Taller de Python Científico

python™

Introducción a NumPy

Departamento de
Ciencias de la Atmósfera y los Océanos
Facultad de Ciencias Exactas y Naturales - Universidad de Buenos Aires

Franco N. Bellomo - fnbellomo@gmail.com

¿Python para programación científica?

* Pro:
  * Programar Python es conveniente.
  * Desarrollo rápido (no se compila, no se linkea).
* Contras:
  * Lenguajes interpretados son muy lentos comparados con los lenguajes compilados.
  * Las listas son costosas e ineficientes para operar.

¡NumPy es la solución!

NumPy

* Un poderoso array N-dimensional.
* Sofisticadas funciones (broadcasting).
* Herramientas para la integración de C / C ++ y Fortran
* Álgebra lineal, transformada de Fourier y generadores de números aleatorio.

Core de Numpy

* Objeto ndarray: encapsula arrays homogeneos multidimensional.
* Todos los elementos tienen que ser del mismo tipo.
* Soporta distintos tipos de datos.
* El tamaño de los arrays es fijo (se realia una allocated).

Creando Arrays

```
>>> import numpy as np

>>> np.array([[1,2,3], [4,5,6]])
array([[1,2,3], [4,5,6]])

>>> # sequences
>>> np.arange(0,10,0.1)
array([0.,  0.1,  0.2, ..., 9.7,  9.8, 9.9])

>>> np.linspace(0,2*np.pi,100)
array([0.,  0.06896552,  0.13793103,
       1.86206897,  1.93103448,  2.])

>>> np.ones((5,5))
array([ [ 1.,  1.,  1.,  1.,  1.],
        ...
        [ 1.,  1.,  1.,  1.,  1.] ])
```

Atributos de ndarray

```
>>> arr = np.arange(10).reshape((2, 5))
>>> arr.ndim
2

>>> arr.shape
(2, 5)

>>> arr.size
10

>>> arr.dtype
int64

>>> arr.T          # transpose
```

Tipos de datos de NumPy

NumPy soporta una gran variedad de tipos de datos, podiendo especificar cuanto bits usa. Por ej:

* bool * int * int8
* int16 * uint32 * uint64
* float32 * float64 * complex64 * complex128

```
>>> a = np.array([0,2,3,4],np.complex128)
>>> a
array([ 0.+0.j, 2.+0.j, 3.+0.j, 4.+0.j])

>>> a = np.array([0,2,3,4], dtype=np.int8)
>>> a[1] += 128
>>> print(a[1])
-126
```

