

**Algorísmica**

# **Introducció als algorismes III**

**Mireia Ribera i Jordi Vitrià**

# Col·leccions de dades i Python

## Exemples de **col·leccions**:

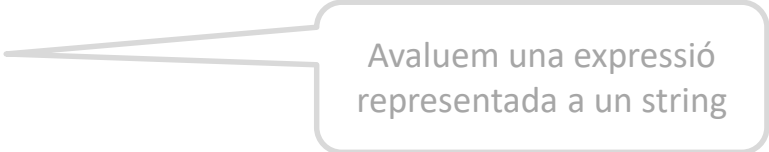
- Paraules d'un text.
- Estudiants d'un curs.
- Dades d'un experiment.
- Clients d'un negoci.
- Els gràfics que es poden dibuixar en una finestra.

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 = input("Enter a number (<Enter> to quit) >> ")  
    while xStr != "":  
        x = eval(xStr)  
        sum = sum + x  
        count = count + 1  
        xStr = input("Enter a number (<Enter> to quit) >> ")  
    print("The average of the numbers is", sum / count)
```



Avaluem una expressió representada a un string

```
main()
```

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

$$s = \sqrt{\frac{\sum_i (x_i - \hat{x})^2}{n - 1}}$$

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**:

```
list(range(10))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
a = 'This is an ex-parrot'  
a.split()
```

```
['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!).

# Col·leccions de dades i Python

```
myList = [34, 45, 67, 78]  
myList[2]
```

67

```
myList[2] = 0  
myList
```

[34, 45, 0, 78]

```
myString = "Hello World"  
myString[2]
```

'l'

```
myString[2] = 'k'
```

```
-----  
---  
TypeError                                Traceback (most recent call la  
st)  
<ipython-input-22-4059e3d608ea> in <module>()  
----> 1 myString[2] = 'k'  
  
TypeError: 'str' object does not 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/borrar-hi coses:

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

```
myList
```

```
[34, 45, 0, 78]
```

```
del myList[1]
```

```
myList
```

```
[34, 0, 78]
```

```
del myList[1:3]
```

```
myList
```

```
[34]
```



# Col·leccions de dades i Python

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

# Col·leccions de dades i Python

```
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 = []  
    # sentinel loop to get numbers  
    xStr = input("Enter a number (<Enter> to quit) >>")  
    while xStr != "":  
        x = eval(xStr)  
        nums.append(x) # add this value to the list  
        xStr = 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 math.sqrt(sumDevSq/(len(nums)-1))
```

$$s = \sqrt{\frac{\sum_i (x_i - \hat{x})^2}{n - 1}}$$

# Col·leccions de dades i Python

La mediana és una mica més complicada.

```
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
```

Aquesta implementació de la mediana no és eficient!  
Més endavant veurem com fer-ho de forma eficient.

# 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("The mean is", xbar)  
    print("The standard deviation is", std)  
    print("The median is", med)
```

# Col·leccions de dades i Python

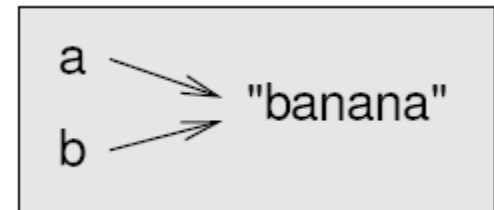
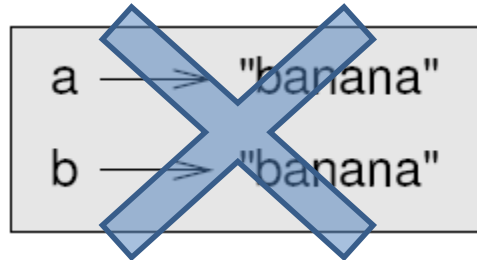
Si executem:

```
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]
print(id(a), id(b))
```

