Librerías científicas en Python:

NumPy

NumPy

- import numpy
- Paquete de python que implementa ***colecciones*** de datos de forma eficiente.
- Similar a las listas estandar.
- Internamente, garantiza que los datos están contiguos en la memoria → rápido acceso.
- Arrays de numpy son considerablemente más rápidos que listas para operaciones numéricas.
- Internamente utiliza tipos de datos de C

NumPy

```
In [6]: numeros = [1,2,3,4]

In [7]: numeros_numpy = numpy.array(numeros)

In [8]: type(numeros)
Out[8]: list

In [9]: type(numeros_numpy)
Out[9]: numpy.ndarray
```

NumPy

- numpy también ofrece mucha funcionalidad extra para aplicaciones numéricas:

```
In [10]: numeros_numpy.
numeros_numpy.T                 numeros_numpy.min
numeros_numpy.all               numeros_numpy.nbytes
numeros_numpy.any               numeros_numpy.ndim
numeros_numpy.argmax            numeros_numpy.newbyteorder
numeros_numpy.argmin            numeros_numpy.nonzero
numeros_numpy.argsort           numeros_numpy.prod
numeros_numpy.astype            numeros_numpy.ptp
numeros_numpy.base              numeros_numpy.put
numeros_numpy.byteswap          numeros_numpy.ravel
numeros_numpy.choose            numeros_numpy.real
numeros_numpy.clip              numeros_numpy.repeat
numeros_numpy.compress          numeros_numpy.reshape
numeros_numpy.conj              numeros_numpy.resize
numeros_numpy.conjugate         numeros_numpy.round
numeros_numpy.copy              numeros_numpy.searchsorted
numeros_numpy.ctypes            numeros_numpy.setasflat
numeros_numpy.cumprod           numeros_numpy.setfield
numeros_numpy.cumsum            numeros_numpy.setflags
numeros_numpy.data              numeros_numpy.shape
```

NumPy

- Se puede saber el tipo de los elementos del array con el atributo dtype:

```
In [17]: numeros_numpy.dtype
Out[17]: dtype('int64')

In [18]: floats_numpy = numpy.array([1.,2.,3,10,5.5])

In [19]: floats_numpy.dtype
Out[19]: dtype('float64')
```

Creando Arrays en NumPy

- Formas abreviadas de crear arreglos:

```
In [32]: ceros = numpy.zeros(100)
```

```
In [51]: lista = [1,2,3,4,5] # Una lista estandar

In [52]: otro_ceros = numpy.zeros_like(lista) # un array de ceros basado en lista

In [53]: otro_ceros
Out[53]: array([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.])
```

Arrays de números aleatorios

```
In [27]: from numpy import random as numpy_random

In [28]: lista_azar = numpy_random.random(10)

In [29]: len(lista_azar)
Out[29]: 10

In [30]: lista_azar
Out[30]:
array([ 0.48643022,  0.67351304,  0.31726036,  0.64209621,  0.82233782,
        0.20268152,  0.03937668,  0.23962648,  0.474803  ,  0.38983222])

In [31]: numpy_random.random(10)
Out[31]:
array([ 0.93184109,  0.3779432 ,  0.53984084,  0.29510925,  0.75508913,
        0.59194663,  0.95704321,  0.54417432,  0.51413258,  0.60133594])

In [32]: numpy_random.random(10)
Out[32]:
array([ 0.48318427,  0.19214245,  0.43004535,  0.85029769,  0.03929986,
        0.77117749,  0.07195387,  0.30978705,  0.6257316 ,  0.81006165])
```

Arrays de números aleatorios

```
In [49]: enteros_random = numpy_random.random_integers(1,100,100)

In [50]: len(enteros_random)
Out[50]: 100

In [51]: enteros_random
Out[51]:
array([18, 68, 66, 25, 73, 17, 72, 82, 75, 27, 15, 56, 47, 81, 96, 59, 94,
       93, 20, 99, 16, 88, 99,  1, 17,  6, 19, 17, 13, 19, 26, 73,  7, 62,
       56, 59, 21, 55,  4,  6, 83, 73, 87, 48, 32, 78, 87,  7, 36, 78, 11,
       31, 99, 43, 57, 42, 92, 73, 88, 59, 43, 13, 73, 29, 66, 28, 73, 19,
        8,  7, 18, 46, 43, 35, 62, 84, 27, 59, 34, 84, 12, 80, 11, 41, 40,
       10,  5, 97, 63, 66, 74, 73, 24, 50, 90, 50, 95, 41, 45, 54])

In [52]:
```

