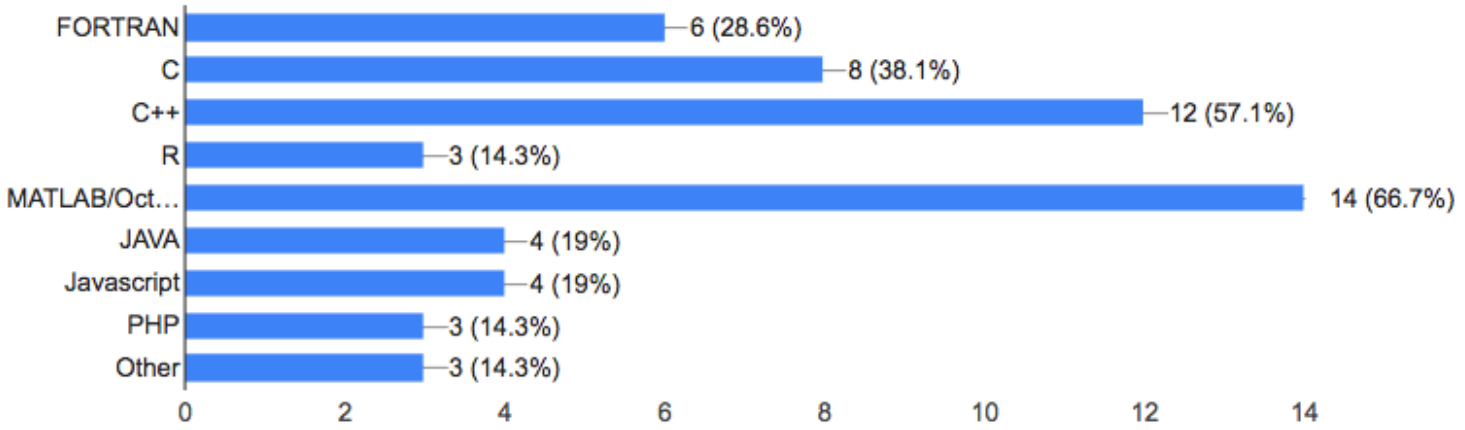
python™

Programming with Python for Data Science

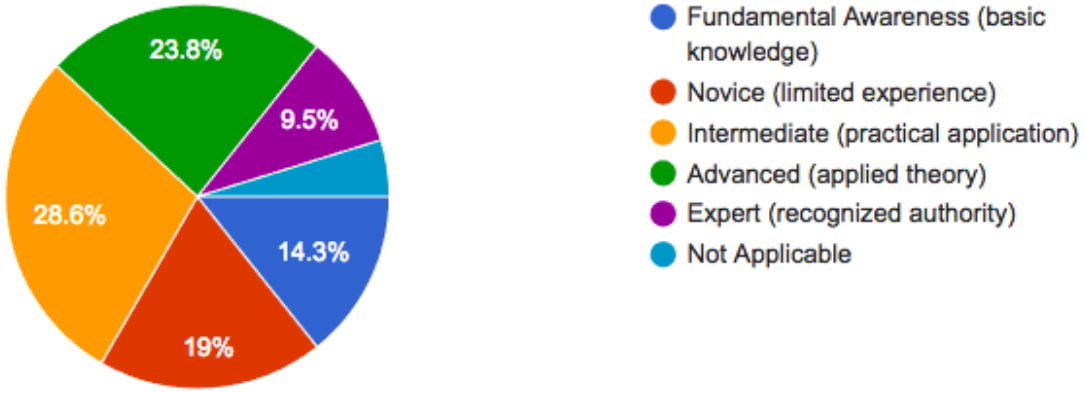
Online resources

- Learn Python the Hard Way
  https://learnpythonthehardway.org/
- Dive into Python
  http://www.diveintopython3.net/
- Think Python
  http://www.greenteapress.com/thinkpython/thinkpython.html
- Python Programming
  https://en.wikibooks.org/wiki/Python_Programming
- A Byte of Python
  https://python.swaroopch.com/
- https://wiki.python.org/moin/BeginnersGuide/Programmers

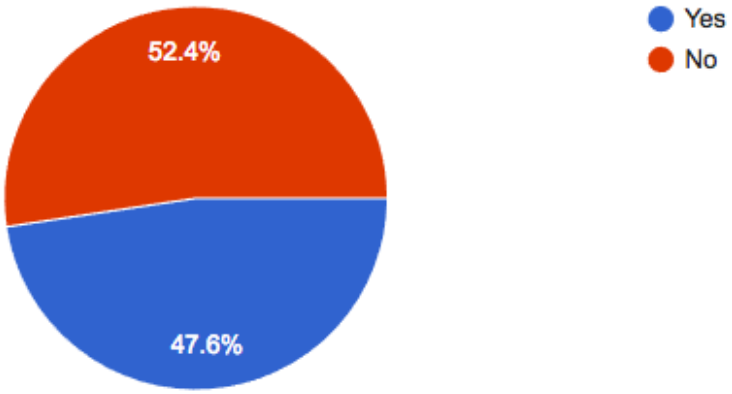
Do you know another programming language? (21 responses)



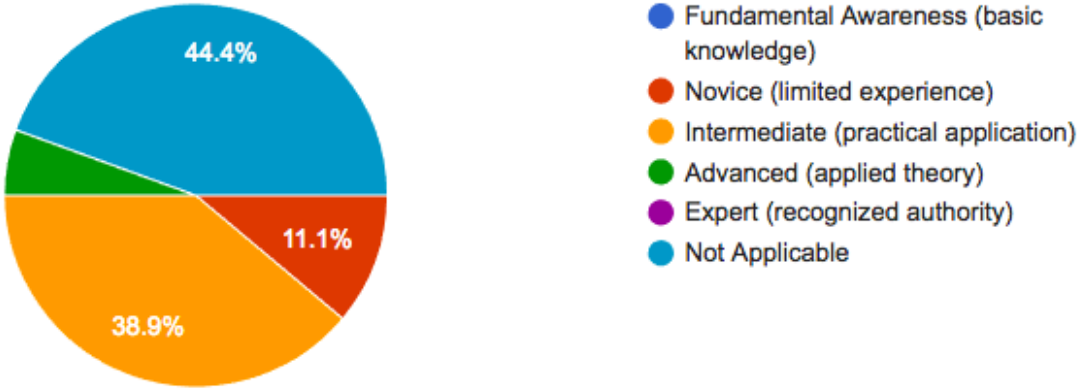
On average at what level? (21 responses)



Do you know the Python language? (21 responses)



At what level? (18 responses)



Unit Topics

- Python: a multipurpose language, open source and free.
- Each Python variable is a object

Learning objectives

- Install Python
- Write your first Python instructions
- Know the basic information about Python
- Describe the Python typing model

Spyder

- Spyder is an interactive development environment for the Python language with advanced editing, live testing, and a numerical computing environment thanks to support of IPython (enhanced interactive Python interpreter). Spyder also includes the popular Python library NumPy for linear algebra, MatPlotLib for interactive 2D/3D graphs, Pandas for dataset manipulation, and SciKit-Learn for machine learning.

