向量、矩阵等线性代数概念对于数据科学和机器学习至关重要。在机器学习中，数据几乎都以矩阵形式存储、运算。毫不夸张地说，没有线性代数就没有现代计算机运算。逐渐地，大家会发现算数、代数、解析几何、微积分、概率统计、优化方法并不是一个个孤岛，而线性代数正是连接它们的重要桥梁之一。

行向量、列向量

若干数字排成一行或一列，并且用中括号括起来，得到的数组叫作向量(vector)。
排成一行的叫作行向量(row vector)，排成一列的叫作列向量(column vector)。
通俗地讲，行向量就是表格的一行数字，列向量就是表格的一列数字。以下两例分别展示了行向量和列向量，即

$$\begin{bmatrix}1 & 2 & 3\end{bmatrix}_{1\times 3}, \begin{bmatrix}1\\2\\3\end{bmatrix}_{3\times 1} \tag{1}$$

式(1)中，下角标“1×3”代表“1行、3列”，“3×1”代表“3行、1列”。

转置

转置符号为上标“T”。行向量转置(transpose)可得到列向量；同理，列向量转置可得到行向量。举例如下，有

$$\begin{bmatrix}1 & 2 & 3\end{bmatrix}^{\mathrm{T}} = \begin{bmatrix}1\\2\\3\end{bmatrix}, \begin{bmatrix}1\\2\\3\end{bmatrix}^{\mathrm{T}} = \begin{bmatrix}1 & 2 & 3\end{bmatrix} \tag{2}$$

矩阵：数字排列成长方形

矩阵(matrix)将一系列数字以长方形方式排列，如

$$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix}_{2\times 3}, \begin{bmatrix}1 & 2\\3 & 4\\5 & 6\end{bmatrix}_{3\times 2}, \begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}_{2\times 2} \tag{3}$$

通俗地讲，矩阵将数字排列成表格，有行、有列。式 (3) 给出了三个矩阵，形状分别是2行3列、3行2列和2行2列。
通常用大写字母代表矩阵，比如矩阵A和矩阵B。
图2所示为一个n×D矩阵X。n是矩阵的行数(number of rows in the matrix)，D是矩阵的列数(number of columns in the matrix)。X可以展开写成表格形式，即

$$X_{n\times D} = \begin{bmatrix}x_{1,1} & x_{1,2} & \cdots & x_{1,D}\\x_{2,1} & x_{2,2} & \cdots & x_{2,D}\\\vdots & \vdots & \ddots & \vdots\\x_{n,1} & x_{n,2} & \cdots & x_{n,D}\end{bmatrix} \tag{4}$$

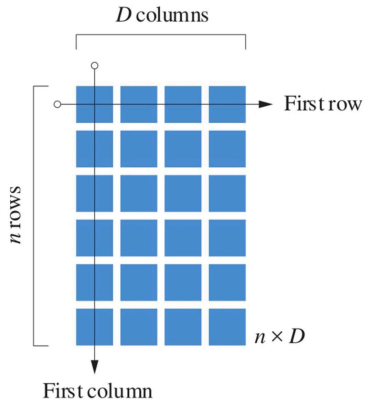

图2　n×D矩阵X

再次强调：先说行号，再说列号。数据矩阵一般采用大写X表达。

矩阵X中，元素(element) $x{i,j}$ *被称作i,j元素（i j entry或j element），也可以说*$x{i,j}$出现在i行j列(appears in row i and column j)。比如，$x_{n,1}$是矩阵X的第n行、第1列元素。
表1 总结了如何用英文读矩阵和矩阵元素。

表1　矩阵有关英文表达

数学表达	英文表达
$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}$	Two by two matrix, first row one two, second row three four
$\begin{bmatrix}a_{1,1} & a_{1,2} & \cdots & a_{1,n}\\a_{2,1} & a_{2,2} & \cdots & a_{2,n}\\\vdots & \vdots & \ddots & \vdots\\a_{m,1} & a_{m,2} & \cdots & a_{m,n}\end{bmatrix}$	*m* by *n* matrix, first row *a* sub one one, *a* sub one two, dot dot dot, *a* sub one n, second row *a* sub two one, *a* sub two two, dot dot dot, *a* sub two *n* dot dot dot last row *a* sub *m* one, *a* sub *m* two, dot dot dot *a* sub *m n*
$a_{i,j}$	Lowercase (small) *a* sub *i* comma *j*
$a_{i,j+1}$	Lowercase *a* double subscript *i* comma *j* plus one
$a_{i,j-1}$	Lowercase *a* double subscript *i* comma *j* minus one

