Test time:

OCT 12 WLB 103 15:30 to 17:00

COMP7035

Python for Data Analytics and Artificial Intelligence

Numpy, Matplotlib, Seaborn

Renjie Wan, Xue Wei

13/10/2022

香港浸會大學
HONG KONG BAPTIST UNIVERSITY

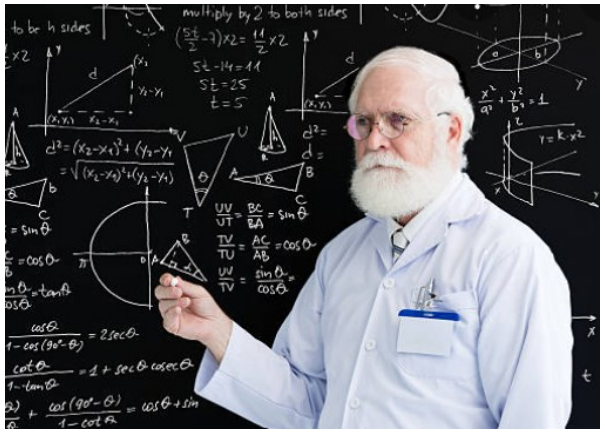
DEPARTMENT OF
COMPUTER SCIENCE
計算機科學系

What we will learn?

Topic		Hours
I.	Python Fundamentals A. Program control and logic B. Data types and structures C. Function D. File I/O	12
II.	Numerical Computing and Data Visualization Tools and libraries such as **A. NumPy** B. Matplotlib C. Seaborn	9
III.	Exploratory Data Analysis (EDA) with Python Tools and libraries such as A. Pandas B. Sweetviz	9
IV.	Artificial Intelligence and Machine Learning with Python Tools and libraries such as A. Keras B. Scikit-learn	9

Platforms for scientific computing

- Matlab
- Python based platforms
  - Numpy
  - Scipy
  - Matplotlib



Matlab

Python based platforms

What is Numpy?

- Numpy, Scipy, and Matplotlib provide MATLAB-like functionality in python.
- Numpy Features:
  - Multidimentional arrays (matrices)
  - Fast numerical computations (matrix math)
    - Additional linear algebra, Fourier transform, and random number capabilities
  - High-level math functions

Why do we need NumPy

- Python does numerical computations slowly.
- 1000×1000 matrix multiply
  - Python triple loop takes > 10 min.
  - Numpy takes ~0.03 seconds
- Arrays are very frequently used in data science, where speed and resources are very important.

$a = (1,2,3,4,5,6)$ $b = (3,4,5,6,7,6)$

multiplying each element in a 1-D sequence with the corresponding element in another sequence of the same length

```
a = [1, 2, 3, 4, 5, 6]
b = [3, 4, 5, 6, 7, 6]
c = []
for t in range(len(a)):
  c.append(a[t]*b[t])
print(c)
```

```
a = np.array([1, 2, 3, 4, 5, 6])
b = np.array([3, 4, 5, 6, 7, 6])
c = a*b
print(c)
```

How to use Numpy

- Just import it!

```
import numpy as np
```

Arrays

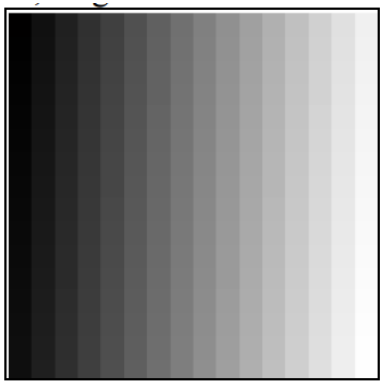
- Structured lists of numbers
  - Vectors
  - Matrics
  - Images

$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$$

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$$

Arrays

- Structured lists of numbers
  - Vectors
  - Matrics
  - Images
- What is the relationship between images and arrays?
  - Images can be regarded as a special type of matrics



0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
1	17	33	49	65	81	97	113	129	145	161	177	193	209	225	241
2	18	34	50	66	82	98	114	130	146	162	178	194	210	226	242
3	19	35	51	67	83	99	115	131	147	163	179	195	211	227	243
4	20	36	52	68	84	100	116	132	148	164	180	196	212	228	244
5	21	37	53	69	85	101	117	133	149	165	181	197	213	229	245
6	22	38	54	70	86	102	118	134	150	166	182	198	214	230	246
7	23	39	55	71	87	103	119	135	151	167	183	199	215	231	247
8	24	40	56	72	88	104	120	136	152	168	184	200	216	232	248
9	25	41	57	73	89	105	121	137	153	169	185	201	217	233	249
10	26	42	58	74	90	106	122	138	154	170	186	202	218	234	250
11	27	43	59	75	91	107	123	139	155	171	187	203	219	235	251
12	28	44	60	76	92	108	124	140	156	172	188	204	220	236	252
13	29	45	61	77	93	109	125	141	157	173	189	205	221	237	253
14	30	46	62	78	94	110	126	142	158	174	190	206	222	238	254
15	31	47	63	79	95	111	127	143	159	175	191	207	223	239	255

