

# Python

*(a crash course)*

Pedro Barahona

2019 / 20

# Graphics with Python



- Several types of graphics (line graphs, pie graphs, histograms, ...) and images can be drawn with Python, namely through the library **matplotlib**.
- Here we will only address line graphs, drawn with the following steps
  1. Clear all previous graph draws
  2. Fill a vector `x` with the x-coordinate values.
  3. Fill one or more vectors with the y-coordinate values.
  4. Use function `plot(x, y, fmt)` to draw each of the lines of the graph.
  5. Define the title of the graph, axes and legend of the graph (all optional)
  6. Show and save the graph in a file (optional)
- The following example illustrates these steps

# Graphics with Python



```
import math as m
import matplotlib.pyplot as plt

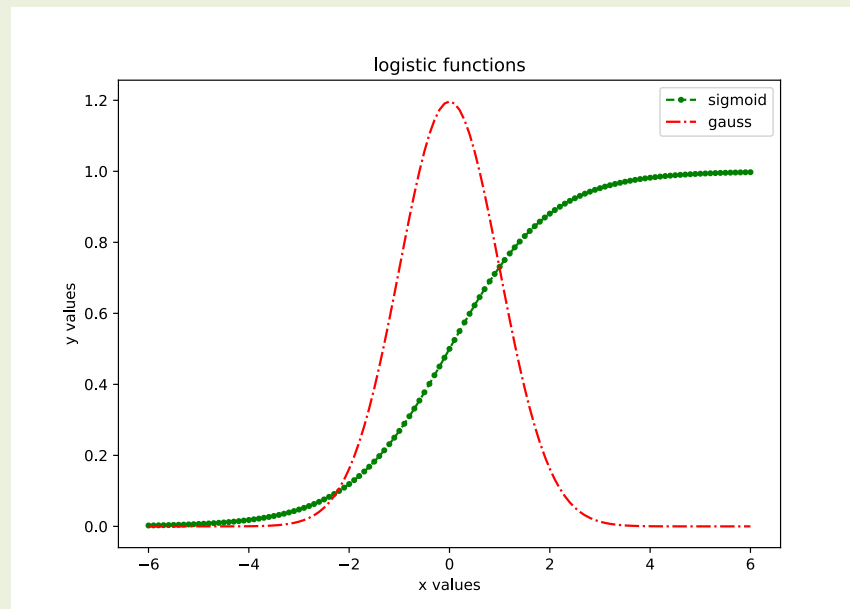
P = list(range(101))          # a list of 101 points
X = [a/100.0 for a in P]      # x- coordinates
S = [m.sin(x*2*m.pi) for x in X] # sine values
C = [m.cos(x*2*m.pi) for x in X] # cosine values
plt.plot(X,S,'g-.')          # sine line format
plt.plot(X,C,'r*')           # cosine line format
plt.title('trigonometric functions') # title
plt.legend(['sin', 'cos'])    # legend
plt.xlabel('x values')        # x-axis label
plt.ylabel('y values')        # y-axis label
plt.show()                   # draw the graph
plt.savefig('trigo.png')      # save the graph
```

- There are many possibilities available to format the graphs. For the style of the lines a number of options can be used in the 3<sup>rd</sup> parameter of the plot function:
  - Colours: 'b': blue, 'g': green, 'r': red, 'y': yellow, 'k': black
  - Markers: ':': point, 'o': circle, '+': plus, 'x'-times, '\*': star
  - Styles: '-' : solid, '--': dotted, ':' : dashed, '-.' : dash-dotsee `help(plt.plot)` for more information on formats
- The graphs can be stored in a file (usually with a png or pdf extension) for further use.

# Python – Exercise on Graphics



- **Exercise 5:** Write a function to draw the graphs of a sigmoid and a gauss curve, in the interval  $x \in [-4 .. 4]$ .
- Use the signature **def sig\_gauss\_graphics():**
- Note : use the definition of the functions used earlier to get a graph similar to



# Dictionaries in Python



- Quite often, data must be grouped together so that access is made not through an index (like in arrays) but through a **key**, i.e. a string.

**Example:** Each element of the periodic table can be associated with:

- A string, **symbol**, with its chemical symbol;
- An integer, **atomic number**, with its atomic number;
- In Python, data can be grouped in data type **dictionary** (type dict), which is a set of **key:value** pairs.
  - The **key** may be a string, int, float or boolean.
  - The **value** can be of any type.