range中使用min、max

02659:Bomb Game

matrices, http://cs101.openjudge.cn/practice/02659/

```
...
for (R, S, P, T) in bombs:
    for i in range(max(0, R - (P - 1) // 2), min(A, R + (P + 1) // 2)):
        for j in range(max(0, S - (P - 1) // 2), min(B, S + (P + 1) // 2)):
...
```

04133:垃圾炸弹

matrices, http://cs101.openjudge.cn/practice/04133/

```
for i in range(max(x-d, 0), min(x+d+1, 1025)):
    for j in range(max(y-d, 0), min(y+d+1, 1025)):
```

（2）字符串（string）
字符串是由字符（字母、数字、符号等）组成的**有序集合**。
字符串与列表类似，也支持序列操作（如索引、切片、拼接等），但**字符串不可变**。

常见方法示例（表1-4）：

表1-4 Python字符串常用方法

Method Name	Use	Explanation
`center`	`astring.center(w)`	返回宽度为 `w` 的居中字符串
`count`	`astring.count(item)`	返回子串出现次数
`lower`	`astring.lower()`	转换为小写
`find`	`astring.find(item)`	返回首次出现子串的下标，若不存在则返回 -1
`split`	`astring.split(schar)`	按分隔符拆分为字符串列表

例如：

```
>>> "CS101".lower()
'cs101'
>>> "a,b,c".split(",")
['a', 'b', 'c']
```

（3）元组（tuple）
元组与列表类似，也是有序集合，但与字符串一样**不可变**。
语法形式为括号 `()` 内一组逗号分隔的值：

```
>>> t = (1, "data", 3.14)
```

元组可用于需要不可修改序列的场景，例如作为字典的键。

（4）集合（set）
集合是零个或多个**不可变对象**的**无序集合**，不允许重复元素。
语法形式为花括号 `{}`，或使用 `set()` 构造函数：

```
>>> s = {1, 2, 3}
>>> emptySet = set()
```

集合支持数学运算（表1-5）：

表1-5 Python集合的基本运算

Operation Name	Operator	Explanation
membership	in	判断元素是否在集合中
union	|	合并
intersection	&	交集
difference	-	差集
subset	<=	判断是否为子集

集合的方法（表1-6）：

表1-6 Python集合常用方法

Method Name	Use	Explanation
`union`	`aset.union(otherset)`	返回并集
`intersection`	`aset.intersection(otherset)`	返回交集
`difference`	`aset.difference(otherset)`	返回差集
`issubset`	`aset.issubset(otherset)`	判断是否为子集
`add`	`aset.add(item)`	添加元素
`remove`	`aset.remove(item)`	移除元素（若不存在则报错）
`pop`	`aset.pop()`	随机移除并返回一个元素
`clear`	`aset.clear()`	清空集合

（5）字典（dict）
字典是由**键-值对**组成的无序集合。语法形式为花括号 `{}` 内一系列 `key:value`：

```
>>> student = {"name": "Alice", "id": 1001}
```

键必须是不可变对象（如字符串、整数、元组），值则没有限制。

常见操作（表1-7）：

表1-7 Python字典的基本操作

Operator	Use	Explanation
`[]`	`myDict[k]`	通过键访问值，若不存在则报错
`in`	`key in adict`	判断键是否存在
`del`	`del adict[key]`	删除指定键

常见方法（表1-8）：

表1-8 Python字典常用方法

Method Name	Use	Explanation
`keys`	`adict.keys()`	返回所有键（dict_keys对象）
`values`	`adict.values()`	返回所有值（dict_values对象）
`items`	`adict.items()`	返回所有键-值对（dict_items对象）
`get`	`adict.get(k)`	获取键对应的值，不存在时返回 `None`
`get`	`adict.get(k,alt)`	获取键对应的值，不存在时返回指定默认值 `alt`

