Python

Loriano Storchi

loriano@storchi.org

http::/www.storchi.org/

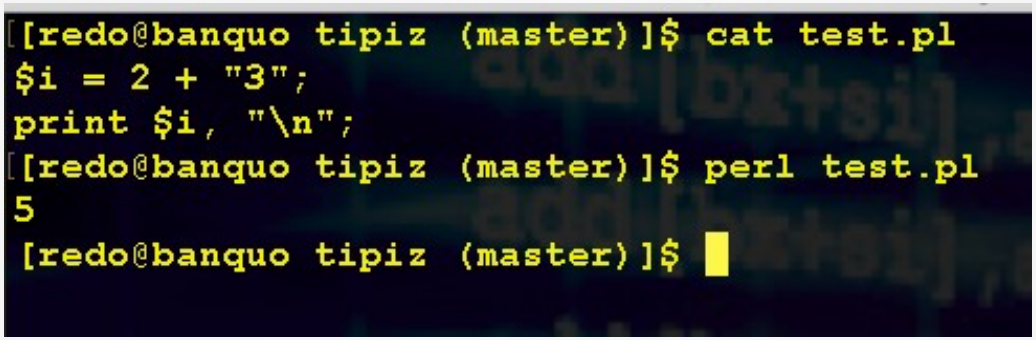
Programming Laguanges

- We have seen that the programming languages may be classified as (not only but...):
  - Declarative
    - Logic
    - Functional
  - Imperative
    - Procedural
    - Object-Oriented

- Python is imperative and both procedural and object-oriented

Programming languages

- **We may classify programming languages also as dynamic typing or static typing and strong and weak e typing**
- Weak and dynamic typing, example using Perl:



- *Python dynamic and strong typing*

the Python interpreter

- Lots of material / documentation available (in fact, this introduction is strongly based on:

  http://tdc-www.harvard.edu/Python.pdf
- Python can be found ready for use on both Linux and Mac OS X. For Windows you can easily find the binaries at the following URL: http://python.org/
- Many modules (libraries) available, in our case some useful / interesting: numpy, matplotlib and pystat
- python 2.x vs python 3.x, we will use python 3.x version 2.x has been now dismissed.

the Python interpreter

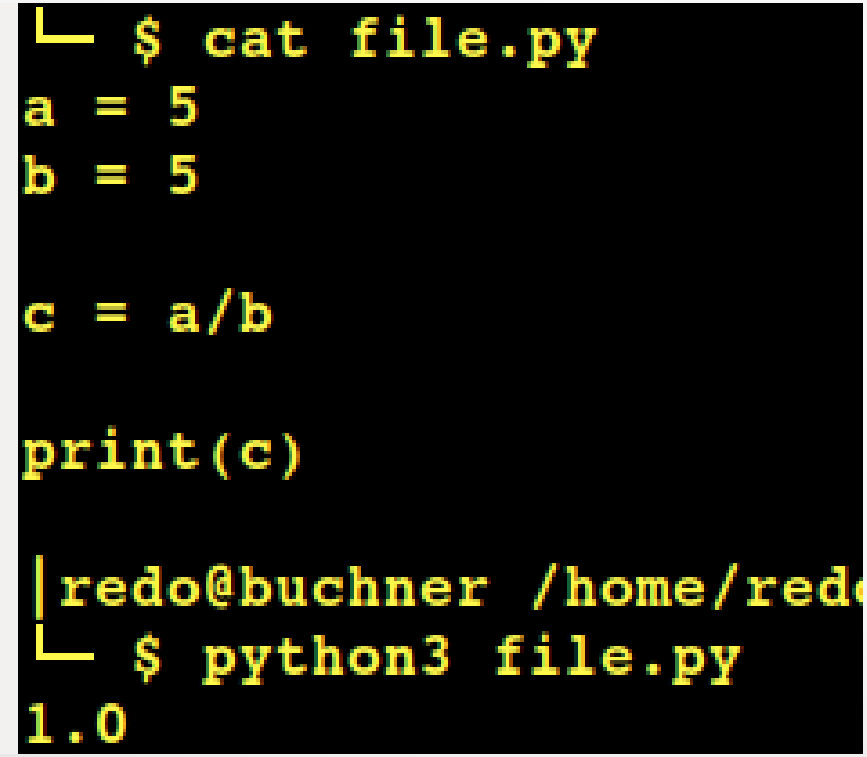
```
[redo@virtualinux ~]$ python
Python 2.7.10 (default, Jun 20 2016, 14:45:40)
[GCC 5.3.1 20160406 (Red Hat 5.3.1-6)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> (4*5)/3
6
>>> exit()
[redo@virtualinux ~]$
```

