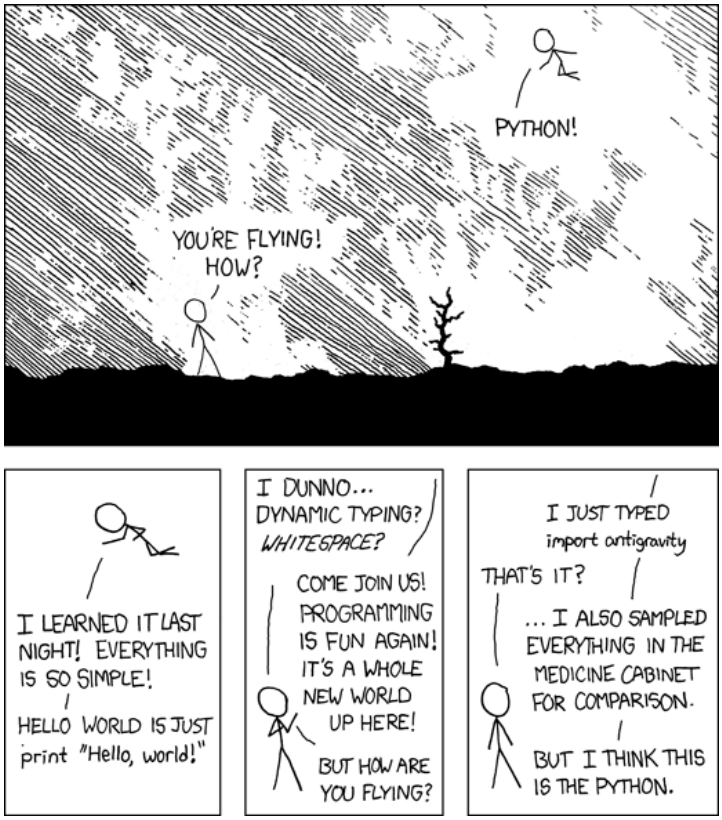
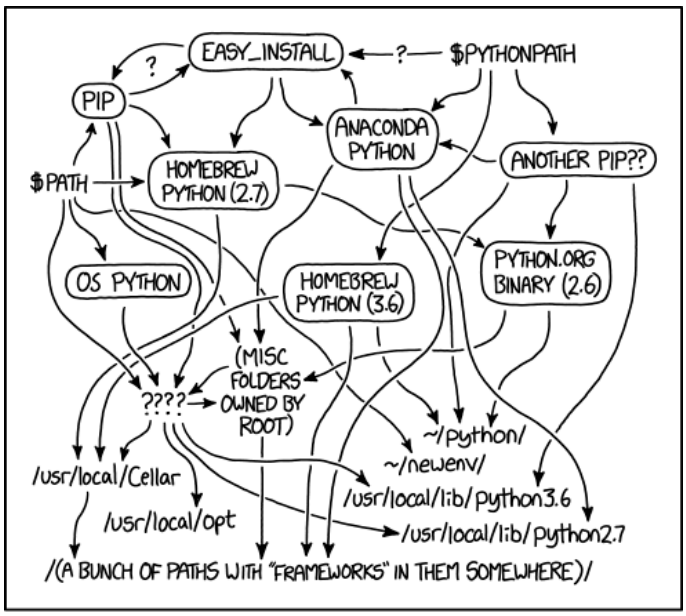
Python


PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

Ambientes



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

Explicación https://www.explainxkcd.com/wiki/index.php/1987:_Python_Environment

La figura quedó un poco vieja, hoy en día (2023) para aplicaciones de computación científica o aprendizaje automático, lo más conveniente es usar miniconda ya que permite utilizar cualquier versión de python (3.6,3.7,3.10,etc) y no tiene ninguna relación con el python del sistema. Otra ventaja, es que miniconda instala paquetes binarios además de paquetes de python y mantiene el ambiente tmb para los binarios y no chocan con el sistema.

```
!which python
!which pip
```

```
/usr/local/bin/python
/usr/local/bin/pip
```

```
!pip install jupyter
!which jupyter
```

```
Collecting jupyter
  Downloading jupyter-1.0.0-py2.py3-none-any.whl (2.7 kB)
Requirement already satisfied: notebook in /usr/local/lib/python3.10/dist-packages (from jupyter) (6.5.5)
Collecting qtconsole (from jupyter)
  Downloading qtconsole-5.5.1-py3-none-any.whl (123 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 123.4/123.4 kB 3.2 MB/s eta 0:00:00
Requirement already satisfied: jupyter-console in /usr/local/lib/python3.10/dist-packages (from jupyter) (6.1.0)
Requirement already satisfied: nbconvert in /usr/local/lib/python3.10/dist-packages (from jupyter) (6.5.4)
Requirement already satisfied: ipykernel in /usr/local/lib/python3.10/dist-packages (from jupyter) (5.5.6)
Requirement already satisfied: ipywidgets in /usr/local/lib/python3.10/dist-packages (from jupyter) (7.7.1)
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter) (0.2.0)
Requirement already satisfied: ipython>=5.0.0 in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter) (7.34.0)
Requirement already satisfied: traitlets>=4.1.0 in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter) (5.7.1)
Requirement already satisfied: jupyter-client in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter) (6.1.12)
Requirement already satisfied: tornado>=4.2 in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter) (6.3.3)
Requirement already satisfied: widgetsnbextension~=3.6.0 in /usr/local/lib/python3.10/dist-packages (from ipywidgets->jupyter) (3
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from ipywidgets->jupyter) (3
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from jupy
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-packages (from jupyter-console->jupyter) (2.16.1)
Requirement already satisfied: lxml in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter) (4.9.4)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter) (4.12.3)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter) (6.1.0)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter) (0.7.1)
Requirement already satisfied: entrypoints>=0.2.2 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter) (0.4)
Requirement already satisfied: jinja2>=3.0 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter) (3.1.3)
Requirement already satisfied: jupyter-core>=4.7 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter) (5.7.2)
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter) (0.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter) (2.1.5)
Requirement already satisfied: mistune<2,>=0.8.1 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter) (0.8.4)
```

```
Requirement already satisfied: nbclient>=0.5.0 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter) (0.10.0)
Requirement already satisfied: nbformat>=5.1 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter) (5.10.4)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter) (24.0)
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter) (1.5.1)
Requirement already satisfied: tinycss2 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter) (1.2.1)
Requirement already satisfied: pyzmq<25,>=17 in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter) (23.2.1)
Requirement already satisfied: argon2-cffi in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter) (23.1.0)
Requirement already satisfied: nest-asyncio>=1.5 in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter) (1.6.0)
Requirement already satisfied: Send2Trash>=1.8.0 in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter) (1.8.3)
Requirement already satisfied: terminado>=0.8.3 in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter) (0.18.1)
Requirement already satisfied: prometheus-client in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter) (0.20.0)
Requirement already satisfied: nbclassic>=0.4.7 in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter) (1.0.0)
Collecting qtpy>=2.4.0 (from qtconsole->jupyter)
  Downloading QtPy-2.4.1-py3-none-any.whl (93 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 93.5/93.5 kB 7.9 MB/s eta 0:00:00
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0->ipykernel->jupyt
Collecting jedi>=0.16 (from ipython>=5.0.0->ipykernel->jupyter)
  Downloading jedi-0.19.1-py2.py3-none-any.whl (1.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 13.2 MB/s eta 0:00:00
Requirement already satisfied: decorator in /usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0->ipykernel->jupyter) (4.
Requirement already satisfied: pickleshare in /usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0->ipykernel->jupyter) (
Requirement already satisfied: backcall in /usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0->ipykernel->jupyter) (0.2
Requirement already satisfied: matplotlib-inline in /usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0->ipykernel->jupy
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0->ipykernel->jupyter) (
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.10/dist-packages (from jupyter-client->ipykernel->j
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.10/dist-packages (from jupyter-core>=4.7->nbconvert->j
Requirement already satisfied: jupyter-server>=1.8 in /usr/local/lib/python3.10/dist-packages (from nbclassic>=0.4.7->notebook->j
Requirement already satisfied: notebook-shim>=0.2.3 in /usr/local/lib/python3.10/dist-packages (from nbclassic>=0.4.7->notebook->
```

Observaciones y operaciones

dir() devuelve todos los métodos de un objeto, o los objetos en el ambiente

```
dir(dict())
```

```
['__class__',
 '__class_getitem__',
 '__contains__',
 '__delattr__',
 '__delitem__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__ior__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__ne__',
 '__new__',
 '__or__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__reversed__',
 '__ror__',
 '__setattr__',
 '__setitem__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'clear',
 'copy',
 'fromkeys',
 'get',
 'items',
 'keys',
 'pop',
 'popitem',
```

```
 'setdefault',
 'update',
 'values']
```

```
dir()
```

```
['In',
 'Out',
 '_',
 '_9',
 '__',
 '___',
 '__builtin__',
 '__builtins__',
 '__doc__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 '_dh',
 '_exit_code',
 '_i',
 '_i1',
 '_i10',
 '_i2',
 '_i3',
 '_i4',
 '_i5',
 '_i6',
 '_i7',
 '_i8',
 '_i9',
 '_ih',
 '_ii',
 '_iii',
 '_oh',
 'exit',
 'get_ipython',
 'quit']
```