例如：

```
>>> student.get("name")
'Alice'
>>> student.get("age", 18)
18
```

1.2 基本语法

1.2.1 输入与输出

程序通常需要与用户交互，以便获取输入或输出结果。多数应用使用图形化对话框来收集数据，但在入门阶段，Python 提供了更简洁的方式。

输入：
Python 内置函数 `input` 用于从用户处获取输入。它接受**一个**字符串作为参数，该字符串常被称为**提示字符串**（prompt），用于提示用户输入内容。

```
aName = input("Please enter your name: ")
```

无论用户输入什么，`input` 都会返回**一个**字符串。若需要使用其他类型（如整数或浮点数），必须显式进行类型转换：

```
age = int(input("Please enter your age: "))
```

输出：
`print` 函数用于向屏幕输出信息。默认情况下，参数之间以空格分隔，末尾带有换行符。可以通过 `sep` 参数更改分隔符，通过 `end` 参数修改结尾字符：

```
print("Hello", "World", sep="-", end="!")
# 输出: Hello-World!
```

当需要更精确地控制输出格式时，可以使用**格式化字符串**。格式化字符串将固定文本与占位符结合，后续由变量填充：

```
name = "Alice"
age = 20
print(f"{name} is {age} years old.")
```

> **04140: 方程求解**
>
> 牛顿迭代法，http://cs101.openjudge.cn/practice/04140/
>
> ```
> ....
> print(f"{root2:.9f}")
> ```

`read`一次性读入，配合`iter`,`next`

> 有的题目输入数据本来是给C++设计的，如果用Python需要一次性都读入。
>
> **04093: 倒排索引查询**
>
> data structures, http://cs101.openjudge.cn/practice/04093/
>
> ...
>
> 样例输入
>
> ```
> 3
> 3 1 2 3
> 1 2
> 1 3
> 3
> 1 1 1
> 1 -1 0
> 1 -1 -1
> ```
>
> ...
>
> ```
> import sys
>
>
> def main():
>  data = sys.stdin.read().split()
> ```
>
> ```
>  it = iter(data)
>
>  # 读入倒排索引的词数
>  N = int(next(it))
>  inverted = []
>  for _ in range(N):
>      # 每个词的出现文档数
>      count = int(next(it))
>      docs = set()
>      for _ in range(count):
>          docs.add(int(next(it)))
>      inverted.append(docs)
>
>  # 读入查询数目
>  M = int(next(it))
>  output_lines = []
>  for _ in range(M):
>      # 每个查询包含 N 个数字
>      query = [int(next(it)) for _ in range(N)]
>      candidate = None
>                  ...
>      # 输出结果
>      if candidate:
>          result_line = " ".join(map(str, sorted(candidate)))
>          output_lines.append(result_line)
>      else:
>          output_lines.append("NOT FOUND")
>
>  sys.stdout.write("\n".join(output_lines))
>
>
> if __name__ == '__main__':
>  main()
> ```
>
> **M12029:水淹七军**
>
> bfs, dfs, http://cs101.openjudge.cn/pctbook/M12029/
>
> ```
> # 读取并处理输入
> data = sys.stdin.read().split()
> k = int(data[0])
> id = 1
> ans = []
> ...
> ```
>
> **09201: Freda的越野跑**
>
> http://cs101.openjudge.cn/practice/09201/
>
> ...
>
> 样例输入
>
> ```
> 5
> 1 3 10 8 5
> ```
>
> ...
>
> ```
> import sys
> ...
> n = int(sys.stdin.readline())
> a = list(map(int, sys.stdin.readline().split()))
> ```

1.2.2 控制结构

算法的执行离不开**迭代**和**分支**两类控制结构。Python 提供了多种方式来实现。

迭代：

- `while` 语句在条件为真时重复执行：

  ```
  while count < 5:
      print(count)
      count += 1
  ```

- `for` 语句更为常用，能够直接遍历序列或其他可迭代对象：

  ```
  for item in [1, 2, 3]:
      print(item)
  ```

分支：
`if` 语句用于条件判断。可与 `else` 和 `elif` 组合，实现多分支逻辑：

```
if x > 0:
    print("Positive")
elif x == 0:
    print("Zero")
else:
    print("Negative")
```

与多数语言类似，Python 允许**嵌套分支**，但推荐使用 `elif` 来减少层次，提高可读性。

列表解析式：
除了常规循环，Python 提供了**列表解析式（list comprehension）**，用于基于条件与处理规则快速构建列表：

```
squares = [x**2 for x in range(10) if x % 2 == 0]
```

