بسم الله الرحمن الرحيم

High Performance Multiprocessor Systems

Lecture 4: Python

part2

NumPy

Outline

➢**NumPy: creating and manipulating numerical data**

1. The Numpy array object
   1. What are Numpy and Numpy arrays?
   2. Creating arrays
   3. Basic data types
   4. Basic visualization
   5. Indexing and slicing
   6. Copies and views
   7. Fancy indexing
2. Numerical operations on arrays
   1. Elementwise operations
   2. Basic reductions
   3. Broadcasting
   4. Array shape manipulation
   5. Sorting data

NumPy

Outline

➢**NumPy: creating and manipulating numerical data**

3. More elaborate arrays
   1. More data types
   2. Structured data types
4. Advanced operations
   1. Polynomials
   2. Loading data files

NumPy

https://numpy.org/

Getting started with Python for science

1. The Numpy array object
   1. What are Numpy and Numpy arrays?

Python objects

- high-level number objects: integers, floating point
- containers: lists, dictionaries,…

Numpy provides

- extension package to Python for multi-dimensional arrays
- closer to hardware (efficiency)
- designed for scientific computation (convenience)
- ⋮

NumPy

Getting started with Python for science

1. The Numpy array object
   1. What are Numpy and Numpy arrays?

Why it is useful: Memory-efficient container that provides fast numerical operations.

```
In [205]: import numpy as np
```

%timeit is an ipython magic function

```
In [146]: L = range(1000)

In [147]: %timeit [i**2 for i in L]
1000 loops, best of 3: 382 µs per loop

In [148]: a = np.arange(1000)

In [149]: %timeit a**2
The slowest run took 11.65 times longer than the fastest. This could mean that an intermediate result
is being cached.
100000 loops, best of 3: 2 µs per loop
```

https://ipython.org/ipython-doc/dev/interactive/magics.html#magic-timeit

NumPy

Getting started with Python for science

1. The Numpy array object
   1. What are Numpy and Numpy arrays?

list

```
In [157]: l1 = [i**2 for i in L]

In [158]: type(l1)
Out[158]: list

In [159]: len(l1)
Out[159]: 1000

In [160]: L[3]
Out[160]: 3

In [161]: l1[3]
Out[161]: 9
```

array

```
In [172]: a = np.arange(1000)

In [173]: type(a)
Out[173]: numpy.ndarray

In [174]: len(a)
Out[174]: 1000

In [175]: a1 = a**2

In [176]: type(a1)
Out[176]: numpy.ndarray

In [177]: a1[3]
Out[177]: 9

In [178]: a[3]
Out[178]: 3
```

Convert array ↔ list

```
In [179]: a2 = np.array(l1)

In [180]: type(a2)
Out[180]: numpy.ndarray

In [181]: len(a2)
Out[181]: 1000

In [182]: l2 = list(a)

In [183]: type(l2)
Out[183]: list

In [184]: len(l2)
Out[184]: 1000
```

NumPy

Getting started with Python for science

1. The Numpy array object
   1. What are Numpy and Numpy arrays?

Time

1

```
In [201]: from time import time
     ...: sec1 = time()
     ...: L = range(1000)
     ...: l1 = [i**2 for i in L]
     ...: sec2 = time()
     ...: print('time_start: %f'%sec1)
     ...: print('time_end: %f'%sec2)
     ...: print('delta_time: %f'%(sec2-sec1))
time_start: 1632145082.514125
time_end: 1632145082.515127
delta_time: 0.001002
```

2

```
In [203]: import time as tm
     ...: from time import time
     ...: sec1 = time()
     ...: L = range(1000)
     ...: l1 = [i**2 for i in L]
     ...: tm.sleep(2)
     ...: sec2 = time()
     ...: print('time_start: %f'%sec1)
     ...: print('time_end: %f'%sec2)
     ...: print('delta_time: %f'%(sec2-sec1))
time_start: 1632145134.013052
time_end: 1632145136.014373
delta_time: 2.001321
```

3

```
In [204]: import time as tm
     ...: from time import time
     ...: sec1 = time()
     ...: a = np.arange(1000)
     ...: a1 = a**2
     ...: sec2 = time()
     ...: print('time_start: %f'%sec1)
     ...: print('time_end: %f'%sec2)
     ...: print('delta_time: %f'%(sec2-sec1))
time_start: 1632145463.830703
time_end: 1632145463.830703
delta_time: 0.000000
```

NumPy

Getting started with Python for science

1. The Numpy array object
   2. Creating arrays

Manual construction of arrays

1-D:

```
In [1]: import numpy as np

In [2]: a = np.array([0, 1, 2, 3])

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

In [4]: type(a)
Out[4]: numpy.ndarray

In [5]: len(a)
Out[5]: 4

In [6]: a.ndim
Out[6]: 1

In [7]: a.shape
Out[7]: (4,)
```

2-D, 3-D, ...:

```
In [13]: b = np.array([[0, 1, 2], [3, 4, 5]]) # 2 x 3 array

In [14]: b
Out[14]:
array([[0, 1, 2],
       [3, 4, 5]])

In [15]: len(b) # returns the size of the first dimension
Out[15]: 2

In [16]: b.ndim
Out[16]: 2

In [17]: b.shape
Out[17]: (2, 3)
```

NumPy

Getting started with Python for science

1. The Numpy array object
   2. Creating arrays

Manual construction of arrays

2-D, 3-D, ...:

```
In [29]: c = np.array([[[1,2,9], [2,0,3], [6,1,0], [3,1,9]],\
    ...:  [[3,5,0], [4,7,8], [4,9,2], [2,1,0]]])

In [30]: c
Out[30]:
array([[[1, 2, 9],
        [2, 0, 3],
        [6, 1, 0],
        [3, 1, 9]],

       [[3, 5, 0],
        [4, 7, 8],
        [4, 9, 2],
        [2, 1, 0]]])

In [31]: c.ndim
Out[31]: 3

In [32]: c.shape
Out[32]: (2, 4, 3)
```

NumPy

Getting started with Python for science

1. The Numpy array object
    2. Creating arrays

Functions for creating arrays

Evenly spaced:

```
In [33]: a = np.arange(10) # 0 .. n-1 (!)

In [34]: a
Out[34]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [35]: b = np.arange(1, 9, 2) # start, end (exclusive), step

In [36]: b
Out[36]: array([1, 3, 5, 7])
```

NumPy

Getting started with Python for science

1. The Numpy array object
   2. Creating arrays

Functions for creating arrays

By number of points:

```
In [37]: c = np.linspace(0, 1, 6) # start, end, num-points

In [38]: c
Out[38]: array([ 0. ,  0.2,  0.4,  0.6,  0.8,  1. ])

In [39]: d = np.linspace(0, 1, 5, endpoint=False)

In [40]: d
Out[40]: array([ 0. ,  0.2,  0.4,  0.6,  0.8])

In [41]: d = np.linspace(0, 1, 5, endpoint=True)

In [42]: d
Out[42]: array([ 0.  ,  0.25,  0.5 ,  0.75,  1.  ])
```

NumPy

Getting started with Python for science

1. The Numpy array object
   2. Creating arrays

Functions for creating arrays

Common arrays:

1

```
In [59]: a = np.ones((3,3))

In [60]: a
Out[60]:
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])

In [61]: type(a[2][0])
Out[61]: numpy.float64

In [62]: type(a[0])
Out[62]: numpy.ndarray

In [63]: type(a)
Out[63]: numpy.ndarray
```

2

```
In [64]: b = np.zeros((2,2))

In [65]: b
Out[65]:
array([[ 0.,  0.],
       [ 0.,  0.]])

In [66]: c = np.eye(3)

In [67]: c
Out[67]:
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

3

```
In [60]: d = np.diag(np.array([1, 2, 3, 4]))

In [61]: d
Out[61]:
array([[1, 0, 0, 0],
       [0, 2, 0, 0],
       [0, 0, 3, 0],
       [0, 0, 0, 4]])
```

NumPy

Getting started with Python for science

1. The Numpy array object
    2. Creating arrays

Functions for creating arrays

np.random:

```
In [62]: a = np.random.rand(4) # uniform in [0, 1]

In [63]: a
Out[63]: array([ 0.08429029,  0.76361858,  0.29844509,  0.90005806])

In [64]: b = np.random.randn(4) # Gaussian

In [65]: b
Out[65]: array([ 0.20160097,  0.58861177,  0.81866177,  0.09365644])
```

```
In [49]: a = np.random.rand(3,3)

In [50]: a
Out[50]:
array([[ 0.84545264,  0.79346299,  0.57642386],
       [ 0.46287648,  0.17773592,  0.56705438],
       [ 0.99553424,  0.17723273,  0.15079199]])

In [51]: a.T
Out[51]:
array([[ 0.84545264,  0.46287648,  0.99553424],
       [ 0.79346299,  0.17773592,  0.17723273],
       [ 0.57642386,  0.56705438,  0.15079199]])
```

NumPy

Getting started with Python for science

1. The Numpy array object
   3. Basic data types

1

```
In [88]: a = np.array([1, 2, 3])

In [89]: a.dtype
Out[89]: dtype('int32')

In [90]: b = np.array([1., 2., 3.])

In [91]: b.dtype
Out[91]: dtype('float64')

In [92]: c = np.array([1, 2, 3], dtype=float)

In [93]: c.dtype
Out[93]: dtype('float64')
```

2

The default data type is floating point:

```
In [94]: a = np.ones((3, 3))

In [95]: a.dtype
Out[95]: dtype('float64')
```

Complex

```
In [96]: d = np.array([1+2j, 3+4j, 5+6*1j])

In [97]: d.dtype
Out[97]: dtype('complex128')
```

Bool

```
In [98]: e = np.array([True, False, False, True])

In [99]: e.dtype
Out[99]: dtype('bool')
```

3

Strings

U: Unicode string

```
In [101]: f = np.array(['Hello world', 'Hello', 'Hallo',])

In [102]: f.dtype # <--- strings containing max. 11 letters
Out[102]: dtype('<U11')
```

NumPy

Getting started with Python for science

1. The Numpy array object
   3. Basic data types

Much more:
- **int32**
- **int64**
- **uint32**
- **uint64**

```
np.array([1,2,3,4], dtype= np.float64)

np.array([1,2,3,4], dtype= np.int64)

np.array([1,2,3,4], dtype= np.uint64)
```

```
In [23]: np.ui
np.uint16
np.uint32
np.uint64
np.uint8
np.uintc
np.uintp
np.ulonglong
np.unicode
```

NumPy

Getting started with Python for science

1. The Numpy array object
   3. Basic data types

```
In [105]: c = np.array([1, 2, 3], dtype='U')

In [106]: c
Out[106]:
array(['1', '2', '3'],
      dtype='<U1')

In [107]: c = np.array([1, 2, 3], dtype='?')

In [108]: c
Out[108]: array([ True,  True,  True], dtype=bool)
```

NumPy

Getting started with Python for science

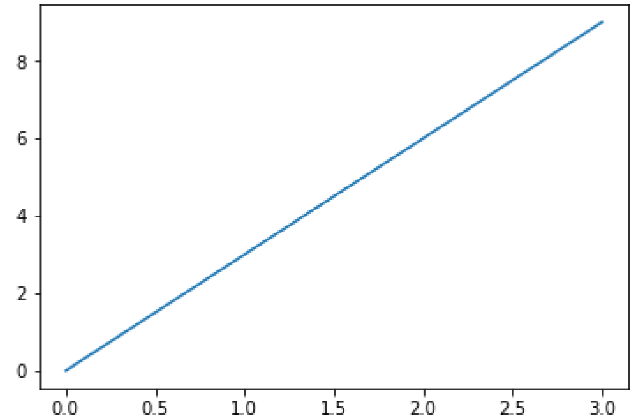
1. The Numpy array object
   4. Basic visualization

***Matplotlib* is a 2D plotting package.**

```
import matplotlib.pyplot as plt
```

- **1D plotting:**

```
In [128]: x = np.linspace(0, 3, 20)

In [129]: y = np.linspace(0, 9, 20)

In [130]: plt.plot(x, y) # line plot
Out[130]: [<matplotlib.lines.Line2D at 0xc06620fd30>]
```



NumPy

Getting started with Python for science

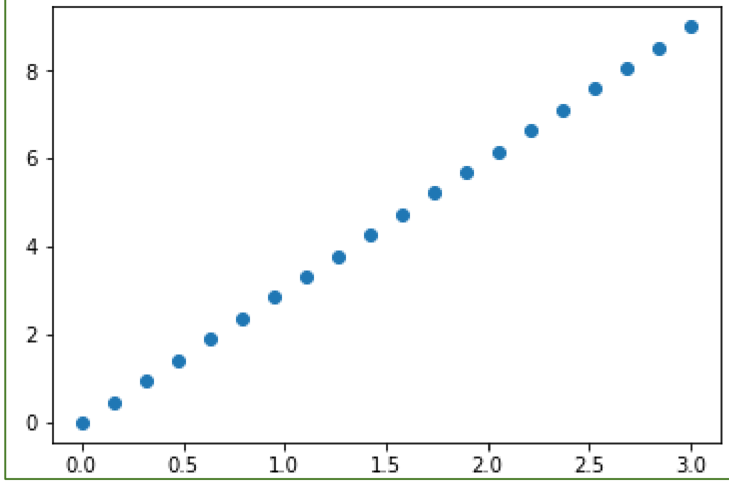
1. The Numpy array object
    4. Basic visualization

***Matplotlib* is a 2D plotting package.**

