---

# Numpy

## 学习目标

- 了解Numpy运算速度上的优势
- 知道数组的属性，形状、类型
- 应用Numpy实现数组的基本操作
- 应用随机数组的创建实现正态分布应用
- 应用Numpy实现数组的逻辑运算
- 应用Numpy实现数组的统计运算
- 应用Numpy实现数组之间的运算

### Numpy

Numpy的优势[重点]

是什么: 是一个科学计算库, 可以用于任意维度数组的运算.

**核心类: ndarray**

numpy优势:

1. 内存风格

ndarray: **元素内置, 速度快, 要求元素数据类型必须一致**

list: 元素外置, 速度慢, 数据类型可以不一致

2. numpy**支持向量化并行**计算

3. 底层是用C语言实现的, **解除GIL锁限制**, 速度更快.

## N维数组-ndarray[*]

```
# 创建
 np.array([[80, 89, 86, 67, 79],[78, 97, 89, 67, 81]])   # 二维数组
```

1. 属性:

- 形状: shape （2,5）二维里面有两个一维，一维里面5个元素
- 维度: ndim 2
- 元素个数: size 5
- 元素元素长度(字节) : itemsize
- 元素数据类型: dtype

2. 数据类型:

- 整数: np.int64
- 浮点数: np.float64
- 字符串: np.string_

### 基本操作

1 生成数组的方法

#### 1.1 生成0和1的数组

根据形状生成全1的数组:

```
np.ones(shape[,dtype,order])
```

根据已知数组的形状生成全1的数组:

```
np.ones_like(arr[], [,dtype,order])
```

根据形状生成全0的数组:

```
np.zeros(shape, [,dtype,order])
```

根据已知数组的形状生成全0的数组:

```
np.zeros_like(arr[], [,dtype,order])
```

### 1.2 从现有数组生成

深拷贝(开辟新存储空间): np.array()

浅拷贝(赋值操作): np.asarray() ， 建立索引指向

### 1.3 生成固定范围的数组

生成等间隔数组(重点):

```
np.linspace(start, stop, num, dtype, endpoint)
# start 序列的起始值
# stop 序列的终止值，
# num 要生成的等间隔样例数量，默认为50
# endpoint 序列中是否包含stop值，默认为ture
```

根据起始值,结束值以及步长生成数组(类似于Python的range方法)(重点):

```
  np.arange(start, stop, step, dtype)
```

从10^start, 10^stop生成num个等比序列:

```
  np.logspace(start, stop, num, endpoint, dtype)
```

## 生成随机数组

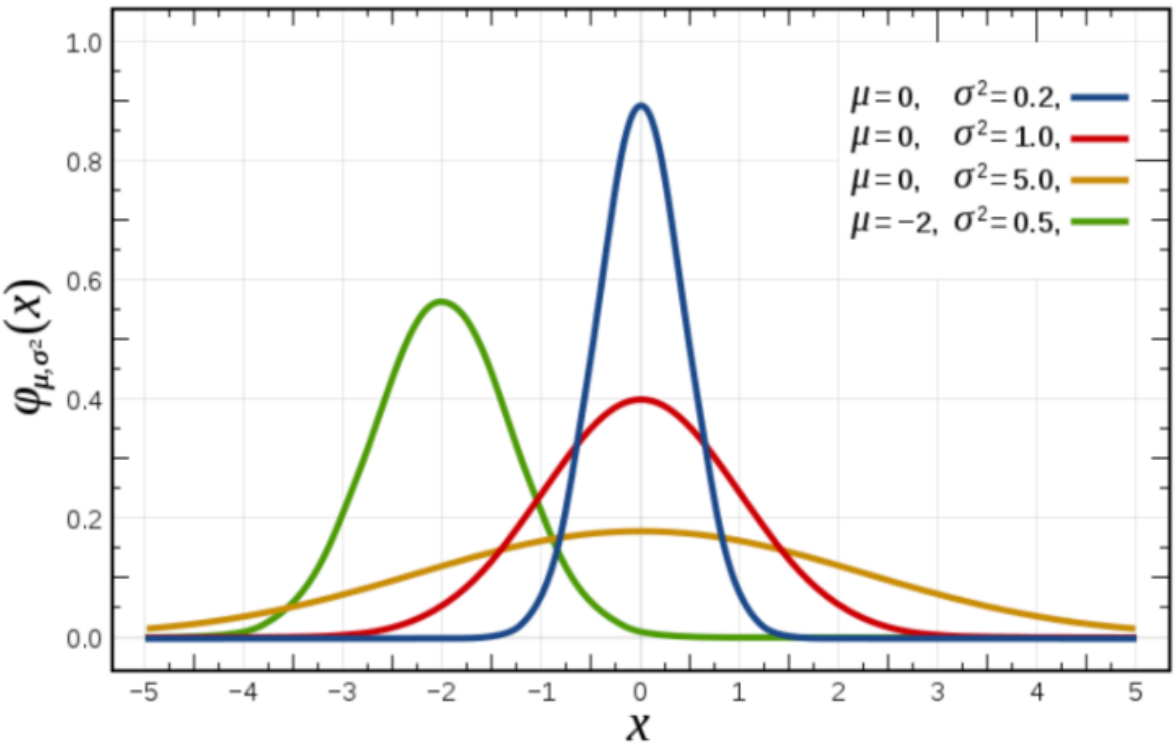
### 1.4 均匀分布

概念: 每一个数出现概率都是一样.

```
np.random.uniform(low, high, size)
# 参数:
  low: 最小值(包含)
```

```
    high: 最大值(不包含)
    size: 大小(可以数也可以是元组)
```

### 1.5 正态分布




1. 正态分布概念:N(均值, 方差)

第一参数μ是服从正态分布的随机变量的均值，第二个参数σ2是此随机变量的方差，所以正态分布记作 N(μ,σ2)

1. 均值 决定了正态分布位置

2. 方差 决定了正则分布矮胖高瘦(方差越小就越高瘦, 方差越大就越矮胖)

$$s^2 = \frac{(x_1 - M)^2 + (x_2 - M)^2 + (x_3 - M)^2 + \cdots + (x_n - M)^2}{n}$$

3. 方差与标准差意义: 用于衡量数据的离散程度的.

