MASTER IN ENTREPRENEURSHIP
INNOVATION MANAGEMENT
IN COLLABORATION WITH MIT SLOAN

IN COLLABORATION WITH
MIT MANAGEMENT SLOAN SCHOOL

UNIVERSITÀ DEGLI STUDI DI NAPOLI
PARTHENOPE

MASTER MEIM 2021-2022

AI for Python

Lesson given by prof. Francesco Camastra

Overview

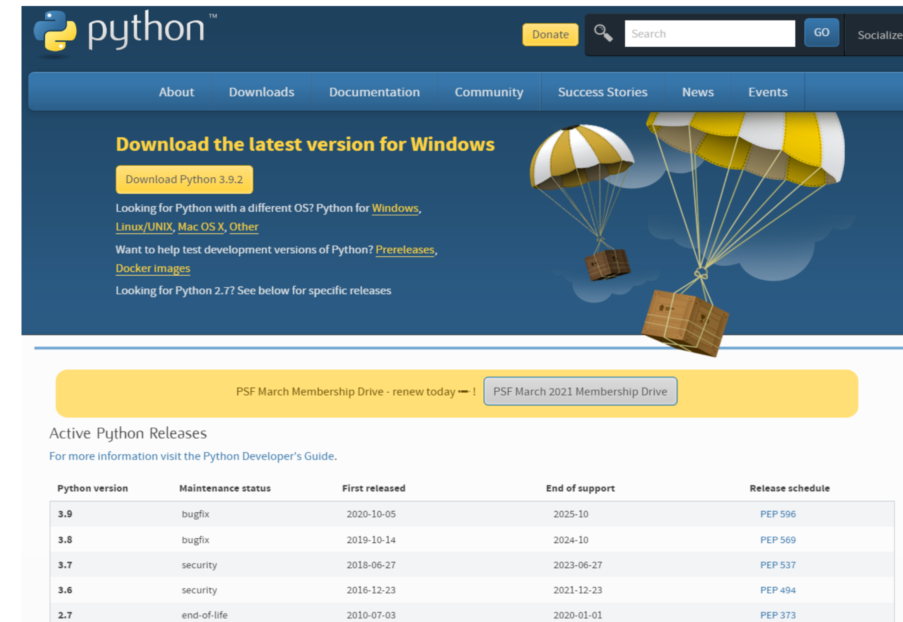
- Installations
- Lists and Tuples
- To write statements using the Boolean data type
- To develop strategies for testing your programs
- To validate user input

Python version

- Python has two main versions, versioni 2.x and 3.x.
- Version 3 is not compatible with version 2 (obsolete). A program, written for version 3, cannot works for version 2, and viceversa. In this course **we will refer only to version 3** (Python 3.x).

Software Installation

- Download from https://www.python.org/downloads/ version **3** for the Operating System of own PC, at 32 or 64 bits
- Example: python-3.8.8.exe or python-3.8.8-amd64.exe

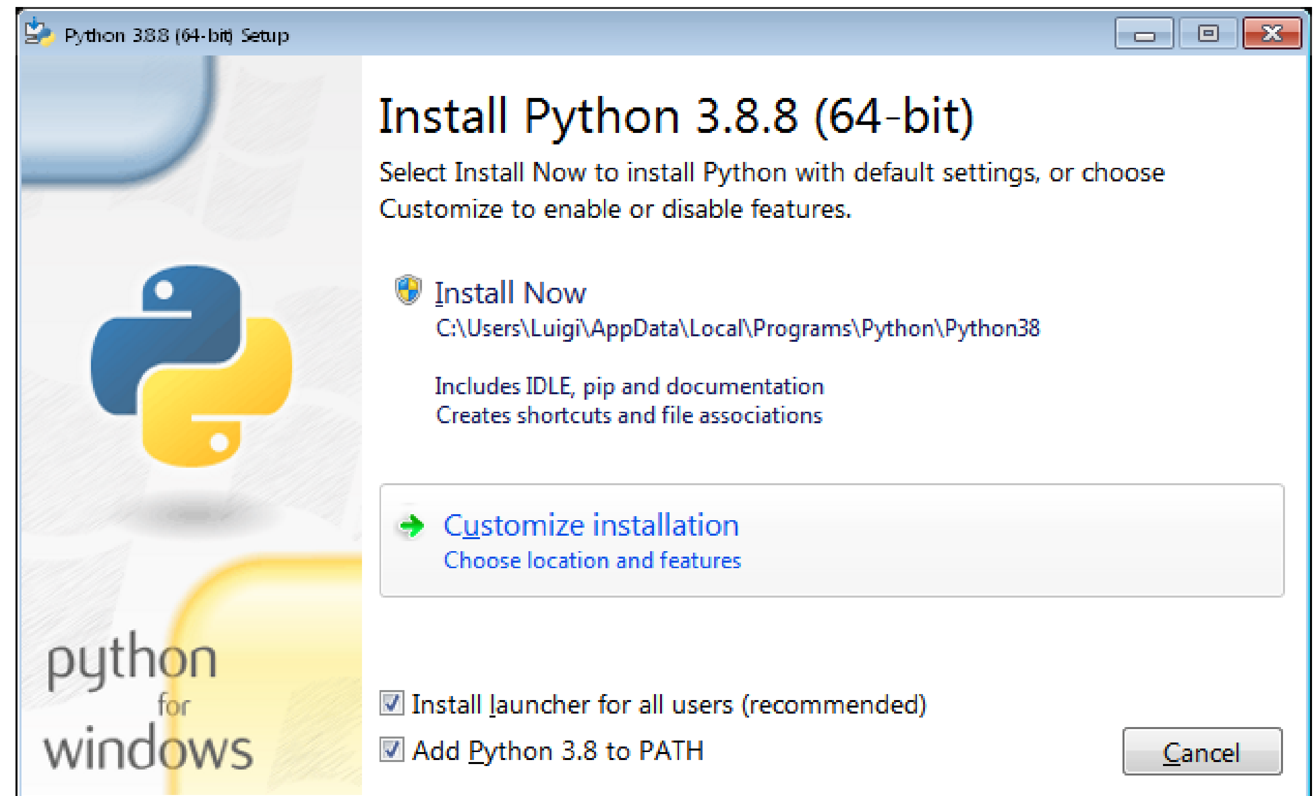


Installation 1

- Launch the .exe file (double click) for installing.

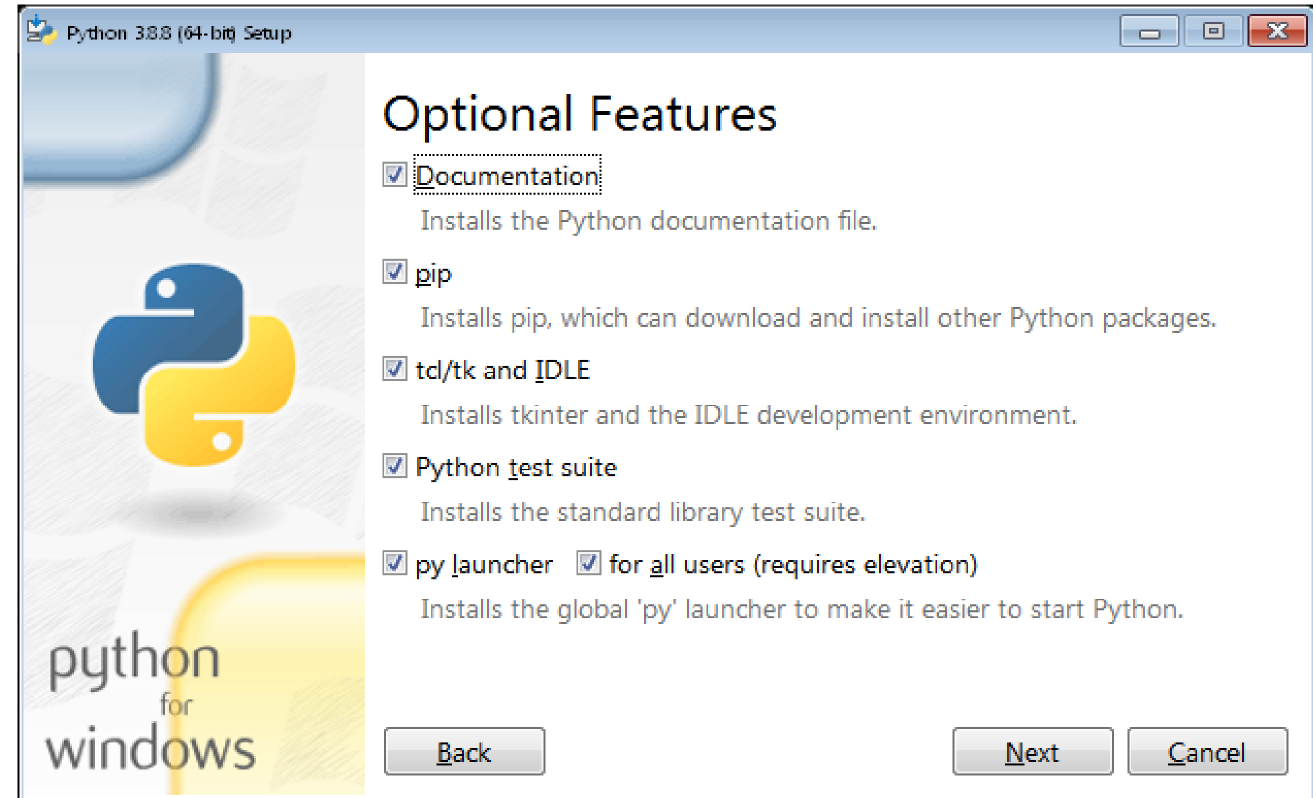
Installation 2

- Starting **with Customize Installation** for activating the use of all users.



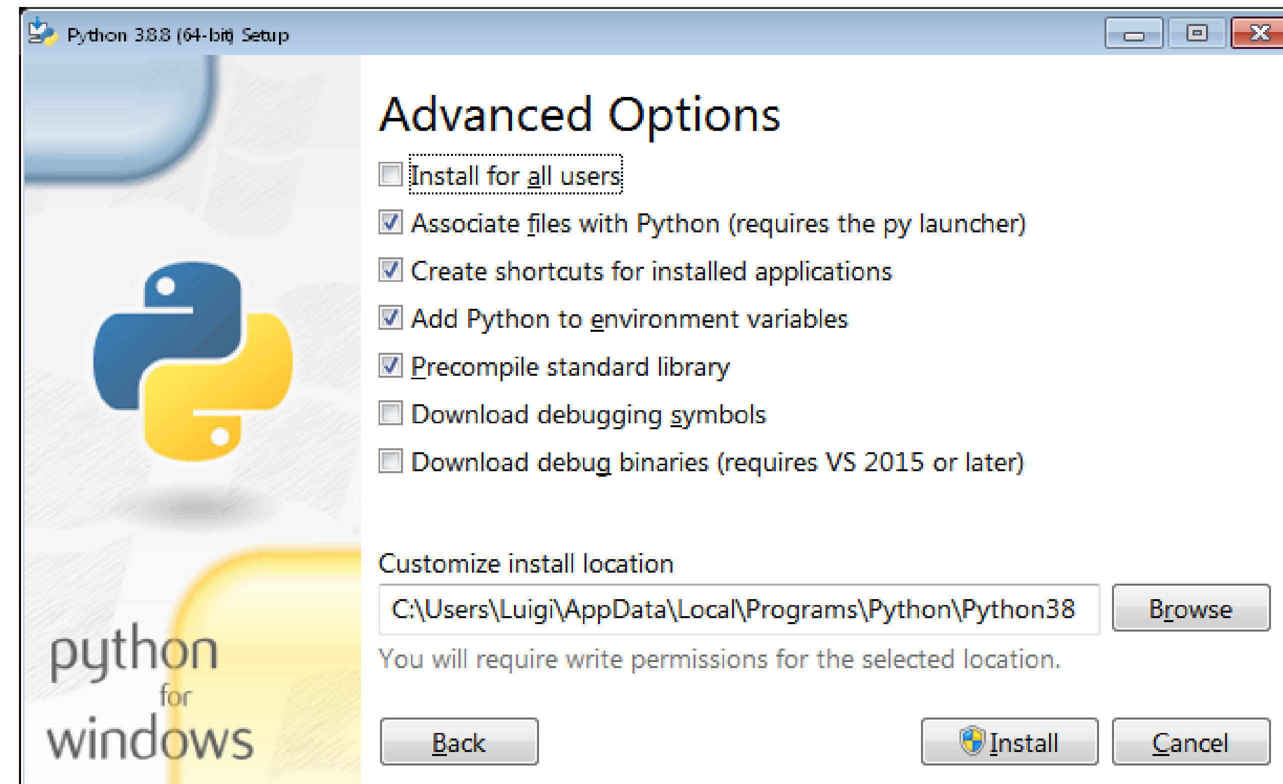
Installation 3

- In this page, leave default Options.



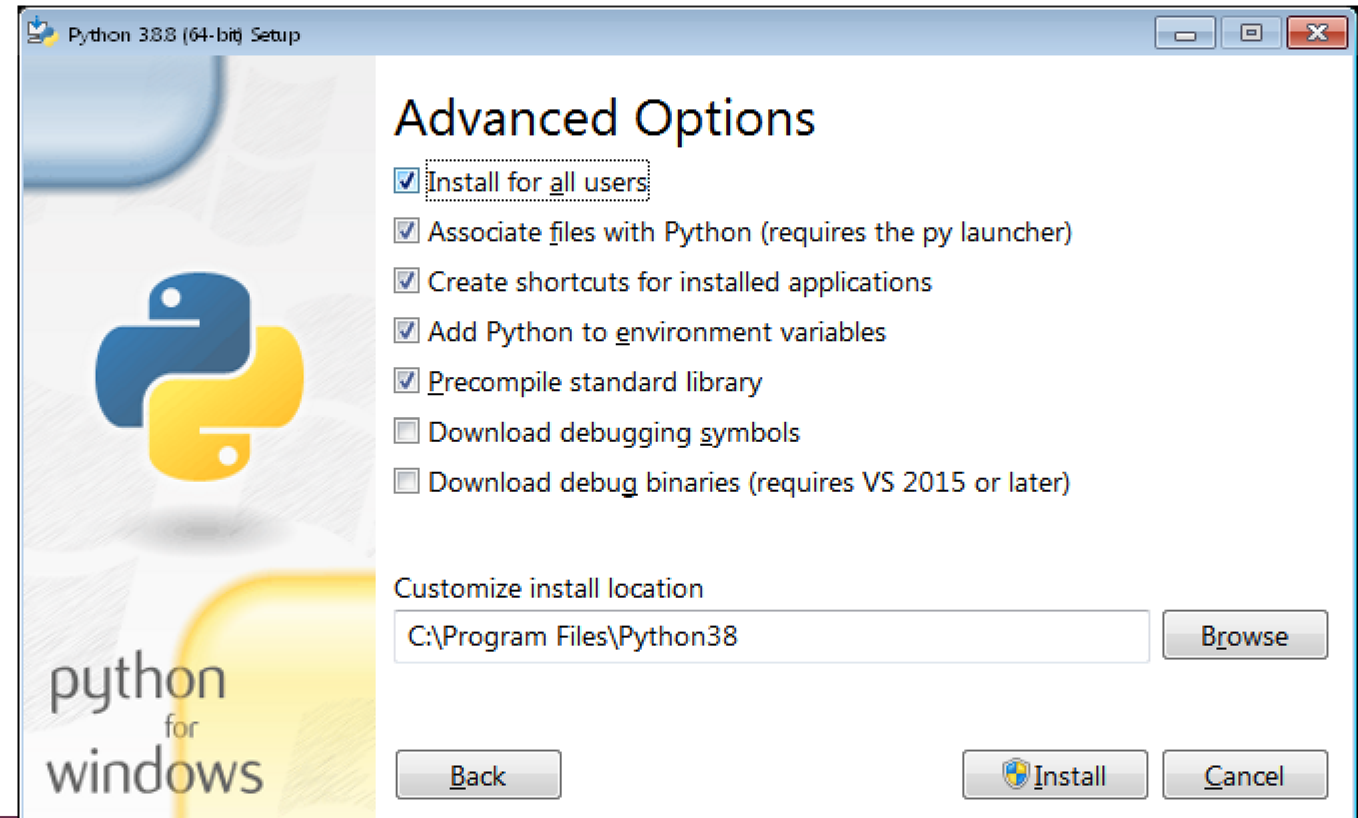
Installation 4

- Tick Install for users option to allow all user the Python access.



