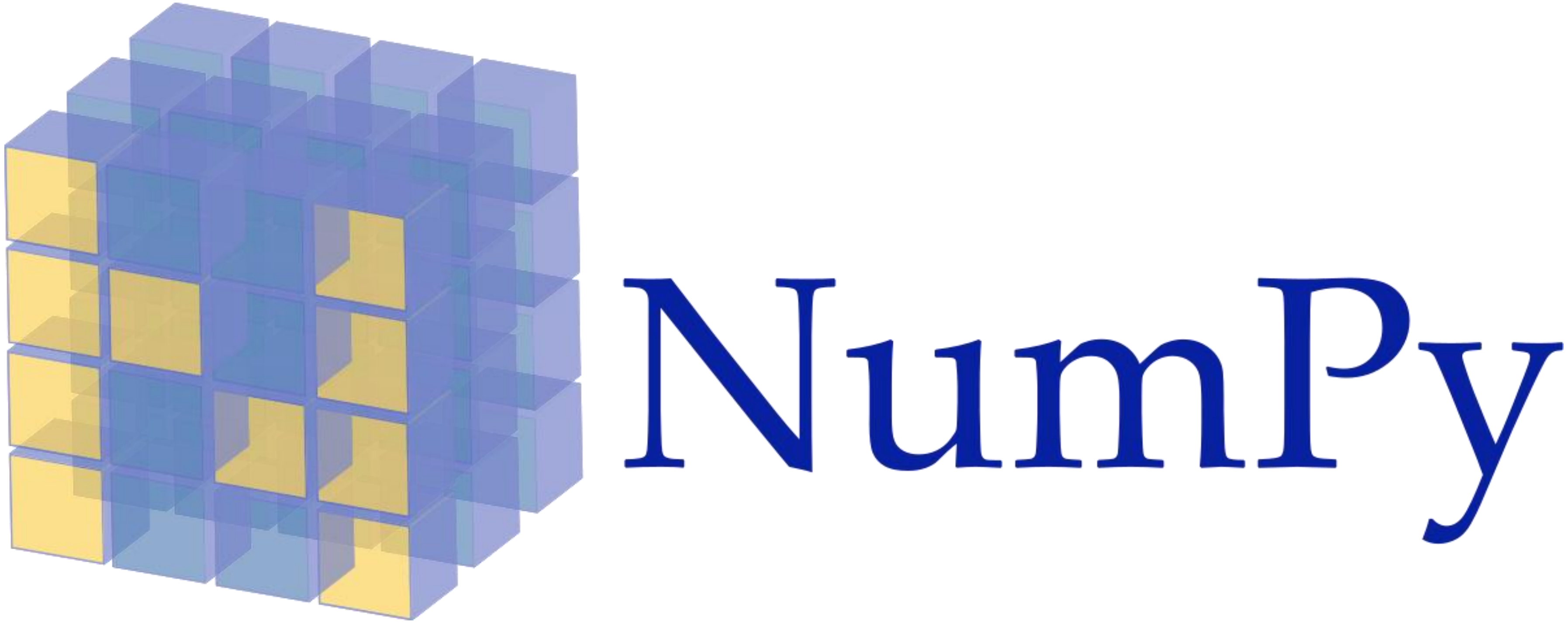
Numpy

Programación II

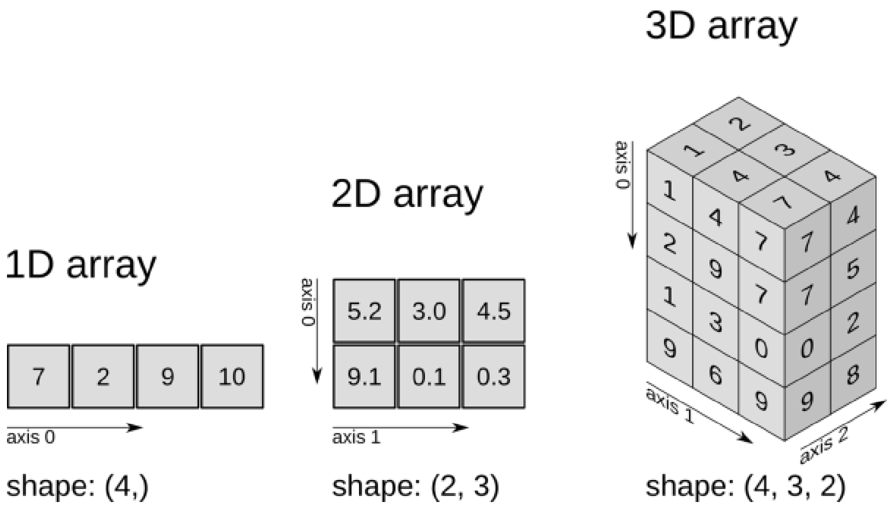


Numpy

Programación II

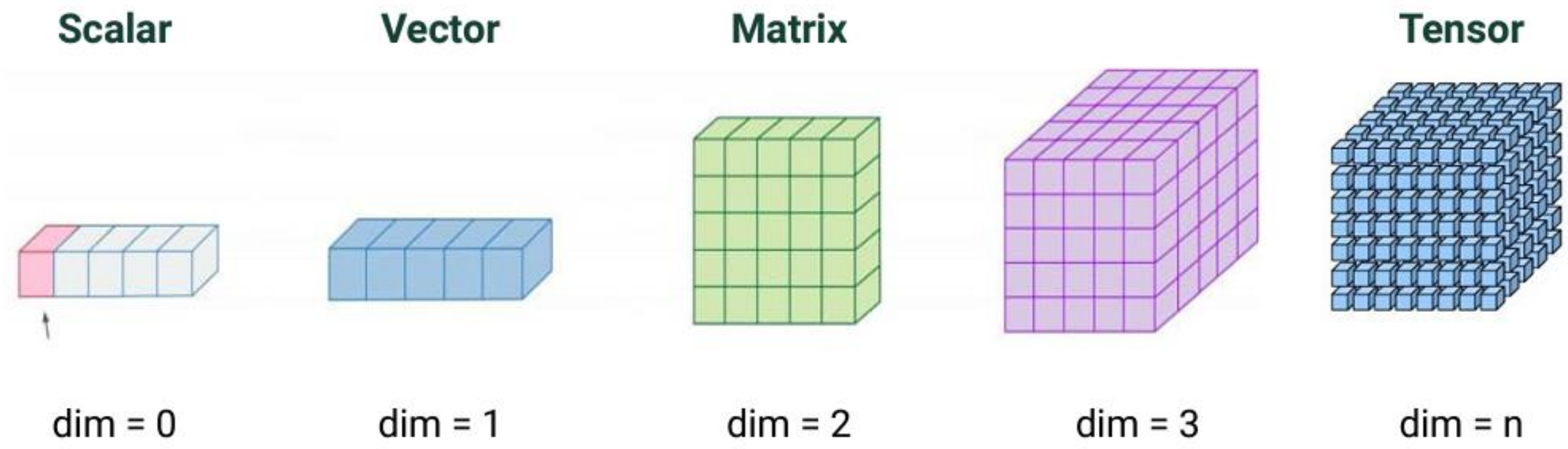
NumPy es una librería de Python especializada en el cálculo numérico y el análisis de datos, especialmente para un gran volumen de datos.