```
import matplotlib.pyplot as plt
```

- **1D plotting:**

```
In [131]: plt.plot(x, y, 'o') # dot plot
Out[131]: [<matplotlib.lines.Line2D at 0xc0662aed68>]
```



NumPy

Getting started with Python for science

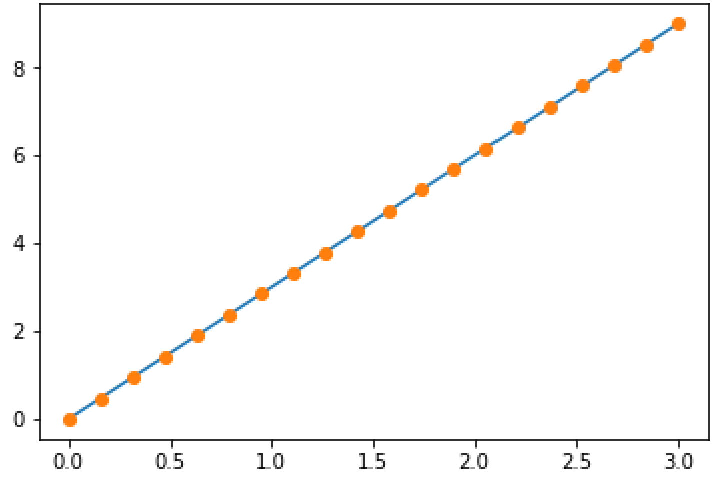
1. The Numpy array object
    4. Basic visualization

***Matplotlib* is a 2D plotting package.**

```
import matplotlib.pyplot as plt
```

- **1D plotting:**

```
In [132]: plt.plot(x, y);plt.plot(x, y, 'o') # dot plot
Out[132]: [<matplotlib.lines.Line2D at 0xc0662bc240>]
```



NumPy

Getting started with Python for science

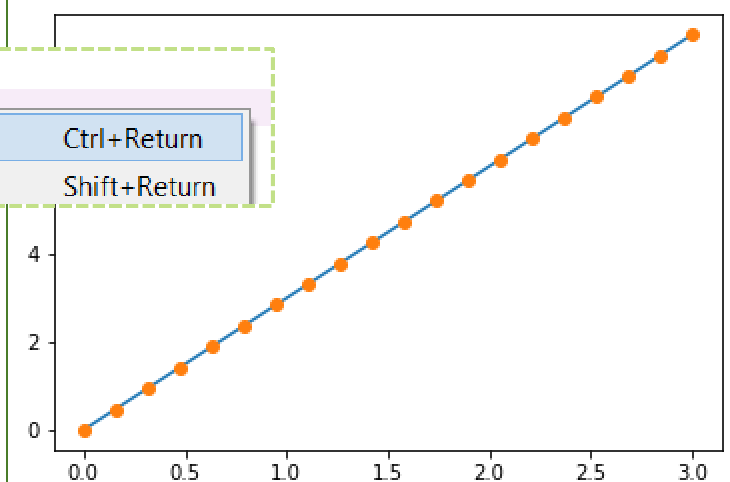
1. The Numpy array object
    4. Basic visualization

***Matplotlib* is a 2D plotting package.**

```
import matplotlib.pyplot as plt
```

- **1D plotting:**

```
In [133]: plt.plot(x, y) # dot plot
     ...: plt.plot(x, y, 'o') # dot plot
Out[133]: [<matplotlib.lines.Line2D at 0xc066332470>]
```




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

Run cell Ctrl+Return
Run cell and advance Shift+Return

NumPy

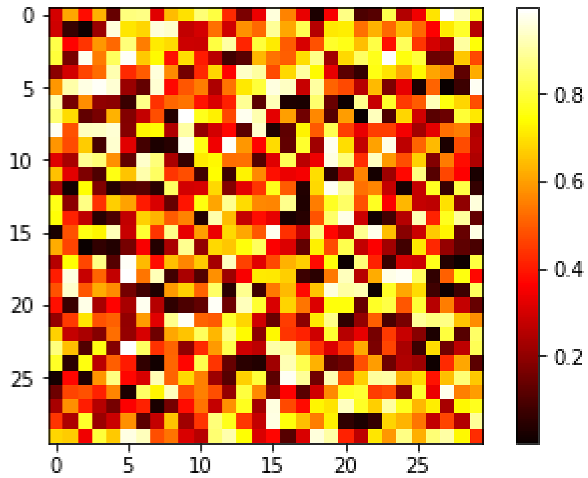
Getting started with Python for science

1. The Numpy array object
   4. Basic visualization

***Matplotlib* is a 2D plotting package.**

- **2D arrays (such as images):**

```
In [137]: image = np.random.rand(30, 30);plt.imshow(image, cmap=plt.cm.hot);\
     ...: plt.colorbar()
Out[137]: <matplotlib.colorbar.Colorbar at 0xc0664a6748>
```



NumPy

Getting started with Python for science

1. The Numpy array object
   5. Indexing and slicing

Indexing:

1

```
In [138]: a = np.arange(10)

In [139]: a
Out[139]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [140]: a[1]
Out[140]: 1

In [141]: a[5]
Out[141]: 5

In [142]: a[-1]
Out[142]: 9

In [143]: a[::-1]
Out[143]: array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
```

2

```
In [144]: a = np.diag(np.arange(3))

In [145]: a
Out[145]:
array([[0, 0, 0],
       [0, 1, 0],
       [0, 0, 2]])

In [146]: a[0][1]
Out[146]: 0

In [147]: a[0,1]
Out[147]: 0

In [148]: a[0,1] = 5

In [149]: a
Out[149]:
array([[0, 5, 0],
       [0, 1, 0],
       [0, 0, 2]])

In [150]: a[0]
Out[150]: array([0, 5, 0])
```

NumPy

Getting started with Python for science

1. The Numpy array object
   5. Indexing and slicing

Slicing:

1

```
In [151]: a = np.arange(10)

In [152]: a
Out[152]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [153]: a[2:9:3] # [start:end:step]
Out[153]: array([2, 5, 8])

In [154]: a[:4]
Out[154]: array([0, 1, 2, 3])

In [155]: a[1:3]
Out[155]: array([1, 2])

In [156]: a[::2]
Out[156]: array([0, 2, 4, 6, 8])

In [157]: a[3:]
Out[157]: array([3, 4, 5, 6, 7, 8, 9])
```

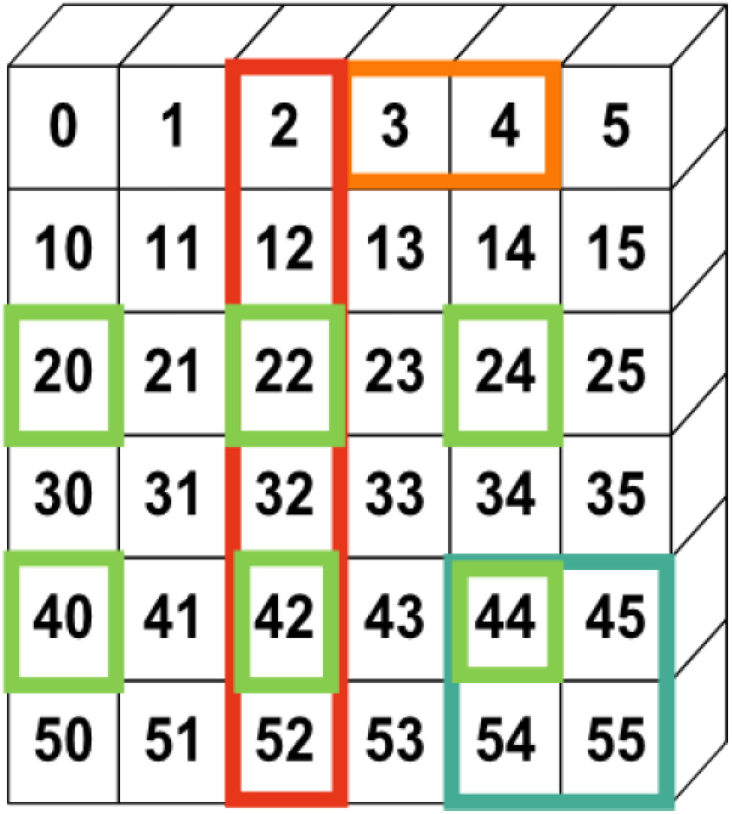
NumPy

Getting started with Python for science

1. The Numpy array object
    5. Indexing and slicing

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]])
```



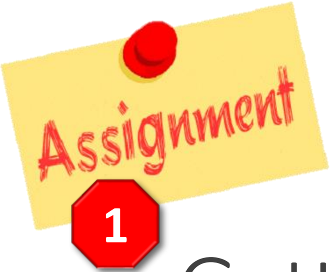
NumPy

Getting started with Python for science

1. The Numpy array object
   5. Indexing and slicing

You can also combine assignment and slicing:

```
In [158]: a = np.arange(10)

In [159]: a[5:] = 10

In [160]: a
Out[160]: array([ 0,  1,  2,  3,  4, 10, 10, 10, 10, 10])

In [161]: b = np.arange(5)

In [162]: b
Out[162]: array([0, 1, 2, 3, 4])

In [163]: a[5:] = b[::-1]

In [164]: a
Out[164]: array([0, 1, 2, 3, 4, 4, 3, 2, 1, 0])
```



Getting started with Python for science

1. The Numpy array object
   5. Indexing and slicing

Exercise: Indexing and slicing

- Create the following arrays according to the example and previous slides:

```
In [165]: np.arange(6) + np.arange(0, 51, 10)[:, np.newaxis]
Out[165]:
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]])
```

1

```
array([[ 0, 11, 22, 33, 44, 55],
       [ 0, 12, 22, 33, 44, 55],
       [ 0, 11, 24, 33, 44, 55],
       [ 0, 11, 22, 36, 44, 55],
       [ 0, 11, 22, 33, 48, 55],
       [ 0, 11, 22, 33, 44, 60]])
```

2

```
array([[ 0,  1,  2,  3,  4,  5],
       [10, 12, 12, 13, 14, 15],
       [20, 21, 24, 23, 24, 25],
       [30, 31, 32, 36, 34, 35],
       [40, 41, 42, 43, 48, 45],
       [50, 51, 52, 53, 54, 60]])
```

3

```
array([[ 0, 11, 22, 33, 44, 55],
       [ 0, 11, 24, 33, 44, 55],
       [ 0, 11, 22, 33, 48, 55]])
```

4

```
array([[ 0, 22, 44],
       [ 0, 22, 44],
       [ 0, 24, 44],
       [ 0, 22, 44],
       [ 0, 22, 48],
       [ 0, 22, 44]])
```

Assignment 1

Getting started with Python for science

1. The Numpy array object
   5. Indexing and slicing

Exercise: Array creation

Create the following arrays (with correct data types):

5

```
array([[ 1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  2.],
       [ 6.,  1.,  1.,  1.]])
```

6

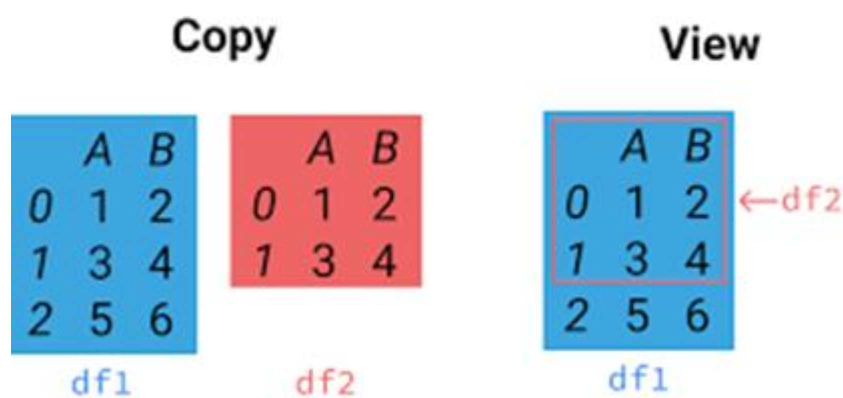
```
array([[ 2.,  0.,  0.,  0.,  0.],
       [ 0.,  3.,  0.,  0.,  0.],
       [ 0.,  0.,  4.,  0.,  0.],
       [ 0.,  0.,  0.,  5.,  0.],
       [ 0.,  0.,  0.,  0.,  6.]])
```

NumPy

Getting started with Python for science

1. The Numpy array object
   6. Copies and views

Copy   vs. View



NumPy

Getting started with Python for science

1. The Numpy array object
   6. Copies and views

1 View

```
In [168]: a = np.arange(10)

In [169]: a
Out[169]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [170]: b = a[::2]

In [171]: b
Out[171]: array([0, 2, 4, 6, 8])

In [172]: np.may_share_memory(a, b)
Out[172]: True

In [173]: b[0] = 12

In [174]: b
Out[174]: array([12,  2,  4,  6,  8])

In [175]: a
Out[175]: array([12,  1,  2,  3,  4,  5,  6,  7,  8,  9])
```

2 Copy

```
In [176]: a = np.arange(10)

In [177]: c = a[::2].copy() # force a copy

In [178]: c[0] = 12

In [179]: c
Out[179]: array([12,  2,  4,  6,  8])

In [180]: a
Out[180]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [181]: np.may_share_memory(a, c)
Out[181]: False
```

NumPy

Getting started with Python for science

1. The Numpy array object
   7. Fancy indexing

Numpy arrays can be indexed with slices, but also with boolean or integer arrays (**masks**). This method is called *fancy indexing*. It creates **copies not views**.