A matrix and its image

0	50	100	150	200	250	300	350
50	100	150	200	250	300	350	400
100	150	200	250	300	350	400	450
150	200	250	300	350	400	450	500
200	250	300	350	400	450	500	550
250	300	350	400	450	500	550	600
300	350	400	450	500	550	600	650
350	400	450	500	550	600	650	700

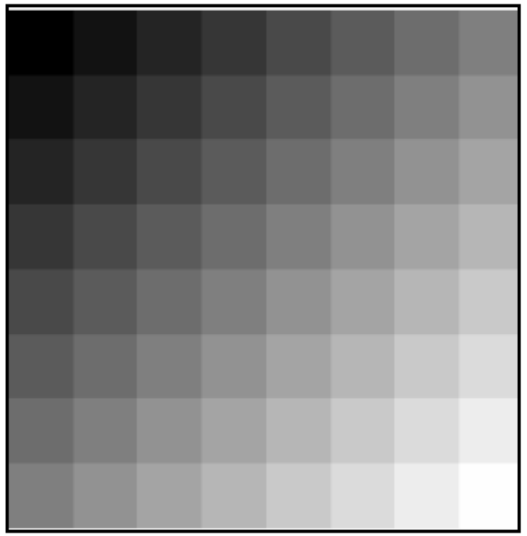
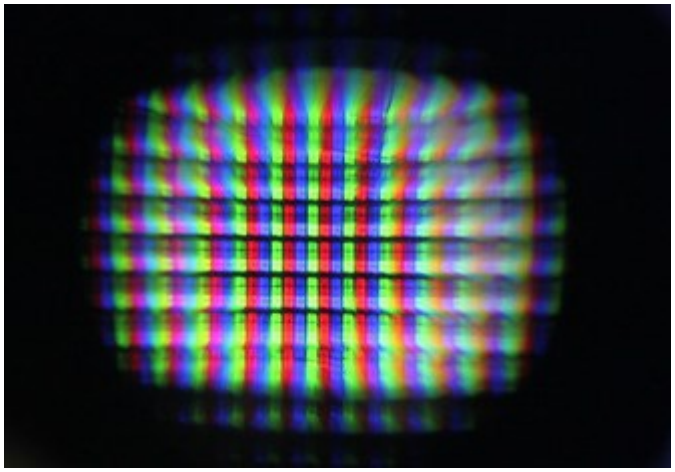


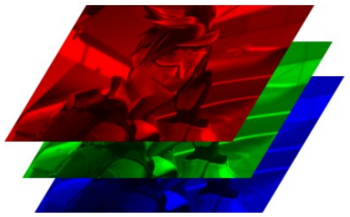
Image arrays

Images are 3D arrays: width, height, and channels

Common image formats:

- height × width × RGB (band-interleaved)
- height × width (band-sequential)





Basic properties of Arrays

- Arrays can have any number of dimensions, including zero (a scalar).
- Arrays are typed: np.uint8, np.int64, np.float32, np.float64
- Arrays are dense. Each element of the array exists and has the same type.

```
#E1
a = np.array([[1,2,3],[4,5,6]],dtype=np.float32)
print(a.ndim, a.shape, a.dtype)
```

```
2 (2, 3) float32
```

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

How to create an array?

- Create a list first.
- Then, please use np.array(list)

```
#E2
python_list = [1,2,3]
np.array(python_list)
```

- Remember how we create a list before?
- We can also create a array using a similar way.

```
#E4
arr = np.array([ 2**i for i in [2,3,9] ])
arr
```

```
array([  4,   8, 512])
```

$(4 \quad 8 \quad 512)$

How to create an array?

