3 - Types of data

1. Numeric and boolean data
2. Strings
3. Lists
4. Tuples
5. Dictionaries

3.1 - Numeric and boolean data

- The main numeric types are: **int** (integer), **long** (long integer), **float** (floating).
- If two expressions have different numeric types and if they are combined with an operator, the one that has the simplest type is converted in the most complex type.
- Boolean expressions have the type **bool** and two possible values: **True** and **False**.
- A boolean expression can be combined with a numeric expression by means of a numeric operator. Its value is thus converted into 0 or 1.
- A numeric expression can be combined with a boolean expression by means of a boolean operator. 0 is thus converted into False and the other values into True.

3.1 - Numeric and boolean data

```
>>> a = 1
>>> while a < 5 :
        print a**a**a , "     " , type(a**a**a)
        a += 1
>>>
1       <type 'int'>
16      <type 'int'>
7625597484987       <type 'long'>
1340780792994259709957402499820584612747936582059239337772356144372176403007354697680187429816690342769003185818648605085375388281194656994643364900608409 6       <type 'long'>

>>>
>>> a = 1.0
>>> while a < 5 :
        print a**a**a , "     " , type(a**a**a)
        a += 1
>>>
1.0     <type 'float'>
16.0    <type 'float'>
7.62559748499e+12       <type 'float'>
1.34078079299e+154      <type 'float'>
>>>
```

3.1 - Numeric and boolean data

```
>>> (2+False)*(2+True)
6
>>> False or 4
True
>>>
```

3.2 - Strings

- A string is a sequence of characters delimited with **simple or double quotes**. It is a constant of type **Str**.
- Special characters can be inserted in a string if they are preceded by the backslash symbol \: \n represents a newline, \t a tabulation, \’ an apostrophe, \" a double quote.
- To enter a string with newline characters on the keyboard, we can use the newline key if we put the string between triple simple or double quotes.

3.2 - Strings

- An expression of any type can be converted into a string by being put into **backquotes** (`). The interpreter computes the value of the expression and then the value is converted into a string.

- A string can be represented with the UTF8 encoding (**Unicode** norm) if preceded with the *u* symbol (for example *u'bonjour'*). The string takes the *unicode* type, which is different from *str*.

3.2 - Strings

```
>>> x='l\'eau\n\"Perrier\" '
>>> x
'l\'eau\n"Perrier" '

>>> print 'l\'eau\n\"Perrier\" '
l'eau
"Perrier"

>>> print ""l'eau
        \"Perrier\""""
l'eau
"Perrier"
```

```
>>> y =2
>>> z = `y+1`
>>> print y+1, type(y+1), z, type(z)
3 <type 'int'> 3 <type 'str'>

>>> for c in 'déjà' :
        print c
d
?
?
j
?
?

>>> for c in u'déjà' :
        print c
d
é
j
à
>>>
```

3.2 - Strings

- Strings like numbers belong to the **literal** type, but they differ from them because they are compound data constituted of atomic data: characters.

- A character located at position *n* in a string *s* can be addressed with the expression *s[n].* The position of the first character in a string is *0*. A character can also be addressed from the end of the string with expression *s[-m], m* being the number of characters counted from the concerned character until the last one, both being included in the counting.

- A substring of a string *s* can be addressed with an expression of the form *s[n:m]* (**slicing**). The value of the expression is the substring going from position *n* to position *m-1*. If $m \leq n$, then the substring is the empty string.

- Two strings can be concatenated with the binary operator +. A string can be repeated any number of times if it is multiplied with an integer.

3.2 - Strings

```
>>> print ‘bonjour’[2]
n
>>> print ‘bonjour’[-2]
u
>>> ‘bonjour’[2:5]
‘njo’
>>> ‘bonjour’[2:-2]
‘njo’
>>> ‘bonjour’[2:2]
‘’
```

```
>>>s='Je vais. Je viens'
>>>fin = False
>>>k = 0
>>>while not fin :
    if s[k:k+2] == '. ' :
        if s[k+2] >= 'A' and s[k+2] <= 'Z' :
            fin = True
    k += 1
print s[0:k]

Je vais.

>>>
```

3.2 - Strings

- To process strings, Python provides **predefined functions**. A function is a program which is callable by means of the function name followed by its arguments between parentheses. It returns a value.