- Go to:

  https://docs.continuum.io/anaconda/install

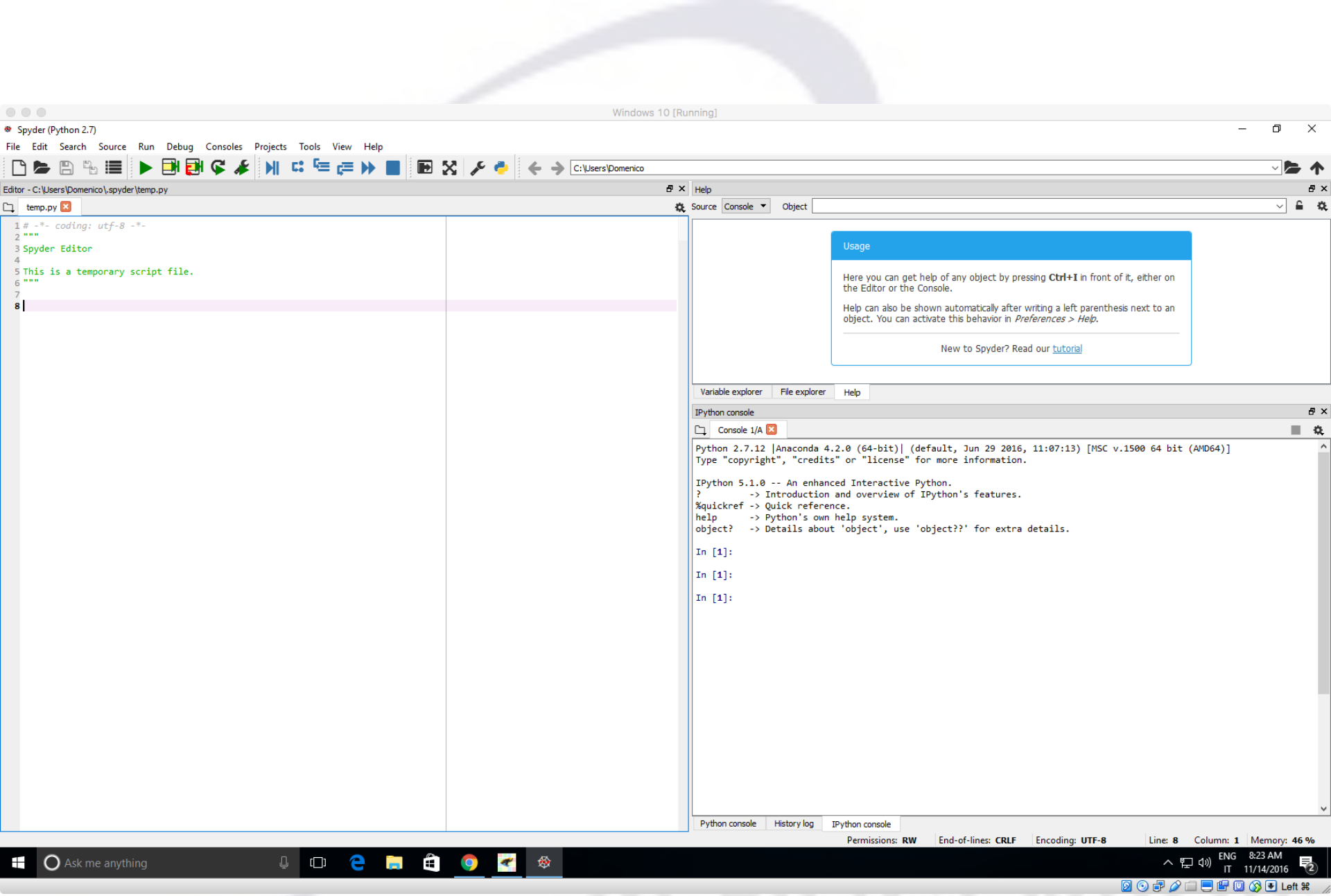
  and execute the installer of your preference for your operating system. This will install Anaconda, Spyder, NumPy, SciPy, SciKit-Learn, Pandas, and MatPlotLib for you automatically.