Using boolean masks

```
In [190]: a = np.random.randint(0,20,15)

In [191]: a
Out[191]: array([ 4,  3, 18,  2, 10, 18,  1,  3, 14,  2,  3,  1, 18,  3,  4])

In [192]: b = a % 3

In [193]: b
Out[193]: array([1, 0, 0, 2, 1, 0, 1, 0, 2, 2, 0, 1, 0, 0, 1], dtype=int32)

In [194]: c = (a % 3 == 0)

In [195]: c
Out[195]:
array([False,  True,  True, False, False,  True, False,  True, False,
       False,  True, False,  True,  True, False], dtype=bool)
```

NumPy

Getting started with Python for science

1. The Numpy array object
   7. Fancy indexing

Numpy arrays can be indexed with slices, but also with boolean or integer arrays (**masks**). This method is called *fancy indexing*. It creates **copies not views**.

Using boolean masks

```
In [196]: mask = (a % 3 == 0)

In [197]: extract_from_a = a[mask] # or, a[a%3==0]

In [198]: extract_from_a
Out[198]: array([ 3, 18, 18,  3,  3, 18,  3])

In [199]: a
Out[199]: array([ 4,  3, 18,  2, 10, 18,  1,  3, 14,  2,  3,  1, 18,  3,  4])
```

NumPy

Getting started with Python for science

1. The Numpy array object
   7. Fancy indexing

Indexing with a mask

```
In [200]: a[a % 3 == 0] = -1

In [201]: a
Out[201]: array([ 4, -1, -1,  2, 10, -1,  1, -1, 14,  2, -1,  1, -1, -1,  4])
```

NumPy

Getting started with Python for science

1. The Numpy array object

   7. Fancy indexing

Indexing with an array of integers

```
In [206]: a
Out[206]: array([   0,   10,   20,   30,   40,   50,   60, -100,   80, -100])

In [207]: a = np.arange(10)

In [208]: a
Out[208]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [209]: idx = np.array([[3, 4], [9, 7]])

In [210]: a[idx]
Out[210]:
array([[3, 4],
       [9, 7]])
```

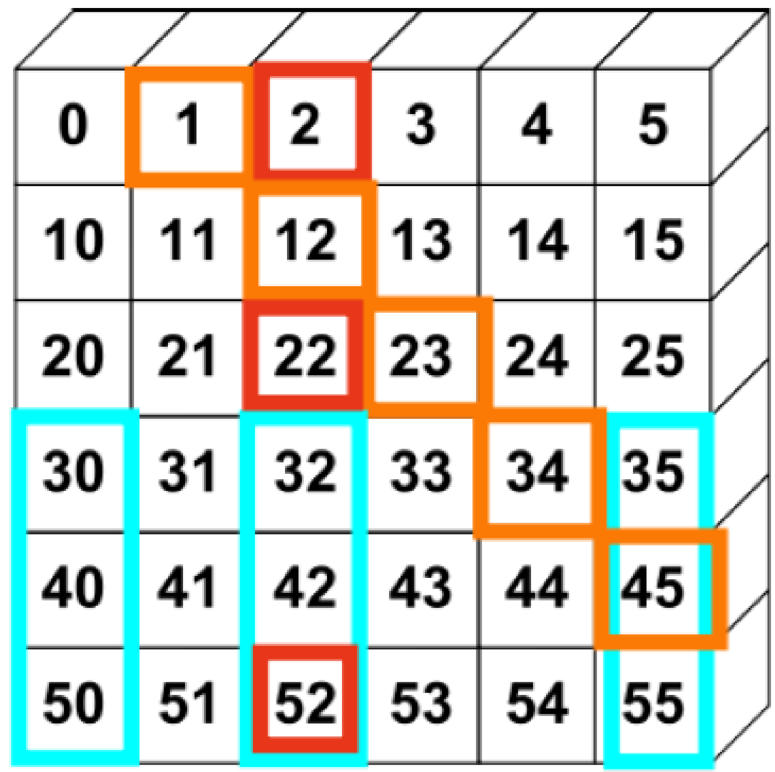
```
In [211]: a = a[::-1]

In [212]: a
Out[212]: array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])

In [213]: a[idx]
Out[213]:
array([[6, 5],
       [0, 2]])
```

NumPy

Getting started with Python for science

1. The Numpy array object
   7. Fancy indexing

```
>>> 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 = np.array([1,0,1,0,0,1],
                    dtype=bool)

>>> a[mask,2]
array([2,22,52])
```



NumPy

Getting started with Python for science

2. Numerical operations on arrays
   1. Elementwise operations

Basic operations

With scalars:

```
In [223]: a = np.array([1, 2, 3, 4])

In [224]: a + 1
Out[224]: array([2, 3, 4, 5])

In [225]: 2**a
Out[225]: array([ 2,  4,  8, 16], dtype=int32)

In [226]: a**2
Out[226]: array([ 1,  4,  9, 16], dtype=int32)
```

All arithmetic operates elementwise:

```
In [231]: b = np.ones(4) + 1

In [232]: a - b
Out[232]: array([-1.,  0.,  1.,  2.])

In [233]: a.shape
Out[233]: (4,)

In [234]: b.shape
Out[234]: (4,)

In [235]: a * b
Out[235]: array([ 2.,  4.,  6.,  8.])

In [236]: j = np.arange(5)

In [237]: 2**(j + 1) - j
Out[237]: array([ 2,  3,  6, 13, 28])
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
   1. Elementwise operations

Basic operations

These operations are of course much faster than if you did them in pure python:

```
In [238]: a = np.arange(10000)

In [239]: %timeit a + 1

100000 loops, best of 3: 6.58 µs per loop

In [240]: l = range(10000)

In [241]: %timeit [i+1 for i in l]
1000 loops, best of 3: 655 µs per loop
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
   1. Elementwise operations

Basic operations

Array multiplication is not matrix multiplication:

```
In [245]: e = 2*np.ones((3, 3))

In [246]: c = 3*np.ones((3, 3))

In [247]: c * e # NOT matrix multiplication!
Out[247]:
array([[ 6.,  6.,  6.],
       [ 6.,  6.,  6.],
       [ 6.,  6.,  6.]])
```

Matrix multiplication:

```
In [248]: c.dot(e)
Out[248]:
array([[ 18.,  18.,  18.],
       [ 18.,  18.,  18.],
       [ 18.,  18.,  18.]])
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
   1. Elementwise operations

Other operations

Comparisons:

```
In [249]: a = np.array([1, 2, 3, 4])

In [250]: b = np.array([4, 2, 2, 4])

In [251]: a == b
Out[251]: array([False,  True, False,  True], dtype=bool)

In [252]: a > b
Out[252]: array([False, False,  True, False], dtype=bool)
```

Array-wise comparisons:

```
In [253]: a = np.array([1, 2, 3, 4])

In [254]: b = np.array([4, 2, 2, 4])

In [255]: c = np.array([1, 2, 3, 4])

In [256]: np.array_equal(a, b)
Out[256]: False

In [257]: np.array_equal(a, c)
Out[257]: True
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
    1. Elementwise operations

Other operations

Logical operations:

```
In [258]: a = np.array([1, 1, 0, 0], dtype=bool)

In [259]: b = np.array([1, 0, 1, 0], dtype=bool)

In [260]: np.logical_or(a, b)
Out[260]: array([ True,  True,  True, False], dtype=bool)

In [261]: np.logical_and(a, b)
Out[261]: array([ True, False, False, False], dtype=bool)
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
    1. Elementwise operations

Other operations

Transcendental functions:

```
In [269]: a = np.arange(5)

In [270]: np.sin(a)
Out[270]: array([ 0.        ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ])

In [271]: np.log(a)
C:\Program Files\Anaconda3\lib\site-packages\spyder\utils\ipython\start_kernel.py:1: RuntimeWarning:
divide by zero encountered in log
  # -*- coding: utf-8 -*-
Out[271]: array([       -inf,  0.        ,  0.69314718,  1.09861229,  1.38629436])

In [272]: np.exp(a)
Out[272]: array([  1.        ,   2.71828183,   7.3890561 ,  20.08553692,  54.59815003])
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
    1. Elementwise operations

Other operations

Transposition:

```
In [273]: a = np.triu(np.ones((3, 3)), 1) # see help(np.triu)

In [274]: a
Out[274]:
array([[ 0.,  1.,  1.],
       [ 0.,  0.,  1.],
       [ 0.,  0.,  0.]])

In [275]: a.T
Out[275]:
array([[ 0.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  1.,  0.]])
```

NumPy

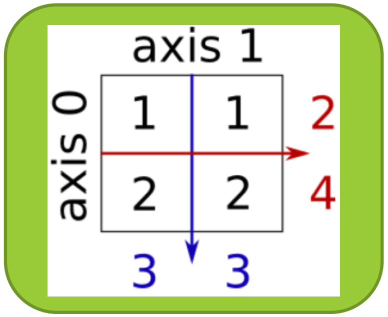
Getting started with Python for science

2. Numerical operations on arrays
   2. Basic reductions

Computing sums

1

```
In [286]: x = np.array([1, 2, 3, 4])

In [287]: np.sum(x)
Out[287]: 10

In [288]: x.sum()
Out[288]: 10
```



Sum by rows and by columns:

2

```
In [289]: x = np.array([[1, 1], [2, 2]])

In [290]: x
Out[290]:
array([[1, 1],
       [2, 2]])

In [291]: x.sum(axis=0) # columns (first dimension)
Out[291]: array([3, 3])

In [292]: x[:, 0].sum(), x[:, 1].sum()
Out[292]: (3, 3)

In [293]: x.sum(axis=1) # rows (second dimension)
Out[293]: array([2, 4])

In [294]: x[0, :].sum(), x[1, :].sum()
Out[294]: (2, 4)
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
   2. Basic reductions

Computing sums

```
In [298]: x = np.random.rand(2, 2, 2)

In [299]: x
Out[299]:
array([[[ 0.69722146,  0.45759804],
        [ 0.84950577,  0.78479037]],

       [[ 0.78136544,  0.77042684],
        [ 0.08319696,  0.91118212]]])

In [300]: x.sum(axis=2)[0, 1]
Out[300]: 1.6342961482576455

In [301]: x[0, 1, :].sum()
Out[301]: 1.6342961482576455
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
   2. Basic reductions

Other reductions

—works the same way (and take axis=)

```
In [302]: x = np.array([1, 3, 2])

In [303]: x.min()
Out[303]: 1

In [304]: x.max()
Out[304]: 3

In [305]: x.argmin() # index of minimum
Out[305]: 0

In [306]: x.argmax() # index of maximum
Out[306]: 1
```

Logical operations:

```
In [3]: np.all([True, True, False])
Out[3]: False

In [4]: np.any([True, True, False])
Out[4]: True

In [5]: np.all([True, True, True])
Out[5]: True

In [6]: np.any([False, False, False])
Out[6]: False
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
   2. Basic reductions

Other reductions

Logical operations:

```
In [315]: a = np.zeros((100, 100))

In [316]: b = (a != 0)

In [317]: b
Out[317]:
array([[False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False],
       ...,
       [False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False]], dtype=bool)

In [318]: np.any(a != 0)
Out[318]: False
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
   2. Basic reductions

Other reductions

Logical operations:

```
In [319]: b = (a == a)

In [320]: b
Out[320]:
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]], dtype=bool)

In [321]: np.all(a == a)
Out[321]: True
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
   2. Basic reductions

Other reductions

Logical operations:

```
In [331]: a = np.array([1, 2, 3, 2])

In [332]: b = np.array([2, 2, 3, 2])

In [333]: c = np.array([6, 4, 4, 5])

In [334]: e = ((a <= b) & (b <= c))

In [335]: e
Out[335]: array([ True,  True,  True,  True], dtype=bool)

In [336]: ((a <= b) & (b <= c)).all()
Out[336]: True
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
   2. Basic reductions

Other reductions

Statistics:

```
In [344]: x = np.array([1, 2, 3, 1])

In [345]: y = np.array([[1, 2, 3], [5, 6, 1]])

In [346]: x.mean()
Out[346]: 1.75

In [347]: np.median(x)
Out[347]: 1.5

In [348]: np.median(y, axis=-1) # last axis
Out[348]: array([ 2.,  5.])

In [349]: y
Out[349]:
array([[1, 2, 3],
       [5, 6, 1]])

In [350]: x.std() # full population standard dev.
Out[350]: 0.82915619758884995

In [351]: y.shape
Out[351]: (2, 3)
```

Given a vector V of length N, the **median** of V is:

- **N is odd:** the middle value of a sorted copy of V, ie., V_sorted[(N-1)/2].
- **N is even:** the average of the two middle values of V_sorted.