Python es interpetado. En el siguiente caso hay error? esto permite ver como funciona la compilación-interpretación en python.

```
a = 7
if a > 0:
    print("si")
else:
    prin("no")
```

```
si
```

```
a = 7
if a > 0:
    print("si")
else:
    prin("no"
```

```
  File "<ipython-input-12-3aa24f38cafd>", line 5
    prin("no"
             ^
SyntaxError: incomplete input
```

La asignacion a un mismo nombre de variable crea objetos nuevos, diferente al modelo de c, en este caso los objetos en desuso se van borrando solos (garbage collector)

```
a = 12123156
mem_a_before = id(a)
a = 1345646
mem_a_after = id(a)
assert(id(mem_a_before) == id(mem_a_after))
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-13-25a4b4333245> in <cell line: 5>()
      3 a = 1345646
      4 mem_a_after = id(a)
----> 5 assert(id(mem_a_before) == id(mem_a_after))

AssertionError:
```

Para poder tener un garbage collector tiene que guardar cuantas veces se hace referencia a un objeto

```
import sys
print(sys.getrefcount(a))
b = [a]
print(sys.getrefcount(a))
```

```
3
4
```

En python todo objeto se pasa por "referencia"

```
def modificar_lista(l):
  l.append('manzana')

l = ['chocolate', 'alfajorcito']
modificar_lista(l)
l
```

```
['chocolate', 'alfajorcito', 'manzana']
```

```
def printid(s):
  print(id(s))

a = 'string'
print(id(a))
printid(a)
```

```
139634100855344
139634100855344
```

Cuando se asigna nuevamente se crea un nuevo objeto

```
def modificar_lista(l):
  l = ['manazana']

l = ['chocolate', 'alfajorcito']
modificar_lista(l)
l
```

```
['chocolate', 'alfajorcito']
```

Debugear con set_trace y los comandos de ipdb

```
from IPython.core.debugger import set_trace
a = 7
b = 2
if a > 0:
    set_trace()
    a = a + 2
    set_trace()
else:
    set_trace()
    b = a + b
    set_trace()
```

Numpy

Numpy tiene su propio review en la revista NATURE

https://www.nature.com/articles/s41586-020-2649-2

nature

Explore content | About the journal | Publish with us

nature > review articles > article

Review Article | Open Access | Published: 16 September 2020

Array programming with NumPy

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, ... Travis E. Oliphant + Show authors

Nature **585**, 357–362 (2020) | Cite this article

301k Accesses | **5362** Citations | **1882** Altmetric | Metrics

Abstract

Array programming provides a powerful, compact and expressive syntax for accessing, manipulating and operating on data in vectors, matrices and higher-dimensional arrays. NumPy is the primary array programming library for the Python language. It has an essential role in research analysis pipelines in fields as diverse as physics, chemistry, astronomy, geoscience, biology, psychology, materials science, engineering, finance and economics. For example, in astronomy, NumPy was an important part of the software stack used in the discovery of gravitational waves[1] and in the first imaging of a black hole[2]. Here we review how a few fundamental array concepts lead to a simple and powerful programming paradigm for

```
import numpy as np
```

```
dir(np)
```

```
['ALLOW_THREADS',
 'BUFSIZE',
 'CLIP',
 'DataSource',
 'ERR_CALL',
 'ERR_DEFAULT',
 'ERR_IGNORE',
 'ERR_LOG',
 'ERR_PRINT',
 'ERR_RAISE',
 'ERR_WARN',
 'FLOATING_POINT_SUPPORT',
 'FPE_DIVIDEBYZERO',
 'FPE_INVALID',
 'FPE_OVERFLOW',
 'FPE_UNDERFLOW',
 'False_',
```

```
 'Inf',
 'Infinity',
 'MAXDIMS',
 'MAY_SHARE_BOUNDS',
 'MAY_SHARE_EXACT',
 'NAN',
 'NINF',
 'NZERO',
 'NaN',
 'PINF',
 'PZERO',
 'RAISE',
 'RankWarning',
 'SHIFT_DIVIDEBYZERO',
 'SHIFT_INVALID',
 'SHIFT_OVERFLOW',
 'SHIFT_UNDERFLOW',
 'ScalarType',
 'True_',
 'UFUNC_BUFSIZE_DEFAULT',
 'UFUNC_PYVALS_NAME',
 'WRAP',
 '_CopyMode',
 '_NoValue',
 '_UFUNC_API',
 '__NUMPY_SETUP__',
 '__all__',
 '__builtins__',
 '__cached__',
 '__config__',
 '__deprecated_attrs__',
 '__dir__',
 '__doc__',
 '__expired_functions__',
 '__file__',
 '__former_attrs__',
 '__future_scalars__',
 '__getattr__',
 '__git_version__',
 '__loader__',
 '__name__',
```

