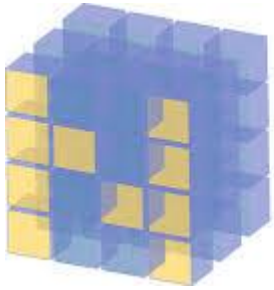
Matriz

Multidimensional

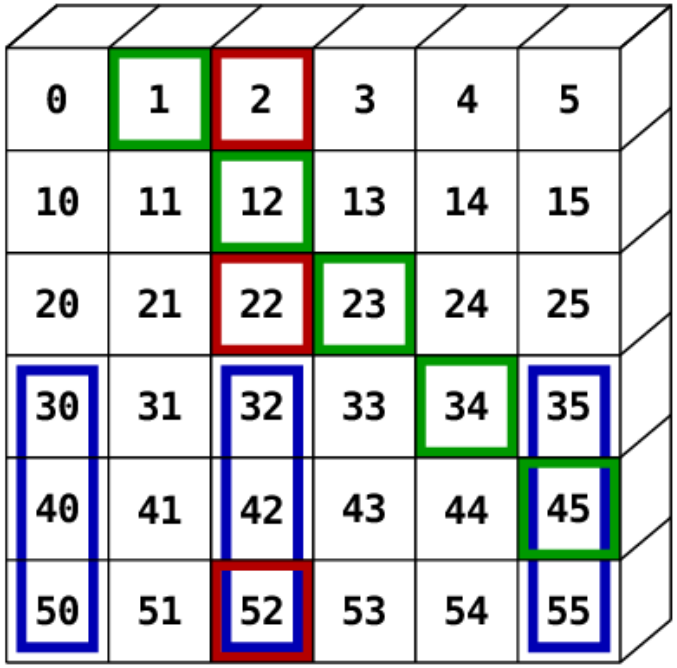


NumPy

```
>>> a[(0,1,2,3,4), (1,2,3,4,5)]
array([1, 12, 23, 34, 45])

>>> a[3:, [0,2,5]]
array([[30, 32, 35],
       [40, 42, 45],
       [50, 52, 55]])

>>> mask = np.array([1,0,1,0,0,1], dtype=bool)
>>> a[mask, 2]
array([2, 22, 52])
```



NumPy

Python es un gran lenguaje de programación de propósito general por sí mismo, pero con la ayuda de algunas bibliotecas populares (numpy, scipy, matplotlib) se convierte en un entorno poderoso para el cálculo científico.

Numpy es la biblioteca central para computación científica en Python. Proporciona un objeto de **matriz multidimensional de alto rendimiento** y herramientas para trabajar con estas matrices.

https://www.youtube.com/watch?v=Zhr0rWYuqfk

Redes neuronales convolucionales CS231n para reconocimiento visual

Python Numpy Tutorial

Este tutorial fue contribuido por Justin Johnson .

http://cs231n.github.io/python-numpy-tutorial/#numpy-arrays

Arrays Numpy

Una matriz numpy es una cuadrícula de valores, todos del mismo tipo, y está indexada por una tupla de enteros no negativos.
El número de dimensiones es el *rango* de la matriz;
la *forma* (shape) una matriz es una tupla de enteros que da el tamaño de la matriz a lo largo de cada dimensión.

Podemos inicializar matrices numpy desde listas anidadas de Python y acceder a elementos mediante corchetes:

```
import numpy as np

a = np.array([1, 2, 3])   # Create a rank 1 array
print(type(a))            # Prints "<class 'numpy.ndarray'>"
print(a.shape)            # Prints "(3,)"
print(a[0], a[1], a[2])   # Prints "1 2 3"
a[0] = 5                  # Change an element of the array
print(a)                  # Prints "[5, 2, 3]"

b = np.array([[1,2,3],[4,5,6]])    # Create a rank 2 array
print(b.shape)                     # Prints "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0])   # Prints "1 2 4"
```