*Incorpora una nueva clase de objetos llamados **arrays** que permite representar colecciones de datos de un mismo tipo en varias dimensiones, y funciones muy eficientes para su manipulación.*



Numpy (Dimensión de los arreglos)

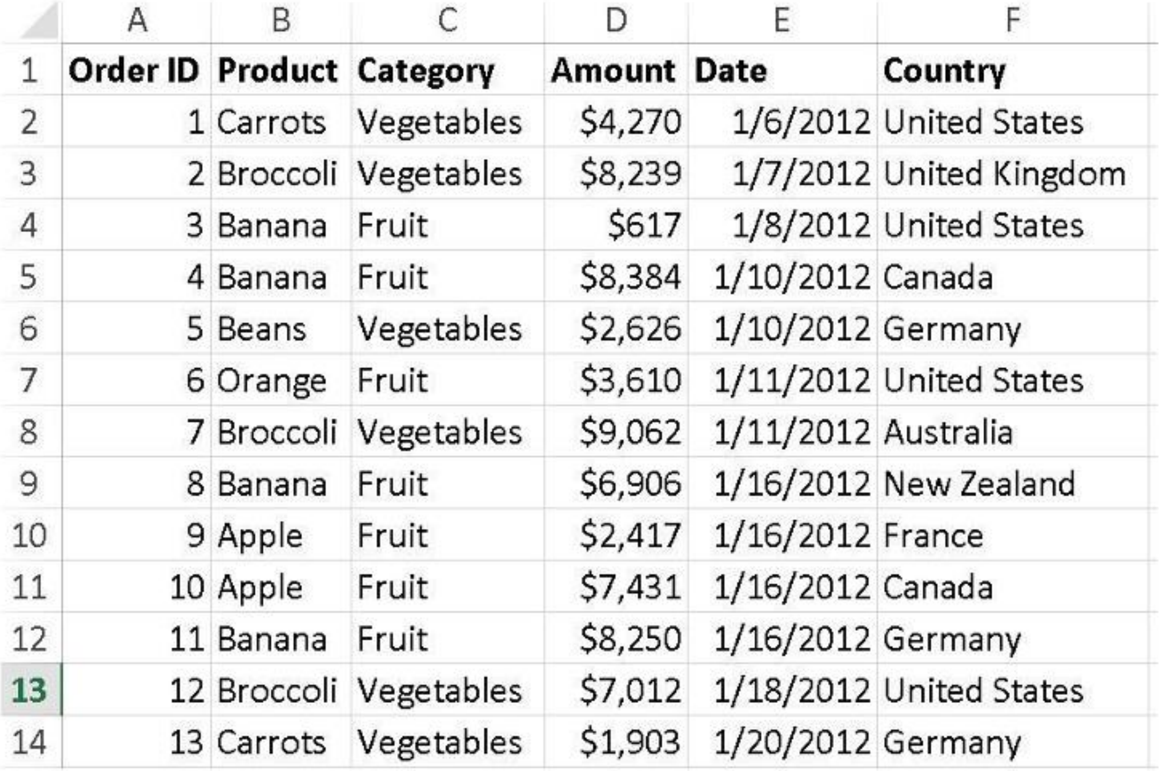
Programación II



Numpy (2D)

Programación II

(Ejemplos, características)

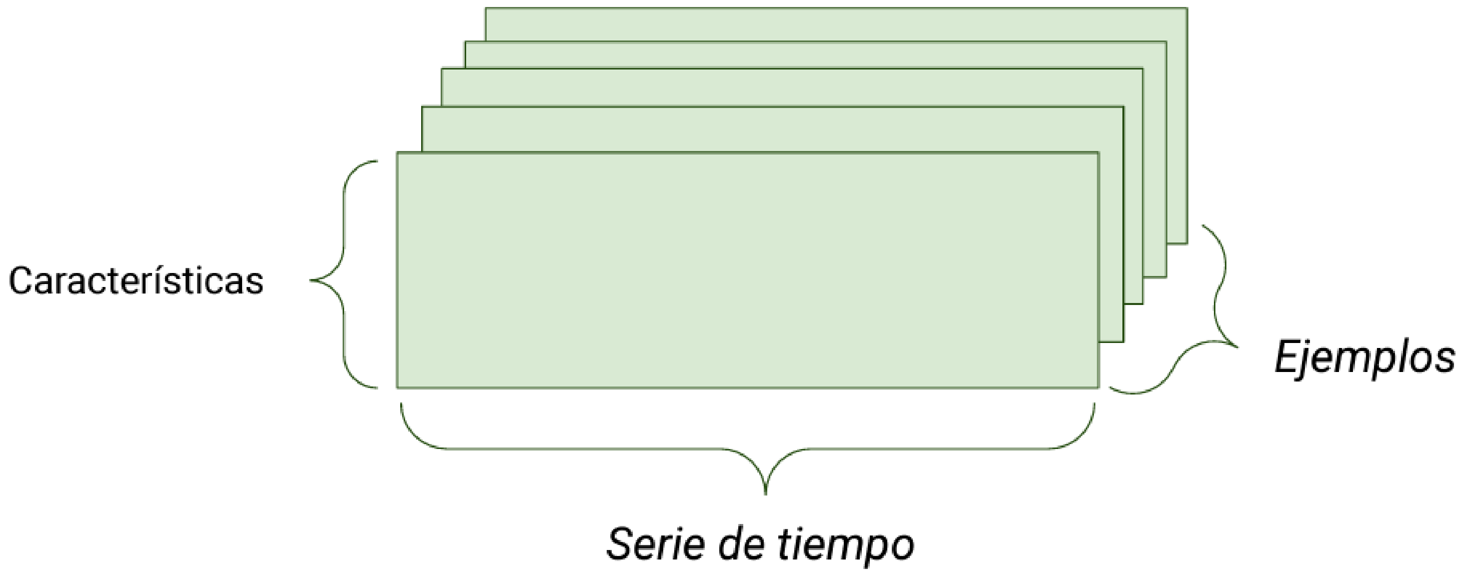


	A	B	C	D	E	F
1	**Order ID**	**Product**	**Category**	**Amount**	**Date**	**Country**
2	1	Carrots	Vegetables	$4,270	1/6/2012	United States
3	2	Broccoli	Vegetables	$8,239	1/7/2012	United Kingdom
4	3	Banana	Fruit	$617	1/8/2012	United States
5	4	Banana	Fruit	$8,384	1/10/2012	Canada
6	5	Beans	Vegetables	$2,626	1/10/2012	Germany
7	6	Orange	Fruit	$3,610	1/11/2012	United States
8	7	Broccoli	Vegetables	$9,062	1/11/2012	Australia
9	8	Banana	Fruit	$6,906	1/16/2012	New Zealand
10	9	Apple	Fruit	$2,417	1/16/2012	France
11	10	Apple	Fruit	$7,431	1/16/2012	Canada
12	11	Banana	Fruit	$8,250	1/16/2012	Germany
13	12	Broccoli	Vegetables	$7,012	1/18/2012	United States
14	13	Carrots	Vegetables	$1,903	1/20/2012	Germany