Espacio lineal

- Un arreglo de n números distribuidos uniformemente en un intervalo.

```
In [2]: import numpy

In [3]: espacio = numpy.linspace(0, 1000, 20) # 20 valores en intervalo 0-1000

In [4]: espacio
Out[4]:
array([    0.        ,    52.63157895,   105.26315789,   157.89473684,
        210.52631579,   263.15789474,   315.78947368,   368.42105263,
        421.05263158,   473.68421053,   526.31578947,   578.94736842,
        631.57894737,   684.21052632,   736.84210526,   789.47368421,
        842.10526316,   894.73684211,   947.36842105,  1000.        ])
```

Espacio lineal

- Puede aplicar funciones directamente sobre un espacio lineal → vectorizacion (*)
- :-O

```
In [6]: def f(x):
   ...:     return 2 * x
   ...:

In [7]: y = f(espacio)

In [8]: y
Out[8]:
array([    0.        ,   105.26315789,   210.52631579,   315.78947368,
         421.05263158,   526.31578947,   631.57894737,   736.84210526,
         842.10526316,   947.36842105,  1052.63157895,  1157.89473684,
        1263.15789474,  1368.42105263,  1473.68421053,  1578.94736842,
        1684.21052632,  1789.47368421,  1894.73684211,  2000.        ])

In [9]:
```

Operaciones aritméticas básicas

```
In [55]: A = numpy.arange(5)

In [56]: B = numpy.arange(5, 10)

In [57]: A
Out[57]: array([0, 1, 2, 3, 4])

In [58]: B
Out[58]: array([5, 6, 7, 8, 9])

In [59]: A + B # Suma elemento por elemento
Out[59]: array([ 5,  7,  9, 11, 13])

In [60]: B - A
Out[60]: array([5, 5, 5, 5, 5])

In [61]: A * B
Out[61]: array([ 0,  6, 14, 24, 36])
```

Operaciones aritméticas básicas

```
In [58]: import math

In [59]: lista1 = numpy.arange(50)

In [60]: lista1
Out[60]:
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49])

In [61]: lista1 * math.pi
Out[61]:
array([   0.        ,    3.14159265,    6.28318531,    9.42477796,
        12.56637061,   15.70796327,   18.84955592,   21.99114858,
        25.13274123,   28.27433388,   31.41592654,   34.55751919,
        37.69911184,   40.8407045 ,   43.98229715,   47.1238898 ,
        50.26548246,   53.40707511,   56.54866776,   59.69026042,
        62.83185307,   65.97344573,   69.11503838,   72.25663103,
        75.39822369,   78.53981634,   81.68140899,   84.82300165,
        87.9645943 ,   91.10618695,   94.24777961,   97.38937226,
       100.53096491,  103.67255757,  106.81415022,  109.95574288,
       113.09733553,  116.23892818,  119.38052084,  122.52211349,
       125.66370614,  128.8052988 ,  131.94689145,  135.0884841 ,
       138.23007676,  141.37166941,  144.51326207,  147.65485472,
       150.79644737,  153.93804003])
```

Comparación de Arreglos

```
In [68]: lista1 = numpy_random.random_integers(1,10,20)

In [69]: lista2 = numpy_random.random_integers(1,10,20)

In [70]: lista1
Out[70]:
array([ 8,  9,  3,  6,  5,  8,  5,  8,  4,  9,  1,  1,  3,  9, 10,  9,  2,
        1,  1,  5])

In [71]: lista2
Out[71]:
array([10, 10,  4,  6,  9,  9,  1,  7,  6,  3,  4,  1,  9,  2,  9,  7,  7,
        8,  7,  6])

In [72]: lista1 == lista2
Out[72]:
array([False, False, False,  True, False, False, False, False, False,
       False, False,  True, False, False, False, False, False, False,
       False, False], dtype=bool)
```