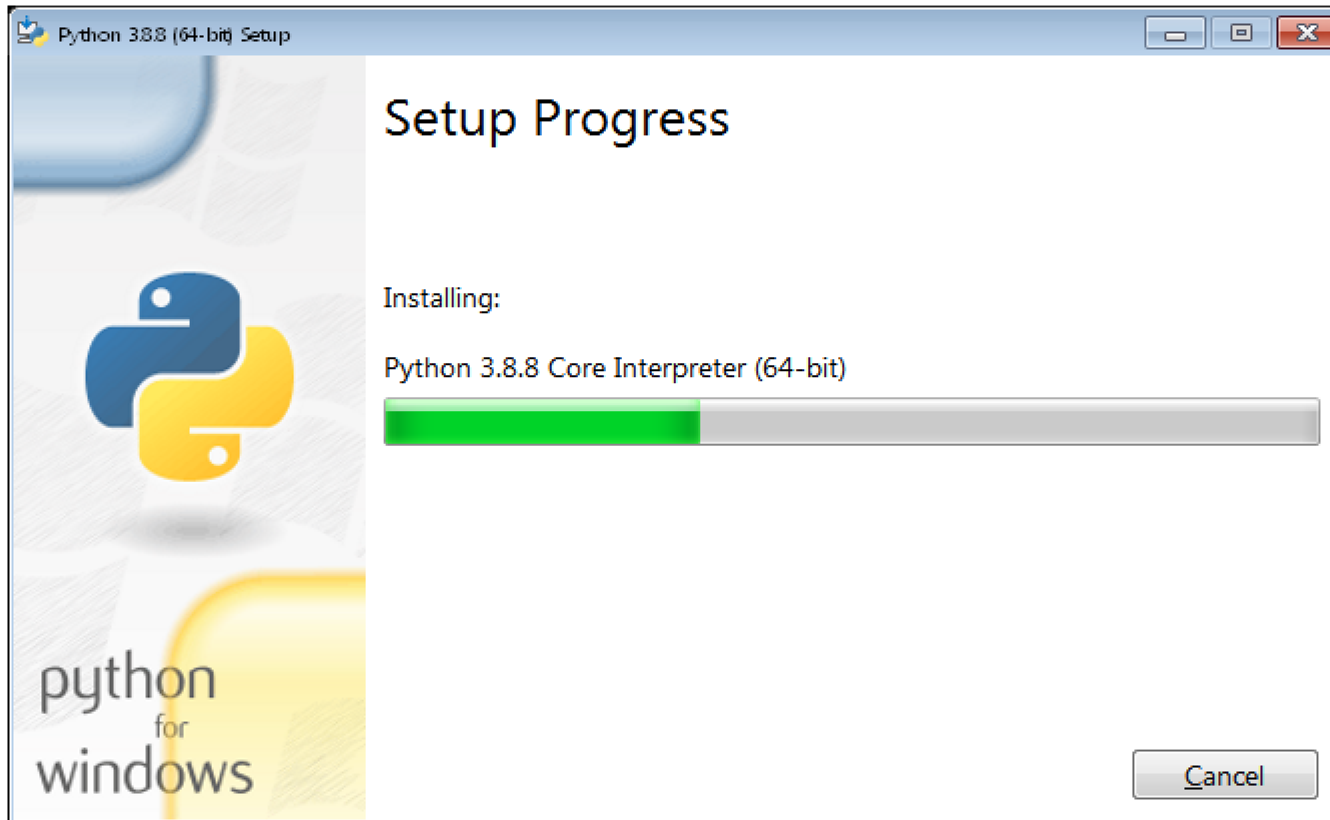
Installation 5

- The installed software, in this case, will go in *C:\Program Files\Python38*
- Proceed with **Install**



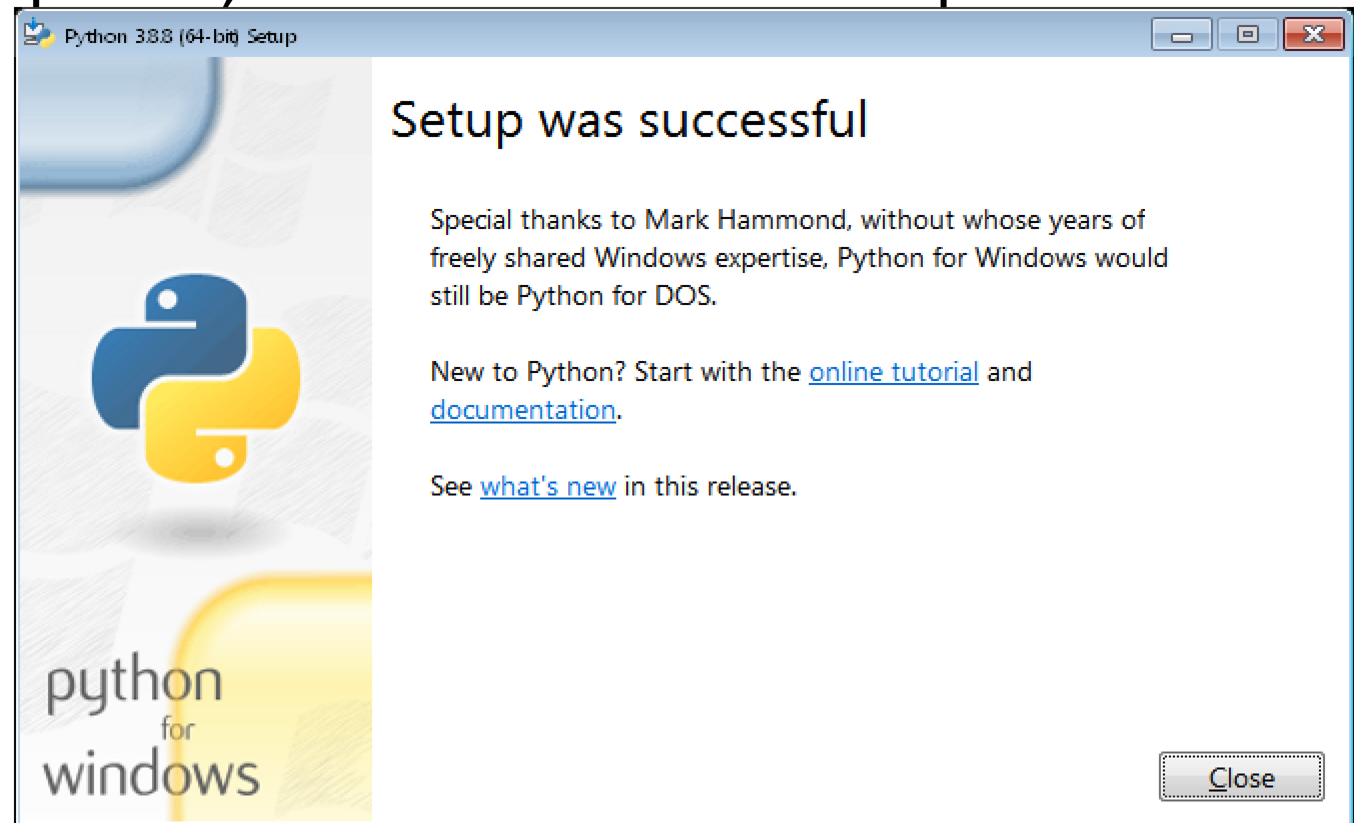
Installation 6

- Waiting for the end of the process…



Installation 7

- When the installation is completed, close the window setup.

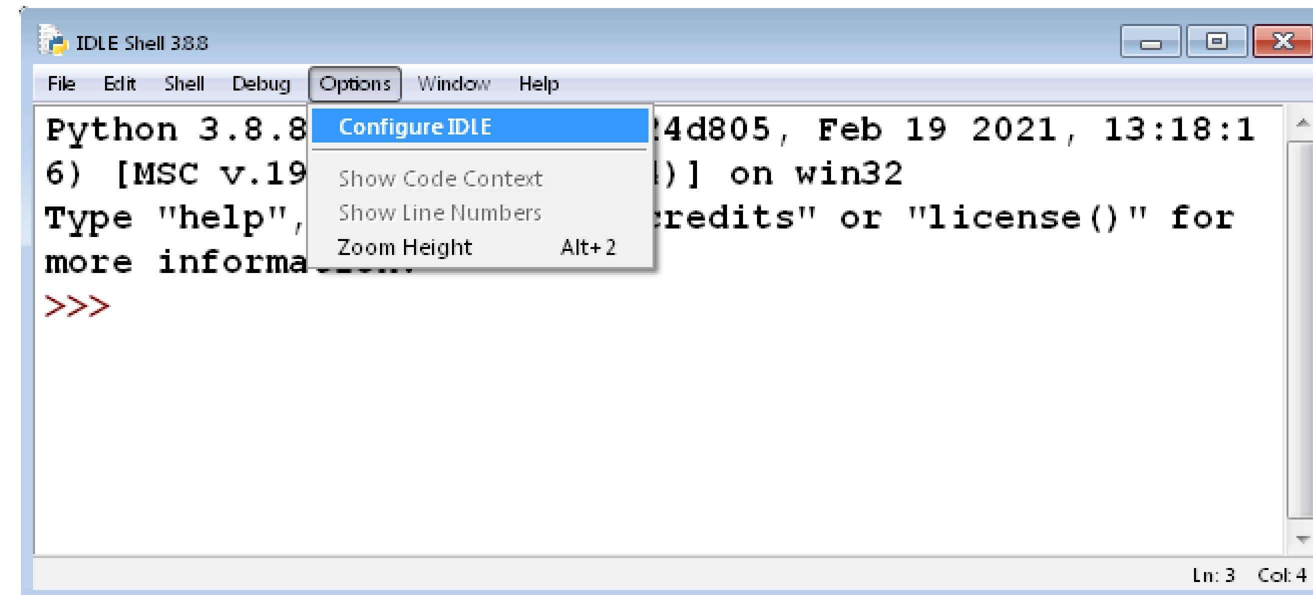


IDLE

- Python installtion provides IDLE, that is a ***development environment*** with Editor and "Comand Line".
- Launch i IDLE that is inserted by Python installation in Windows programs.

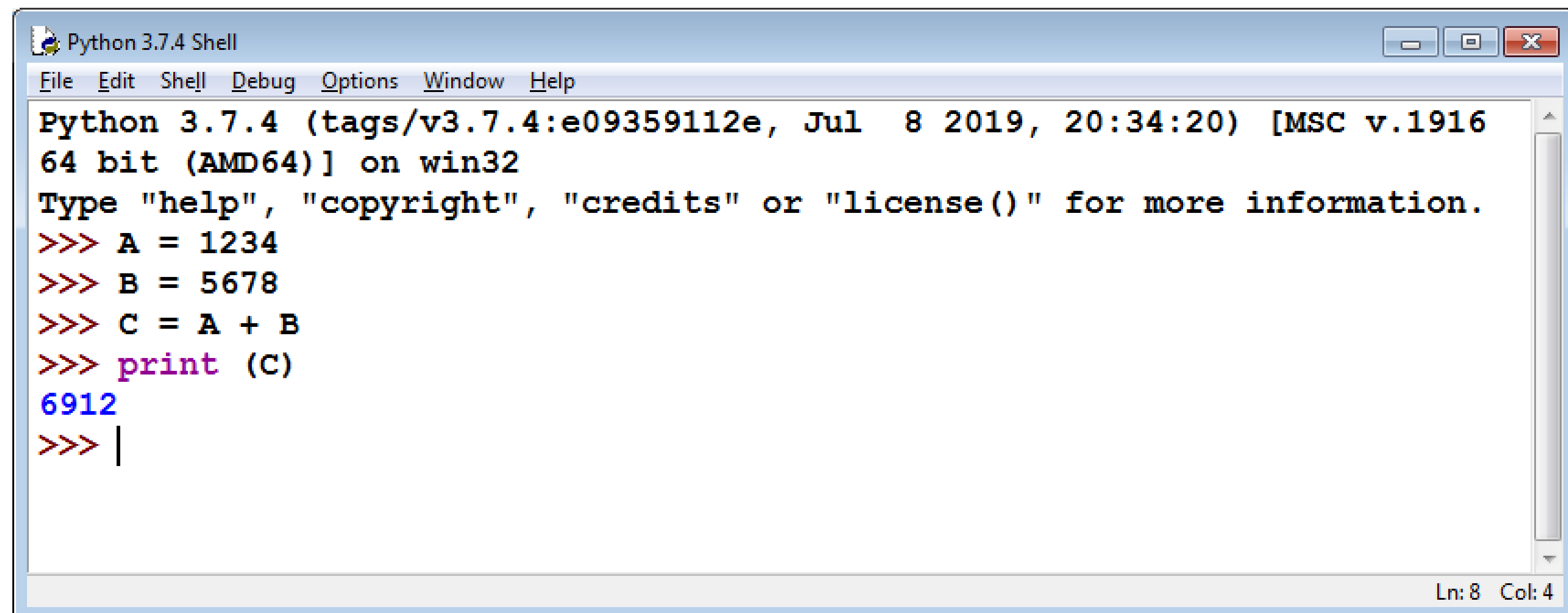
IDLE personalization

- The 3 angle brackets **>>>** represent the command prompt.
- To personalize IDLE according to own requirements, choose the voice Configure IDLE in the menu **Options**.



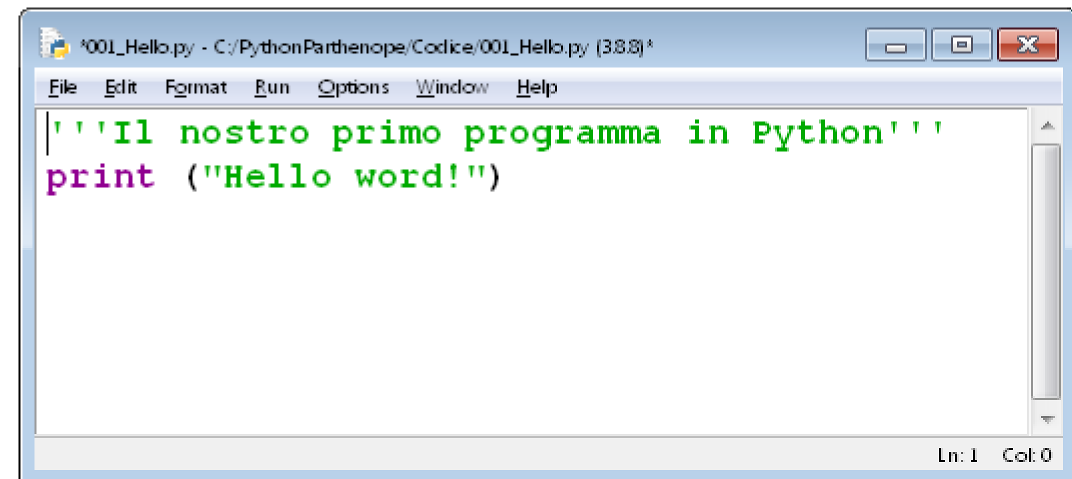
Python Interpreter

- Python Interpreter can be used writing an instruction at a time in IDLE window, close to the ***prompt*** **>>>**



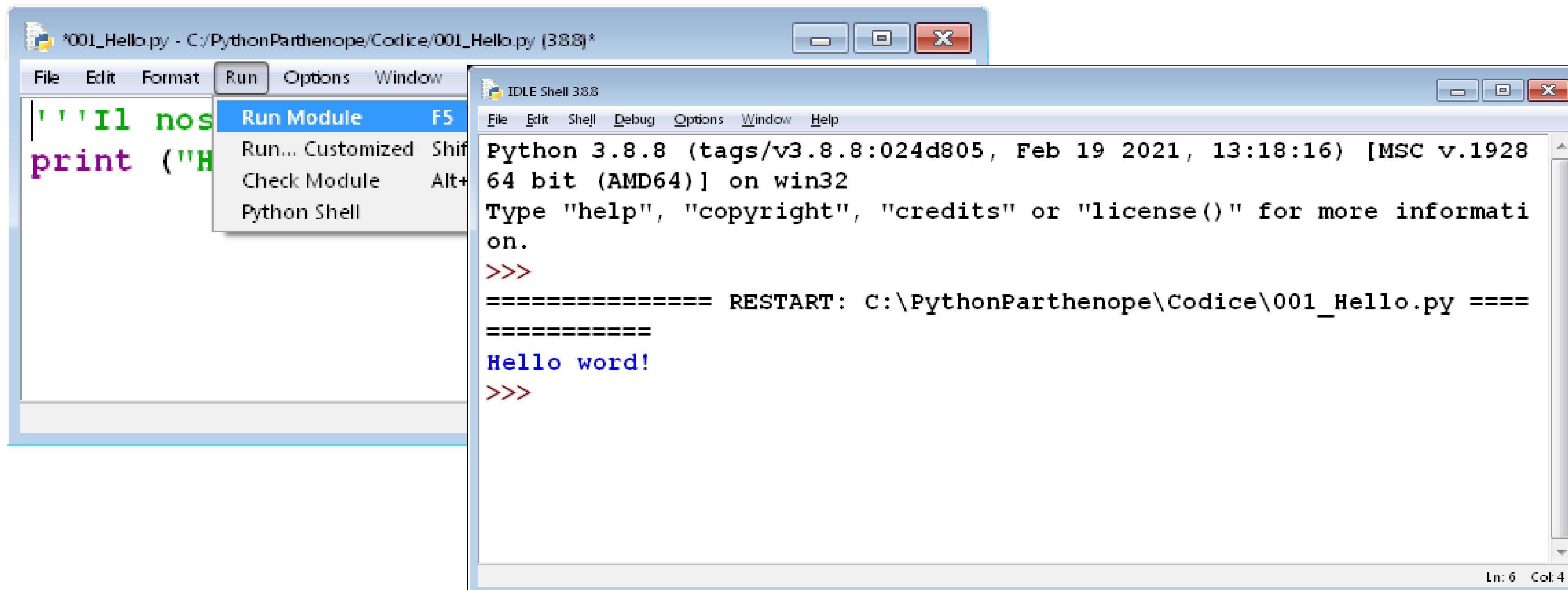
Python Program

A Python Program is a file with extension **.py** written by a text editor. If you want to use IDLE text editor: from IDLE menu select File/New, write the program in the editor window and then save.



