专业技能实训

授课教师：王洪松

邮箱：hongsongwang@seu.edu.cn

专业技能实训

- 时间：2023.08.21 – 2023.09.15，共4周
- 地点：教四-103，QQ群：126701099

	周一	周三	周五
第一周（8.8-8.14）	6-9节， 教四-103	1-5节，教四-103	2-5节，教四-103
第二周（8.15-8.21）	6-9节，教四-103	1-5节，教四-103	2-5节，教四-103
第三周（8.22-8.28）	6-9节，教四-103	1-5节，教四-103	2-5节，教四-103
第四周（8.29-9.4）	6-9节，教四-103	1-5节，教四-103	2-5节，教四-103

专业技能实训

- 题目：**基于OpenCV的图像视频分析工具**
- 要求
  - 项目代码5-6人分工协作完成
  - 实现多个图像分析功能，比如区域打码、图像分割等
  - 实现多个视频分析功能，比如视频压缩、目标跟踪等
  - 制作视频图像分析的GUI工具

专业技能实训

- 评分标准
  - 平时考勤占10%
  - 小组成绩占50%
    - 第四周周五集中展示，小组互评
    - 去掉一个最高分和最低分，取平均值
  - 个人成绩占40%
    - 个人实验报告
    - 代码提交

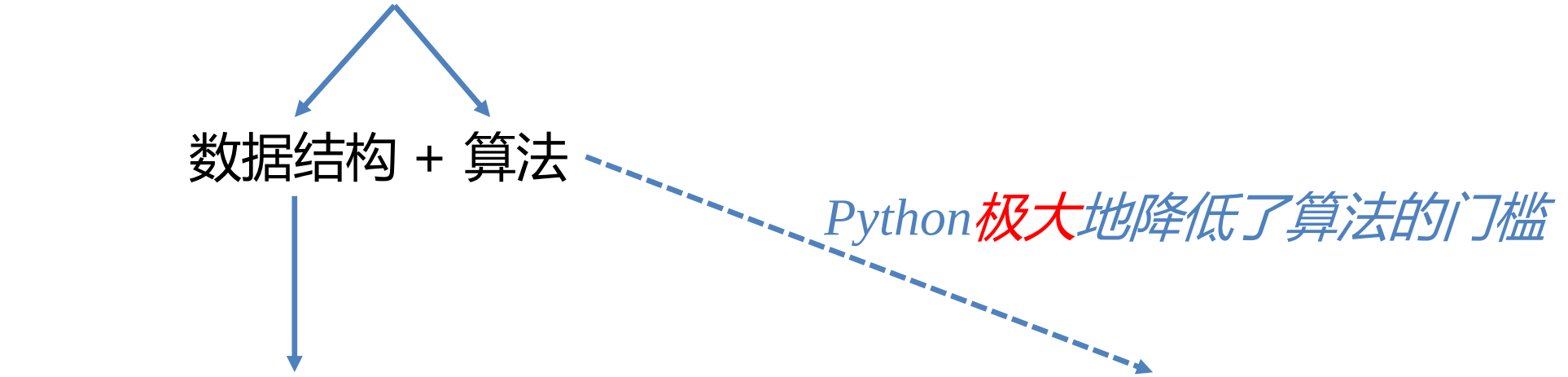
专业技能实训

- 授课内容
  - 第一周：Python及OpenCV基础
  - 第二周：OpenCV与图像处理
  - 第三周：OpenCV与视频处理
  - 第四周：系统集成与展示

Python环境配置

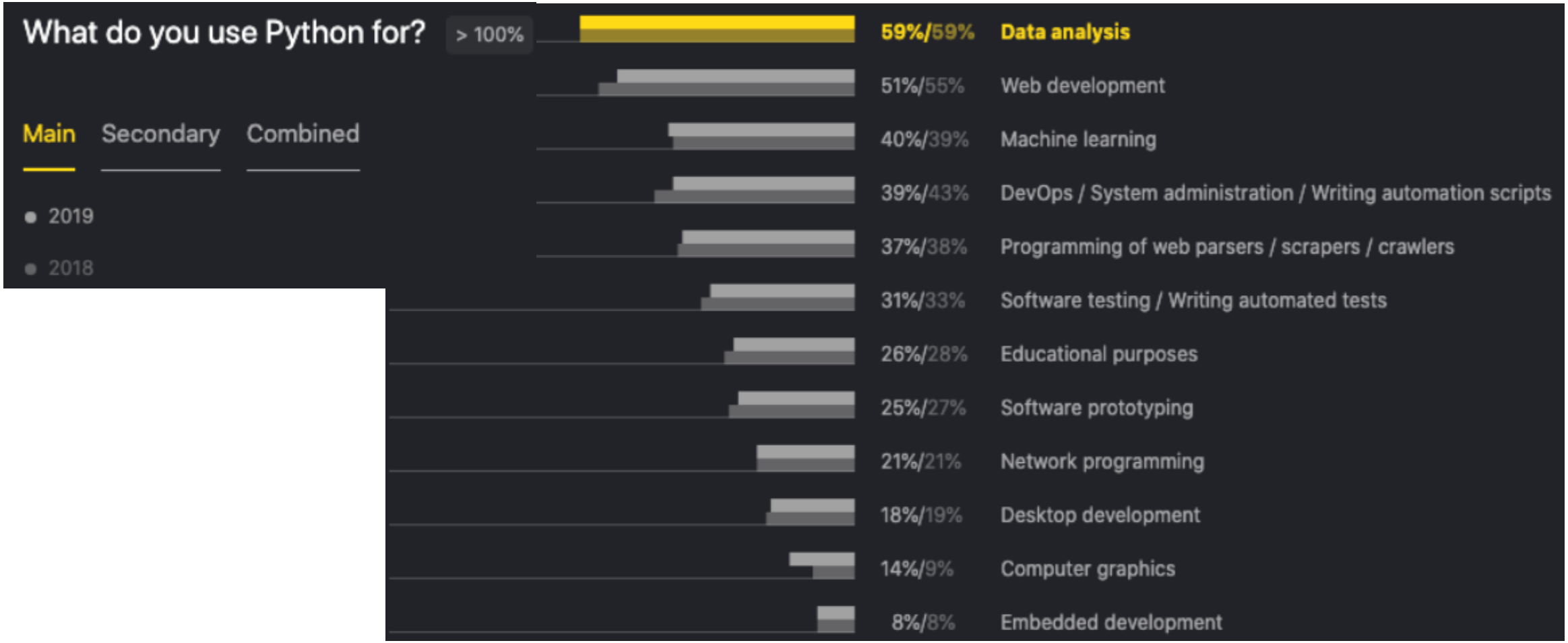
Python编程：1.基础知识

- 编程已经是我们生活中的重要组成元素
- 程序开发产生软件，那什么是软件？
  - 软件 = 程序 + 数据 + 文档



Python编程：基础知识

- 程序员都用Python做什么?



Python编程：基础知识

- python -m freegames.snake

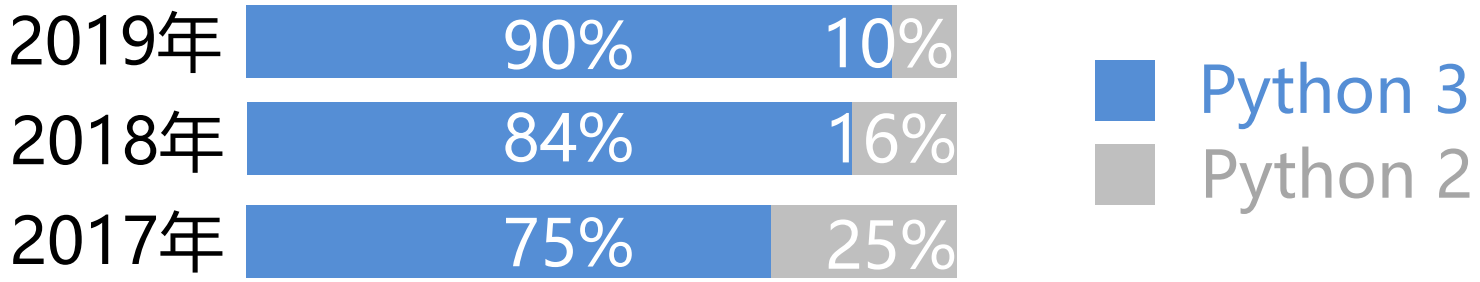
贪食蛇游戏

贪食蛇游戏仅用45行有效代码！

```
from turtle import *
from random import randrange
from freegames import square, vector
food = vector(0, 0)
snake = [vector(10, 0)]
aim = vector(0, -10)
def change(x, y):
    "Change snake direction."
    aim.x = x
    aim.y = y
def inside(head):
    "Return True if head inside boundaries."
    return -200 < head.x < 190 and -200 < head.y < 190
def move():
    "Move snake forward one segment."
    head = snake[-1].copy()
    head.move(aim)
```

