# Basic Operation of Python

**说明** 代码前面的 `In []:` 以及输出前面的 `Out []:` 仅为标识，并不代表实际意义，部分标号不连续，是因为有精简的关系

## Python 的赋值

在 Python 中，每个变量在使用前都必须赋值，并且是运用等号（`=`）来给变量赋值，等号（`=`）运算符左边是一个变量名，右边是存储该变量中的值，即

```
variable = value
```

Python 的变量名可以用英文字母、数字和下划线构成，但是必须记住以下四点注意事项

1. 可以单独用英文字母或者下划线作为变量名，同时英文字母需要注意区分大小写
2. 数字不能单独用于变量名
3. 变量名不能以数字开头
4. 变量名应该简洁易懂，比如，对于变量的命名尽可能运用该变量的英文名、英文缩写或者英文名的首字母

其他提醒

- 下关键字不能声明为变量名

```
['and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif',
'else', 'except', 'exec', 'finally', 'for', 'from', 'global', 'if',
'import', 'in', 'is', 'lambda', 'not', 'or', 'pass', 'print', 'raise',
'return', 'try', 'while', 'with', 'yield']
```

- 驼峰体和下划线都可以，官方更推荐第二种

**示例**

```
In [1]: loan_prime_rate = 0.0385    # 贷款市场报价利率等于 3.85%
```

或者是

```
In [2]: lpr = 0.0385
```

这里请注意，凡是以 `#` 开始输入的内容，属于注释语句，Python 是不会读取的，因此编程者可以通过注释语句让阅读者更好地理解代码的含义；此外百分数在 Python 输入中必须用小数表示，因为 `%` 在 Python 中有特殊的含义

## Python 的数据类型

Python 拥有丰富的数据类型，典型的数据类型包括数字、字符串、布尔值与空值

## 整型

在 Python 中，与整数相对应的就是整型，显示为 int（即整数英文单词 integer 的缩写）；此外，长整型可以表示无限大的整数

**示例**

```
In [3]: number = 300    # 数量等于 300

In [4]: type(number)
Out[4]: int
```

注意，在 Python 中，整型有以下特征

1. 必须为数字
2. 不能出现小数点

## 浮点型

浮点型（也称为浮点数）是最常见的数据类型，简单来说就是由整数，小数点和小数（非必须）构成的数字

我们在前面关于 lpr 的实例就是一个浮点型

```
In [5]: type(lpr)
Out[5]: float
```

需要注意的是，在 Python 输入数字时，如果数字中带有小数点，无论实质上是否为整数，Python 均会将其视作浮点类型

比如，我们在前面 `number` 变量输入时，在 100 后增加了一个小数点

```
In [6]: number = 300.

In [7]: type(number)
Out[7]: float
```

## 字符串

在 Python 中针对表示文本的基本数据类型就是需要用到字符串（string），关于字符串有四大特征

### 需要引号

在 Python 中，用英文的单引号''，双引号" "以及三引号''' '''来标识字符串，其中，三引号往往是用于多行文本，引号内无论中文、英文字母、符号、数字甚至是空格，均被视为字符串

```
In [10]: a = 'Python'

In [11]: type(a)
Out[11]: str

In [12]: b = 'Machine Learning'

In [13]: type(b)
Out[13]: str
```

注意，Machine 与 Learning 之间的空格也占据一个字符

```
In [14]: c = '机器学习'

In [15]: type(c)
Out[15]: str

In [16]: d = '2020'

In [17]: type(d)
Out[17]: str

In [18]: e = '1+1'

In [19]: type(e)
Out[19]: str
```



## 可索引性

字符串是可以被索引的，并且如果针对字符串是从左向右的索引，则默认是从 0 开始，最大范围就是字符串长度减去 1，并且需要用到中括号；相反，从右向左索引，默认是从 -1 开始，最大范围是字符串的开头

以上的索引规则不仅仅适用于字符串，对于 Python 中涉及的其他索引也全都适用

```
In [20]: a[0]   # 索引首字母
Out[20]: 'P'

In [21]: a[3]   # 索引第 4 个字母
Out[21]: 'h'

In [22]: a[-1]  # 索引最后一个字母
Out[22]: 'n'
```