Comparación de Arreglos

- La comparación devuelve otro numpy array.
- Tiene operaciones como .any() y .all()

```
In [75]: comparacion = lista1 == lista2

In [76]: comparacion
Out[76]:
array([False, False, False,  True, False, False, False, False, False,
       False, False,  True, False, False, False, False, False, False,
       False, False], dtype=bool)

In [77]: comparacion.any()
Out[77]: True

In [78]: comparacion.all()
Out[78]: False
```

Comparación de desempeño

```
In [20]: from operator import add

In [21]: lista1 = range(100000)

In [22]: lista2 = range(100000)

In [23]: %timeit lista3 = map(add, lista1, lista2)
100 loops, best of 3: 10 ms per loop
```



```
In [30]: import numpy

In [31]: nlista1 = numpy.arange(100000)

In [32]: nlista2 = numpy.arange(100000)

In [33]: %timeit nlista3 = nlista1 + nlista2
10000 loops, best of 3: 184 us per loop
```

Arreglos multidimensionales

```
In [88]: A = numpy.arange(16)

In [89]: A
Out[89]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])

In [90]: matrix_A = A.reshape(4,4)

In [91]: matrix_A
Out[91]:
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

```
In [103]: matrix_A[:, 3] # Extraer la columna 3
Out[103]: array([ 3,  7, 11, 15])

In [104]: matrix_A[2, :] # Extraer la fila 2
Out[104]: array([ 8,  9, 10, 11])
```

Indexación avanzada

```
In [124]: lista = numpy.arange(5, 10)

In [125]: lista
Out[125]: array([5, 6, 7, 8, 9])

In [126]: lista[[0,2]]
Out[126]: array([5, 7])

In [127]: lista[[0,2]] = 0

In [128]: lista
Out[128]: array([0, 6, 0, 8, 9])
```

```
In [135]: otro_array = numpy.random.random_integers(0,100,100)

In [136]: otro_array
Out[136]:
array([ 53,  52,  57,  63,  86,  44,  19,  45,  50,   6,  40,  58,  37,
         0,  38,  76,  36,  35,  46,  88,  83,  22,   7,   7,  12,  95,
        11,  53, 100,   4,  21,  18,  88,  25,  98,  54,  53,  61,  53,
        60,  17,  11,  53,  58,  36,  92,  73,  71,  37,  28,  95,  53,
        38,  97,  86,  10,  81,  94,  43,  85,  18,  24,  90,  64,  88,
        68,  76,  81,  42,  29,  90,  79,   9,  40,   0,  44,  48,  54,
        50,  81,  44,  86,  69,  29,   7,  98,  81,   4,  70,   6,  76,
        16,  21,  57,  13,  14,  95,   7,  43,  99])

In [137]: otro_array[otro_array>50] # Un array con elementos > 50
Out[137]:
array([ 53,  52,  57,  63,  86,  58,  76,  88,  83,  95,  53, 100,  88,
        98,  54,  53,  61,  53,  60,  53,  58,  92,  73,  71,  95,  53,
        97,  86,  81,  94,  85,  90,  64,  88,  68,  76,  81,  90,  79,
        54,  81,  86,  69,  98,  81,  70,  76,  57,  95,  99])
```

Diferencias con listas y tupas estándar

- Suma y multiplicación con escalares devuelven resultados diferentes.

```
>>> numpy.arange(5)*2
array([0, 2, 4, 6, 8])
>>> range(5)*2
[0, 1, 2, 3, 4, 0, 1, 2, 3, 4]
```

```
>>> numpy.arange(5) + numpy.arange(5)
array([0, 2, 4, 6, 8])
>>> range(5) + range(5)
[0, 1, 2, 3, 4, 0, 1, 2, 3, 4]
```

Diferencias con listas y tupas estándar

- En numpy, los “slices” son referencias, en listas y tuplas normales, son copias.

```
>>> A = numpy.arange(5)
>>> B = A[0:1]
>>> B[0] = 42
>>> A
array([42,  1,  2,  3,  4])
>>> >>> A = range(5)
>>> B = A[0:1]
>>> B[0] = 42
>>> A
[0, 1, 2, 3, 4]
```

Operaciones sobre numpy arrays

- Mínimo, máximo, promedio, desviación estándar, sumatoria.