Tipos de numpy

```
np.bool_, np.byte, np.ubyte, np.short, np.ushort, np.intc, np.uintc, np.int_, np.uint, np.longlong, np.ulonglong, np.half, np.float16,
```

```
(numpy.bool_,
 numpy.int8,
 numpy.uint8,
 numpy.int16,
 numpy.uint16,
 numpy.int32,
 numpy.uint32,
 numpy.int64,
 numpy.uint64,
 numpy.longlong,
 numpy.ulonglong,
 numpy.float16,
 numpy.float16,
 numpy.float32,
 numpy.float64,
 numpy.longdouble,
 numpy.complex64,
 numpy.complex128,
 numpy.clongdouble)
```

Arrays

Los arrays son el tipo de datos más importante que provee NumPy. En su versión más básica representan vectores pero pueden tener más dimensiones y representar matrices y tensores en general.

El tipo se llama ndarray pero también se lo conoce en la librería simplemente como array.

En su versión más simple podemos pensar que no es más que una lista de python pero:

- A diferencia de las listas en python, solo pueden tener un tipo de datos adentro.

- Existen un montón de operaciones matemáticas definidas y optimizadas para trabajar con este tipo de datos.

Al ser una clase de python, además de métodos tiene atributos. Veamos algunos de ellos

```
an_array = np.arange(20)
an_array
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19])
```

```
an_array.shape
```

```
(20,)
```

```
an_array.ndim
```

```
1
```

```
an_array.size
```

```
20
```

```
an_array.dtype
```

```
dtype('int64')
```

Armando arrays

Por un lado, tenemos el constructor de la clase que admite como parámetro listas de valores .Por otro lado, existen diversas funciones que crean arrays. Veamos algunas

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

```
array([1, 2, 3, 4])
```

```
np.arange(10,30,2)
```

```
array([10, 12, 14, 16, 18, 20, 22, 24, 26, 28])
```

```
np.linspace(0.1,0.5,4)
```

```
array([0.1       , 0.23333333, 0.36666667, 0.5       ])
```

```
np.logspace(1,10,10)
```

```
array([1.e+01, 1.e+02, 1.e+03, 1.e+04, 1.e+05, 1.e+06, 1.e+07, 1.e+08,
       1.e+09, 1.e+10])
```

```
np.zeros((3,2))
```

```
array([[0., 0.],
       [0., 0.],
       [0., 0.]])
```

```
np.ones((2,3))
```

```
array([[1., 1., 1.],
       [1., 1., 1.]])
```

```
np.random.rand(3,2) # random entre 0 y 1
```

```
array([[0.3398017 , 0.54306144],
       [0.04379483, 0.84854646],
       [0.51480131, 0.36702182]])
```

```
np.random.randint(1,10,(3,2)) # enteros random entre [1 y 10)
```

```
array([[1, 5],
       [9, 2],
       [5, 5]])
```

```
np.random.standard_normal((8,2))
```

```
array([[ 1.10032851, -0.12632543],
       [ 1.94452798, -0.22144086],
       [-0.41193579,  0.99722222],
       [ 0.43732368,  0.15380123],
       [-0.23332081, -1.07509161],
       [ 1.22743252,  1.39926978],
       [ 0.72040099, -0.33557141],
       [ 0.96141869,  0.0137898 ]])
```

Propiedades de un arreglo

```
an_array = np.ones((2,3))
print(an_array.shape, an_array.ndim, an_array.size, an_array.dtype)
```

```
(2, 3) 2 6 float64
```

Se puede cambiar el tipo de un array

```
np.arange(0,10,0.5)
```

```
array([0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5, 5. , 5.5, 6. ,
       6.5, 7. , 7.5, 8. , 8.5, 9. , 9.5])
```

```
np.arange(0,10,0.5).astype(np.int32)
```

```
array([0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9],
      dtype=int32)
```

Matrices

Las matrices no son más que array con 2 dimensiones. Si bien existe un tipo específico para matrices en numpy cayó en desuso (y obsolescencia).

```
a_matrix = np.array([[ 0,  1,  2,  3],
                     [ 4,  5,  6,  7],
                     [ 8,  9, 10, 11],
                     [12, 13, 14, 15]])
a_matrix
```

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

```
print(a_matrix.shape, a_matrix.ndim, a_matrix.size, a_matrix.dtype)
```

```
(4, 4) 2 16 int64
```