Python编程：基础知识

- Python版本选择



- Python开发环境选择
  - PyCharm
  - VS Code
  - Vim
  - Spyder
  - Python IDLE
  - Jupyter Notebook
  - Sublime text
  - …

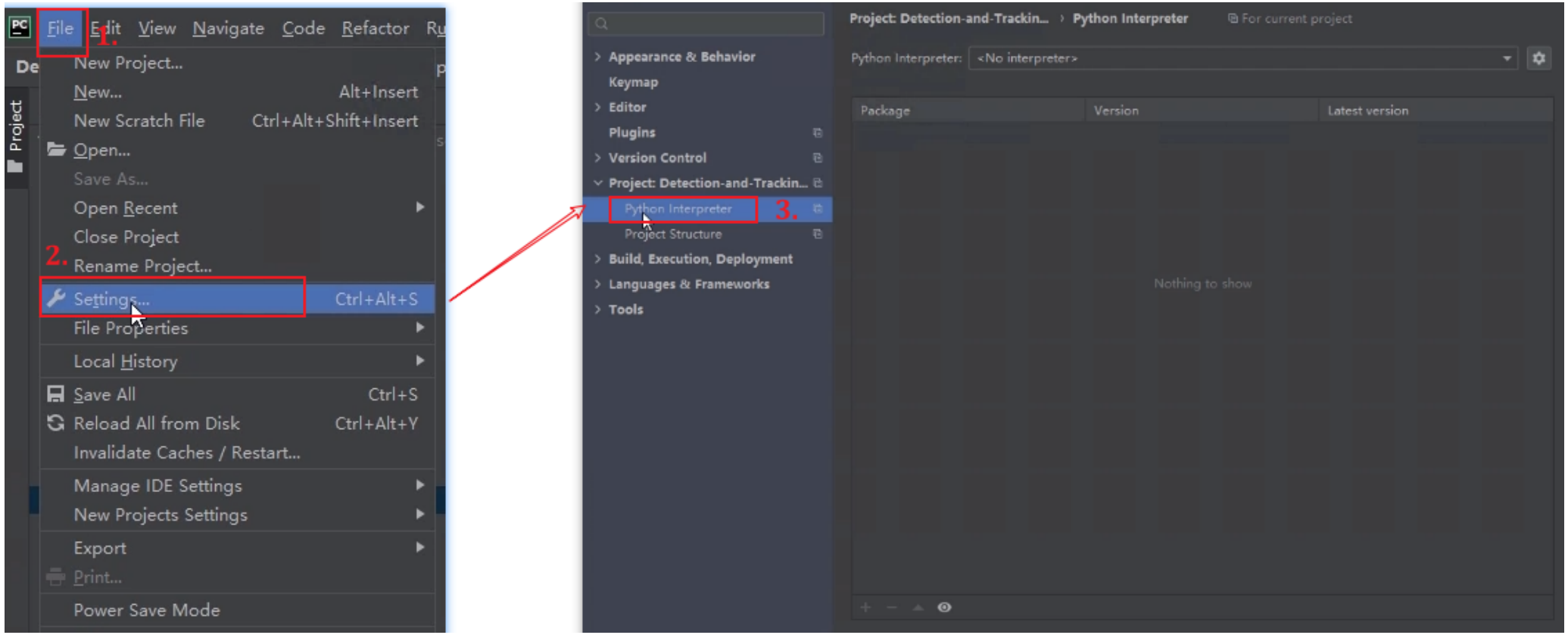
Python编程：基础知识

- Windows环境下安装
  - 在Python官网主页下载并安装Python基本开发和运行环境
    https://www.python.org/downloads/windows/
  - 安装Anaconda包、环境管理器 （推荐）
    https://www.anaconda.com/download
  - 安装PyCharm集成开发环境(IDE)
    https://www.jetbrains.com/pycharm/download/?section=windows

Python编程：基础知识

- PyCharm中配置使用Anaconda环境

  开始配置环境。点击“File”菜单，选择“Setting”选项，在“Project”中找到“Python Interpreter”，输入python的安装目录



Python编程：基础知识

- 搜索“PyCharm中配置使用Anaconda环境”，以下仅供参考

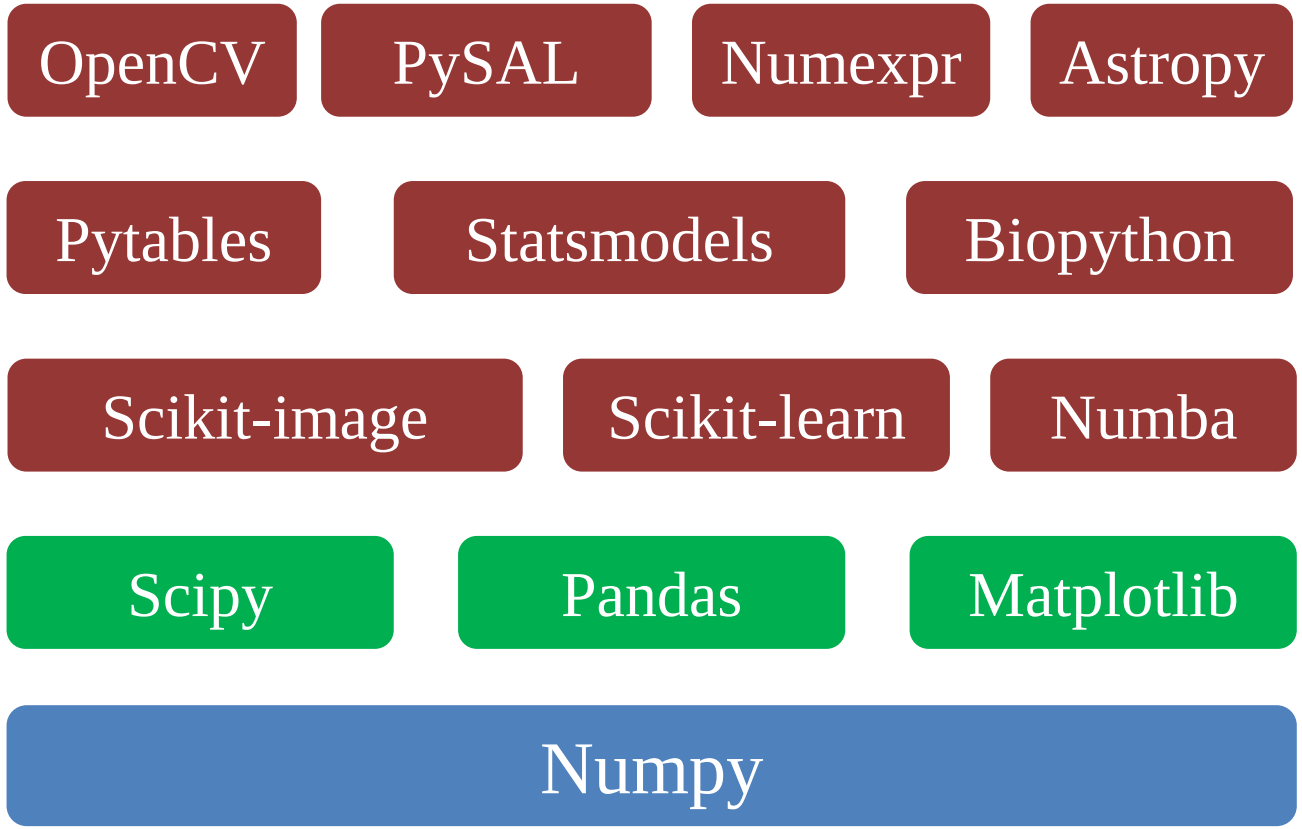
  安装Anaconda手动配置anaconda环境变量（path）方法
  https://blog.csdn.net/fangweijiex/article/details/115825000