```
In [10]: import numpy

In [11]: from numpy.random import random_integers

In [12]: lista = random_integers(0,100,100)

In [13]: lista
Out[13]:
array([33, 34, 16, 78, 40, 71, 97, 95, 62, 96, 77, 44, 18, 58, 76, 17, 20,
        5, 97, 50, 34, 83, 59,  4, 95, 42, 52, 37, 82, 73, 18, 11, 47, 19,
       38, 70,  0, 95, 27, 47, 58, 77, 47, 62, 14, 43, 15, 59, 11, 75, 30,
       99, 61, 91, 43, 37, 12, 14, 33,  6, 13, 18, 36, 72, 94, 75, 71, 49,
       23,  4, 24, 57, 21, 16, 44, 57, 77, 83, 17, 87, 24, 40, 26, 66, 46,
       96, 32, 28, 30, 77, 57, 52, 34,  8, 72, 74, 48, 52, 48, 18])

In [14]: lista.min()
Out[14]: 0

In [15]: lista.max()
Out[15]: 99

In [16]: lista.mean()
Out[16]: 47.700000000000003

In [17]: lista.sum()
Out[17]: 4770
```

Escribir y leer archivos

```
In [66]: lista = random_integers(0,10,1000)

In [67]: lista.tofile('./numpy_numeros.dat', ',')
```

numpy_numeros.dat

```
1 5,6,3,2,5,5,2,4,0,8,4,2,7,0,10,0,7,2,1,6,0,1,4,1,7,10,2,
```

```
In [74]: lista_leida = numpy.fromfile('./numpy_numeros.dat', sep=',')

In [75]: len(lista_leida)
Out[75]: 1000

In [76]: type(lista_leida)
Out[76]: numpy.ndarray

In [77]: lista_leida[0:100]
Out[77]:
array([  5.,   6.,   3.,   2.,   5.,   5.,   2.,   4.,   0.,   8.,   4.,
         2.,   7.,   0.,  10.,   0.,   7.,   2.,   1.,   6.,   0.,   1.,
         4.,   1.,   7.,  10.,   2.,   0.,   7.,   2.,   8.,   9.,   8.,
         4.,   4.,   1.,   9.,   3.,  10.,   4.,   8.,   2.,   6.,   3.,
         7.,   9.,   0.,   5.,   7.,   1.,   1.,   5.,   9.,   6.,   7.,
         8.,   4.,   2.,   0.,   1.,   0.,   3.,   2.,   6.,   6.,   2.,
         2.,   6.,   7.,   0.,   1.,   1.,   7.,  10.,   1.,   1.,  10.,
         9.,   3.,   0.,   1.,   4.,   4.,   6.,   9.,   8.,   8.,   9.,
         8.,   9.,  10.,   3.,   8.,   7.,  10.,   0.,   3.,   6.,   4.,
         1.])
```

Matrices en numpy

```
In [121]: A = numpy.matrix([[0, 1],
                            [2, 3]])

In [122]: B = numpy.matrix([[4, 5],
                            [6, 7]])

In [123]: A.I # Inversa de A
Out[123]:
matrix([[-1.5,  0.5],
        [ 1. ,  0. ]])

In [124]: A.T # Transpuesta de A
Out[124]:
matrix([[0, 2],
        [1, 3]])

In [125]: A.diagonal()
Out[125]: matrix([[0, 3]])

In [126]: A+B
Out[126]:
matrix([[ 4,  6],
        [ 8, 10]])

In [127]: A*B
Out[127]:
matrix([[ 6,  7],
        [26, 31]])
```

Polinomios con Numpy

```
In [14]: import numpy as np

In [15]: p = np.poly1d([3, 4, 5])

In [16]: print p
   2
3 x + 4 x + 5

In [17]: print p.deriv() # Derivdad del polinomio

6 x + 4

In [18]: print p.integ(k=6)
   3     2
1 x + 2 x + 5 x + 6
```

```
In [34]: p2 = np.poly1d([6, 7, 8]) # Definimos otro polinomio

In [35]: print p + p2 # Suma de polinomios
   2
9 x + 11 x + 13

In [36]: p.order # Orden del polinomio
Out[36]: 2

In [37]: print p(10) # Evaluacion del polimonio con la variable en 10
345
```

… y mucho, mucho más!

http://docs.scipy.org/doc/numpy/genindex.html

Para usuarios de Matlab:

http://www.scipy.org/NumPy_for_Matlab_Users