To exit press CTRL-D or type exit()

the Python interpreter

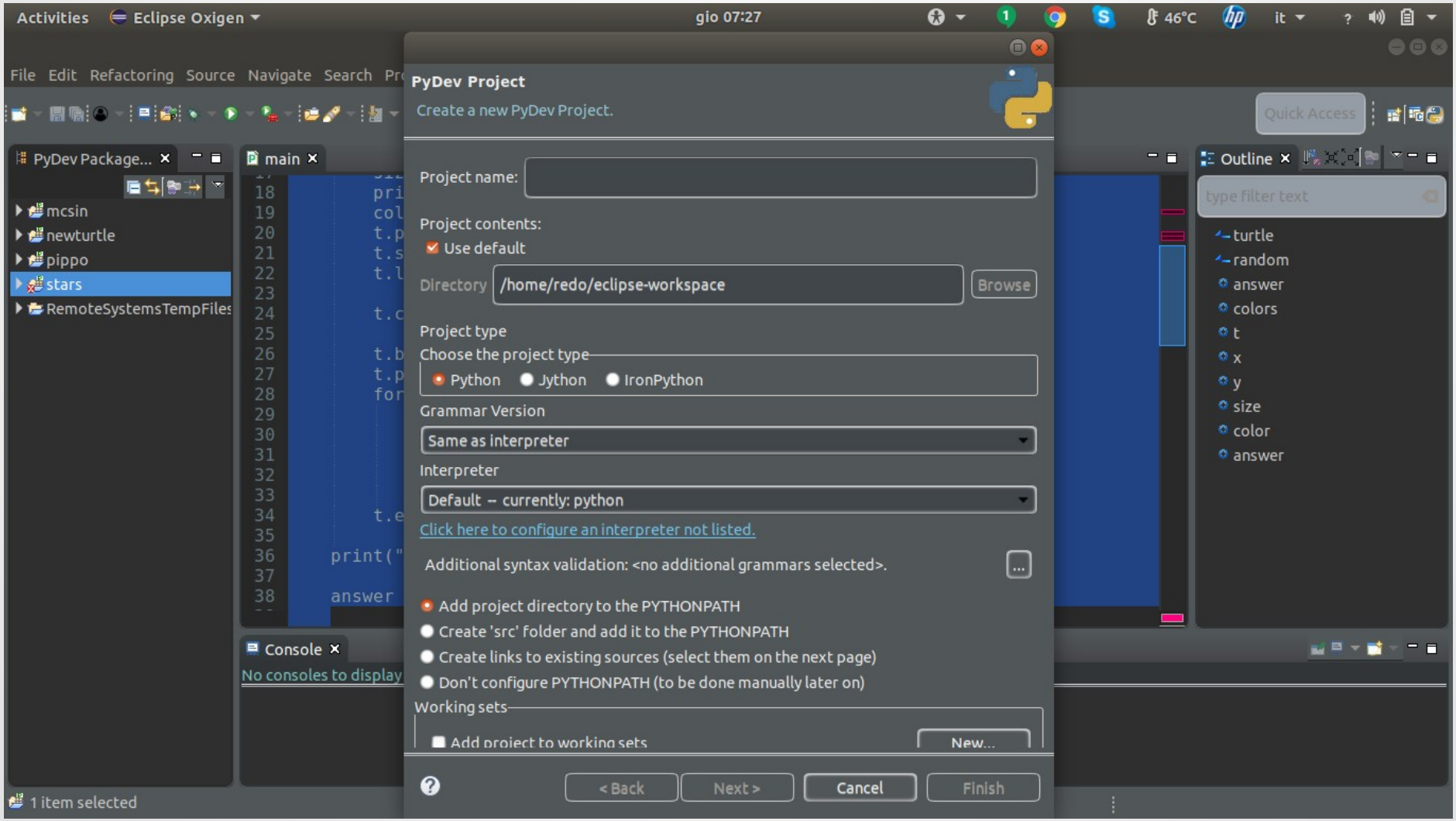
Another possibility is simply to write the source in an ASCII file and then:

$ python file.py



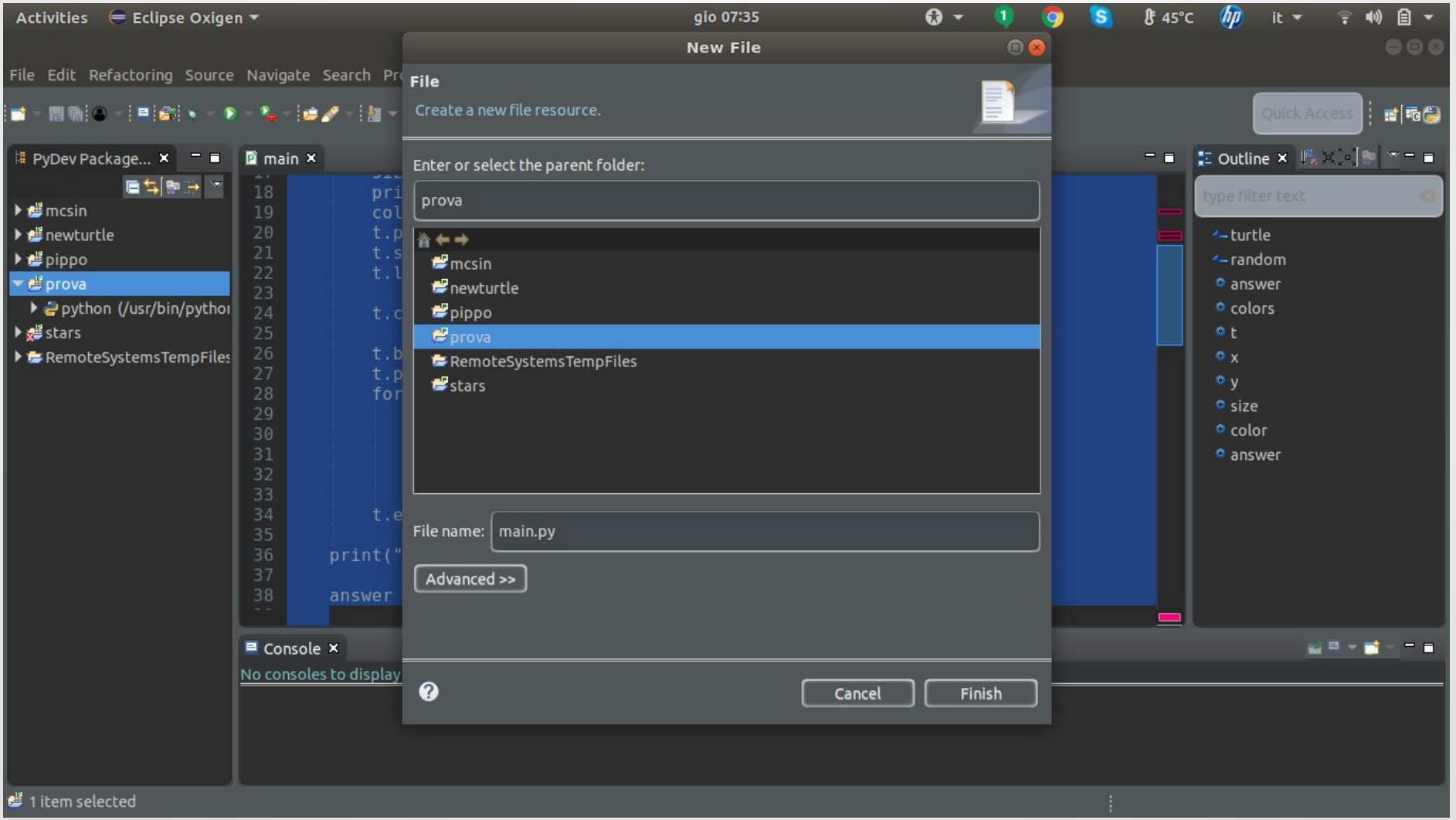
the Python interpreter

Using an IDE: File → PyDev Project



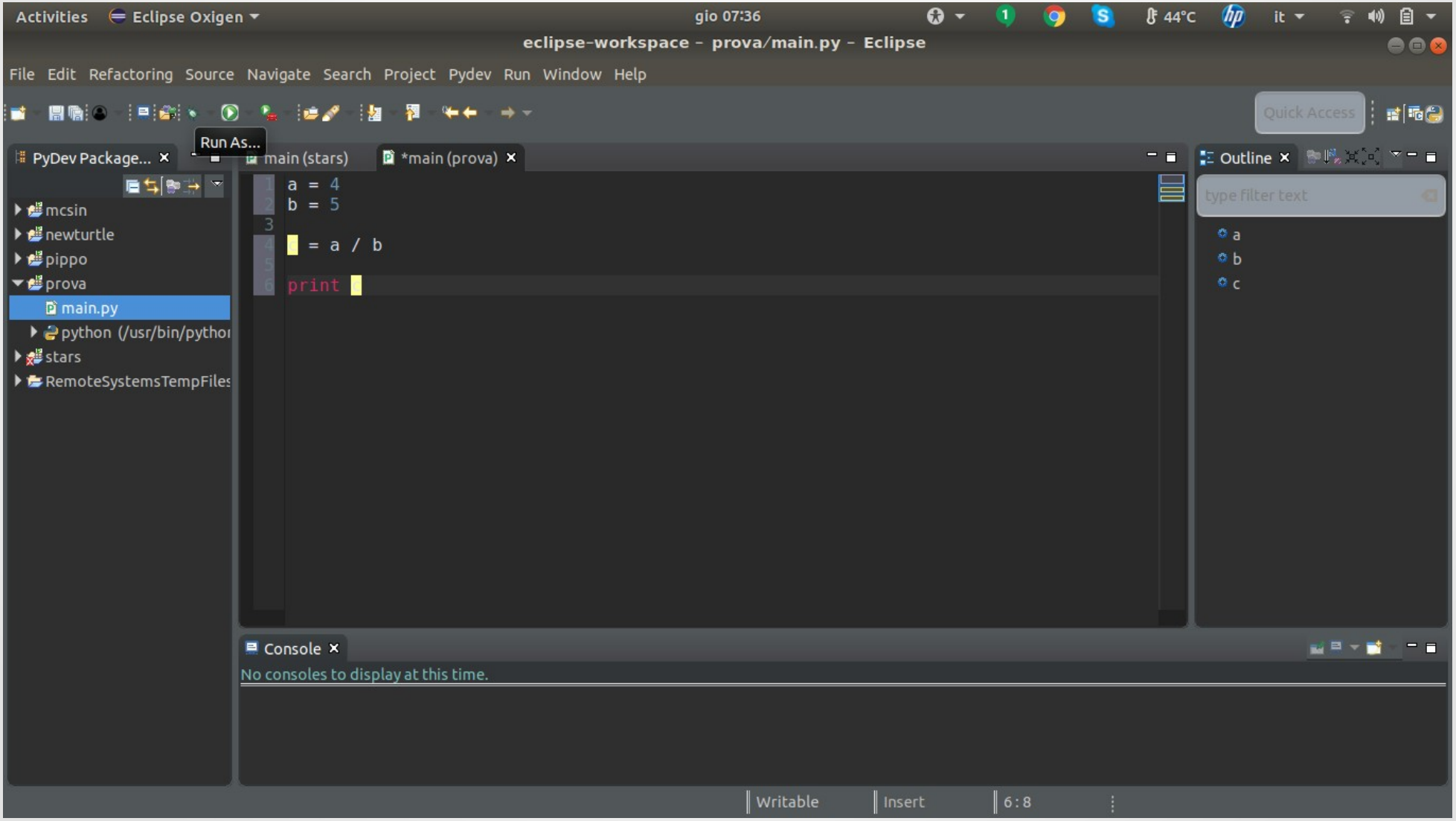
the Python interpreter

Click project and using the right mouse button New → File



the Python interpreter

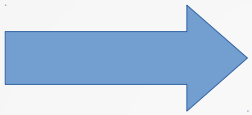
Click Run As



