In [1]:

```
import numpy as np
a  = np.array([0, 1, 2, 3])
a
```

Out[1]:

```
array([0, 1, 2, 3])
```

In [2]:

```
L = range(1000)
```

In [3]:

```
%timeit [i**2 for i in L]
```

```
10000 loops, best of 3: 60.4 µs per loop
```

In [4]:

```
a = np.arange(1000)
```

In [5]:

```
%timeit a**2
```

```
100000 loops, best of 3: 1.99 µs per loop
```

In [6]:

```
np.array?
```

In [7]:

```
np.lookfor('create array')
```

```
Search results for 'create array'
---------------------------------
numpy.array
    Create an array.
numpy.memmap
    Create a memory-map to an array stored in a *binary* file on disk.
numpy.diagflat
    Create a two-dimensional array with the flattened input as a diagonal.
numpy.fromiter
    Create a new 1-dimensional array from an iterable object.
numpy.partition
    Return a partitioned copy of an array.
numpy.ma.diagflat
    Create a two-dimensional array with the flattened input as a diagonal.
numpy.ctypeslib.as_array
    Create a numpy array from a ctypes array or a ctypes POINTER.
numpy.ma.make_mask
    Create a boolean mask from an array.
numpy.ctypeslib.as_ctypes
    Create and return a ctypes object from a numpy array.  Actually
numpy.ma.mrecords.fromarrays
    Creates a mrecarray from a (flat) list of masked arrays.
numpy.lib.format.open_memmap
    Open a .npy file as a memory-mapped array.
numpy.ma.MaskedArray.__new__
    Create a new masked array from scratch.
numpy.lib.arrayterator.Arrayterator
    Buffered iterator for big arrays.
numpy.ma.mrecords.fromtextfile
    Creates a mrecarray from data stored in the file `filename`.
numpy.oldnumeric.ma.fromfunction
    apply f to s to create array as in umath.
numpy.oldnumeric.ma.masked_object
    Create array masked where exactly data equal to value
numpy.oldnumeric.ma.masked_values
    Create a masked array; mask is nomask if possible.
numpy.asarray
```

```
    Convert the input to an array.
numpy.ndarray
    ndarray(shape, dtype=float, buffer=None, offset=0,
numpy.recarray
    Construct an ndarray that allows field access using attributes.
numpy.chararray
    chararray(shape, itemsize=1, unicode=False, buffer=None, offset=0,
numpy.pad
    Pads an array.
numpy.sum
    Sum of array elements over a given axis.
numpy.asanyarray
    Convert the input to an ndarray, but pass ndarray subclasses through.
numpy.copy
    Return an array copy of the given object.
numpy.diag
    Extract a diagonal or construct a diagonal array.
numpy.load
    Load an array(s) or pickled objects from .npy, .npz, or pickled files.
numpy.sort
    Return a sorted copy of an array.
numpy.array_equiv
    Returns True if input arrays are shape consistent and all elements equal.
numpy.dtype
    Create a data type object.
numpy.choose
    Construct an array from an index array and a set of arrays to choose from.
numpy.nditer
    Efficient multi-dimensional iterator object to iterate over arrays.
numpy.swapaxes
    Interchange two axes of an array.
numpy.full_like
    Return a full array with the same shape and type as a given array.
numpy.ones_like
    Return an array of ones with the same shape and type as a given array.
numpy.empty_like
    Return a new array with the same shape and type as a given array.
numpy.zeros_like
    Return an array of zeros with the same shape and type as a given array.
numpy.asarray_chkfinite
    Convert the input to an array, checking for NaNs or Infs.
numpy.diag_indices
    Return the indices to access the main diagonal of an array.
numpy.ma.choose
    Use an index array to construct a new array from a set of choices.
numpy.chararray.tolist
    a.tolist()
numpy.matlib.rand
    Return a matrix of random values with given shape.
numpy.savez_compressed
    Save several arrays into a single file in compressed ``.npz`` format.
numpy.ma.empty_like
    Return a new array with the same shape and type as a given array.
numpy.ma.make_mask_none
    Return a boolean mask of the given shape, filled with False.
numpy.ma.mrecords.fromrecords
    Creates a MaskedRecords from a list of records.
numpy.around
    Evenly round to the given number of decimals.
numpy.source
    Print or write to a file the source code for a Numpy object.
numpy.diagonal
    Return specified diagonals.
numpy.histogram2d
    Compute the bi-dimensional histogram of two data samples.
numpy.fft.ifft
    Compute the one-dimensional inverse discrete Fourier Transform.
numpy.fft.ifftn
    Compute the N-dimensional inverse discrete Fourier Transform.
numpy.busdaycalendar
    A business day calendar object that efficiently stores information
```