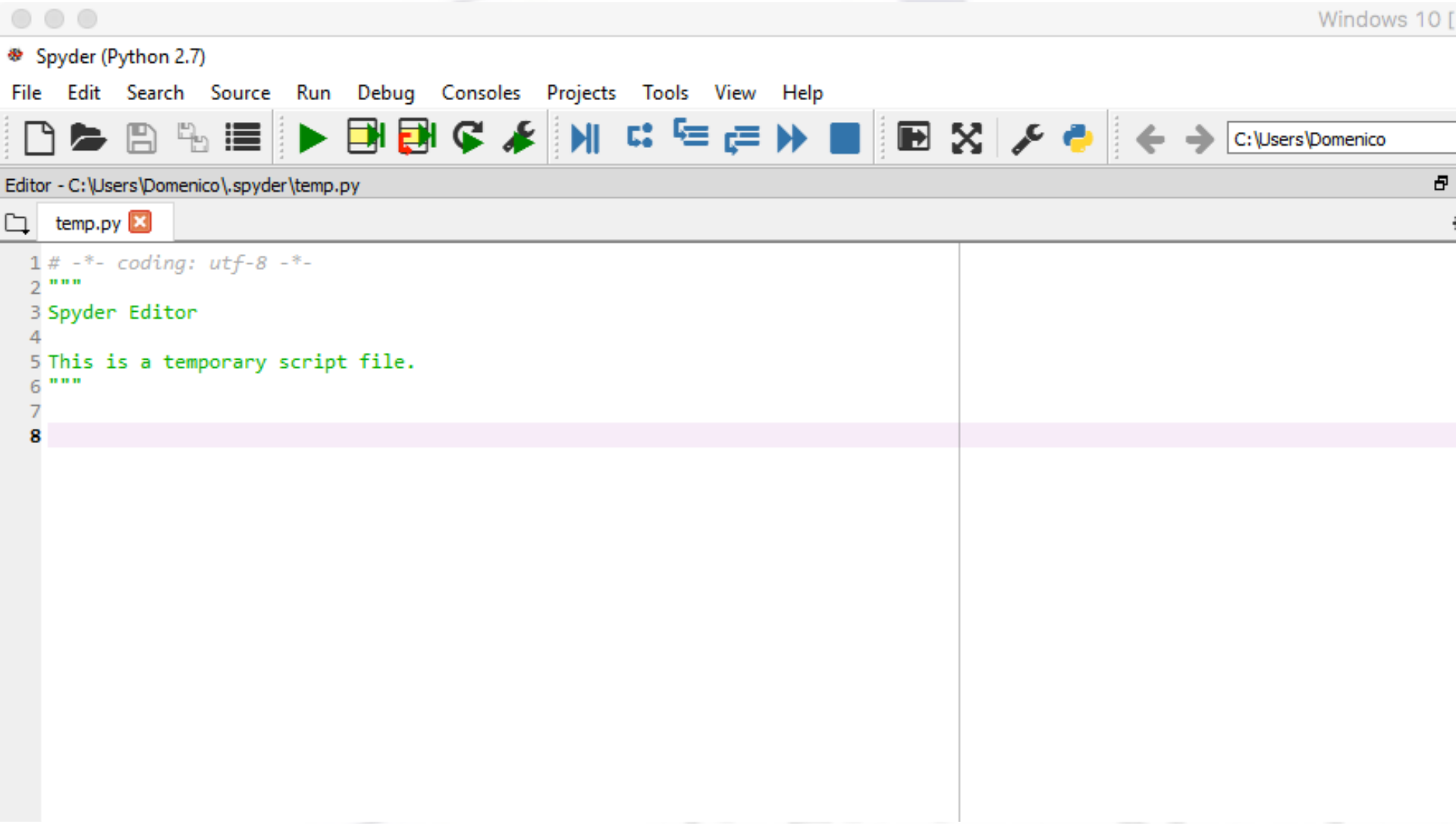
- https://conda.io/docs/py2or3.html

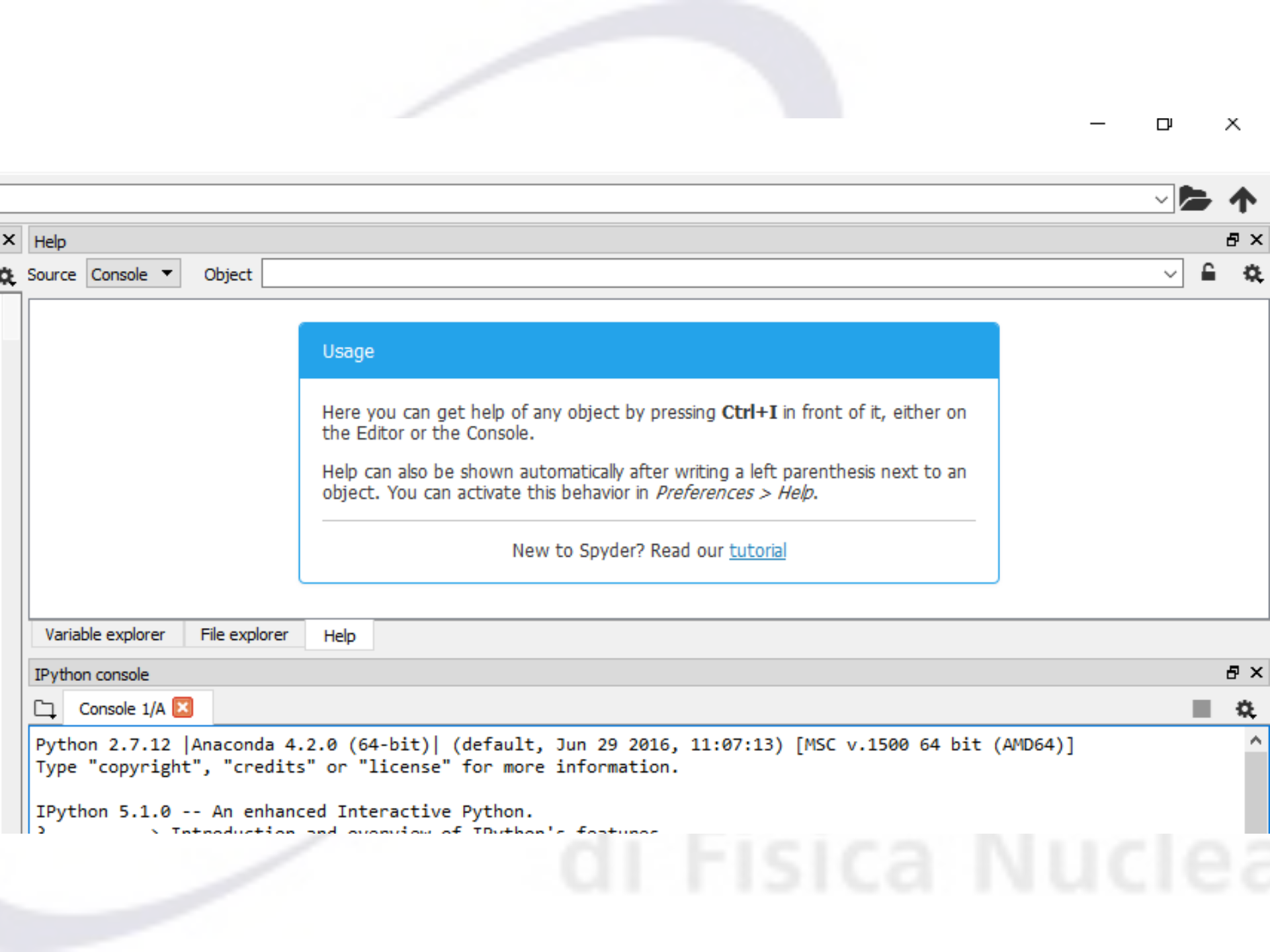
  conda create -n Python27 python=2.7
  activate Python27