the Python interpreter

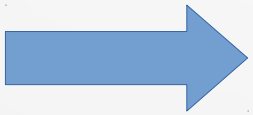
Click Run As

```
1 a = 4
2 b = 5
3
4 c = a / b
5
6 print c
```



```
Console ×
<terminated> main.py [/usr/bin/python]
0
```

```
1 a = 4.0
2 b = 5.0
3
4 c = a / b
5
6 print c
```



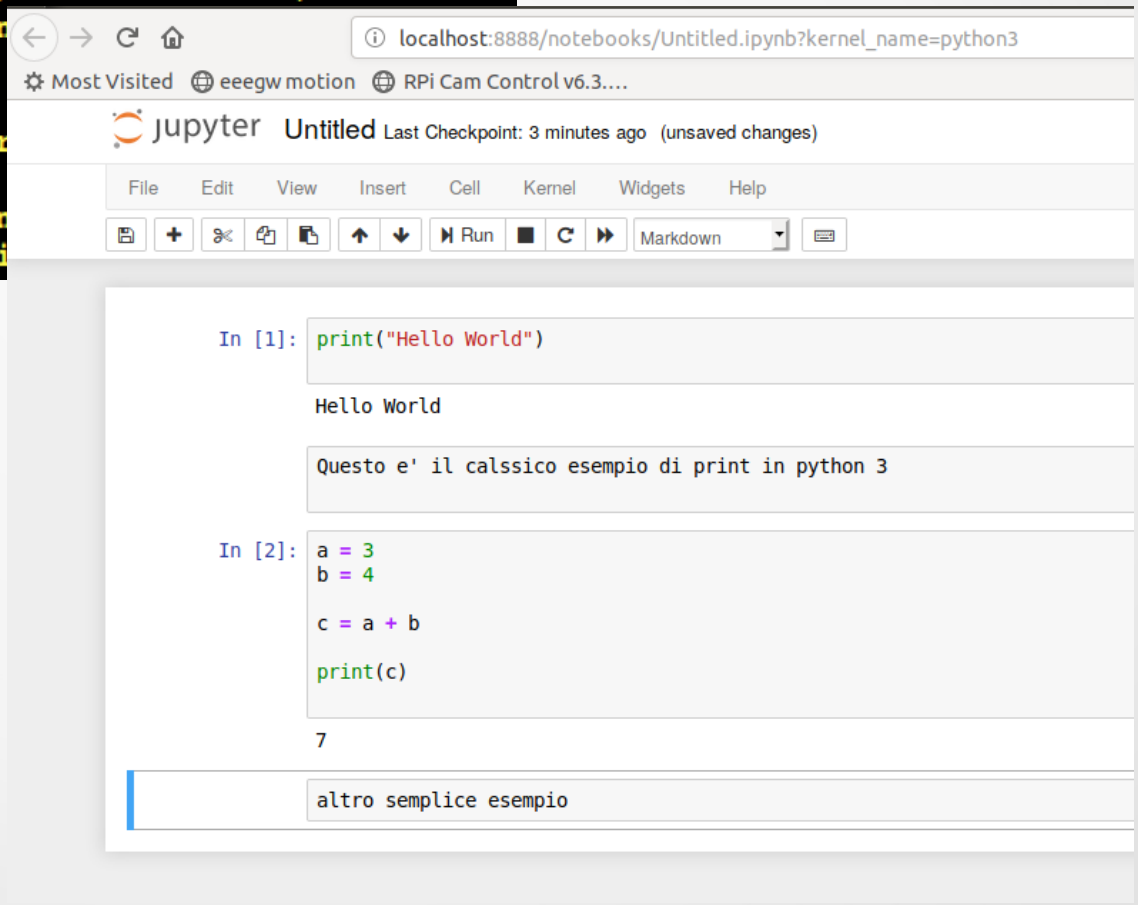
```
Console ×
<terminated> main.py [/usr/bin/python]
0.8
```