NumPy

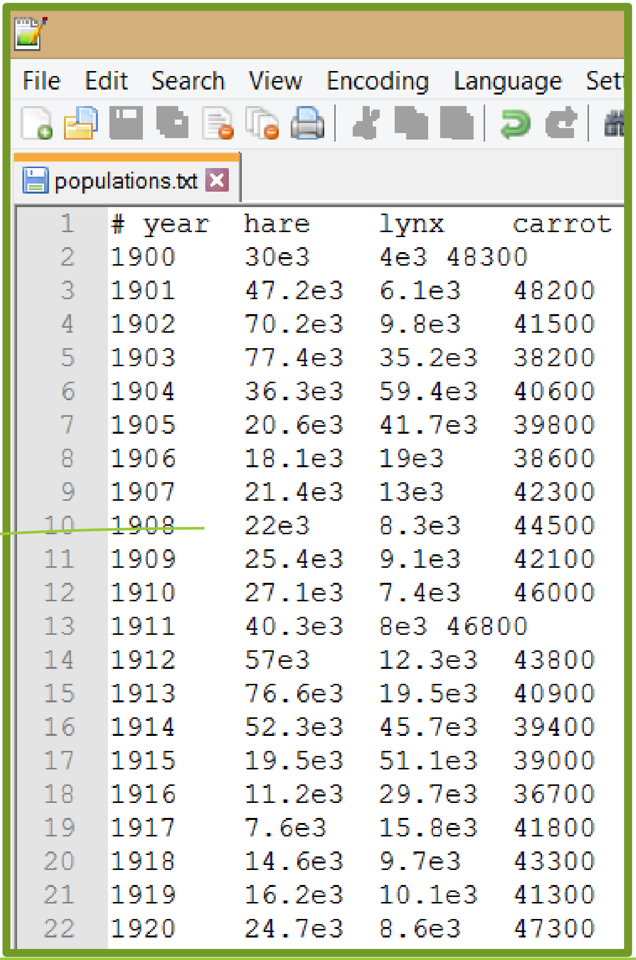
Getting started with Python for science

2. Numerical operations on arrays
   2. Basic reductions

Other reductions

Worked Example: data statistics:

<TAB> or space



# year	hare	lynx	carrot
1900	30e3	4e3	48300
1901	47.2e3	6.1e3	48200
1902	70.2e3	9.8e3	41500
1903	77.4e3	35.2e3	38200
1904	36.3e3	59.4e3	40600
1905	20.6e3	41.7e3	39800
1906	18.1e3	19e3	38600
1907	21.4e3	13e3	42300
1908	22e3	8.3e3	44500
1909	25.4e3	9.1e3	42100
1910	27.1e3	7.4e3	46000
1911	40.3e3	8e3	46800
1912	57e3	12.3e3	43800
1913	76.6e3	19.5e3	40900
1914	52.3e3	45.7e3	39400
1915	19.5e3	51.1e3	39000
1916	11.2e3	29.7e3	36700
1917	7.6e3	15.8e3	41800
1918	14.6e3	9.7e3	43300
1919	16.2e3	10.1e3	41300
1920	24.7e3	8.6e3	47300

NumPy

Getting started with Python for science

2. Numerical operations on arrays
    2. Basic reductions

Other reductions

Worked Example: data statistics:

```
In [13]: data = np.loadtxt('E:/gpu/presentations/final/data/populations.txt')

In [14]: year, hares, lynxes, carrots = data.T # trick: columns to variables
```

variable explorer

```
In [142]: type(data)
Out[142]: numpy.ndarray

In [143]: data.shape
Out[143]: (21, 4)
```

Name	Type	Size	Value
carrots	float64	(21,)	array([48300., 48200., 41500., 38200., 40600., 39800., 38600., 42300., 44500., 42100., 46000., 46800., 43800., 40900.,
data	float64	(21, 4)	array([[1900., 30000., 4000., 48300.], [1901., 47200., 6100., 48200.],
hares	float64	(21,)	array([30000., 47200., 70200., 77400., 36300., 20600., 18100., 21400., 22000., 25400., 27100., 40300., 57000., 76600.,
lynxes	float64	(21,)	array([4000., 6100., 9800., 35200., 59400., 41700., 19000., 13000., 8300., 9100., 7400., 8000., 12300., 19500.,
year	float64	(21,)	array([1900., 1901., 1902., 1903., 1904., 1905., 1906., 1907., 1908., 1909., 1910., 1911., 1912., 1913., 1914., 1915.,

Getting started with Python for science

2. Numerical operations on arrays
   2. Basic reductions

Other reductions

Worked Example: data statistics:

```
In [39]: plt.axes([0, 0, 0.5, 0.8]);plt.plot(year, hares, year, lynxes, year,
carrots);plt.legend(('Hare', 'Lynx', 'Carrot'), loc=(1.05, 0.5))
Out[39]: <matplotlib.legend.Legend at 0x87118afeb8>
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
   2. Basic reductions

Other reductions

Worked Example: data statistics:

1 **The mean populations over time:**

```
In [40]: populations = data[:, 1:]

In [41]: populations.shape
Out[41]: (21, 3)

In [42]: populations.mean(axis=0)
Out[42]: array([ 34080.95238095,  20166.66666667,  42400.        ])
```

2 **The sample standard deviations:**

```
In [43]: populations.std(axis=0)
Out[43]: array([ 20897.90645809,  16254.59153691,   3322.50622558])
```

3 **Which species has the highest population each year?:**

```
In [44]: np.argmax(populations, axis=1)
Out[44]: array([2, 2, 0, 0, 1, 1, 2, 2, 2, 2, 2, 2, 0, 0, 0, 1, 2, 2, 2, 2, 2], dtype=int64)
```

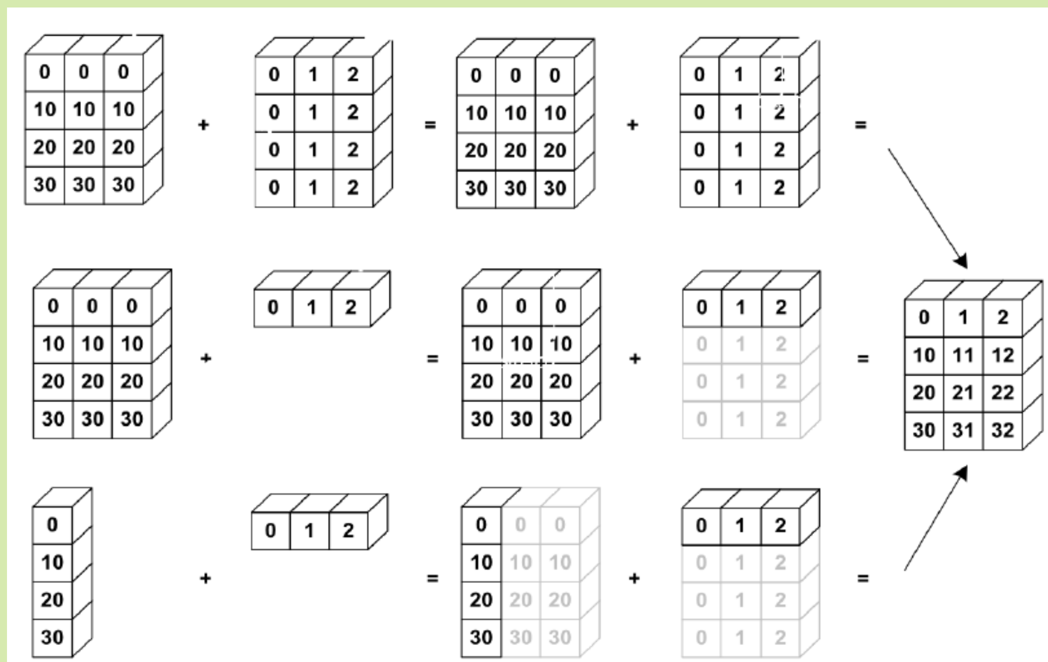
NumPy

Getting started with Python for science

2. Numerical operations on arrays
   3. Broadcasting

- Basic operations on numpy arrays (addition, etc.) are **elementwise**
- This works on arrays of the **same size**.

Nevertheless, It’s also possible to do operations on arrays of different sizes if Numpy can transform these arrays so that they all have the same size: this conversion is called broadcasting.
The image below gives an example of **broadcasting**:



NumPy

Getting started with Python for science

2. Numerical operations on arrays
   3. Broadcasting

```
In [49]: a = np.tile(np.arange(0, 40, 10), (3, 1)).T

In [50]: a
Out[50]:
array([[ 0,  0,  0],
       [10, 10, 10],
       [20, 20, 20],
       [30, 30, 30]])

In [51]: a1=np.arange(0, 40, 10)

In [52]: a2 = np.tile(np.arange(0, 40, 10), (3, 1))
```

variable explorer

Name	Type	Size	
a	int32	(4, 3)	array([[0, 0, 0], [10, 10, 10],
a1	int32	(4,)	array([0, 10, 20, 30])
a2	int32	(3, 4)	array([[0, 10, 20, 30], [0, 10, 20, 30],

NumPy

Getting started with Python for science

2. Numerical operations on arrays
   3. Broadcasting

```
In [13]: a = np.tile(np.arange(0, 40, 10), (3, 1)).T

In [14]: b = np.array([0, 1, 2])

In [15]: c = a + b

In [16]: a
Out[16]:
array([[ 0,  0,  0],
       [10, 10, 10],
       [20, 20, 20],
       [30, 30, 30]])

In [17]: b
Out[17]: array([0, 1, 2])

In [18]: c
Out[18]:
array([[ 0,  1,  2],
       [10, 11, 12],
       [20, 21, 22],
       [30, 31, 32]])
```

Name	Type	Size	
a	int32	(4, 3)	array([[0, 0, 0], [10, 10, 10],
b	int32	(3,)	array([0, 1, 2])
c	int32	(4, 3)	array([[0, 1, 2], [10, 11, 12],

```
In [10]: a.shape
Out[10]: (4, 3)

In [11]: b.shape
Out[11]: (3,)

In [12]: c.shape
Out[12]: (4, 3)
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
   3. Broadcasting

```
In [21]: b = np.array([0,1,2,3])
```

✗

```
In [22]: a + b
Traceback (most recent call last):

  File "<ipython-input-22-f96fb8f649b6>", line 1, in <module>
    a + b

ValueError: operands could not be broadcast together with shapes (4,3) (4,)


In [23]: b + a
Traceback (most recent call last):

  File "<ipython-input-23-a50d69c311c0>", line 1, in <module>
    b + a

ValueError: operands could not be broadcast together with shapes (4,) (4,3)


In [24]: a + b.T
Traceback (most recent call last):

  File "<ipython-input-24-a85037d5d521>", line 1, in <module>
    a + b.T

ValueError: operands could not be broadcast together with shapes (4,3) (4,)
```

```
In [14]: b = np.array([0, 1, 2])
```

✓

```
In [19]: a+b.T
Out[19]:
array([[ 0,  1,  2],
       [10, 11, 12],
       [20, 21, 22],
       [30, 31, 32]])

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

NumPy

Getting started with Python for science

2. Numerical operations on arrays
   3. Broadcasting

1

```
In [25]: b = np.array([[0,1,2]])

In [26]: b.shape
Out[26]: (1, 3)

In [27]: b
Out[27]: array([[0, 1, 2]])

In [28]: a + b
Out[28]:
array([[ 0,  1,  2],
       [10, 11, 12],
       [20, 21, 22],
       [30, 31, 32]])
```

2

```
In [29]: a + b.T
Traceback (most recent call last):

  File "<ipython-input-29-a85037d5d521>", line 1, in <module>
    a + b.T

ValueError: operands could not be broadcast together with shapes (4,3) (3,1)


In [30]: b.T.shape
Out[30]: (3, 1)
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
   3. Broadcasting

```
In [31]: b = np.array([[0],[1],[2],[3]])

In [32]: b.shape
Out[32]: (4, 1)

In [33]: c = a + b

In [34]: c
Out[34]:
array([[ 0,  0,  0],
       [11, 11, 11],
       [22, 22, 22],
       [33, 33, 33]])

In [35]: c.shape
Out[35]: (4, 3)
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
   3. Broadcasting

```
In [42]: a = np.ones((4, 5))

In [43]: a
Out[43]:
array([[ 1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.]])