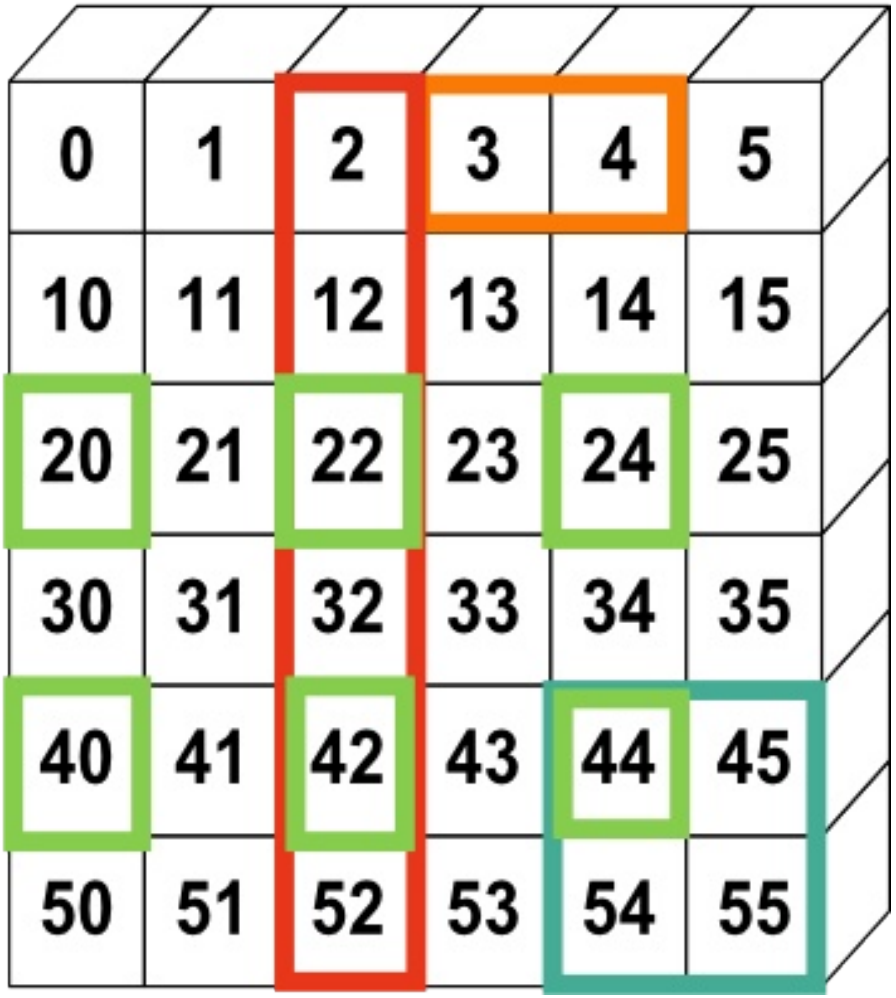
Indexing and slicing

```
>>> a[0,3:5]
array([3,4])


>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,12,22,32,42,52])

>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```

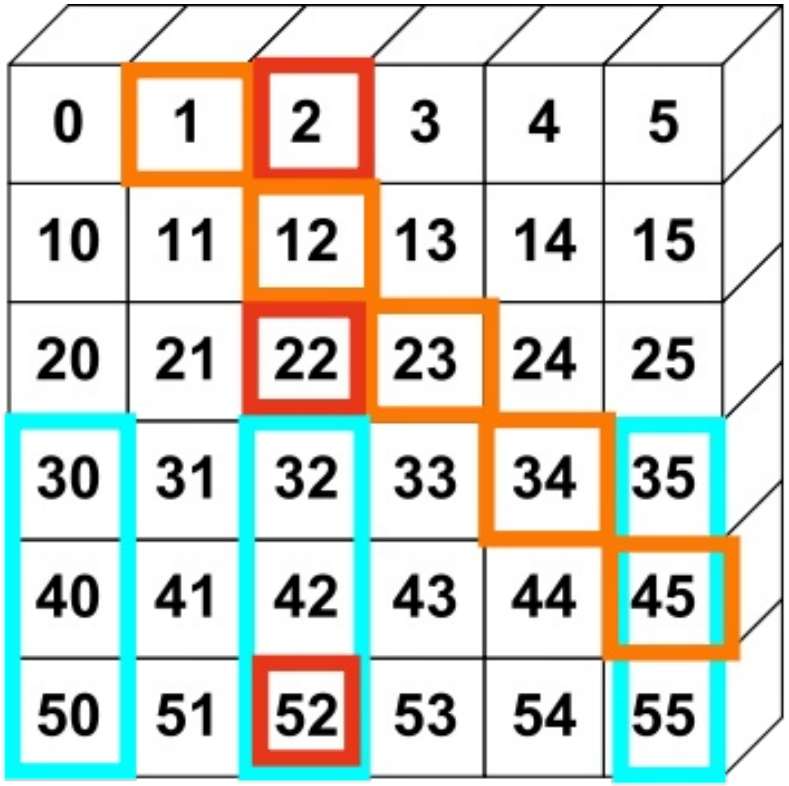


Indexing and slicing

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



Assignment, shallow and deep copy

```
>>> a = np.array([1,2,3,4,5], dtype='uint8')
>>> b = a
>>> b[0] = 90
>>> a
```

Assignment, shallow and deep copy

```
>>> a = np.array([1,2,3,4,5], dtype='uint8')
>>> b = a
>>> b[0] = 90
>>> a
[90, 2, 3, 4]
>>> id(a), id(b)
140181694751648 140181694751648