Or Jupyter

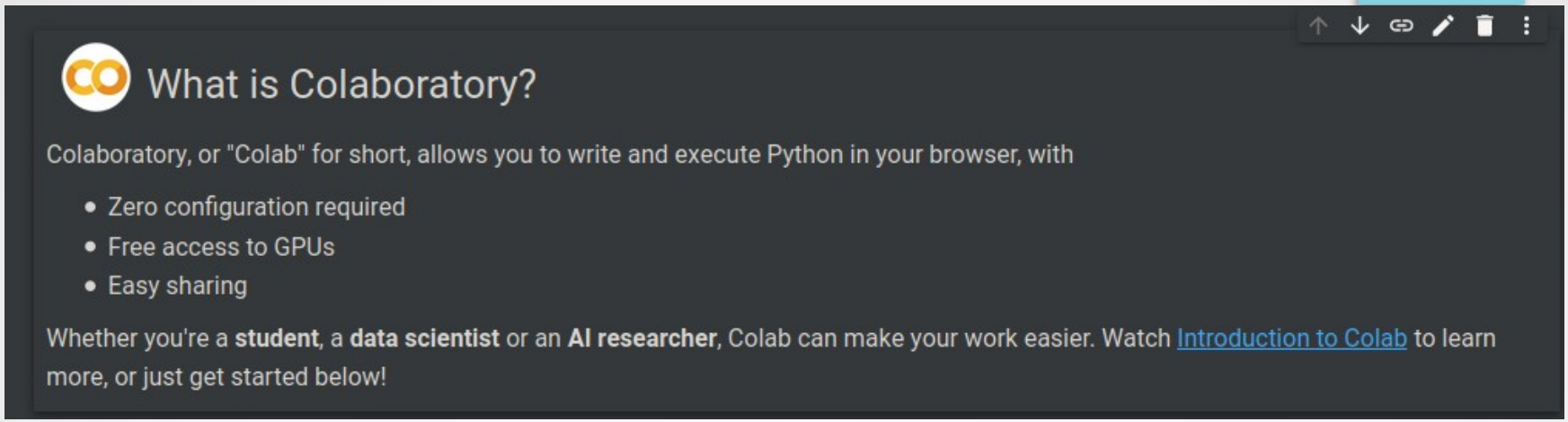
As a web application in which you can create and share documents that contain live code, equations, visualizations as well as text, the Jupyter Notebook is one of the ideal tools to help you to gain the data science skills you need.

Or Jupyter

```
[redo@buchner FondamentiDiProgrammazione]$ jupyter notebook
[I 18:32:09.367 NotebookApp] Serving notebooks from local dir
nmazione
[I 18:32:09.367 NotebookApp] The Jupyter Notebook is running
[I 18:32:09.367 NotebookApp] http://localhost:8888/?token=510
[I 18:32:09.367 NotebookApp] Use Cor
[C 18:32:09.368 NotebookApp]

    Copy/paste this URL into your br
    to login with a token:
        http://localhost:8888/?token
[I 18:32:09.617 NotebookApp] Accepti
```

