Numpy

نامپی یک بسته مهم برای محاسبات علمی با پایتون است. این بسته شامل اجزای دیگری نیز می‌شود، اجزایی مانند : یک شیء از آرایه چند بعدی قدرتمند، توابع پیشرفته، توانایی مجتمع شد با زبان‌هایی مانند C/C++ و Fortran. و نیز ابزارهای مفیدی مانند جبر خطی، تبدیل فوریه و تولید اعداد تصادفی.

علاوه بر آنچه گفته شد، در کاربردهای علمی، نامپی می‌تواند به عنوان یک نگهدارنده برای داده‌های چند بعدی به طور بسیار کارآمدی استفاده شود. و چون در این آرایه می‌توانید جنس داده‌ها را نیز مشخص کنید بنابراین خیلی سریع و راحت با سایر پایگاه‌های داده نیز می‌توانند هماهنگ شود.

آرایه‌های نامپی:

۱- یک بسته برای محاسبات آرایه‌های چند بعدی در پایتون ۲- چون نزدیک به سخت افزار نوشته شده است، بنابراین پر سرعت و کارآمد است. ۳- برای راحتی محاسبات علمی طراحی شده است.

In [ ]:
```
!pip install --user numpy
```

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

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

In [2]:
```
type(arr)
```

Out[2]:
```
numpy.ndarray
```

یک آرایه مثلا می‌تواند به صورت زیر باشد:

مقدارهایی برای شبیه سازی گام‌های گسسته زمان است.

سیگنال صوتی ذخیره شده با یک میکروفون باشد

پیکسل‌های یک تصویر باشد.

اندازه‌گیری‌های ۳ بعدی از مختصات در یک روبش MRI

....

In [3]:
```
# calculate time spends for power 2 of numbers from 0 to 999 by python:
L = range(1000)
%timeit [i**2 for i in L]
```

```
1000 loops, best of 5: 484 µs per loop
```

In [4]:
```
np.arange(10)
```

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

In [5]:
```
# calculate time spends for power 2 of numbers from 0 to 999 by numpy:
a = np.arange(1000)
%timeit a**2
```

```
The slowest run took 13.71 times longer than the fastest. This could mean tha
t an intermediate result is being cached.
100000 loops, best of 5: 3.82 µs per loop
```

گرفتن راهنما:

In [6]:
```
np.array?
```

In [7]:
```
np.arange?
```

In [8]:
```
help(np.array)
```

```
Help on built-in function array in module numpy:

array(...)
    array(object, dtype=None, copy=True, order='K', subok=False, ndmin=0)

    Create an array.

    Parameters
    ----------
    object : array_like
        An array, any object exposing the array interface, an object whose
        __array__ method returns an array, or any (nested) sequence.
    dtype : data-type, optional
        The desired data-type for the array.  If not given, then the type wil
l
        be determined as the minimum type required to hold the objects in the
        sequence.  This argument can only be used to 'upcast' the array.  For
        downcasting, use the .astype(t) method.
    copy : bool, optional
        If true (default), then the object is copied.  Otherwise, a copy will
        only be made if __array__ returns a copy, if obj is a nested sequenc
e,
        or if a copy is needed to satisfy any of the other requirements
        (`dtype`, `order`, etc.).
    order : {'K', 'A', 'C', 'F'}, optional
        Specify the memory layout of the array. If object is not an array, th
e
        newly created array will be in C order (row major) unless 'F' is
        specified, in which case it will be in Fortran order (column major).
        If object is an array the following holds.

        ===== ========= ===================================================
        order  no copy                     copy=True
        ===== ========= ===================================================
        'K'   unchanged F & C order preserved, otherwise most similar order
        'A'   unchanged F order if input is F and not C, otherwise C order
        'C'   C order   C order
        'F'   F order   F order
        ===== ========= ===================================================

        When ``copy=False`` and a copy is made for other reasons, the result
is
        the same as if ``copy=True``, with some exceptions for `A`, see the
        Notes section. The default order is 'K'.
    subok : bool, optional
        If True, then sub-classes will be passed-through, otherwise
        the returned array will be forced to be a base-class array (default).
    ndmin : int, optional
        Specifies the minimum number of dimensions that the resulting
        array should have.  Ones will be pre-pended to the shape as
        needed to meet this requirement.

    Returns
    -------
    out : ndarray
        An array object satisfying the specified requirements.

    See Also
    --------
    empty_like : Return an empty array with shape and type of input.
    ones_like : Return an array of ones with shape and type of input.
    zeros_like : Return an array of zeros with shape and type of input.
    full_like : Return a new array with shape of input filled with value.
    empty : Return a new uninitialized array.
    ones : Return a new array setting values to one.
    zeros : Return a new array setting values to zero.
    full : Return a new array of given shape filled with value.
```

