

Algorísmica

Introducció als algorismes III

Jordi Vitrià

Col·leccions de dades i Python

Exemples de **col·leccions**:

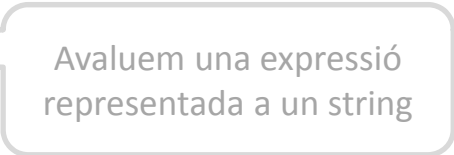
- Paraules d'un text.
- Estudiants d'un curs.
- Dades d'un experiment.
- Clients d'un negoci.
- Els gràfics que es poden dibuixar en una finestra.

Python ens dona suport per a la manipulació d'aquest tipus de dades.

Col·leccions de dades i Python

```
# average4.py
def main():
    sum = 0.0
    count = 0
    xStr = raw_input("Enter a number (<Enter> to quit) >> ")
    while xStr != "":
        x = eval(xStr)
        sum = sum + x
        count = count + 1
        xStr = raw_input("Enter a number (<Enter> to quit) >> ")
    print "\nThe average of the numbers is", sum / count

main()
```



Avaluem una expressió representada a un string

Suposem que també volem calcular la mediana i la desviació estàndard....

$$s = \sqrt{\frac{\sum (\bar{x} - x)^2}{n - 1}}$$

Necessitem guardar tots els nombres que han entrat.

Col·leccions de dades i Python

El que necessitem és emmagatzemar una col·lecció de coses (a priori no sabem quantes) en un “objecte”.

De fet, aquest tipus d’“objecte” ja l’hem fet servir, i es diu **llista**:

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> string.split("This is an ex-parrot!")
['This', 'is', 'an', 'ex-parrot!']
```

Una **llista** és una seqüència ordenada de coses.

$$S = s_0, s_1, s_2, s_3, \dots, s_{n-1}$$

Els elements estan indexats per subíndexos

Col·leccions de dades i Python

De fet les llistes i els *strings* són conceptualment molt semblants, i podem aplicar-hi operadors semblants:

Operator	Meaning
<seq> + <seq>	Concatenation
<seq> * <int-expr>	Repetition
<seq>[]	Indexing
len(<seq>)	Length
<seq>[:]	Slicing
for <var> in <seq>:	Iteration

La diferència és el que contenen. Les llistes **poden contenir qualsevol tipus de dades**, incloent “classes” definides pel programador. Les llistes són **mutables**, és a dir, es poden canviar sobre la mateixa estructura (**els strings no!**).

```
>>> myList = [34, 26, 15, 10]
>>> myList[2]
15
>>> myList[2] = 0
>>> myList
[34, 26, 0, 10]
>>> myString = "Hello World"
>>> myString[2]
'l'
>>> myString[2] = 'z'
Traceback (innermost last):
  File "<stdin>", line 1, in ?
TypeError: object doesn't support item assignment
```

Col·leccions de dades i Python

Les llistes en Python són **dinàmiques**, poden créixer i decreïxer durant l'execució del programa.

Les llistes en Python són **inhomogènies**, poden contenir tipus diferents de dades.

En resum, les llistes són seqüències mutables d'objectes arbitraris.

Es creen així:

```
odds = [1, 3, 5, 7, 9]
food = ["spam", "eggs", "back bacon"]
silly = [1, "spam", 4, "U"]
empty = []
```

Col·leccions de dades i Python

Podem crear llistes d'objectes idèntics així:

```
zeroes = [0] * 50
```

O afegir-hi/borra-hi coses:

```
nums = []
x = input('Enter a number: ')
while x >= 0:
    nums.append(x)
    x = input("Enter a number: ")
```

```
>>> myList
[34, 26, 0, 10]
>>> del myList[1]
>>> myList
[34, 0, 10]
>>> del myList[1:3]
>>> myList
[34]
```

Col·leccions de dades i Python

Method	Meaning
<code><list>.append(x)</code>	Add element x to end of list.
<code><list>.sort()</code>	Sort the list. A comparison function may be passed as parameter.
<code><list>.reverse()</code>	Reverses the list.
<code><list>.index(x)</code>	Returns index of first occurrence of x.
<code><list>.insert(i,x)</code>	Insert x into list at index i. (Same as <code>list[i:i] = [x]</code>)
<code><list>.count(x)</code>	Returns the number of occurrences of x in list.
<code><list>.remove(x)</code>	Deletes the first occurrence of x in list.
<code><list>.pop(i)</code>	Deletes the ith element of the list and returns its value.
<code>x in <list></code>	Checks to see if x is in the list (returns a Boolean).

