Programació per a *Data Science*

Unitat 4: Llibreries científiques en Python - NumPy

Instruccions d'ús

A continuació es presentaran explicacions i exemples d'ús de la llibreria NumPy. Recordeu que podeu anar executant els exemples per obtenir-ne els resultats.

Primers passos

Importarem la llibreria:

In [1]:
```
# A la línia següent, importem NumPy i li donem un nom més curt
# perquè ens sigui més còmode fer les crides
import numpy as np
```

A NumPy, l'objecte bàsic és una llista multidimensional de nombres (normalment) del mateix tipus.

In [2]:
```
# Exemple bàsic, un punt a l'espai:
p = np.array([1, 2, 3])
```

A NumPy, a les dimensions se les coneix amb el nom d'eixos (*axes*), i al nombre d'eixos, rang (*rank*). *Array* és un àlies per referir-se al tipus d'objecte *numpy.ndarray*.

Algunes propietats importants dels *arrays* són les següents:

- **ndarray.ndim**: el nombre d'eixos de l'objecte *array* (matriu).
- **ndarray.shape**: una tupla de nombres enters indicant la longitud de les dimensions de la matriu.
- **ndarray.size**: el nombre total d'elements de la matriu.

In [3]:
```
# Crearem una matriu bidimensional 3x2 (tres files, dues columnes).
a = np.arange(3*2) # Creem un array unidimensional de sis elements
print 'Array unidimensional:'
print a
a = a.reshape(3,2) # Li donem "forma" de matrix 3x2.
print 'Matriu 3x2:'
print a
```

```
Array unidimensional:
[0 1 2 3 4 5]
Matriu 3x2:
[[0 1]
 [2 3]
 [4 5]]
```

In [4]:
```
# La dimensió és 2.
a.ndim
```

Out[4]: 2

In [5]:
```
# Longitud de les dimensions.
a.shape
```

Out[5]: (3, 2)

In [6]:
```
# El nombre total d'elements:
a.size
```

Out[6]: 6

A l'hora de crear *arrays*, tenim diferents opcions:

In [7]:
```
# Creem un array (vector) de deu elements:
z = np.zeros(10)
print z
```

```
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
```

In [8]:
```
# Podem canviar qualsevol dels valors d'aquest vector accedint a la seva posició:
z[4] = 5.0
z[-1] = 0.1
print z
```

```
[ 0.   0.   0.   0.   5.   0.   0.   0.   0.   0.1]
```

In [9]:
```
# La funció 'arange' ens permet definir diferents opcions, com el punt d'inici i el de final:
a = np.arange(10,20)
print a
```

```
[10 11 12 13 14 15 16 17 18 19]
```

In [10]:
```
# L'últim argument ens permet utilitzar un pas de 2:
a = np.arange(10,20,2)
print a
```

```
[10 12 14 16 18]
```

In [11]:
```
# Podem crear arrays des de llistes de Python de diverses dimensions:
a = np.array([[1, 2, 3], [4, 5, 6]])
print a
```

```
[[1 2 3]
 [4 5 6]]
```

Operacions amb matrius

NumPy implementa totes les operacions habituals amb matrius.

In [12]:
```
A = np.array([[1,0], [0,1]])
B = np.array([[1,2], [3,4]])
```

In [13]:
```
# Suma de matrius
print A+B
```

```
[[2 2]
 [3 5]]
```

In [14]:
```
# Resta de matrius
print A-B
```

```
[[ 0 -2]
 [-3 -3]]
```

In [15]:
```
# Multiplicació element per element
print A*B
```

```
[[1 0]
 [0 4]]
```

In [16]:
```
# Multiplicació de matrius
print A.dot(B)
```

```
[[1 2]
 [3 4]]
```

In [17]:
```
# Potència
print B**2
```

```
[[ 1  4]
 [ 9 16]]
```

Slicing i iteració

Els *arrays* en NumPy suporten la tècnica de *slicing* de Python:

In [18]:
```
# Definim un array de 0 a 9.
a = np.arange(10)
# Obtenim els 5-2 elements des de la posició 3 de l'array (els índexs comencen a 0).
print a[2:5]
```

```
[2 3 4]
```

In [19]:
```
# Tots els elements a partir de la tercera posició:
a[2:]
```

Out[19]:
```
array([2, 3, 4, 5, 6, 7, 8, 9])
```

In [20]:
```
# Podem iterar per cada element de l'array:
for i in a:
    print i
```

```
0
1
2
3
4
5
6
7
8
9
```

In [21]:
```
# Ara definim un array multidimensional.
A = np.arange(18).reshape(6,3)
print A
```

```
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]
 [12 13 14]
 [15 16 17]]
```

In [22]:
```
# Obtenim fins a la cinquena fila.
print A[:5]
```

```
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]
 [12 13 14]]
```

In [23]:
```
# Igualem tots els elements a 0 (l'operador ... afegeix tots els : necessaris).
A[...] = 0
print A
```

```
[[0 0 0]
 [0 0 0]
 [0 0 0]
 [0 0 0]
 [0 0 0]
 [0 0 0]]
```

Exemple: el joc de la vida de Conway

El joc de la vida és un exemple clàssic d'autòmat cel·lular creat el 1970 pel famós matemàtic John H. Conway. Al problema clàssic, es representen en una matriu bidimensional cèl·lules que viuran o moriran depenent del nombre de veïns en un determinat pas de la simulació. Cada cèl·lula té vuit veïns (les caselles adjacents en un tauler bidimensional) i pot estar viva (1) o morta (0). Les regles clàssiques entre transició vida/mort són les següents:

- Una cèl·lula morta amb exactament tres cèl·lules veïnes vives al torn següent estarà viva.
- Una cèl·lula viva amb dues o tres cèl·lules veïnes vives segueix viva, en un altre cas mor o roman morta (solitud en cas d'un número més petit que 2, superpoblació si és major a 3).

Aquest problema (o joc) té moltes variants depenent de les condicions inicials, si el tauler (o món) té vores o no, o si bé les regles de vida o mort són alterades.

A continuació teniu un exemple del joc clàssic implementat a NumPy:

In [24]:
```
# Autor: Nicolas Rougier
# Font: http://www.labri.fr/perso/nrougier/teaching/numpy.100/

SIZE = 10
STEPS = 10

def iterate(Z):
    # Comptem veïns:
    N = (Z[0:-2,0:-2] + Z[0:-2,1:-1] + Z[0:-2,2:] +
         Z[1:-1,0:-2]                + Z[1:-1,2:] +
         Z[2:  ,0:-2] + Z[2:  ,1:-1] + Z[2:  ,2:])

    # Apliquem les regles:
    birth = (N==3) & (Z[1:-1,1:-1]==0)
    survive = ((N==2) | (N==3)) & (Z[1:-1,1:-1]==1)
    Z[...] = 0
    Z[1:-1,1:-1][birth | survive] = 1
    return Z

#Creem un tauler amb cèl·lules vives o mortes de manera aleatòria.
Z = np.random.randint(0,2,(SIZE, SIZE))
# Simulem durant els passos indicats:
for i in range(STEPS):
    Z = iterate(Z)
# Mostrem el tauler en el pas final:
print Z
```

```
[[0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]]
```