http://docs.scipy.org/

In [9]:
```
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.ctypeslib.as_array
    Create a numpy array from a ctypes array or POINTER.
numpy.ma.diagflat
    Create a two-dimensional array with the flattened input as a diagonal.
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.ma.mvoid.__new__
    Create a new masked array from scratch.
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.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.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 arrays 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.ufunc
    Functions that operate element by element on whole arrays.
numpy.choose
    Construct an array from an index array and a set of arrays to choose fro
m.
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
```

In [10]:
```
np.con*?
```

ایجاد یک آرایه

In [11]:
```
# 1-D manual:
a = np.array([0, 1, 2, 3])
print(a)
```

```
[0 1 2 3]
```

In [12]:
```
a.ndim
```

Out[12]:
```
1
```

In [13]:
```
a.shape
```

Out[13]:
```
(4,)
```

In [14]:
```
len(a)
```

Out[14]:
```
4
```

In [16]:
```
# 2-D , 3-D and ...
b = np.array([[0, 1, 2], [3, 4, 5]])    # 2 x 3 array

c = np.array([[0, 1, 2],
              [3, 4, 5]])
b
```

Out[16]:
```
array([[0, 1, 2],
       [3, 4, 5]])
```

In [17]:
```
b.ndim
```

Out[17]:
```
2
```

In [18]:
```
b.shape
```

Out[18]:
```
(2, 3)
```

In [19]:
```
len(b)
```

Out[19]:
```
2
```

In [20]:
```
c = np.array([[[1], [2], [5]], [[3], [4], [6]]])
```

In [21]:
```
c.shape
```

Out[21]:
```
(2, 3, 1)
```

In [22]:
```
len(c)
```

Out[22]:
```
2
```

In [23]:
```
a = np.arange(10) # 0, .... n - 1
a
```

Out[23]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [24]:
```
start = 1; stop = 9; step = 2
b = np.arange(start, stop+1, step)
b
```

Out[24]:
```
array([1, 3, 5, 7, 9])
```

In [25]:
```
start = 0; stop = 1; num_points = 6
c = np.linspace(start, stop, num_points)
c
```

Out[25]:
```
array([0. , 0.2, 0.4, 0.6, 0.8, 1. ])
```

In [26]:
```
d = np.linspace(0, 1, num_points-1, endpoint=False)
d
```

Out[26]:
```
array([0. , 0.2, 0.4, 0.6, 0.8])
```

آرایه‌های مشهور:

In [27]:
```
a = np.ones((3,3))
a
```

Out[27]:
```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

In [30]:
```
b = np.zeros((2,))
b
```

Out[30]:
```
array([0., 0.])
```

In [31]:
```
c = np.eye(3)
c
```

Out[31]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [32]:
```
d = np.diag(c)
d
```

Out[32]:
```
array([1., 1., 1.])
```

In [33]:
```
d = np.diag(np.array([1,2,3,4]))
d
```

Out[33]:
```
array([[1, 0, 0, 0],
       [0, 2, 0, 0],
       [0, 0, 3, 0],
       [0, 0, 0, 4]])
```

تولید اعداد تصادفی

In [ ]:
```
```

In [38]:
```
a = np.random.rand(4,1)
print(a)
b = np.random.randn(4)*2 + 1
print(b)
np.random.seed(1234)
```

```
[[0.19151945]
 [0.62210877]
 [0.43772774]
 [0.78535858]]
[-0.44117747  2.77432588  2.71917683 -0.27304701]
```

جنس داده‌ها در نامپی

In [39]:
```
a = np.array([1,2,3])
print(a)
print(a.dtype)
```

```
[1 2 3]
int64
```

In [40]:
```
b = np.array([1., 2., 3., 4.])
print(b)
print(b.dtype)
```