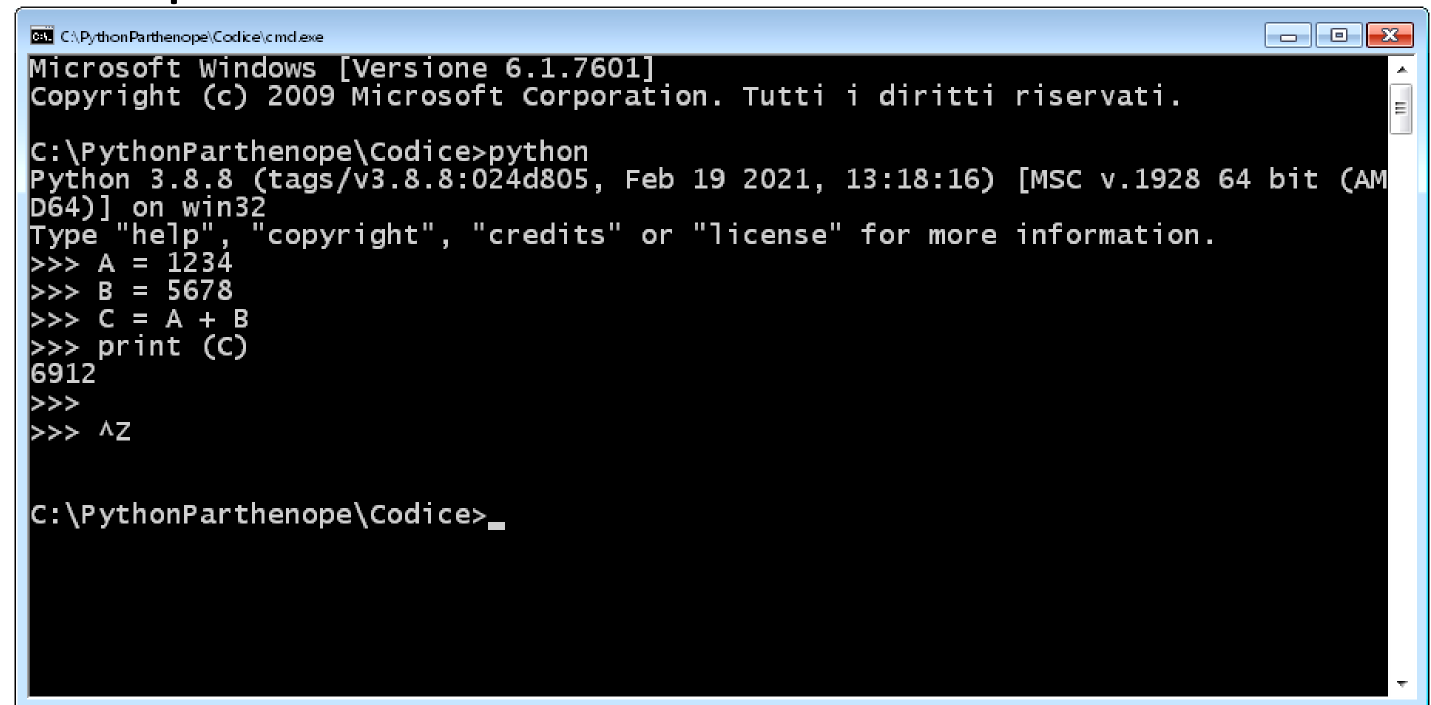
Program Execution

- For the execution of a program .py in IDLE, select **RUN Module** in Run menu or press the key **F5**.



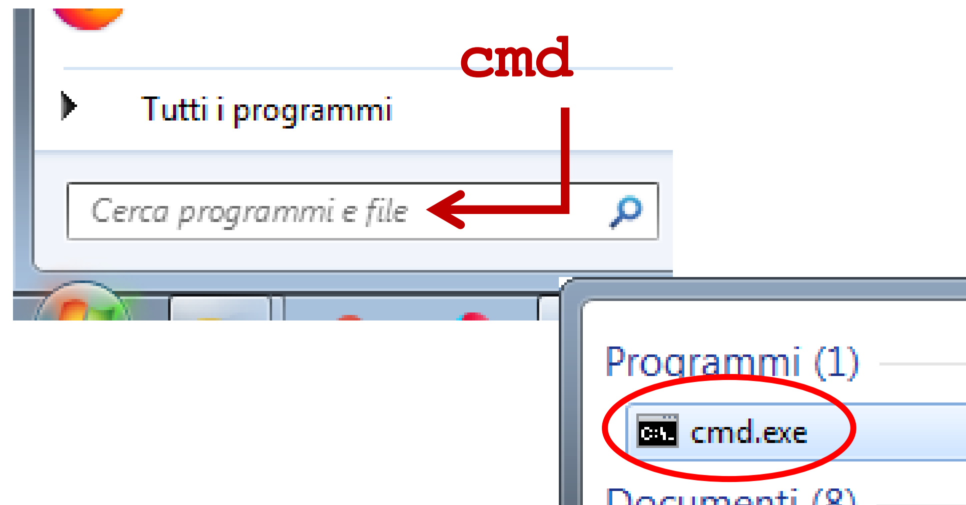
Python Interpreter from console

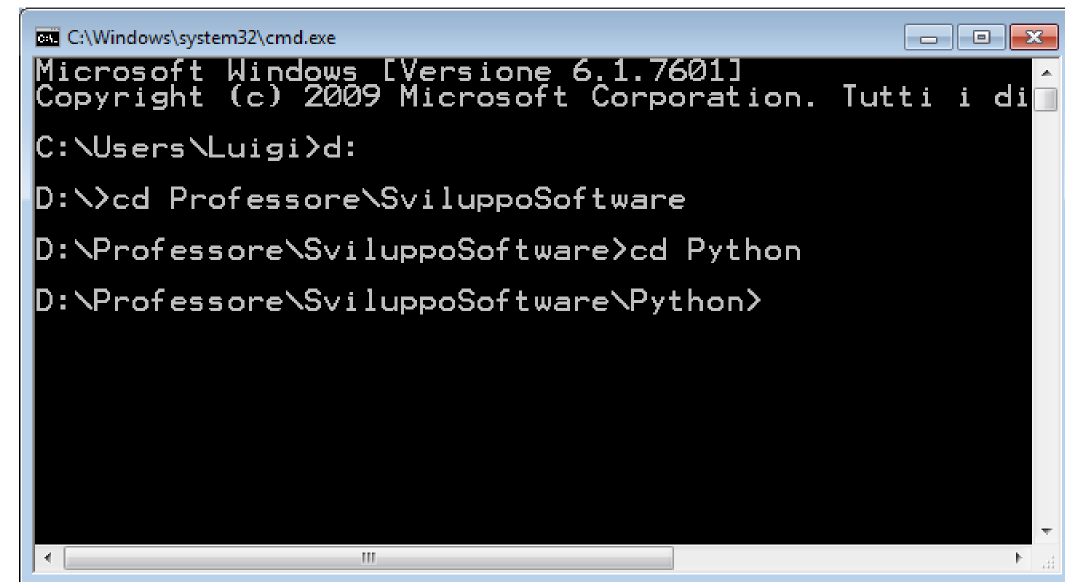
- From comman prompt, Python interpreter can be recalled "*at single instruction*". **Ctrl-Z** close the interpreter.



Console

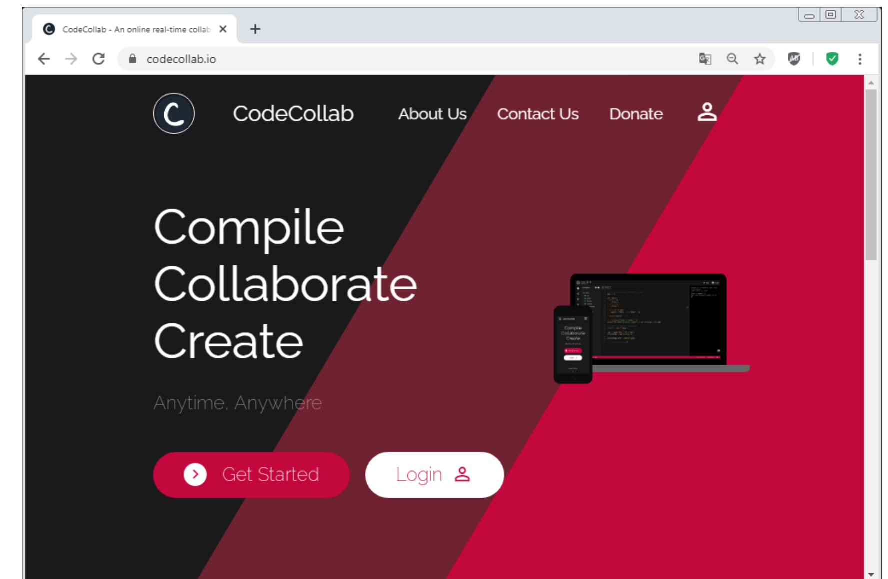
- To open the console, run the program **Cmd.exe** from menu Start of Windows.





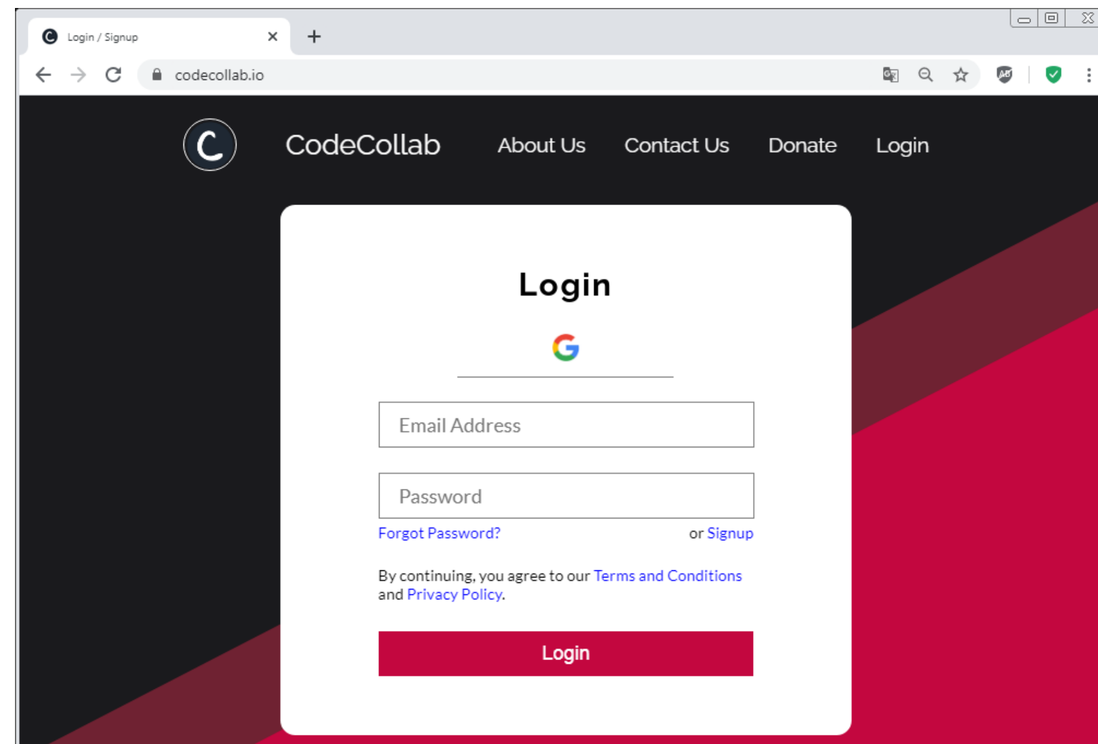
Python online RealTime

- If you do not want to install Python interpreter on own computer, you can use services provided by some online platform.
- For instance , you can use **CodeCollab** that allows sharing a project among some programmers.



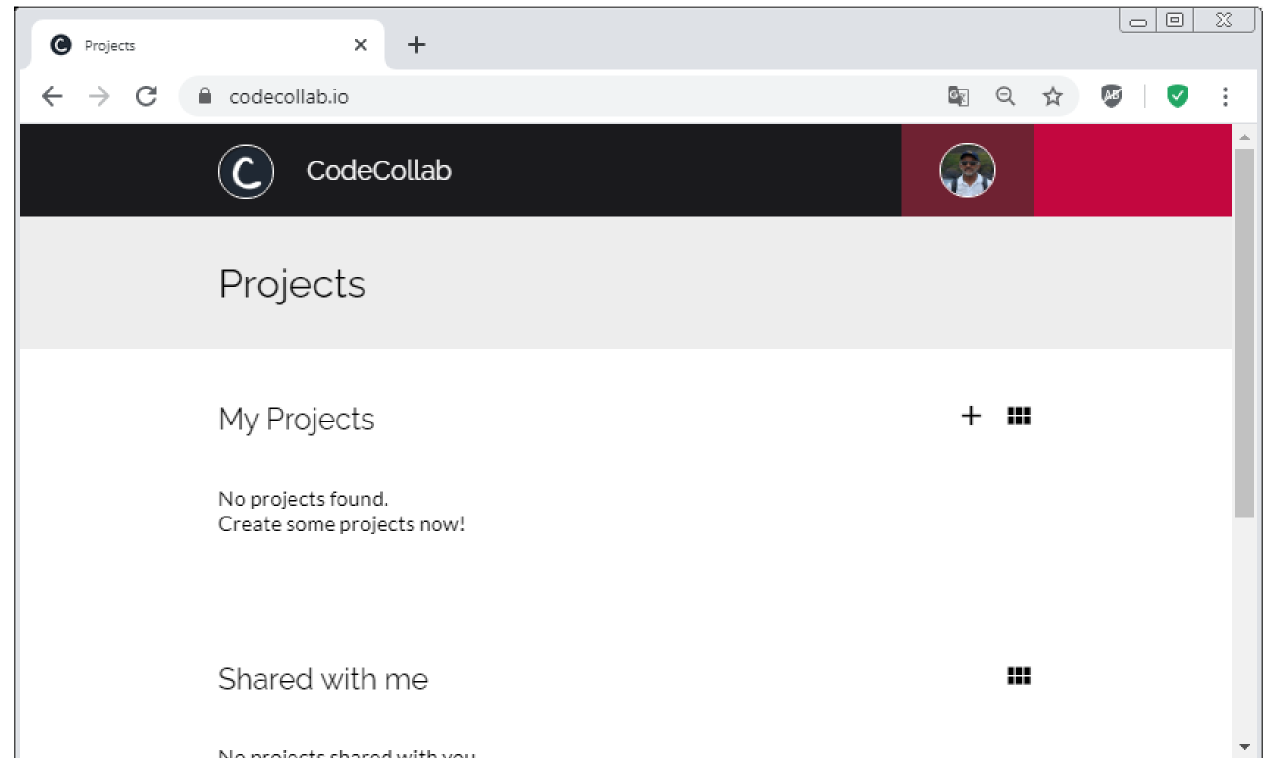
Codecollab Registration

- To use https://codecollab.io/ it is necessary to make a registration using a Google account.



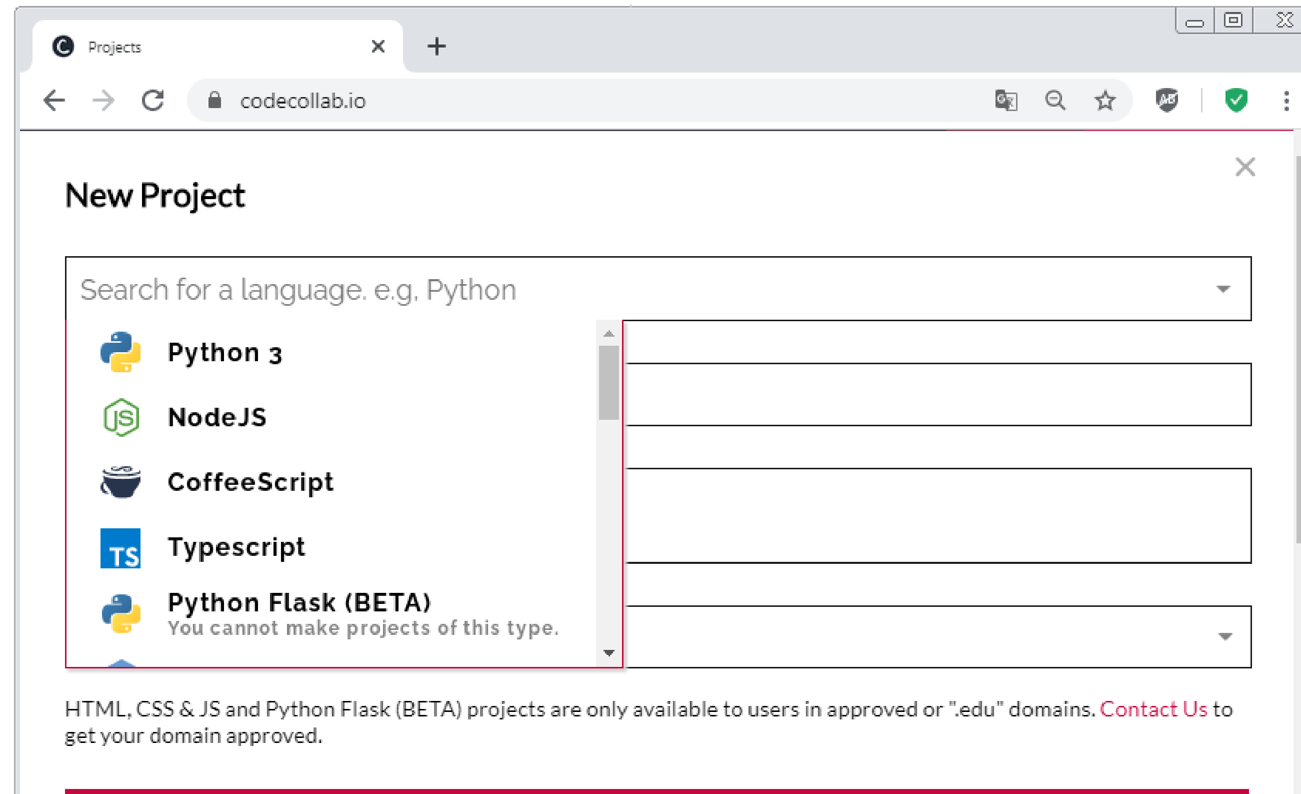
Projects

- In the project window, select **+** for the creation of a new project.