  在PyCharm中配置使用Anaconda环境
  https://blog.csdn.net/TuckX/article/details/115681862

Python数据分析

Numpy生态系统

- Numpy生态系统

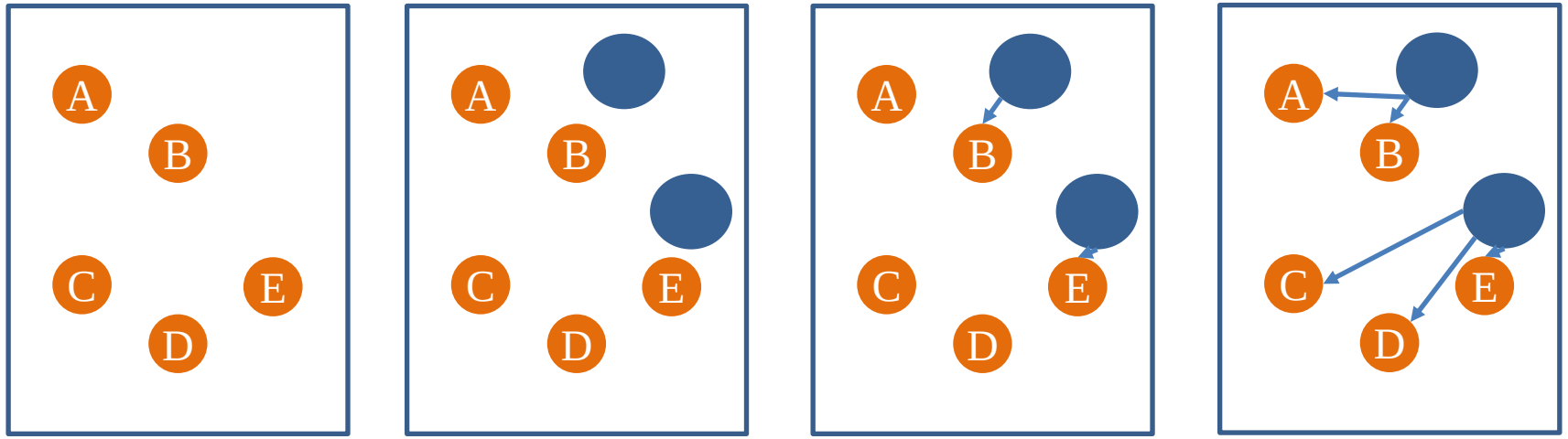


数据分析常用库

- Python科学计算中的常用库
  - **Numpy**
    - /ˈnʌmpaɪ/，Numerical Python，用C语言实现
  - **Pandas**
    - /ˈpændəs/，Panel Data，基于Numpy实现
  - **Matplotlib**
    - Plotting library for basic Python and Numpy
  - **Scipy**
    - /ˈsaɪpaɪˈ/，科学计算库，线性代数、最优化
  - **Scikit-Learn**
    - 机器学习库

Numpy数据分析

- 从一个例子开始：k-means
- 数据个数 n = 5
- 中心点 k = 2

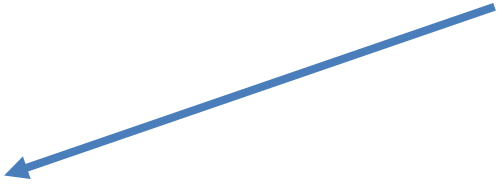


Numpy数据分析

- Python怎么实现 k-means算法?

```
import sklearn.datasets
import sklearn.cluster
import matplotlib.pyplot as plt
import time

n = 1000     # 数据个数
k = 4        # 中心点个数

start = time.time()

# 生成初始数据集，n_samples：数据个数，n_features：数据维度
# centers：数据点中心数
data, labels = sklearn.datasets.make_blobs(
    n_samples=n, n_features=2, centers=k)

# scikit-learn:k-means聚类
kmeans = sklearn.cluster.KMeans(k, max_iter=300)
kmeans.fit(data)
centroids = kmeans.cluster_centers_

print('sklearn k-means:', round(time.time()-start, 2), 'seconds')
# 绘图
plt.scatter(data[:, 0], data[:, 1], c=labels)
plt.scatter(centroids[:, 0], centroids[:, 1], s=300)
plt.show()
```

典型的工具人案例!



Numpy简介

- Numpy的诞生
  - Numerical Python的缩写，底层代码使用C语言实现
  - 1995年Jim Hugunin开发Numeric包
  - 随后，Numarray等包陆续出现
  - 2005年Travis Oliphant 通过将Numarray的功能集成到Numeric包中来创建NumPy包
  - 大量开发者陆续投入到此项目中，日渐强大！

Numpy简介

- Numpy：Python中最重要的科学计算基础库
  - 提供一种高效的多维数组结构，ndarray
  - 提供一系列针对ndarray的高效操作方法
  - 提供一系列文件读写的方法
  - 提供常用的随机数生成、线性代数等方法
  - 提供一套面向C语言的API，能够同其他用C/C++, Fortran语言编写的库共同协作

  - 在线文档
    - https://www.runoob.com/numpy/numpy-tutorial.html

Numpy的优势

- Numpy的优势
  - Numpy采用类似于C语言中的多维数组内存管理机制来管理ndarray
  - Numpy使用C语言实现，应用过程中直接操作底层内存
  - ndarray占用的内存空间较小
  - 为处理大规模数组数据而设计，内含多种高效操作方法

Numpy的优势

- Numpy的优势

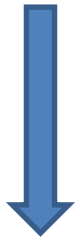
```
import time
import numpy as np

a_arr = np.arange(1000000)
a_list = list(range(1000000))

start = time.time()
for _ in range(10):
    b_arr = a_arr * 2
ndarray_time = round(time.time()-start, 5)
print('Ndarray:\t', b_arr[0],b_arr[1],b_arr[2],'...',b_arr[-1])
print('Ndarray:\t', ndarray_time, 'seconds')

start = time.time()
for _ in range(10):
    b_list = [x * 2 for x in a_list]
list_time = round(time.time()-start, 5)
print('List:\t\t', b_list[0],b_list[1],b_list[2],'...',b_list[-1])
print('List:\t\t', list_time, 'seconds')

print('List消耗时间是ndarray的', list_time//ndarray_time, '倍！')
```

Numpy的优势

- Numpy的优势

```
a_arr = np.arange(1000000)
a_list = list(range(1000000))

for _ in range(10):
    b_arr = a_arr * 2
```

b_arr: 0 2 4 ... 1999998
Time: 0.02s



40倍速度!

b_list: 0 2 4 ... 1999998
Time: 0.80s

```
for _ in range(10):
    b_list = [x * 2 for x in a_list]
```

Numpy的优势

- Numpy的优势

```
import math
start = time.time()
np.sin(np.arange(1000000)) ** 2
ndarray_time = round(time.time()-start, 5)
print('Ndarray:\t', ndarray_time, 'seconds')

start = time.time()
[math.sin(i) ** 2 for i in range(1000000)]
range_time = round(time.time()-start, 5)
print('List:\t\t', range_time, 'seconds')
print('List消耗时间是ndarray的', range_time//ndarray_time, '倍! ')
```

```
Ndarray:          0.012 seconds
List:             0.33508 seconds
List消耗时间是ndarray的 27.0 倍!
```

操作数组，ndarray比list通常快10-100倍!

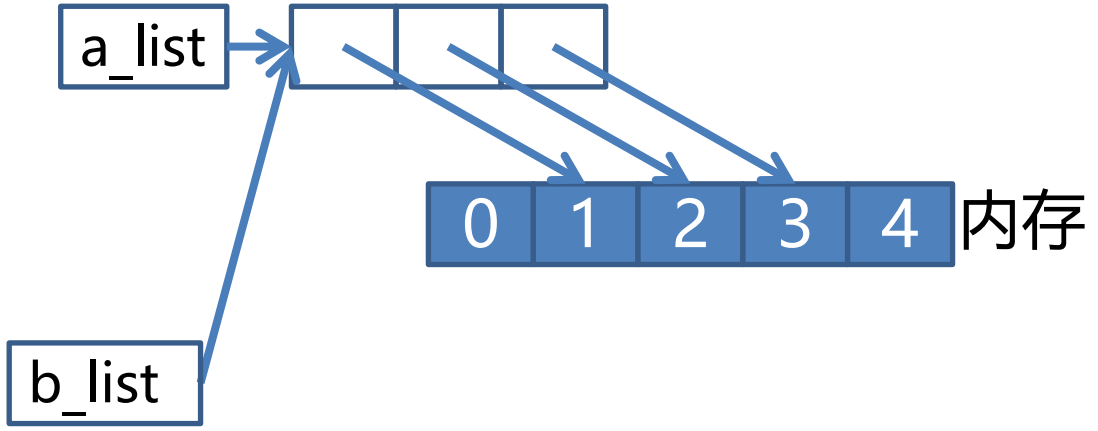
Numpy：ndarray

- ndarray，N-dimensional array的缩写
  - 用于存放同类型元素的多维数组
  - 每个元素在内存中都有相同存储大小的区域
- ndarray的组成部分
  - 一个指向数据（内存或内存映射文件中的一块数据）的指针
  - 数据类型dtype
  - 一个表示数组形状（shape）的元组
  - 一个跨度元组（stride），其中的整数指的是为了前进到当前维度下一个元素需要"跨过"的字节数