# Dictionaries - Creation



- In Python a dictionary is na object of type dict, created with the following

## Syntax:

```
variable = { keyName1 : valOfKey1,  
             ....  
             keyName1 : valOfKeyN}
```

- **Example:**

```
In : actinium = {  
...: 'symbQ' : 'Ac',  
...: 'nAtom' : 89}  
In : actinium  
Out: {'symbQ': 'Ac', 'nAtom': 89}
```

# Dictionaries – Key Values

- The value of the keys in a dictionary can be accessed (and changed), using **.get method**):

`variable.get(KeyName)`

Or by an equivalent syntax

`variable[KeyName]`

```
In : lithium = {  
...: 'symbQ' : 'Li',  
...: 'nAtom' : None}  
In : lithium['nAtom']  
In : lithium['nAtom'] = 3  
In : lithium['nAtom']  
Out: 3  
In : lithium.get('symbQ')  
Out: 'Li'  
In : lithium  
Out: {'symbQ': 'Li', 'nAtom': 3}
```



# Dictionaries – Keys Addition



- Once a dictionary is created, additional key:value pairs can be included through method **.update**

`variable.update(KeyName)`

## Example:

- Add the atomic mass to an atom description.

```
In : lithium
Out: {'symbQ': 'Li', 'nAtom': 3}
In : lithium.update({'mAtom': 6.941})
In : lithium
Out: {'symbQ': 'Li', 'nAtom': 3, 'mAtom': 6.941}
```

# Dictionaries – Keys Elimination



- Similarly, key:value pairs can be eliminated from a dictionary through method **.pop**

`variable.pop(KeyName)`

## Example:

- Eliminate the atomic mass from na atom description.

```
In : lithium
Out: {'symbQ': 'Li', 'nAtom': 3, 'mAtom': 6.941}
In : lithium.pop('mAtom')
Out: 6.941
In : lithium
Out: {'symbQ': 'Li', 'nAtom': 3}
```

# Dictionaries – What Keys?



- To know the keys belonging to a dictionary, the method **.keys** can be used

```
variable.keys()
```

- In particular, it may also be used to check whether some key does exists  

```
key in variable.keys()
```

## Example:

- Attributes associated to a chemical element

```
In : lithium = {  
...: 'symbQ' : 'Li',  
...: 'nAtom' : None}  
In : lithium.keys()  
In : dict_keys(['symbQ', 'nAtom'])  
In : 'nAtom' in lithium.keys()  
Out: True
```

# Dictionaries – Key Iteration



- The different attributes of a dictionary can be iterated (for example in a FOR loop) through their keys as in

```
for key in variable.keys()
```

Or, with the simplified syntax:

```
for key in variable
```

- **Exemplo:**

```
In : lithium
Out: {'symbQ': 'Li', 'nAtom': 3}
In : for key in lí lithium.keys():
...: print(key)
symbQ
nAtom
In : for key in lithium :
...: print(key)
symbQ
nAtom
```

# Strings



- Before addressing the reading and writing of text files it is convenient to overview the data type string (**str**).
- Strings are just sequences of characters (usually implemented through their representations in a code scheme such as ASCII or UTF).
- Strings are denoted with framing quotes or double quotes.
- They may include visible characters (letters, upper or lower, digits, punctuation symbols) or invisible symbols such as spaces, or tabs and end-of-line characters, denoted by escape sequences `\t` and `\n`, respectively).
- Like lists, their size may be obtained with method `len()` and individual characters accessed by their index in the string.

# String Splitting

- For example

```
In : s = "This is a string with 225 characters of which:\n\t11  
characters which are digits;\n\t1 characters that are upper case  
;\n\t165 characters that are lower case ;\n\t6 characters that  
are punctuation marks; and\n\t42 whitespace characters"
```

```
In : s
```

```
'This is a string with 225 characters of which:\n\t11 characters  
which are digits;\n\t1 characters that are upper case;\n\t165 characters that are lower case;\n\t6 characters that are  
punctuation marks; and\n\t42 whitespace characters'
```

```
In : print(s)
```

```
This is a string with 223 characters of which:  
    11 characters which are digits;  
    1 characters that are upper case;  
   165 characters that are lower case;  
    6 characters that are punctuation marks; and  
   40 whitespace characters
```

# String Splitting

- Method **split**, as in

`str.split(sep)`

**split**s string `str` by its **separator** and returns a list with each of the sub strings.

- A **separator** is also a string, that does not appear in sub-strings
- Usual separators are : `' , ' , ' ; ' , ' ' , '\t'`

```
In : s = '113 4.5 7.9 2019'
In : s.split(' ')
Out: ['113', '4.5', '7.9', '2019']
In : t = '113; 4.5; 7.9; 2019'
In : s.split('; ')
Out: ['113', '4.5', '7.9', '2019']
```

# String Stripping

- Method **strip**, as in

`str.strip()`

returns a string without leading and trailing white spaces (i.e. spaces, end of line, tabs)..