## 可截取性

截取也称为切片（slice），是从一个字符串中获取子字符串，需要使用中括号、起始索引值（start）、终止索引值（end）以及可选的步长（step）来定义，截取的输入格式如下

```
string[start: end: step]
```

**示例**

```
In [23]: x = 'I love AI'

In [24]: x[2:6]     # 截取字符串 love
Out[24]: 'love'
```

**注意**

1. 在截取的操作中，截取的子字符串是截止到输入的终止索引值的前一位
2. 不输入步长的参数时，默认步长是 1
3. 如果仅输入起始索引值但不输入终止索引值，则表示截取从起始索引值一直到字符串末尾的子字符串
4. 不输入起始索引值但输入终止索引值，表示截取从字符串的部首一直到终止索引值前一位的子字符串

```
In [25]: x[2:]      # 从第 3 位开始一直到最后一位
Out[25]: 'love AI'

In [26]: x[:6]
Out[26]: 'I love'

In [27]: x = 'I love artificial intelligence'

In [28]: x[2:17:3]  # 截取从第 3 位至第 17 位并且步长为 3 的子字符串
Out[28]: 'lerfi'
```

### 可替换性

如果用户发现在 Python 输入字符串中的部分子字符串出现错误，这时用户就可以运用 replace 函数实现修正和替换

```
In [29]: y = 'I love AI'        # 输入有误

In [30]: y.replace('AI', 'artificial intelligence')     # 进行更正
Out[30]: 'I love artificial intelligence'
```

在 `replace()` 中，第一个引号中是输入原有内容，第二个引号中是输入修正后的内容

## Python 的数据结构

目前，Python 主要包括元组、列表、集合、字典、数组和数据框这 6 种数据格式

| 数据结构类型 | 简介和特征 | 特征性标识 |
| --- | --- | --- |
| 元组 tuple | 一种高级的数据结构、可索引，但不可修改 | 用小括号 () 标识 |
| 列表 list | 与元组类似，除了可索引以外，更重要的是可以修改，能够理解为元组的升级版，而元组是稳定版的列表 | 用中括号 [] 标识 |
| 集合 set | 类似于数学的集合，每个集合中的元素具有无序性、不重复性的特征 | 用大括号 {} 标识 |
| 字典 dict | 与日常生活中的字典用法类似，通过“名称（键 key）→内容（值 value）”来构建 | 用大括号 {} 标识 |
| 数组 array | 科学计算和代数运算常用的数据类型，类似于线性代数的向量和矩阵 | 用 array、小括号、中括号共同标识 |
| 数据框 dataframe | 数据分析处理常用的数据类型，带有索引（index）和列名（column），类似于 Excel 的工作表 | 用 DataFrame、小括号、中括号共同标识 |

关于数组和数据框，将分别在 NumPy 模块和 Pandas 模块中讨论

# 元组

元组的创建方式如下

```
variable = (element1, element2, element3, ...)
```

## 元组的创建

```
In [31]: tup1 = ( )

In [32]: type(tup1)
Out[32]: tuple

In [33]: tup2 = (3,)    # 在元素后面加一个逗号

In [34]: type(tup2)
Out[34]: tuple

In [35]: tup3 = (3)     # 没有在元素后面加一个逗号

In [36]: type(tup3)
Out[36]: int

In [37]: tup4 = ('AI', '人工智能', 2020, 0.0385)

In [38]: type(tup4)
Out[38]: tuple
```

### 元组的访问

元组一旦创建成功后，元组中的元素是不可以修改的，只能进行访问，访问的方式与前面介绍的字符串索引是非常类似的

```
In [39]: tup4[0]         # 访问元组的第一个元素
Out[39]: 'AI'

In [40]: tup4[-1]        # 访问元组的最后一个元素
Out[40]: 0.0385

In [41]: tup4[1:3]       # 访问元组第 2 个至第 3 个元素
Out[41]: ('人工智能', 2020)
```