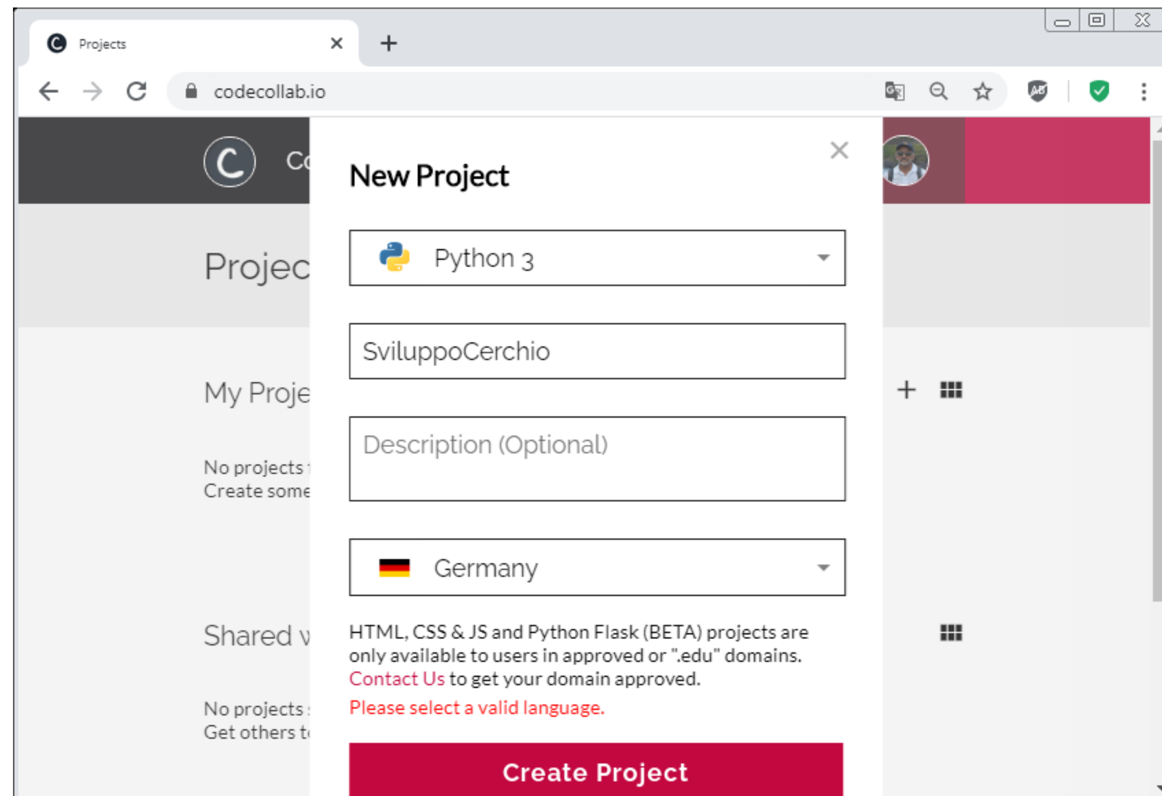
A new Project

- Choose the Programming Language Python



A new Project

- Insert the name of the project and proceed.



Python for Android

QPython is a Python application for Android.

In Qpython there are many resources such as Python interpreter, runtime environment, editor, QPYI library and SL4A.

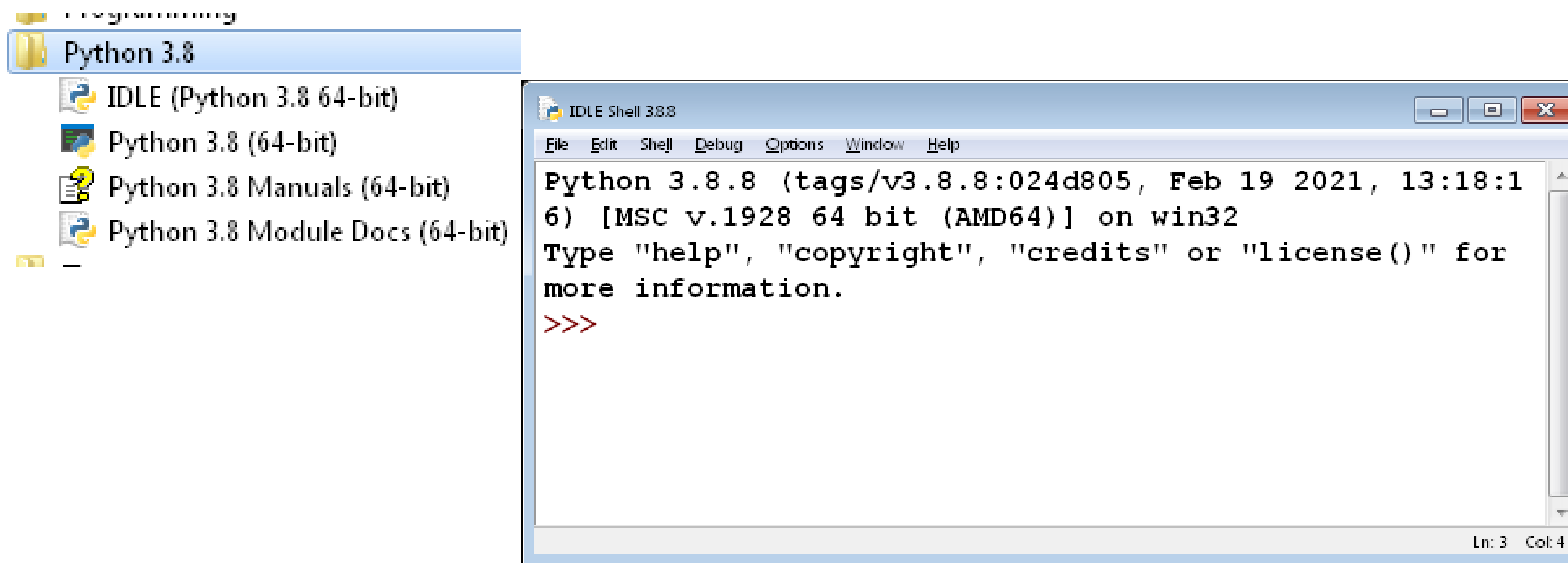
Python for Linux

```
sudo apt-get install build-essential          #Install Dependencies
sudo apt-get install libreadline-gplv2-dev libncursesw5-dev libssl-dev libsqlite3-dev tk-dev libgdbm-dev libc6-dev libbz2-dev
#Download Python
wget http://python.org/ftp/python/2.7.5/Python-2.7.5.tgz
tar -xvf Python-2.7.5.tgz
cd Python-2.7.5
./configure                                   #Install Python
make
sudo make install
```

IDLE

IDLE provides a tool that allows the user of the Python Interpreter an instruction at a time:

- Run the program IDLE in the program menu of Windows.



Python Syntax

- Python is a case sensitive language, i.e., it consider diverse Uppercase and lowrcase.
- For instance HELLO and hello are different expressions.

Python Keyword

Python has the following Keywords:

and as assert break class

continue def del elif else

except False finally for from

global if import in is

lambda None nonlocal not or

pass raise return True try

while with yield

Python indentation

- Python uses indentation to identify the nested blocks, in conjunction with the character (**:**), therefore in Python instructions **MUST** be **Indented**
- A **block** of code is a sequence of instructions grouped on the basis of the alignment and they are handled by the interpreter as they were a single instruction.
- This rule requires Python programs indented correctly, increasing, in this way, the code readibility.
- The standard ***indentation*** is composed of **4 spaces (click 4 bar space)**.

Variables

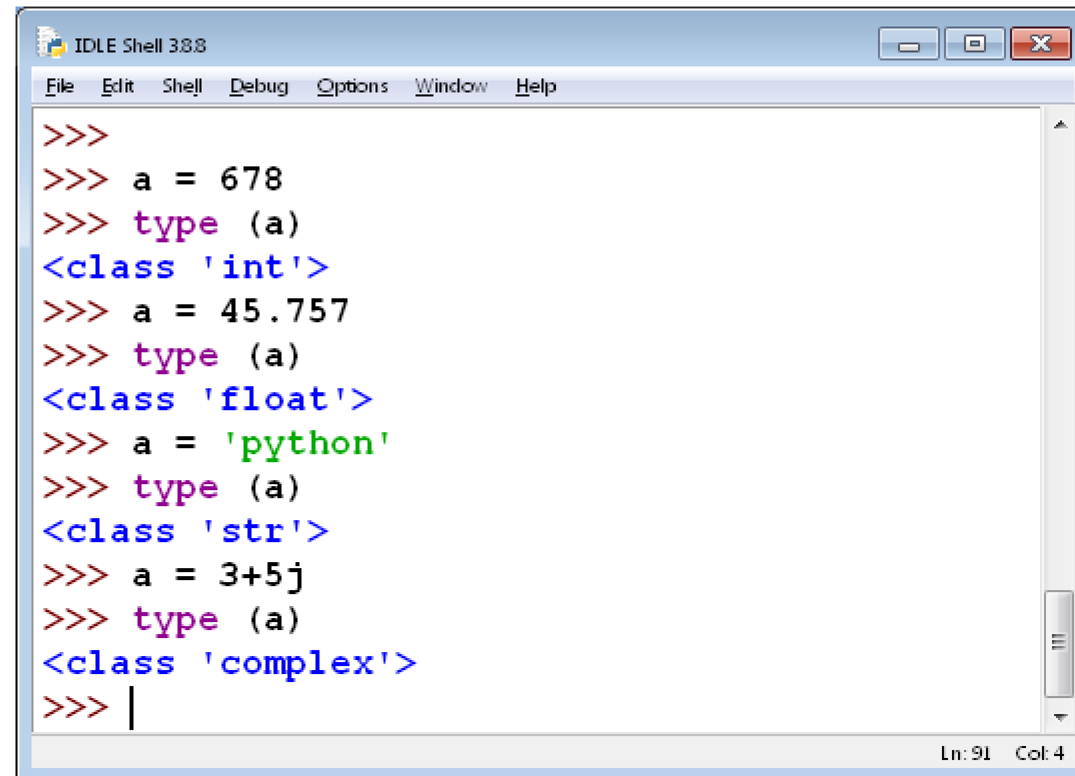
- A variable is a container with a label (the name of the variable) that can be associated to different types during his life time.
- A name is composed of **letters**, **digits**, or **underscore** (_ character), but it must start with a letter or underscore (e.g., maria22, _maria but not 2maria).
- A Variable is **created at his first use** of its assignment (e.g., maria22 **=** "girl")

Numbers

- Python for representing numbers offers two different types, **int** (to represent integer number), **float** (to represent real number)

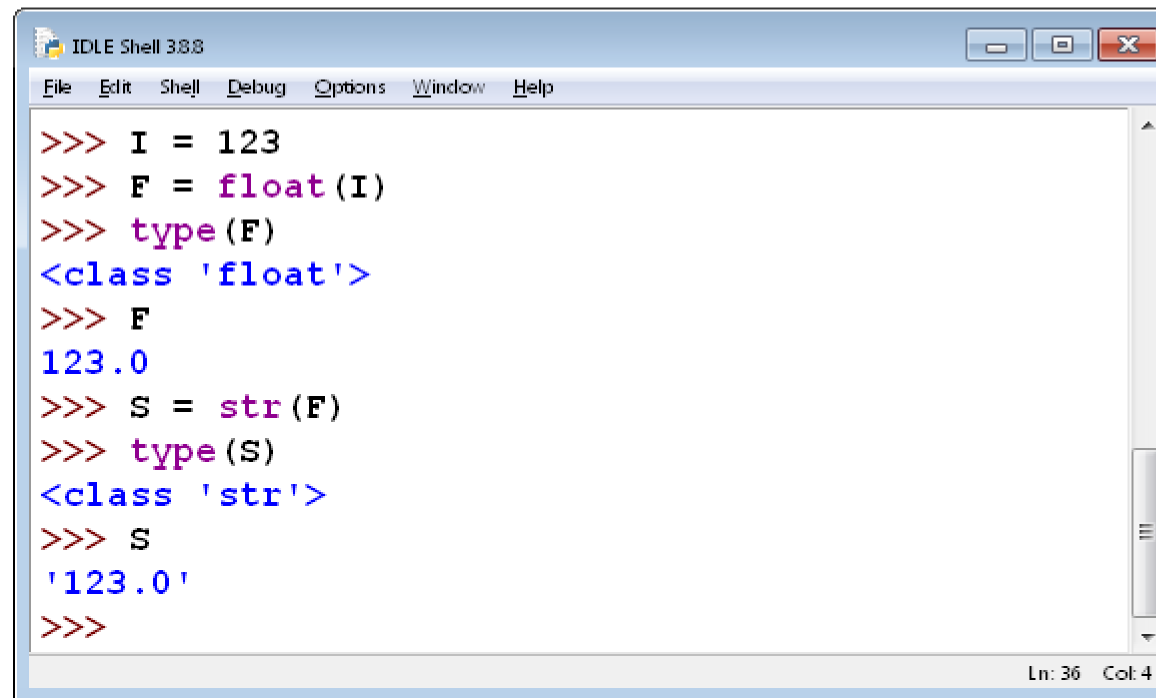
Types of a variable

- The same variable can be used associated several time, each time associated at a different **type**



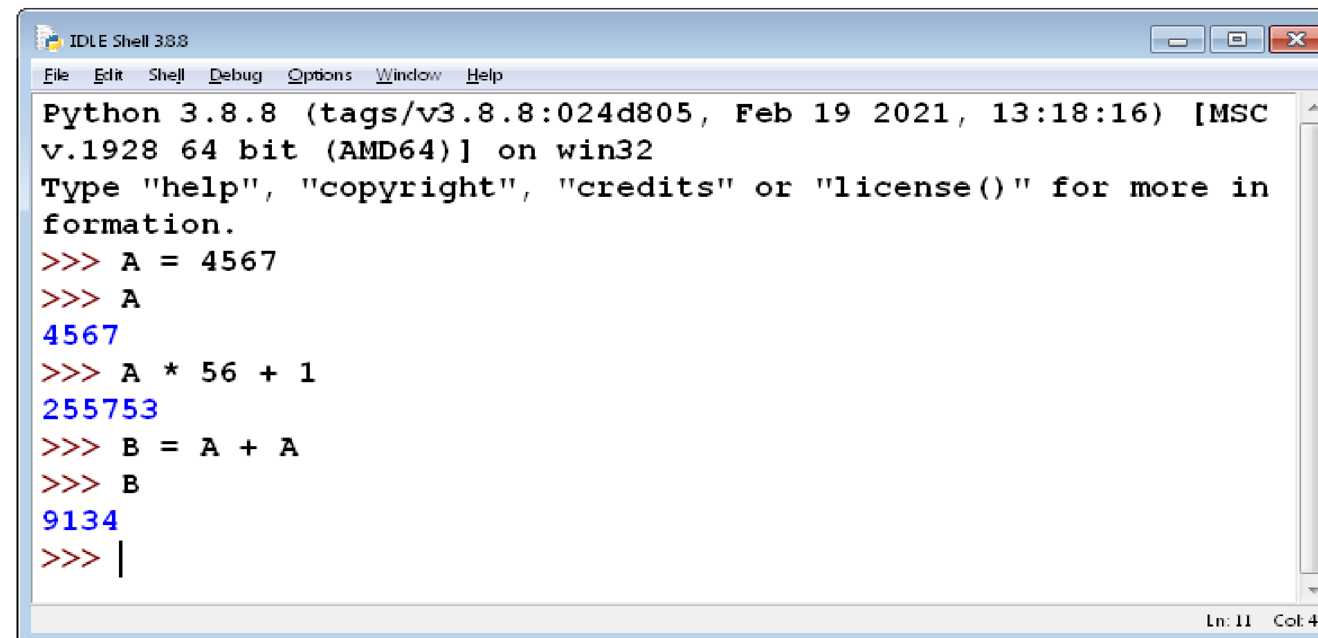
Type conversion

- If we want convert a value from a type to another, we can use functions **int**(), **float**(), **str**()



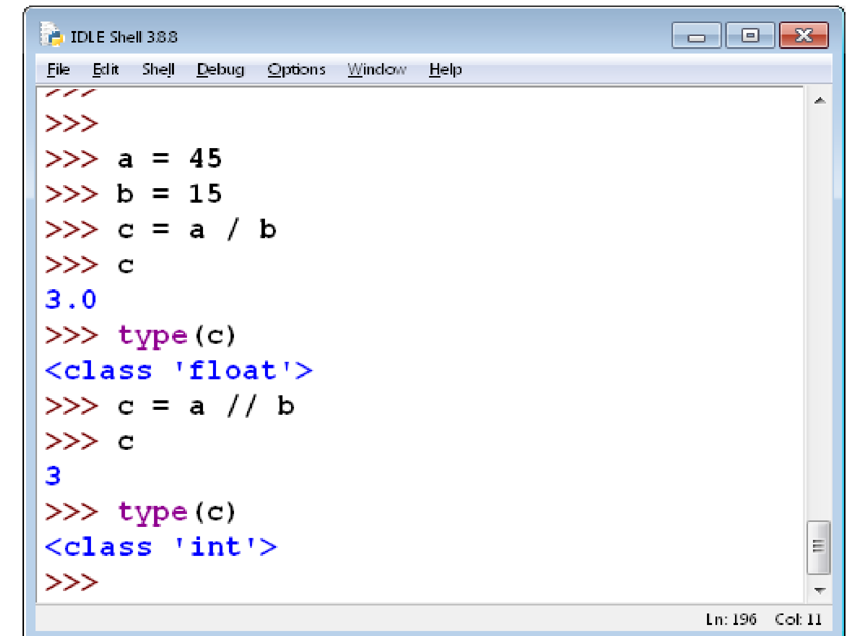
Value of a variable

- In the IDLE it is adequate to write the name of a variable to get its value.
- Moreover, the result of an operation is immediately shown.



Division

- Division / always yields a real number (e.g., float type) even if it is applied to integer numbers .
- If we want to yield an integer number it is necessary to use //



Multiple assignments

- In Python we can multiple assignments with a single command, with a collection of variables and a collection of values, separated with a comma.
- Python makes assignment on the basis of order. The first value is assigned to the first variable, the second value to the second variable, and so on.