>>> b = a[2:]
>>> b[0] = 44
>>> print(b)
[44, 4, 5]
>>> print(b)
```

Cuidado!!

Assignment, shallow and deep copy

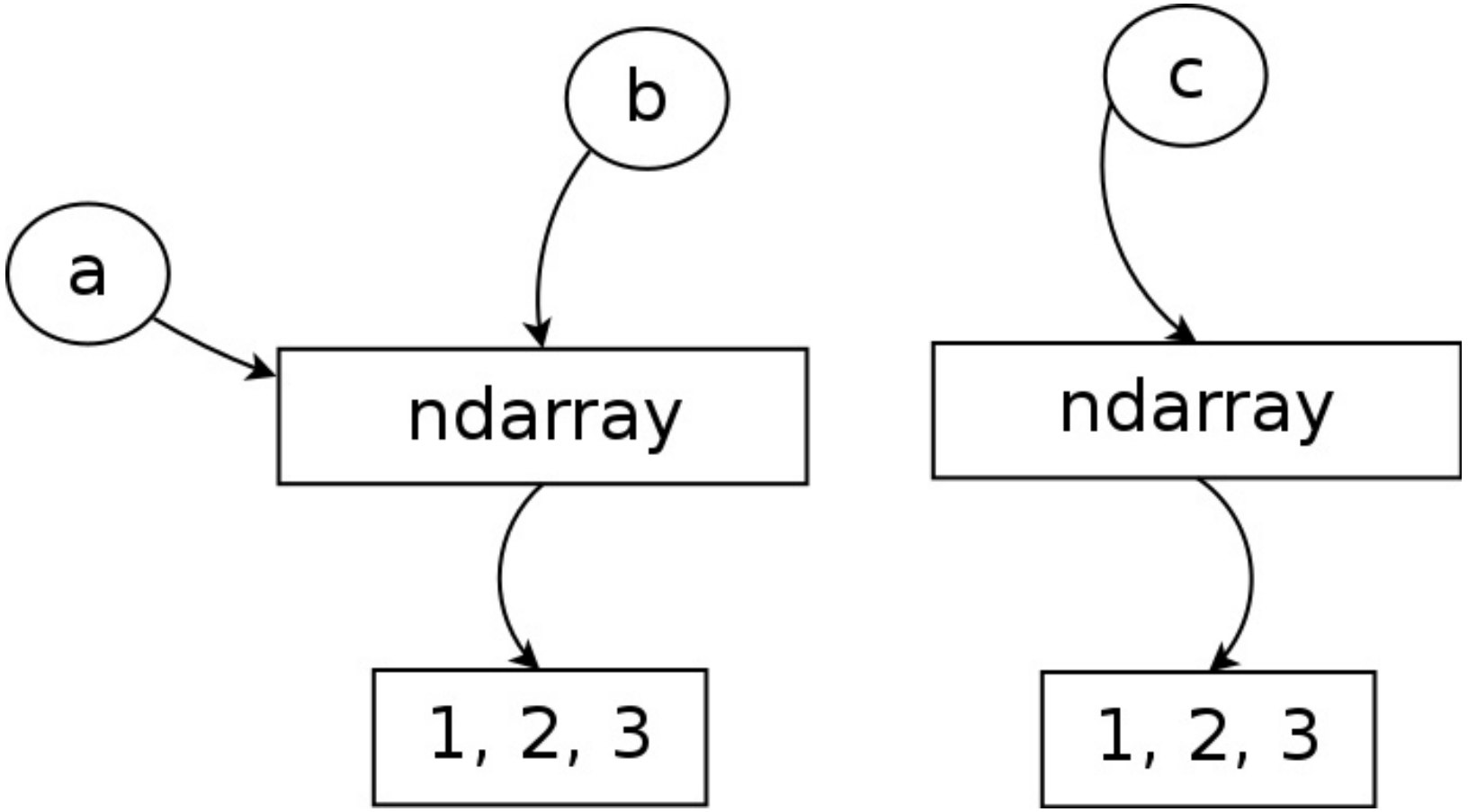
```
>>> a = np.array([1,2,3,4,5], dtype='uint8')
>>> b = a
>>> b[0] = 90
>>> a
[90, 2, 3, 4]
>>> id(a), id(b)
140181694751648 140181694751648
```

Cuidado!!