In [44]: a[0] = 2 # we assign an array of dimension 0 to an array of dimension 1

In [45]: a
Out[45]:
array([[ 2.,  2.,  2.,  2.,  2.],
       [ 1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.]])

In [46]: a[0]
Out[46]: array([ 2.,  2.,  2.,  2.,  2.])
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
   3. Broadcasting

1

```
In [66]: a = np.arange(0, 40, 10)

In [67]: a
Out[67]: array([ 0, 10, 20, 30])

In [68]: a.shape
Out[68]: (4,)

In [69]: a = a[:, np.newaxis]

In [70]: a
Out[70]:
array([[ 0],
       [10],
       [20],
       [30]])

In [71]: a.shape
Out[71]: (4, 1)
```

2

```
In [72]: a = a[:, np.newaxis]

In [73]: a.shape
Out[73]: (4, 1, 1)

In [74]: a = np.arange(0, 40, 10)

In [75]: a = a[:2, np.newaxis]

In [76]: a.shape
Out[76]: (2, 1)

In [77]: a
Out[77]:
array([[ 0],
       [10]])
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
    3. Broadcasting

```
In [78]: a = np.arange(0, 40, 10)

In [79]: a.shape
Out[79]: (4,)

In [80]: b = np.array([0, 1, 2])

In [81]: b.shape
Out[81]: (3,)

In [82]: a + b
Traceback (most recent call last):

  File "<ipython-input-82-f96fb8f649b6>", line 1, in <module>
    a + b

ValueError: operands could not be broadcast together with shapes (4,) (3,)
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
   3. Broadcasting

1

```
In [83]: a = a[:, np.newaxis]

In [84]: a + b
Out[84]:
array([[ 0,  1,  2],
       [10, 11, 12],
       [20, 21, 22],
       [30, 31, 32]])

In [85]: a.shape
Out[85]: (4, 1)

In [86]: a
Out[86]:
array([[ 0],
       [10],
       [20],
       [30]])
```

2

```
In [87]: (a+b).shape
Out[87]: (4, 3)

In [88]: (b+a).shape
Out[88]: (4, 3)

In [89]: b = b[:, np.newaxis]

In [90]: b.shape
Out[90]: (3, 1)

In [91]: a = np.arange(0, 40, 10)

In [92]: a.shape
Out[92]: (4,)

In [93]: (a+b).shape
Out[93]: (3, 4)
```

NumPy

Self Study

Getting started with Python for science

2. Numerical operations on arrays
   3. Broadcasting

Worked Example: Broadcasting

Let's construct an array of distances (in miles) between cities of Route 66: Chicago, Springfield, Saint- Louis, Tulsa, Oklahoma City, Amarillo, Santa Fe, Albuquerque, Flagstaff and Los Angeles.

```
In [100]: mileposts = np.array([0, 198, 303, 736, 871, 1175, 1475, 1544,
     ...: ... 1913, 2448])

In [101]: mileposts.shape
Out[101]: (10,)

In [102]: distance_array = np.abs(mileposts - mileposts[:, np.newaxis])

In [103]: distance_array.shape
Out[103]: (10, 10)
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
   3. Broadcasting

```
In [100]: mileposts = np.array([0, 198, 303, 736, 871, 1175, 1475, 1544,
     ...: ... 1913, 2448])

In [101]: mileposts.shape
Out[101]: (10,)

In [102]: distance_array = np.abs(mileposts - mileposts[:, np.newaxis])

In [103]: distance_array.shape
Out[103]: (10, 10)
```

```
In [104]: distance_array
Out[104]:
array([[   0,  198,  303,  736,  871, 1175, 1475, 1544, 1913, 2448],
       [ 198,    0,  105,  538,  673,  977, 1277, 1346, 1715, 2250],
       [ 303,  105,    0,  433,  568,  872, 1172, 1241, 1610, 2145],
       [ 736,  538,  433,    0,  135,  439,  739,  808, 1177, 1712],
       [ 871,  673,  568,  135,    0,  304,  604,  673, 1042, 1577],
       [1175,  977,  872,  439,  304,    0,  300,  369,  738, 1273],
       [1475, 1277, 1172,  739,  604,  300,    0,   69,  438,  973],
       [1544, 1346, 1241,  808,  673,  369,   69,    0,  369,  904],
       [1913, 1715, 1610, 1177, 1042,  738,  438,  369,    0,  535],
       [2448, 2250, 2145, 1712, 1577, 1273,  973,  904,  535,    0]])
```

NumPy

Self Study

Getting started with Python for science

2. Numerical operations on arrays
   3. Broadcasting

```
In [105]: mileposts = np.array([0, 198, 303, 736, 871, 1175, 1475, 1544,
     ...: ... 1913, 2448])
```

```
In [104]: distance_array
Out[104]:
array([[   0,  198,  303,  736,  871, 1175, 1475, 1544, 1913, 2448],
       [ 198,    0,  105,  538,  673,  977, 1277, 1346, 1715, 2250],
       [ 303,  105,    0,  433,  568,  872, 1172, 1241, 1610, 2145],
       [ 736,  538,  433,    0,  135,  439,  739,  808, 1177, 1712],
       [ 871,  673,  568,  135,    0,  304,  604,  673, 1042, 1577],
       [1175,  977,  872,  439,  304,    0,  300,  369,  738, 1273],
       [1475, 1277, 1172,  739,  604,  300,    0,   69,  438,  973],
       [1544, 1346, 1241,  808,  673,  369,   69,    0,  369,  904],
       [1913, 1715, 1610, 1177, 1042,  738,  438,  369,    0,  535],
       [2448, 2250, 2145, 1712, 1577, 1273,  973,  904,  535,    0]])
```

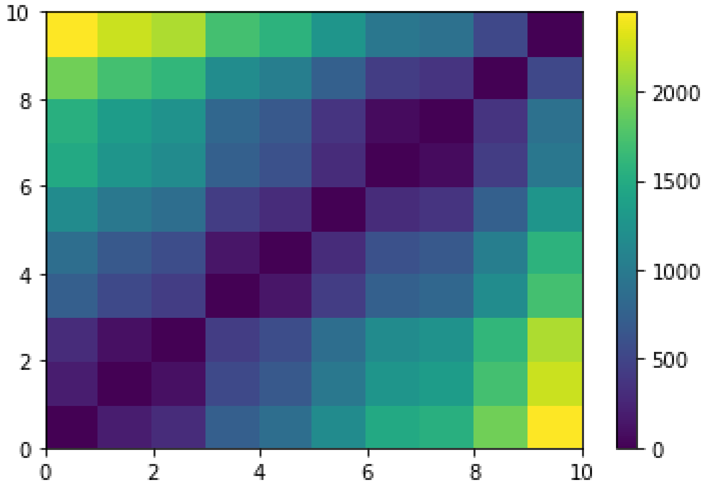
```
In [106]: mileposts[:, np.newaxis]
Out[106]:
array([[   0],
       [ 198],
       [ 303],
       [ 736],
       [ 871],
       [1175],
       [1475],
       [1544],
       [1913],
       [2448]])
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
   3. Broadcasting

```
In [124]: plt.pyplot.pcolor(distance_array);plt.pyplot.colorbar()
Out[124]: <matplotlib.colorbar.Colorbar at 0x85e402a4e0>
```



NumPy

Getting started with Python for science

2. Numerical operations on arrays
   3. Broadcasting

The **numpy.ogrid** function allows to directly create vectors x and y of with two “significant dimensions”:

```
In [12]: x, y = np.ogrid[0:5, 0:5]

In [13]: x, y
Out[13]:
(array([[0],
        [1],
        [2],
        [3],
        [4]]), array([[0, 1, 2, 3, 4]]))

In [14]: x.shape, y.shape
Out[14]: ((5, 1), (1, 5))
```

```
In [15]: distance = np.sqrt(x ** 2 + y ** 2)

In [16]: distance
Out[16]:
array([[ 0.        ,  1.        ,  2.        ,  3.        ,  4.        ],
       [ 1.        ,  1.41421356,  2.23606798,  3.16227766,  4.12310563],
       [ 2.        ,  2.23606798,  2.82842712,  3.60555128,  4.47213595],
       [ 3.        ,  3.16227766,  3.60555128,  4.24264069,  5.        ],
       [ 4.        ,  4.12310563,  4.47213595,  5.        ,  5.65685425]])
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
   3. Broadcasting

np.mgrid directly provides matrices full of indices for cases where we can’t (or don’t want to) benefit from broadcasting:

```
In [17]: x, y = np.mgrid[0:4, 0:4]

In [18]: x
Out[18]:
array([[0, 0, 0, 0],
       [1, 1, 1, 1],
       [2, 2, 2, 2],
       [3, 3, 3, 3]])

In [19]: y
Out[19]:
array([[0, 1, 2, 3],
       [0, 1, 2, 3],
       [0, 1, 2, 3],
       [0, 1, 2, 3]])
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
   4. Array shape manipulation

Flattening

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

In [21]: a.ravel()
Out[21]: array([1, 2, 3, 4, 5, 6])

In [22]: a.T
Out[22]:
array([[1, 4],
       [2, 5],
       [3, 6]])

In [23]: a.T.ravel()
Out[23]: array([1, 4, 2, 5, 3, 6])
```

The inverse operation to flattening:

Reshaping

```
In [24]: a.shape
Out[24]: (2, 3)

In [25]: b = a.ravel()

In [26]: b.shape
Out[26]: (6,)

In [27]: b = b.reshape((2, 3))

In [28]: b.shape
Out[28]: (2, 3)

In [29]: b
Out[29]:
array([[1, 2, 3],
       [4, 5, 6]])
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
   4. Array shape manipulation

Reshaping

```
In [30]: a.reshape((2, -1)) # unspecified (-1) value is inferred
Out[30]:
array([[1, 2, 3],
       [4, 5, 6]])
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
    4. Array shape manipulation

View or Copy?

1

```
In [64]: a = np.array([[1, 2, 3], [4, 5, 6]])

In [65]: b = a.ravel()

In [66]: b = b.reshape(3, 2)

In [67]: b[0, 0] = 22

In [68]: b
Out[68]:
array([[22,  2],
       [ 3,  4],
       [ 5,  6]])

In [69]: a
Out[69]:
array([[22,  2,  3],
       [ 4,  5,  6]])
```

2

```
In [70]: a = np.array([[1, 2, 3], [4, 5, 6]])

In [71]: b = a.ravel()

In [72]: b[0] = 33

In [73]: b
Out[73]: array([33,  2,  3,  4,  5,  6])

In [74]: a
Out[74]:
array([[33,  2,  3],
       [ 4,  5,  6]])
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
   4. Array shape manipulation

View or Copy?

1

```
In [79]: a = np.array([[1, 2, 3], [4, 5, 6]])

In [80]: b = a.reshape(3, 2)

In [81]: b
Out[81]:
array([[1, 2],
       [3, 4],
       [5, 6]])

In [82]: b[0, 1] = 44
```

2

```
In [83]: b
Out[83]:
array([[ 1, 44],
       [ 3,  4],
       [ 5,  6]])

In [84]: a
Out[84]:
array([[ 1, 44,  3],
       [ 4,  5,  6]])
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
   4. Array shape manipulation

View or Copy?

1

```
In [103]: a = np.zeros((3, 2))

In [104]: b = a.T.reshape(3*2)

In [105]: b[0] = 22

In [106]: b
Out[106]: array([ 22.,   0.,   0.,   0.,   0.,   0.])

In [107]: a
Out[107]:
array([[ 0.,  0.],
       [ 0.,  0.],
       [ 0.,  0.]])
```

2

```
In [112]: a = np.array([[1, 2, 3], [4, 5, 6]])

In [113]: b = a.T.reshape(3, 2)

In [114]: b[0 , 0] = 22

In [115]: b
Out[115]:
array([[22,  4],
       [ 2,  5],
       [ 3,  6]])

In [116]: a
Out[116]:
array([[22,  2,  3],
       [ 4,  5,  6]])
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
   4. Array shape manipulation

View or Copy?