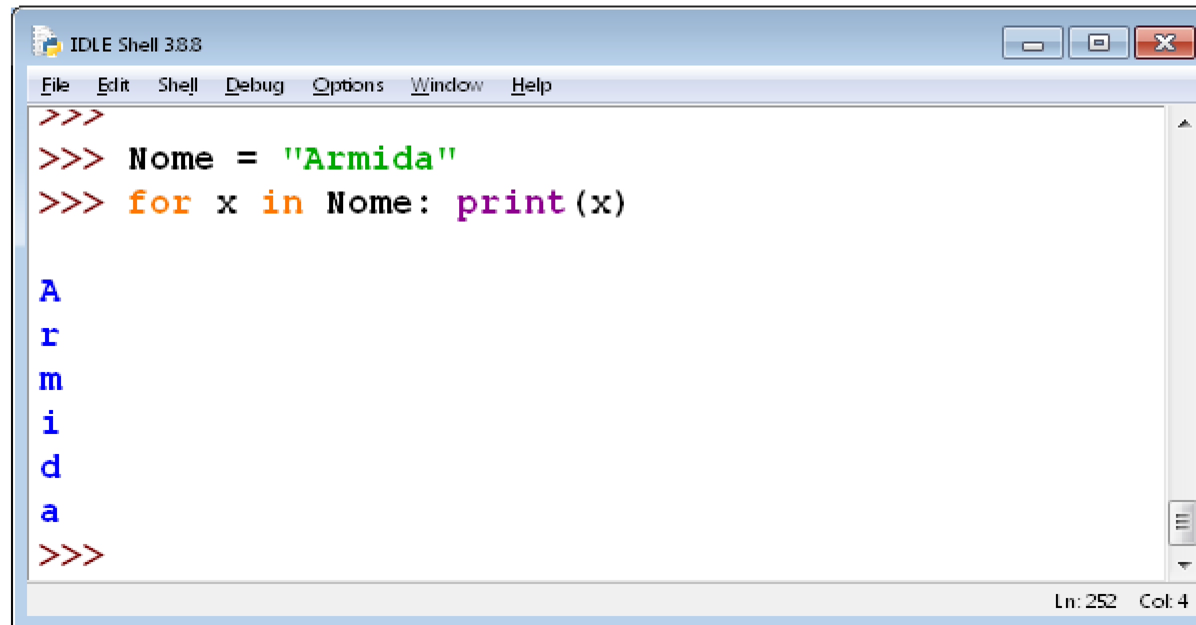
A, B, C = 12, 34, 57

- Multiple assignment also allows swapping the values of two variables:

X, Y = Y, X

for

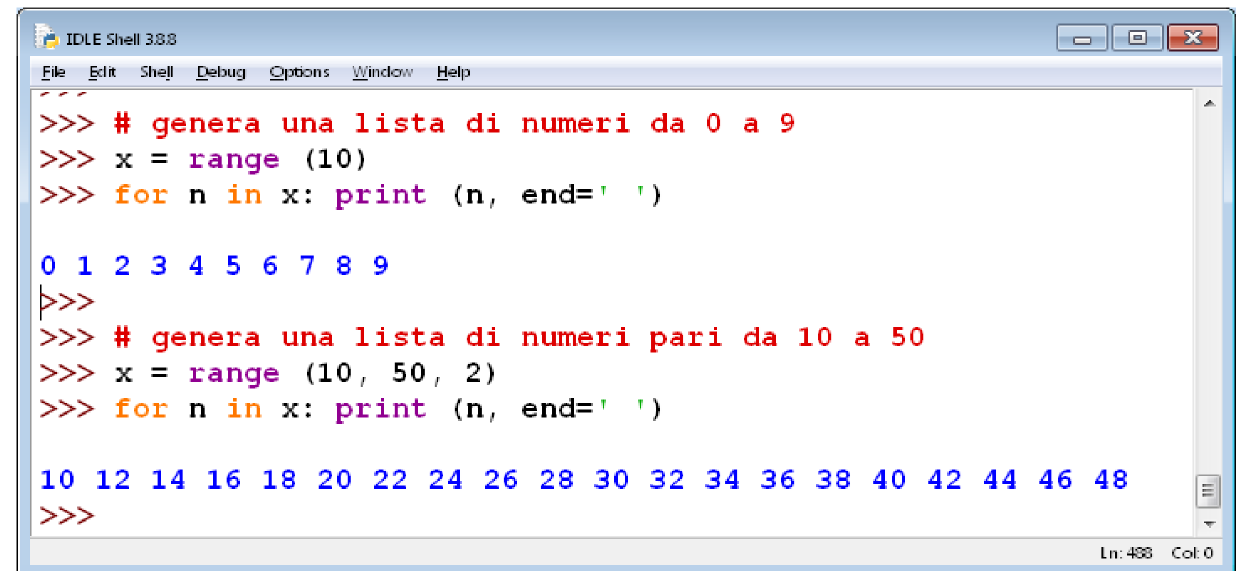
- The command **for** in Python iterates on all elements of a sequence in the order they appear in the sequence.



range

- The command **range** in Python yields a sequence of numbers from Start to **Stop** except

range (start, stop, step)

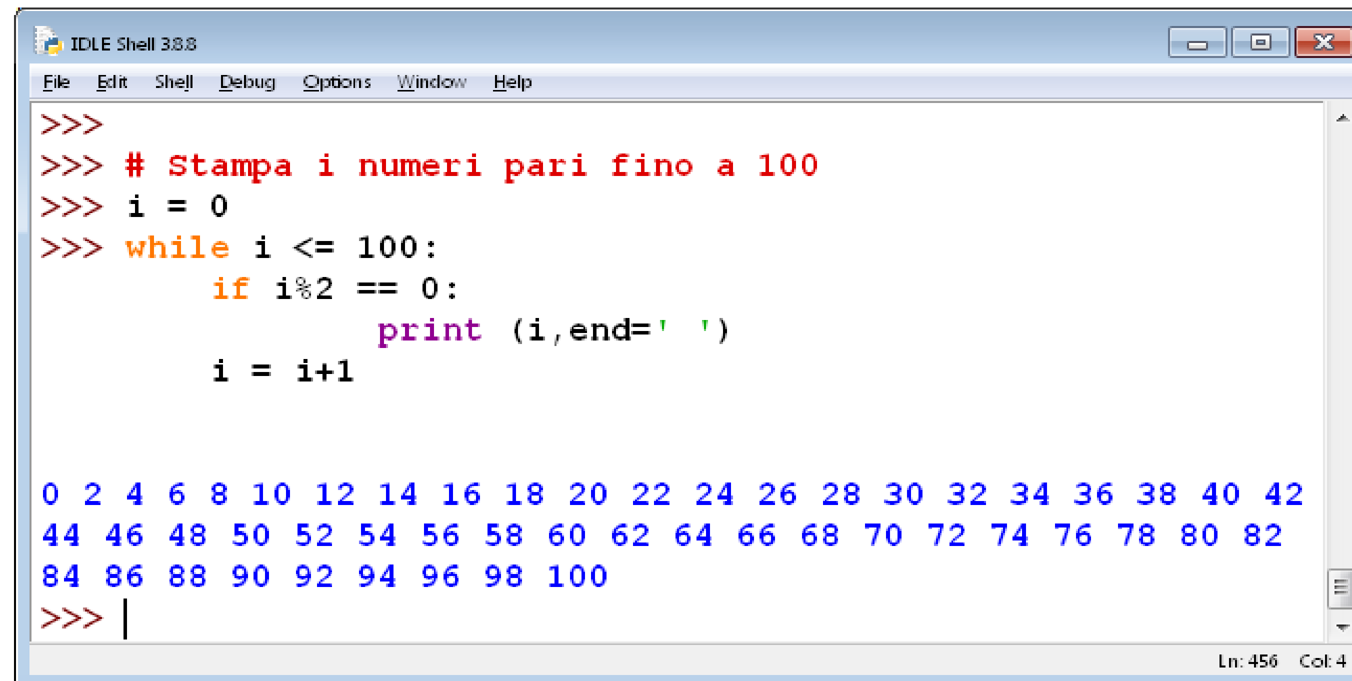


Summary: Boolean

- The type `boolean` has two values, **true** and **false**.
  - Python has two Boolean operators that combine conditions: **and** and **or**.
  - To invert a condition, use the *`not`* operator.
  - When checking for equality use the `!` operator.
  - The **and** and **or** operators are computed lazily:
    - As soon as the truth value is determined, no further conditions are evaluated.

while

- The command **while** in Python executes the commands in the next block, repeating them until the condition of while remains true.



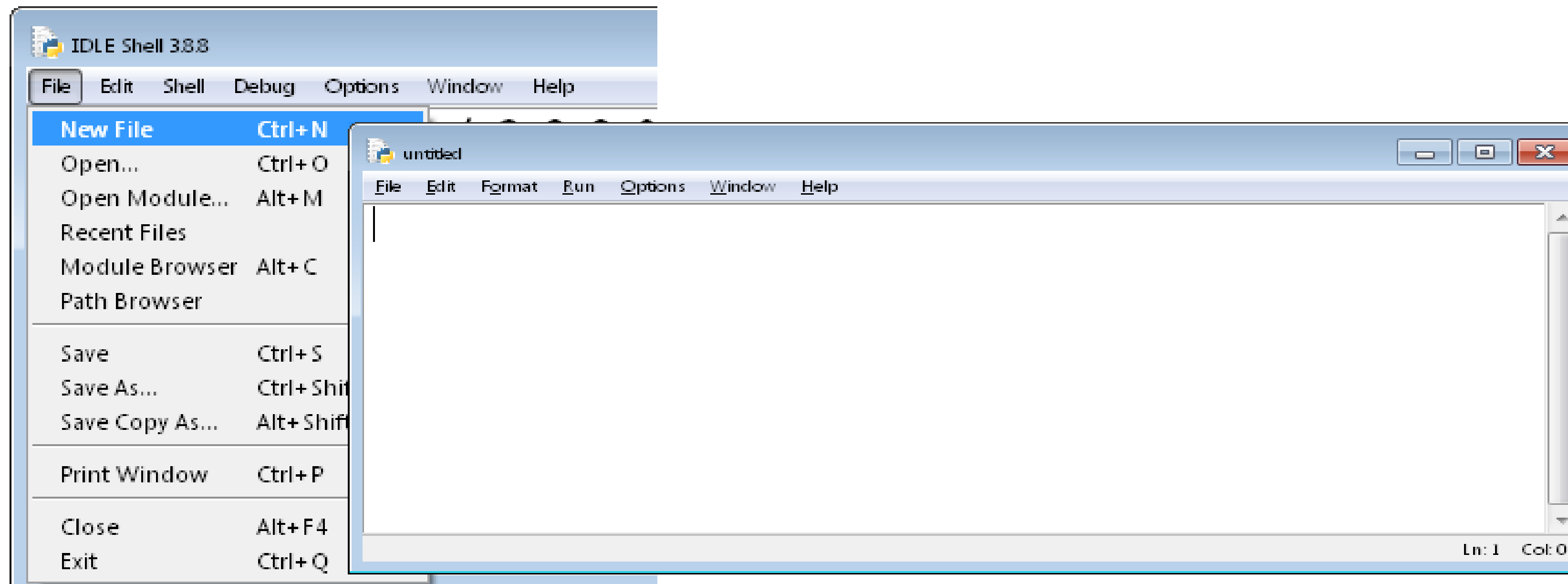
I/O (Input/Output)

- **print(param,…,sep=' ',end='\n')**
- The function **print**() print on the screen.All arguments can be converted in strings and printed on the screen, separated by **sep** e followed by **end**.

```
>>> print ("Joe", "Eva", "Ted")
Joe Eva Ted
>>> print ("Joe", "Eva", "Ted", sep="+++")
Joe+++Eva+++Ted
>>> print ("Joe", "Eva", "Ted", sep="\n")
Joe
Eva
Ted
```

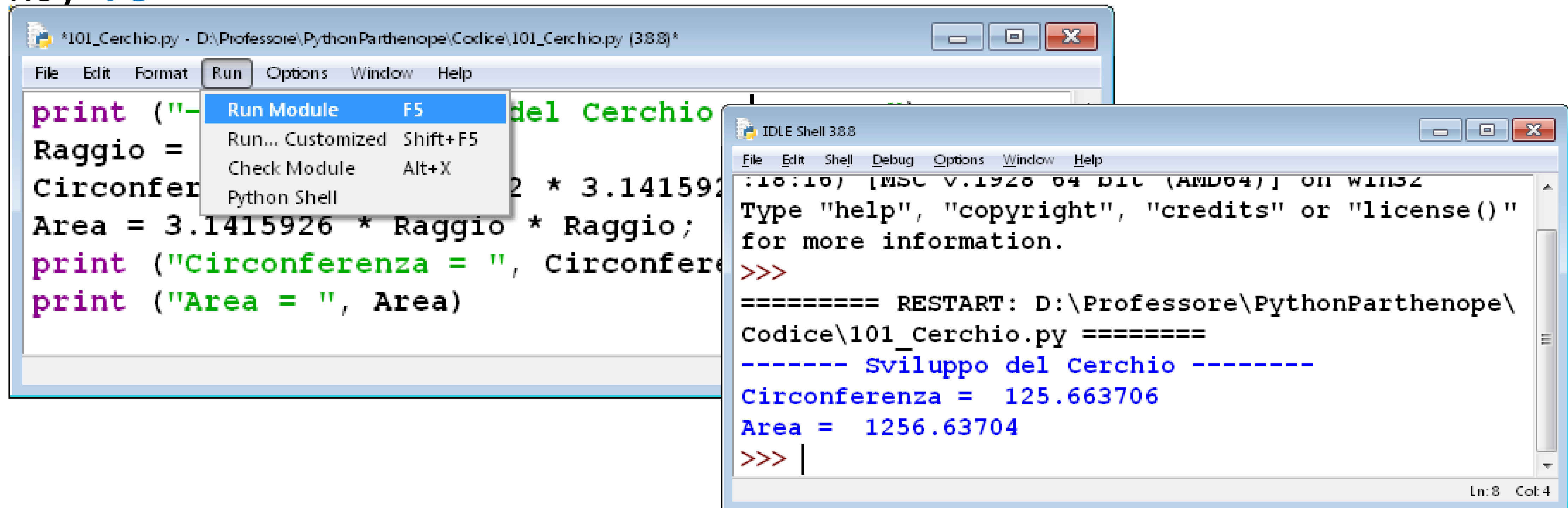
Programming in Python

- The sequence of commands to execute shall be stored into a file with extension **.py** .We can use any text editor (e.g., notepad,wordpad,…) to write the code, but Idle provides a default editor:



Execution

- To run the code, we can use the menu **Run** / Run Module or press the key **F5**



Exercise: Compute the Factorial of a Number

- Given a positive integer, compute the Factorial of a number

N! = 1 * 2 * 3 * 4 * ... * (N-1) * N

- The Factorial of 0 is 1, by definition.

*104_Fattoriale.py - D:\Professore\PythonParthenope\Codice\104_Fattor...

File Edit Format Run Options Window Help

```python
'''=====================================
 Calcolo del Fattoriale
 N! = 1 * 2 * 3 * 4 * ... * (N-1) * N
-------------------------------------'''
Max = 25
N   =  1
F   =  1
print(N,"! = ",F)

while N < Max:
    N += 1
    F *= N
    print(N,"! = ",F)
```

Ln: 8 Col: 17

IDLE Shell 3.8.8

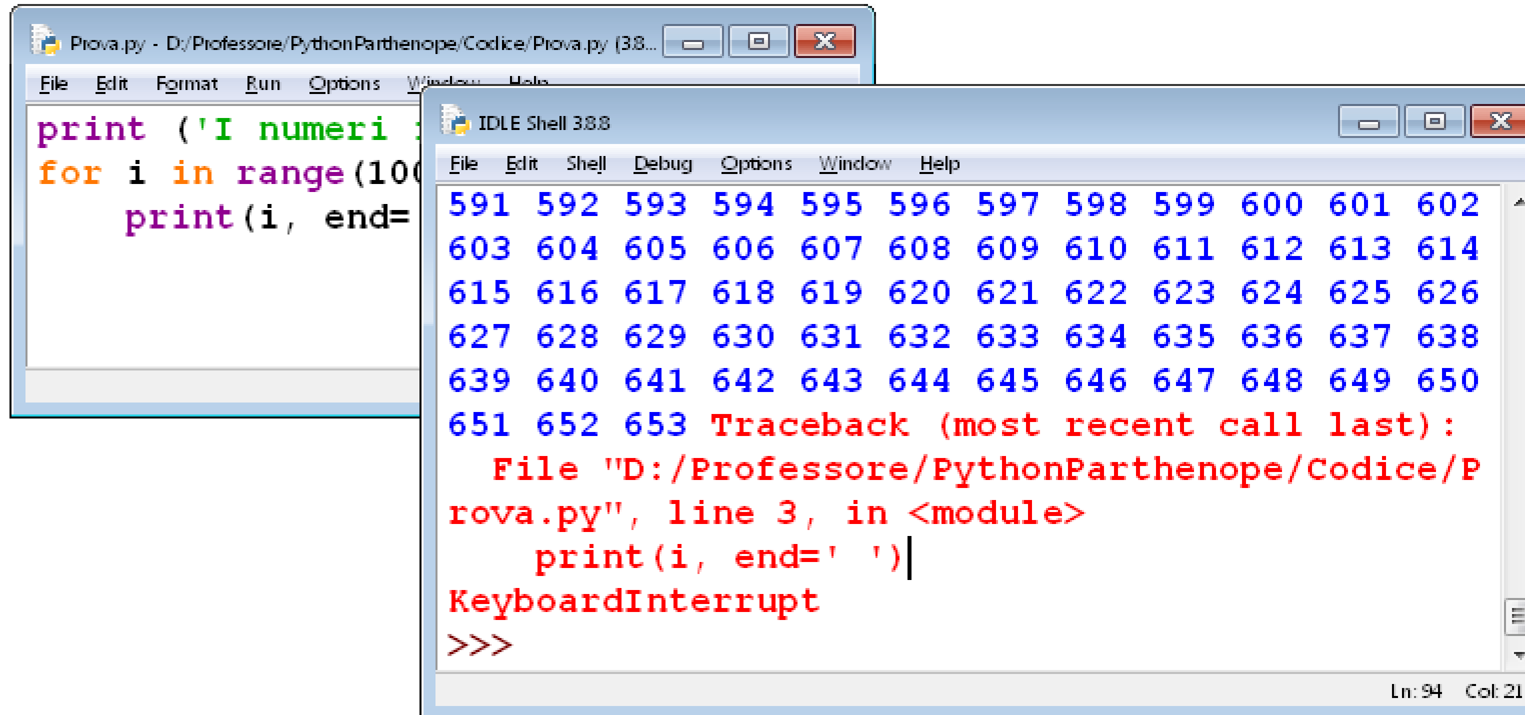
File Edit Shell Debug Options Window Help

```
1 ! =  1
2 ! =  2
3 ! =  6
4 ! =  24
5 ! =  120
6 ! =  720
7 ! =  5040
8 ! =  40320
9 ! =  362880
10 ! =  3628800
11 ! =  39916800
12 ! =  479001600
13 ! =  6227020800
14 ! =  87178291200
15 ! =  1307674368000
16 ! =  20922789888000
17 ! =  355687428096000
18 ! =  6402373705728000
19 ! =  121645100408832000
20 ! =  2432902008176640000
21 ! =  51090942171709440000
22 ! =  1124000727777607680000
23 ! =  25852016738884976640000
24 ! =  620448401733239439360000
25 ! =  15511210043330985984000000
>>>
```

Ln: 76 Col: 12

How to stop a Program

- If we want to stop a program, we can use **CTRL+C**



How to stop a Program (cont.)