In [8]:

```
np.con*?
```

In [9]:

```
import numpy as nop
a=np.array([0,1,2,3])
a
```

Out[9]:

```
array([0, 1, 2, 3])
```

In [10]:

```
a.ndim
```

Out[10]:

```
1
```

In [11]:

```
a.shape
```

Out[11]:

```
(4,)
```

In [12]:

```
len(b)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-12-f311c9fb5505> in <module>()
----> 1 len(b)

NameError: name 'b' is not defined
```

In [13]:

```
len(a)
```

Out[13]:

```
4
```

In [14]:

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

Out[14]:

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

       [[3],
        [4]]])
```

In [15]:

```
c.shape
```

Out[15]:

```
(2, 2, 1)
```

In [16]:

```
a = np.arange(10) # 0 .. n-1  (!)
a
```

Out[16]:

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

In [17]:

```
b = np.arange(1, 9, 2) # start, end (exclusive), step
b
```

Out[17]:

```
array([1, 3, 5, 7])
```

In [18]:

```
c = np.linspace(0, 1, 6)   # start, end, num-points
c
```

Out[18]:

```
array([ 0. ,  0.2,  0.4,  0.6,  0.8,  1. ])
```

In [19]:

```
d = np.linspace(0, 1, 5, endpoint=False)
d
```

Out[19]:

```
array([ 0. ,  0.2,  0.4,  0.6,  0.8])
```

In [20]:

```
a = np.ones((3, 3))  # reminder: (3, 3) is a tuple
a
```

Out[20]:

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

In [21]:

```
b = np.zeros((2, 2))
b
```

Out[21]:

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

In [22]:

```
c = np.eye(3)
c
```

Out[22]:

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

In [23]:

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

Out[23]:

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

In [24]:

```
a = np.random.rand(4)       # uniform in [0, 1]
a
```

Out[24]:

```
array([ 0.10176711,  0.97271389,  0.93421141,  0.48654429])
```

In [25]:

```
b = np.random.randn(4)      # Gaussian
b
```

Out[25]:

```
array([ 0.50870177,  0.20030178,  0.24868499,  1.29624503])
```

In [26]:

```
np.random.seed(1234)
```

In [27]:

```
a = np.array([1, 2, 3])
a.dtype
```

Out[27]:

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

In [28]:

```
b = np.array([1., 2., 3.])
b.dtype
```

Out[28]:

```
dtype('float64')
```

In [29]:

```
c = np.array([1, 2, 3], dtype=float)
c.dtype
```

Out[29]:

```
dtype('float64')
```

In [30]:

```
a = np.ones((3, 3))
a.dtype
```

Out[30]:

```
dtype('float64')
```

In [31]:

```
d = np.array([1+2j, 3+4j, 5+6*1j])
d.dtype
```

Out[31]:

```
dtype('complex128')
```

In [32]:

```
e = np.array([True, False, False, True])
e.dtype
```

Out[32]:

```
dtype('bool')
```

In [33]:

```
f = np.array(['Bonjour', 'Hello', 'Hallo',])
f.dtype     # <--- strings containing max. 7 letters
```

Out[33]:

```
dtype('S7')
```

In [34]:

```
%pylab
```

```
Using matplotlib backend: TkAgg
Populating the interactive namespace from numpy and matplotlib

WARNING: pylab import has clobbered these variables: ['e', 'f']
`%matplotlib` prevents importing * from pylab and numpy
```

In [35]:

```
import matplotlib.pyplot as plt  # the tidy way
```

In [36]:

```
plt.plot(x, y)       # line plot
plt.show()           # <-- shows the plot (not needed with pylab)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-36-cca45a0ba107> in <module>()
----> 1 plt.plot(x, y)       # line plot
      2 plt.show()           # <-- shows the plot (not needed with pylab)

NameError: name 'x' is not defined
```

In [37]:

```
x = np.linspace(0, 3, 20)
y = np.linspace(0, 9, 20)
plt.plot(x, y)       # line plot
```

Out[37]:

```
[<matplotlib.lines.Line2D at 0x7f6575e543d0>]
```

In [38]:

```
plt.plot(x, y, 'o')
```

Out[38]:

```
[<matplotlib.lines.Line2D at 0x7f6575bcfc50>]
```

In [39]:

```
image = np.random.rand(30, 30)
plt.imshow(image, cmap=plt.cm.hot)
plt.colorbar()
```

Out[39]:

```
<matplotlib.colorbar.Colorbar instance at 0x7f65754df9e0>
```

In [40]:

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

Out[40]:

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

In [41]:

```
a[0], a[2], a[-1]
```

Out[41]:

```
(0, 2, 9)
```

In [42]:

```
a[0], a[2], a[-1]
```

Out[42]:

```
(0, 2, 9)
```

In [43]:

```
a = np.diag(np.arange(3))
a
```

Out[43]:

```
array([[0, 0, 0],
       [0, 1, 0],
       [0, 0, 2]])
```

In [44]:

```
a[1, 1]
```

Out[44]:

```
1
```

In [45]:

```
a[2, 1] = 10 # third line, second column
a
```

Out[45]:

```
array([[ 0,  0,  0],
       [ 0,  1,  0],
       [ 0, 10,  2]])
```

In [46]:

```
a[1]
```

Out[46]:

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

In [47]:

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

Out[47]:

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

In [48]:

```
a[2:9:3] # [start:end:step]
```

Out[48]:

```
array([2, 5, 8])
```

In [49]:

```
a[:4]
```

Out[49]:

```
array([0, 1, 2, 3])
```

In [50]:

```
a[1:3]
```

Out[50]:

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

In [51]:

```
a[::2]
```

Out[51]:

```
array([0, 2, 4, 6, 8])
```

In [52]:

```
a[3:]
```

Out[52]:

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

In [53]:

```
a = np.arange(10)
a[5:] = 10
a
```

Out[53]:

```
array([ 0,  1,  2,  3,  4, 10, 10, 10, 10, 10])
```

In [54]:

```
b = np.arange(5)
a[5:] = b[::-1]
a
```

Out[54]:

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

In [55]:

```
np.arange(6) + np.arange(0, 51, 10)[:, np.newaxis]
```

Out[55]:

```
array([[ 0,  1,  2,  3,  4,  5],
       [10, 11, 12, 13, 14, 15],
       [20, 21, 22, 23, 24, 25],
       [30, 31, 32, 33, 34, 35],
       [40, 41, 42, 43, 44, 45],
       [50, 51, 52, 53, 54, 55]])
```

In [56]:

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

Out[56]:

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

In [57]:

```
b = a[::2]
b
```

Out[57]:

```
array([0, 2, 4, 6, 8])
```

In [58]:

```
np.may_share_memory(a, b)
```

Out[58]:

```
True
```

In [59]:

```
b[0] = 12
b
```

Out[59]:

```
array([12,  2,  4,  6,  8])
```

In [60]:

```
a   # (!)
```

Out[60]:

```
array([12,  1,  2,  3,  4,  5,  6,  7,  8,  9])
```

In [61]:

```
a = np.arange(10)
c = a[::2].copy()  # force a copy
c[0] = 12
a
```

Out[61]:

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

In [62]:

```
is_prime = np.ones((100,), dtype=bool)
```

In [63]:

```
is_prime[:2] = 0
```

In [64]:

```
N_max = int(np.sqrt(len(is_prime)))
for j in range(2, N_max):
    is_prime[2*j::j] = False
```

In [65]:

```
np.random.seed(3)
a = np.random.random_integers(0, 20, 15)
a
```

Out[65]:

```
array([10,  3,  8,  0, 19, 10, 11,  9, 10,  6,  0, 20, 12,  7, 14])
```

In [66]:

```
(a % 3 == 0)
```

Out[66]:

```
array([False,  True, False,  True, False, False, False,  True, False,
        True,  True, False,  True, False, False], dtype=bool)
```

In [67]:

```
mask = (a % 3 == 0)
extract_from_a = a[mask] # or,  a[a%3==0]
extract_from_a           # extract a sub-array with the mask
```

Out[67]:

```
array([ 3,  0,  9,  6,  0, 12])
```

In [68]:

```
a[a % 3 == 0] = -1
a
```

Out[68]:

```
array([10, -1,  8, -1, 19, 10, 11, -1, 10, -1, -1, 20, -1,  7, 14])
```

In [69]:

```
a = np.arange(0, 100, 10)
a
```

Out[69]:

```
array([ 0, 10, 20, 30, 40, 50, 60, 70, 80, 90])
```

In [70]:

```
a[[2, 3, 2, 4, 2]]  # note: [2, 3, 2, 4, 2] is a Python list
```

Out[70]:

```
array([20, 30, 20, 40, 20])
```

In [71]:

```
a[[9, 7]] = -100
a
```

Out[71]:

```
array([   0,   10,   20,   30,   40,   50,   60, -100,   80, -100])
```

In [72]:

```
a = np.arange(10)
idx = np.array([[3, 4], [9, 7]])
idx.shape
```

Out[72]:

```
(2, 2)
```

In [73]:

```
a[idx]
```

Out[73]:

```
array([[3, 4],
       [9, 7]])
```

In []:

```
```