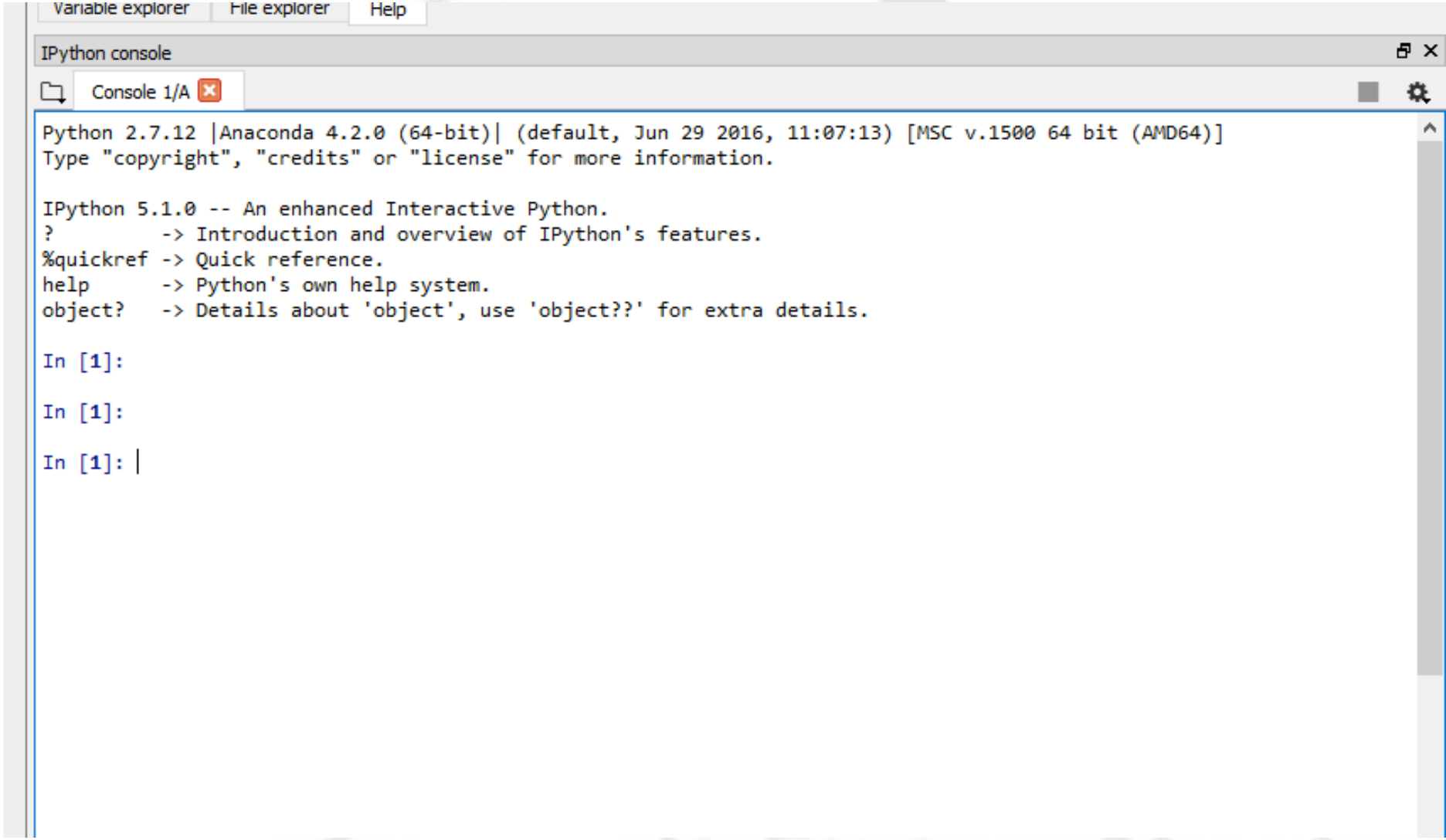
Using Spyder

- You can use Spyder as your IDE for this course. Like other IDEs, Spyder will give you realtime syntax validation, and intelligent code completion for members, parameters, variables, etc. In essence, it'll speed up your Python development. Spyder also provides you with an interactive console you may use to:
  - Step through your scripts, line by line
  - Interact and alter your script's variables in realtime
  - Code directly in the interactive console
  - Render graphs and charts to the user interface
- One important hint that will make your life easier is to *enable interactive 3d plots* in Spyder. To turn them on, go to Tools > Preferences > IPython Console > Graphics. From this page, set **Backend** to **Automatic**.


Windows 10 [Running]
Spyder (Python 2.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Domenico
Editor - C:\Users\Domenico\.spyder\temp.py
temp.py
-*- coding: utf-8 -*-
"""
Spyder Editor
This is a temporary script file.
"""
Help
Source Console Object
Usage
Here you can get help of any object by pressing Ctrl+I in front of it, either on the Editor or the Console.
Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in Preferences > Help.
New to Spyder? Read our tutorial
Variable explorer File explorer Help
IPython console
Console 1/A
Python 2.7.12 |Anaconda 4.2.0 (64-bit)| (default, Jun 29 2016, 11:07:13) [MSC v.1500 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.
IPython 5.1.0 -- An enhanced Interactive Python.
? -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help -> Python's own help system.
object? -> Details about 'object', use 'object??' for extra details.
In [1]:
In [1]:
In [1]:
Python console History log IPython console
Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 8 Column: 1 Memory: 46 %
Ask me anything
ENG IT 8:23 AM 11/14/2016
Left ⌘


Windows 10 [
Spyder (Python 2.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Domenico
Editor - C:\Users\Domenico\.spyder\temp.py
temp.py
1 # -*- coding: utf-8 -*-
2 """
3 Spyder Editor
4
5 This is a temporary script file.
6 """
7
8


Help
Source
Console
Object
Usage
Here you can get help of any object by pressing Ctrl+I in front of it, either on the Editor or the Console.
Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in Preferences > Help.
New to Spyder? Read our tutorial
Variable explorer
File explorer
Help
IPython console
Console 1/A
Python 2.7.12 |Anaconda 4.2.0 (64-bit)| (default, Jun 29 2016, 11:07:13) [MSC v.1500 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.
IPython 5.1.0 -- An enhanced Interactive Python.


Variable explorer
File explorer
Help
IPython console
Console 1/A
Python 2.7.12 |Anaconda 4.2.0 (64-bit)| (default, Jun 29 2016, 11:07:13) [MSC v.1500 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.
IPython 5.1.0 -- An enhanced Interactive Python.
? -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help -> Python's own help system.
object? -> Details about 'object', use 'object??' for extra details.
In [1]:
In [1]:
In [1]:

Docker alternative

- If you do not want to install anaconda on your personal computer, you can use docker to download an image which contains anaconda and all the files needed for the course.

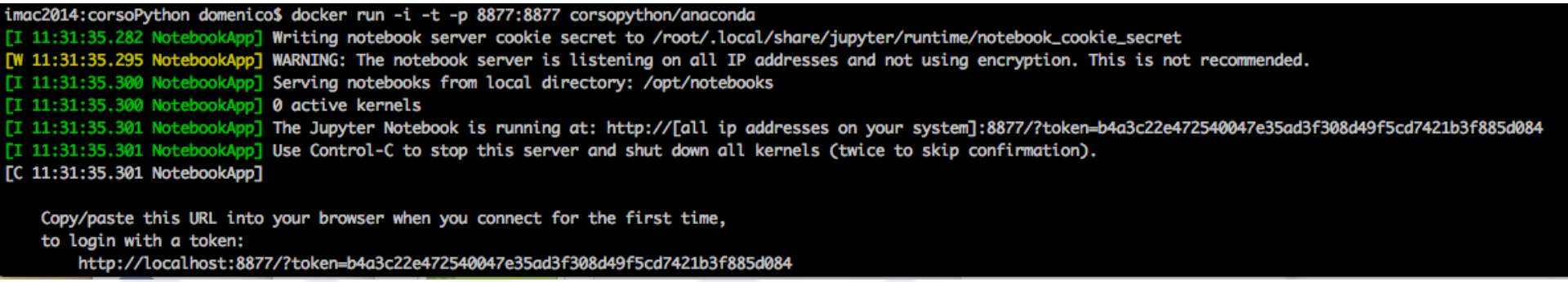
- Docker is a new technology, aimed to obtain a lightweight virtual environments, that you can run instantaneously on every host that can run docker.

