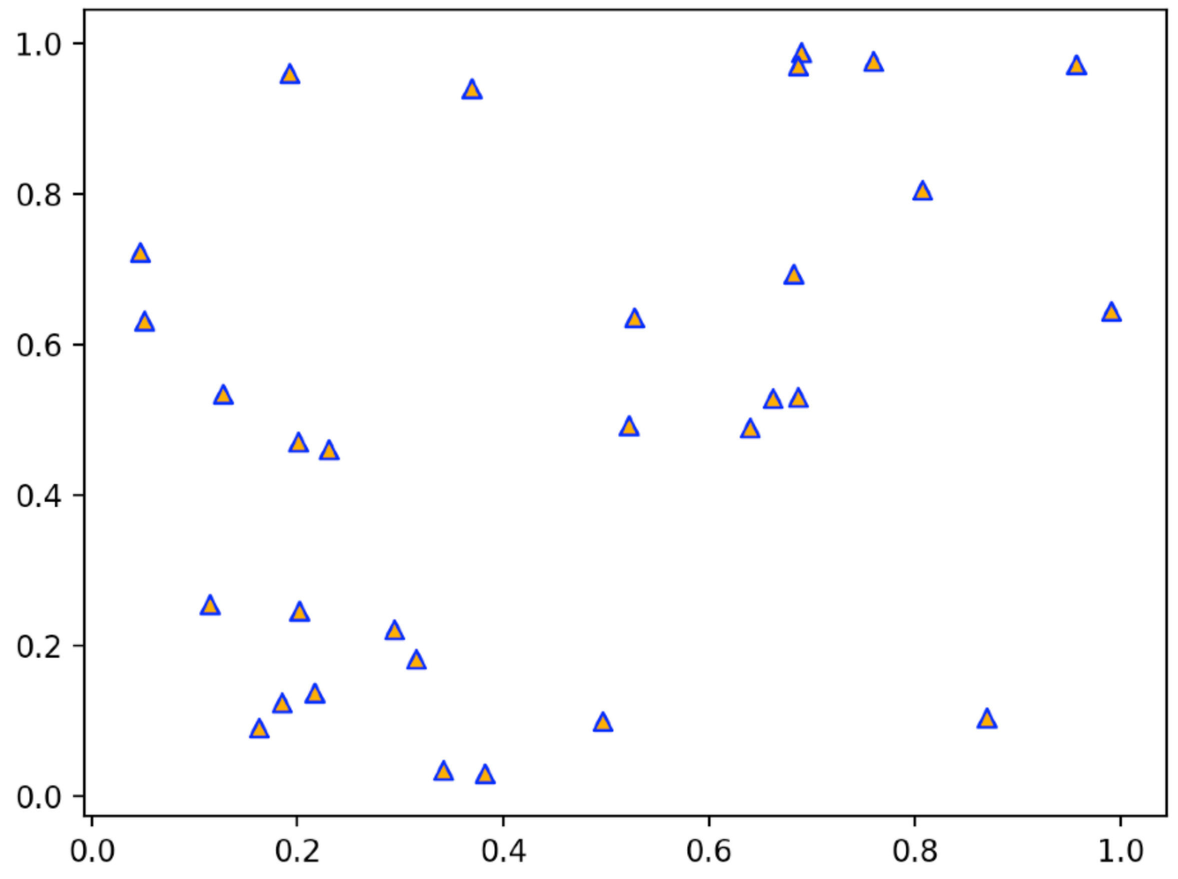
Scatter plot

```
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