- Many easy ways can be used to create an array
- We can also create a array using a similar way.
- `np.zeros` returns a new array of given shape and type, filled with zeros.
- `np.ones` returns a new array of given shape and type, filled with one.

```
#E6 arr = np.zeros(5)
#E7 arr = np.ones(5)
```

The creation of array

- np.ones, np.zeros

```
#E8 np.ones((3, 5), dtype=np.float32)
```

```
>>> np.ones((3,5),dtype=np.float32)
array([[ 1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.]], dtype=float32)
```

```
#E9 np.zeros((6, 2), dtype=np.int8)
```

```
>>> np.zeros((6,2),dtype=np.int8)
array([[0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0]], dtype=int8)
```

The creation of array

- np.ones, np.zeros
- **np.arange**
  - Return evenly spaced values within a given interval
- np.concatenate
- np.zeros_like, np.ones_like
- np.random.random

#E10

```
>>> np.arange(1334,1338)
array([1334, 1335, 1336, 1337])
```

The creation of array

- np.ones, np.zeros
- np.arange
- **np.concatenate**
  - Join a sequence of arrays along an existing axis.
- np.zeros_like, np.ones_like
- np.random.random

You can understand the axis as the axis that will change after the concatenation.

```
#E11
A = np.ones((2,3))
B = np.zeros((3,3))
np.concatenate([A,B],axis=0)
```

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

The creation of array

- np.ones, np.zeros
- np.arange
- **np.concatenate**
- np.zeros_like, np.ones_like
- np.random.random

```
#E11
>>> A = np.ones((4,1))
>>> B = np.zeros((4,2))
>>> np.concatenate([A,B], axis=1)
array([[ 1.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  0.,  0.]])
```

The creation of array

- np.ones, np.zeros
- np.arange
- np.concatenate
- **np.zeros_like, np.ones_like**
  - Return an array of zeros with the same shape and type as a given array.
  - Return an array of ones with the same shape and type as a given array.
- np.random.random

```
#E12
>>> a = np.ones((2,2,3))
>>> b = np.zeros_like(a)
>>> print(b.shape)
```

The creation of array

- np.ones, np.zeros
- np.arange
- np.concatenate
- np.astype
- np.zeros_like, np.ones_like
- **np.random.random**
  - Return random floats in the half-open interval [0.0, 1.0). Alias for random_sample to ease forward-porting to the new random API.

#E13

```
>>> np.random.random((10,3))
array([[ 0.61481644,  0.55453657,  0.04320502],
       [ 0.08973085,  0.25959573,  0.27566721],
       [ 0.84375899,  0.2949532 ,  0.29712833],
       [ 0.44564992,  0.37728361,  0.29471536],
       [ 0.71256698,  0.53193976,  0.63061914],
       [ 0.03738061,  0.96497761,  0.01481647],
       [ 0.09924332,  0.73128868,  0.22521644],
       [ 0.94249399,  0.72355378,  0.94034095],
       [ 0.35742243,  0.91085299,  0.15669063],
       [ 0.54259617,  0.85891392,  0.77224443]]
```

The creation of array

- Must be dense, no holes.
- Must be one type
- Cannot combine arrays of different shape

```
#E14
>>> np.ones([7,8]) + np.ones([9,3])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: operands could not be broadcast together
 with shapes (7,8) (9,3)
```

Shaping

- Total number of elements cannot change.
- Use -1 to infer axis shape, numpy allow us to give one of new shape parameter as -1

```
#E15
a = np.array([1, 2, 3, 4, 5, 6])
print("a:", a)
print("the shape of a:", a.shape)
b = a.reshape(3,2)
print("b:", b)
print("the shape of b:", b.shape)
c = a.reshape(2,-1)
print("c:", c)
print("the shape of c:", c.shape)
```

Transposition

- np.transpose permutes axes.
- a.T transposes the first two axes. “a is a matrix to be transposed in this place”

```
#E16
a = np.array([[1.,2.],[3.,4.]])
print("a:", a)
b = a.T
print("b:", b)
```

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$$

Array sorting

- Sorting is to arrange the elements of an array in hierarchical order either ascending or descending. By default, numpy does sorting in ascending order.

```
#E19
array4 = np.array([1,0,2,-3,6,8,4,7])
array4.sort()
print(array4)
```

```
[-3 0 1 2 4 6 7 8]
```

```
#E20
array4 = np.array([[10,-7,0, 20],[-5,1,200,40],[30,1,-1,4]])
print("original array \n", array4)
array4.sort()
print("After sorting \n", array4)
```

```
original array
 [[ 10  -7   0  20]
 [ -5   1 200  40]
 [ 30   1  -1   4]]
After sorting
 [[ -7   0  10  20]
 [ -5   1  40 200]
 [ -1   1   4  30]]
```

$$\begin{matrix} 10 & -7 & 0 & 20 \\ -5 & 1 & 200 & 40 \\ 30 & 1 & -1 & 4 \end{matrix} \xrightarrow{\text{sorting}} \begin{matrix} -7 & 0 & 10 & 20 \\ -5 & 1 & 40 & 200 \\ -1 & 1 & 4 & 30 \end{matrix}$$