## 列表

列表的创建方式与元组是比较相似的，只不过运用中括号

```
variable = [element1, element2, element3, ...]
```

### 列表的创建

```
In [42]: list1 = [ ]

In [43]: type(list1)
Out[43]: list

In [44]: list2 = ['AI', 'artificial intelligence', '人工智能', 2020, 0.0385]

In [45]: type(list2)
Out[45]: list
```

### 访问列表和访问

访问列表与访问元组的方式是类似的

```
In [46]: list2[0]
Out[46]: 'AI'

In [47]: list2[-1]
Out[47]: 0.0385

In [48]: list2[2:4]
Out[48]: ['人工智能', 2020]
```

如果需要得到元素 `2020` 在列表中的索引值，需要运用 `index` 函数

```
In [49]: list2.index(2020)
Out[49]: 3
```

索引值 3 就是代表第 4 个元素，如果没有

**注意**

- 如果检索的元素不在列表中，则会出现 `ValueError`
- 如果检索的元素在列表中不唯一，则返回值较小的索引

## 列表的修改——添加

对于已有的列表，在新增元素时可以运用 `append` 函数，并且新元素时添加到列表的结尾

```
In [50]: list1.append(0.0385)

In [51]: list1.append(0.0405)

In [52]: list1.append(0.0415)

In [53]: list1.append(0.0420)

In [54]: list1.append(0.0425)

In [55]: list1
Out[55]: [0.0385, 0.0405, 0.0415, 0.042, 0.0425]
```

## 列表的修改——删除

对列表中元素的删除分为两类

- 删除指定的元素，运用 `remove` 函数
- 删除列表中全部的元素，也就是清空列表，需要运用 `clear` 函数

```
In [56]: list1.remove(0.0425)

In [57]: list1
Out[57]: [0.0385, 0.0405, 0.0415, 0.042]

In [58]: list3 = [2, 4, 6, 8, 10, 2, 6, 2]

In [59]: list3.remove(2)    # 删除列表中值为 2 的第 1 个元素

In [60]: list3
Out[60]: [4, 6, 8, 10, 2, 6, 2]
```

需要注意的是，如果在一个列表中有多个相同值的元素 x，则 `remove(x)` 是删除列表中值为 x 的第 1 个元素，而非全部 x 的元素

```
In [61]: list3.clear()

In [62]: list3
Out[62]: []
```

### 列表的修改——插入

针对列表的指定位置插入元素，就需要运用 `insert` 函数，该函数需要输入两个参数

1. 位置参数，相当于索引值
2. 需要插入的元素值

```
In [63]: list1.insert(2, 0.0555)

In [64]: list1
Out[64]: [0.0385, 0.0405, 0.0555, 0.0415, 0.042]
```

### 列表的排序

- 由小到大排序，需要用到 `sort` 函数
- 由大到小排序，需要用到 `reverse` 函数

```
In [65]: list1.sort()       # 由小到大排序

In [66]: list1
Out[66]: [0.0385, 0.0405, 0.0415, 0.042, 0.0555]

In [67]: list1.reverse()    # 由大到小排序

In [68]: list1
Out[68]: [0.0555, 0.042, 0.0415, 0.0405, 0.0385]
```

### 列表的计数

在一个列表中，假定某个元素出现多次，可能需要计算该元素出现的次数，需要用到 `count` 函数

```
In [69]: list4 = [1, 2, 3, 1, 5, 1, 6, 2, 9, 1, 2, 7]

In [70]: list4.count(1)     # 计算列表中数字 1 出现的次数
Out[70]: 4

In [71]: list4.count(2)     # 计算列表中数字 2 出现的次数
Out[71]: 3
```

## 集合

在 Python 中，集合（set）的概念接近数学上的集合，每个集合中的元素是无序且不重复的，因此就可以通过集合区判断数据的从属关系，有适还可以通过集合把数据结构中重复的元素过滤掉