Numpy (3D)

Programación II

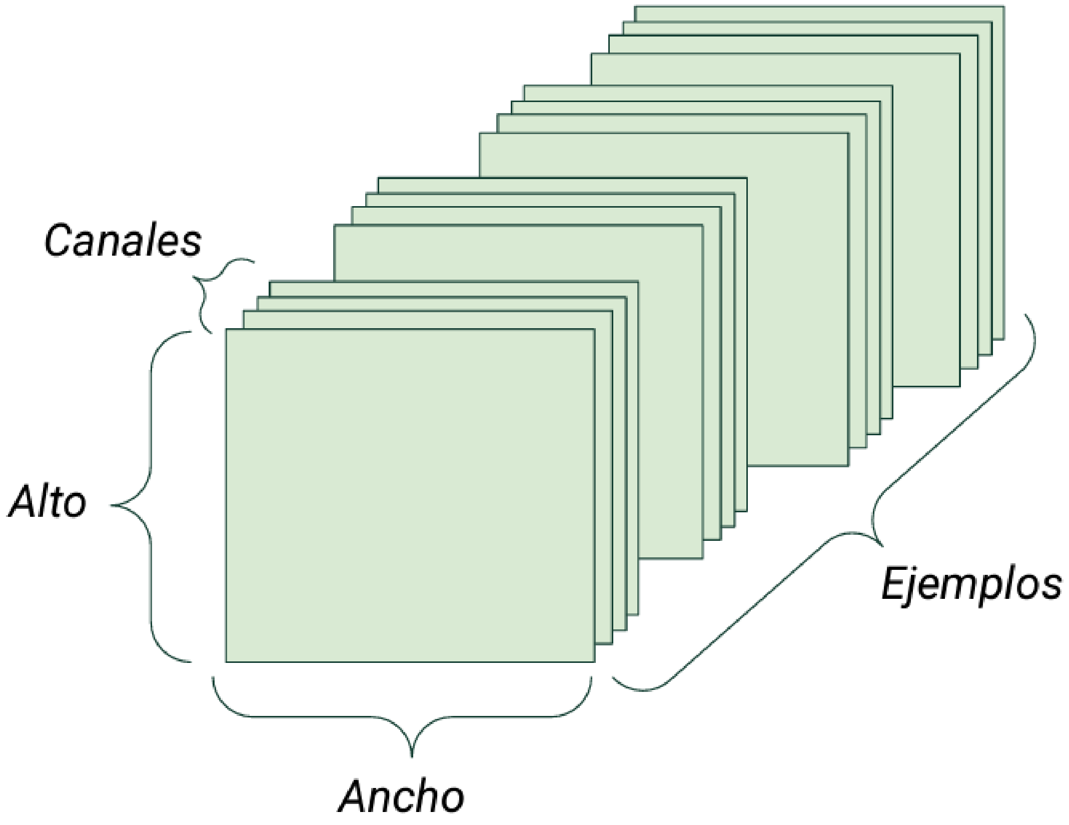
(Ejemplos, Serie de tiempo, Características)



Numpy (4D)

Programación II

(Ejemplos, Ancho, Alto, Canales de color)



Numpy

Programación II

Características adicionales:

- *Matrices multidimensionales rápidas*
- *Bibliotecas de funciones científicas confiables y probadas*
- *Plotting tolos*

NumPy: *es el núcleo de casi todas las aplicaciones o módulos científicos de Python, ya que proporciona un tipo de datos de matriz N-d rápido que se puede manipular en forma vectorizada.*

Numpy

Programación II

NumPy

Es el paquete fundamental necesario para la computación científica con Python.

- *Un poderoso objeto de matriz N-dimensional*
- *Funciones básicas de álgebra lineal*
- *Transformadas básicas de Fourier*
- *Capacidades sofisticadas de números aleatorios*
- *Herramientas para integrar código Fortran*
- *Herramientas para integrar código C/C++*

Numpy

Programación II

- *Documentación Oficial*
  - *http://docs.scipy.org/doc/*
- *Libro de numpy*
  - *http://web.mit.edu/dvp/Public/numpybook.pdf*
- *Lista de ejemplos*
  - *https://docs.scipy.org/doc/numpy/reference/routines.html*

Matrices - Python numérico (Numpy)

Listas para almacenar pequeñas cantidades de datos unidimensionales

```
>>> a = [1,3,5,7,9]
>>> print(a[2:4])
[5, 7]
>>> b = [[1, 3, 5, 7, 9], [2, 4, 6, 8, 10]]
>>> print(b[0])
[1, 3, 5, 7, 9]
>>> print(b[1][2:4])
[6, 8]
```

```
>>> a = [1,3,5,7,9]
>>> b = [3,5,6,7,9]
>>> c = a + b
>>> print c
[1, 3, 5, 7, 9, 3, 5, 6, 7, 9]
```

*Puede usar directamente con operadores aritméticos (+, -, *, /, …)*

Arreglos eficientes con aritmética y mejores herramientas multidimensionales

Numpy

```
>>> import numpy
```

Similar a las listas, pero mucho más capaz

Numpy (Algunos metodos de narray

Programación II

- ***ndarray. tolist ()***

El contenido de self como una lista anidada

- ***ndarray. copy ()***

Devuelve una copia de la matriz.

- ***ndarray. fill (scalar)***

Rellenar una matriz con el valor escalar

Numpy (Funciones de numpy)

Programación II

- *abs()*
- *add()*
- *binomial()*
- *cumprod()*
- *cumsum()*
- *floor()*
- *histogram()*

- *min()*
- *max()*
- *multipy()*
- *polyfit()*
- *randint()*
- *shuffle()*
- *transpose()*

Numpy (Vectores)

Programación II

- Para listas
  - np.array

```
# as vectors from lists
>>> a = np.array([1,3,5,7,9])
>>> b = np.array([3,5,6,7,9])
>>> c = a + b
>>> print(c)
[4, 8, 11, 14, 18]

>>> type(c)
(<type ‘np.ndarray'>)

>>> c.shape
(5,)
```

Numpy (Matrices)