- The table below presents the main functions for manipulating strings. The complete list of predefined functions is available at the following Web address:

http://docs.python.org/lib/built-in-funcs.html

Function call	Meaning
len(s)	Returns the length of the string s
chr(n)	Returns the character with n as its ASCII code
ord(c)	Returns the integer that is the ASCII code of character c
unichr(n)	Returns the character with n as its UTF8 code

3.2 - Strings

```
>>>entree = u'Je vais. Je viens'
>>>sortie = u”
>>>for c in entree :
      if c >= u’a' and c <= u’z' :
            code = ord(c )
            sortie += unichr(code - 32)
      else :
            sortie += c
print sortie

JE VAIS. JE VIENS

>>>
```

```
>>>entree = u'Je vais. Je viens'
>>>sortie = u”
>>> longueur = len(entree)
>>>for k in range(longueur):
      c = entree[k]
      if c >= u’a'  and <= u’z' :
            code = ord(c )
            sortie += unichr(code - 32)
      else :
            sortie += c
print sortie

JE VAIS. JE VIENS

>>>
```

3.2 - Strings

- Strings, as Python objects, are associated with functions linked to their types and called **methods**. For a given object *x*, it is possible to be aware of the associated methods by means of the function call *dir(x)*.

- Calling a method *m* associated with the object x with the arguments $a_1\ a_2\ \ldots\ a_n$ is performed through the expression $x.m(a_1, a_2, \ldots a_n)$.

- The table below presents some methods associated with strings (see the complete list at the following Web address: *http://docs.python.org/lib/string-methods.html*)

Method	Meaning
split(sep)	Returns the list of the substrings resulting from splitting the string with the separator *sep* (if this is missing, the default separator is the standard separator)
find(subs, debut, fin)	Returns the position of the first occurrence of the substring *subs* found in the segment *[debut : fin]* of the string. Returns -1 if no occurrence is found. The two last arguments are optional.

3.2 - Strings

```
>>>phrase = u'J\'ai mangé de l\'époisse, de la tome et du comté'
>>>mot = u'tome'
>>>trouve = False
>>>for m in phrase.split() :
        if m == mot :
                trouve = True
                break
if trouve :
        print 'le mot \" ', mot, '\" est dans la phrase '\" , phrase, '\"  '
else :
        print 'le mot \" ', mot, '\" n\'est pas dans la phrase \ \"', phrase, '\" '


Le mot " tome " est dans la phrase " J'ai mangé de l'époisse, de la tome et du comté "

>>>
```

3.2 - Exercises

1. Give the result of running the following program when the value “tout” is entered on the keyboard. The same question when the value “tot” is entered. What is the function of the program ?

```
w = input(“Enter a word !\n”)
l = len(w)
correct = True
k = 0
while correct and k < (l+1)/2:
    if w[k] != w[-k-1] :
        correct = False
    k += 1
print  correct
```

3.2 - Exercises

2. Give the result of running the program below when the following text is entered on the keyboard: “Marie is coming and buying a ring.” . What is the function of the program ?

```
text = input(“Enter a text ! \n”)
list_words = text.split()
k = 0
for w in list_words:
    position = w.find(‘ing’)
            if position != -1 and position + 3 == len(w) :
                k += 1
print  k
```

3.2 - Exercises

3. Write a Python program for each of the following specifications.

a) Count the number of occurrences of a given word in a text (the word separators are those used by default in the split method).

b) Replace all occurrences of a word with another one in a text (the definition of separator is the same as in the previous question).

c) From a text, extract the list of all words beginning with a capital letter.

d) From a text, extract the list of all numeric values (for a decimal number, the scientific notation is allowed with the symbols “E” or “e” for the beginning of the exponent).

3.2 - Solution of the exercises

1. Execution of the program

   - >>>
   - Enter a word !
   - "tout"
   - False
   - >>>

   - >>>
   - Enter a word !
   - "tot"
   - True
   - >>>

- The program detects if the words entered at the keyboard are palindromes.