集合的创建方式如下

```
variable = {element1, element2, element3, ...}
```

但是，集合不可以被截取，也不能被索引，只能包括并集、差集、交集等集合运算，同时，集合可以被添加和删除

## 集合的创建

```
In [72]: set1 = {'PLA', 'KNN', 'PCA', 'CNN', 'SVM'}

In [73]: type(set1)
Out[73]: set

In [74]: set2 = {'CNN', 'PCA', 'RNN', 'GAN', 'SVM'}

In [75]: type(set2)
Out[75]: set
```

## 集合的运算——并集

针对集合求并集时，需要运用到符号 `|`

```
In [76]: set1 | set2
Out[76]: {'CNN', 'GAN', 'KNN', 'PCA', 'PLA', 'RNN', 'SVM'}
```

## 集合的运算——交集

针对集合求交集时，需要运用到符号 `&` 或者运用 `intersection` 函数

```
In [77]: set1 & set2
Out[77]: {'CNN', 'PCA', 'SVM'}

In [78]: set1.intersection(set2)
Out[78]: {'CNN', 'PCA', 'SVM'}
```

## 集合的运算——差集

针对集合求差集时，需要运用到符号 `-`

```
In [79]: set1 - set2        # set1 对 set2 的差集
Out[79]: {'KNN', 'PLA'}

In [80]: set2 - set1        # set2 对 set1 的差集
Out[80]: {'GAN', 'RNN'}
```

### 集合的修改

可以在已经创建的集合中添加新的元素，需要运用 `add` 函数，并且输出的结果可能会自行排列

```
In [81]: set1.add('BPNN')

In [82]: set1
Out[82]: {'BPNN', 'CNN', 'KNN', 'PCA', 'PLA', 'SVM'}
```

### 集合的删除

对集合元素删除时，运用 `discard` 函数，并且输出的结果可能会自行排列

```
In [83]: set1.discard('CNN')

In [84]: set1
Out[84]: {'BPNN', 'KNN', 'PCA', 'PLA', 'SVM'}
```

## 字典

字典的形式如下

```
variable = {key1: value1, key2: value2, key3: value3, ...}
```

需要注意的是，字典有 3 个特征

- 字典中的元素必须以键（key）和值（value）的形式成对出现，也就是所谓的键-值存储
- 键不可以重复，但值可以重复
- 键不可以修改，但值可以修改，并且可以是任意的数字类型

### 字典的创建

字典的创建可以采用两种不同的方法

- 直接法，一次输入所有全部的键与值
- 间接法，先创建一个空字典，然后逐对输入键与值

```
In [85]: dict1 = {'课程名称': 'AI 从 0 到 N', '课程代码': 'AI666', '所属公司':
'九州云'}    # 直接法创建

In [86]: dict1
Out[86]: {'课程名称': 'AI 从 0 到 N', '课程代码': 'AI666', '所属公司': '九州云'}
```

```
In [87]: type(dict1)
Out[87]: dict

In [88]: dict2 = {}      # 间接法创建

dict2['课程名称'] = 'AI 从 0 到 N'
dict2['课程代码'] = 'AI666'
dict2['所属公司'] = '九州云'

In [89]: dict2['课程名称'] = 'AI 从 0 到 N'
    ...: dict2['课程代码'] = 'AI666'
    ...: dict2['所属公司'] = '九州云'

In [90]: dict2
Out[90]: {'课程名称': 'AI 从 0 到 N', '课程代码': 'AI666', '所属公司': '九州云'}

In [91]: type(dict2)
Out[91]: dict
```

## 字典的访问

通过 `keys` 函数访问并输出字典中的全部键，用 `values` 函数访问并输出字典中的全部值

```
In [92]: dict1.keys()        # 输出全部键
Out[92]: dict_keys(['课程名称', '课程代码', '所属公司'])

In [93]: dict1.values()      # 输出全部值
Out[93]: dict_values(['AI 从 0 到 N', 'AI666', '九州云'])
```