Matrices especiales

```
np.eye(3) # Identidad de 3x3
```

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

```
np.diag([77,90]) # Matriz diagonal
```

```
array([[77,  0],
       [ 0, 90]])
```

```
np.diag(np.ones(3),1) # Superdiagonal 1
```

```
array([[0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.],
       [0., 0., 0., 0.]])
```

```
np.vander(np.arange(4),increasing=True) # Matriz de Vandermonde
```

```
array([[ 1,  0,  0,  0],
       [ 1,  1,  1,  1],
       [ 1,  2,  4,  8],
       [ 1,  3,  9, 27]])
```

Reshape

Algunos métodos que modifican las dimensiones:

- reshape: Devuelve un nuevo array con las dimensiones indicadas cómo parámetro. Si alguno de los parámetros es igual a -1, se calculan las dimensiones para que sea factible el cambio.
- resize: el mismo efecto que “reshape” pero modifica el array en vez de devolver uno nuevo.
- T: sirve para transponer una matriz.
- ravel, flattened: “aplana” el array devolviendo todo en una sola dimensión.

```
a_matrix.reshape(1,16)
```

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

```
a_matrix.T
```

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

```
a_matrix.ravel()
```

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

```
a_matrix.resize(1,16)
a_matrix
```

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

Operaciones con arrays

Existen muchísimas operaciones definidas para arrays.

Operadores básicos:

Los operadores básicos de sumas, restas, potencias, etc se encuentran sobrecargados.

```
v = np.arange(1,7)
w = np.ones(6)
v, w
```

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

```
v + w
```

```
array([2., 3., 4., 5., 6., 7.])
```

```
v > w
```

```
array([False,  True,  True,  True,  True,  True])
```

```
v - w
```

```
array([0., 1., 2., 3., 4., 5.])
```

```
v ** 2
```

```
array([ 1,  4,  9, 16, 25, 36])
```

```
np.sqrt(v)
```

```
array([1.        , 1.41421356, 1.73205081, 2.        , 2.23606798,
       2.44948974])
```

```
print(np.sin(v), "\n", np.cos(v), "\n", np.floor(np.cos(v)), "\n", np.round(np.cos(v)))
```

```
[ 0.84147098  0.90929743  0.14112001 -0.7568025  -0.95892427 -0.2794155 ]
 [ 0.54030231 -0.41614684 -0.9899925  -0.65364362  0.28366219  0.96017029]
 [ 0. -1. -1. -1.  0.  0.]
 [ 1. -0. -1. -1.  0.  1.]
```

Operaciones entre vectores

```
np.dot(v,w) #Producto iterno (o escalar)
```

```
21.0
```

```
np.outer(v,w) #Producto externo
```

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

Multiplicación de matrices

Existe la multiplicación clásica entre matrices (con el símbolo @) o el producto elemento a elemento (producto Hadamard)

```
M1 = np.arange(1,10).reshape(3,3)
M2 = np.ones((3,3))
M1, M2
```

```
(array([[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]]),
 array([[1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.]]))
```

```
M1 * M2 #Producto Hadamard o elemento a elemento
```

```
array([[1., 2., 3.],
       [4., 5., 6.],
       [7., 8., 9.]])
```

```
np.matmul(M1,M2) #Producto interno de matrices
```

```
array([[ 6.,  6.,  6.],
       [15., 15., 15.],
       [24., 24., 24.]])
```

```
M1 @ M2 #Producto interno de matrices
```

```
array([[ 6.,  6.,  6.],
       [15., 15., 15.],
       [24., 24., 24.]])
```

