# 2 – Control of the instruction flow

1. Simple instructions and instruction sequences
2. Conditional instructions
3. Repetition instructions
4. Embedded instructions

# 2.1 - Simple instructions and instruction sequences

- A simple or elementary instruction is a basic action defined by the Python language. It is an **expression evaluation**, an **assignment** or an **instruction with a very specific function**.
- An **expression evaluation** has the form of the expression to evaluate. The value of the expression is computed (in interactive mode, it is displayed on the standard output).
- **An assignment** has the form : *target* = *expression* . The target often is a variable but it can be a more complex object able to receive and to store a value (tuple, array entry …). The = assignment operator can be preceded by an operator used to express the new value of a store from the old one.
- The most simple way of composing instructions is to organize them in **sequences**. In a sequence, two instructions are separated by a semicolon or a newline.

# 2.1 - Simple instructions and instruction sequences

```
>>> 2+3
5

>>> "la" == "le"
False

>>> x = 2 + 3
>>> x
5

>>> x *=2
>>> x
10

>>> y = [1, 2]
>>> y[0] = 7
>>> y
[7, 2]

>>> x = 7; x
7
```

# 2.2 - Conditional instructions

- A **conditional instruction** is used to perform a choice between instructions according to **conditions**.

- The most simple conditional instruction is the **if** instruction, which has the following syntax : ***if*** *<condition> :*
  *<block of instructions>*

- *<condition>* is an expression and *<block of instructions>* is a sequence of instructions which are indented in the same manner with respect to the line where the condition is.

- The meaning of the instruction is the following: if the value of *<condition>,* interpreted as a boolean expression, is *True* , *<block of instructions>* is executed, otherwise no action is performed.

# 2.2 - Conditional instructions

```
>>> x=0
>>> if x >=0:
        x
        x+1
        x-1

0
1
-1

>>> x= -1
>>> if x >=0:
        x
x+1
x-1

0
-2
```

# 2.2 - Conditional instructions

- The **if … else** instruction has the following syntax:

  ***if*** *<condition> :*

  *<block1 of instructions>*

  ***else*** *:*

  *<block2 of instructions>*

- The meaning of the instruction is the following: if the value of *<condition>,* interpreted as a boolean expression, is *True , <block1 of instructions> is* executed, otherwise *<block2 of instructions>* is executed.

# 2.2 - Conditional instructions

```
>>> x = 7
>>> if x % 2 == 0 :
        print "x is even"
else:
        print "x is odd"

x is odd


>>> x = 8
>>> if x % 2 == 0 :
        print "x is even"
else:
        print "x is odd"

x is even

>>>
```

# 2.2 - Conditional instructions

- It is possible to introduce a test on a sequence of alternative conditions by using the key word **elif** in the **if … else** instruction with the following syntax:

***if*** *<condition 1> :*
    *<block 1 of instructions>*
***elif*** *<condition 2>:*
    *<block 2 of instructions>*
        .
        .
        .
***elif*** *<condition n>:*
    *<block n of instructions>*
***else*** *:*
    *<block n+1 of instructions>*

- The meaning of the instruction is the following: if the value of *<condition 1>* is *True , <block 1 of instructions>* is executed, otherwise if the value of *<condition 2> is True , <block 2 of instructions>* and so on. If no condition is true, *<block n+1 of instructions>* is executed.

# 2.2 - Conditional instructions

```
>>> hour = 15

>>> if hour > 6 and hour <= 13 :
        print ”it is morning"
elif hour > 13 and hour <= 19 :
        print ”it is afternoon"
elif hour > 19 and hour <= 23 :
        print ”it is evening"
else:
        print ”it is night"


it is afternoon
```

# 2.3 - Repetition instructions

- **Repetition instructions** are used to repeat a block of instructions.

- The **for** instruction has the following syntax:

  ***for*** *<controller> in <seq>:*

  *<block of instructions>*

- In this instruction, *<controller>* is a variable used to control the number of iterations and *<seq>* is an object of type *sequence (string, list, tuple, dictionary).*

- The meaning of the instruction is the following: variable *<controller>* successively takes the different values of *<seq>* in the order of their apparition. At each instantiation of the controller, *<block of instructions> is* executed.

# 2.3 - Repetition instructions

```
>>> range(5)
[0, 1, 2, 3, 4]

>>> for n in range(5) :
        print ”the square number of ", n, ”is: ", n**2


the square number of  0 is:  0
the square number of 1 is:  1
the square number of 2 is:  4
the square number of 3 is:  9
the square number of 4 is:  16
>>>
```

# 2.3 - Repetition instructions

- The **while** instruction has the following syntax:

  ***while*** *<condition> :*

  *<block of instructions>*

- In this instruction, *<condition>* is a boolean expression.

- The meaning of the instruction is the following: expression *<condition>* is evaluated and if its value is True, *<bloc d'instructions>* is executed; then, expression *<condition>* is evaluated again and so on. As soon as *<condition>* takes the value False, the execution of the instruction ends.

- "For" instructions are used when iterations are known in advance and "while" instructions when iterations are not known in advance but they depend on a condition known in advance.

# 2.3 - Repetition instructions

```
>>> n = 0
>>> while n <= 5 :
        print ”the square of ", n, ”is: ", n**2
        n += 1

the square of  0 is:  0
the square of 1 is:  1
the square of 2 is:  4
the square of 3 is:  9
the square of 4 is:  16
the square of 5 is:  25

>>>
```

# 2.3 - Repetition instructions

- The **break** instruction has to be used in a block of instructions inside a loop. It breaks the execution of the most internal loop in which it is located.

```
>>> for k in range(6) :
        for i in range(k+1) :
            if i > 1:
                if i == k :
                    print k
                elif  k % i == 0:
                    break

2
3
5
```

# 2.4 - Embedded instructions

- Compound instructions may be embedded in each other.

```
>>> input =  ”John is coming."
>>> output = ""
>>> for c in input :
      if c == " ":
            output += "\n"
      else :
            output += c


>>> print output
John
is
coming.
>>>
```

# 2.5 Exercises

1. Give the result of the execution of the following program when value “level” is entered for w*1*. The same question when value “test” is entered. What is the function of this program ?

```
w1 = input(“Enter a word !\n”)
w2 = “”
for c in w1 :
    w2 = c + w2
print  w1 == w2
```

2. Give the result of the execution of the following program when value 5 is entered for x. The same question when value 6 is entered. What is the function of this program ?

```
x = input(”Enter an integer ! ")

y =2
goon = True
while y < x and goon :
    z = y
    while z < x and goon :
        z += y
        if z == x :
            goon = False
    y += 1
print  goon
```

# 2.5 Exercises

3. Write a Python program for each of the following specifications (without using Python libraries of functions).

a) Compute the length of a string (go through the string and increment a variable at each step).

b) Count the number of punctuation signs of each kind in a text.

c) Display all words of a sentence, one per line (two words are assumed to be separated by exactly one space character).

d) Verify that a date written as a string has the format *mm/dd/yyyy*.For instance, the date 04/31/2010 is wrong written.

# 2.5 Solution of the exercises

1.

```
>>>
Enter a word !
"test"
False
>>>


>>>
Enter a word !
 "level"
True
>>>
```

- The program detects palindroms (words that remain unchanged when reversed)

# 2.5 Solution of the exercises

2.

```
>>>
Enter an integer ! 5
True
>>>
Enter an integer ! 6
False
>>>
```

The function of the program is to detect if an integer entered on the keyboard is a prime number.