也可以通过 `items` 遍历字典的全部元素，也就是将字典中的每个元素（即每对键与值）组成一个元组并存放在列表中输出

```
In [94]: dict1.items()
Out[94]: dict_items([('课程名称', 'AI 从 0 到 N'), ('课程代码', 'AI666'), ('所属公司', '九州云')])
```

如果仅仅是查询某个键对应的值，可以直接通过在中括号内输入建码的方式完成

```
In [95]: dict1['所属公司']       # 注意是用中括号
Out[95]: '九州云'
```

## 字典的修改

```
In [96]: dict1['课程代码'] = 'AI1314'

In [97]: dict1
Out[97]: {'课程名称': 'AI 从 0 到 N', '课程代码': 'AI1314', '所属公司': '九州云'}
```

如果在已经创建的字典中，新增加建与值，可以运用 `update` 函数

```
In [98]: dict1.update({'课程语言': 'Python', '学习时间': '20 小时'})    # 注意外面是小括号，外面是大括号

In [99]: dict1
Out[99]:
{'课程名称': 'AI 从 0 到 N',
 '课程代码': 'AI1314',
 '所属公司': '九州云',
 '课程语言': 'Python',
 '学习时间': '20 小时'}
```

如果在已经创建好的字典中，删除相应的键与值，需要运用 `del` 命令

```
In [100]: del dict1['学习时间'], dict1['课程代码']

In [101]: dict1
Out[101]: {'课程名称': 'AI 从 0 到 N', '所属公司': '九州云', '课程语言': 'Python'}
```

# Python 的运算符号

## 基本算术运算符号

假设变量 `a=10`，变量 `b=23`

| 运算符 | 描述 | 实例 |
| --- | --- | --- |
| + | 加 —— 两个对象相加 | `a + b` 输出结果 33 |
| - | 减 —— 得到负数或是一个数减去另一个数 | `a - b` 输出结果 -13 |
| * | 乘 —— 两个数相乘或是返回一个被重复若干次的字符串 | `a * b` 输出结果 230 |
| / | 除 —— x 除以 y | `b / a` 输出结果 2.3 |
| % | 取模 —— 返回除法的余数 | `b % a` 输出结果 3 |
| ** | 幂 —— 返回 x 的 y 次幂 | `a ** b` 为 10 的 23 次方 |
| // | 取整除 —— 向下取接近商的整数 | `-b // a` 的结果为 -3 |

# 关系运算符

**相等运算符** 用 `==` 表示

```
In [136]: 2 == 2
Out[136]: True

In [137]: True == False
Out[137]: False
```

**不相等运算符** 用 `!=` 表示

```
In [138]: '2' != 2
Out[138]: True

In [139]: 2 != 2
Out[139]: False
```

**大于运算符** 用 `>` 表示

```
In [140]: True > False
Out[140]: True

In [141]: 2 > 2.0
Out[141]: False
```

**大于等于运算符** 用 `>=` 表示

```
In [142]: 'a' >= 'A'
Out[142]: True

In [143]: '%' >= 'A'
Out[143]: False
```

**小于运算符** 用 `<` 表示

```
In [144]: -1 < False
Out[144]: True

In [145]: 10 < 3.1416 ** 2
Out[145]: False
```

**小于等于运算符** 用 `<=` 表示

```
In [146]: -10 // 3 <= -4
Out[146]: True

In [147]: 'a' <= 'A'
Out[147]: False
```

## 赋值运算符号

Python 除了最基础的赋值方法，也就是基本赋值运算符号 `=` 外，还有将不同的算数符号与基本赋值运算符号结合在一起而形成的高级赋值运算符号，`+=`，`-=`，`*=`，`/=`，`**=`，`%=`，`//=` 等

## 成员运算符

Python 的成员运算符号，可以判断一个元素是否在某一个列表中，可以判断一个字符是否属于某个字符串

| 成员运算符 | 描述 |
| --- | --- |
| in | 如果一个变量在指定的另一个变量（列表、元组、字符串等）中找到相应的值，则返回 `True`，否则返回 `False` |
| not in | 如果一个变量在指定的另一个变量中没有找到相应的值，则返回 `True`，否则返回 `False` |