```
>>> b = a[2:]
>>> b[0] = 44
>>> print(b)
[44, 4, 5]
>>> print(b)
[1, 2, 44, 4, 5]
```

Cuidado!!

Assignment, shallow and deep copy



Assignment, shallow and deep copy

```
>>> a = np.array([1,2,3,4,5])
>>> c = a.copy()
>>> c[0] = 88
>>> print(c)
[88  2  3  4  5]
>>> print(a)
[1 2 3 4 5]
>>> id(a), id(c)
140181694752608
140181694751168
```

Eliminando fors

```
%% timeit
>>> for i in range(1000):
>>>     for i in range (len(a)) :
>>>         a [ i ] += 1
# 1 loops, best of 3: 8.68s perloop

%% timeit
>>> for i in range(1000):
>>>     b[:] += 1
# 100 loops, best of 3: 18.1 ms per loop
```

Operaciones elemento a elemento

```
>>> a = np.array([1,2,3],float)
>>> b = np.array([5,2,6],float)
>>> a + b
array([ 6., 4., 9.])
>>> a - b
array([-4., 0., -3.])
>>> a * b
array([ 5., 4., 18.])
>>> b / a
array([ 5., 1., 2.])
>>> a % b
array([ 1., 0., 3.])
>>> b ** a
array([5., 4., 216.])
```

Operaciones matemáticas

NumPy posee un gran cantidad de operaciones matemáticas optimizadas para ser aplicadasen los array.
Ej: abs, sign, sqrt, log, log10, exp, sin, cos, tan, ...

```
>>> a = np.linspace(0.3,0.6,4)
>>> print(a)
[ 0.3 0.4 0.5 0.6]
>>> np.sin(a)
array([ 0.29552021, 0.38941834, 0.56464247])
>>> np.exp(a)
array([ 1.34985881, 1.4918247 , 1.8221188 ])
```

Operaciones de Reducción

```
>>> a = np.array([2,4,3],dtype=np.float64)
>>> a.sum()
9.0
>>> a.prod()
24.0
>>> np.sum(a)
9.0
>>> a.mean()
3.0
>>> a.var()
0.6666666666666663
>>> a.std()
0.8164965809277260
```

Operaciones booleanas

* Operadores booleanos pueden ser utilizados en toda arrays y después produce una matriz de booleanos.
* Las comparaciones se pueden utilizar como "filtros"

```
>>> a = np.array([[6,4],[5,9]])
>>> print (a >= 6)
[[ True False]
[False True]]
>>> print (a[ a >= 6])
[6 9]
>>> b = a < 6
>>> print (a[b])
[4 5]
```

Operaciones de Algebra Lineal

Las operaciones sobre matrices y vectores in NumPy son muy eficientes por que estan linqueadas BLAS/LAPACK code (can use MKL, OpenBLAS, ACML, ATLAS, etc.)

Operaciones de Algebra Lineal

```
>>> a = np.array([[0,1],[2,3]],float)
>>> b = np.array([2,3],float)
>>> c = np.array([[1,1],[4,0]],float)
>>> np.dot(b,a)
array([ 6., 11.])
>>> np.dot(a,b)
array([ 3., 13.])
>>> np.dot(a,c)
array([[ 4., 0.],
[ 14., 2.]])
>>> np.outer(b,b)
array([[ 4., 6.],
       [ 6., 9.]])
```

Operaciones de Algebra Lineal

La mayoria de las operaciones de algebra lineal estan ubicadas en el submodulo linalg

```
>>> a = np.array([[4,2,0],[9,3,7],[1,2,1]],float)
>>> np.linalg.det(a)
-48.000000000000028
>>> vals,vecs = np.linalg.eig(a)
>>> vals
array([ 8.85591316, 1.9391628 , -2.79507597])
>>> vecs
array([[-0.3663565 , -0.54736745, 0.25928158],
       [-0.88949768, 0.5640176 , -0.88091903],
       [-0.27308752, 0.61828231, 0.39592263]])>
```

Operaciones de Algebra Lineal

```
>>> a = np.array([[0,1],[2,3]],float)
>>> b = np.array([2,3],float)
>>> c = np.array([[1,1],[4,0]],float)
>>> np.dot(b,a)
array([ 6., 11.])
>>> np.dot(a,b)
array([ 3., 13.])
>>> np.dot(a,c)
array([[ 4., 0.],
[ 14., 2.]])
>>> np.outer(b,b)
array([[ 4., 6.],
       [ 6., 9.]])
```

NumPy Submodulos

* Polynomial mathematics
* Statistical computations
* Pseudo random number generators
* Discrete Fourier transforms

SciPy

* Clustering
* Transformada de Fourier
* Integración numérica
* Interpolación
* Datos I/O
* Matrices Sparsas
* Resolvedores lineales
* Optimización
* Procesamiento de Señales
* Funciones de estadistica
* Eigenvalue