Statistical operations

- **Arithmetic operations are element-wise**
- Logical operator return a bool array
- In place operations modify the array

```
#E17
a = np.array([1, 2, 3])
b = np.array([4, 4, 10])
a*b
```

```
array([ 4, 8, 30])
```

Statistical operations

- Arithmetic operations are element-wise
- **Logical operator return a bool array**
- In place operations modify the array

```
#E18
a = np.random.random((5, 3))
print("a:", a)
c = a>0.5
print("c:", c)
```

```
a: [[0.1217121  0.97908648 0.8537458 ]
 [0.53775343 0.7860607  0.88921186]
 [0.853963   0.25478302 0.15270884]
 [0.18679235 0.83077973 0.24887868]
 [0.29220583 0.43745045 0.91972215]]
c: [[False  True  True]
 [ True  True  True]
 [ True False False]
 [False  True False]
 [False False  True]]
```

Statistical operations

- Arithmetic operations are element-wise
- Logical operator return a bool array
- **In place operations modify the array**

```
>>> a
array([[ 4, 15],
       [20, 75]])
>>> b
array([[ 2,  5],
       [ 5, 15]])
>>> a /= b
>>> a
array([[2, 3],
       [4, 5]])
```

Statistical operations

- sqrt(), max(), min(), sum(), mean(), std()
  - sqrt() non-negative square-root of an array, element-wise.
  - max() finds the maximum element from an array
  - min() finds the minimum element from an array
  - mean() finds the average of elements of the array
  - std() finds the standard deviation of an array of elements

```
#E22
a = np.array([[1, 4], [9, 16], [25, 36]])
b = np.sqrt(a)
print(b)                    [[1. 2.] [3. 4.] [5. 6.]]
```

```
#E23
arrayA = np.array([1,0,2,-3,6,8,4,7])
arrayB = np.array([[3,6],[4,2]])
print(arrayA.max())      8
print(arrayB.max())      6
```

```
#E24
arrayA = np.array([1,0,2,-3,6,8,4,7])
arrayB = np.array([[3,6],[4,2]])
print(arrayA.min())      -3
print(arrayB.min())      2
```

```
#E25
arrayA = np.array([1,0,2,-3,6,8,4,7])
arrayB = np.array([[3,6],[4,2]])
print(arrayA.mean())    3.125
print(arrayB.mean())    3.75
```

```
#E26
arrayA = np.array([1,0,2,-3,6,8,4,7])
arrayB = np.array([[3,6],[4,2]])
print(arrayA.std())  3.550968177835448
print(arrayB.std())  1.479019945774904
```

Statistical operations

- You can find more interesting and important numpy functions from the link below:

  https://numpy.org/doc/stable/reference/routines.math.html

Array spliting

- numpy.split(ary, indices_or_sections, axis=0) splits an array along the specified axis.
  - If indices_or_sections is an integer, N, the array will be divided into N equal arrays along axis. If such a split is not possible, an error is raised.

```
#E21
array4 = np.array([[10,-7,0,20],[-5,1,200,40],[30,1,-1,4],[1,2,1,0],[0,1,0,2],[0,1,0,2]])
first, second = np.split(array4, 2,axis=0)
print("array4:\n")
print(array4)
print("first:\n", first)
print("second:\n", second)
```

```
array4:

[[ 10  -7   0  20]
 [ -5   1 200  40]
 [ 30   1  -1   4]
 [  1   2   1   0]
 [  0   1   0   2]
 [  0   1   0   2]]
first:
 [[ 10  -7   0  20]
 [ -5   1 200  40]
 [ 30   1  -1   4]]
second:
 [[1 2 1 0]
 [0 1 0 2]
 [0 1 0 2]]
```

Array Indexing

- How to access an element in an array
  - The position for the first element in an array is 0 not 1
  - The position for the last element in an array is -1 or length-1

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

```
arr[5], arr[-3]
```

```
(5, 7)
```

```
arr[10-1], arr[-1]
```

```
(9, 9)
```

```
arr[3:7]
```

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

```
arr[2:]
```

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

```
arr[0:-2]
```

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

```
arr[0:6:2] #same as range(0,6,2)
```

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

Numpy for Image

- You need PIL to read image

```
from PIL import Image
import numpy as np

im = np.array(Image.open('data/src/lena.jpg'))
print(type(im))
# <class 'numpy.ndarray'>
print(im.dtype)
# uint8
print(im.shape)
# (225, 400, 3)
```