3.2 - Solution of the exercises

2. Execution of the program:

   - >>>
   - Enter a text !
   - "Marie is coming and buying a ring."
   - 2
   - >>>

- The program counts the number words ending with ”ing” in a text.

3.2 - Solution of the exercises

3.

```
#a)

text =input("Enter a text ! \n")
word =input("Enter a word ! \n")
list_words = text.split()
k = 0
for w in list_words:
    if w==word :
        k += 1
print  "number of occurrences of ", word, " in the text: ", k

#b)
text1 =input("Enter a text ! \n")
word1 =input("Enter a word to replace ! \n")
word2 =input("Enter a word to substitute ! \n")
list_words1 = text1.split()
list_words2 = []
for w in list_words1:
    if w==word1 :
        list_words2.append(word2)
    else:
        list_words2.append(w)
text2 = ' '.join(list_words2)
print  "After replacement of ", word1, " by ", word2, " the text becomes:\n", text2
```

3.3 - Solution of the exercises

3.

```
#c)

text =input("Enter a text ! \n")
list_words1= text.split()
list_words2 = []
for w in list_words1:
    if w[0] >= 'A' and w[0] <='Z' :
        list_words2.append(w)
print "list of words starting with a capital letter: ", list_words
```

3.3 - Solution of the exercises

3.

```
#d
text = input("Input a text !\n")
liste_numbers = []
number = ''
number_found = False
dot_found = False
exp_found = False
n=0
for n in range(len(text)):
    c = text[n]
    if number_found :
        if c == '.':
            if dot_found:
                liste_numbers += [number]
                number = ''
                number_found = False
                dot_found = False
                exp_found = False
            else:
                dot_found = True
                number += '.'
elif c == 'E' or c=='e':
            if exp_found:
                liste_numbers += [number]
                number = ''
                number_found = False
                dot_found = False
                exp_found = False
            else:
                dot_found = True
                exp_found = True
                number += 'E'
        elif c >= '0' and c <= '9' :
            number += c
            if n == len(text)-1 :
                liste_numbers += [number]
        else:
            liste_numbers += [number]
            number = ''
            number_found = False
            dot_found = False
            exp_found = False
    elif c >= '0' and c <= '9' :
        number_found = True
        number += c
print liste_numbers

# Exemple d'entrée : 122.44 abbEcaa6.4E7666E90.5
```

3.3 - Lists

- Lists are sequences of Python objects, which are possibly heterogeneous. They have the type **list.** Their syntax is the following: $[\ o_1, o_2, \ldots, o_n\ ]$

- Like strings, lists are compound objects of type **sequence**. Thus, + et * operators apply to them and one can address one element of a list, as well as a sublist with the operation of slicing.

- Unlike strings, lists are **mutable** objects: an element of a list is modifiable while a character of a string is not. Therefore, element addressing and slicing are usable for modifying a list.

3.3 - Lists

```
>>>liste = [23, ‘bonjour’, [1,2]]
>>>print liste[2]
[1,2]
>>>print liste[2][0]
1
>>>liste[1] = ‘bonsoir’
>>>print liste
[23, ‘bonsoir’, [1,2]]
>>> liste[1:1] = [‘je’, ‘dis’]
>>>print liste
[23, ‘je’, ‘dis’, ‘bonsoir’, [1,2]]
>>> liste += [3,4]
>>>print liste
[23, ‘je’, ‘dis’, ‘bonsoir’, [1,2],
     3, 4]
```

```
>>>texte = u’Je suis tombé malade. Je n’ai pas travaillé.’
>>> liste_mots = texte.split()
>>> liste_mots_etiquetes = []
>>> for mot in liste_mots :
      liste_mots_etiquetes += [[mot, len(mot)]]

>>> print liste_mots_etiquetes
[[u'Je', 2], [u'suis', 4], [u'tomb\xe9', 5], [u'malade.', 7],
     [u'Je', 2], [u'n\u2019ai', 4], [u'pas', 3], [u'travaill\xe9.',
     10]]
```