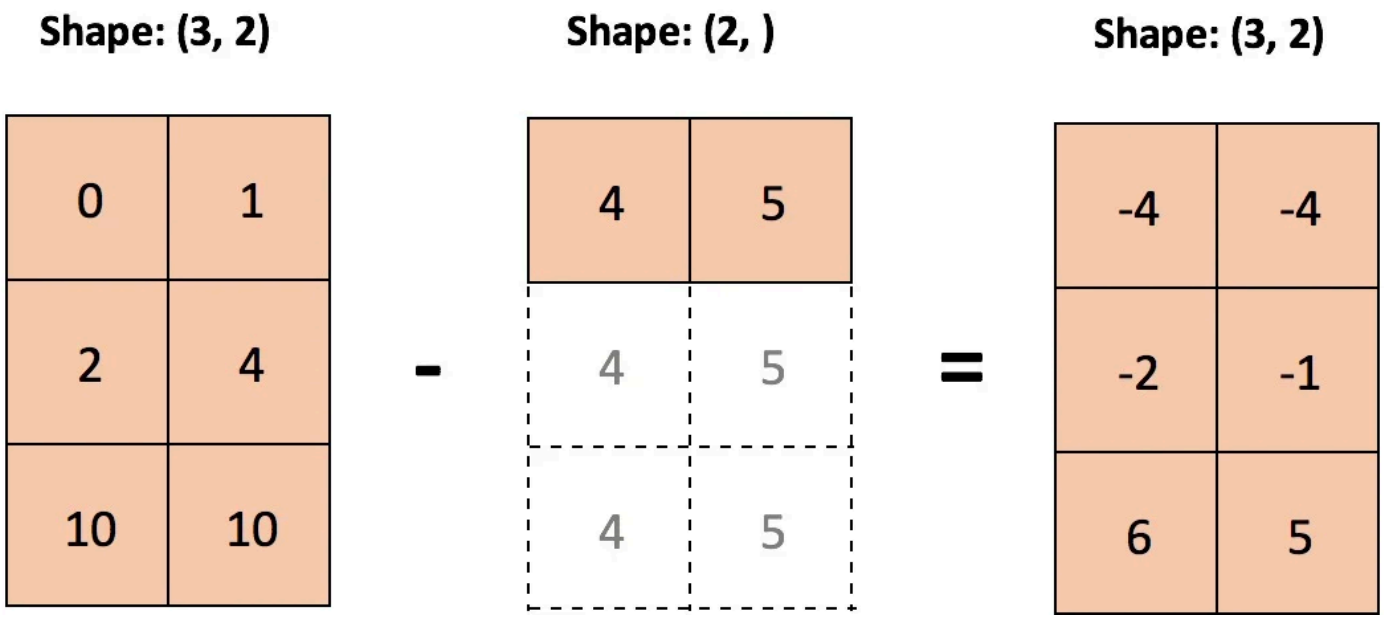
EJERCICIO: Multiplicar $M_1 v$ siendo $v = (1, 1, 1)$ como columna

```
...
```

```
Ellipsis
```

Broadcasting

Dado un operador, si las matrices no tienen el mismo tamaño Numpy tiene reglas definidas para "estirar", en alguna dimensión conveniente, una de las matrices y así arreglarselas para computar la operación. Esto se lo conoce como **Broadcasting**. Más info https://numpy.org/doc/stable/user/basics.broadcasting.html



EJERCICIO: Regenerar el ejemplo de la figura

```
Empieza a programar o a crear código con IA.
```

Universal functions y Performance

Son todas aquellas funciones que operan elemento a elemento sobre un array de manera predefinida. Varios de los ejemplos vistos en las slides anteriores caen en este tipo de funciones.

Estas funciones se dicen que son vectorizadas y están particularmente optimizadas para hacer muy rápidamente la misma función sobre todas las posiciones de un vector de manera muy rápida (sí, numpy usa SIMD https://numpy.org/doc/stable/reference/simd/index.html)

Se podría perfectamente obtener el mismo resultado iterando el array y aplicando la función requerida, pero tomaría mucho más tiempo de cómputo.

```
import time
import math

def compute_cos_native(array):
    result = []
    for element in array:
        result.append(math.cos(element))
    return result

def compute_cos_numpy(array):
    result = np.cos(array)
    return result



an_array = np.arange(10000000)
start_time_native = time.time()
result_native = compute_cos_native(an_array)
end_time_native = time.time()

start_time_numpy = time.time()
result_numpy = compute_cos_numpy(an_array)
end_time_numpy = time.time()
print("Tiempo sin numpy:", end_time_native - start_time_native)
print("Tiempo con numpy:", end_time_numpy - start_time_numpy)
assert(np.allclose(result_native, result_numpy))
```

```
Tiempo sin numpy: 2.7974767684936523
Tiempo con numpy: 0.2026679515838623
```

Numpy tiene una función "vectorize" es acazo una forma de ganar performance? Pues veamos

```
def cos_native(x):
    return math.cos(x)
vectorized_cos = np.vectorize(cos_native)


an_array = np.arange(10000)
```

Usamos el magick command %timeit para calcular promedios de tiempos de ejecuciones

```
%timeit compute_cos_native(an_array)
```

```
2.56 ms ± 62.1 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

```
%timeit vectorized_cos(an_array)
```

```
3.77 ms ± 1.15 ms per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

```
%timeit np.cos(an_array)
```

```
227 µs ± 61.4 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

Numba

Podemos usar Numba para compilar funciones escritas en puro python + numpy y lograr mejoras en performance

https://numba.pydata.org/numba-doc/latest/user/jit.html

```
def matvecmul(M,x):
    out = np.zeros_like(x)
    for i in range(M.shape[0]):
        for j in range(M.shape[1]):
            out[i] += M[i,j]*x[j]
    return out
```