1

```
In [117]: a = np.zeros((3, 2))

In [118]: b = a.reshape(2, 3)

In [119]: b[0, 0] = 33

In [120]: b
Out[120]:
array([[ 33.,   0.,   0.],
       [  0.,   0.,   0.]])

In [121]: a
Out[121]:
array([[ 33.,   0.],
       [  0.,   0.],
       [  0.,   0.]])
```

2

```
In [125]: a = np.zeros((3, 2))

In [126]: b = a.reshape(3*2).copy()

In [127]: b[0] = 44

In [128]: b
Out[128]: array([ 44.,   0.,   0.,   0.,   0.,   0.])

In [129]: a
Out[129]:
array([[ 0.,  0.],
       [ 0.,  0.],
       [ 0.,  0.]])
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
   4. Array shape manipulation

View or Copy?

```
In [125]: a = np.zeros((3, 2))

In [126]: b = a.reshape(3*2).copy()

In [127]: b[0] = 44

In [128]: b
Out[128]: array([ 44.,   0.,   0.,   0.,   0.,   0.])

In [129]: a
Out[129]:
array([[ 0.,  0.],
       [ 0.,  0.],
       [ 0.,  0.]])
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
   4. Array shape manipulation

Adding a dimension

1

```
In [147]: z = np.array([1, 2, 3])

In [148]: z
Out[148]: array([1, 2, 3])

In [149]: z.shape
Out[149]: (3,)

In [150]: z1 = z[:, np.newaxis]

In [151]: z1
Out[151]:
array([[1],
       [2],
       [3]])

In [152]: z1.shape
Out[152]: (3, 1)
```

2

```
In [153]: z2 = z[np.newaxis, :]

In [154]: z2
Out[154]: array([[1, 2, 3]])

In [155]: z2.shape
Out[155]: (1, 3)
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
   4. Array shape manipulation

Dimension shuffling

```
In [156]: a = np.arange(4*3*2).reshape(4, 3, 2)

In [157]: a.shape
Out[157]: (4, 3, 2)
```

1

```
In [158]: a
Out[158]:
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]]])
```

2

```
In [160]: b = a.transpose(1, 2, 0)

In [161]: b
Out[161]:
array([[[ 0,  6, 12, 18],
        [ 1,  7, 13, 19]],

       [[ 2,  8, 14, 20],
        [ 3,  9, 15, 21]],

       [[ 4, 10, 16, 22],
        [ 5, 11, 17, 23]]])
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
    4. Array shape manipulation

1

```
In [167]: b1 = a.T

In [168]: b1
Out[168]:
array([[[ 0,  6, 12, 18],
        [ 2,  8, 14, 20],
        [ 4, 10, 16, 22]],

       [[ 1,  7, 13, 19],
        [ 3,  9, 15, 21],
        [ 5, 11, 17, 23]]])

In [169]: b
Out[169]:
array([[[ 0,  6, 12, 18],
        [ 1,  7, 13, 19]],

       [[ 2,  8, 14, 20],
        [ 3,  9, 15, 21]],

       [[ 4, 10, 16, 22],
        [ 5, 11, 17, 23]]])
```

2

```
In [170]: b1.shape
Out[170]: (2, 3, 4)

In [171]: a.shape
Out[171]: (4, 3, 2)

In [172]: b.shape
Out[172]: (3, 2, 4)
```

```
In [149]: b=a.transpose(2,1,0)

In [150]: b
Out[150]:
array([[[ 0,  6, 12, 18],
        [ 2,  8, 14, 20],
        [ 4, 10, 16, 22]],

       [[ 1,  7, 13, 19],
        [ 3,  9, 15, 21],
        [ 5, 11, 17, 23]]])
```

3

```
In [173]: b1[0 , 0, 0] = 200

In [174]: a
Out[174]:
array([[[200,   1],
        [  2,   3],
        [  4,   5]],

       [[  6,   7],
        [  8,   9],
        [ 10,  11]],

       [[ 12,  13],
        [ 14,  15],
        [ 16,  17]],

       [[ 18,  19],
        [ 20,  21],
        [ 22,  23]]])
```

4

```
In [175]: b
Out[175]:
array([[[200,   6,  12,  18],
        [  1,   7,  13,  19]],

       [[  2,   8,  14,  20],
        [  3,   9,  15,  21]],

       [[  4,  10,  16,  22],
        [  5,  11,  17,  23]]])
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
   4. Array shape manipulation

Resizing

```
In [26]: a = np.arange(4)

In [27]: a.resize((8,))

In [28]: a
Out[28]: array([0, 1, 2, 3, 0, 0, 0, 0])
```

It must not be referred to somewhere else:

```
In [29]: a = np.arange(4)

In [30]: b = a

In [31]: a.resize((8,))
Traceback (most recent call last):

  File "<ipython-input-31-8d601eb51aa2>", line 1, in <module>
    a.resize((8,))

ValueError: cannot resize an array that references or is referenced
by another array in this way.  Use the resize function
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
   5. Sorting data

Sorting along an axis:

```
In [1]: import numpy as np

In [2]: a = np.array([[4, 3, 5], [1, 2, 1]])

In [3]: a
Out[3]:
array([[4, 3, 5],
       [1, 2, 1]])

In [4]: b = np.sort(a, axis=1)

In [5]: b
Out[5]:
array([[3, 4, 5],
       [1, 1, 2]])
```

In-place sort:

```
In [6]: a
Out[6]:
array([[4, 3, 5],
       [1, 2, 1]])

In [7]: a.sort(axis=1)

In [8]: a
Out[8]:
array([[3, 4, 5],
       [1, 1, 2]])
```

NumPy

Getting started with Python for science

2. Numerical operations on arrays
   5. Sorting data

Sorting with fancy indexing:

```
In [9]: a = np.array([4, 3, 1, 2])

In [10]: j = np.argsort(a)

In [11]: j
Out[11]: array([2, 3, 1, 0], dtype=int64)

In [12]: a
Out[12]: array([4, 3, 1, 2])

In [13]: a[j]
Out[13]: array([1, 2, 3, 4])
```

Finding minima and maxima:

```
In [14]: a = np.array([4, 3, 1, 2])

In [15]: j_max = np.argmax(a)

In [16]: j_min = np.argmin(a)

In [17]: j_max, j_min
Out[17]: (0, 2)
```

NumPy

Getting started with Python for science

3. More elaborate arrays
   1. More data types

Casting

"Bigger" type wins in mixed-type operations:

```
In [1]: import numpy as np

In [2]: np.array([1, 2, 3]) + 1.5
Out[2]: array([ 2.5,  3.5,  4.5])
```

Assignment never changes the type!

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

In [4]: a.dtype
Out[4]: dtype('int32')

In [5]: a[0] = 1.9 # <-- float is truncated to integer

In [6]: a
Out[6]: array([1, 2, 3])
```

Forced casts:

```
In [7]: a = np.array([1.7, 1.2, 1.6])

In [8]: b = a.astype(int) # <-- truncates to integer

In [9]: b
Out[9]: array([1, 1, 1])
```

NumPy

Getting started with Python for science

3. More elaborate arrays
   1. More data types

Casting

Rounding:

```
In [10]: a = np.array([1.2, 1.5, 1.6, 2.5, 3.5, 4.5])

In [11]: b = np.around(a)

In [12]: b # still floating-point
Out[12]: array([ 1.,  2.,  2.,  2.,  4.,  4.])

In [13]: c = np.around(a).astype(int)

In [14]: c
Out[14]: array([1, 2, 2, 2, 4, 4])
```

NumPy

Self Study

Getting started with Python for science

3. More elaborate arrays
   1. More data types

Different data type sizes

Integers (signed):

int8	8 bits
int16	16 bits
int32	32 bits (same as int on 32-bit platform)
int64	64 bits (same as int on 64-bit platform)

```
In [19]: np.array([1], dtype=int).dtype
Out[19]: dtype('int32')

In [20]: np.iinfo(np.int32).max, 2**31 - 1
Out[20]: (2147483647, 2147483647)

In [21]: np.iinfo(np.int64).max, 2**63 - 1
Out[21]: (9223372036854775807, 9223372036854775807)
```

Unsigned integers:

uint8	8 bits
uint16	16 bits
uint32	32 bits
uint64	64 bits

```
In [17]: np.iinfo(np.uint32).max, 2**32 - 1
Out[17]: (4294967295, 4294967295)
```

```
In [47]: np.array([1], dtype=float).dtype
Out[47]: dtype('float64')
```

NumPy

Self Study

Getting started with Python for science

3. More elaborate arrays
   1. More data types

Different data type sizes

Floating-point numbers:

float16	16 bits
float32	32 bits
float64	64 bits
float96	96 bits, platform-dependent
float128	128 bits, platform-dependent

Complex floating-point numbers:

complex64	two 32-bit floats
complex128	two 64-bit floats
complex192	two 96-bit floats, platform-dependent
complex256	two 128-bit floats, platform-dependent

```
In [26]: np.finfo(np.float32).eps
Out[26]: 1.1920929e-07

In [27]: np.finfo(np.float64).eps
Out[27]: 2.220446049250313e-16

In [28]: np.float32(1e-8) + np.float32(1) == 1
Out[28]: True

In [29]: np.float64(1e-8) + np.float64(1) == 1
Out[29]: False
```

NumPy

Self Study

Getting started with Python for science

3. More elaborate arrays
   1. More data types

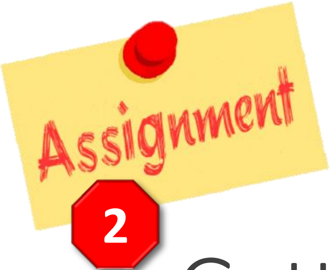
Different data type sizes

Smaller data types

If you don't know you need special data types, then you probably don't.
Comparison on using float32 instead of float64:

- Half the size in memory and on disk
- Half the memory bandwidth required (may be a bit faster in some operations)
- But: bigger rounding errors—sometimes in surprising places (i.e., don't use them unless you really need them)

```
In [33]: a = np.zeros((int(1e6),), dtype=np.float64)

In [34]: int(1e6)
Out[34]: 1000000

In [35]: b = np.zeros((int(1e6),), dtype=np.float32)

In [36]: %timeit a*a
100 loops, best of 3: 3.27 ms per loop

In [37]: %timeit b*b
1000 loops, best of 3: 1.5 ms per loop
```



Getting started with Python for science

1. Create ten arrays by combination of the following functions and techniques.
2. Describe how each is produced.
3. Use each of the following functions and techniques at least once.

Techniques:
Indexing
Slicing
Mask
Broadcasting

Functions:
array
arange
ones
zeros
eye
diag
rand
randn
newaxis

Functions:
randint
triu
tile
ogrid
mgrid
transpose
resize
dtype
astype

NumPy

Self Study

Getting started with Python for science

3. More elaborate arrays
   2. Structured data types

sensor_code	(4-character string)
position	(float)
value	(float)

```
In [41]: samples = np.zeros((6,), dtype=[('sensor_code', 'S4'),
    ...: ... ('position', float), ('value', float)])

In [42]: samples.ndim
Out[42]: 1

In [43]: samples.shape
Out[43]: (6,)

In [44]: samples.dtype.names
Out[44]: ('sensor_code', 'position', 'value')

In [45]: samples[:] = [('ALFA', 1, 0.37), ('BETA', 1, 0.11), ('TAU', 1, 0.13),
    ...: ... ('ALFA', 1.5, 0.37), ('ALFA', 3, 0.11), ('TAU', 1.2, 0.13)]

In [46]: samples
Out[46]:
array([(b'ALFA',  1. ,  0.37), (b'BETA',  1. ,  0.11),
       (b'TAU',  1. ,  0.13), (b'ALFA',  1.5,  0.37),
       (b'ALFA',  3. ,  0.11), (b'TAU',  1.2,  0.13)],
      dtype=[('sensor_code', 'S4'), ('position', '<f8'), ('value', '<f8')])
```

NumPy

Self Study

Getting started with Python for science

3. More elaborate arrays
    2. Structured data types

```
In [38]: samples = np.zeros((6,), dtype=[('sensor_code11', 'S6'),
    ...: ... ('position12', np.float64), ('value5', int), ('value8', np.complex)])

In [39]: samples.dtype.names
Out[39]: ('sensor_code11', 'position12', 'value5', 'value8')

In [40]: samples[:]
Out[40]:
array([(b'',  0., 0,  0.+0.j), (b'',  0., 0,  0.+0.j),
       (b'',  0., 0,  0.+0.j), (b'',  0., 0,  0.+0.j),
       (b'',  0., 0,  0.+0.j), (b'',  0., 0,  0.+0.j)],
      dtype=[('sensor_code11', 'S6'), ('position12', '<f8'), ('value5', '<i4'), ('value8', '<c16')])
```

Getting started with Python for science

3. More elaborate arrays
   2. Structured data types

'f' is the shorthand for 'float32'.

'f4' also means 'float32' because it has 4 bytes and each byte has 8 bits.

Similarly, 'f8' means 'float64' because 8*8 = 64.

For the difference between '>f4' and '`<f4`', it is related to how the 32 bits are stored in 4 bytes.

('>')Big Endian Byte Order: The most significant byte (the "big end") of the data is placed at the byte with the lowest address. The rest of the data is placed in order in the next three bytes in memory.

('<')Little Endian Byte Order: The least significant byte (the "little end") of the data is placed at the byte with the lowest address. The rest of the data is placed in order in the next three bytes in memory.

NumPy

Getting started with Python for science

3. More elaborate arrays
   2. Structured data types

Field access works by indexing with field names:

```
In [48]: samples['sensor_code']
Out[48]:
array([b'ALFA', b'BETA', b'TAU', b'ALFA', b'ALFA', b'TAU'],
      dtype='|S4')

In [49]: samples['value']
Out[49]: array([ 0.37,  0.11,  0.13,  0.37,  0.11,  0.13])

In [50]: samples[0]
Out[50]: (b'ALFA',  1.,  0.37)

In [51]: samples[0]['sensor_code'] = 'TAU'

In [52]: samples[0]
Out[52]: (b'TAU',  1.,  0.37)

In [53]: s11 = b'TAU'

In [54]: type(s11)
Out[54]: bytes

In [55]: s11.decode('utf-8')
Out[55]: 'TAU'
```

NumPy

Self Study

Getting started with Python for science

3. More elaborate arrays
    2. Structured data types

Field access works by indexing with field names:

```
In [56]: samples = np.zeros((6,), dtype=[('sensor_code', 'U4'),
    ...: ... ('position', float), ('value', float)])

In [57]: samples
Out[57]:
array([('',  0.,  0.), ('',  0.,  0.), ('',  0.,  0.), ('',  0.,  0.),
       ('',  0.,  0.), ('',  0.,  0.)],
      dtype=[('sensor_code', '<U4'), ('position', '<f8'), ('value', '<f8')])

In [58]: samples[:] = [('ALFA', 1, 0.37), ('BETA', 1, 0.11), ('TAU', 1, 0.13),
    ...: ... ('ALFA', 1.5, 0.37), ('ALFA', 3, 0.11), ('TAU', 1.2, 0.13)]

In [59]: samples
Out[59]:
array([('ALFA',  1. ,  0.37), ('BETA',  1. ,  0.11), ('TAU',  1. ,  0.13),
       ('ALFA',  1.5,  0.37), ('ALFA',  3. ,  0.11), ('TAU',  1.2,  0.13)],
      dtype=[('sensor_code', '<U4'), ('position', '<f8'), ('value', '<f8')])
```

NumPy

Getting started with Python for science

3. More elaborate arrays
   2. Structured data types

Multiple fields at once:

```
In [60]: samples[['position', 'value']]
Out[60]:
array([( 1. ,  0.37), ( 1. ,  0.11), ( 1. ,  0.13), ( 1.5,  0.37),
       ( 3. ,  0.11), ( 1.2,  0.13)],
      dtype=[('position', '<f8'), ('value', '<f8')])
```

Fancy indexing works, as usual:

```
In [61]: samples[samples['sensor_code'] == 'ALFA']
Out[61]:
array([('ALFA',  1. ,  0.37), ('ALFA',  1.5,  0.37), ('ALFA',  3. ,  0.11)],
      dtype=[('sensor_code', '<U4'), ('position', '<f8'), ('value', '<f8')])
```

NumPy

Getting started with Python for science

4. Advanced operations
   1. Polynomials

Numpy also contains polynomials in different bases:
For example, $3x^2 + 2x - 1$:

```
In [66]: p = np.poly1d([3, 2, -1])

In [67]: p(0)
Out[67]: -1

In [68]: p(2)
Out[68]: 15

In [69]: p.roots
Out[69]: array([-1.        ,  0.33333333])

In [70]: p.order
Out[70]: 2
```

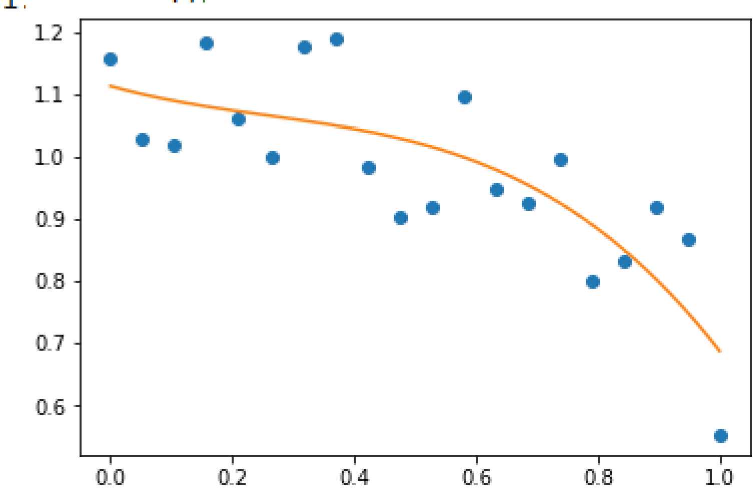
NumPy

Getting started with Python for science

4. Advanced operations

1. Polynomials

```
In [80]: x = np.linspace(0, 1, 20)

In [81]: x
Out[81]:
array([ 0.        ,  0.05263158,  0.10526316,  0.15789474,  0.21052632,
        0.26315789,  0.31578947,  0.36842105,  0.42105263,  0.47368421,
        0.52631579,  0.57894737,  0.63157895,  0.68421053,  0.73684211,
        0.78947368,  0.84210526,  0.89473684,  0.94736842,  1.        ])

In [82]: y = np.cos(x) + 0.3*np.random.rand(20)

In [83]: p = np.poly1d(np.polyfit(x, y, 3))

In [84]: t = np.linspace(0, 1, 200)

In [85]: import matplotlib as plt

In [86]: plt.pyplot.plot(x, y, 'o', t, p(t), '-')
Out[86]:
[<matplotlib.lines.Line2D at 0x4eea84780>,
 <matplotlib.lines.Line2D at 0x4eea84908>]
```



NumPy

Getting started with Python for science

4. Advanced operations
   2. Loading data files

Text files

Example: populations1.txt:

```
# year   hare     lynx     carrot
1900     30e3     4e3 48300
1901     47.2e3   6.1e3    48200
1902     70.2e3   9.8e3    41500
1903     77.4e3   35.2e3   38200
```

```
In [88]: data = np.loadtxt('E:/gpu/presentations/final/data/populations1.txt')

In [89]: data
Out[89]:
array([[  1900.,  30000.,   4000.,  48300.],
       [  1901.,  47200.,   6100.,  48200.],
       [  1902.,  70200.,   9800.,  41500.],
       [  1903.,  77400.,  35200.,  38200.]])

In [90]: np.savetxt('E:/gpu/presentations/final/data/pop2.txt', data)

In [91]: data2 = np.loadtxt('E:/gpu/presentations/final/data/pop2.txt')

In [92]: data2
Out[92]:
array([[  1900.,  30000.,   4000.,  48300.],
       [  1901.,  47200.,   6100.,  48200.],
       [  1902.,  70200.,   9800.,  41500.],
       [  1903.,  77400.,  35200.,  38200.]])
```

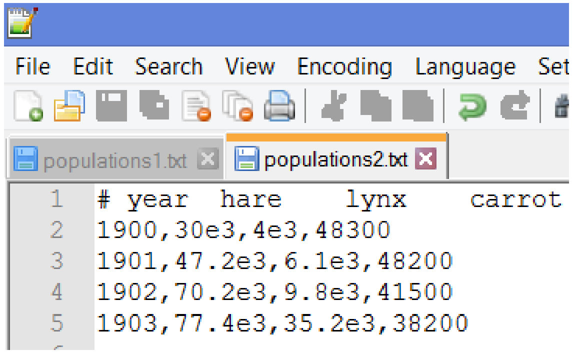
NumPy

Getting started with Python for science

4. Advanced operations
    2. Loading data files

Text files

Example: populations2.txt:




```
In [93]: data = np.loadtxt('E:/gpu/presentations/final/data/populations2.txt')
Traceback (most recent call last):

  File "<ipython-input-93-14d012c1824a>", line 1, in <module>
    data = np.loadtxt('E:/gpu/presentations/final/data/populations2.txt')

  File "C:\Program Files\Anaconda3\lib\site-packages\numpy\lib\npyio.py", line 1024, in loadtxt
    items = [conv(val) for (conv, val) in zip(converters, vals)]

  File "C:\Program Files\Anaconda3\lib\site-packages\numpy\lib\npyio.py", line 1024, in <listcomp>
    items = [conv(val) for (conv, val) in zip(converters, vals)]

  File "C:\Program Files\Anaconda3\lib\site-packages\numpy\lib\npyio.py", line 725, in floatconv
    return float(x)

ValueError: could not convert string to float: b'1900,30e3,4e3,48300'
```

NumPy

Getting started with Python for science

4. Advanced operations
   2. Loading data files

Reminder: Navigating the filesystem with IPython

```
In [104]: pwd # show current directory
Out[104]: 'C:\\Users\\MSN'

In [105]: cd E:\gpu\presentations\final\data
E:\gpu\presentations\final\data

In [106]: ls
 Volume in drive E has no label.
 Volume Serial Number is B4F8-0683

 Directory of E:\gpu\presentations\final\data

10/05/2021  10:51 AM    <DIR>          .
10/05/2021  10:51 AM    <DIR>          ..
10/05/2021  10:51 AM               400 pop2.txt
09/26/2021  04:48 PM               549 populations.txt
10/05/2021  10:48 AM               124 populations1.txt
10/05/2021  10:52 AM               124 populations2.txt
               4 File(s)          1,197 bytes
               2 Dir(s)  81,653,284,864 bytes free
```

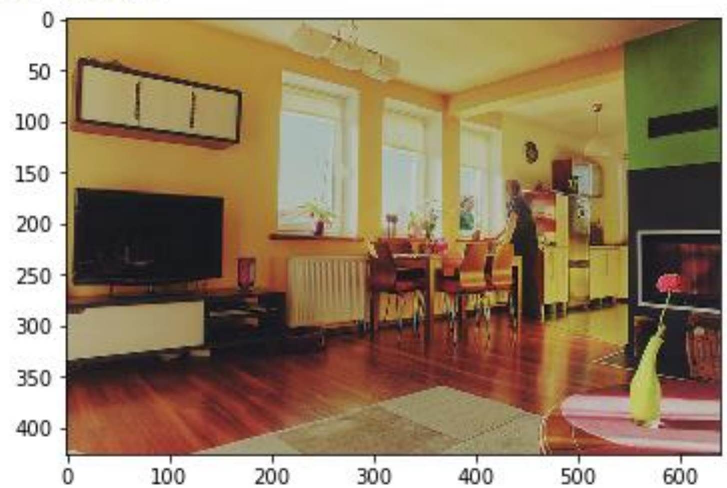
Getting started with Python for science

4. Advanced operations
    2. Loading data files

Images

Using Matplotlib:

```
In [113]: img = plt.pyplot.imread('images/img1.jpg')

In [114]: img.shape, img.dtype
Out[114]: ((426, 640, 3), dtype('uint8'))

In [115]: plt.pyplot.imshow(img)
Out[115]: <matplotlib.image.AxesImage at 0x4eeb9f828>

In [116]:
```



NumPy

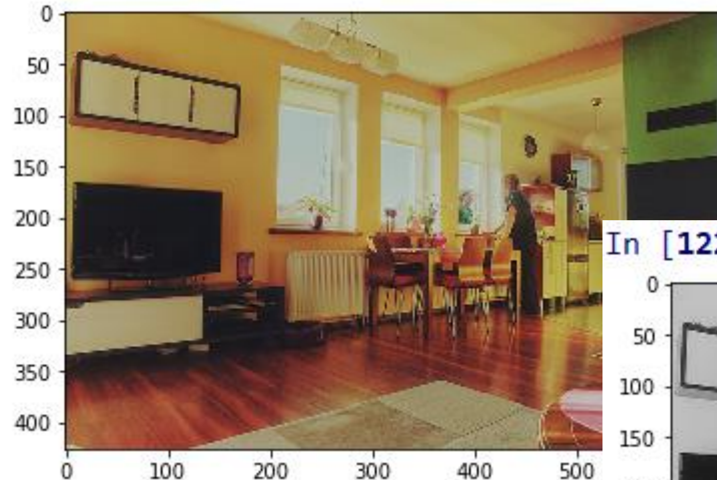
Getting started with Python for science

4. Advanced operations
   2. Loading data files

Images

Using Matplotlib:

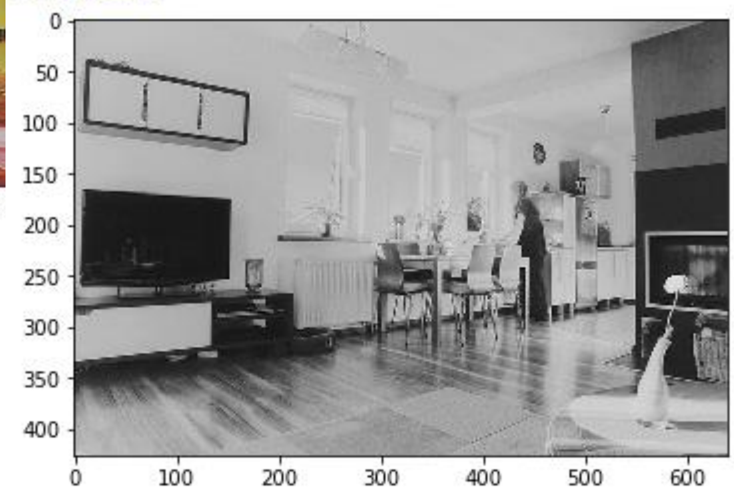
```
In [120]: plt.pyplot.imshow(img);plt.pyplot.savefig('plot.png');\
     ...: plt.pyplot.imsave('red_img1.png', img[:,:,0], cmap=plt.cm.gray)
```



This saved only one channel (of RGB):