4. 标准正态分布: 均值为0, 方差为1的正态分布.

2. API:

```
np.random.normal(loc, scale, size)
# 参数:
    loc: 均值
    scale: 标准差
    size: 大小

```

```
# 创建符合正态分布的8只股票10天的涨跌幅数据
stock_change = np.random.normal(0, 1, (8, 10))  # size二维 8个，每个10个数据
stock_change
```

### 2. 数组的索引、切片

切片: arr[起始索引:结束索引, 起始索引:结束索引]

索引: arr[行索引, 列索引]

### 3. 形状修改

返回相同元素新形状ndarray:

对象.reshape(shape[, order]) # 元素还是按顺序

```
# 从0—24列表中生成4组每组带6个元素的数据
np.arange(24).reshape(4,6)
```

修改原数组的形状: # 元素按顺序

对象.resize()

数组转置: 行列互换

对象.T

### 4. 类型修改

获取新类型的ndarray:

对象.astype(新类型)

返回新类型的数组, 原数组不改变

把ndarray转为二进制数据:

对象.tostring()

### 5 数组的去重,返回去重后的一个一维数组

np.unique(数组)

## ndarray运算[**]

1 逻辑运算

比较运算符: >, < , >=, <=, !=

原理: 使用数组每一个元素与指定的值进行比较

修改满足条件数据(给满足条件的元素赋值):

arr[条件表达式] = 值

```
# BOOL赋值，将满足条件的设置为指定的值-布尔索引
>>> stock_change[stock_change > 0.5] = 1
```

2 通用判断函数

np.all(): 全为True, 才为True

np.any(): 只要有True, 就为True

```
# 判断前5只股票这段期间是否有上涨的
>>> np.any(stock_change[0:5, :] > 0 )
True
```

3. np.where（三元运算符）：

np.where(条件, 满足条件值, 不满足条的值)

```
# 判断前四个股票前四天的涨跌幅 大于0的置为1，否则为0
temp = stock_change[:4, :4]
np.where(temp > 0, 1, 0)
```

逻辑与(并且)

np.logical_and()

逻辑或(或者)

np.logical_or()

```
# 判断前四个股票前四天的涨跌幅 大于0.5并且小于1的，换为1，否则为0
# 判断前四个股票前四天的涨跌幅 大于0.5或者小于-0.5的，换为1，否则为0
np.where(np.logical_and(temp > 0.5, temp < 1), 1, 0)
np.where(np.logical_or(temp > 0.5, temp < -0.5), 1, 0)
```

4. 统计运算

最小值: min(a[, axis,]

最大值: max(axis)

中位数: np.median(数组, axis)

中位数概念: 按从小到大的顺序排序, 元素个数为奇数中间的数, 元素个数为偶数, 中间两个数的平均值

平均值: mean(axis)

方差: var(axis)

标准差: std(axis)

**axis 0代表列, axis 1代表行去进行统计**

最大值所在的索引: argmax(axis)

最小值所在的索引: argmin(axis)

```
# 获取股票指定哪一天的涨幅最大
print("前四只股票前四天内涨幅最大{}".format(np.argmax(temp, axis=1)))
print("前四天一天内涨幅最大的股票{}".format(np.argmax(temp, axis=0)))
```

**矩阵[*]**

矩阵，英文matrix，**和array的区别矩阵必须是2维的，但是array可以是多维的。**

矩阵: 可以理解为二维数组. 由行列组成

向量: 只有一列矩阵就列向量, 只有一行矩阵就行向量, 如果没有特殊说明, 向量指的都列向量

矩阵加法: 矩阵中每一个元素相加得到新矩阵

矩阵与标量相乘: 矩阵的每一个元素与标量相乘得到新矩阵

矩阵相乘[重点]

(M, N) * (N, L) = (M, L)

(M行, N列) x (N行, L列) = (M行, L列)

矩阵乘法性质:

1. 不满足交换律
2. 满足结合律 AxBxC = Ax(BxC)

单位矩阵:
主对角线(左上角到右下角)上的元素都为1, 其他位置全为0
性质; 所有矩阵与单位矩阵相乘还是它本身

矩阵的逆: 矩阵乘以矩阵的逆为单位矩阵
矩阵转置: 行列互换.

4.6 数组间运算[**]
1. 数组与数的运算
数组中每一个元素与数进行运算得到新数组
支持; +, -, * , /

**2. 数组与数组的运算**

广播机制:

Broadcast机制的功能是为了方便不同形状的ndarray（numpy库的核心数据结构）进行数学运算。

只有在下述情况下，两个数组才能够进行数组与数组的运算。

1. 维度相同
2. shape的其中一方为1

**3. 矩阵运算**

## 矩阵乘法api

1. np.matmul(a, b)
2. np.dot(a, b)

相同点: 都做矩阵乘法
不同点: matmul不能做矩阵和标量的乘法, dot可以.