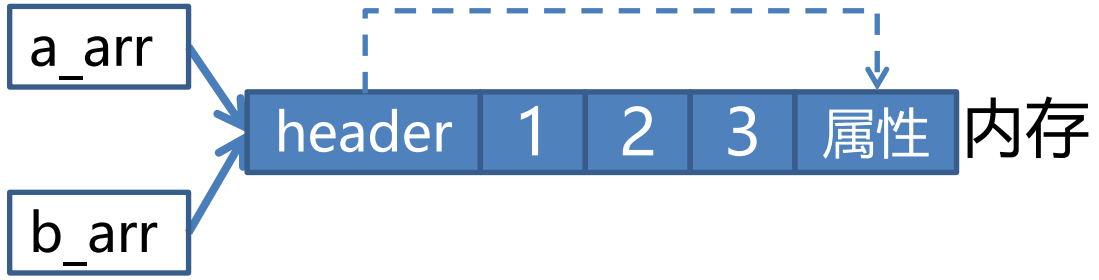
Numpy：ndarray

- ndarray *VS.* list

```
a_list = [1, 2, 3]
b_list = a_list
a_list[0] = 4
print(b_list)
```




```
a_arr = np.array([1,2,3])
b_arr = a_arr
a_arr[0] = 4
print(b_arr)
```



Numpy：ndarray

- ndarray的组成部分
  - 一个指向数据（内存或内存映射文件中的一块数据）的指针
  - 数据类型dtype
  - 一个表示数组形状（shape）的元组
  - **一个跨度元组（stride）**，其中的整数指的是为了前进到当前维度下一个元素需要"跨过"的字节数

```
import numpy as np
arr = np.array([[0,1,2],[3,4,5]], dtype='int16')
```

```
>>> [[0 1 2]
     [3 4 5]]
```

Numpy：ndarray的跨度元组

- ndarray的内存存储机制

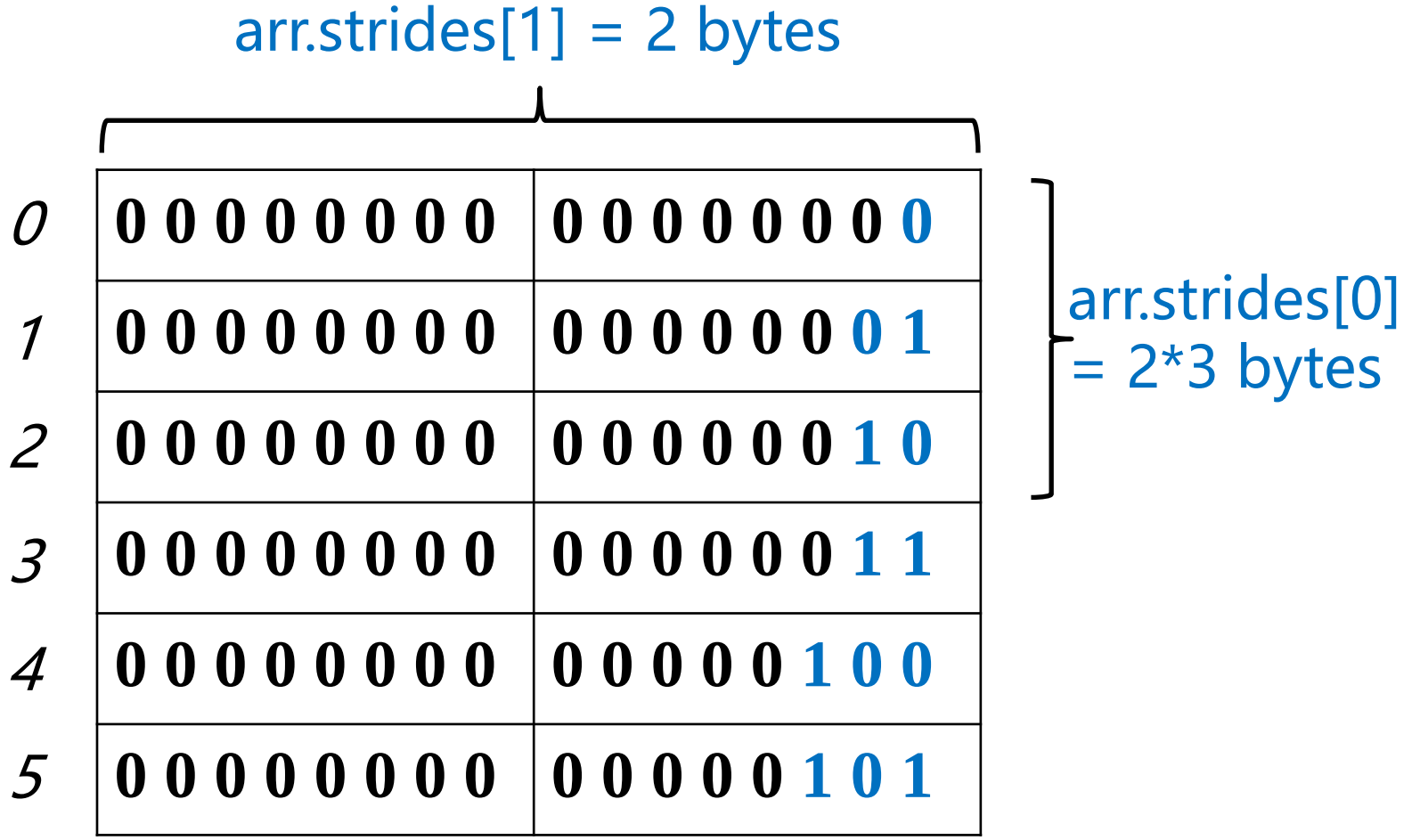
```
arr = [[0 1 2]
       [3 4 5]]
```

arr.dtype = int16

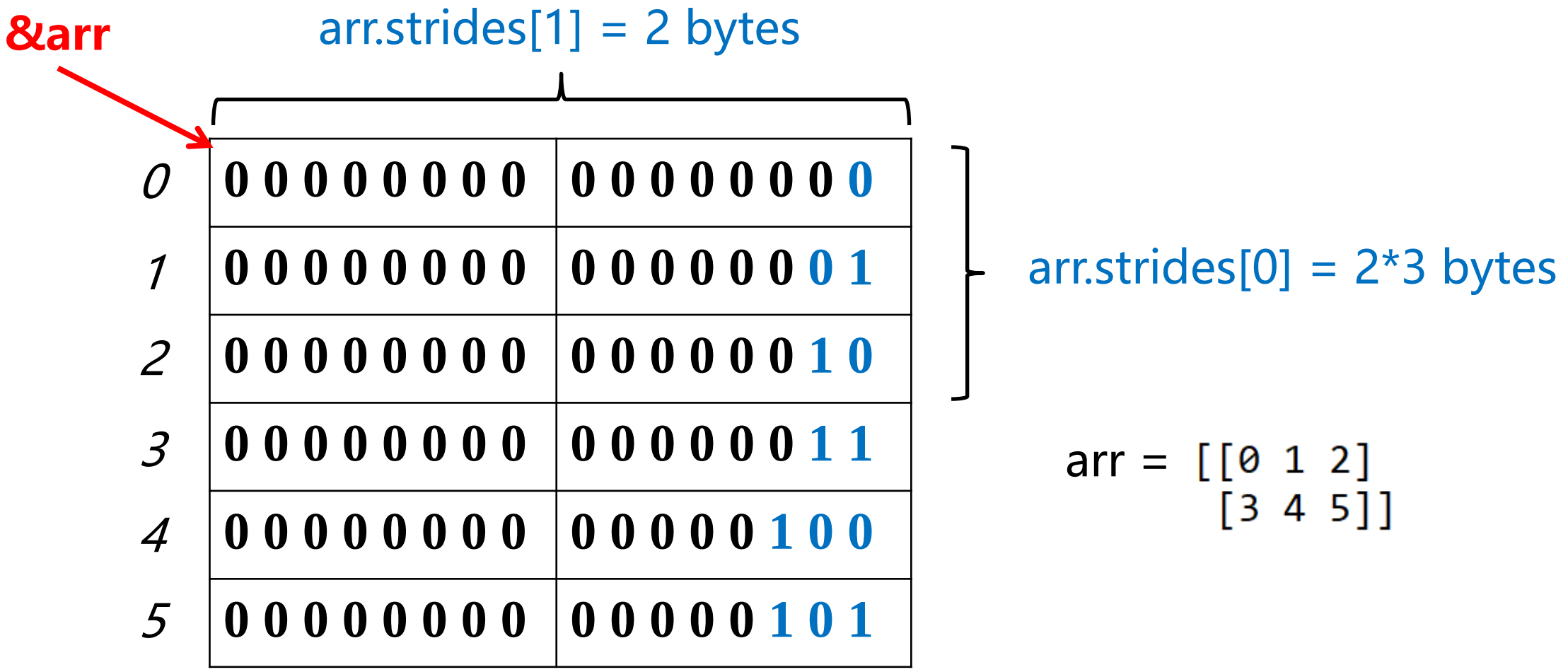
arr.shape = (2, 3)

arr.strides = (6, 2)