```
[1. 2. 3. 4.]
float64
```

In [42]:
```
a[0] = 1.5
a
```

Out[42]:
```
array([1, 2, 3])
```

این که جنس داده‌ها در نامپی می‌تواند متفاوت باشد، در مدیریت فضای حافظه کمک بسیار زیادی می‌کند.

In [43]:
```
c = np.array([1, 2, 3], dtype=float)
print(c)
print(c.dtype)
```

```
[1. 2. 3.]
float64
```

In [44]:
```
a = np.ones((3,3))
print(a.dtype)
```

```
float64
```

In [45]:
```
# complex type:
d = np.array([1+2j, 3+4j])
d.dtype
```

Out[45]:
```
dtype('complex128')
```

In [46]:
```
e = np.array([True, False])
e.dtype
```

Out[46]:
```
dtype('bool')
```

In [48]:
```
f = np.array(['Bonjour', 'Hello', 'Hallo', '12345678'])
f.dtype
```

Out[48]:
```
dtype('<U8')
```

In [49]:
```
np.array(['یک مثال'])
```

Out[49]:
```
array(['یک مثال'], dtype='<U7')
```

نمایش:

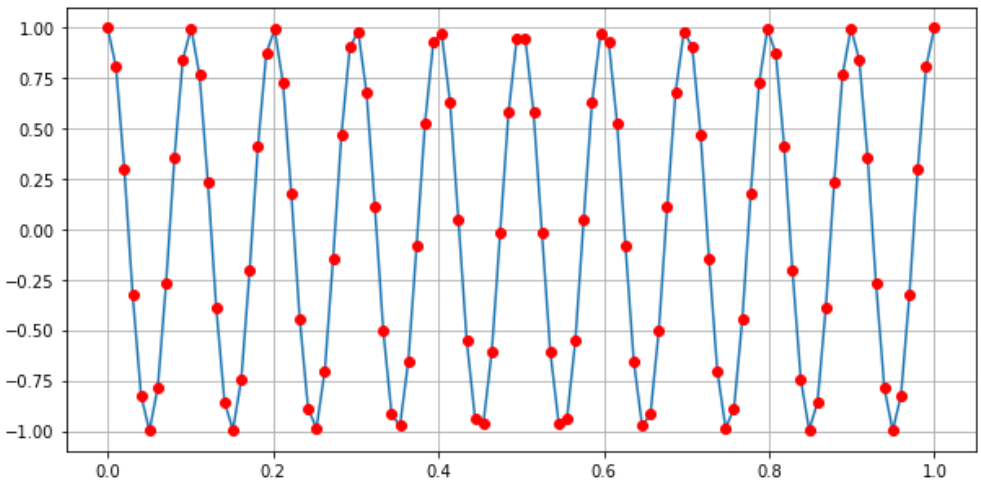
داده‌های نامپی در حالت کلی می‌توانند یک سیگنال، یک تصویر و یا هر داده دیگری باشند، که برای نمایش آن‌ها می‌توان از ابزارهای مختلفی استفاده کرد. یکی از این ابزارها بسته matplotlib است.

In [ ]:
```
!pip install --user matplotlib
```

In [50]:
```
from matplotlib import pyplot as plt
```

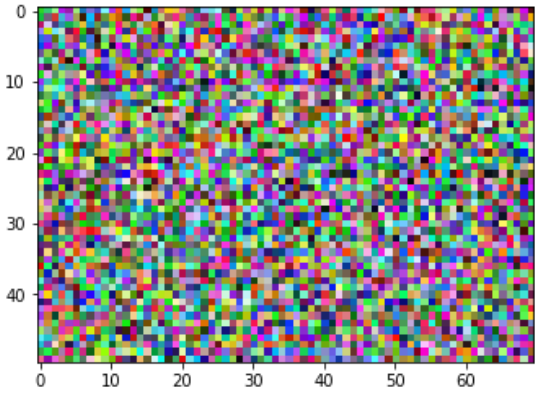
In [51]:
```
t = np.linspace(0, 1, 100)
sig = np.cos(2 * np.pi * 10 * t)

plt.figure(figsize=(10,5))
plt.plot(t, sig)
plt.plot(t, sig, 'ro')
plt.grid()
```



In [52]:
```
a = np.random.randint(0, 256, size=(50, 70, 3))
plt.imshow(a)
```

Out[52]: <matplotlib.image.AxesImage at 0x7f27e42657b8>



اندیس گذاری و برش:

In [53]:
```
a = np.arange(10)
print(a)
a[0], a[2], a[-1]
```