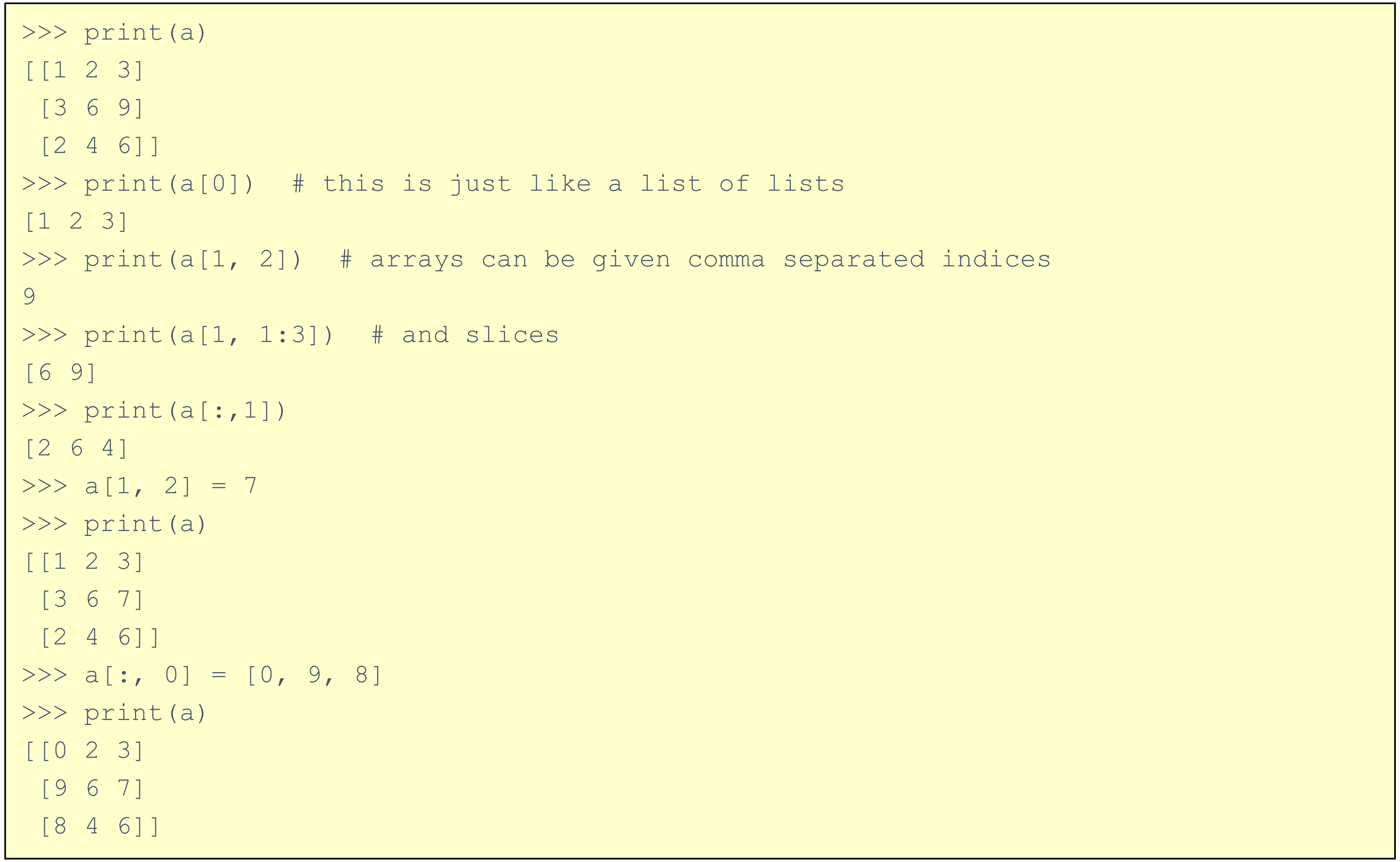
Programación II

```
>>> l = [[1, 2, 3], [3, 6, 9], [2, 4, 6]]  # create a list
>>> a = np.array(l)  # convert a list to an array
>>>print(a)
[[1 2 3]
 [3 6 9]
 [2 4 6]]
>>> a.shape
(3, 3)
>>> print(a.dtype)  # get type of an array
int64

# or directly as matrix
>>> M = array([[1, 2], [3, 4]])
>>> M.shape
(2,2)
>>> M.dtype
dtype('int64')
```

```
#only one type
>>> M[0,0] = "hello"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for long() with base 10: 'hello`

>>> M = np.array([[1, 2], [3, 4]], dtype=complex)
>>> M
array([[ 1.+0.j,  2.+0.j],
       [ 3.+0.j,  4.+0.j]])
```

Numpy (Uso de Matrices)

Programación II



```
>>> print(a)
[[1 2 3]
 [3 6 9]
 [2 4 6]]
>>> print(a[0])  # this is just like a list of lists
[1 2 3]
>>> print(a[1, 2])  # arrays can be given comma separated indices
9
>>> print(a[1, 1:3])  # and slices
[6 9]
>>> print(a[:,1])
[2 6 4]
>>> a[1, 2] = 7
>>> print(a)
[[1 2 3]
 [3 6 7]
 [2 4 6]]
>>> a[:, 0] = [0, 9, 8]
>>> print(a)
[[0 2 3]
 [9 6 7]
 [8 4 6]]
```

Numpy (Arreglos)

Programación II

- Generation functions

```
>>> x = arange(0, 10, 1) # arguments: start, stop, step
>>> x
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

>>> np.linspace(0, 10, 25)
array([  0.        ,   0.41666667,   0.83333333,   1.25      ,
         1.66666667,   2.08333333,   2.5       ,   2.91666667,
         3.33333333,   3.75      ,   4.16666667,   4.58333333,
         5.        ,   5.41666667,   5.83333333,   6.25      ,
         6.66666667,   7.08333333,   7.5       ,   7.91666667,
         8.33333333,   8.75      ,   9.16666667,   9.58333333,  10.        ])


>>> np.logspace(0, 10, 10, base=numpy.e)
array([  1.00000000e+00,   3.03773178e+00,   9.22781435e+00,
         2.80316249e+01,   8.51525577e+01,   2.58670631e+02,
         7.85771994e+02,   2.38696456e+03,   7.25095809e+03,
         2.20264658e+04])
```

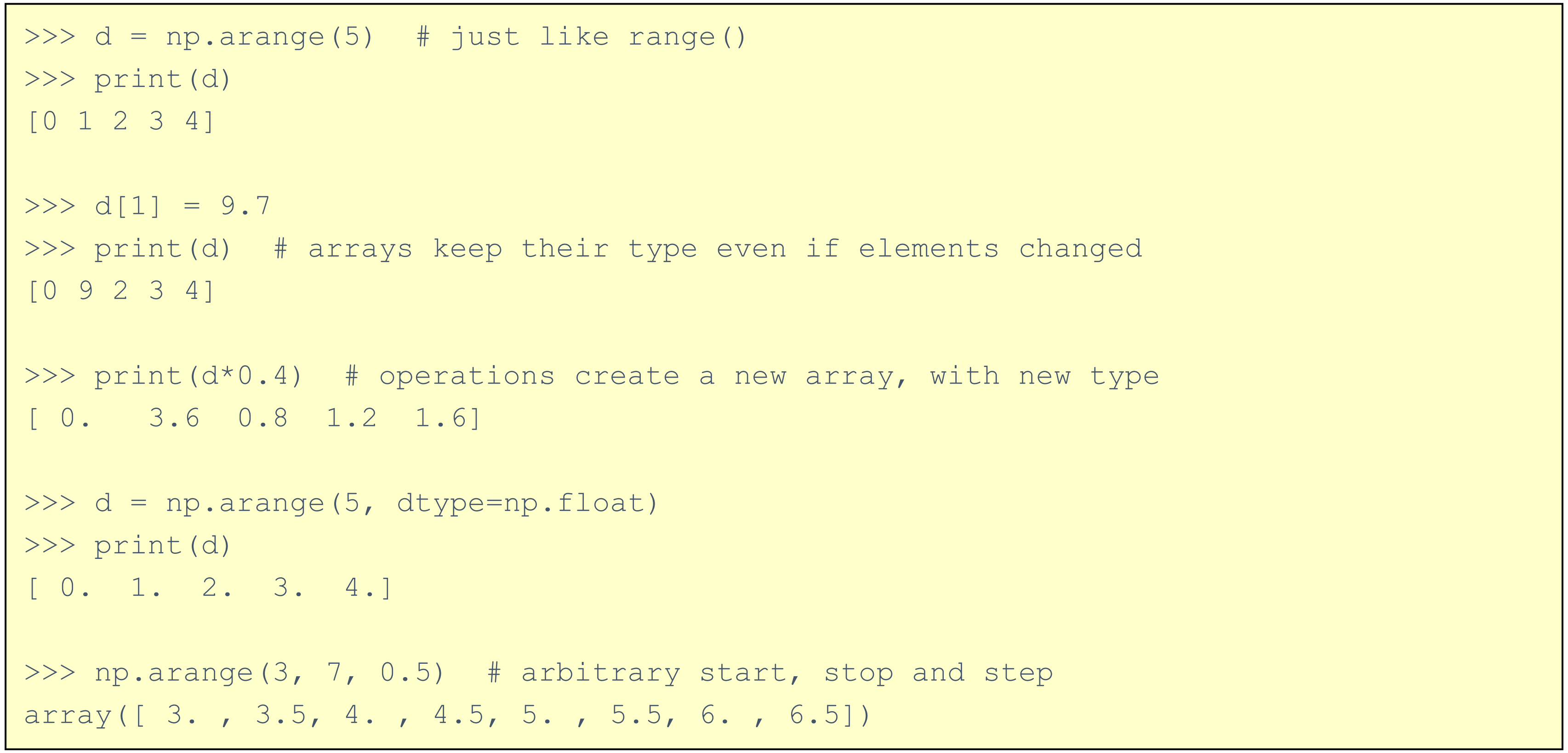
Numpy (Arreglos)