- There are docker versions for Windows, Mac and Linux:

  https://www.docker.com/products/overview

When docker is up and running on your host, all that you have to do is open a Command Line Interface (cmd on Windows, a terminal otherwise) and run the following command:

```
docker run -i -t -p 8877:8877 corsopython/anaconda
```



Note that the token will be different.
Now you can open a browser and point it to the address shown in the last line of output. You will then have access to the Jupyter notebook.

Jupyter notebook

- A Jupiter notebook is an environment that will enable you to use Python 3 in a browser.
- Each code cell is a independent unit of elaboration: you can evaluate the instructions pressing **Shift+Enter**
- With **Alt+Enter** a new code cell will be added below
- You can save the notebook (in a .ipynb file) for later reference, o start a new one if you need it. If you are using the docker container you must **Download** the notebook from the File menu.
- The kernel menu is useful for restarting the kernel if there are problems.
- If you enter **Shift+Tab** after a function name the notebook will show you the available parameters.
- The code cell can be converted in a markdown cell to take notes on the code.

An interpreted code

- An **interpreter** it's a software application that runs other programs. When you write a command in the Python shell, or when you run a Python script, the Python interpreter reads your code and executes the instructions: you can imagine it as a logical layer between the code and the hardware.

- Python writes a intermediate **byte-code** ,a "native" Python command list, that will be executed in the **Python Virtual Machine**.

- Python does NOT have a «*make*» phase that translates the text code in binary machine code.

  PROS:
  ---The code is "highly" portable, and it's quite easy to write code that does not depend from the machine operating system: you can write the code on your Linux server, and than copy it on the Windows desktop. (or viceversa).
  ---It is possible to run code from an interactive shell.

  CONS:
  ---The execution generally cannot have the same speed that can be reached with a compiled code

Getting started

```
x = 2.0
y = 1
z=x+y
print(z)
```

```
x = (24 * 2) / 3
y = 2.5
print(x + (y * 2))
```

```
x = 'John'
print("hello",x)
```

```
x = "dog"
y = "cat"
x = 2
print(x, y)
```

Your first program

- Write in a Python script (left side of spyder) or in a Jupyter notebook the algorithm to convert a temperature from Celsius to Fahrenheit:
  1. Set the input temperature
  2. Multiply it by 9
  3. Divide the result of previous step by 5
  4. Add 32 to the result of step three

What is a VARIABLE?
A variable is a reserved location in a computer memory to store values.
We can store different data types, such as integers, floating point numbers, and strings in a variable. Python automatically reserves the memory space for each variable.

What is a IDENTIFIER?
An identifier is the name given to a variable, or other entities such as functions, objects, and so on.

Variables and identifiers are not the same.
The name of the variable is an identifier, but a variable has other properties such as value, type, scope.
Identifiers rules:

1. ID are case sensitive
2. ID can contain only letters, numbers and underscores (no ?,*,+)
3. ID cannot start with a number
4. ID cannot be a Python keyword

Exercise

Write a script that given the height (20) and width(30) of a rectangle, calculates and prints area and perimeter

Built-in types

- There are four built-in **numeric types**:

  1. integers `int: 0, 1, -3 (C long int)`

     They have at least 32 bit precision

  2. Long integers `long: 0L, 1L, -3L.`

     May have arbitrary length, their limit depends on the amount of memory in the machine.

  3. Double precision real `float: 0., .1, -0.0165, 1.89E+14 (C double)`

  4. Double precision complex `complex: 0j, 1+.5j, -3.14-2j`

- Information on the accuracy and internal representation of floating point for the machine on which you're running a script are available in `sys.float_info`

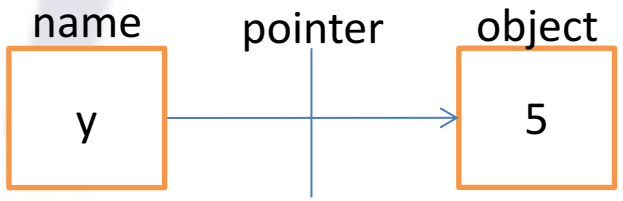
- Python has a Boolean type **bool**, that can take the values `True` or `False`, respectively interchangeable with 1 and 0.

- And of course there is a **string** type `str`

- With the built-in function `type()` you can query the type of a variable

Variable creation

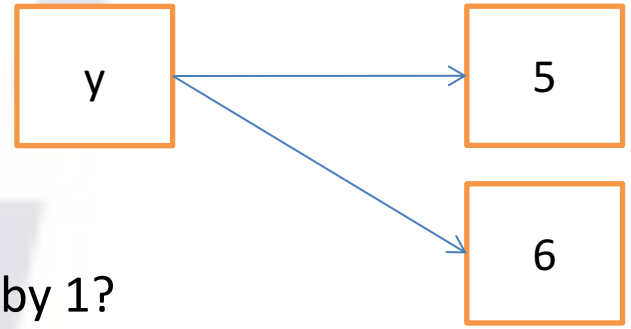
```
>>> y = 5
```



1. First of all, an integer 5 it's created and placed in a memory location
2. The identifier y is created
3. The identifier y is assigned (by means of a pointer) to the memory location that contains the new number 5
4. Now when we say that "*the value of y is 5*" we mean that the name y is assigned to a reference that points towards a memory location, which contains the object 5
5. Notice that Python now has a variable such that:
    1. The identifier is 'y'
    2. The value is 5
    3. The type is **int**
6. **You do not have to DECLARE the variable type!** Python is a "**strong dynamically typed**" language: you can use a new variable without before declare its type.

Assignment

```
>>> y += 1
>>> print(y)
6
```



What happens when the value of y is increased by 1?
Python…

1. …searchs the pointer whose name is y.
2. …retrieves the value (5) of the object to which the reference points
3. …computes the sum 5+1, generates a **new** integer object 6 in a **new** memory location
4. …assigns the name y to a reference towards the new memory location
5. …deletes the old memory location (5), **if there are no pointers left pointing at it**.

Immutable variables

```
>>> x=3              #creates 3, x points to 3
>>> y=x              #creates the name y, that points to 3
>>> x=4              #creates 4, x now points to 4
>>> print(y)
3                    #y points yet to 3
```

- When you modify the data x, you change the reference, but you do NOT touch the reference named y.

- But in Python there are other data types that are **mutable**, or changeable: lists, dictionaries, and user defined class data types

Getting the input from the user

To get the input from the keyboard, you should use the function called `input`.
The general format of the input function is that you write input, an open parenthesis with some optional prompt, and then close parenthesis.
Here is an example of it:

```
user_name=input("Write your name: ")
print("Glad to meet you", user_name)
```

The function `input` returns a string, even if the user types a number. So if in your program you need a number, you must **cast** an explicit data type conversion:

```
user_response=input("Write the current T in C: ")
celsius=int(user_response)
fahrenheit=((celsius*9)/5)+32
print("This is your T in F", fahrenheit)
```

`float` can convert the input to a floating point number.