```
[0 1 2 3 4 5 6 7 8 9]
```

Out[53]: (0, 2, 9)

In [54]:
```
a[::-1]
```

Out[54]: array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])

In [55]:
```
a = np.diag(np.arange(3))
a
```

Out[55]:
```
array([[0, 0, 0],
       [0, 1, 0],
       [0, 0, 2]])
```

In [56]:
```
a[1, 1]
```

Out[56]: 1

In [57]:
```
a[1][1]
```

Out[57]: 1

In [61]:
```
a[1]
```

Out[61]: array([0, 1, 10])

In [62]:
```
a[:,1]
```

Out[62]: array([0, 1, 0])

In [60]:
```
a[:][1][2] = 10
a
```

Out[60]:
```
array([[ 0,  0,  0],
       [ 0,  1, 10],
       [ 0,  0,  2]])
```

In [63]:
```
a = np.arange(10)
a[2:9:3]
```

Out[63]:
```
array([2, 5, 8])
```

In [64]:
```
a[::2]
```

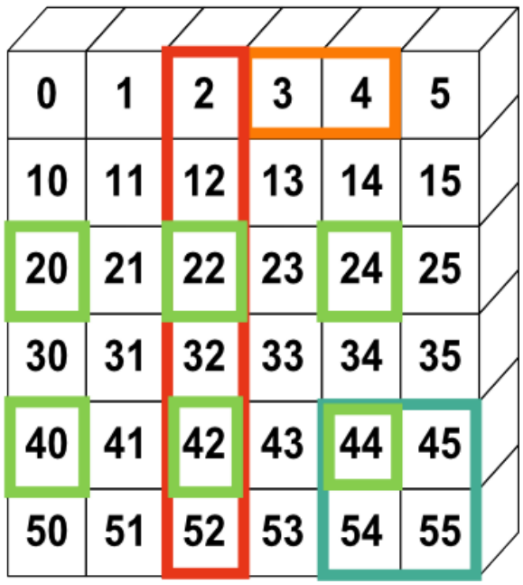
Out[64]:
```
array([0, 2, 4, 6, 8])
```

```
>>> 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]])
```



In [65]:
```
# check it yourself
a = np.arange(6) + np.arange(0, 51, 10)[:, np.newaxis]
a
```

Out[65]:
```
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]])
```

کپی کردن:

```
In [66]: a = np.arange(10)
         b = a
         b[5] = 50
         print('b = ', b)
         print('a = ', a)

         b =  [ 0  1  2  3  4 50  6  7  8  9]
         a =  [ 0  1  2  3  4 50  6  7  8  9]

In [67]: np.may_share_memory(a, b)

Out[67]: True

In [68]: a = np.arange(10)
         b = a.copy()
         b[5] = 50
         print('b = ', b)
         print('a = ', a)

         b =  [ 0  1  2  3  4 50  6  7  8  9]
         a =  [0 1 2 3 4 5 6 7 8 9]

In [69]: np.may_share_memory(a, b)

Out[69]: False
```

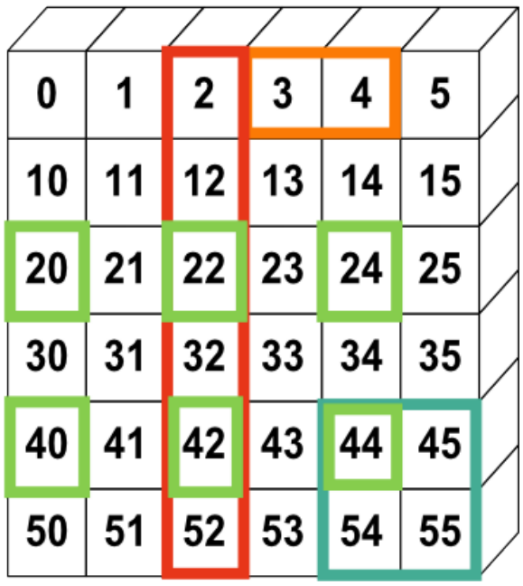
یک تمرین: بیایید با دستور العمل زیر یک آرایه برای مشخص کردن اعداد اول زیر ۱۰۰ درست کنیم. یعنی هر جای این آرایه ۱ بود، عدد متناظر با آن اول است. و اگر در این آرایه صفر باشد، عدد متناظر با آن اول نیست. شکل زیر را نگاه کنید:

```
>>> 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]])
```



ابتدا یک آرایه بولی به طول ۱۰۰ درست می‌کنیم. و در شروع همه مقدارهای آن را «درست» فرض می‌کنیم:

In [70]:
```
is_prime = np.ones((100,), dtype=bool)
is_prime
```

Out[70]:
```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True])
```

دو عدد صفر و یک را می‌دانیم اول نیستند:

In [71]:
```
is_prime[:2] = 0 # also False
is_prime
```

Out[71]:
```
array([False, False,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True])
```

برای هر عدد صحیح j که از ۲ شروع کنیم، مضارب ۲ دیگر اول نیستند!

In [72]:
```
is_prime[2*2::2] = False
is_prime
```

Out[72]:
```
array([False, False,  True,  True, False,  True, False,  True, False,
        True, False,  True, False,  True, False,  True, False,  True,
       False,  True, False,  True, False,  True, False,  True, False,
        True, False,  True, False,  True, False,  True, False,  True,
       False,  True, False,  True, False,  True, False,  True, False,
        True, False,  True, False,  True, False,  True, False,  True,
       False,  True, False,  True, False,  True, False,  True, False,
        True, False,  True, False,  True, False,  True, False,  True,
       False,  True, False,  True, False,  True, False,  True, False,
        True, False,  True, False,  True, False,  True, False,  True,
       False,  True, False,  True, False,  True, False,  True, False,
        True])
```

بگذارید همین کار را برای مضارب اعداد از ۳ تا جذر ۱۰۰ را ادامه دهیم:

In [73]:
```
N_max = int(np.sqrt(len(is_prime) +1))
for j in range(3, N_max + 1):
    is_prime[2*j::j] = False
```

In [74]:
```
is_prime[95]
```

Out[74]:
```
False
```

In [75]:
```
is_prime[79]
```

Out[75]:
```
True
```

In [76]:
```
is_prime
```

Out[76]:
```
array([False, False,  True,  True, False,  True, False,  True, False,
       False, False,  True, False,  True, False, False, False,  True,
       False,  True, False, False, False,  True, False, False, False,
       False, False,  True, False,  True, False, False, False, False,
       False,  True, False, False, False,  True, False,  True, False,
       False, False,  True, False, False, False, False, False,  True,
       False, False, False, False, False,  True, False,  True, False,
       False, False, False, False,  True, False, False, False,  True,
       False,  True, False, False, False, False, False,  True, False,
       False, False,  True, False, False, False, False, False,  True,
       False, False, False, False, False, False, False,  True, False,
       False])
```

In [77]:
```
np.nonzero(is_prime)
```

Out[77]:
```
(array([ 2,  3,  5,  7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59,
        61, 67, 71, 73, 79, 83, 89, 97]),)
```

اندیس گذاری با لیست:

In [78]:
```
np.random.seed(3)
a = np.random.randint(0, 21, 15)
print('a =', a)
mask = (a % 3 == 0)
print('mask =', mask)
extract_from_a = a[mask]
print('extract_from_a =', extract_from_a)
```

```
a = [10  3  8  0 19 10 11  9 10  6  0 20 12  7 14]
mask = [False  True False  True False False False  True False  True  True Fal
se
  True False False]
extract_from_a = [ 3  0  9  6  0 12]
```

In [79]:
```
a[a % 3 == 0] = -1
a
```

Out[79]:
```
array([10, -1,  8, -1, 19, 10, 11, -1, 10, -1, -1, 20, -1,  7, 14])
```

In [80]:
```
a = np.arange(0, 100, 10)
print(a)
a[[2, 3, 2, 4 ,2]]  # note that [2, 3, 2, 4, 2] is a python list
```