Programación II

```
# a diagonal matrix
>>> np.diag([1,2,3])
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])

>>> b = np.zeros(5)
>>> print(b)
[ 0.  0.  0.  0.  0.]
>>> b.dtype
dtype(‘float64’)
>>> n = 1000
>>> my_int_array = np.zeros(n, dtype=np.int)
>>> my_int_array.dtype
dtype(‘int32’)

>>> c = np.ones((3,3))
>>> c
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
```

Numpy (Arreglos: Creación y uso)

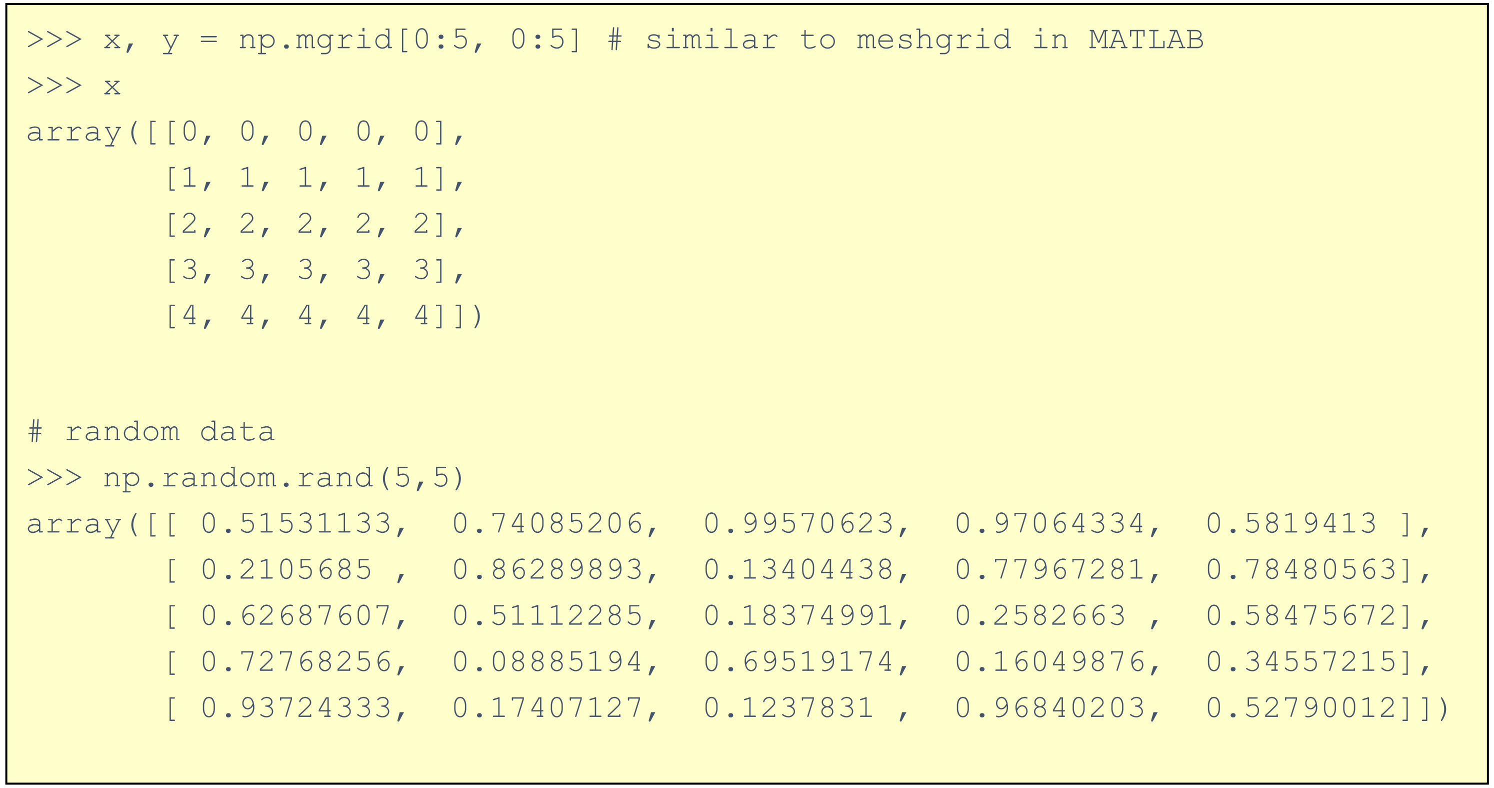
Programación II



```
>>> d = np.arange(5)  # just like range()
>>> print(d)
[0 1 2 3 4]

>>> d[1] = 9.7
>>> print(d)  # arrays keep their type even if elements changed
[0 9 2 3 4]

>>> print(d*0.4)  # operations create a new array, with new type
[ 0.   3.6  0.8  1.2  1.6]

>>> d = np.arange(5, dtype=np.float)
>>> print(d)
[ 0.  1.  2.  3.  4.]

>>> np.arange(3, 7, 0.5)  # arbitrary start, stop and step
array([ 3. , 3.5, 4. , 4.5, 5. , 5.5, 6. , 6.5])
```

Numpy (Arreglos: Creación y uso)