**示例**

```
In [176]: a = 1

In [177]: b = 3

In [178]: c = [1, 2, 4, 8, 16]

In [179]: a in c
Out[179]: True

In [180]: b in c
Out[180]: False

In [181]: d = 'AI'

In [182]: e = '机器学习'

In [183]: f = ['artificial intelligence', '机器学习', '支持向量机']

In [184]: d not in f
Out[184]: True
```

```
21
22 In [185]: e not in f
23 Out[185]: False
```

## Python 的内置函数

Python 内置函数就是指直接调用而无须导入（`import`），是 Python 自带的函数，任何时候都可以被直接使用，内置函数数目众多，可以运用命令 `dir(__builtins__)` 查看，这里省略部分结果

```
1  In [186]: dir(__builtins__)
2  Out[186]:
3  ['ArithmeticError',
4   'AssertionError',
5   'AttributeError',
6   'BaseException',
7   ...
8   'type',
9   'vars',
10  'zip']
```

以下介绍一些常见的内置函数

1. `abs` 求绝对值（abs 是英文单词 absolute 的缩写）

   ```
   1 In [187]: a = -5
   2  ...: abs(a)
   3 Out[187]: 5
   ```

2. `enumerate` 将对象（如列表、元组或字符串）组合成为一个带有索引的序列，一般用于 `for` 循环中，要输出结果需要用到 `list` 函数， `start=1` 代表 `1` 是索引的起始值，也可以任意设定索引起始值

   ```
   1 In [188]: ml = ['kNN', 'LR', 'SVM', 'BPNN']
   2
   3 In [189]: list(enumerate(ml, start=1))
   4 Out[189]: [(1, 'kNN'), (2, 'LR'), (3, 'SVM'), (4, 'BPNN')]
   ```

3. `float` 数字或字符串转换为浮点型

```
# 整数转化为浮点型
In [190]: b = 6

In [191]: float(b)
Out[191]: 6.0

# 字符串转化为浮点型
In [192]: c = '28'

In [193]: float(c)
Out[193]: 28.0
```

4. `int` 数字或字符串转为整型

```
# 浮点型转化为整型
In [194]: d = 4.6

In [195]: int(d)
Out[195]: 4

# 字符串转化为整型
In [196]: e = '12'

In [197]: int(e)
Out[197]: 12
```

5. `len` 输出对象（包括字符串、列表、元组等）长度或元素个数，len 是长度英文单词 length 的缩写

```
# 输出字符串的长度
In [198]: a = 'AI'

In [199]: len(a)
Out[199]: 2

In [200]: c = '机器学习'

In [201]: len(c)
Out[201]: 4

# 输出列表的元素个数
In [202]: list2 = ['AI', 'artificial intelligence', '人工智能', 2020,
0.0385]

In [203]: len(list2)
Out[203]: 5
```

6. `max` 求最大值（max 是最大值英文单词 maximum 的缩写）

```
In [204]: list1 = [0.0385, 0.0405, 0.0415, 0.042, -0.0425]

In [205]: max(list1)
Out[205]: 0.042
```

7. `min` 求最小值（min 是最小值英文单词 minimum 的缩写）

```
In [206]: min(list1)
Out[206]: -0.0425
```

8. `pow` 幂函数（破位是取了幂英文单词 power 的缩写），`pow(x, y)` 是输出 $x^y$ 的值

```
In [207]: pow(2, 5)
Out[207]: 32

In [208]: pow(5, 2)
Out[208]: 25
```

9. `print` 输出字符串和变量等，输入的字符串需要放在引号的中间，并且字符串与输入的变量之间需要用逗号隔开

```
In [209]: print(list2[2], '的英文是:', list2[1])
人工智能 的英文是: artificial intelligence
```