```
[ 0 10 20 30 40 50 60 70 80 90]
```

Out[80]:
```
array([20, 30, 20, 40, 20])
```

In [81]:
```
a[[9, 7]] = -100
a
```

Out[81]:
```
array([   0,   10,   20,   30,   40,   50,   60, -100,   80, -100])
```

حواستان باشد که وقتی یک آرایه را با یک آرایه اعداد صحیح نامپی اندیس گذاری می‌کنید، خروجی به ابعاد همان اندیس‌ها در می‌آید.
برای این که متوجه این صحبت شوید، مثال زیر را ببینید:

In [82]:
```
a = np.arange(10)
print('a.shape =', a.shape)
idx = np.array([[3, 4], [9, 7]])
print('idx.shape =', idx.shape)
print('a[idx].shape = ', a[idx].shape)
a[idx]
```

```
a.shape = (10,)
idx.shape = (2, 2)
a[idx].shape =  (2, 2)
```

Out[82]:
```
array([[3, 4],
       [9, 7]])
```

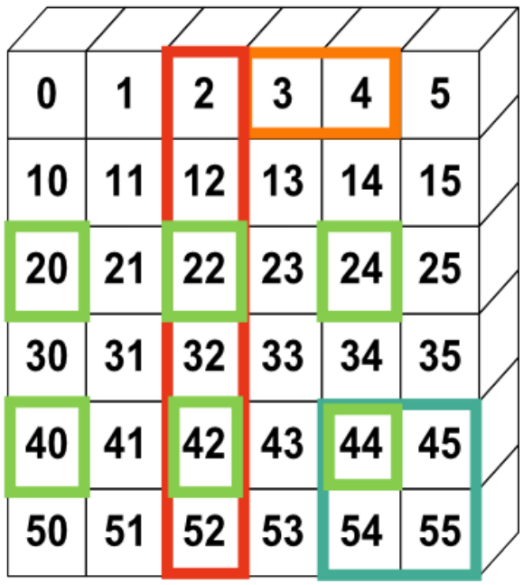
تصویر زیر چند حالت مختلف این نوع اندیس گذاری را نشان می‌دهد:

```
>>> 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]])
```



In [83]:
```
# check it yourself
a = np.arange(6) + np.arange(0, 51, 10)[:, np.newaxis]
a
```

Out[83]:
```
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]])
```

عملیات عددی روی آرایه‌ها:

عملیات عنصر به عنصر:

In [84]:
```
a = np.array([1, 2, 3, 4])
print(a + 1)
print(2 ** a)
```

```
[2 3 4 5]
[ 2  4  8 16]
```

In [85]:
```
# All arithmatic operator operates elementwise:
b = np.ones(4) + 1
print(a - b)
print(a * b)
j = np.arange(5)
print( 2**(j+1) - j)
```

```
[-1.  0.  1.  2.]
[2. 4. 6. 8.]
[ 2  3  6 13 28]
```

In [86]:
```
c = np.ones((3, 3))
# this is obviousely is not matrix production:
c * c
```

Out[86]:
```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

In [87]:
```
# But this is a matrix multiplication you know:
c.dot(c)
```

Out[87]:
```
array([[3., 3., 3.],
       [3., 3., 3.],
       [3., 3., 3.]])
```

مقایسه‌ها:

In [88]:
```
a = np.array([1,2,3,4])
b = np.array([4,2,2,4])
c = np.array([1,2,3,4])
print(a == b)
print(a > b)
print(np.array_equal(a, b))
print(np.array_equal(a, c))
```

```
[False  True False  True]
[False False  True False]
False
True
```

عملگرهای منطقی:

In [89]:
```
a = np.array([1, 1, 0, 0], dtype=bool)
b = np.array([1, 0, 1, 0], dtype=bool)
np.logical_or(a, b)
```

Out[89]:
```
array([ True,  True,  True, False])
```

توابع ریاضی:

In [90]:
```
t = np.arange(5)
print(np.sin(t))
print(np.log(t))
print(np.exp(t))
```

```
[ 0.          0.84147098  0.90929743  0.14112001 -0.7568025 ]
[      -inf 0.          0.69314718 1.09861229 1.38629436]
[ 1.          2.71828183  7.3890561  20.08553692 54.59815003]
```

عدم برابری شکل‌ها و ابعاد:

In [91]:
```
a = np.arange(4)
print('a.shape =', a.shape)
a + np.array([1, 2])
```

```
a.shape = (4,)

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-91-2423595a37f6> in <module>()
      1 a = np.arange(4)
      2 print('a.shape =', a.shape)
----> 3 a + np.array([1, 2])

ValueError: operands could not be broadcast together with shapes (4,) (2,)
```