3.3 - Lists

- As Python objects, lists are associated with **methods**. The table below presents some methods associated with lists (see the complete list at the Web address: *http://docs.python.org/lib/typesseq-mutable.html*)

Method	Meaning
count(elt)	Returns the number of elements from the list that have the value of *elt*
index(elt)	Returns the lowest index of an element that have the value of *elt*. If there is no element, an error is triggered.
insert(i,elt)	Inserts the value of *elt* just after the element at position *i.*
remove(elt)	Deletes the first element that has the value of *elt.*
reverse()	Reverses the order of the list.
sort()	Order the list according the default order associated with the type of its elements.

3.3 - Lists

```
>>>texte = u'Le médecin a examiné le malade mais le malade n'était pas vraiment
    malade.'
>>> liste_mots = texte.split()
>>> lexique = [u'médecin', u'malade']
>>> for mot in lexique :
    i =lexique.index(mot)
    lexique[i] = [mot, liste_mots.count(mot)]

>>> print lexique

[[u'm\xe9decin', 1], [u'malade', 2]]
```

3.3 - Exercises

1. Write a Python program for each of the following specifications.

a) Insert a word into a sorted list of words according the alphabetic order. The list must remain sorted and if the word is already present in the list, nothing is done.

b) Check that a list is a sublist of another list (its elements must be present in the second list in the same order).

c) Split a text in the form of a string into a list of sentences, each sentence being represented as a list of words (punctuation signs are taken as words).

3.3 - Solution of the exercises

1. Programs:

a)

```
list_words =input("Enter a list of words sorted according the alphabetic order ! \n")
word = input("Enter a word to insert ! \n")
k=0
insertion = False
while k < len(list_words) and not insertion:
    w=list_words[k]
    if w > word:
        list_words.insert(k-1,word)
        insertion= True
    elif w==word:
        insertion=True
    k+=1
print "list after insertion of word  ", word, ": ", list_words
```

3.3 - Solution of the exercises

1. Programs:

b)

```
list1= input("Enter a first list !\n")
list2= input("Enter a second list !\n")
correct = True
l1 = len(list1)
k1=0
k2=0
while k1 < l1 and correct:
    e1=list1[k1]
    try: # We try to run the block just below
        k2 = list2.index(e1,k2)+1
    except ValueError:#If there is an error of type "ValueError" in the execution of the block just
          above, the block just below is executed
        correct= False
    k1 +=1
if correct:
    print "The first list is a sublist of the second list."
else:
    print "The first list is not a sublist of the second list."
```

3.3 - Solution of the exercises

1. Programs:

c)

```
text= input("Enter a text !\n")
list_words = text.split()
sentence_stop = ['.', ';',':','?','!']
list_sentences =[]
sentence =[]
for w in list_words:
    sentence.append(w)
    if w in sentence_stop:
        list_sentences.append(sentence)
        sentence=[]
print "List of sentences: \n"
for s in list_sentences :
    print s

#check the program with the sentence : "Where is John ? I don't know , maybe at
school . Ok ! "
```

3.4 - Tuples

- Tuples are sequences of Python objects, which are possibly heterogeneous. They have the type **tuple** and the following syntax: $o_1, o_2, \ldots, o_n$ or $(o_1, o_2, \ldots, o_n)$. The empty tuple is represented with *()* .

- Like strings and lists, tuples are compound objects of type **sequence**. Thus + and * operators apply and tuple elements are addressable, as well as sub-tuples with the operation of slicing.

- Unlike lists, tuples are **unmutable**: their elements are not modifiable individually. Tuples are used instead of lists, to avoid errors and to economise storage space.