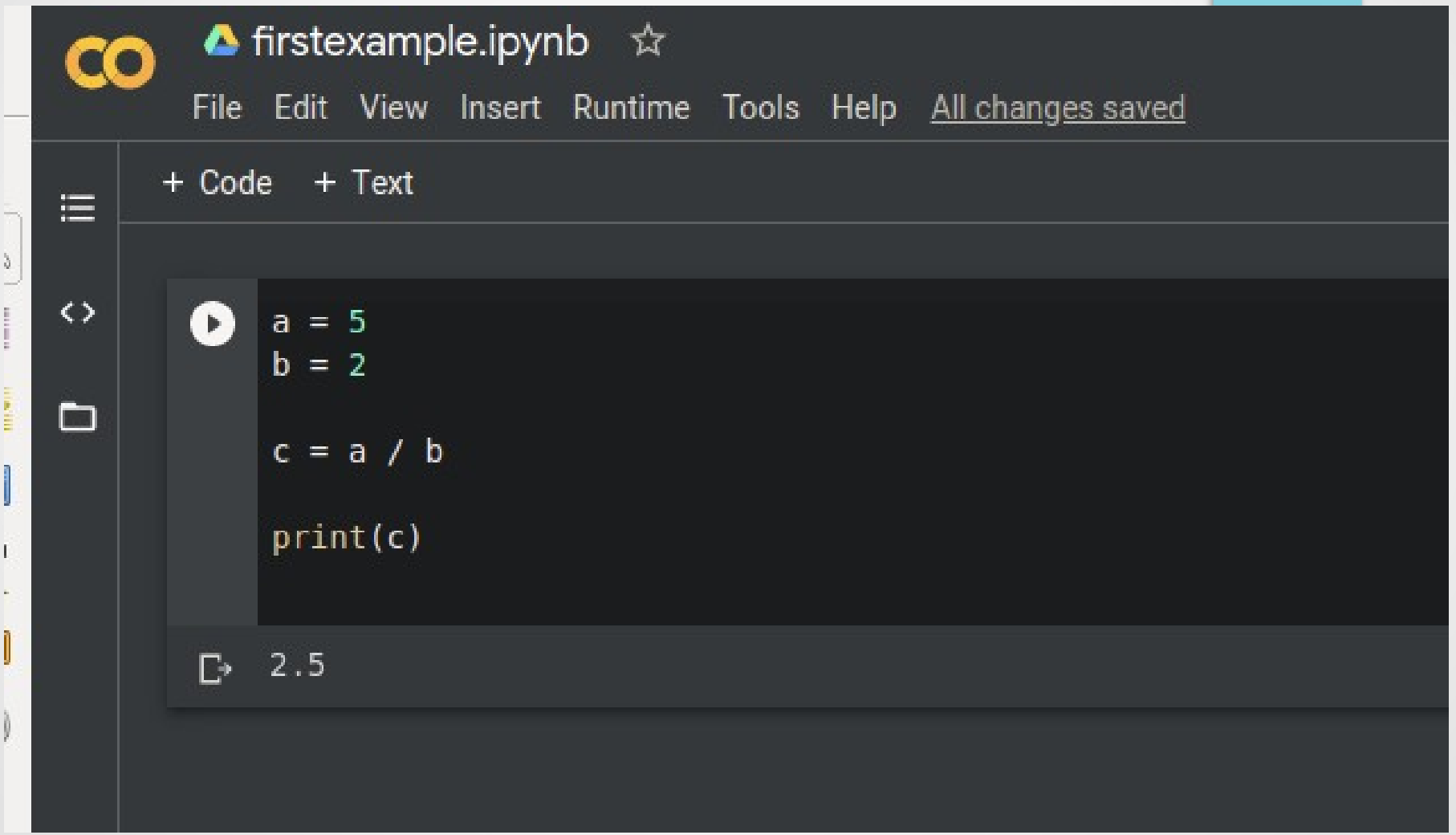


Or Colab



Let’s start →

Or Colab


firstexample.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
a = 5
b = 2
c = a / b
print(c)
2.5

HELLO WORLD!

Hello world

- The classic program used to illustrate the syntactic basics of any programming language

```
print ("Hello World")
redo@buchner /home/redo/L
└─ $ python3 hello.py
Hello World
```

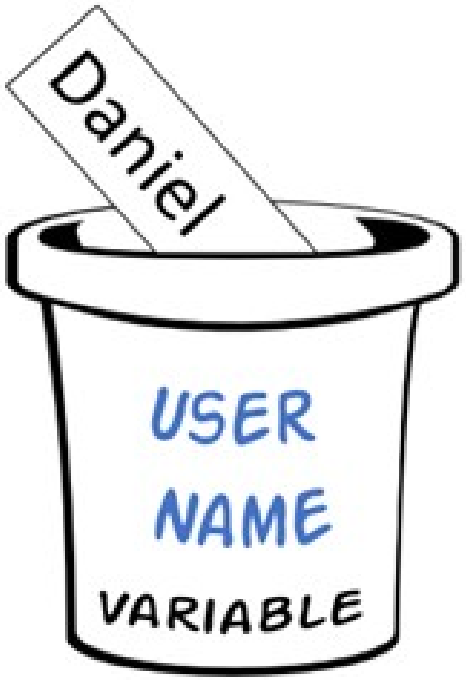
HELLO WORLD 2

Hello world

```
└─ $ cat name.py
name = input("Insert your name: ")

print("Ciao ", name)
```

```
└─ $ python3 name.py
Insert your name: Loriano
Ciao  Loriano
```

Hello world



The variable has no value until the user enters a name.

PYTHON BASICS

Python Basics

- Let's see a simple code that allows us to illustrate some of the basic features of python syntax:
  - I can add comments using the # character
  - = is used to assign values to variables
  - To do operations between numbers and variables I can use the usual operators +, -, *, /

```
x = 34 - 23 # commentare il codice
y = "Hello"
z = 3.45
```

Python Basics

- Let's see a simple code that allows us to illustrate some of the basic features of python syntax:
  - == is the operator that server compare values
  - The logical operators are instead: and, or, not
  - The + operator can also be used to concatenate strings

```
if z == 3.45 or y == "Hello":
    x = x + 1
    y = y + " World"
```

Python Basics

- Let's see a simple code that allows us to illustrate some of the basic features of python syntax:
  - print is the basic command used to print on the screen
  - **Variables do not need to be explicitly declared the first time that I assign a value the variable is created and given a type**

```
print("Valore di x: ", x)
print("Valore di y: ", y)

print(z)
```

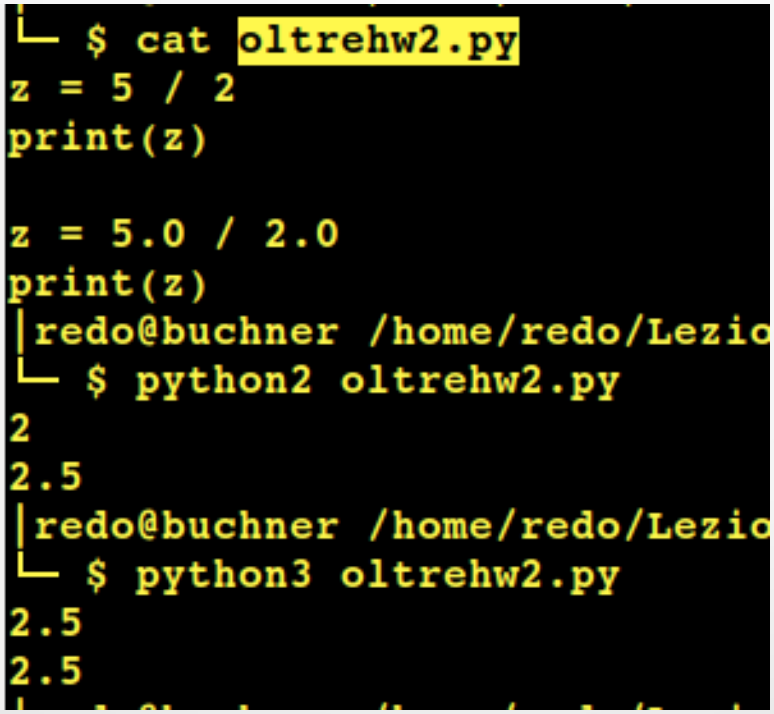
Python Basics

```
x = 34 - 23 # commentare il codice
y = "Hello"
z = 3.45

if z == 3.45 or y == "Hello":
    x = x + 1
    y = y + " World"

print("Valore di x: ", x)
print("Valore di y: ", y)

print(z)
```

```
Valore di x:  12
Valore di y:  Hello World
3.45
```