- or we close the window IDLE making a click on on the top at right.

Input from the Keyboard

- If we want to require the user an input from the keyboard, we use the command: `input(prompt)`
- If the argument **prompt** is indicated, it will be written on the screen.
- The command (function) input read a line from the keyboard (i.e., any sequence of the characters ended by Return Key CR ("Invio").
- If we want to convert the sequence of character provided by input in a number we must use `int()` or `float()`

(cont.)

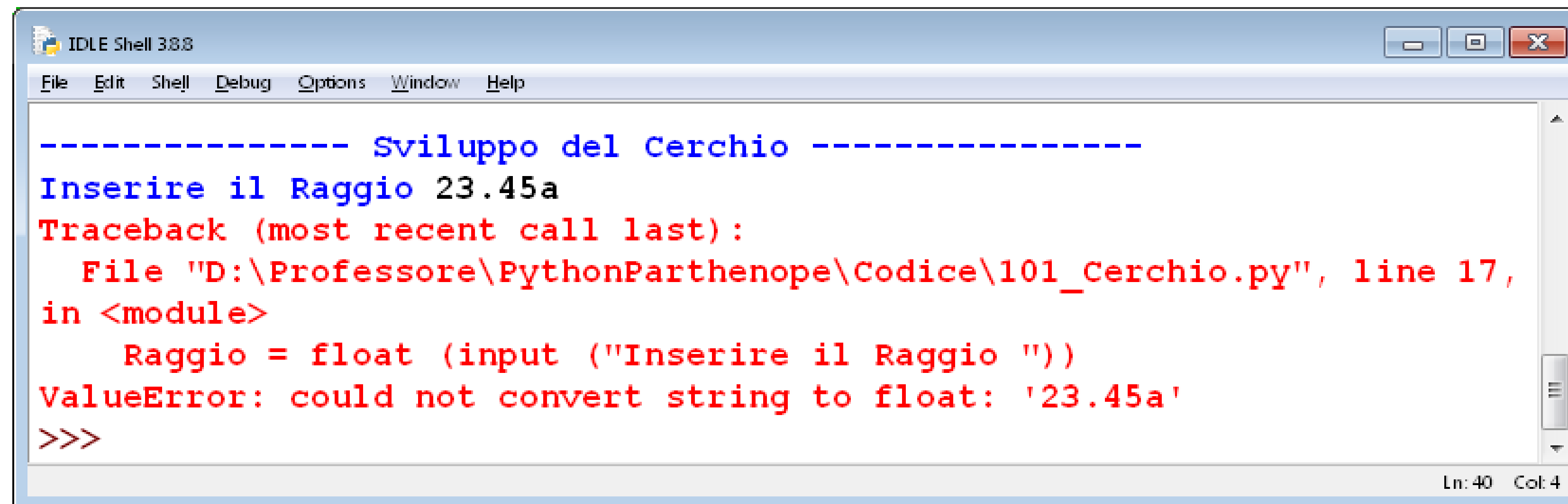
```
Radius = float (input("Insert the Radius "))
Area = Radius * Radius * math.pi
print('Area', Area, sep=' = ')
```

Program for computing area of a circle

```
from math import *      # for using  pi = 3.14159265

print ("------- Circle -------")
Radius = -1.0
while Radius < 0:
    Radius = float (input ("Insert Radius "))

Circumference = Radius * 2 * pi
Area = pi * Radius * Radius;

print ("Circonferenza =", Circonferenza)
print ("Area =", Area)

input(‘press Return ...')
```

Input Mistakes

- If we insert as input characters not allowed, the interpreter interrupts the program.



Example: Computing Fibonacci Numbers

- **Fibonacci** series denotes a series of integers where each integer is given by the sum of two previous integers except the first two that are equals to 1:

  F(1) = 1

  F(2) = 1

  F(n) = F(n-1) + F(n-2)

- First numbers of the series are 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...
- The rate **F(n) / F(n-1)** when **n** is larger and larger tends to **golden section** Tau = (1 + sqrt(5))/2

Fibonacci Program

115_Fibonacci.py - D:\Professore\PythonParthenope\Codice\115_Fibonacci.py (3.8.8)

File Edit Format Run Options Window Help

```
        1,618033 9887498 9484820 4586834 36563
----------------------------------------------
N = int(input("\nQuanti numeri vuoi ? "))
F = 1;              # F(1)
Cont = 1;
print("F( 1 )=", F)
B = F
F = 1               # F(2)
T = F / B
Cont += 1
print("F(",Cont,")=", F," T =",T)

while Cont < N:
    A = B; B = F; F = A+B
    T = F / B
    Cont += 1
    print("F(",Cont,")=", F," T =",T)

input("\nbatti INVIO ...")
```

Ln: 16 Col: 46

IDLE Shell 3.8.8

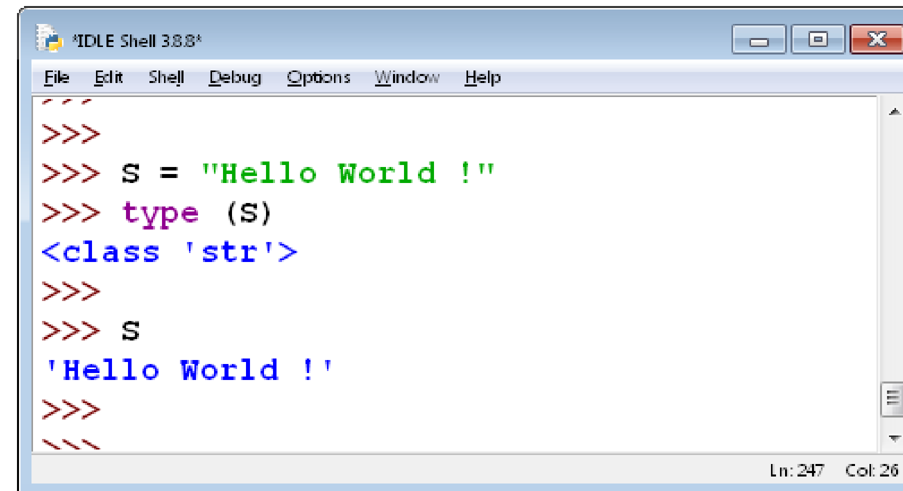
File Edit Shell Debug Options Window Help

```
Quanti numeri vuoi ? 20
F( 1 )= 1
F( 2 )= 1  T = 1.0
F( 3 )= 2  T = 2.0
F( 4 )= 3  T = 1.5
F( 5 )= 5  T = 1.6666666666666667
F( 6 )= 8  T = 1.6
F( 7 )= 13  T = 1.625
F( 8 )= 21  T = 1.6153846153846154
F( 9 )= 34  T = 1.619047619047619
F( 10 )= 55  T = 1.6176470588235294
F( 11 )= 89  T = 1.6181818181818182
F( 12 )= 144  T = 1.6179775280898876
F( 13 )= 233  T = 1.6180555555555556
F( 14 )= 377  T = 1.6180257510729614
F( 15 )= 610  T = 1.6180371352785146
F( 16 )= 987  T = 1.618032786885246
F( 17 )= 1597  T = 1.6180344478216819
F( 18 )= 2584  T = 1.6180338134001253
F( 19 )= 4181  T = 1.618034055727554
F( 20 )= 6765  T = 1.6180339631667064

batti INVIO ...
>>>
```

Ln: 114 Col: 4

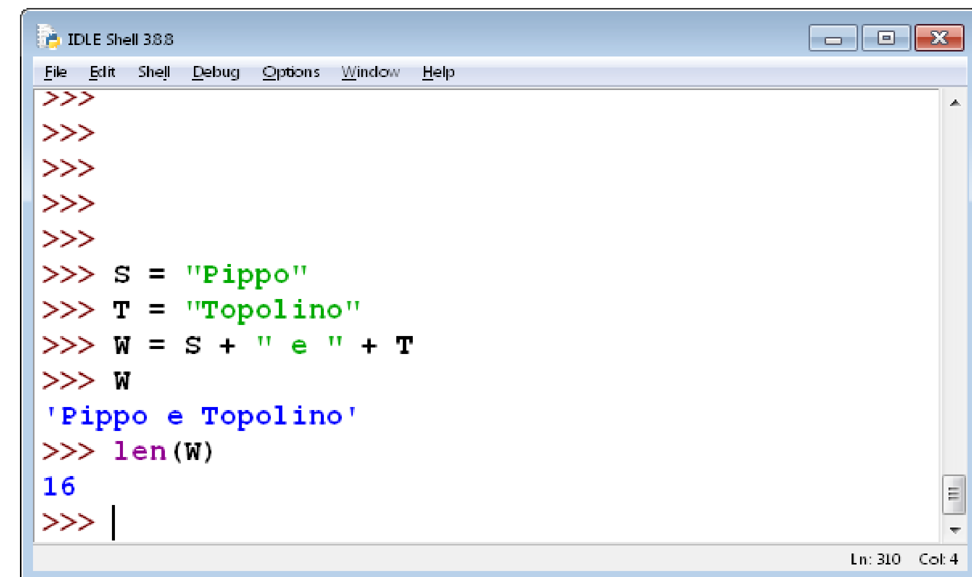
String

- A string is an ordered sequence of characters.
- There is no limit about the length of a string (i.e., there is no limit about the number of characters that can compose a string).
- To create a string it is adequate to assign to a variable a sequence of character enclosed in a tick



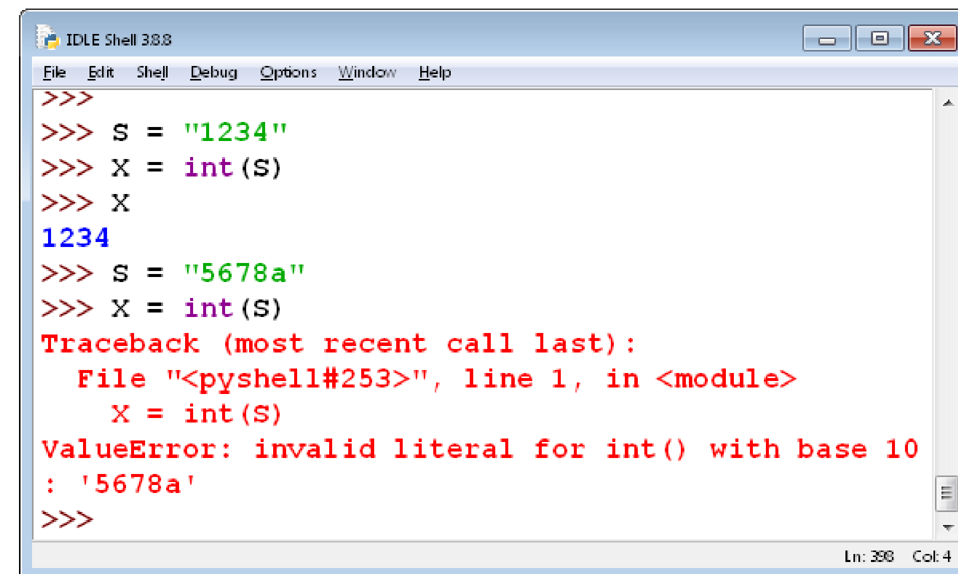
Working with strings

- The operator **+** allows the concatenation of strings.
- To know the length of a string (i.e., the number of characters that compose the string), it is necessary to use the function **len()**



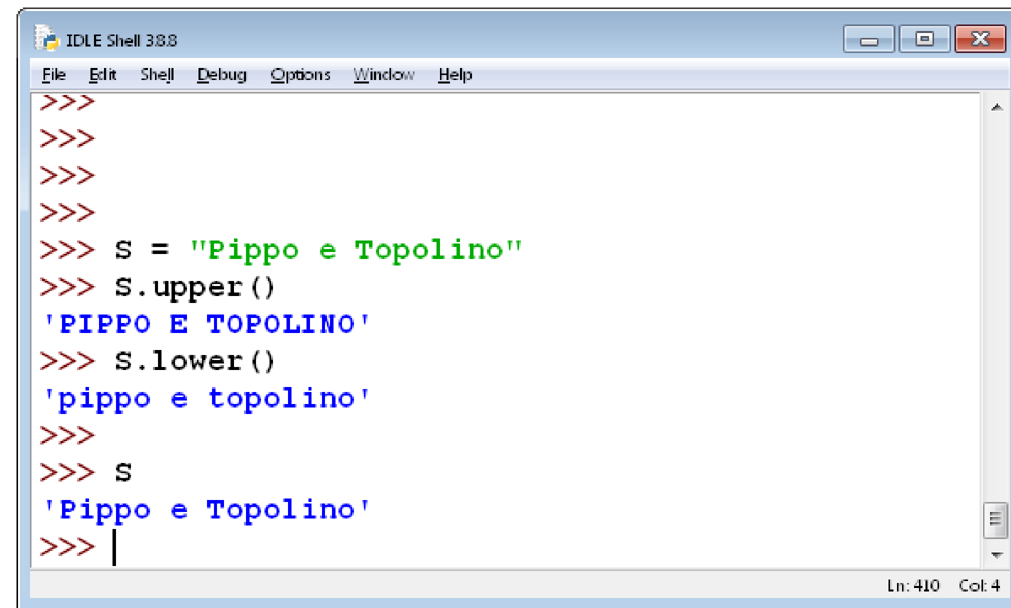
Working with strings

- To convert a string, composed of digits, it is necessary to use the command (function) **int()**
- If in the string there are no digits, the interpreter generates an error.



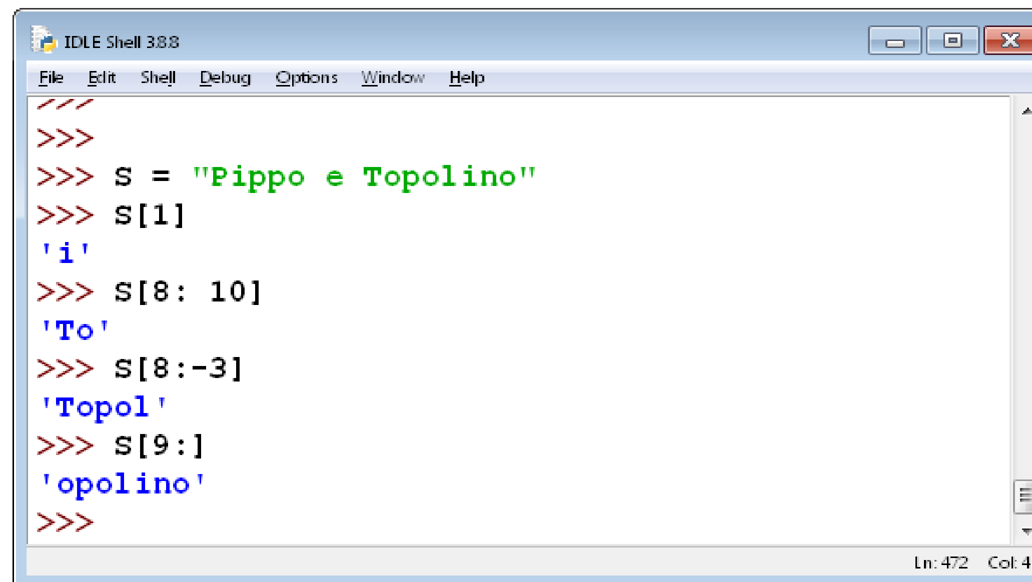
Working with strings

- If we want to convert in uppercase or lowercase, we have to use the commands upper() and lower().



Substrings

- A string is stored as a sequence of characters (the first character of the string is indexed by ‘0’) .
- We can read the string specifying the single character or the couple of indices [start : last+1]



Main Commands on strings

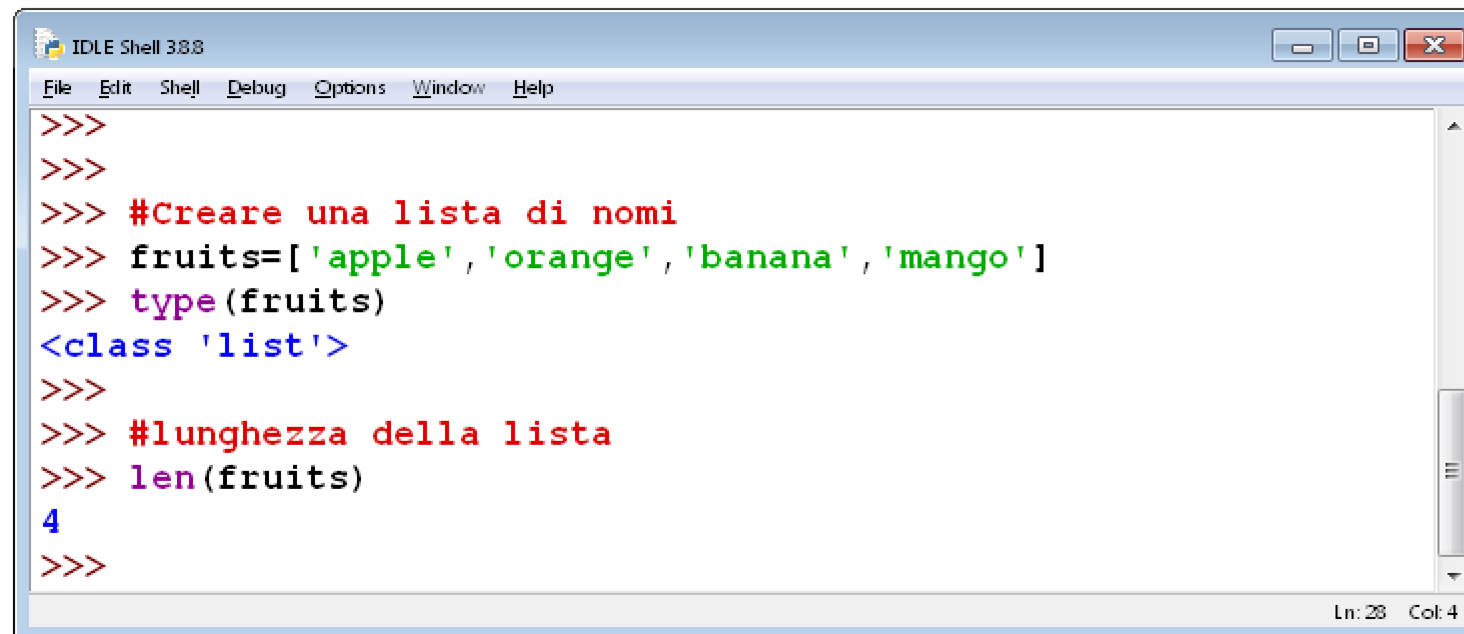
- capitalize()
  - convert in uppercase the first character of the string.
- count()
  - count the occurrences of a character in a string.
- find()
  - find a character in a string and gives the position.
- isalpha()
  - gives **true** if the string has only characters or digits.