```
>>> z=[0] * 10
>>> z
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>> z.insert(1,1)
>>> z
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
>>> 1 in z
True
>>> y = z.pop(1)
>>> y
1
>>> z
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```


Col·leccions de dades i Python

Amb el que sabem podem reescriure el codi del càlcul estadístic.
Primer, recollim les dades de l'usuari:

```
def getNumbers():  
    nums = []      # start with an empty list  
  
    # sentinel loop to get numbers  
    xStr = raw_input("Enter a number (<Enter> to quit) >> ")  
    while xStr != "":  
        x = eval(xStr)  
        nums.append(x)    # add this value to the list  
        xStr = raw_input("Enter a number (<Enter> to quit) >> ")  
    return nums
```

Col·leccions de dades i Python

Després calculem la mitja:

```
def mean(nums):  
    sum = 0.0  
    for num in nums:  
        sum = sum + num  
    return sum / len(nums)
```

I el programa original queda:

```
def main():  
    data = getNumbers()  
    print 'The mean is', mean(data)
```

Un cop tenim la mitja podem calcular la desviació :

```
def stdDev(nums, xbar):  
    sumDevSq = 0.0  
    for num in nums:  
        dev = xbar - num  
        sumDevSq = sumDevSq + dev * dev  
    return sqrt(sumDevSq / (len(nums) - 1))
```

$$s = \sqrt{\frac{\sum (\bar{x} - x)^2}{n - 1}}$$

Col·leccions de dades i Python

La mediana és una mica més complicada.

```
sort the numbers into ascending order
if the size of data is odd:
    median = the middle value
else:
    median = the average of the two middle values
return median
```

```
def median(nums):
    nums.sort()
    size = len(nums)
    midPos = size / 2
    if size % 2 == 0:
        median = (nums[midPos] + nums[midPos-1]) / 2.0
    else:
        median = nums[midPos]
    return median
```

Col·leccions de dades i Python

```
def main():  
    print 'This program computes mean, median and standard deviation.'  
  
    data = getNumbers()  
    xbar = mean(data)  
    std = stdDev(data, xbar)  
    med = median(data)  
  
    print '\nThe mean is', xbar  
    print 'The standard deviation is', std  
    print 'The median is', med
```

Col·leccions de dades i Python

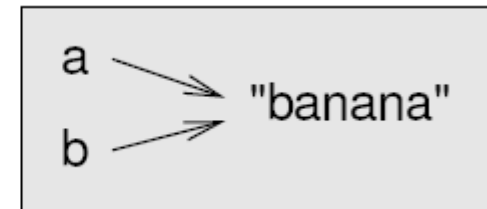
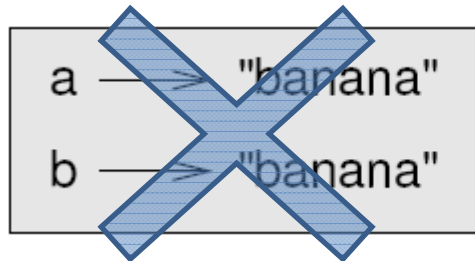
Si executem:

```
a = "banana"  
b = "banana"
```

a i b “apunten” a un *string* amb el mateix valor, però és el mateix *string*?

Cada objecte té un **identificador** únic, que podem obtenir amb la funció id:

```
>>> id(a)  
135044008  
>>> id(b)  
135044008
```

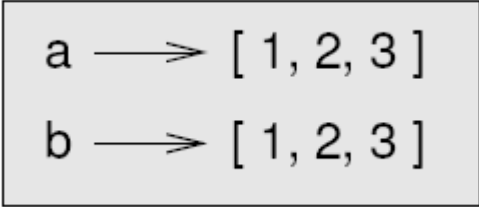


Per tant, en aquest cas Python ha creat una estructura “banana” i les dues variables en fan referència.

Col·leccions de dades i Python

Les **l·listes** funcionen diferent (a i b tenen el mateix valor però no es refereixen al mateix objecte):

```
>>> a = [1, 2, 3]
>>> b = [1, 2, 3]
>>> id(a)
135045528
>>> id(b)
135041704
```

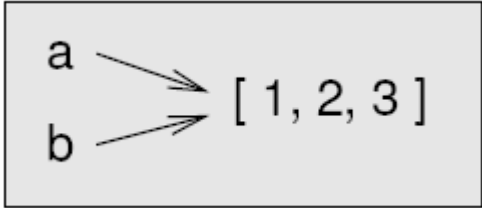


a → [1, 2, 3]
b → [1, 2, 3]

The diagram illustrates that variables 'a' and 'b' each point to a distinct memory location, both containing the list [1, 2, 3].