- Note that whitespaces inside the string are not removed.

```
In : s = '  113\t4.5\t7.9\t2019. \n '  
In : s.strip(' ')  
Out: '113\t4.5\t7.9\t2019.'
```



# Types of String

---

Other methods return the type of a string.

- **isalnum()**
  - True if the string is an alpha-numeric string, False otherwise.
  - A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.
- **isalpha()**
  - True if the string is an alphabetic string, False otherwise.
  - A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.
- **isnumeric()**
  - True if the string is a numeric string, False otherwise.
  - A string is numeric if all characters in the string are numeric and there is at least one character in the string.

# Types of String

---

Other methods return the type of a string.

- **isdecimal()**
  - True if the string is a decimal string, False otherwise.
  - A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.
- **isspace()**
  - True if the string is whitespace, False otherwise.
  - Spaces, tabs and newlines are considered spaces..
- **isprintable ()**
  - True if the string is printable, False otherwise.
  - Tabs and newlines are not printable.

More info with `help(str)`

- **Exercise 6b:**

Check whether string

```
s = "This is a string with 225 characters of which:\n\t11 characters  
which are digits;\n\t1 characters that are upper case;\n\t165  
characters that are lower case;\n\t6 characters that are punctuation  
marks; and\n\t42 whitespace characters"
```

has indeed the number of the different character types that it indicates (suggestion: Use a dictionary).

- **Exercise 6b:** Obtain the frequency of occurrences of the characters appearing in the string (again, use a dictionary).

# Text Files with Python



- 
- To read from or write to a text file, it is necessary to open it, in the corresponding mode.
  - If successful, this operation returns an object identifying the file that is to be read or written. Reading and writing, sequentially, are performed to calls of methods of these objects.
  - (We will assume that the file is in the current directory, otherwise it is necessary to set a path to it.)
  - Once the read / write operations are completed, the file should be closed, again through a call to a method of the file identifier object.

# Opening Text Files

- The instruction

```
fich = open( FileName, mode)
```

opens a file with name “Filename” and mode:

- **Reading:** if mode = 'r'
  - **Writing:** if mode = 'w'
- The function returns an object, **fich**, that should be used for all subsequent accesses to the file.
  - Of course, a file that does not exist cannot be read. Reading files that do not exist raise error

```
In : x = fopen('texto.pdf') # the file does not exist
FileNotFoundError: [Errno 2] No such file or directory:
'texto.pdf'
```

# Closing Text Files

- The method close, effectively closes the file

**fich.close()**

- Again, a file that is not opened, or has already been closed, can not be closed.

```
In : fich = open(teste.txt, 'r')
In : ... # reading the file
In : fich.close()
In : fichX.close()
NameError: name 'fichX' is not defined
```

# Writing in Text Files

- Function (method) **write** writes a string in a text file

**file.write( string )**

- It returns the number of characters written, in the file.
- Some special format characters:
  - \n** new line
  - \t** horizontal tab

example.txt

This is the first line  
and this is the second  
end

```
In : x = open('example.txt', 'w')
In : x.write('This is the first line\n and this is the second\n')
Out[12]: 46
In : x.write('fim\n')
Out[12]: 4
In : x.close()
```

# Reading from Text Files [1]



- Function (method) **read**

**read()**

reads all the text file (from the current position until the end) and returns a string with all the characters that were read (including the new lines).

- Reading beyond the end of returns an empty string.

example.txt

This is the first line  
and this is the second  
end

```
In : a = open('example.txt')
In : a.read()
Out: 'This is the first line\nand this is the second\nend\n'
In : a.read()
Out: ''
In : a.close()
```



# Reading from Text Files [2]



- Function (method) **readlines**

**readlines()**

reads all the text file (from the current position until the end) and returns a list of strings with all characters read.

