

Second lesson: Operating systems. Linux.

Compilation vs. interpretation.

>>A computer program in the form of a [human-readable](#), computer programming language is called [source code](#). Source code may be converted into an [executable image](#) by a [compiler](#) or [executed](#) immediately with the aid of an [interpreter](#).

Compilers are used to translate source code from a programming language into either [object code](#) or [machine code](#).^[19] Object code needs further processing to become machine code, and machine code consists of the [central processing unit's](#) native instructions, ready for execution. Compiled computer programs are commonly referred to as executables, binary images, or simply as [binaries](#) – a reference to the [binary file format](#) used to store the executable code.

Interpreters are used to execute source code from a programming language line-by-line. The interpreter [decodes](#) each [statement](#) and performs its behaviour. One advantage of interpreters is that they can easily be extended to an [interactive session](#). The programmer is presented with a prompt, and individual lines of code are typed in and performed immediately.

The main disadvantage of interpreters is computer programs run slower than when compiled. Interpreting code is slower because the interpreter must decode each statement and then perform it. However, software development may be faster using an interpreter because testing is immediate when the compiling step is omitted. Another disadvantage of interpreters is an interpreter must be present on the executing computer. By contrast, compiled computer programs need no compiler present during execution.

One common scripting language is [Unix shell](#), and its executing environment is called the [command-line interface](#).

No properties of a programming language require it to be exclusively compiled or exclusively interpreted. The categorization usually reflects the most popular method of language execution. For example, Java is thought of as an interpreted language and C a compiled language, despite the existence of Java compilers and C interpreters.

*

```
#####  
# very basic, example-based overview to Python (2) types and expressions  
# 2nd lesson of the course 'Sistemi Operativi', Laurea Triennale in Matematica  
# Universita' di Roma, "La Sapienza"  
# academic year 2017-2018  
# miguel.berganza@roma1.infn.it  
#####
```

```
# variable assignment and logical equality  
#####
```

```
a=3  
a==3
```

```
a+=1
a is 3
a > 10 or a is 4
```

```
# printing strings and variables with 'print'
#####
```

```
print a, b, 'this is a string'
    # it prints strings (formatted), and it also admits the
C-style formatting:
print("%d %e %.3f \n" % (a,b,c))
```

```
# Module import examples
#####
```

```
import matplotlib.pyplot as plt
import numpy as np
from numpy import zeros
import random as rnd
from random import random
```

```
rnd.random()
```

```
#if the module has changed in the meanwhile:
reload(modulename)
```

```
# Whitespace formatting
#####
for i in range(4):
    for j in range(i):
        print i,j
    print ''
```

```
# Lists
#####
lista=range(10) #it works also range(min,max)
len(lista)
sum(lista)
```

```
# slice syntaxis:
lista[-1]
lista[1:-1]
lista[-3:]
```

```
#slice assignment can change its length:
```

```

lista[:]=[]

#membership
lista.append(17)
17 in lista

#concatenating lists:
lista.extend([4, 5, 6])

#one uses also '+'
lista+lista #and also '*' between lists and integers
#note the difference with 'append':
lista.append([4, 5, 6]) # (an example of list nesting) c.f.
lista[-1]

#an exercise: write a Python code that prints the elements of a list in the format 'index
listelement'

#two alternatives to this the first is using 'in':
i=0
for elemento in lista:
    print i elemento
    i+=1

#very important: assignment in lists
lista=range(4)
lista2=lista
lista.extend([4]) #what happens to lista2? and what if now
you do lista2=lista[:]
#indeed, lista[:] is a copy of 'lista', that may be used to iterate over members of a list to be
modified

lista=['a','ab','abc']
for w in lista[:]:
    if len(w) > 2:
        lista.append(w)

#sorting a list:
sorted(['a','c','b'])
list.sort()

#for more operations on Lists, see 'https://docs.python.org' chapter 5. the most important are:
# insert pop remove index sort

# Functions
#####
def somma(a=4,b=3):