برخی متدها:

In [92]:
```
x = np.array([1, 2, 3, 4])
print(np.sum(x))
x.sum()
```

```
10
```

Out[92]:
```
10
```

In [95]:
```
x = np.array([[1, 1],
              [2, 2]])
print('sum over columns (1st dim):', x.sum(axis=0))
print('sum over rows (2nd dim):', x.sum(axis=1))
print('sum all elements :', x.sum())
```

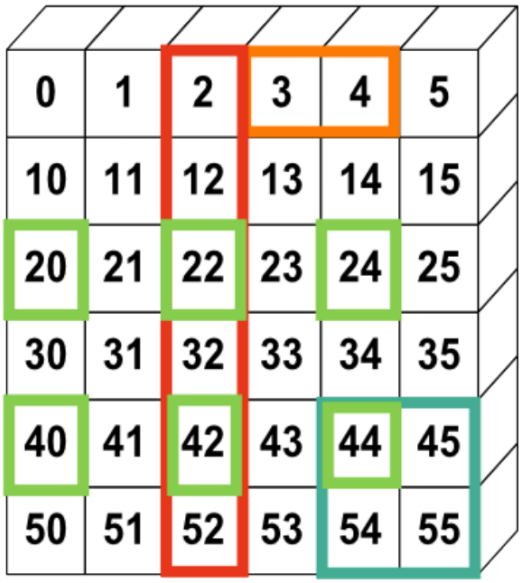
```
sum over columns (1st dim): [3 3]
sum over rows (2nd dim): [2 4]
sum all elements : 6
```

```
>>> 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]])
```



In [99]:
```
x = np.array([-1, 3, 2])
print('x.min() :', x.min())
print('x.argmx() :', x.argmax())
x.T
```

```
x.min() : -1
x.argmx() : 1
```

Out[99]:
```
array([-1,  3,  2])
```

In [97]:
```
x = np.array([[1,2,3,4]])
x
```

Out[97]:
```
array([[1, 2, 3, 4]])
```

In [98]:
```
x.T
```

Out[98]:
```
array([[1],
       [2],
       [3],
       [4]])
```

In [ ]:
```

```

In [100]:
```
np.all([True, True, True])
```

Out[100]:
```
True
```

In [101]:
```
np.any([True, False, False])
```

Out[101]:
```
True
```

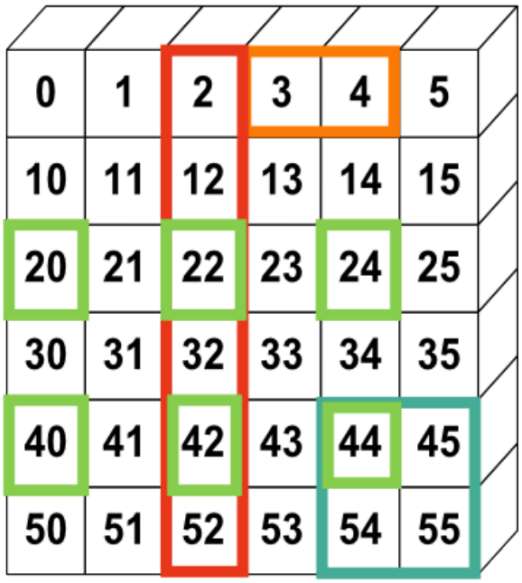
هنگام جمع کردن ماتریس و بردار به این نکته توجه کنید:

```
>>> 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]])
```



In [102]:
```
a = np.tile(np.array([0, 10, 20, 30]), (3, 1)).T
a
```

Out[102]:
```
array([[ 0,  0,  0],
       [10, 10, 10],
       [20, 20, 20],
       [30, 30, 30]])
```

In [110]:
```
b = np.array([[0, 1, 2]])
b
```

Out[110]:
```
array([[0, 1, 2]])
```

In [111]:
```
a + b
```

Out[111]:
```
array([[ 0,  1,  2],
       [10, 11, 12],
       [20, 21, 22],
       [30, 31, 32]])
```

In [ ]:

In [ ]:

برای مقایسه بسته نامپی با متلب می‌توانید توضیحات اضافی زیر را که از سایت راهنمای scipy گرفته شده‌اند را ببنید.

```
>>> 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]])
```