Python Basics

- There is no end-of-line character, if a line of code has to be broken on several lines, use \
- **Python 2** By default the numbers are integers so z = 5/2 will give as a result 2



IF … THEN … ELSE

If … then … else

```
INSERT NUMBER I
IF I LOWER THEN 0
    PRINT “il numero è minore di zero”
ELSE IF I EQUAL TO 0
    PRINT “Il numero è uguale a zero”
ELSE
    PRINT “Il numero è maggiore di zero”
ENDIF
```

If … then … else

- To identify blocks of code in python, empty spaces are used, not for example the {} as in C / C ++

```
└─ $ cat oltrehw4.py
si = input ("inserisci un numero: " )
i = float(si)

if i < 0 :
    print("numero inferiore a zero")
elif i == 0:
    print("inseiro uguale a zero")
else:
    print("numero maggiore di zero")
```

```
inserisci un numero: -31
numero inferiore a zero
inserisci un numero: 3.0
numero maggiore di zero
```

If … then … else

- You can use eval()

```
si = eval(input ("inserisci un numero: " )) # The expression argument
                                            # is parsed and evaluated as a
                                            # Python expression (technically
                                            # speaking, a condition list)
                                            # using the globals and locals
                                            # dictionaries as global and local
                                            # namespace.

if si < 0 :
    print("numero inferiore a zero")
elif si == 0:
    print("inseiro uguale a zero")
else:
    print("numero maggiore di zero")
```

```
└─ $ python3 oltrehw4.py
inserisci un numero: 10
numero maggiore di zero
inserisci un numero: print("si ", si)
si  10
Traceback (most recent call last):
  File "oltrehw4.py", line 20, in <module>
    if si < 0 :
TypeError: '<' not supported between instances of 'NoneType' and 'int'
```

LOOPS

Loops

SET N TO 0

SET n TO 0

Repeat the following:

a. If n >= 10, terminate the repetition, otherwise.
b. Increment N by n
c. PRINT n

PRINT N

Loops

- To identify blocks of code in python, empty spaces are used, not for example the {} as in C / C ++
- **Python is case sensitive**

```
└─ $ less oltrehw3.py
N = 0

for n in range(0,10):
  N = N + n
  print(n)

print("Valore finale: ", N)
```

Loops

```
N = 0

for n in range(0,10):
  N = N + n
  print(n)

print("Valore finale: ", N)
```

```
0
1
2
3
4
5
6
7
8
9
Valore finale:  45
```

EXAMPLE

Example of a numerical procedure

- It is possible to find an algorithm to solve almost any problems, but not all. For example, calculate the solutions of a second-order equation:

```
INPUT  A, B, C
COMPUTE D = (B*B)-(4 * A * C)
IF D >= 0.0
  SOL1 = (-1,0 * B + SQRT(D)) / (2,0 * A)
  SOL2 = (-1,0 * B - SQRT(D)) / (2,0 * A)
  PRINT SOL1 AND SOL2
ELSE
  PRINT “non ci sono soluzioni reali”
```

Example of a numerical procedure

```
└─ $ cat solv.py
import math

a = float(input("insert a:"))
b = float(input("insert b:"))
c = float(input("insert c:"))

print("a = ", a, " b = ", b, " c = ", c)

delta = math.pow(b, 2.0) - (4.0 * a * c)

if (delta >= 0):
  tn = math.sqrt(delta)
  sol1 = ((-1.0 * b) + tn) / (2.0 * a)
  sol2 = ((-1.0 * b) - tn) / (2.0 * a)
  print(sol1, sol2)
else:
  print("No real solutions")
```

Example of a numerical procedure

```
    c = float(c1)
```

Console

```
<terminated> main.py [/usr/bin/python]
insert a:1
insert b:15
insert c:2.5
a =  1.0  b =  15.0  c =  2.5
-0.168560850692 -14.8314391493
```

EXERCISE 1

EXERCISE 1

- Write a program in python that reads 10 numbers, after it calculates the average value and prints the result

```
[redo@banquo esercizipython (master)]$ python ex1.py
inserisci il numero 1 45
inserisci il numero 2 67
inserisci il numero 3 84
inserisci il numero 4 2
inserisci il numero 5 4
inserisci il numero 6 6
inserisci il numero 7 7
inserisci il numero 8 8
inserisci il numero 9 9.0
inserisci il numero 10 13.0
la somma:  245.0
valore medio:  24.5
[redo@banquo esercizipython (master)]$
```

BREAK

Break

- The usually syntactically nested break in a for loop or while, ends the nearest loop, skipping the optional else clause if the loop has one.