Programación II



```
>>> x, y = np.mgrid[0:5, 0:5] # similar to meshgrid in MATLAB
>>> x
array([[0, 0, 0, 0, 0],
       [1, 1, 1, 1, 1],
       [2, 2, 2, 2, 2],
       [3, 3, 3, 3, 3],
       [4, 4, 4, 4, 4]])


# random data
>>> np.random.rand(5,5)
array([[ 0.51531133,  0.74085206,  0.99570623,  0.97064334,  0.5819413 ],
       [ 0.2105685 ,  0.86289893,  0.13404438,  0.77967281,  0.78480563],
       [ 0.62687607,  0.51112285,  0.18374991,  0.2582663 ,  0.58475672],
       [ 0.72768256,  0.08885194,  0.69519174,  0.16049876,  0.34557215],
       [ 0.93724333,  0.17407127,  0.1237831 ,  0.96840203,  0.52790012]])
```

Numpy (Creando arreglos)

Programación II

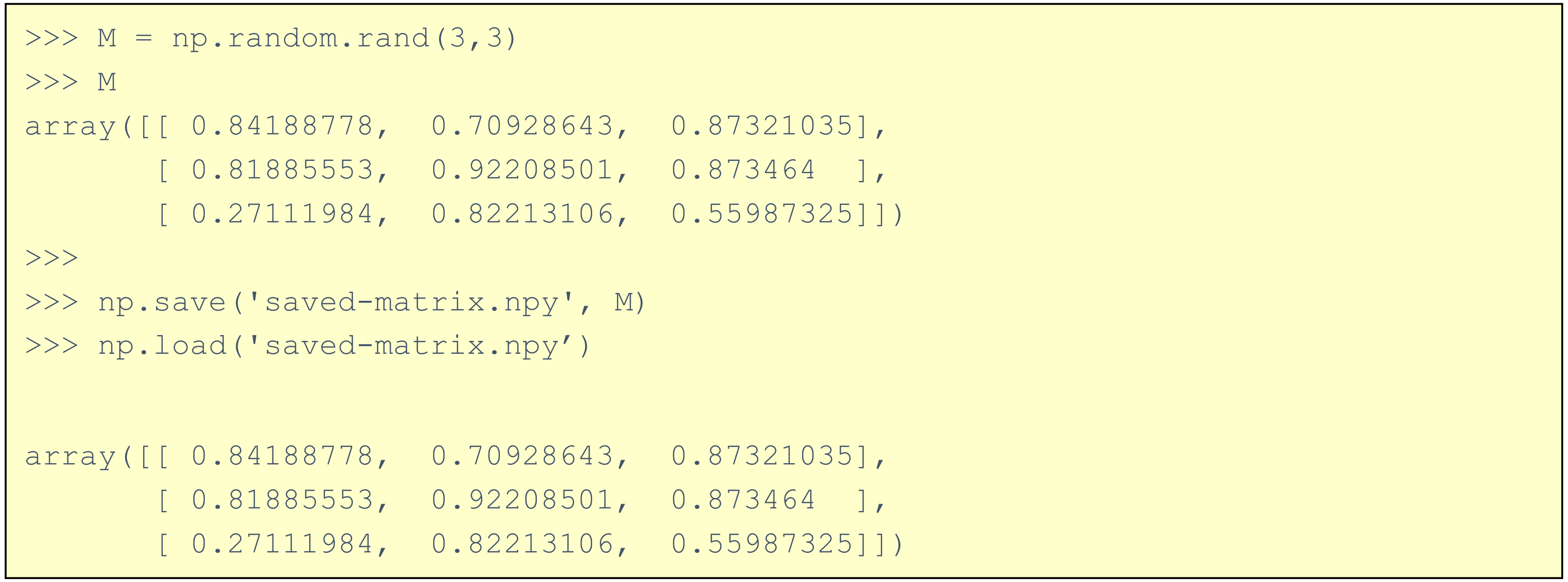
- File I/O

```
>>> os.system('head DeBilt.txt')
"Stn", "Datum", "Tg", "qTg", "Tn", "qTn", "Tx", "qTx"
001, 19010101,   -49, 00,   -68
001, 19010102,   -21, 00,   -36
001, 19010103,   -28, 00,   -79
001, 19010104,   -64, 00,   -91
001, 19010105,   -59, 00,   -84
001, 19010106,   -99, 00,  -115
001, 19010107,   -91, 00,  -122
001, 19010108,   -49, 00,   -94
001, 19010109,    11, 00,   -27
0

>>> data = np.genfromtxt('DeBil
>>> data.shape
(25568, 8)
```

```
>>> np.savetxt('datasaved.txt', data)
>>> os.system('head datasaved.txt')
1.000000000000000000e+00 1.901010100000000000e+07 -4.900000000000000000e+01
0.000000000000000000e+00 -6.800000000000000000e+01 0.000000000000000000e+00 -
2.200000000000000000e+01 4.000000000000000000e+01
1.000000000000000000e+00 1.901010200000000000e+07 -2.100000000000000000e+01
0.000000000000000000e+00 -3.600000000000000000e+01 3.000000000000000000e+01 -
1.300000000000000000e+01 3.000000000000000000e+01
1.000000000000000000e+00 1.901010300000000000e+07 -2.800000000000000000e+01
0.000000000000000000e+00 -7.900000000000000000e+01 3.000000000000000000e+01 -
5.000000000000000000e+00 2.000000000000000000e+01
```

Numpy (Creando arreglos)