Numpy también proporciona muchas funciones para crear matrices:

```
import numpy as np

a = np.zeros((2,2))   # Create an array of all zeros
print(a)              # Prints "[[ 0.  0.]
                      #          [ 0.  0.]]"

b = np.ones((1,2))    # Create an array of all ones
print(b)              # Prints "[[ 1.  1.]]"

c = np.full((2,2), 7)  # Create a constant array
print(c)               # Prints "[[ 7.  7.]
                       #          [ 7.  7.]]"

d = np.eye(2)         # Create a 2x2 identity matrix
print(d)              # Prints "[[ 1.  0.]
                      #          [ 0.  1.]]"

e = np.random.random((2,2))  # Create an array filled with random values
print(e)                     # Might print "[[ 0.91940167  0.08143941]
                             #              [ 0.68744134  0.87236687]]"
```

```
import numpy as np

x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

# Elementwise sum; both produce the array
# [[ 6.0  8.0]
#  [10.0 12.0]]
print(x + y)
print(np.add(x, y))

# Elementwise difference; both produce the array
# [[-4.0 -4.0]
#  [-4.0 -4.0]]
print(x - y)
print(np.subtract(x, y))

# Elementwise product; both produce the array
# [[ 5.0 12.0]
#  [21.0 32.0]]
print(x * y)
print(np.multiply(x, y))
```

[1, 2] [5, 6]

[3, 4] [7, 8]

A diferencia de MATLAB, * es la multiplicación por elementos, no la multiplicación de matrices.

En su lugar, utilizamos la función **dot** para calcular los productos internos de los vectores, para multiplicar un vector por una matriz y para multiplicar las matrices.

dot está disponible como una función en el módulo numpy y como un método de instancia de objetos de matriz:

```
# Matrix / vector product; both produce the rank 1 array [29 67]
print(x.dot(v))
print(np.dot(x, v))

# Matrix / matrix product; both produce the rank 2 array
# [[19 22]
#  [43 50]]
print(x.dot(y))
print(np.dot(x, y))
```

Numpy proporciona muchas funciones útiles para realizar cálculos en arreglos; Una de las más útiles es `sum`:

```
import numpy as np

x = np.array([[1,2],[3,4]])

print(np.sum(x))  # Compute sum of all elements; prints "10"
print(np.sum(x, axis=0))  # Compute sum of each column; prints "[4 6]"
print(np.sum(x, axis=1))  # Compute sum of each row; prints "[3 7]"
```

Práctica

- Experimenta alguno de los ejemplos de esta web

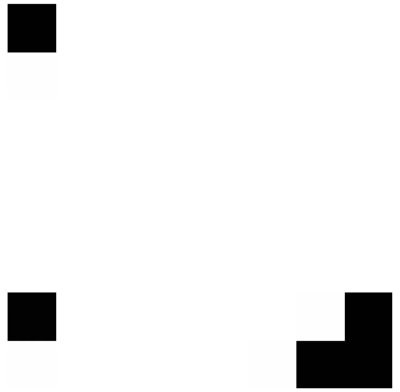
  http://cs231n.github.io/python-numpy-tutorial/#numpy-arrays

- Visualiza el vídeo de introducción a NumPy

  https://www.youtube.com/watch?v=8JfDAm9y_7s

Juego de la vida de Conway con Python usando NumPy

- El tablero es una rejilla de celdas cuadradas que tienen dos posibles estados: **viva** y **muerta**.
- Cada célula tiene **ocho** células vecinas: se cuentan también las de las diagonales.
- En cada paso, todas las células se actualizan instantáneamente teniendo en cuenta la siguiente regla:
  - Cada célula viva con 2 o 3 células vecinas vivas sobrevive.
  - Cada célula con 4 o más células vecinas vivas muere por superpoblación. Cada célula con 1 o ninguna célula vecina viva muere por soledad.
  - Cada célula muerta con 3 células vecinas vivas nace.



La clave para calcular el número de células vivas va a estar en la función roll, que toma un array de NumPy y desplaza todos los elementos en la dirección indicada.

https://pybonacci.org/2012/11/30/juego-de-la-vida-de-conway-con-python-usando-numpy/