The Python 2.x `raw_input` has been replaced by `input`

Exercise

Write a program that ask the user the value for radius of a circle, and then print the area and the perimeter.

Unit Topics

- List, Dictionary and Tuple

Learning objectives

- Learn the basic information about Python list
- Describe the difference between the mutable and immutable objects in Python, particularly regarding the identifiers assignments
- Know how to slice a list

Lists

- A list in Python has the same concept as a list in everyday life: it is a sequence of items, in a defined order (so the items order matters).

- You can think of a list in Python as a sequence of containers, arranged side by side, that can hold ANY type of object. The list itself is then a container of objects.

- To create a list in Python all you have to do is put the items that you want to be part of the list inside of a pair of square brackets, separate them with a comma, and assign the result to an identifier.

Totò

Suppose that we have started our list of Antonio De Curtis film productions, with the following three entries:

I soliti ignoti
I due Marescialli
Uccellacci e uccellini

Let's write it in Python

```
>>> film = ["I soliti ignoti", "I due Marescialli", "Uccellacci e uccellini"]
```

How did we make the translation?

1. Each line has been trasformed in a **string**, using **double quote "**
2. Each list item is separated from the next by a **comma ,**
3. The list items are written between **square brackets []**
4. The list is assigned to an identifier (`film`) with the **assignment operator =**

Items types

Can we add the production year?

```
>>> film = ["I soliti ignoti", 1958, "I due Marescialli", 1961, "Uccellacci
e uccellini", 1966]
```

Remember: the Python list is a collection of objects, and any object can be of any known Python type. (until now floats, integer, booleans, strings).

But now you know another Python object: the list! Can we make a list of lists?

```
>>> film2 = [["I soliti ignoti", 1958],
             ["I due Marescialli", 1961],
             ["Uccellacci e uccellini", 1966]]
```

Accessing list elements

In Python square brackets, [], are used to access list elements.
There are two ways to access list elements:

1. Positive indexing (ZERO based)
2. Negative indexing (from -1)

```
In [1]: film = ["I soliti ignoti", 1958, "I due Marescialli", 1961,
INDEX:                  0              1              2          3
                       -6             -5             -4         -3
"Uccellacci e uccellini", 1966]
            4          5
           -2         -1


In [2]: film[3]
Out[2]: 1961

In [3]: film[-2]
Out[3]: 'Uccellacci e uccellini'

In [4]: film[4]
Out[4]: 'Uccellacci e uccellini'
```

List Manipulation

Every list is a object, and as a object has a bunch of METHODS available.

A method is basically a function which is written for a specific object.

The only thing that you need to know at this time is that to use a method, you write the name of the object, put a dot after it, and then write the name of the method. You can think of a method as an action that the object can take on its data.

Methods, as functions, can also take some input arguments.

`list.insert(i,x)`	Insert an item at a given position.
`list.append(x)`	Add an item to the end of the list
`list.remove(x)`	Remove the first item from the list whose value is *x*. It is an error if there is no such item.
`list.pop([i])`	Remove the item at the given position in the list, and return it.
`list.extend(L)`	Extend the list by appending all the items in the given list.
`list.index(x)`	Return the index in the list of the first item whose value is *x*.
`list.count(x)`	Return the number of times *x* appears in the list.
`list.sort()`	Sort the items of the list, in place.
`list.reverse()`	Reverse the elements of the list, in place.

Example: lists operations

```
>>> classe = ["Anna", 'Giulia', 'Vito', "Michele"]
>>> print(classe)
['Anna', 'Giulia', 'Vito', 'Michele']
>>> print(len(classe))
4
>>> print(classe[2])
Vito
>>> classe.append("Rosa")
>>> print(classe)
['Anna', 'Giulia', 'Vito', 'Michele', 'Rosa']
>>> classe.pop()
'Rosa'
>>> classe.extend(["Rosa",'Gioacchino'])
>>> print(classe)
['Anna', 'Giulia', 'Vito', 'Michele', 'Rosa', 'Gioa
>>> classe.remove('Vito')
>>> print(classe)
['Anna', 'Giulia', 'Michele', 'Rosa', 'Gioacchino']
>>> classe.insert(0, 'Maurizio')
>>> classe.insert(0, 'Vito')
>>> classe.insert(0, 'Elio')
>>> print(classe)
['Elio,'Vito','Maurizio','Anna', 'Giulia', 'Michele', 'Rosa', 'Gioacchino']
```

len is a Built-In Function(BIF)

append adds an item to the end

pop Removes the item at the given position in the list, and **returns** it

extend Extends the list by appending all the items in the given list

insert Inserts an item at a given position. The first argument is the index of the element before which to insert

Again on accessing list elements

If you want to modify an item in the list, you can simply put the element you would like to change on the left side on an equal sign, and assign a new value to it.

And this correlates with an important list feature: the list is a **mutable data type**, that is it can be modified directly in the original memory location.

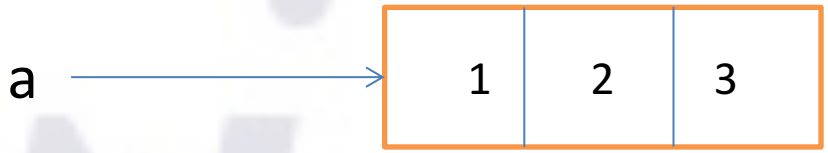
This gives raise to an interesting, and somewhat weird, behavior:

```
x=5
y=x
x=6
print(y)
5

a=[1,2,3]          #a points to the list[1,2,3]
b=a                #b point to the same list object
a.append(4)        #this modifies the list object
print(b)
[1,2,3,4]              #b is also changed
```

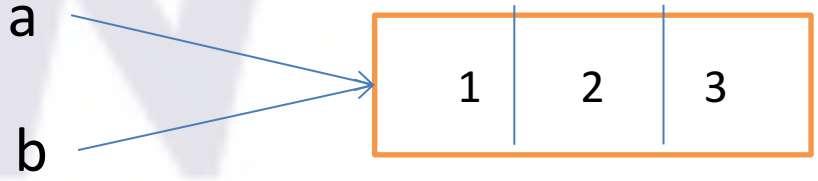
As you can easily imagine you have to pay a special care when you pass an identifier as a input parameter of a function.