1.2.3 异常处理

在编程过程中，错误主要分为两类：

1. **语法错误**：编写语句或表达式时出错，解释器无法执行。
2. **逻辑错误**：程序能够运行，但结果错误。例如除以零、访问越界，这些通常会引发**运行时错误**，也称为**异常**。

Python 提供了机制来处理异常，而不是简单地终止程序。可以使用 `try-except` 语句捕获并处理异常：

```
try:
    result = 10 / 0
except ZeroDivisionError:
    print("Division by zero is not allowed.")
```

此外，程序员还可以在特定情况下主动抛出异常，以便提醒调用者：

```
raise ValueError("Invalid input!")
```

处理不定行输入的常用方法
在 Python 中，处理多行输入有以下几种常见方式：

使用 try...except 捕获输入结束（如 EOFError）
利用 sys.stdin 逐行读取
通过 sys.stdin.read() 一次性读取所有输入

04015: 邮箱验证

strings, http://cs101.openjudge.cn/practice/04015

这题目输入没有明确结束，需要套在try ... except里面。测试时候，需要模拟输入结束，看你是window还是mac。If the user hits EOF (*nix: Ctrl-D, Windows: Ctrl-Z+Return), raise EOFError.

```
while True:
try:
  s = input()
except EOFError:
  break

        ...
print('NO' if q==-1 else 'YES')
```

1.2.4 定义函数

函数是一种过程抽象工具，可以将复杂的计算隐藏在简洁的调用中。
在 Python 中，函数定义包含：

- **函数名**
- **参数列表**（可以为空）
- **函数体**
- （可选）**返回值**

示例：

```
def square(x):
    return x * x

print(square(5))  # 输出 25
```

通过函数，程序员可以将问题分解为更小的部分，提高代码的可读性与复用性。

19942: 二维矩阵上的卷积运算

matrices, http://cs101.openjudge.cn/practice/19942/

思路：
定义函数真的太方便啦！

18161:矩阵运算(先乘再加)

1. 代码实现 & 解题思路

```
def juzhen (a, b):
    l = []
    for i in range(a):
        l.append(list(map(int, input().split())))
    return l

a, b = map(int, input().split())
l1 = juzhen(a, b)
a, b = map(int, input().split())
l2 = juzhen(a, b)
a, b = map(int, input().split())
l3 = juzhen(a, b)

def cheng(A, B):
    row_A, col_A = len(A), len(A[0])
    row_B, col_B = len(B), len(B[0])
    result = [[0] * col_B for _ in range(row_A)]
    for i in range(row_A):
        for j in range(col_B):
            for k in range(col_A):
                result[i][j] += A[i][k] * B[k][j]
    return result

def jia(A, B):
    row, col = len(A), len(A[0])
    result = [[A[i][j] + B[i][j] for j in range(col)] for i in range(row)]
    return result

if len(l1[0]) != len(l2):
    print("Error!")
elif len(l1) != len(l3) or len(l2[0]) != len(l3[0]):
    print("Error!")
else:
    AB = cheng(l1, l2)
    result = jia(AB, l3)
    for row in result:
        print(*row)
```

解题思路：为了方便，把所有的过程（乘，加，建矩阵）设置成函数了（def juzhen, def cheng, def jia）新获得的知识：print(*row)真方便！

1.3 面向对象编程

Python 是一门支持面向对象编程（Object-Oriented Programming, OOP）的语言。到目前为止，我们已经使用了一些内置类来展示数据结构与控制流程的基本用法。面向对象编程最强大的特性之一，是允许程序员（即问题求解者）定义全新的类，以精确地对现实世界或抽象问题中的数据进行建模。

此前，我们通过**抽象数据类型**（Abstract Data Type, ADT）从逻辑上描述了数据对象的状态及其行为。通过构建实现这些抽象数据类型的类，我们既能利用抽象思维，又能为程序中实际使用这些抽象提供具体的实现细节。每当需要实现一个抽象数据类型时，创建一个新的类便成为自然的选择。

Q: ADT是Class还高级的抽象？

简单来说：**是的，ADT（抽象数据类型）比 Class（类）更高级、更抽象。**

可以从以下几个层面来理解它们之间的关系：

1. **定义上的区别**