Com que les variables es refereixen a objectes, si fem referir una variable a una altra tenim:

```
>>> a = [1, 2, 3]
>>> b = a
```



a → [1, 2, 3]
b → [1, 2, 3]

The diagram illustrates that after the assignment 'b = a', both variables 'a' and 'b' point to the same memory location, which contains the list [1, 2, 3].

Col·leccions de dades i Python

Com que la llista té dos noms, direm que té un **alias**:

```
>>> b[0] = 5
>>> print a
[5, 2, 3]
```

Això és perillós per objectes mutables!!! Pels immutables no hi ha problema (*strings*).

El **clonatge** és una tècnica per la que fem una còpia de l'objecte en si, no de la referència. Pel cas de les llistes ho podem fer així:

```
>>> a = [1, 2, 3]
>>> b = a[:]
>>> print b
[1, 2, 3]
```

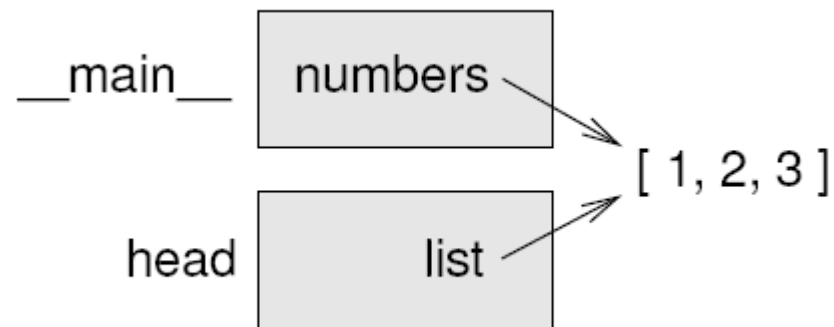
```
>>> b[0] = 5
>>> print a
[1, 2, 3]
```

Col·leccions de dades i Python

Si passem una llista com argument d'una funció, **passem una referència, no una còpia**. Considerem aquesta funció:

```
def head(list):  
    return list[0]
```

```
>>> numbers = [1, 2, 3]  
>>> head(numbers)  
1
```



Col·leccions de dades i Python

Considerem aquesta altra funció:

```
def deleteHead(list):  
    del list[0]
```

```
>>> numbers = [1, 2, 3]  
>>> deleteHead(numbers)  
>>> print numbers  
[2, 3]
```

Si retornem una llista també retornem una referència:

```
def tail(list):  
    return list[1:]  
  
>>> numbers = [1, 2, 3]  
>>> rest = tail(numbers)  
>>> print rest  
[2, 3]
```

Com que la llista **s'ha creat amb : és una nova llista**. Qualsevol modificació de rest no té efectes a numbers.

Col·leccions de dades i Python

```
>>> numbers = [1, 2, 3]
>>> def test(l):
...     return l.reverse()
...
>>> test(numbers)
>>> numbers
[3, 2, 1]
>>>
```

Col·leccions de dades i Python

Una **llista nidada** és una llista que apareix com a element d'una altra llista.

```
>>> list = ["hello", 2.0, 5, [10, 20]]
```

Per obtenir un element d'una llista nidada ho podem fer de dues maneres:

```
>>> elt = list[3]
>>> elt[0]
10

>>> list[3][1]
20
```

Les llistes nidades es fan servir per representar matrius

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

```
>>> matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> matrix[1]
[4, 5, 6]

>>> matrix[1][1]
5
```

Col·leccions de dades i Python

Python ens proporciona un altre tipus de col·lecció no gaire estàndard, però molt útil: els **diccionaris**.

La raó de la seva existència és que no sempre serà possible accedir a una dada pel seu índex, sinó per exemple, pel seu valor (p.e. pel DNI d'un conjunt d'empleats).

És a dir, volem accedir a un valor per una **clau**.

Una col·lecció que ens permet això es diu un “*mapping*” (altres llenguatges ho anomenen *taules hash* o *vectors associatius*).