```
In [121]: plt.pyplot.imshow(plt.pyplot.imread('red_img1.png'))
Out[121]: <matplotlib.image.AxesImage at 0x4f0844860>
```

```
In [122]:
```



Getting started with Python for science

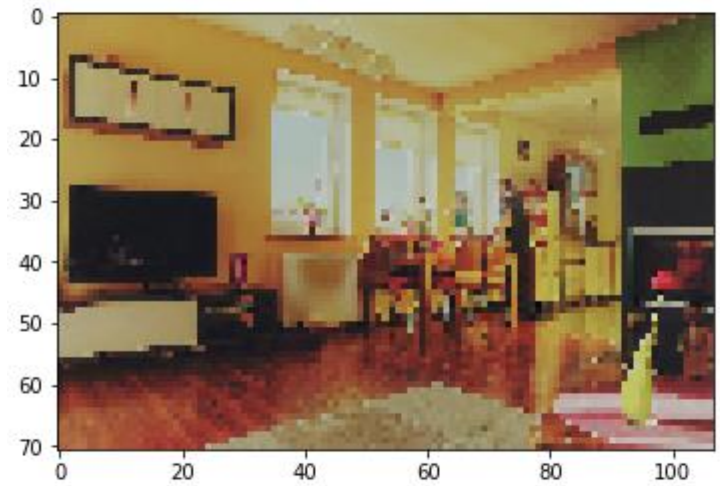
4. Advanced operations
   2. Loading data files

Images

Other libraries:

```
In [128]: from scipy.misc import imsave

In [129]: imsave('tiny_img1.png', img[::6,::6])

In [130]: plt.pyplot.imshow(plt.pyplot.imread('tiny_img1.png'), interpolation='nearest')
Out[130]: <matplotlib.image.AxesImage at 0x4f0436b70>
```

```
In [131]: img_t = img[::6,::6]

In [132]: img_t.shape
Out[132]: (71, 107, 3)

In [133]: img.shape
Out[133]: (426, 640, 3)
```



NumPy

Getting started with Python for science

4. Advanced operations
   2. Loading data files

Numpy's own format

Numpy has its own binary format, not portable but with efficient I/O:

```
In [134]: data = np.ones((3, 3))

In [135]: np.save('pop.npy', data)

In [136]: data3 = np.load('pop.npy')

In [137]: data3
Out[137]:
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
```

NumPy

Getting started with Python for science

4. Advanced operations
   2. Loading data files

Well-known (& more obscure) file formats

- HDF5: h5py, PyTables
- NetCDF: `scipy.io.netcdf_file`, netcdf4-python, ...
- Matlab: `scipy.io.loadmat`, `scipy.io.savemat`
- MatrixMarket: `scipy.io.mmread`, `scipy.io.mmwrite`
- IDL: `scipy.io.readsav`

1. HDF5:

An HDF5 file is a container for two kinds of objects: **datasets**, which are array-like collections of data, and **groups**, which are folder-like containers that hold datasets and other groups. The most fundamental thing to remember when using h5py is:

✓ **Groups** work like dictionaries, and **datasets** work like NumPy arrays.

NumPy

Getting started with Python for science

4. Advanced operations
   2. Loading data files

Well-known (& more obscure) file formats

HDF5: Datasets

```
In [14]: import numpy as np

In [15]: import h5py

In [16]: d1 = np.random.random(size = (1000,20))
    ...: d2 = np.random.random(size = (1000,200))
    ...:

In [17]: hf = h5py.File('E:/gpu/presentations/final/data/data1.h5', 'w')

In [18]: hf.create_dataset('dataset_1', data=d1)
    ...: hf.create_dataset('dataset_2', data=d2)
    ...:
Out[18]: <HDF5 dataset "dataset_2": shape (1000, 200), type "<f8">

In [19]: hf.close()
```

NumPy

Getting started with Python for science

4. Advanced operations
   2. Loading data files

Well-known (& more obscure) file formats

HDF5: Datasets

```
In [38]: hf = h5py.File('E:/gpu/presentations/final/data/data1.h5', 'r')

In [39]: k1 = list(hf.keys())

In [40]: k1
Out[40]: ['dataset_1', 'dataset_2']

In [41]: k1 = list(hf.values())

In [42]: k1
Out[42]:
[<HDF5 dataset "dataset_1": shape (1000, 20), type "<f8">,
 <HDF5 dataset "dataset_2": shape (1000, 200), type "<f8">]

In [43]: k1 = list(hf.items())

In [44]: k1
Out[44]:
[('dataset_1', <HDF5 dataset "dataset_1": shape (1000, 20), type "<f8">),
 ('dataset_2', <HDF5 dataset "dataset_2": shape (1000, 200), type "<f8">)]
```

```
In [45]: for name in hf:
    ...: ...     print(name)
    ...:
dataset_1
dataset_2

In [46]: n1 = hf.get('dataset_1')

In [47]: type(n1)
Out[47]: h5py._hl.dataset.Dataset

In [48]: n1 = np.array(n1)

In [49]: n1.shape
Out[49]: (1000, 20)

In [50]: hf.close()
```

NumPy

Getting started with Python for science

4. Advanced operations
   2. Loading data files

Well-known (& more obscure) file formats

HDF5: Groups

Groups are the basic container mechanism in a HDF5 file, allowing hierarchical organisation of the data. **Groups** are created similarly to **datasets**, and **datasets** are then added using the group object.

NumPy

Getting started with Python for science

4. Advanced operations
   2. Loading data files

Well-known (& more obscure) file formats

HDF5: Groups

```
In [1]: import numpy as np

In [2]: import h5py

In [3]: d1 = np.random.random(size = (100,33))
   ...: d2 = np.random.random(size = (100,333))
   ...: d3 = np.random.random(size = (100,3333))
   ...:

In [4]: hf = h5py.File('data.h5', 'w')

In [5]: hf = h5py.File('E:/gpu/presentations/final/data/data2.h5', 'w')

In [6]: g1 = hf.create_group('group1')

In [7]: g1.create_dataset('data1',data=d1)
   ...: g1.create_dataset('data2',data=d1)
   ...:
Out[7]: <HDF5 dataset "data2": shape (100, 33), type "<f8">
```

NumPy

Getting started with Python for science

4. Advanced operations
   2. Loading data files

Well-known (& more obscure) file formats

HDF5: Groups

We can also create subfolders. Just specify the group name as a directory format.

```
In [8]: g2 = hf.create_group('group2/subfolder')

In [9]: g2.create_dataset('data3',data=d3)
Out[9]: <HDF5 dataset "data3": shape (100, 3333), type "<f8">

In [10]: group2 = hf.get('group2/subfolder')

In [11]: k1 = list(group2.items())

In [12]: k1
Out[12]: [('data3', <HDF5 dataset "data3": shape (100, 3333), type "<f8">)]

In [13]: group1 = hf.get('group1')
```

NumPy

Getting started with Python for science

4. Advanced operations
   2. Loading data files

Well-known (& more obscure) file formats

HDF5: Groups

```
In [14]: list(group1.items())
Out[14]:
[('data1', <HDF5 dataset "data1": shape (100, 33), type "<f8">),
 ('data2', <HDF5 dataset "data2": shape (100, 33), type "<f8">)]

In [15]: n1 = group1.get('data1')

In [16]: np.array(n1).shape
Out[16]: (100, 33)

In [17]: hf.close()
```

NumPy

Getting started with Python for science

4. Advanced operations
    2. Loading data files

Well-known (& more obscure) file formats

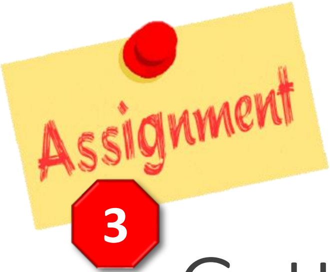
HDF5: Groups

```python
hf = h5py.File('E:/gpu/presentations/final/data/data4.h5', 'w')
g1 = hf.create_group('group1')
g1.create_dataset('data1',data=d1)
g1.create_dataset('data2',data=d2)
g2 = hf.create_group('group2/subfolder')
g2.create_dataset('data3',data=d3)
g2 = hf.create_group('group2/subfolder1')
g2.create_dataset('data4',data=d3)
group2 = hf.get('group2')
list(group2.items())
```

```
In [9]: list(hf.items())
Out[9]:
[('group1', <HDF5 group "/group1" (2 members)>),
 ('group2', <HDF5 group "/group2" (2 members)>)]
```

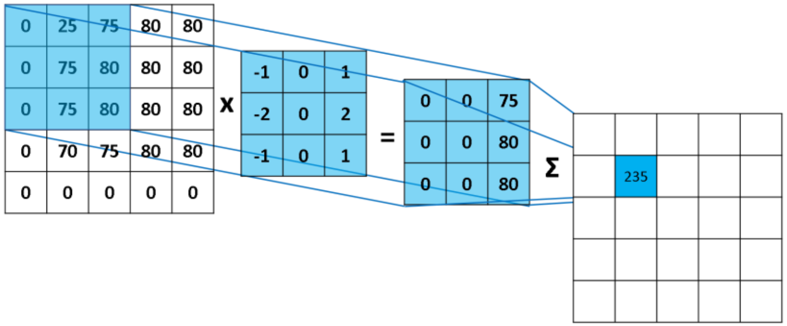
```
In [34]: list(group2.items())
Out[34]:
[('subfolder', <HDF5 group "/group2/subfolder" (1 members)>),
 ('subfolder1', <HDF5 group "/group2/subfolder1" (1 members)>)]
```

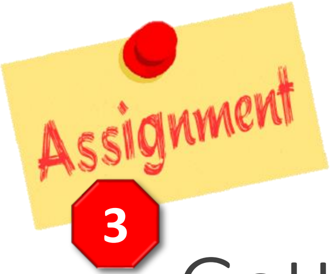
```
In [10]: group1 = hf.get('group1')

In [11]: list(group1.items())
Out[11]:
[('data1', <HDF5 dataset "data1": shape (1000, 20), type "<f8">),
 ('data2', <HDF5 dataset "data2": shape (1000, 200), type "<f8">)]
```



Getting started with Python for science

1. Organize the address of the images in the './data/images' folder into a list.
2. Create three 3*3 Gaussian filters using random Gaussian values for three channels (RGB) of image.
3. Perform the convolution operation as shown in the figure on each image channel with correspond filter and save the results in another array with the appropriate size. Use 'step=1' in moving the filter on the image. If needed, use the zero-padding technique for creating an image margin to apply the filter to all pixels.





Getting started with Python for science

4. Create a new folder called 'Results' and save the filtered image in form of a '.npy' file in the folder.
5. Display original and the resulting image, also save the result in form of a '.png' file.
6. Do this process for all images.

Getting started with Python for science

1. Load the 'vid0_incp_v3.npy, vid1_incp_v3.npy, vid2_incp_v3.npy' files from 'data/video' address that contain the high-level features of three videos with (26, 8, 8, 2048) dimensions.
2. Reshape the features to the (26, 64, 2048) dimensions.
3. Load the 'vid0_map.npy, vid1_map.npy, vid2_map.npy' files from 'data/video' address that contain the three maps correspond to three videos with (26, 64) dimensions.
4. Convert loaded map to a binary map with 'thresholds = 0.4, 0.6, 0.8'.
5. Display the original map and the resulting maps and save them in the '.png' files.
6. Apply the binary map to the video features (for all the values in the last dimension of the features there is a value in the map that must be multiplied).

Getting started with Python for science

7. Reshape the video features to (26, 8, 8, 2048) dimensions.
8. Perform this process for all videos with all corresponding binary maps and save the results in an organized and hierarchical HDF5 file.

Getting started with Python for science

1. Load 'feat.txt' file from 'data/video' address that contain the high-level features of four videos.
2. There are some frames for each video. Split features of each frame and convert them to the appropriate type of data.
3. Organize features of each frame for all videos using array and dictionary containers so that specific frame features of the desired video can be accessed.
4. Save them in a '.pkl' file.
5. Load '.pkl' file and create 'feat.txt' file.