Intuition

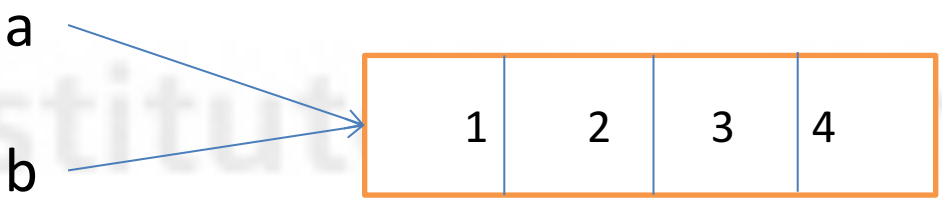
a=[1,2,3]



b=a



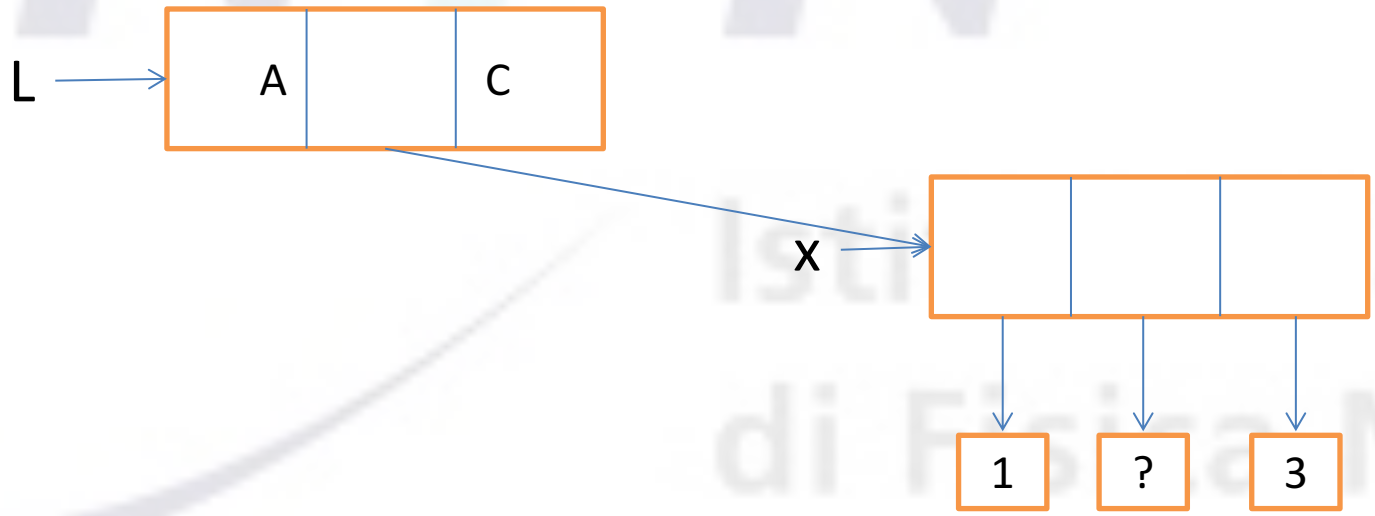
a.append(4)



Shared references

```
>>> x = [1,2,3]
>>> L = ['A',x,'C']
>>> L
['A', [1,2,3], 'C']
>>> x[1] = 'sorpresa!'
>>> L
['A', [1,'sorpresa!',2], 'C']
```

The change in the object pointed by x is visible from every reference that point to the same object.



List slicing

With a single index you can access to a single list item.

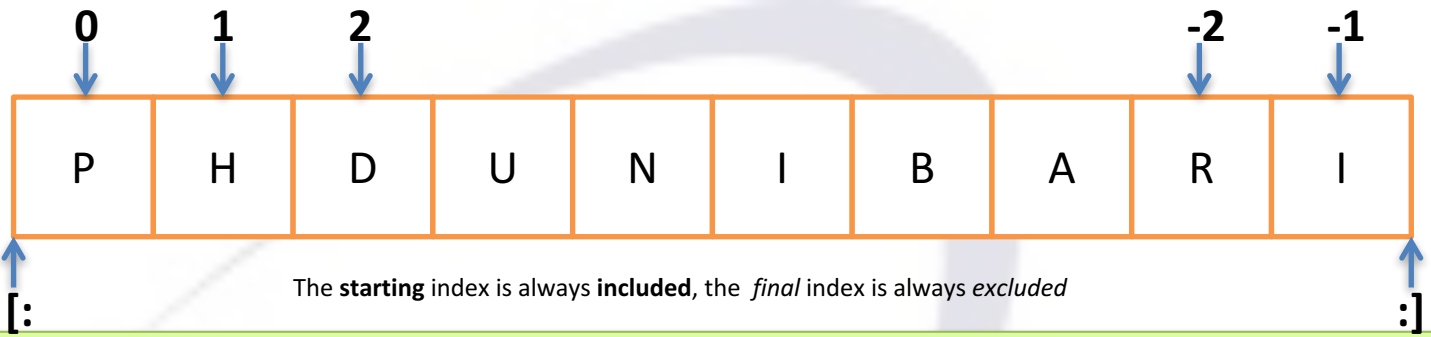
Slicing allows you to extract or refer to a section of the original list (like a slice in a cake)

To slice a list you must use two indexes, separated by a column in this format:

```
[ start : end ]
```

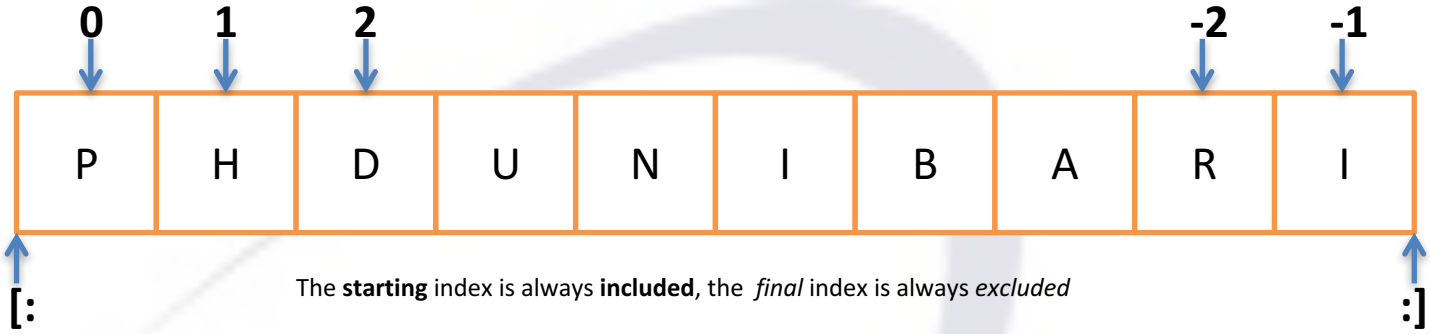
inclusive **exclusive**

1. If you assign a value to a slice you change the original list **on site**.
2. If you assign a slice to an identifier, you create **a copy** of the original list
3. Always remember that the first index is ZERO
4. A third parameter can indicate the slicing step.