```
M = np.random.rand(100,100)
x = np.random.rand(100,1)
```

```
np.allclose(matvecmul(M,x),M@x)
```

```
True
```

```
%timeit matvecmul(M,x)
```

```
38.7 ms ± 1.1 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

```
from numba import jit

def matvecmul_lento(M,x):
    out = np.zeros_like(x)
    for i in range(M.shape[0]):
        for j in range(M.shape[1]):
            out[i] += M[i,j]*x[j]
    return out

@jit(nopython=True)
def matvecmul(M,x):
    out = np.zeros_like(x)
    for i in range(M.shape[0]):
        for j in range(M.shape[1]):
            out[i] += M[i,j]*x[j]
    return out
```

```
matvecmul(M,x);
```

```
%timeit matvecmul_lento(M,x)
```

```
38 ms ± 529 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

```
%timeit matvecmul(M,x)
```

```
1.08 ms ± 42.2 µs per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
%timeit M@x
```

```
8.91 µs ± 3.21 µs per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

Acceso

Los arrays en NumPy se pueden acceder a posiciones particulares o mediante slicing de maneras parecidas a las listas nativas.

```
an_array = np.arange(0,40,2)
an_array
```

```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32,
       34, 36, 38])
```

```
an_array[6]
```

```
12
```

```
start = 2
end = 18
step = 2

an_array[start:end:step]
```

```
array([ 4,  8, 12, 16, 20, 24, 28, 32])
```

```
# Son equivalentes
#an_array[0:end:]
#an_array[:end:]
an_array[:end]
```

```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32,
       34])
```

```
an_array[:-1]
```

```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32,
       34, 36])
```

```
an_array[::]
```

```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32,
       34, 36, 38])
```

```
an_array[::-1]
```

```
array([38, 36, 34, 32, 30, 28, 26, 24, 22, 20, 18, 16, 14, 12, 10,  8,  6,
        4,  2,  0])
```

```
for element in an_array[:5]:
    print(element)
```

```
0
2
4
6
8
```

```
a_matrix = np.arange(16).reshape((4,4))
a_matrix
```

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

```
a_matrix[:,2]
```

```
array([ 2,  6, 10, 14])
```

```
a_matrix[:,1:3]
```

```
array([[ 1,  2],
       [ 5,  6],
       [ 9, 10],
       [13, 14]])
```

```
a_matrix[-1,:]
```

```
array([12, 13, 14, 15])
```

```
for row in a_matrix:
    print(row)
```

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

```
a_matrix[1:3,2:4]
```

```
array([[ 6,  7],
       [10, 11]])
```

Máscaras

Es posible seleecionar elementos de un arreglo con otro arreglo de booleanos

```
an_array = np.arange(4)
mask = np.array([False,True,True,False])
an_array[mask]
```

```
array([1, 2])
```

Las operaciones con máscaras son más rápidas que operar elemento a elemento con condiciones

```
an_array = np.arange(100)
an_array
```

```
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, 50,
       51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
       68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
       85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99])
```

```
np.array([x for x in an_array if x>50])
```

```
array([51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
       68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
       85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99])
```

```
an_array[an_array>50]
```

```
array([51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
       68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
       85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99])
```

```
%timeit np.array([x for x in an_array if x>50])
```

```
18.5 µs ± 3.11 µs per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

```
%timeit an_array[an_array>50]
```

```
2.03 µs ± 66.9 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

Tipos de asignación

Referencia

```
an_array = np.arange(10)
an_array
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
another_array = an_array
```

```
another_array[3] = 8
an_array
```

```
array([0, 1, 2, 8, 4, 5, 6, 7, 8, 9])
```

Vista

```
an_array = np.arange(10)
an_array
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
another_array = an_array.view()
```

```
another_array = another_array.reshape((5,2))
another_array
```

```
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7],
       [8, 9]])
```

```
an_array[3] = 8
```

```
another_array
```

```
array([[0, 1],
       [2, 8],
       [4, 5],
       [6, 7],
       [8, 9]])
```

Copia

```
another_array = np.arange(10)
```

```
an_array = another_array.copy()
```

```
an_array[3] = 8
```

```
an_array, another_array
```

```
(array([0, 1, 2, 8, 4, 5, 6, 7, 8, 9]), array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]))
```

Operaciones matemáticas

```
np.sum(np.arange(1000)) # suma un arreglo
```

```
499500
```

```
arr = np.arange(16).reshape(4,4,)
np.sum(arr,0) #  de un arreglo 2d, suma solo un eje, el 0, el primero
```

```
array([24, 28, 32, 36])
```

```
np.diff(np.arange(10)) # resta elementos consecutivos
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1])
```

```
np.cumsum(np.arange(10)) # suma acumulada
```

```
array([ 0,  1,  3,  6, 10, 15, 21, 28, 36, 45])
```