10. `range` 输出一个整数数列，一般用于 `for` 循环中，函数的语法结构为 `range(x, y, z)`
    - 参数 `x` 表示计数的起始值，默认（不填）是从 `0` 开始
    - 参数 `y` 表示计数的终止值，但不包括 `y`
    - 参数 `z` 表示步长，默认为 `1`

```
In [210]: list(range(0, 10))
Out[210]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [211]: list(range(10))
Out[211]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [212]: list(range(2, 13, 3))
Out[212]: [2, 5, 8, 11]

In [213]: list(range(3, 13, 5))
Out[213]: [3, 8]
```

    在 Python 3.X 的版本中输出结果需要用 `list` 函数

11. `reversed` 输出一个逆序排列的序列（包括数列、元组、字符串）

```
In [214]: list1 = [0.0385, 0.0405, 0.0415, 0.042, -0.0425]

In [215]: list1_reversed = reversed(list1)

In [216]: list(list1_reversed)
Out[216]: [-0.0425, 0.042, 0.0415, 0.0405, 0.0385]
```

12. `sorted` 输出一个从小到大排列的序列（包括数列、元组、字符串）

```
In [217]: list1_sorted = sorted(list1)

In [218]: list(list1_sorted)
Out[218]: [-0.0425, 0.0385, 0.0405, 0.0415, 0.042]
```

13. `sum` 求和

```
In [219]: sum(list1)
Out[219]: 0.12
```

14. `zip` 将对象中对应的元素打包成一个元组，并返回有这些元组组成的列表

```
In [220]: en = ['Linear Regression', 'Logistic Regression', 'Decision
Trees']

In [221]: chn = ['线性回归', '逻辑回归', '决策树']

In [222]: list(zip(en, chn))
Out[222]:
[('Linear Regression', '线性回归'),
('Logistic Regression', '逻辑回归'),
('Decision Trees', '决策树')]
```

对于其他内置函数，可以运用 `help` 函数查询具体用法，`help` 函数的运用非常简单，输入的格式就是 `help()` 的括号内输入需要查询的函数，比如

```
In [223]: help(bin)
Help on built-in function bin in module builtins:

bin(number, /)
    Return the binary representation of an integer.

    >>> bin(2796202)
    '0b1010101010101010101010'
```

# 模块的导入

模块在 Python 中扮演非常重要的作用，可以理解为 Python 的扩展工具，换而言之，虽然 Python 在默认情况下提供了一些可用的东西，但是这些默认情况下提供的还远远无法满足编程的需要，于是就有人专门制作了另外一些工具以弥补 Python 的不足，这些工具被称之为 ”模块“

注意，当安装好 Python 之后，有一些模块就已经默认安装好了，这些模块称为 ”标准库“，”标准库“ 中的模块无需另行安装，就可以直接使用，比如 `math` 模块属于标准库，不用安装就可以直接使用，而其他的模块（非标准库）如 Numpy、SciPy、Pandas、Matplotlib 等则需要安装，当运用 Anaconda 时，大量的非标准库也都已经集成安装完成了

## 模块导入的各种方法

1. `import 模块名`

   直接导入整个模块，这种方式比较占用内存

   ```
   In [248]: import math
   ```

2. `import 模块名 as 名称缩写`

   导入整个模块的同时，给该模块取一个别名，往往是用在模块名字比较长的情况下，这样能节省调用该模块时的输入时间

   ```
   In [249]: import matplotlib as mp
   ```

3. `import 模块名.子模块名`

   导入某个模块的子模块，并且给该子模块取一个别名（可选），这样占用的内存也比较少

   ```
   In [250]: import matplotlib.pyplot as plt
   ```

4. `from 模块名 import 函数`

   从特定模块中导入某个或某几个函数（不同函数之间用英文逗号隔开，这样不仅占用的内存比较少，而且这些函数可以直接以函数名称的方式使用）

   ```
   In [251]: from math import exp, log, sqrt

   In [252]: log(10)
   Out[252]: 2.302585092994046

   In [253]: exp(3)
   Out[253]: 20.085536923187668
   ```

5. `from 模块名.子模块名 import 函数`

   与方式 4 很相似，只不过是从特定模块的子模块中导入某个或某几个函数

   ```
   In [254]: from matplotlib.pyplot import figure, plot
   ```