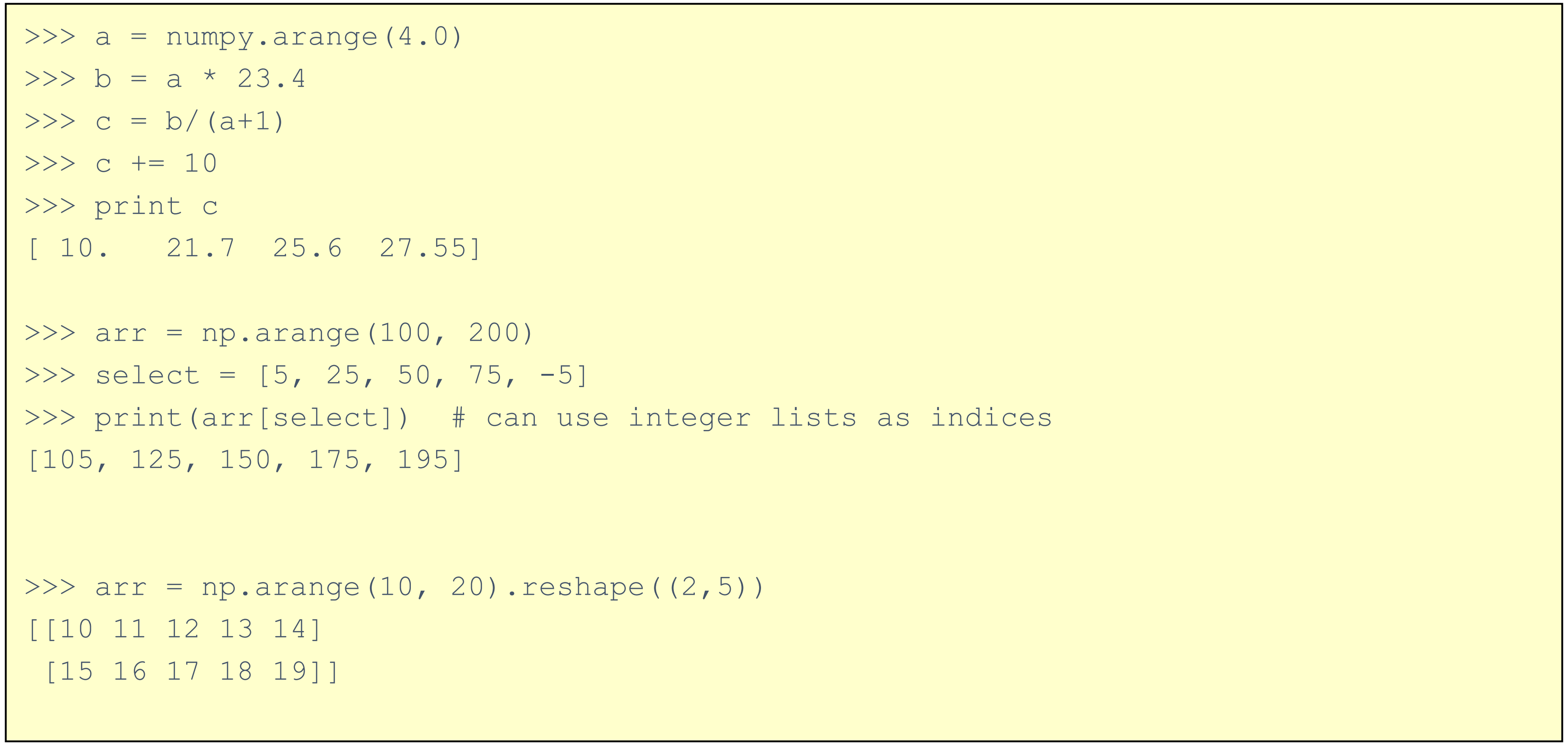
Programación II



```
>>> M = np.random.rand(3,3)
>>> M
array([[ 0.84188778,  0.70928643,  0.87321035],
       [ 0.81885553,  0.92208501,  0.873464  ],
       [ 0.27111984,  0.82213106,  0.55987325]])
>>>
>>> np.save('saved-matrix.npy', M)
>>> np.load('saved-matrix.npy’)


array([[ 0.84188778,  0.70928643,  0.87321035],
       [ 0.81885553,  0.92208501,  0.873464  ],
       [ 0.27111984,  0.82213106,  0.55987325]])
```

Numpy (Creando arreglos)

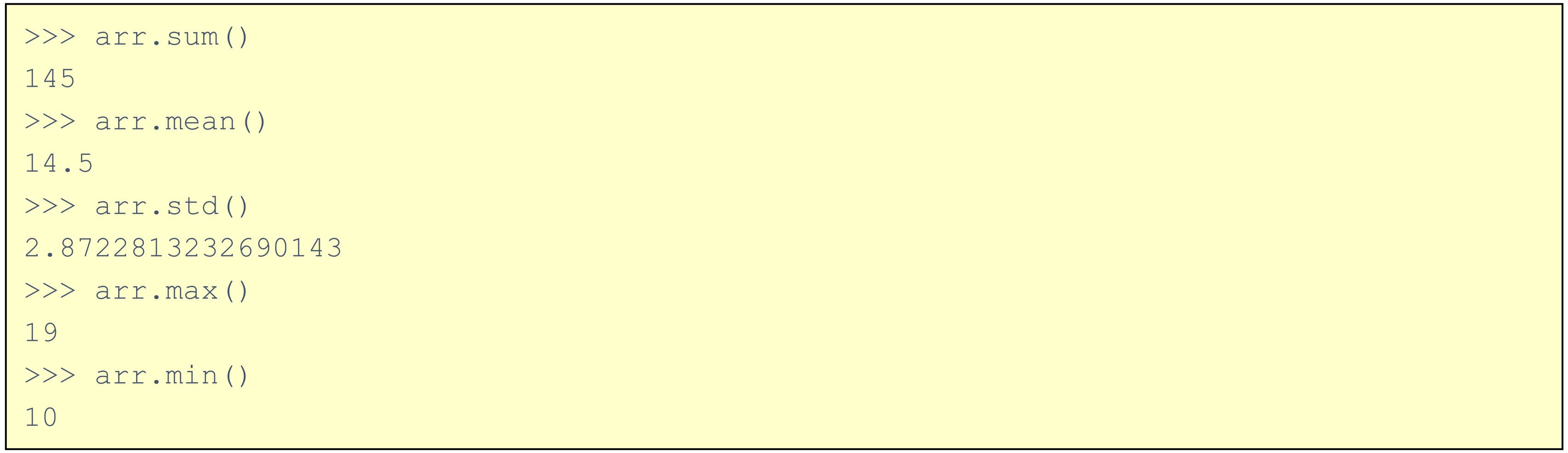
Programación II



```
>>> a = numpy.arange(4.0)
>>> b = a * 23.4
>>> c = b/(a+1)
>>> c += 10
>>> print c
[ 10.   21.7  25.6  27.55]

>>> arr = np.arange(100, 200)
>>> select = [5, 25, 50, 75, -5]
>>> print(arr[select])  # can use integer lists as indices
[105, 125, 150, 175, 195]


>>> arr = np.arange(10, 20).reshape((2,5))
[[10 11 12 13 14]
 [15 16 17 18 19]]
```

Numpy (Metodos)

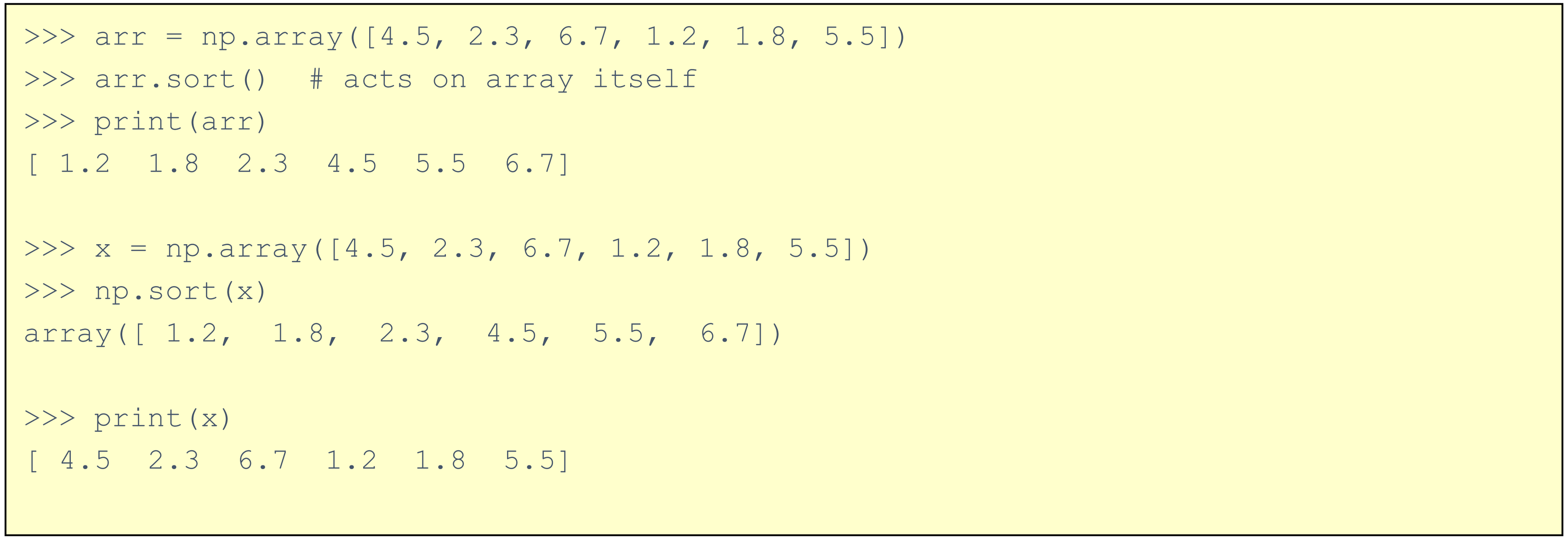
Programación II



```
>>> arr.sum()
145
>>> arr.mean()
14.5
>>> arr.std()
2.8722813232690143
>>> arr.max()
19
>>> arr.min()
10
```