3.4 - Tuples

```
>>>tuple = (23, ‘bonjour’, [1,2])
>>>print tuple[2]
[1,2]
>>>tuple[1] = ‘bonsoir’
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    tuple[1] = 'bonsoir'
TypeError: 'tuple' object does not support item assignment
>>>print tuple[1:-1]
( ‘bonjour’,)
>>> tuple += 3,4
>>>print tuple
(23, ‘bonjour’, [1,2], 3, 4)
>>> u,v,x,y,z = tuple
>>> print v
‘bonjour’
```

```
>>>texte = u’Je suis tombé malade. Je n’ai pas travaillé.’
>>> liste_mots = texte.split()
>>> liste_mots_etiquetes = []
>>> for mot in liste_mots :
        liste_mots_etiquetes += [(mot, len(mot))]

>>> print liste_mots_etiquetes
[(u'Je', 2), (u'suis', 4), (u'tomb\xe9', 5), (u'malade.', 7),
    (u'Je', 2), (u'n\u2019ai', 4), (u'pas', 3), (u'travaill\xe9.',
    10)]
```

3.5 - Dictionaries

- Dictionaries are collections of Python objects which are possibly heterogeneous and which are addressable with a **key**. They have the type **dict**, which is an instanciation of the more general type **mapping**. They have the syntax : $\{c_1 : o_1, c_2 : o_2, \ldots, c_n : o_n\}$, where c_i are keys and o_i the corresponding values.

- The element of dictionary *d* associated with key *c*, called **entry**, is accessible with the expression *d[c].*

- Dictionaries are **mutable** objects. To add entry *v* associated with key *c* to dictionary *d*, or to update it, one uses the instruction *d[c] = v* . To delete the entry associated with key *c* from dictionary *d*, one uses the function call *del(d[c]).*

- As Python objects, dictionaries are associated with **methods**, their complete list is available at the Web address: *http://docs.python.org/lib/typesmapping.html*

3.5 - Dictionaries

```
>>>d = {‘a’ : 23, 2 : ‘bon’}
>>>print d[‘a’]
23
>>>d[‘b’] = ‘mal’ ; d[‘a’] = 12
>>>print d
{‘a’ : 12, 2 : ‘bon’, ‘b’ : ‘mal’}
>>> del(d[‘a’])
>>>print d
{2 : ‘bon’, ‘b’ : ‘mal’}
>>>
```

```
>>>texte = u’Jean le voit avec le patron’
>>> liste_mots = texte.split()
>>> freq = {}
>>> for mot in liste_mots :
    if freq.has_key(mot):
        freq[mot] += 1
    else:
        freq[mot] = 1

>>> print freq
{u’avec’ :  1, u’Jean’ : 1, u’le’ : 2, u’voit’ : 1, u’patron.', 2}
```

3.5 - Exercises

1. Write a Python program for each of the following specifications.

a) Count the number of occurrences of each character in a text using a dictionary (the text is given in the form of a string).

b) Reverse a French-English dictionary. For sake of simplification, we assume that the correspondence between French words and English words is one-to-one.

c) Memorise the position of each word in a text in a dictionary. The position is represented by a pair (position of the first character, position of the last character). If a word appears several times, all its positions must be memorised.

3.5 - Solution of the exercises

1. Python programs:

```
#a)
text = input("Enter a text !\n")
dico={}
for c in text:
    if dico.has_key(c):
        dico[c] +=1
    else:
        dico[c]=1
print "Number of characters of each type in the text:\n", dico

#b)
dico_fr_en = input("Enter a French-English dictionary !\n")
dico_en_fr={}
for fr_w in dico_fr_en:
    en_w=dico_fr_en[fr_w]
    dico_en_fr[en_w]= fr_w
print "English-French dictionary:\n", dico_en_fr
```

3.5 - Solution of the exercises

1. Python programs:

```
# c)

text = input("Entrez un texte !  ")
d = {}
word = ""
word_found = False
separator = ['\n', '\t', ' ']
word_beginning = 0
for k in range(len(text)):
    c = text[k]
    if word_found :
        if c in separator or k == len(text)-1:
            word_found = False
            if k == len(text)-1 :
                word += c
                k +=1
            if d.has_key(word) :
                d[word] +=[(word_beginning, k-1)]
            else :
                d[word] = [(word_beginning, k-1)]
        else :
            word += c
    elif not (c in separator) :
        word_found = True
        word_beginning = k
        word = c
    k += 1

print "Position of the words in the text:\n",d
```