第一维度，从元素0到元素3
第二维度，从元素0到元素1



Numpy：ndarray的跨度元组




```
arr = [[0 1 2]
       [3 4 5]]
```

arr[1, 2] = &arr+1*arr.strides[0]+2*arr.strides[1] = &arr + “10” = 5

Numpy：ndarray的跨度元组

```
arr = np.arange(1,25).reshape((2,2,2,3))
```

```
>>> [[[[ 1  2  3]
       [ 4  5  6]]

      [[ 7  8  9]
       [10 11 12]]]


     [[[13 14 15]
       [16 17 18]]

      [[19 20 21]
       [22 23 24]]]]
```

arr.dtype = int32

arr.shape = (2, 2, 2, 3)

arr.strides = (48, 24, 12, 4)

Numpy：性能分析

- 性能对比分析

```
from datetime import datetime
import numpy as np
import matplotlib.pyplot as plt
```

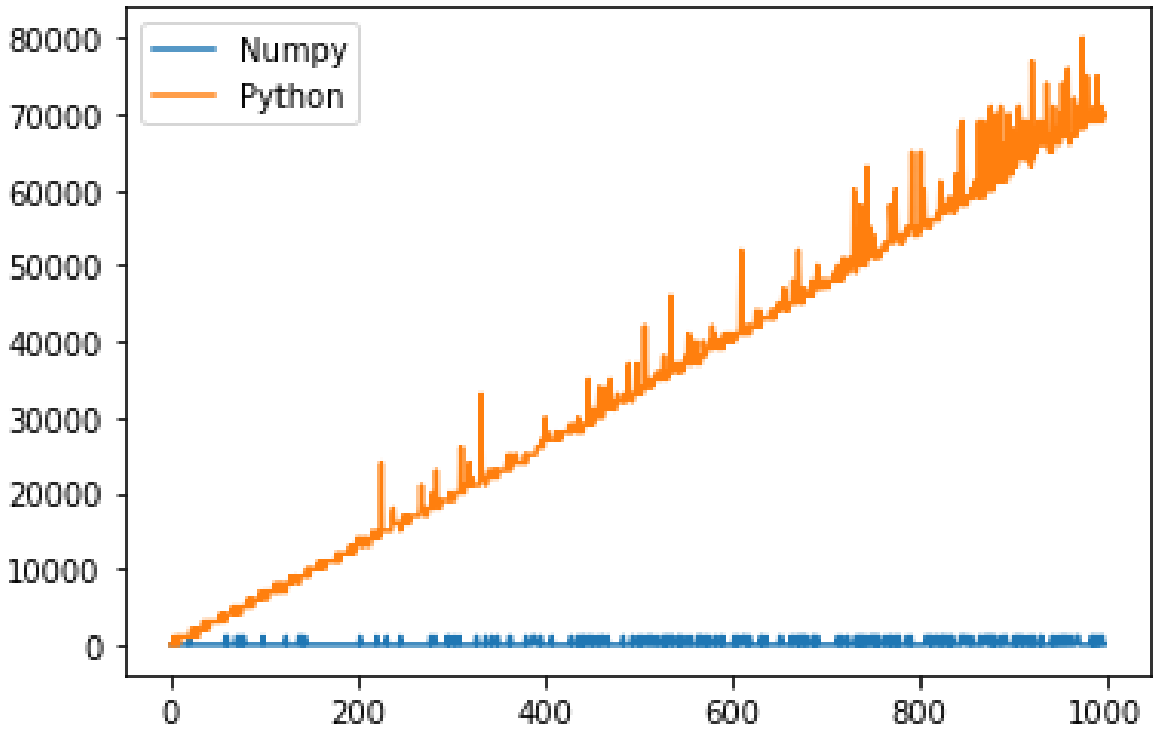
```
# 使用Numpy计算
def numpysum(n):
    a = np.arange(n)**2
    b = np.arange(n)**3
    c = a+b return c
```

```
# 使用Python计算
def pythonsum(n):
    a = list(range(n))
    b = list(range(n))
    c = []
    for i in range(len(a)):
        a[i] = i**2
        b[i] = i**3
        c.append(a[i]+b[i])
    return c
```

```
#性能分析
def printest(func, size):
    start = datetime.now()
    temp = func(size)
    delta = datetime.now() - start
    return delta.microseconds
```

Numpy：性能分析

```
# 用于作 n-time 图
def timeplot():
    pts = []
    for i in range(100,100000,100):
        t_numpy = printest(numpysum,i)
        t_python = printest(pythonsum,i)
        pts.append([t_numpy,t_python])
    plt.plot(pts)
    plt.legend(['Numpy','Python'])
    plt.show()
if __name__ == "__main__":
    size = 100000
    printest(numpysum,size)
    printest(pythonsum,size) timeplot()
```



Numpy：ndarray的创建

- 创建方式
  - np.array(object, dtype = None, copy = True, order = None, subok = False, ndmin = 0)

名称	描述
object	可迭代序列结构
dtype	数组元素的数据类型，可选
copy	对象是否需要复制，可选
order	创建数组的样式，C为行方向，F为列方向，A为任意方向（默认）
subok	默认返回一个与基类类型一致的数组
ndmin	指定生成数组的最小维度

Numpy扩展

https://www.labri.fr/perso/nrougier/from-python-to-numpy/

Numpy：ndarray的创建

- 创建方式
  - np.array(object, dtype = None, copy = True, order = None, subok = False, ndmin = 0)

```
import numpy as np
arr = np.array([1,2,3],dtype='float64')
```
```
[1. 2. 3.]
```

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

```
arr = np.array([[1,2.],[0,0],[1+1j,2.]])
```
```
[[1.+0.j 2.+0.j]
 [0.+0.j 0.+0.j]
 [1.+1.j 2.+0.j]]
```

```
arr = np.array([[i, i * i] for i in [1, 2]])
```
```
[[1 1]
 [2 4]]
```

Numpy：ndarray的创建

- ndarray其他创建方式：随机元素数组
  - np.empty(shape, dtype = float, order = 'C')
  - 数组元素为内存中的随机元素

```
arr = np.empty(shape=3)
[4.24399158e-314 8.48798317e-314 1.27319747e-313]
```

```
arr = np.empty(3)
[4.24399158e-314 8.48798317e-314 1.27319747e-313]
```

```
arr = np.empty(shape=(2,3),dtype='bool')
[[False False False]
 [False  True False]]
```

Numpy：ndarray的创建

- ndarray其他创建方式：随机元素数组
- np.random.randn(d0, d1, ..., dn)
  - 生成一个(d0, d1, ..., dn)维的数组，数组的元素取自服从正态分布的随机数，若没有参数输入，则生成一个数
- np.random.rand(d0, d1, ..., dn)
  - 生成一个(d0, d1, ..., dn)维的数组，数组的元素取自[0, 1)上的均分布，若没有参数输入，则生成一个数
- np.random.uniform(low=0.0, high=1.0, size=None)
  - 生出size个符合均分布的浮点数，取值范围为[low, high)，默认取值范围为[0, 1.0)
- np.random.randint(low, high=None, size=None, dtype='l')
  - 生成size个整数，取值区间为[low, high)，若没有输入参数high则取值区间为[0, low)

Numpy：ndarray的创建

- ndarray其他创建方式：全0/1数组
  - np.zeros(shape, dtype = float, order = 'C')
  - np.ones(shape, dtype = None, order = 'C')

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

```
[[0. 0. 0.]
 [0. 0. 0.]]
```

```
arr = np.zeros((2,3),dtype='int64')
```

```
[[0 0 0]
 [0 0 0]]
```

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

```
[[1. 1. 1.]
 [1. 1. 1.]]
```

Numpy：ndarray的创建

- ndarray其他创建方式：指定数值范围
  - np.arange(start, stop, step, dtype)

参数	描述
start	起始值，默认为0
stop	终止值（不包含）
step	步长，默认为1
dtype	返回ndarray的数据类型，如缺省则会使用输入数据的类型

```
arr = np.arange(4)          [0 1 2 3]

arr = np.arange(4.0)        [0. 1. 2. 3.]
arr = np.arange(4.)
```

Numpy：ndarray的创建

- ndarray其他创建方式：
  - np.full(shape, fill_value, dtype=None, order='C')
  - np.tile(a, reps)
  - np.r_[a_list, b_list]
  - np.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)

```
arr = np.full((2, 3), 1)
```

```
[[1 1 1]
 [1 1 1]]
```

```
a_arr = np.array([1,2])
b_arr = np.tile(a_arr, (2, 3))
```

```
[[1 2 1 2 1 2]
 [1 2 1 2 1 2]]
```

```
arr = np.r_[[1,2],[3,4,5]]
```

```
[1 2 3 4 5]
```

```
arr = np.linspace(-5, 5, 5, False)
```

```
[-5. -3. -1.  1.  3.]
```