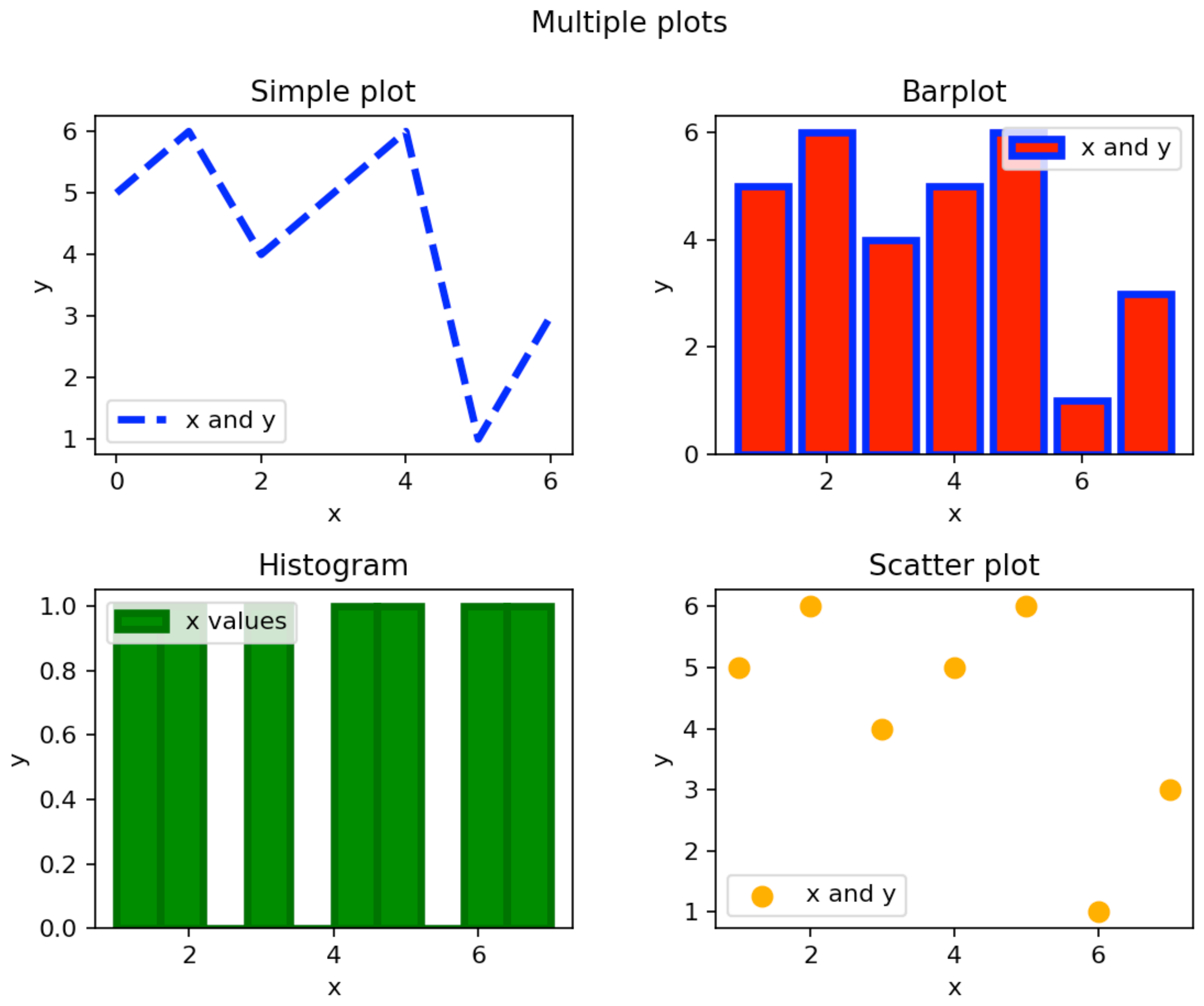
```
import matplotlib as np
from pylab import *