Python les crea així:

```
>>> passwd = {"guido": "superprogrammer", "turing": "genius", "bill": "monopoly"}
```

Col·leccions de dades i Python

I ens permet accedir-hi així:

```
>>> passwd["guido"]  
'superprogrammer'  
>>> passwd["bill"]  
'monopoly'
```

Els diccionaris són mutables:

```
>>> passwd["bill"] = "bluescreen"  
>>> passwd  
{ 'turing': 'genius', 'bill': 'bluescreen', 'guido': 'superprogrammer' }
```

(els diccionaris no tenen ordre, i Python els imprimeix amb un ordre propi, no el que hem entrat!)

Col·leccions de dades i Python

Podem entrar-hi dades:

```
>>> passwd['newuser'] = 'ImANewbie'
>>> passwd
{'turing': 'genius', 'bill': 'bluescreen', \
 'newuser': 'ImANewbie', 'guido': 'superprogrammer'}
```

..des d'un fitxer:

```
>>> f = open('passwords.txt', 'r')
>>> for line in f.readlines():
...     print line,
...
jordi vitria
bill clinton
>>>
```



Llegeix strings

```
passwd = {}
for line in open('passwords', 'r').readlines():
    user, pass = string.split(line)
    passwd[user] = pass
```

Col·leccions de dades i Python

Method	Meaning
<code><dict>.has_key(<key>)</code>	Returns true if dictionary contains the specified key, false if it doesn't.
<code><dict>.keys()</code>	Returns a list of the keys.
<code><dict>.values()</code>	Returns a list of the values.
<code><dict>.items()</code>	Returns a list of tuples (key,value) representing the key-value pairs.
<code>del <dict>[<key>]</code>	Delete the specified entry.
<code><dict>.clear()</code>	Delete all entries.

```
>>> passwd.keys()
['turing', 'bill', 'newuser', 'guido']
>>> passwd.values()
['genius', 'bluescreen', 'ImANewbie', 'superprogrammer']
>>> passwd.items()
[('turing', 'genius'), ('bill', 'bluescreen'), ('newuser', 'ImANewbie'), \
 ('guido', 'superprogrammer')]
>>> passwd.has_key('bill')
1
>>> passwd.has_key('fred')
0
>>> passwd.clear()
>>> passwd
{}
```

Col·leccions de dades i Python

Hi ha una altra classe de col·lecció a Python que és semblant a la llista, però que és immutable.: la **tupla**.

```
>>> tuple = 'a', 'b', 'c', 'd', 'e'
```

Que també es pot (i es sol) escriure com:

```
>>> tuple = ('a', 'b', 'c', 'd', 'e')
```

Si només té un element hem de posar-hi una coma, sinó crea un string!

```
>>> t1 = ('a',)  
>>> type(t1)  
<type 'tuple'>
```

```
>>> t2 = ('a')  
>>> type(t2)  
<type 'str'>
```


Col·leccions de dades i Python

Les operacions són les mateixes que per les llistes (tenint en compte que són **immutablees!**):

```
>>> tuple = ('a', 'b', 'c', 'd', 'e')
>>> tuple[0]
'a'
```

```
>>> tuple[1:3]
('b', 'c')
```

```
>>> tuple[0] = 'A'
TypeError: object doesn't support item assignment
```

```
>>> tuple = ('A',) + tuple[1:]
>>> tuple
('A', 'b', 'c', 'd', 'e')
```

Col·leccions de dades i Python

Exemple: com fer l'estadística de les paraules que apareixen en un document.

```
# wordfreq.py
import string

def compareItems((w1,c1), (w2,c2)):
    if c1 > c2:
        return - 1
    elif c1 == c2:
        return cmp(w1, w2)
    else:
        return 1

def main():
    print "This program analyzes word frequency in a file"
    print "and prints a report on the n most frequent words.\n"

    # get the sequence of words from the file
    fname = raw_input("File to analyze: ")
    text = open(fname,'r').read()
    text = string.lower(text)
    for ch in '!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~':
        text = string.replace(text, ch, ' ')
    words = string.split(text)

    # construct a dictionary of word counts
```

Col·leccions de dades i Python

```
counts = {}
for w in words:
    try:
        counts[w] = counts[w] + 1
    except KeyError:
        counts[w] = 1

# output analysis of n most frequent words.
n = input("Output analysis of how many words? ")
items = counts.items()
items.sort(compareItems)
for i in range(n):
    print "%-10s%5d" % items[i]

if __name__ == '__main__': main()
```