Numpy：ndarray的数据类型

- ndarray中的数据类型dtype

名称	描述
bool_	布尔型数据类型（True 或者 False）
int_	默认的整数类型（类似于 C 语言中的 long，int32 或 int64）
intc	与 C 的 int 类型一样，一般是 int32 或 int 64
intp	用于索引的整数类型（类似于 C 的 ssize_t，一般情况下仍然是 int32 或 int64）
int8	字节（-128 to 127）
int16	整数（-32768 to 32767）
int32	整数（-2147483648 to 2147483647）
int64	整数（-9223372036854775808 to 9223372036854775807）

Numpy：ndarray的数据类型

- ndarray中的数据类型dtype

名称	描述
uint8	无符号整数（0 to 255）
uint16	无符号整数（0 to 65535）
uint32	无符号整数（0 to 4294967295）
uint64	无符号整数（0 to 18446744073709551615）
float_	float64 类型的简写
float16	半精度浮点数，包括：1 个符号位，5 个指数位，10 个尾数位
float32	单精度浮点数，包括：1 个符号位，8 个指数位，23 个尾数位
float64	双精度浮点数，包括：1 个符号位，11 个指数位，52 个尾数位
complex_	complex128 类型的简写，即 128 位复数
complex64	复数，表示双 32 位浮点数（实数部分和虚数部分）
complex128	复数，表示双 64 位浮点数（实数部分和虚数部分）

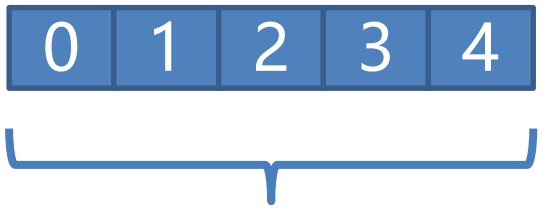
Numpy：ndarray的属性

- ndarray的属性

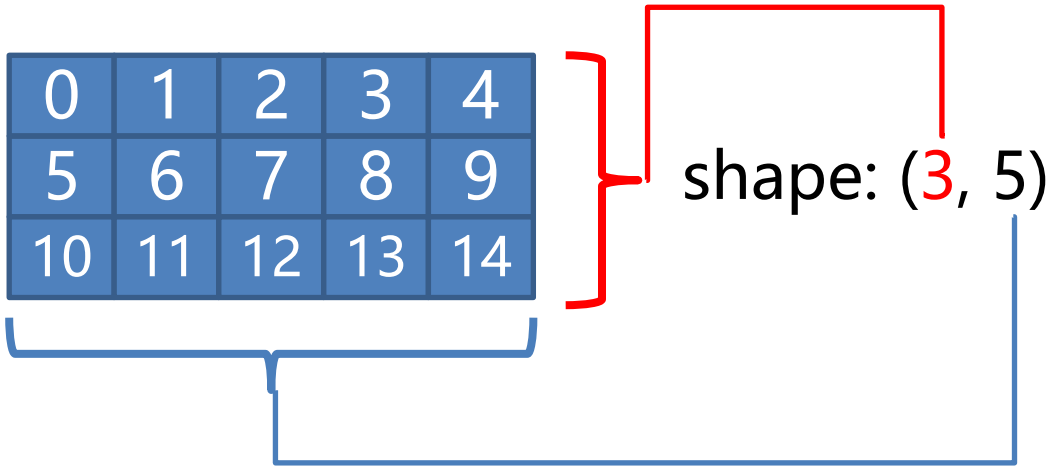
常用属性	说明
ndarray.ndim	秩，即轴的数量或维度的数量
ndarray.shape	数组的形状，对于矩阵，n 行 m 列
ndarray.size	数组元素的总个数，相当于 .shape 中 n*m 的值
ndarray.dtype	ndarray 对象的元素类型
ndarray.itemsize	ndarray 对象中每个元素的大小，以字节为单位
ndarray.flags	ndarray 对象的内存信息
ndarray.real	ndarray元素的实部
ndarray.imag	ndarray 元素的虚部
ndarray.data	包含实际数组元素的缓冲区，由于一般通过数组的索引获取元素，所以通常不需要使用这个属性。

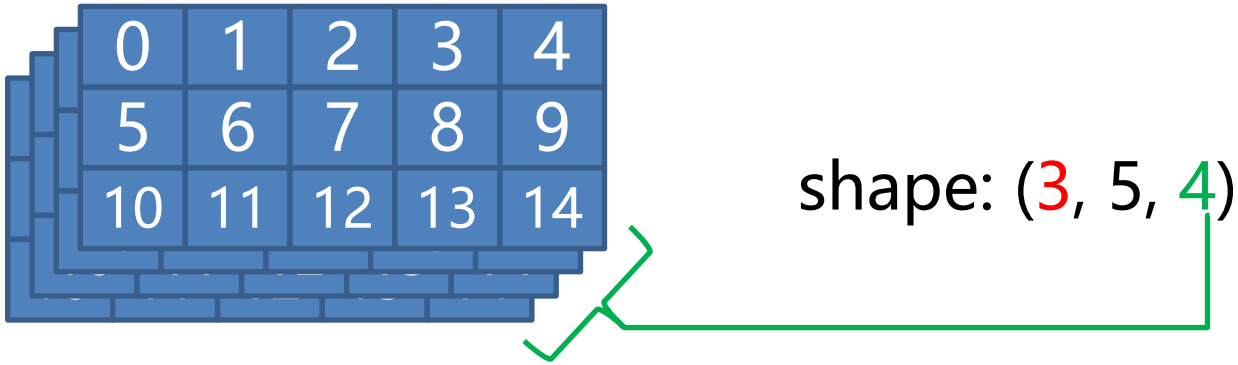
Numpy：ndarray的属性

- ndarray的shape属性



shape: (5,)





Numpy：ndarray的操作

- np.reshape()改变数组形状

```
arr = np.arange(10,20,1).reshape(5,2)
  [[10 11]
   [12 13]
   [14 15]
   [16 17]
   [18 19]]
```

- np.astype()改变数组元素类型

```
arr = np.array([['1','2'],['5','6']]).astype('float64')
    [[1. 2.]
     [5. 6.]]
```

Numpy：ndarray的操作

- ndarray的简单应用

```
arr = np.array([(1,2,3),[4,5,6]])
print('arr:\n',arr)
print('arr - arr:\n',arr - arr)
print('arr * arr:\n',arr * arr)
arr_2 = np.array([(7,8,9),[4,5,6]])
print('arr_2 > arr:\n',arr_2 > arr)
```

```
arr:
 [[1 2 3]
 [4 5 6]]
arr - arr:
 [[0 0 0]
 [0 0 0]]
arr * arr:
 [[ 1  4  9]
 [16 25 36]]

arr_2 > arr:
 [[ True  True  True]
 [False False False]]
```

Numpy：ndarray的操作

- ndarray的数组向量化

```
[ [ 1,  1,  1,  1],          [ [ 1,  1,  1,  1],
  [ 1,  1,  1,  1],     +      [ 1,  1,  1,  1],     = ?
  [ 1,  1,  1,  1] ]           [ 1,  1,  1,  1] ]
```

- 线性计算：
  - [] < 1+1, [2,] < 1+1, [2,2,] < 1+1 …
- 向量化计算

```
[ [ 1+1,  1+1,  1+1,  1+1],
  [1+1,  1+1,  1+1,  1+1],
  [1+1,  1+1,  1+1,  1+1] ]
```

这也是TPU/GPU大放异彩的原因！

Numpy：ndarray切片

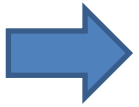
- 一维数组切片

```
import numpy as np
arr = np.arange(10)
# 从索引2开始到索引7停止，间隔为2
s1 = slice(2,7,2)
print(arr[s1])                    [2 4 6]
arr_s2 = arr[2:7:2]
print(arr_s2)                     [2 4 6]
```

Numpy：ndarray切片

- 二维数组切片

```
import numpy as np

arr =np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12],\
               [13,14,15,16],[17,18,19,20]])

print('初始数据: \n',arr)
print('数据维度: ', arr.shape)

print('切片[1,2): ',arr[1:2])
print('初始数据全部切片: \n',arr[:])

print('第一列数据: \n',arr[:,0])
print('第一行数据: \n',arr[0])
print('第一行数据: \n',arr[0,:])

print('右上角数据: \n',arr[:2,1:])
```



```
初始数据:
 [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]
 [17 18 19 20]]
数据维度:  (5, 4)
切片[1,2):  [[5 6 7 8]]
初始数据全部切片:
 [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]
 [17 18 19 20]]
第一列数据:
 [ 1  5  9 13 17]
第一行数据:
 [1 2 3 4]
第一行数据:
 [1 2 3 4]
右上角数据:
 [[2 3 4]
 [6 7 8]]
```