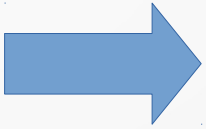
```
└─ $ cat testbreaks.py
for i in range(0,10):
    print("i: ", i)
    if i > 5:
        break;
```



```
i:  0
i:  1
i:  2
i:  3
i:  4
i:  5
i:  6
```

Break and nested loops

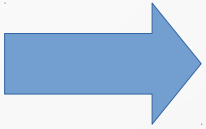
```
└─ $ cat testbreaks1.py
for i in range(10):
    print("loop 1: ", i)
    for j in range(10):
        print("    loop 2: ", j)
        if (j > 5):
            break;
```



```
loop 1:  0
    loop 2:  0
    loop 2:  1
    loop 2:  2
    loop 2:  3
    loop 2:  4
    loop 2:  5
    loop 2:  6
loop 1:  1
    loop 2:  0
    loop 2:  1
    loop 2:  2
    loop 2:  3
    loop 2:  4
    loop 2:  5
    loop 2:  6
loop 1:  2
    loop 2:  0
    loop 2:  1
    loop 2:  2
```

Break and nested loops

```
└─ $ cat testbreaks2.py
for i in range(10):
    print("1) loop 1: ", i)
    for j in range(10):
        print("    loop 2: ", j)
        if (j > 5):
            break;
    print("2) loop 1: ", i)
```



```
1) loop 1:  0
    loop 2:  0
    loop 2:  1
    loop 2:  2
    loop 2:  3
    loop 2:  4
    loop 2:  5
    loop 2:  6
2) loop 1:  0
1) loop 1:  1
    loop 2:  0
    loop 2:  1
    loop 2:  2
    loop 2:  3
    loop 2:  4
    loop 2:  5
    loop 2:  6
2) loop 1:  1
1) loop 1:  2
```

RANDOM NUMBER

Random Number

- https://docs.python.org/2/library/random.html
  - Pseudo Random vs Random
  - cat /dev/urandom
  - cat /dev/urandom | od -vAn -N2 -tu
  - cat /dev/random
  - cat /dev/random | od -vAn -N2 -tu

```
[redo@buchner ~]$ cat /dev/random | od -vAn -N2 -tu2
 38004

^C
[redo@buchner ~]$ cat /dev/random
```

Random Number

- https://docs.python.org/2/library/random.html
  - random.randint(a, b)generates a random integer N such that a <= N <= b

```
└─ $ cat rnd.py
import random

for i in range(100):
  print(random.randint(0,10))
```

Random Number

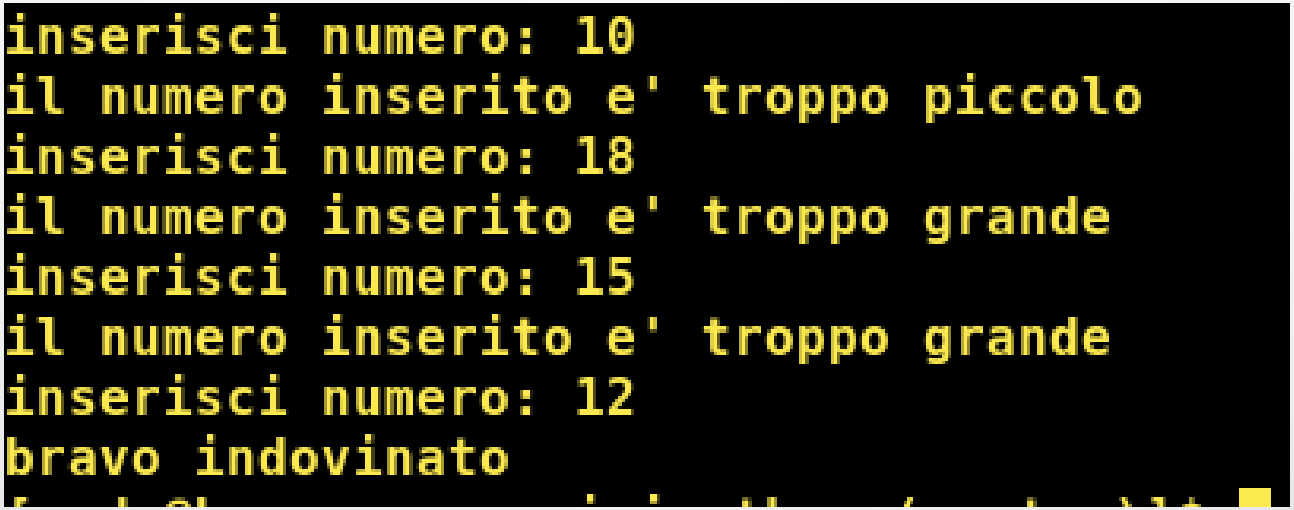
```python
import random

for i in range(100):
  print(random.randint(0,10))
```

```
9
3
3
4
2
4
0
5
4
10
10
2
3
10
```

EXERCISE 2

Exercise 2

- Write a program that generates a random number R between 0 and 20 and asks to the user to guess the number with a maximum of 10 attempts. Each time the program will simply writes if the number inserted is greater or less than R. Clearly if the inserted number is equal to the generated R random number the program will exit



Pseudocode

```
GENERATE A RANDOM NUMBER rnd

Repeat the following:
    INPUT b
    IF b IS EQUAL TO rnd
        PRINT “well done”
        BREAK
    ELSE IF b < rnd
        PRINT “inserted number is too small”
    ELSE
        PRINT “inserted number is too big”
    ENDIF
```