Main Commands on strings

- isnumeric()
  - returns **true** if the string has only digits.
- replace()
  - replace in a string a given value with another (provided).
- title()
  - Convert the first charcater of each word in the string.

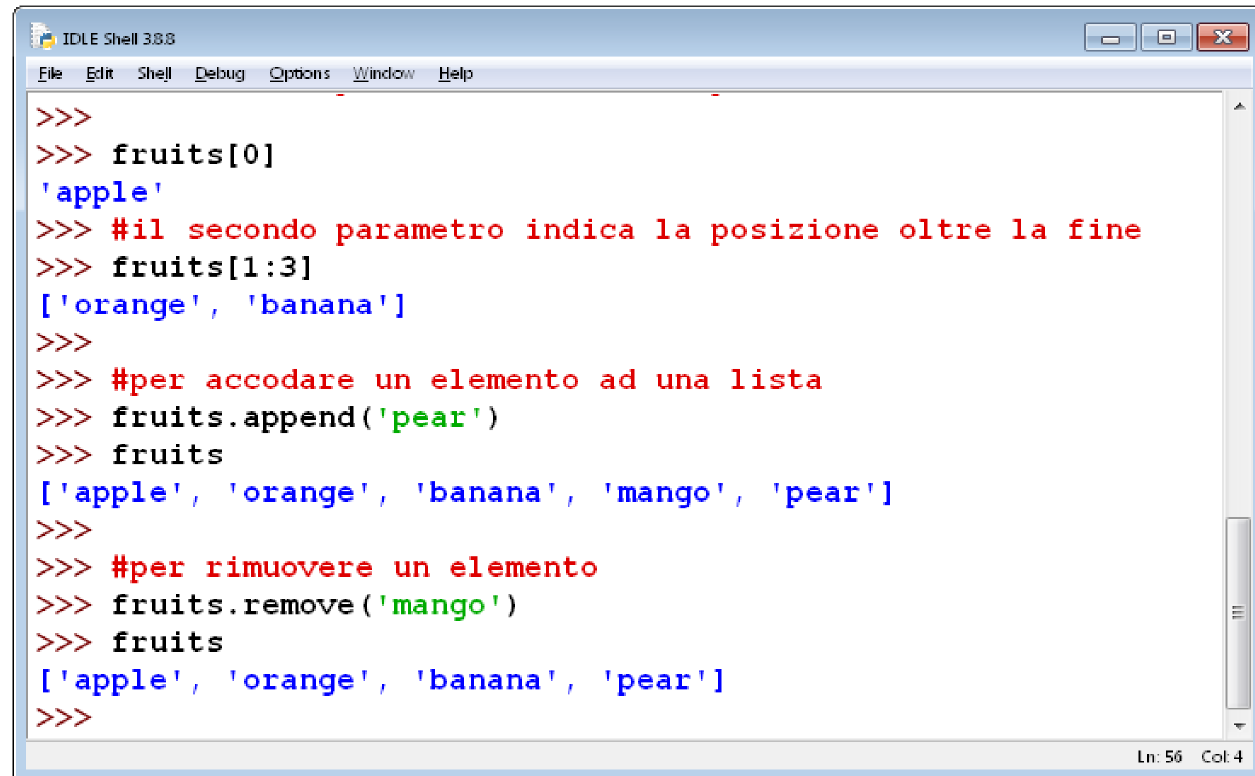
List

- A **List** is a data structure in Python for grouping data.
- A list contains data separated by **commas** and enclosed in **square brackets**.



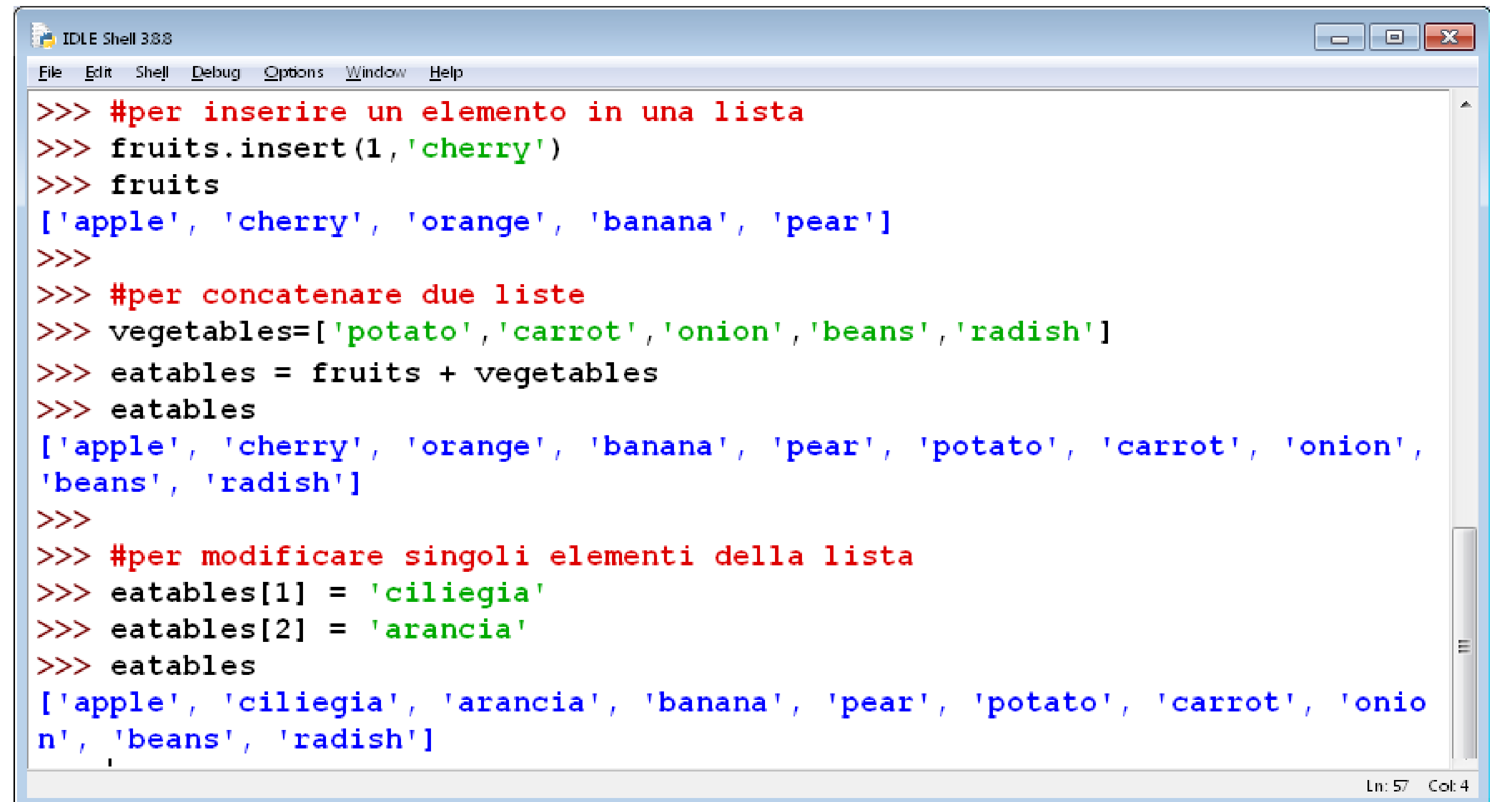
Working with the elements of a list

- To each element of list a position (or ***index***) that must be used to manipulate the element.



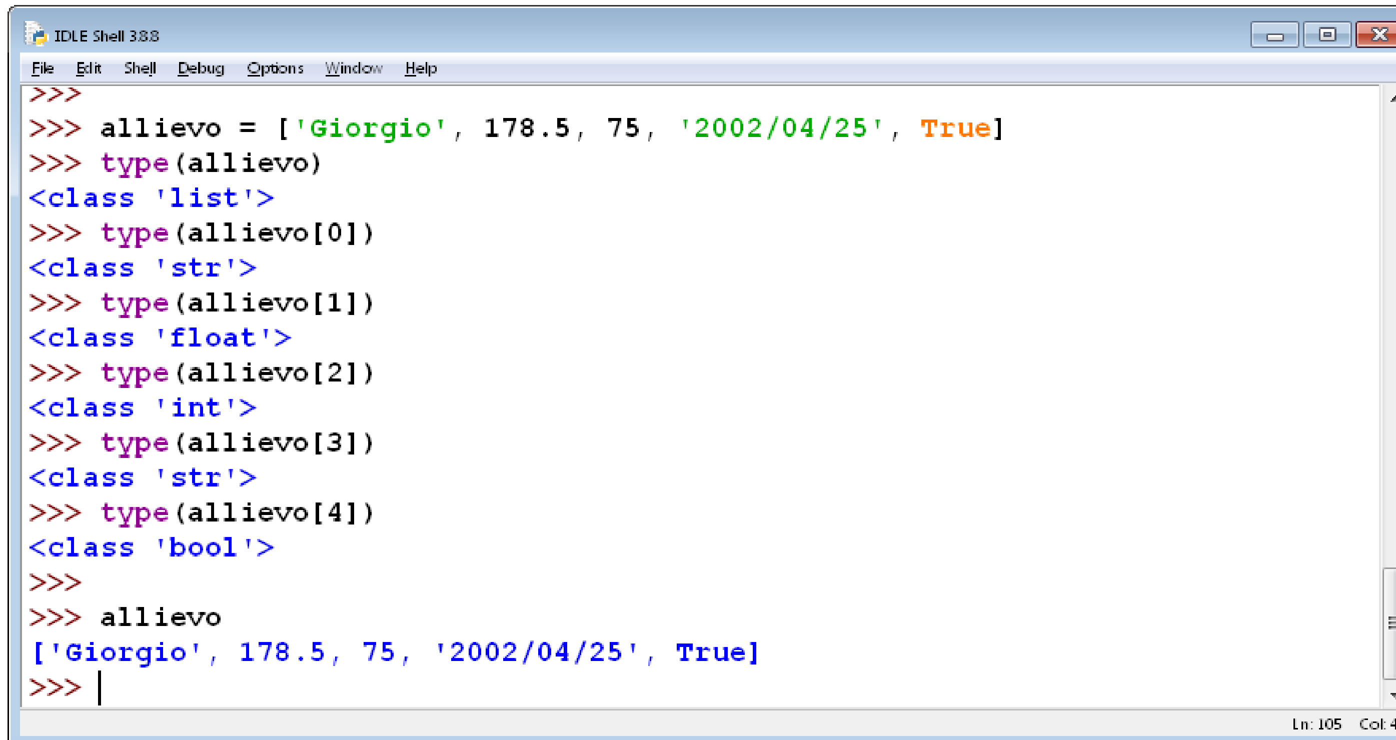
List can be modified

- In Python we can modify a list. For instance, we can add, change, remove elements in a list.



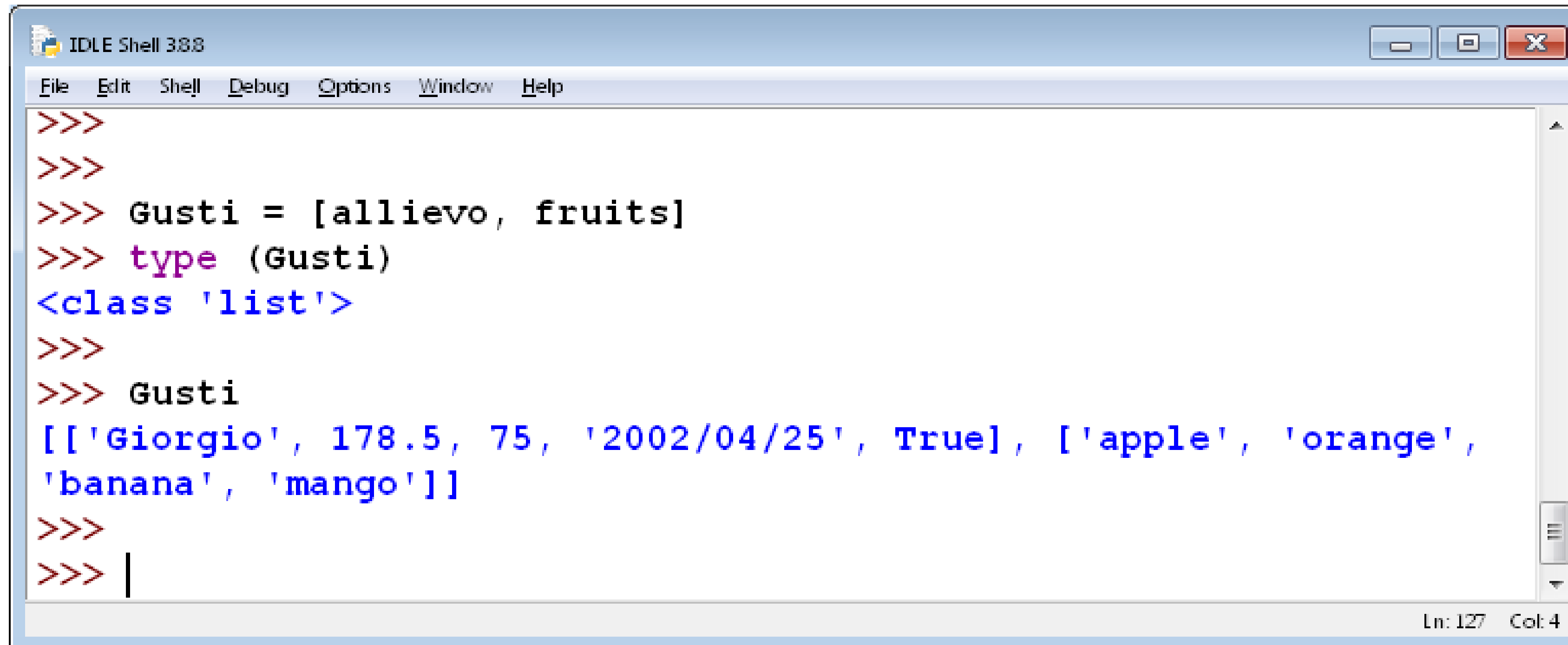
Different types in List

- A list can contain data of different types.



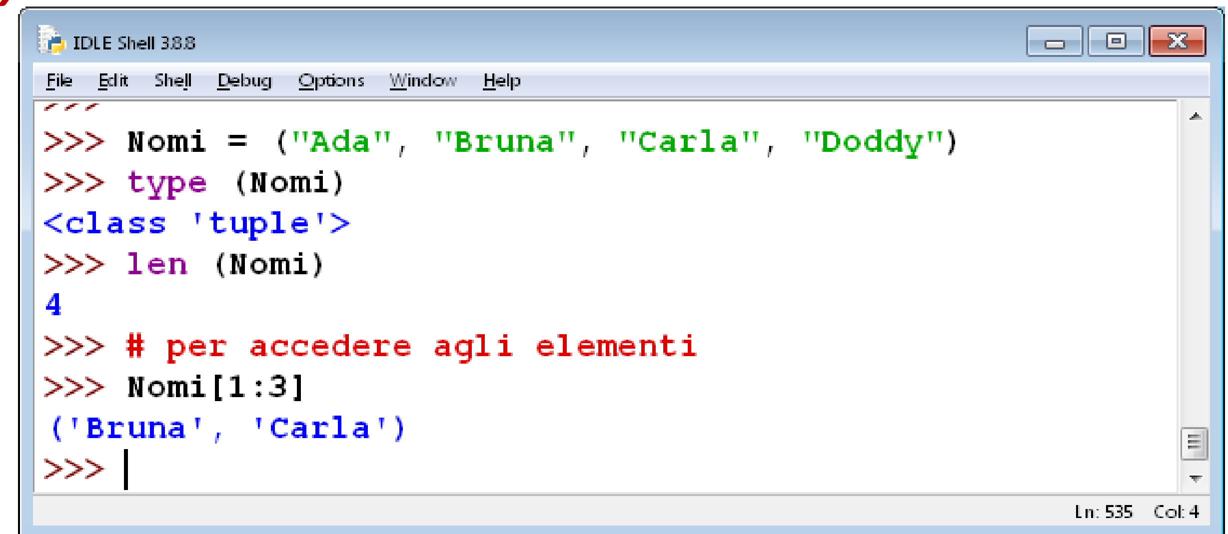
Nested Lists

- A List can be nested, that is a list can be an element of another list.



Tuple

- A **tuple** is a sequence of data, similar to a list.
- A tuple consists of a sequence of data, separated by **commas** and enclosed in **round brackets**. Elements of tuples **cannot be changed**.
- Tuples can be viewed as *read-only lists*.



Programs

- Most of programs are transient, i.e., when they are executed, they produce results , and when program finish, their data vanish. If you execute again a program, it restarts from the beginning.
- The simpler way to save the data of programs consists in writing (and then reading) them on a file.