Numpy：ndarray切片

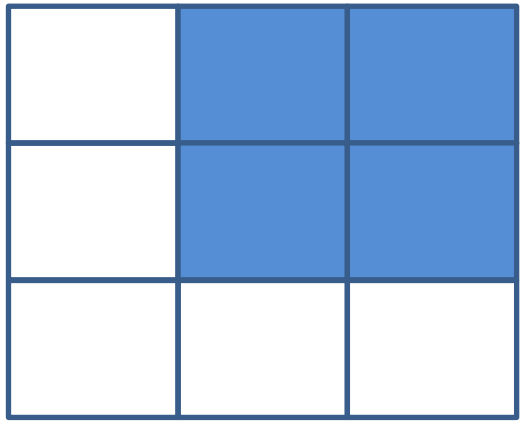
- 二维数组的元素布局

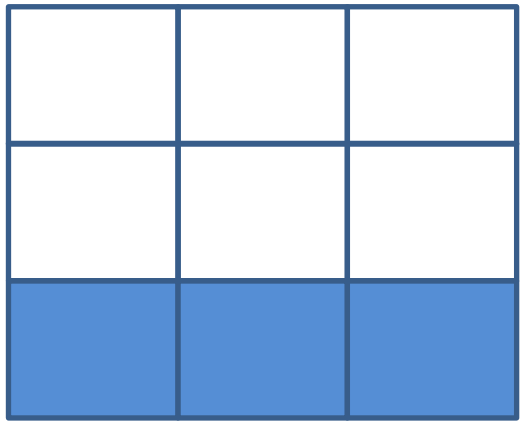
axis1

axis0	0	1	2
0	0,0	0,1	0,2
1	1,0	1,1	1,2
2	2,0	2,1	2,2

Numpy：ndarray切片

- 二维数组的元素布局

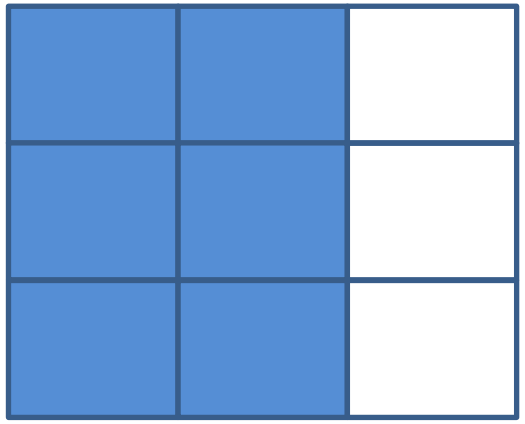


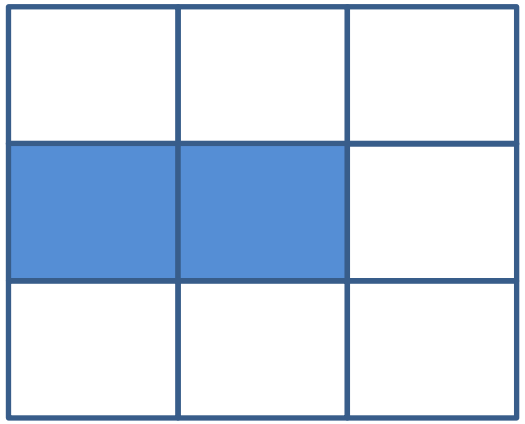


切片表达式	shape
arr[: 2, 1 :]	(2, 2)
arr[2]	(3,)
arr[2, :]	(3,)
arr[2 : , :]	(1, 3)

Numpy：ndarray切片

- 二维数组的元素布局





切片表达式	shape
arr[: , : 2]	(3, 2)
arr[1, : 2]	(2,)
arr[1: 2 : , : 2]	(1, 2)

Numpy：ndarray索引

- 数组索引

```
import numpy as np

arr = np.array([[1, 2],  [3, 4],  [5, 6]])
x = arr[[0,1,2],  [0,1,0]]
print (x)
```

```
[1 4 5]
```

```
arr = np.array([[0, 1, 2],[3, 4, 5],[6, 7, 8],\
                [9, 10, 11]])
rows = np.array([[0,0],[3,3]])
cols = np.array([[0,2],[0,2]])
x = arr[rows,cols]
print  ('这个数组的四个角元素是: \n',x)
```

```
[[ 0  2]
 [ 9 11]]
```

Numpy：ndarray索引

- 布尔索引

```
chars = np.array(['a','b','a','c'])
arr = np.random.rand(4,4)
print('arr: \n',arr)
print('chars = a: \n', chars == 'a')
print('arr[chars = a] \n', arr[chars == 'a'])
```

```
arr:
 [[0.93086288 0.62076943 0.10139825 0.33603834]
 [0.15763767 0.30167788 0.12422861 0.56300044]
 [0.97671332 0.76585451 0.94892057 0.88279619]
 [0.01195383 0.94468459 0.57054435 0.21849045]]
chars = a:
 [ True False  True False]
arr[chars = a]
 [[0.93086288 0.62076943 0.10139825 0.33603834]
 [0.97671332 0.76585451 0.94892057 0.88279619]]
```

Numpy：ndarray索引

- 布尔索引

```
arr = np.array([[0, 1, 2],[3, 4, 5],[6, 7, 8],\
                [9, 10, 11]])
print('大于 5 的元素是: \n', arr[arr>5])
```

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

Numpy：ndarray索引

- 花式索引（Fancy indexing）：利用整数数组进行索引，这里的整数数组可以是Numpy数组也可以是Python中列表、元组等可迭代类型。

```
arr = np.arange(32).reshape((8,4))
print (arr[[4,2,1,7]])
```

```
[[16 17 18 19]
 [ 8  9 10 11]
 [ 4  5  6  7]
 [28 29 30 31]]
```

```
arr = np.arange(32).reshape((8,4))
print(arr[[4,2,1,7,6,4,4,4],:3:2])
```

```
[[16 18]
 [ 8 10]
 [ 4  6]
 [28 30]
 [24 26]
 [16 18]
 [16 18]
 [16 18]]
```

Numpy：ndarray索引

- 除了花式索引，其他切片与索引为浅复制 （花式索引生成一个新的数组，不像切片，花式索引生成的是新的数据对象）

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

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

```
arr_s2 = arr[2:7:2]
```

```
[2 4 6]
```

```
arr_s2[0]=999
```

```
print(arr_s2)
print(arr)
```

```
[999   4   6]
[  0   1 999   3   4   5   6   7   8   9]
```

Numpy：ndarray索引

```
import numpy as np
a = np.arange(6)
print('我们的数组是：')
print(a)
print('调用 id() 函数：')
print(id(a))
print('a 赋值给 b：')
b = a
print(b)
print('b 拥有相同 id()：')
print(id(b))
print('修改 b 的形状：')
b.shape =  3,2
print(b)
print('a 的形状也修改了：')
print(a)
```

```
我们的数组是：
[0 1 2 3 4 5]
调用 id() 函数：
1468387953920
a 赋值给 b：
[0 1 2 3 4 5]
b 拥有相同 id()：
1468387953920
修改 b 的形状：
[[0 1]
 [2 3]
 [4 5]]
a 的形状也修改了：
[[0 1]
 [2 3]
 [4 5]]
```

Numpy：ndarray索引

- 视图：切片与大部分索引均会创建视图
  - ndarray.view() 方会创建一个新的数组对象，该方法创建的新数组的维数更改不会更改原始数据的维数
  - 使用切片创建视图修改数据会影响到原始数组

```
import numpy as np
# 最开始 a 是个 3X2 的数组
a = np.arange(6).reshape(3,2)
print('数组 a: ')
print(a)
print('创建 a 的视图: ')
b = a.view()
print(b)
print('两个数组的 id() 不同: ')
print('a 的 id(): ')
print(id(a))
print('b 的 id(): ')
print(id(b))
# 修改 b 的形状，并不会修改 a
b.shape =  2,3
print('b 的形状: ')
print(b)
print('a 的形状: ')
print(a)
```

```
数组 a:
[[0 1]
 [2 3]
 [4 5]]
创建 a 的视图:
[[0 1]
 [2 3]
 [4 5]]
两个数组的 id() 不同:
a 的 id():
1468388037456
b 的 id():
1468388037376
b 的形状:
[[0 1 2]
 [3 4 5]]
a 的形状:
[[0 1]
 [2 3]
 [4 5]]
```