The **starting** index is always **included**, the *final* index is always *excluded*

```
>>> a = 'demonstrate slicing in Python'.split()
>>> print(a)
['demonstrate', 'slicing', 'in', 'Python']

>>> a[-1]          # the last entry
'Python'

>>> a[:-1]         # everything up to but, not including, the last entry
['demonstrate', 'slicing', 'in']   #Equal to a[0:len(a)-1]

>>> a[:]           # everything
['demonstrate', 'slicing', 'in', 'Python']

>>> a[2:]          # everything from index 2 and upwards
['in', 'Python'] #Equal to a[2:len(a)]

>>> a[-1:]         # the last entry
['Python']

>>> a[-2:]         # the last two entries
['in', 'Python']
```

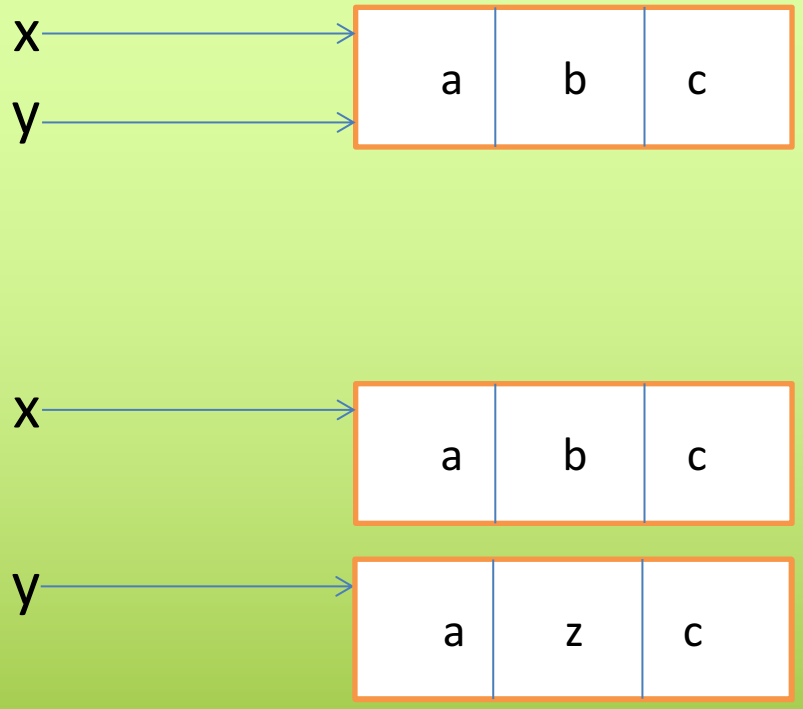



```
>>> print(a)
['demonstrate', 'slicing', 'in', 'Python']
>>> a[1:3]                      # from index 1 to 3-1=2
['slicing', 'in']
>>> a[:0] = 'here we'.split()   # add list in the beginning (to the
                                # first, not included)
>>> a
['here', 'we', 'demonstrate', 'slicing', 'in', 'Python']


>>> b = [2.0]*6                 # create list of 6 entries,each equal 2.0
>>> b
[2.0, 2.0, 2.0, 2.0, 2.0, 2.0]
>>> b[1] = 10                   # b[1] becomes the integer 10
>>> c = b[:3]
>>> c
[2.0, 10, 2.0]
>>> c[1] = 20                   # is b[1] affected?
>>> b
[2.0, 10, 2.0, 2.0, 2.0, 2.0] # no c is a copy of b[:3] (slicing is a copy)
>>> b[:3] = [-1]                # first three entries replaced by one entry
>>> b
[-1, 2.0, 2.0, 2.0]
```

How to copy a list

You do have any element now to make a real copy of a list object: any idea?

```
>>> x = ['a','b','c']
>>> y = x
>>> y[1] = 'z'
>>> x
['a','z','c']
>>> y
['a','z','c']

>>> x = ['a','b','c']
>>> y=x[:]
>>> y[1] = 'z'
>>> x
['a','b','c']
>>> y
['a','z','c']
```



Another list…

At this point you already know another Python type, that is also a list of items, but that is different from the list type…

```
>>> x = 'mezzo'
>>> y = x
>>> y[1] = 'o'
---------------------------------------------------------------------------
TypeError Traceback (most recent call last) <ipython-input-3-dfc31d13e542>
in <module>() ----> 1
y[1]='o' TypeError: 'str' object does not support item assignment
```

The string is a list of characters, but it's IMMUTABLE, so you cannot change its items in place, but you can access the characters with the index notation and slice it.

Any string is also a object, and has a bunch of methods available. (Use dir or help or the online documentation)

tuple, an immutable list

Python has also a immutable list of items, called **tupla**. You can access it with the usual index notation, but you cannot modify it in place. The tupla is ordered.

```
>>> x = ('a',1,'b',2)
>>> x[1] = '3'
---------------------------------------------------------------------------
TypeError Traceback (most recent call last) <ipython-input-7-487b187cc2fa>
in <module>() ----> 1 x[1]=3
TypeError: 'tuple' object does not support item assignment
```

An **unordered** tupla with an **automatic selection** of identical item is a **set**

```
>>> x = {'a',1,'b',2,'a',1}
>>> print(x)
{'b', 1, 2, 'a'}
```

What do you think: if the tuple contains a list, can you change the values of the list on-site?

Dictionaries

A dictionary in Python is a **mutable unordered** collection of items. Each item in a dictionary has a key, and each key is associated with a value.

You can think of a dictionary as a bag of items rather than a sequence of items. Since items in a dictionary are not in sequence, you cannot use an index to refer to them.

Keys can be of any type, as long as is an IMMUTABLE one.

Values can be of any type, *including lists and dictionaries*

A single dictionary can contain *different types* of values.

You can define, edit, view, search, and delete key-value pairs in the dictionary.

In the dictionary it is not maintained any information about the element location: the keys provide the symbolic location, not physical, of the items of a dictionary

Examples

```
>>> d1 = { 'key1' : 'value1', 'key2' : 'value2' }
>>> d2 = dict(key1='value1', key2='value2')

>>> d3={}
>>> d3['key1'] = 'value1'
>>> d3['key2'] = 'value2'
>>> d3['key3'] = 'value3'

>>> for key in d3:
        print("d3['%s']=%s" % (key, d3[key]))
d3['key2']=value2
d3['key1']=value1
d3['key3']=value3

>>> for key in sorted(d3):
        print("d3['%s']=%s" % (key, d3[key]))
d3['key1']=value1
d3['key2']=value2
d3['key3']=value3
```

Copy and assignment

The dictionaries behave like lists with regard to copy and assignment.

```
>>> a = dict(q=6, error=None)
>>> b = a
>>> a['r'] = 2.5
>>> b
{'q': 6, 'r': 2.5, 'error': None}
>>> b is a
True
>>> a = 'a string'              # make a refer to a new (string) object
>>> b                           # new contents in a do not affect b
{'q': 6, 'r': 2.5, 'error': None}
>>> b is a
False
```

What if you want a copy?

```
>>> a = dict(q=6, error=None)
>>> b = a.copy()
>>> b is a
False
>>> b['q'] = 7
>>> a['q']                      # not affected by assignment to b['q']
6
```

References

- Microsoft - DAT208x
- Microsoft - DAT210x
- UTAx - CSE1309x