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]
```



10. Matplotlib

Scatter plot

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

figure(figsize=(8,6), dpi=80)
suptitle("Multiple plots")
subplot(2, 2, 1)
plot(y, color="blue", linewidth=3, linestyle="--", label="x and y")
legend()
title("Simple plot")
xlabel("x")
ylabel("y")
subplot(2, 2, 2)
bar(x, y, color="red", edgecolor='blue', linewidth=3, label="x and y")
legend()
title("Barplot")
xlabel("x")
ylabel("y")
subplot(2, 2, 3)
hist(x, color="green", linewidth=3, edgecolor="darkgreen", label="x values")
legend()
title("Histogram")
xlabel("x")
ylabel("y")
subplot(2, 2, 4)
scatter(x, y, color="orange", linewidth=3, linestyle="-", label="x and y")
legend()
title("Scatter plot")
xlabel("x")
ylabel("y")
subplots_adjust(left=0.12, bottom=0.11, right=0.90, top=0.88, wspace=0.30, hspace=0.40)
show()
```

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: Find max/min/sum/mean

```
import numpy as np

def findMax(listName):
    tempMax = 0
    for i in range(len(listName)):
        if (listName[i] > tempMax):
            tempMax = listName[i]
    return tempMax

def findMin(listName):
    tempMin = listName[0]
    for i in range(len(listName)):
        if (listName[i] < tempMin):
            tempMin = listName[i]
    return tempMin

def sumAllValues(listName):
    total = 0
    for i in range(len(listName)):
        total += listName[i]
    return total
…
```

```
…
def meanAllValues(listName):
    meanVal = np.mean(listName)
    print("The mean of your values is",
       meanVal)

# Main
l = []
for i in range(5):
    print("Please give a number")
    nb = float(input())
    l.append(nb)

maxValue = findMax(l)
print("The maximum value is", maxValue)
minValue = findMin(l)
print("The minimum value is", minValue)
sumValue = sumAllValues(l)
print("The sum of all your values is",
  sumValue)
meanAllValues(l)
```

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.

```
    lab[name] = [hour, minutes, duration]
```

Applications

Exercise: Lab schedule manager (version 3)

```
lab = {}
nbResearchers = 2

for i in range(nbResearchers):
    print("Please write the name of researcher", i)
    name = input()
    print("Please give the hour of the experiment beginning for", name)
    hour = float(input())
    while not (0 <= hour <= 23):
        print("Please give a correct time hour (0 to 23)")
        hour = float(input())
    print("Please give the minutes of the experiment beginning for", name)
    minutes = float(input())
    while not (0 <= minutes <= 59):
        print("Please give a correct minutes value (0 to 59)")
        minutes = float(input())
    print("Please give the duration of the experiment (in minutes) for", name)
    duration = float(input())

    lab[name] = [hour, minutes, duration]

print(lab)

for researcher in lab.keys():
    print("%s arrives at %d:%d and stays %d minutes in the lab" %(researcher,
lab[researcher][0], lab[researcher][1], lab[researcher][2]))
```

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)