Numpy：ndarray索引

- ndarray.copy()

```
import numpy as np
a = np.array([[10,10],  [2,3],  [4,5]])
print('数组 a: ')
print(a)
print('创建 a 的深层副本: ')
b = a.copy()
print('数组 b: ')
print(b)
# b 与 a 不共享任何内容
print('我们能够写入 b 来写入 a 吗? ')
print(b is a)
print('修改 b 的内容: ')
b[0,0]  =  100
print('修改后的数组 b: ')
print(b)
print('a 保持不变: ')
print(a)
```

```
数组 a:
[[10 10]
 [ 2  3]
 [ 4  5]]
创建 a 的深层副本:
数组 b:
[[10 10]
 [ 2  3]
 [ 4  5]]
我们能够写入 b 来写入 a 吗?
False
修改 b 的内容:
修改后的数组 b:
[[100  10]
 [  2   3]
 [  4   5]]
a 保持不变:
[[10 10]
 [ 2  3]
 [ 4  5]]
```

Numpy：ndarray计算

```
import numpy as np

arr = np.array([0,30,45,60,90])
print ('不同角度的正弦值：')
# 通过乘 pi/180 转化为弧度
print (np.sin(arr*np.pi/180))
print ('\n')
print ('数组中角度的余弦值：')
print (np.cos(arr*np.pi/180))
print ('\n')
print ('数组中角度的正切值：')
print (np.tan(arr*np.pi/180))
```

Numpy：ndarray计算

不同角度的正弦值：
```
[0.         0.5        0.70710678 0.8660254  1.        ]
```

数组中角度的余弦值：
```
[1.00000000e+00 8.66025404e-01 7.07106781e-01 5.00000000e-01
 6.12323400e-17]
```

数组中角度的正切值：
```
[0.00000000e+00 5.77350269e-01 1.00000000e+00 1.73205081e+00
 1.63312394e+16]
```

Numpy：ndarray计算

- 比较运算: <, <=, ==, !=, >=, >
- 算术: +, -, *, /, reciprocal, square
- 指数计算: exp, expm1, exp2, log, log10, log1p, log2, power, sqrt
- 三角函数: sin, cos, tan, acsin, arccos, atctan
- 双曲线函数: sinh, cosh, tanh, acsinh, arccosh, atctanh
- 位运算: &, |, ~, ^, left_shift, right_shift
- 逻辑运算: and, logical_xor, not, or
- 谓词: isfinite, isinf, isnan, signbit
- 其他运算: abs, ceil, floor, mod, modf, round, sinc, sign,trunc

Numpy：ndarray的广播机制(Broadcast)

- ndarray的数学计算

a	0	1	2	3	4	5
b	0	10	20	30	40	50
c	0	11	22	33	44	55

c = a+ b

Numpy：ndarray的广播机制(Broadcast)

- 广播(Broadcast)： numpy 对不同形状(shape)的数组进行数值计算的方式。当运算中的 2 个数组的形状不同时，numpy 将自动触发广播机制。

a

0	1	2	3	4	5

b

10	*10*	*10*	*10*	*10*	*10*

c = a+ b

c

10	11	12	13	14	15

Numpy：ndarray的广播机制(Broadcast)

- 广播(Broadcast)
  - numpy 对不同形状(shape)的数组进行数值计算的方式
  - 被广播的数组通常会复制元素填充或者赋填充值为1（高维）

```
[ [ 1,  1,  1,  1],          [ [ 2,  2,  2,  2],          [ [ 1,  1,  1,  1],
  [ 1,  1,  1,  1],     +      [ 2,  2,  2,  2],     =      [ 1,  1,  1,  1],     +   [2]
  [ 1,  1,  1,  1] ]           [ 2,  2,  2,  2] ]           [ 1,  1,  1,  1] ]
```

0	1
2	3
4	5

+

10	*10*
20	*20*
30	*30*

=

10	11
22	23
34	35

Numpy：广播机制与 np.newaxis

- np.newaxis 的功能是增加新的维度，np.newaxis 放的位置不同，产生的矩阵形状也不同。

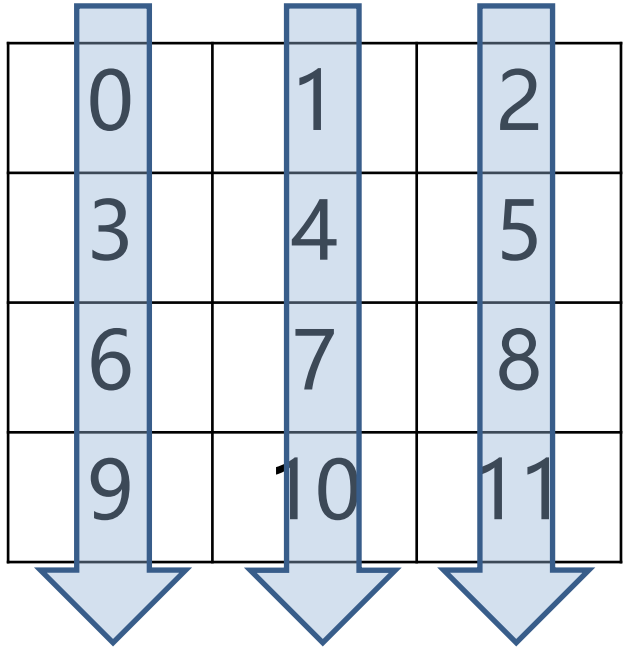
```
a = np.array([ [ 1,  1,  1,  1], [ 1,  1,  1,  1], [ 1,  1,  1,  1] ])
b = np.array([2, 3, 4])
c = np.array([2, 3, 4, 5])

print(a.shape, b.shape)
print(a + b[:, np.newaxis])
print(a + c)
print(a + c[np.newaxis])
```

Numpy：ndarray降维

```
arr = np.arange(12).reshape(4,3)
```

```
>>> [[ 0  1  2]
     [ 3  4  5]
     [ 6  7  8]
     [ 9 10 11]]
```

arr.sum(axis=0) = [18 22 26]



axis=0

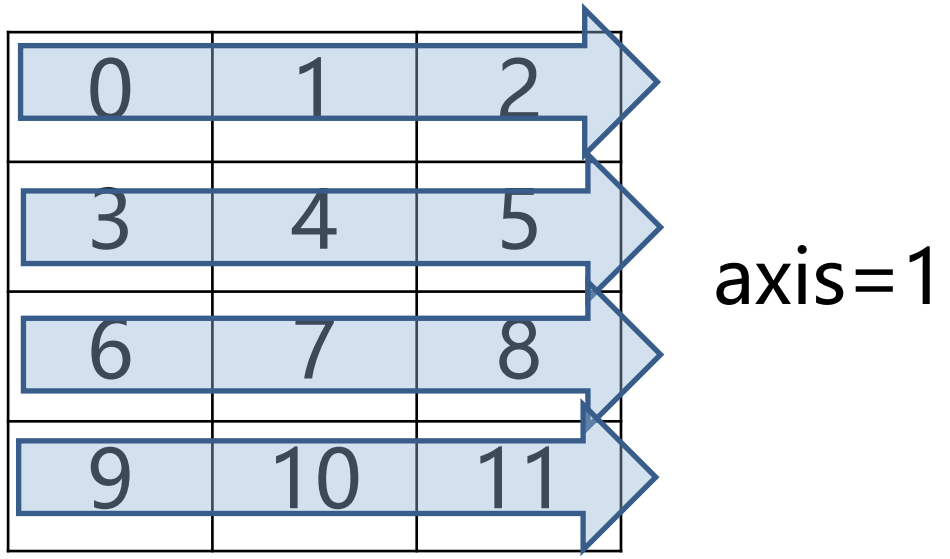
Numpy：ndarray降维

```
arr = np.arange(12).reshape(4,3)
```

```
>>> [[ 0  1  2]
     [ 3  4  5]
     [ 6  7  8]
     [ 9 10 11]]
```

arr.sum(axis=0) = [18 22 26]

arr.sum(axis=1) = [3 12 21 30]



练习题

- 计算矩阵秩。
- 考虑一个数组Z=[1,2,3,4,5,6,7,8,9,10,11,12,13,14]，如何生成一个矩阵R=[[1,2,3,4]，[2,3,4,5]，[3,4,5,6]……，[11,12,13,14]]?
- 如何查找数组中最频繁的值。
- 考虑任意一个16x16矩阵，如何获得矩阵块的和（块大小为4x4）。
- 给定一个矩阵，输出没有相同元素的所有的行。
- 给定两个向量A和B，基于np.einsum实现等价于inner, outer, sum, 和mul函数的表达式。
- 给定两个维度分别为nx4和mx4的矩阵，计算两两之间的IoU距离，得到nxm的矩阵（用numpy实现，不用for循环）。

练习题

- 计算任意函数与x轴围成的面积
  - 定义一个函数，比如sin(x)；
  - 用数值的方法计算该函数在某个区间比如[0, pi]与x轴围成的面积；
  - 用numpy实现，不能用for循环。
- 计算二元函数体积
  - 在x轴[0, 1], y轴[0, 1], 定义一个任意连续的二元函数；
  - 用数值的方法该函数与x,y平面在该区间下的体积；
  - 用numpy实现，不能用for循环；
  - 精确到20位小数点。