2388708167816 2388708169480

a → [ 1, 2, 3 ]  
b → [ 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
print(id(a), id(b))
```

2388708167752 2388708167752

a → [ 1, 2, 3 ]  
b → [ 1, 2, 3 ]



# Col·leccions de dades i Python

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

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

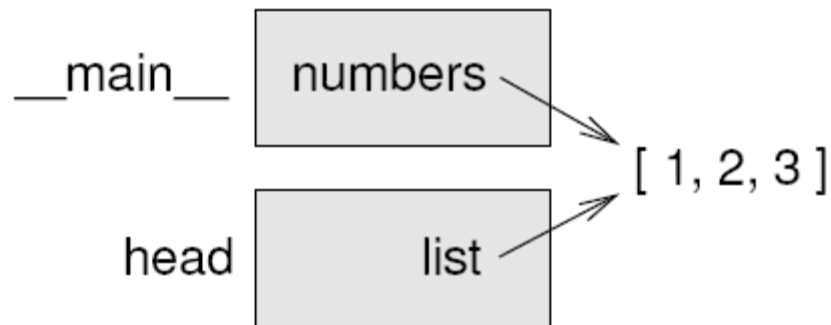
# Col·leccions de dades i Python

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

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

```
a = [1,2,3]  
head(a)
```

1



# Col·leccions de dades i Python

Considerem aquesta altra funció:

```
def deleteHead(list):  
    del list[0]  
  
a = [1,2,3]  
deleteHead(a)  
print(a)
```

[2, 3]

Si retornem una llista també retornem una referència:

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

[2, 3]

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

# Col·leccions de dades i Python

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

test(numbers)
print(numbers)
```

[3, 2, 1]

# Col·leccions de dades i Python

Una **llista imbricada** 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 imbricada ho podem fer de dues maneres:

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

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

Les llistes imbricades es fan servir per representar matrius

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

```
[4, 5, 6]
```

```
matrix[1][1]
```

```
5
```

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

# Col·leccions de dades i Python

Python ens proporciona un altre tipus de col·lecció 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, per algun **valor** que el defineix (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 = {'a':'A', 'b':'B', 'c':'C'}  
passwd['a']  
  
'A'
```

Els diccionaris són mutables:

```
passwd['a'] = 'AA'  
passwd  
  
{ 'a': 'AA', 'b': 'B', 'c': 'C' }
```

(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 = {'a':'A', 'b':'B', 'c':'C'}  
passwd['d']='D'  
passwd
```

```
{'a': 'A', 'b': 'B', 'c': 'C', 'd': 'D'}
```

..des d'un fitxer: 

```
f = open("passwords.txt", "r")  
for line in f.readlines():  
    print(line,
```

```
    turing genius  
    bill bluescreen  
    newuser ImANewbie  
    guido superprogrammer
```

```
passwd = {}  
for line in open("passwords.txt", "r").readlines():  
    user, passw = line.split()  
    passwd[user] = passw  
print(passwd)
```

Llegeix strings



# Col·leccions de dades i Python

```
p = {"a": "A", "b": "B", "c": "C", 36: "D"}
```

```
for i in p.keys():  
    print(i, end=",")
```

a,b,c,36,

```
for i in p.values():  
    print(i, end=",")
```

A,B,C,D,

```
for i in p.items():  
    print(i, end=",")
```

('a', 'A'),('b', 'B'),('c', 'C'),(36, 'D'),

```
list(p.values())
```

['A', 'B', 'C', 'D']

```
"a" in p
```

True

```
p.clear()
```

p

{}

# 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 se 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',)    t2 = ('a')
```

```
type(t1)      type(t2)  
tuple         str
```

# Col·leccions de dades i Python

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

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

```
'a'
```

```
tupla[1:3]
```

```
('b', 'c')
```

```
tupla[0]='a'
```

```
-----  
---  
TypeError                                Traceback (most recent call last)  
st)  
<ipython-input-67-a0783c5a8121> in <module>()  
----> 1 tupla[0]='a'
```

```
TypeError: 'tuple' object does not support item assignment
```

# Col·leccions de dades i Python

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

```
def main():
    print("This program analyzes word frequency in a file",)
    print("and prints a report on the n most frequent words.")
    # get the sequence of words from the file
    fname = input("File to analyze: ")
    text = open(fname, 'r').read()
    text = text.lower()
    for ch in '!"#$%&()*+,-./:;<=>?@[\\]^_`{|}':
        text.replace(ch, ' ')
    words = text.split()
    # construct a dictionary of word counts
    counts = {}
    for w in words:
        if w in counts:
            counts[w] = counts[w] + 1
        else:
            counts[w] = 1
    # output analysis of n most frequent words
    n = eval(input("Output analysis of how many words? "))
    listfreq = []
    for w in counts:
        listfreq.append((counts[w], w))
    print(listfreq)
    listfreq.sort(reverse=True)
    print(listfreq)
    for i in range(n):
        print(listfreq[i][1], listfreq[i][0])
```