- Reading beyond the end of file returns an empty string
- The new lines are returned in the strings.

example.txt

This is the first line  
and this is the second  
end

```
In : a = open('example.txt')
In : a.readlines()
Out: ['This is the first line\n', 'and this is the second\n', 'end\n']
In : a.readlines()
Out: []
In : a.close()
```

# Reading from Text Files [3]



- Function (method) **readline**

`readline()`

reads one line of the text file (from the current position until the new line) and returns a strings with all characters read.

- Reading beyond the end of file returns an empty string
- The new lines are returned in the strings.

example.txt

This is the first line  
and this is the second  
end

```
In : a = open('example.txt')
In : a.readline()
Out: 'This is the first line\n'
In : a.readline()
Out: 'and this is the second\n'
In : a.readline()
Out: 'end\n'
In : a.close()
```

# Reading from Text Files [4]



- The different methods can be used together.
  - Notice the current position!

example.txt

This is the first line  
and this is the second  
end

```
In : a = open('example.txt')
In : a.readline()
Out: 'This is the first line\n'
In : a.readlines()
Out: ['and this is the second\n', 'end\n']
In : a.readlines()
Out: []
In : a.read()
Out: ''
In : a.close()
```

# Reading from Text Files [5]



- Notice that Reading text files return strings, even when they “represent” numbers. To obtain the numbers, strings must be converted into numbers.

numbers.txt

25 34.7

...

```
In : a = open('numbers.txt')
In : line = a.readline()
In : line
Out: ' 25\t34.7\n'
In : line.strip().split('\t')
In : line
Out: ['25', '34.7']
In : line[0] = int(line[0])
In : line[1] = float(line[1])
In : line
Out: [25, 35]
```

# Python – Exercise on Text Files



- **Exercise 7a:**

Read a text file (dimacs.txt) representing a labelled undirected graph encoded with the DIMACS format, where

- The first line has two integers (separated by spaces) representing the number of nodes and number of arcs, respectively.
- the subsequent lines, one for each arc, have 3 numbers (separated by spaces):
  - the first two, are integers and denote the the nodes of the arc;
  - the third number, (an integer or a float) represent the label of the arc (e.g. a length, or a
- Obtain the adjacency matrix of the graph (a matrix  $M$ , where  $M[i][j]$  denotes the label of the arc  $(i,j)$ ).

dimacs.txt

```
5 8
1 2 5
1 3 43
1 4 95
1 5 22
2 3 68
2 5 35
3 4 39
3 5 90
```

# Python – Exercise on Text Files



- **Exercise 7b:**

Write the adjacency matrix of the graph just read into a text file adjacency.txt.

dimacs.txt

```
5 8
1 2 5
1 3 43
1 4 95
1 5 22
2 3 68
2 5 35
3 4 39
3 5 90
```

- The first line has one integer representing the number of nodes of the graph.
- the subsequent n lines, one for each node, have n numbers (separated by tabs) denoting the labels of the arcs starting in each of the nodes.
- Label 0 denotes that there is no arc.

adjacency.txt

```
5
0 5 43 95 22
5 0 68 0 35
43 68 0 39 90
95 0 39 0 0
22 35 90 0 0
```

# Random Numbers in Python



- Many programs, namely Monte Carlo simulation and Local Search Optimization require the use of (pseudo-)random numbers .
- In Python, library random includes a (pseudo-)random generator in the range ]0..1[.

```
In : import random as r
In : r.random()
Out: 0.8029391687796316
In : r.random()
Out: 0.5334949790540763
In : r.random()
Out: 0.206611672213879
```

- Other distributions, namely discrete uniform distributions in the range 0..k can be obtained from the basic one.

# Random Numbers in Python



- All that is required is to do the proper transformation, scaling and then truncating the random number obtained.
- For example, a distribution in the range  $0..k$ , may be obtained with the transformation `trunc(u*(k+1))`

```
In : import random as r
In : import math as m
In : k = 4
In : u = r.random()
In : u
Out: 0.8029391687796316
In : m.trunc(u*(k+1))
Out: 4
In : m.trunc(r.random()*(k+1))
Out: 3
In : m.trunc(r.random()*(k+1))
Out: 3
In : m.trunc(r.random()*(k+1))
Out: 0
```



## Exercise 8:

- Given an adjacency matrix  $M$  of a graph, obtain a random tour starting and ending at node 1, and visiting each of the other nodes, exactly once.
- The signature of the function should be  

```
def tour(M)
```
- The function should return a pair, composed of a tour (a list of the nodes in the order in which they are visited) and its length (the sum of the distances of the nodes in the tour).
- The order in which the nodes are visited is random, so different calls to the function return, in general, different results.