Álgebra Lineal

Numpy utiliza BLAS y LAPACK para las operaciones de álgebra lineal

https://numpy.org/doc/stable/reference/routines.linalg.html#module-numpy.linalg

Descomposiciones, autovalores, normas, determinante, rango, sistemas de ecuaciones, inversa

```
import numpy.linalg as lng
```

```
A = np.random.randint(1,10,(3,3))
A
```

```
array([[9, 5, 4],
       [4, 3, 4],
       [5, 5, 6]])
```

```
lng.inv(A)  # Inversa
```

```
array([[ 0.11111111,  0.55555556, -0.44444444],
       [ 0.22222222, -1.88888889,  1.11111111],
       [-0.27777778,  1.11111111, -0.38888889]])
```

Resolver sistemas de ecuaciones

```
A = np.random.randint(-9,10,(5,5))
x = np.random.randint(-9,10,(5,1))
b = A@x
A, x, b
```

```
(array([[ 1, -7, -8,  3, -1],
        [-4, -4, -5,  7, -3],
        [-5, -5, -6,  3,  5],
        [-8, -4, -8, -7, -5],
        [-4,  1,  5, -5,  2]]),
 array([[ 8],
        [-7],
        [-6],
        [-6],
        [ 4]]),
 array([[ 83],
        [-28],
        [ 33],
        [ 34],
        [-31]]))
```

```
x_solve = lng.solve(A,b)
print(x_solve)
np.allclose(x,x_solve)
```

```
[[ 8.]
 [-7.]
 [-6.]
 [-6.]
 [ 4.]]
True
```

Ejercicios

Ejercicio 1

Guarde en una lista todos los numeros menores a 100 que sean pares, por un lado con un for y por otro construyendo un arreglo de numpy con un método apropiado en una linea

```
numeros_pares_menores_a_100 = []
for ...
    ...
    numeros_pares_menores_a_100.append(...)

numeros_pares_menores_a_100 = np.
```

Ejercicio 2

Dada una secuencia aleatoria de valores False y True, contar cuantas transiciones False->True hay. Hacerlo con puro python y puro numpy.

```
secuencia = list(np.random.choice([False, True], size=100000))
```

Ejercicio 3

Un número palíndromo se lee del derecho y el reves de la misma forma. El palíndromo hecho por el producto de dos numeros de dos digitos mas grande es 9009 pues es producto de $91 * 99$ (https://projecteuler.net/problem=4)

Escriba un código para crear el palindromo más grande hecho por el producto de 2 numeros de 3 digitos. Hacerlo usando puro python y puro numpy.

Ejercicio 4

Dados x_1, ..., x_n una muestra de una variable aleatoria, implementar rutinas que calculen la media y la varianza utilizando operaciones vectoriales.

```
def mean(x):
    # COMPLETAR

def var(x):
    # COMPLETAR
```

```
N = 4
x = np.array(np.random.rand(N,1))
print("x:" + str(x) + "\n media: " + str(mean(x)) + " varianza: " + str(var(x)))
assert(np.abs(mean(x) - np.mean(x)) < 1e-6 )
assert(np.abs(var(x) - np.var(x)) < 1e-6 )
print("OK")
```

Ejercicio 5

Sea A matriz en $R^{m \times n}$

1. Demostrar que $A^t A$ y AA^t son simétricas
2. Implementar una rutina que dada una matriz cuadrada verifique si la misma es simétrica

```
def esSimetrica(A):
    # COMPLETAR
```

```
A = np.array([[1,2],[2,6],[3,7],[4,8]])
B = np.random.rand(4,4)

print("A: \n"  + str(A))
print("B: \n" + str(B))
assert(esSimetrica(A @ A.T))
assert(esSimetrica(B.T @ B))
print("OK")
```

Ejercicio 6

Analizar la función implementada en el item anterior con la matriz B generada de la siguiente forma:

```
A = np.random.rand(4,4)
B = A.T * A * 0.1 / 0.1
```

Analizar el resultado, revisar la implementación y (eventualmente) reimplementar la función.

```
D = np.array(np.diag([1,2]))
I = np.array(np.eye(5))
A = np.random.rand(4,4)

print(A)
assert(not(esSimetrica(A)))

# Ojo! usamos @ para el producto matricial
assert(esSimetrica(A.T@(A*0.10)/0.10))
assert(esSimetrica(D) and esSimetrica(I))
print("Ok")
```

Ejercicio 7

Sean A, B en $R^{n \times n}$, con n par y B triangular inferior,

1. Realizar la multiplicación AB por bloques, partiendo ambas matrices en bloques de tamaño n/2.
2. Implementar una rutina que realice la multiplicación por bloques, **evitando cuentas innecesarias**.

```
A = np.array(np.random.rand(N,N))
n = A.shape[0]
A, A[n//2:,n//2:]


def block_multiplication(A,B):

    n = A.shape[0]
    assert(n % 2 == 0)

    # Completar
    A11 = None
```