Numpy (Metodo sort())

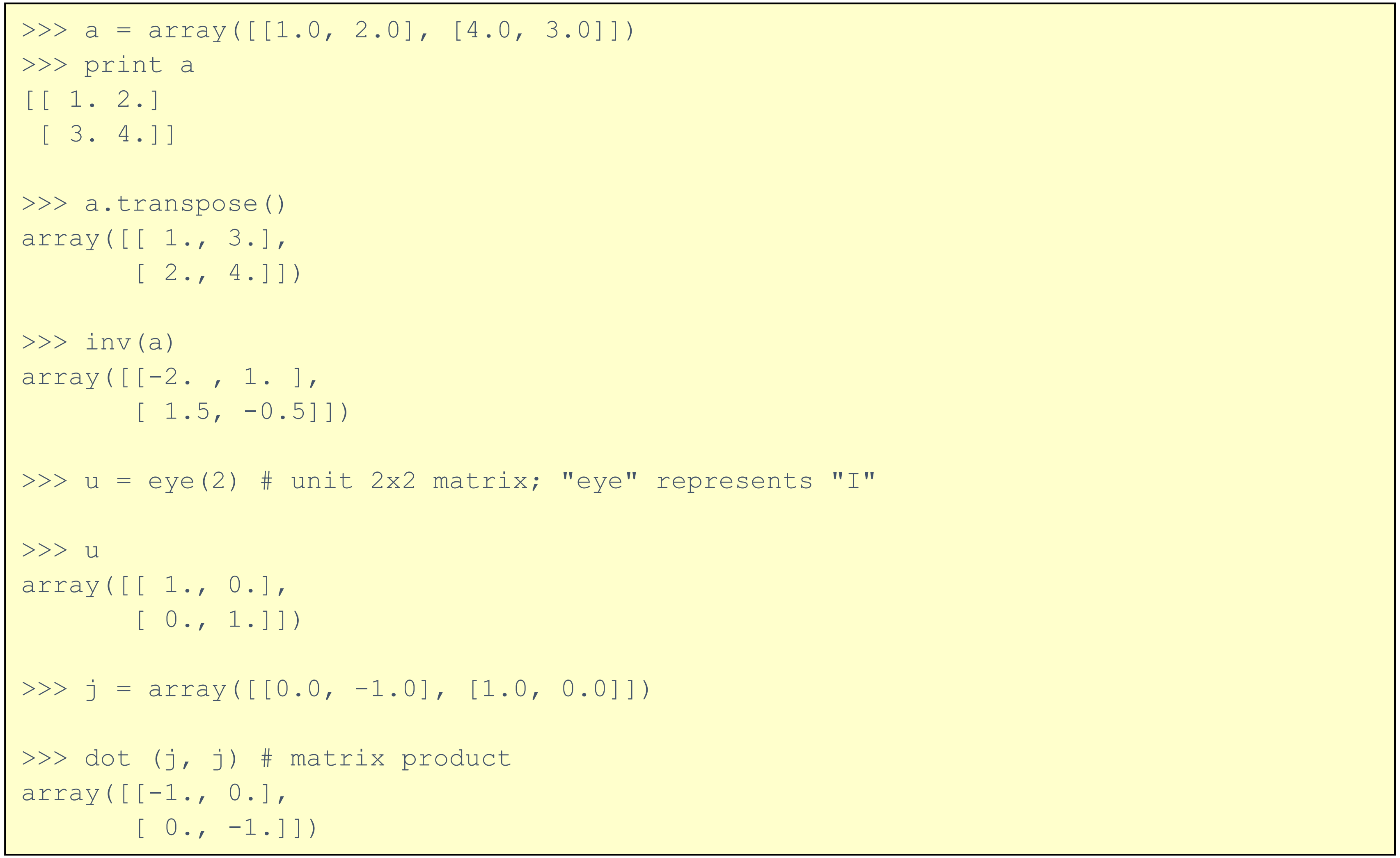
Programación II



```
>>> arr = np.array([4.5, 2.3, 6.7, 1.2, 1.8, 5.5])
>>> arr.sort()  # acts on array itself
>>> print(arr)
[ 1.2  1.8  2.3  4.5  5.5  6.7]

>>> x = np.array([4.5, 2.3, 6.7, 1.2, 1.8, 5.5])
>>> np.sort(x)
array([ 1.2,  1.8,  2.3,  4.5,  5.5,  6.7])

>>> print(x)
[ 4.5  2.3  6.7  1.2  1.8  5.5]
```

Numpy (Funciones)

Programación II



```
>>> a = array([[1.0, 2.0], [4.0, 3.0]])
>>> print a
[[ 1. 2.]
 [ 3. 4.]]

>>> a.transpose()
array([[ 1., 3.],
       [ 2., 4.]])

>>> inv(a)
array([[-2. , 1. ],
       [ 1.5, -0.5]])

>>> u = eye(2) # unit 2x2 matrix; "eye" represents "I"

>>> u
array([[ 1., 0.],
       [ 0., 1.]])

>>> j = array([[0.0, -1.0], [1.0, 0.0]])

>>> dot (j, j) # matrix product
array([[-1., 0.],
       [ 0., -1.]])
```

Numpy (Funciones)

Programación II

```
>>> 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.])

>>> a = np.array([[1, 2], [3, 4], [5, 6]], float)
>>> b = np.array([-1, 3], float)
>>> a
array([[ 1.,  2.],
       [ 3.,  4.],
       [ 5.,  6.]])
>>> b
array([-1.,  3.])
>>> a + b
array([[ 0.,  5.],
       [ 2.,  7.],
       [ 4.,  9.]])
```

```
>>> a = np.array([[1, 2], [3, 4], [5, 6]], float)
>>> b = np.array([-1, 3], float)

>>> a * a
array([[  1.,   4.],
       [  9.,  16.],
       [ 25.,  36.]])
>>> b * b
array([ 1.,  9.])
>>> a * b
array([[ -1.,   6.],
       [ -3.,  12.],
       [ -5.,  18.]])
>>>
```