```

```

        return a+b

def applytoone(f):
    return f(1)

# Tuples: with () or without. they are immutable lists.
#####
mytuple = 'a','b','c'

def funzione(a,b):
    return a*b , a/b

mytuple=funzione(10,20) #OCCHIO: why is the result this way
(beware the types)?

x,y = mytuple

x,y = y,x # "pythonic" variable swapping


# Loops and conditionals
#####
x=0
while x < 10:
    print x, " is less than 10"
    x+=1

for x in range(10):
    if x==3:
        continue # go immediately to the next iteration
    if x==5:
        break # quit the loop entirely
    print x


# List (and set) construction (they are called 'comprehensions')
#####
lista = [1.*lista[i] for i in range(len(lista))]
lista = [1.*i for i in lista] #much easier

[x**2 for x in range(15) if x % 2 == 0]

[[i,j] for i in range(4) for j in range(i)] #they are executed
in inverse order


# Strings
#####

```

```

#some syntax conventions
mystring='this is \n a 2-line string with \t a tab space'
    #string concatenation
print 'T'+mystring[1:]
#finding a substring
'this' in mystring or 'Miguel' in mystring

#splitting a string (and returning a list)
mystring.split(' ')

#initializing a string
nuovastringa=''

```

#Preliminar exercises:

```
#####
```

#EX0a. Write a function that returns a list that contains only the elements that are common between two input lists (the output should not present element repetition).

#EX0b. Write a 10x10 matrix with elements $a_{ij}=i-j$. Write a 1-line comprehension that transpose a general MxN matrix. Write a function that transposes a general non-square matrix (in a 1-line comprehension) taken as argument.

#EX0c. Flatten (eliminate inter-parenthesis []) the vector `vec = [[1,2,3], [4,5,6], [7,8,9]]` with a one-line comprehension

#EX0d. Write a Python function receiving a string and printing the string with the words in alphabetic order.

#EX0e. Write a Python function that receives (as argument) a given integer (up to five, otherwise it prints an error message -in Neapolitan dialect-) and prints its name. Write a different function returning the name of the rest of the division by 10 (if it is ≤ 5) of a given input integer.

#EX0f. Write a Python function receiving a list of numbers and returning a list of the outputs of the function in EX0e. Test it with a list of 25 random numbers.

#EX0g. Write a Python function ordering a list (without using the 'sorted' function). Do it with and without the list function 'insert' (see the script 'ordinalista.py' for some solutions).

```
#####
```

basic I/O

```
#####
```

#opening and closing files

```
myfile=open(filename,mode)
```

#mode: 'r' read, 'w' write, 'a' append, 'r+' read and write

#closing a file:

```
myfile.close()
```

#a conditional:

```
myfile.closed()
```

#reading and writing:

```
myfile.read()          #reading the entire file and putting it
into a string
myfile.readline()      #as you expect
myfile.write(string)    #writing the string 'string'. write()
works similarly as in C for writing variables. For instance:
myfile.write('miei valori = %f %d %.12e \n' % (a1,a2,a3))
```

#reading a file line-by-line (i.e. one line at once)

```
for riga in myfile:
    print riga
```

#asking the user for an input:

```
cena=input('cosa vuoi per cena?: ')
```

#basic random numbers

```
#####
import random
random.random()          #returns a uniformly distributed
random number between 0. and 1.
random.seed(myseed)      #idem, with a given seed (the result
depends on 'myseed' only
```

```
#####
```

#Some exercises

```
#####
```

#EX1. Develop a function that prints a Fibonacci series up to the n-th number. Idem but returning the list with the Fibonacci series.

#EX2. Develop a function that prints all the prime numbers lower than a given number. Also: develop a function that prints the lower prime number smaller than a given number.

#EX3. Develop a function that reads a text file, and writes in a different file the text of the original file without the vocals.

#EX4. Generate a random integer between 1 and 10. Ask the user to guess a number, then tell her whether the guessed number is too low or too high, and repeat the question until she guesses the generated random number.

#EX5. Repeat exercise EX0d. but avoiding the function sorted(): write a function receiving a string and returning a string with the words in alphabetic order.

#EX6. Write a Python function that opens a file and creates a second file in which it is written, in one column, the finite difference between the elements of the n-th column of the

original file (i.e., the i -th row of the output column should be the difference $a(i+1)-a(i)$, being $a(i)$ the i -th row of the n -th column of the original file). The function should take n and the filename as arguments. You can test your program with the test file 'BinderL128.dat'.

```
#####  
#####  
#####
```