Working with files

- If we want to access a file, we will ask to the Operating System (the program that manages the computer, e.g., Windows 10, Linux, MAC-OSF) to create a communication channel, declaring the name of the file and the opening modality.
- The communication channel is provided by the command **open()**:
  ***ChannelName* = open (*FileName*, *Modality*)**
- After the opening, we can read and/or write on the file. After having ended all operations on the file, we can close the file using the command **close()**, even if the channel is closed when the program ends.

File Opening Modalities

- ‘r’
  - Open the file in reading, If the file does not exist, opening file fails.
- ‘w’
  - Open an empty file for writing. If the file exists, its content is deleted.
- ‘a’
  - Open the file at the end of file in writing. If the file exists, its content is not deleted. If the file does not exist, the file is created.
- ‘x’
  - Creates an empty file for writing. If the file exists, the command returns an error.

File Opening Modalities (cont.)

- ‘r+’
  - Open an existing file, allowing reading and writing.
- ‘w+’
  - Open an empty file, allowing reading and writing. If the file exists, its content is deleted.
- ‘a+’
  - Open a file at the end of file for reading and writing. If the file does not exist, the file is created.

Writing on a file

- After the opening in modality 'w', 'a' or 'x' write() writes data in the file and return the overall number of characters that have been written.

```
NomeFile = ''
while len(NomeFile) < 1:
   NomeFile = input("Insert the name of the file you want to create:  ")

Out_file = open(NomeFile,"w")
Out_file.write("Hello"\n")
Out_file.close()
```

Format

- The argument of write() must be a **string**
- To insert as arguments values that are not string (e.g., numbers or real number) it is necessary to converts values in string, by the command (function) **str()**

```
Out_file = open(NomeFile,"w")

for n in range(1,10):
    Out_file.write('Side = ' + str(n) + " area = " + str(n*n) + '\n')

Out_file.close()
```

```
Lato = 1    area = 1
Lato = 2    area = 4
Lato = 3    area = 9
Lato = 4    area = 16
Lato = 5    area = 25
Lato = 6    area = 36
Lato = 7    area = 49
Lato = 8    area = 64
Lato = 9    area = 81
```

Writing in a file with print

```
print(param,…,sep=' ',end='\n',file=sys.stdout)
```

- The command **print**() writes the value into ***file*** (by default screen).
- All arguments are converted in strings and written into file, separated from **sep** and followed by **end**.
- If no parameters are provided, **print()** will write only solo the default value of **end** namely '**\n**'

Reading from a file

- After the file opening in modality '**r**', the command **read**() extract ALL characters from the file, until achieving a **maximum** number provided as a parameter of command read

Reading from a file (cont.)

```
FileName = ''
while len(FileName) < 1:
    NameFile = input("Name of the file to read? ")
In_file = open(FileName,"r")      #file opening
Line = In_file.read(10); print (Line)     #read first 10 characters from the file
Text = In_file.read(); print (Line)  #read the rest of the file
In_file.close()                   #close the file
```

Reading by rows

- The command **readline**() read characters from a file until achieving a CR ("invio"), and returns the result under the form of a string.

Reading by rows (cont.)

```
FileName = ''
while len(FileName) < 1:
    FileName = input("Name of the file to read? ")

In_file = open(FileName,"r")      #file opening
row = In_file.readline()
print (row)
In_file.close()                   #close the file
```

Reading all rows of a file

- The command **for** can be used for reading a file row by row; at each row the separator '**\n**' is read. To avoid that **print** add another "CR" (invio), remove automatic "CR".

Reading all rows of a file (cont.)

```
In_file = open(FileName,"r")      #file opening
Cont = 0                          #read a row at a time
for row in In_file:
    print (row, end=“)            #remove a CR
    Cont = Cont + 1

print ("Ho letto",Cont,"righi")
In_file.close()                    #close the file
```

Moving through the file

- In order to know in what location we are in the file (namely, the number of characters from the beginning of the file), it can use the command **tell**()

```
CurrentPos = File.tell()
```

(cont.)

- To move in the file, we can use the command **seek**(): SEEK_SET e SEEK_END are defined in the module **os** and denotes the beginning and the end of the file.

  - import os
  - File.seek(10, os.SEEK_SET) #move the position forward
  - File.seek(0, os.SEEK_END) #go to the end of the file
  - File.seek(-2, os.SEEK_END) #go to 2 characters before the end of the file

File Opening: Errors

- When we try to open a file, we can do some errors.
- In Writing
  - We do not have the write permission
  - File is blocked by another program
  - File exists, but it is "read only"
  - The device where it is a file is "read only" (e.g., a CD)

File Opening: Errors (cont.)

- In Reading
  - File does not exist
  - We do not have "reading permission"
  - File is blocked by another program

Functions in Python

- Python interpreter has some functions that are always available.
- The ***Incorporate*** (o ***built-in***) functions of Python are more than 60.
- For more details go to:

  https://docs.python.org/3/library/functions.html

- In IDLE, after having typed the name of the function and the round bracket "**(**", appears a box with a suggestion (***call tip***) that helps to understand what the function does and how it can be used.

Built-in Functions

abs()	dict()	id()	object()	str()
all()	dir()	input()	open()	sum()
any()	divmod()	int()	ord()	super()
ascii()	eval()	iter()	pow()	tuple()
bin()	exec()	len()	print()	type()
bool()	float()	list()	range()	vars()
bytearray()	format()	locals()	round()	__import__()
bytes()	globals()	map()	set()	
compile()	help()	max()	slice()	
complex()	hex()	min()	sorted()	

Some Functions

abs()	Return the absolute value of a number
float()	Transform the argument in a real number
int()	Transform the argument in an integer
type()	Return the type of a datum
max()	Return the maximum of a list of values
min()	Return the minimum of a list of values
range()	Return a list of n values from 0 to n-1

Standard Library of Python

- In addition to built-in function, the installation provides a collection of functions, grouped in **Modules**
- To use the functions of a module, it must **import** it.
- For instance, to use **sqrt()** of extern library **math**
-

```
import math
Radix = math.sqrt(Arg)
X = math.ceil(Radix)
Y = math.floor(Radix)
```

Modules

- It can import from a module, all what that is in the module, otherwise it can import only what it is required in a given moment.

```
from math import *
```

or

```
from math import sqrt, ceil, floor
```

- In the last case the **name of module** is omitted:

```
Rad = sqrt(Arg)
X = ceil(Rad)
Y = floor(Rad)
```

Pseudo-random numbers

The **random** library has some functions for generating random numbers:

```
import random

# randint(a,b) returns an integer in [a,b]
I = random.randint(-10, 10)

# random() returns a float in [0, 1)
F = random.random()
```

Random seed

- The function seed() allows to initialize the generator of random numbers, with a fixed value.
- With seed() we can fix the initial value of the random series and it can obtain the same sequence of random values.

119_random_seed.py - D:/Professore/PythonParthenope/Codice/119_random...

```
from random import *
for x in range(10):
    N = randint(100, 999)
    print (N, end=' ')
print ()
for x in range(10):
    N = randint(100, 999)
    print (N, end=' ')
```

IDLE Shell 3.8.8

```
>>>
= RESTART: D:/Professore/PythonParthenope/Codice
/119_random_seed.py
267 561 272 776 902 119 178 674 762 849
465 186 531 213 509 527 771 210 424 733
>>>
```

*119_random_seed.py - D:/Professore/PythonParthenope/Codice/119_rando...

```
from random import *
seed(123)
for x in range(10):
    N = randint(100, 999)
    print (N, end=' ')
print ()
seed(123)
for x in range(10):
    N = randint(100, 999)
    print (N, end=' ')
```

IDLE Shell 3.8.8

```
>>>
= RESTART: D:/Professore/PythonParthenope/Codice
/119_random_seed.py
153 374 189 887 517 372 210 958 995 139
153 374 189 887 517 372 210 958 995 139
>>>
```

Functions

- A **function** in Python is a block of code that starts with the keyword **def** followed by the name of the functions and parameters (i.e., data that the function receives as input) in round brackets.
- The block of code in a function begins after **colons** che are immediately after the brackets that enclose the parameters.All the body of function MUST be indented.
- The first instruction of a function can be optionally a comment or **docstring**. At the end of function, the function can return one or more values.
- To conclude a function, insert an empty row.

Definition of a function

- The function can be defined before it can be used.

```
def FunctionName (Par1, Par2, ...):
    ''' string of comment '''
    block of instructions of function
    return Result
```

Can parameters be modified ?

- In Python, a function **CANNOT** modify the value of a variable that is passed as argument.

```
def Modify (A):
    A = 456

B = 123
Modify (B)
print (B)
```

generates as result

123

docstring

- After the definition of a function, there is the string of documentation (**docstring**), a comment that is useful to describe the task of the function;
- IDLE provides a suggestion about the use of a function during writing, showing its docstring as **call tip.**

return

- When a function ends without the instruction return, Python interpreter returns the value **None**. The same happens when there is a return without a value.

```
def F1 ():
    X = 1234

def F2 ():
    return

print (F1(), F2())
None None
```

Multiple return

- Python functions can return more than a value simoultaneously.

```
def Division (N, D):
        Q = N // D          # integer division
        R = N % D           # rest of division
        return Q, R

A, B = Division (33, 5)
print (‘quotient',A,'rest',B)
```

Default parameters

- It can define functions with a variable number of arguments, specifying a default value for one or more arguments.

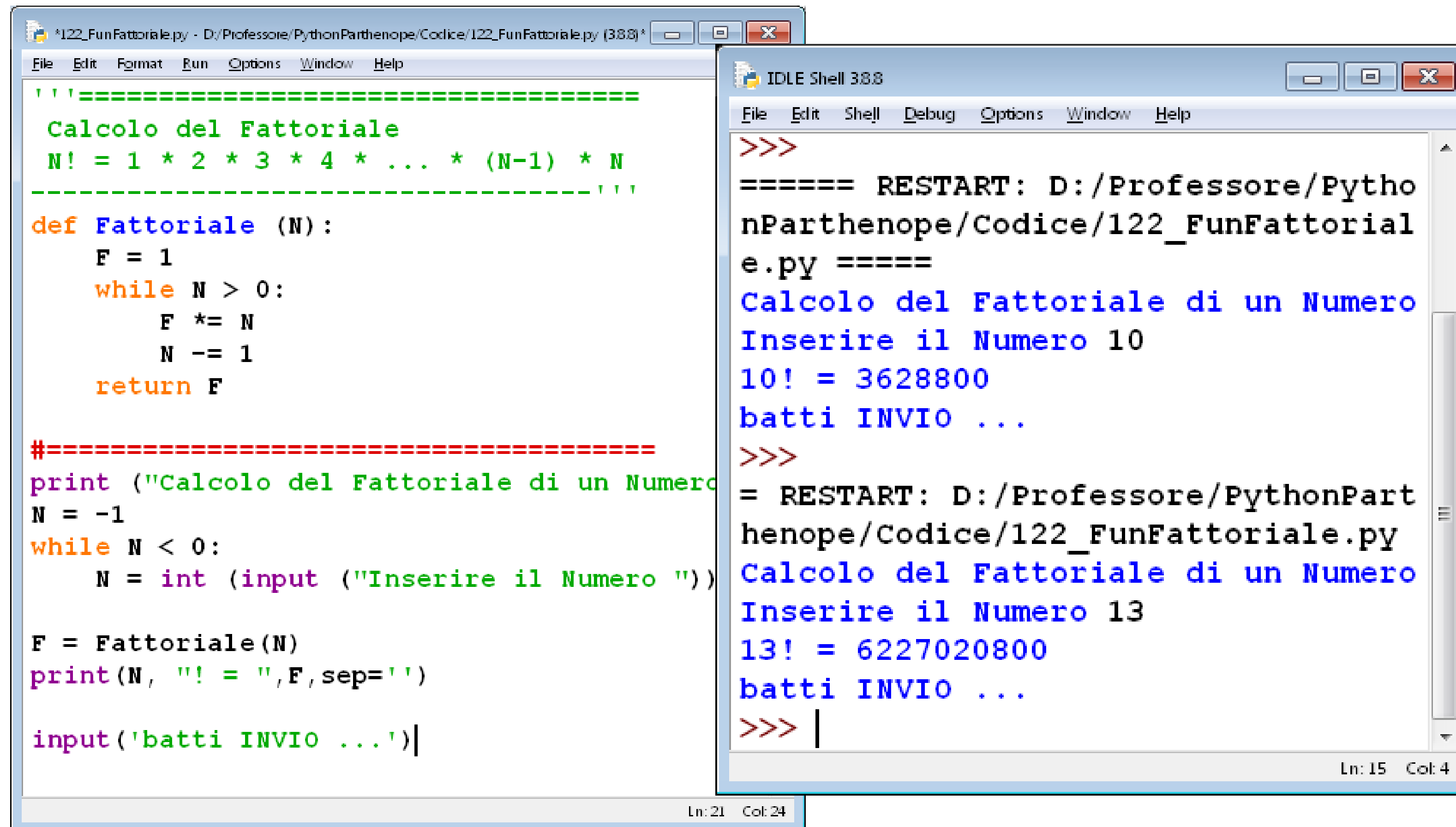
```
def Funzione(Par1,Par2=valore,Par3=valore):
  .........
```

- Hence, the function can be called with less arguments

```
Function(10, 22, 456)
Function(40, 122)
Function(13)
```

- Missing parameters will be replaced with default values, fixed in the definition of the function.

An Example: The Factorial of a Number



Personalized Modules

- If our **function** must be used in different programs, we can save in a separed file call **Module**.
- In Python modules are file **.py**, to write a module, it is adequate to write the code of functions in a file.py
- Create file **Algebra.py** with:

```
def Equazione1 (A, B):
    '''solve equation of a first degree'''
    if A == 0: X = 0  #impossible
    else: X = -B / A
    return X
```

Importing a module

- To use functions of Module, we must import it

```
import Algebra
```

- To refer to a member of the module, it must prefix the name of module at each member.

```
Algebra.Equazione1(...)
```

Importing a module (cont.)

- Or we can import directly functions

```
from Algebra import *
```

Or even

```
from Algebra import Equazione1
```

In this case we do not prefix the name of the module.

Use of a module

- '''Solution of an equation of first degree'''

```
import Algebra
A = 123.4
B = 567.8
X = Algebra.Equazione1(A,B)
print ("Solution =", X)
```

Use of a module

- *or*

```
from Algebra import *
A = 123.4
B = 567.8
X = Equazione1(A,B)
print ("Solution =", X)
```

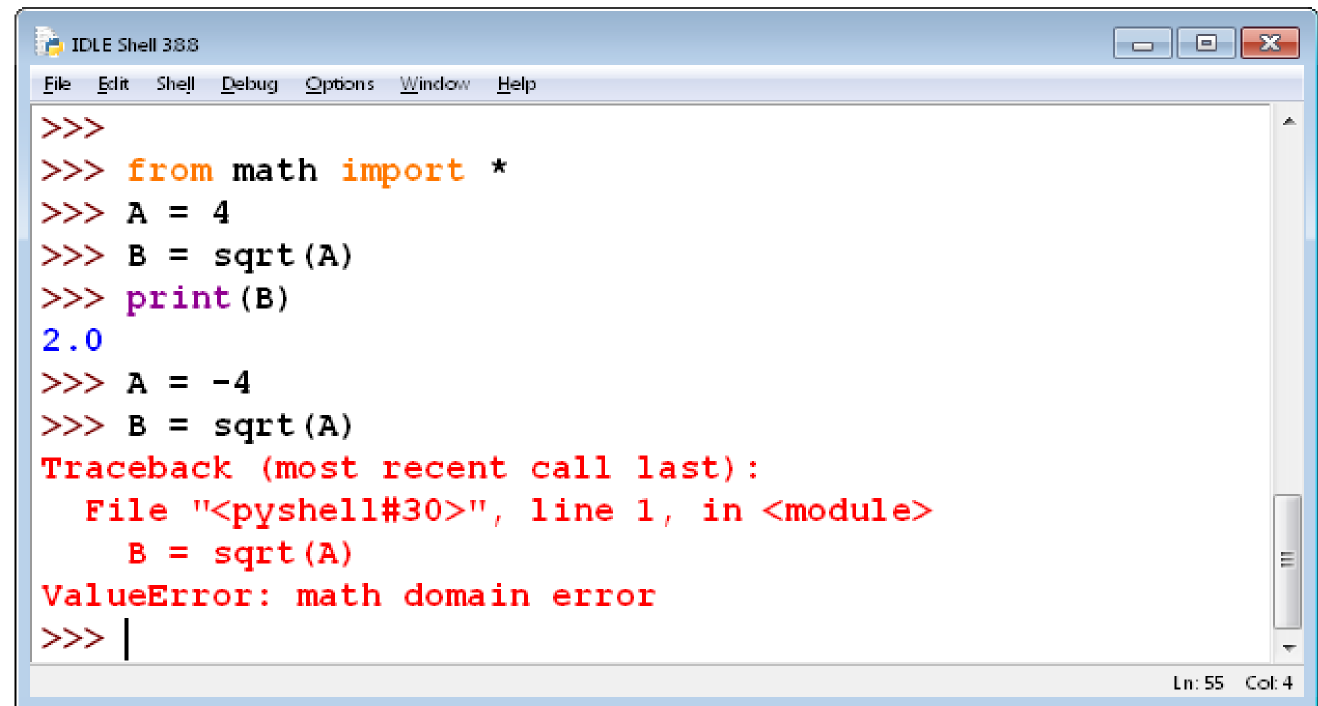
Ambiguity in names

- If use two different modules that che contain functions with same name, we must **import only the name of module**, specifying then the name of module.We suppose that exists **Equazione()** either in the module **Algebra** or in **Geometria**

```
import Algebra
import Geometria
  A = 123.4
  B = 567.8
  print ("Soluzione =",  Algebra.Equazione(A,B)
  print ("Soluzione =", Geometria.Equazione(A,B)
```

Square Root float

- In Python, the function **sqrt()** of the module **math** extracts the square root of the argument (even it is integer), but it generates an error if the argument is negative.



Looking for a module

- Firstly, the file module.py is searched in the current directory, where the program is executed.
- Then, the system variable PATH is used that provides the list of the directories when looking for the files. To show the content of variable, digit PATH