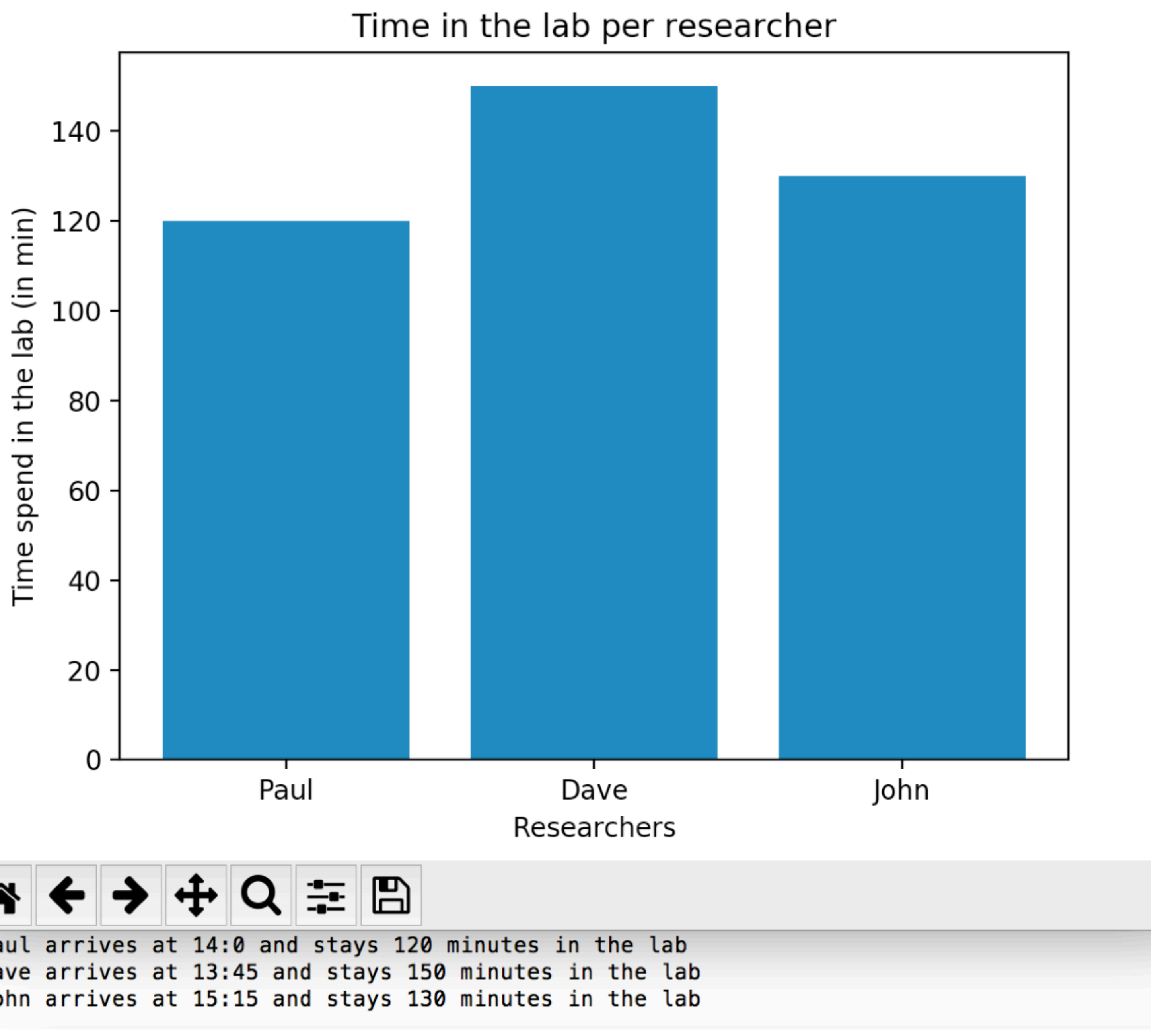
You will pay attention that the numbers entered for hours are from 0 to 23 and from 0 to 59 for the minutes.

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
```

Applications

Exercise: Lab schedule manager (version 4)

```
for researcher in lab.keys():
   print("%s arrives at %d:%d and stays %d minutes in the
lab" %(researcher, lab[researcher][0], lab[researcher][1],
lab[researcher][2]))

timeInTheLab=[]
for researcher in lab.keys():
   timeInTheLab.append(lab[researcher][2])

bar(range(len(lab)), timeInTheLab)
xticks(range(len(lab)), list(lab.keys()))
xlabel("Researchers")
ylabel("Time spend in the lab (in min)")
title("Time in the lab per researcher")

show()
```

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()` (change directory) :

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

If you want to check, use `getcwd()` (get current working directory) :

```
>>> 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:

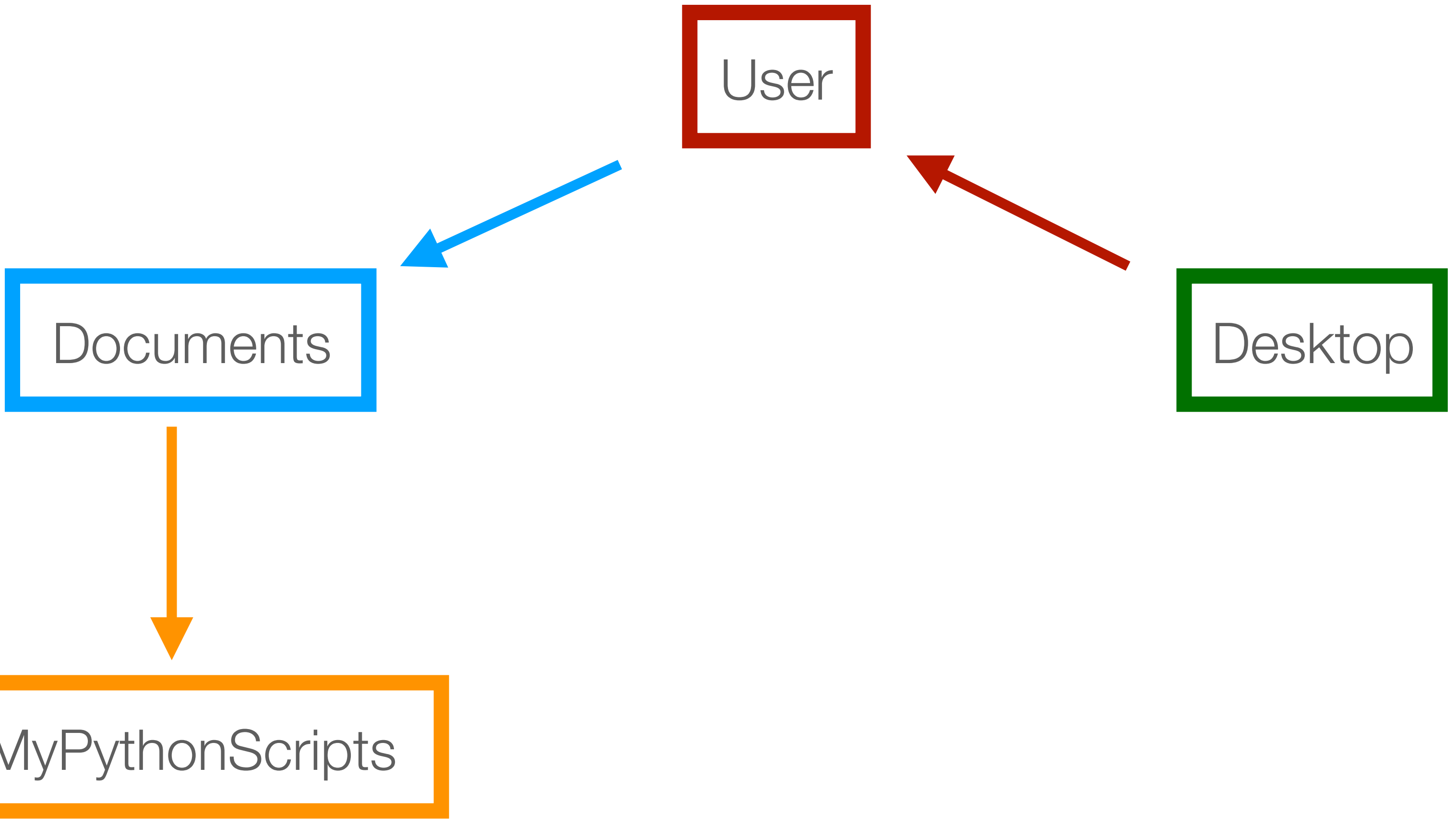
- **Linux/Mac** : you can use the Unix command line with the Terminal to set your working directory. The command `pwd` (`print working directory`) is to check your current directory and `cd` (`change directory`) to set it :

```
>>> 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"])

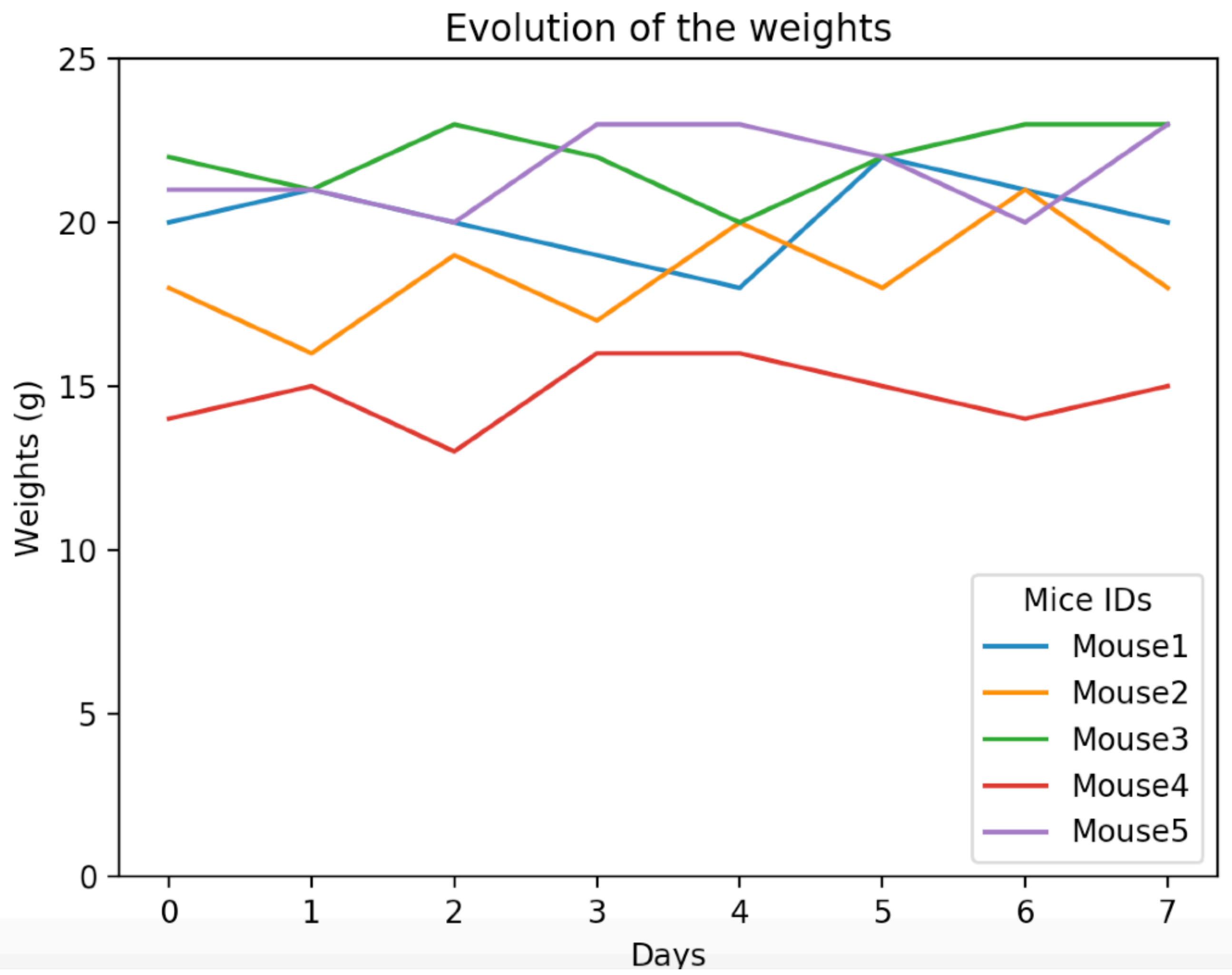
```
0    20
1    21
2    20
3    19
4    18
5    22
6    21
7    20
Name: Mouse1, dtype: int64
```

→ 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.



11. Write / Read data files

Working with Excel files

```
import pandas as pd
import pylab as plt

# Excel file
myFile = 'mice.xlsx'

# Excel spreadsheet
sp = pd.ExcelFile(myFile)

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

for mouse in myData.keys():
     # print(mouse, ":", myData[mouse])
     print(mouse)
     if (mouse != "Weights"):
          plt.plot(range(len(myData[mouse])), myData[mouse])

plt.legend(title="Mice IDs", loc="lower right")
plt.title("Evolution of the weights")
plt.xlabel("Days")
plt.ylabel("Weights (g)")
plt.ylim(0, 25)
plt.show()
```