

期末复习提纲(基础篇)

在线视频:

- 《基础篇总结1》
- 《基础篇总结2》
- 《基础篇总结3》

1. 变量

《观看在线视频》

可以给变量取任何名字，只要它遵守以下5条规则：

- (1) 只能是一个词。
- (2) 只能包含字母、数字和下划线。
- (3) 不能以数字开头。
- (4) 不要使用Python保留字（关键字）。
- (5) 常用变量的含义命名。

Python中的常用的保留字如下表所示：

and	as	assert	break
class	continue	def	del
elif	else	except	finally
for	from	global	if
import	in	is	lambda
None	not	or	pass
return	try	while	with
True	False		

变量名是区分大小写的。

2. 表达式

《观看在线视频》

表达式包含“值”和“操作符”，并且总是可以求值。

Python 表达式中也可以使用大量其他操作符。例如，下表列出了Python 的所有数学操作符。

操作符	操作	例子	求值为
**	指数	2 ** 3	8
%	取模/取余数	22 % 8	6
//	整除/商数取整	22 // 8	2
/	除法	22 / 8	2.75
*	乘法	2 * 8	16
-	减法	5 - 2	3

操作符	操作	例子	求值为
+	加法	2 + 2	4

数学操作符的操作顺序（也称为“优先级”）与数学中类似：

- **操作符首先求值；
- 接下来是*、/、//和%操作符，从左到右；
- +和-操作符最后求值，也是从左到右；
- 一般来说都会用括号来改变通常的优先级。

下表从低到高列出了运算符的优先级。同一行中的运算符具有相同优先级，然后运算符的优先级是运算表达式从左到右。

优先级	运算符	描述
1	or	布尔“或”
2	and	布尔“与”
3	not x	布尔“非”
4	in;not in	成员测试
5	<,<=,>,>=,! =,==	比较
6	+, -	加法与减法
7	*,/,%	乘法、除法与取余
8	**	指数

一般情况下，运算符优先级决定了哪个运算符在别的运算符之前计算。但是，如果你想要改变它们默认计算顺序，你得使用圆括号。例如，你想要在一个表达式中让加法在乘法之前计算，那么你就得写成类似(10 + 55) * 13的样子。

```
In [ ]: 5 + 3 * 6 / 2 ** 3 % 2 - 3
```

增强运算符

除了基本赋值运算符“=”外，Python中还有将不同算术运算符与基本赋值运算符相结合在一起的高级赋值运算符（增强运算符）。

运算符	举例	x的结果	功能说明
+=	x = 10 x += 8	18	加法赋值运算符，等价于x = x + 8
-=	x = 10 x -= 8	2	减法赋值运算符，等价于x = x - 8
*=	x = 10 x *= 8	80	乘法赋值运算符，等价于x = x * 8
/=	x = 10 x /= 8	1.25	除法赋值运算符，等价于x = x / 8
//=	x = 10 x //= 8	1	整除赋值运算符，等价于x = x // 8
%=	x = 10 x %= 8	2	取余数赋值运算符，等价于x = x % 8
**=	x = 3 x **= 2	9	幂运算赋值运算符，等价于x = x ** 2

```
In [ ]: a = 10
a *= 10
a
```

3. 赋值语句

在Python中，=表示赋值，==表示比较。

```
In [ ]: a = 10
        b = 20
        a = b
        a
```

```
In [ ]: a = 10
        b = 20
        a == b
```

在Python中，可以同时给多个变量赋值：

<变量1>, ..., <变量N> = <表达式1>, ..., <表达式N>

```
In [ ]: a, b = 10, 20
        print(a)
        print(b)
```

4. 注释

注释是程序员为了便于人类阅读，在代码中加入的说明信息，不会被计算机执行。

注释的两种方法：

- 单行注释以#开头

#这里是单行注释语句，不会被执行

- 多行注释以'''开头和结尾

```
'''
    这里是多行注释语句，
    不会被执行
'''
```

5. 数据类型

整数类型（int），浮点数类型（float），字符串类型（str），布尔类型（Boolean），列表（list），元组（tuple），集合（set），字典（dictionary）。

5.1 数字类型

《观看在线视频》

int, float 为数字类型；可以进行数学运算。

数字类型	例子
整型	-2, -1, 0, 1, 2, 3, 4, 5
浮点型	-1.25, -1.0, -0.5, 0.0, 0.5, 1.0, 1.25

整数没有大小限制，共有4种进制表示：十进制、二进制、八进制和十六进制；

浮点数的数值范围存在限制，小数精度也存在限制，这种限制与在不同计算机系统有关，浮点数可以用科学计数法表示。

```
In [ ]: 10 / 5
```

```
In [ ]: 9 % 5
```

```
In [ ]: 9 // 5
```

```
In [ ]: 9 // 5
```

```
In [ ]: 10.0 + 5
```

```
In [ ]: 2 ** 5
```

数字类型的转换

通常来说，两个数字类型的数字进行算术运算时，运算后生成结果为**最宽类型**：

- (1) 两个整数运算，结果为整数（除法运算“/”除外，其运算结果为小数）。
- (2) 两个浮点数运算，结果为浮点数。
- (3) 整数和浮点数运算，结果为浮点数。

此外，通过内置的数字类型转换函数可以显式地在数字类型之间进行转换：

- (1) `int(x)`，将 x 转换为整数，x 可以是浮点数或字符串。
- (2) `float(x)`，将 x 转换为浮点数，x 可以是整数或字符串。

Python中共有75个内置函数，这些是Python自带的函数，在需要使用时可以直接调用。下表列出了在处理财经数据时常用的Python内置函数：

函数名称	代码示例	结果	功能描述
abs	x = -10 abs(x)	10	求绝对值
float	float('10.8')	10.8	将整数或者字符串转换为浮点数
int	int(10.8)	10	将浮点数截断小数部分后转换为整数或者将只包含数字的字符串转换为整数
len	len('金融科技')	4	获得字符串、列表、元组、集合或字典的长度（即元素个数）
max	max(10,-10.8,3,58,20.5)	58	获得一组数据中最大值
min	min(10,-10.8,3,58,20.5)	-10.8	获得一组数据中最小值
range	list(range(0,6,2))	[0,2,4]	产生一个可迭代对象，在此列中从0开始到6之前（即不包括6），以步长为2产生可迭代对象后利用list函数转换成列表
sum	sum([10,3,-10.8,58])	60.2	获得一组数据的和，在此列中获得列表[10,3, -10.8,58]中的数据之和

5.2 序列类型

《观看在线视频》

`str`, `list`, `tuple` 为序列类型，元素的位置很重要。

```
In [ ]: "swufe" == "sufew"
```

```
In [ ]: [1,2,3] == [3,2,1]
```

```
In [ ]: ('a','b','c') == tuple('abc')
```


`set`, `dictionary` 则不同，里面的元素是无序的。

```
In [ ]: {1, 2, 3} == {3, 2, 1}
```

```
In [ ]: {"a":3, "b":1} == {"b":1, "a":3}
```

序列类型的索引和切片

- 可以通过单个索引值获得特定位置的元素
- 可以通过两个索引值(切片)获得特定范围内的值
- 索引值有正向（从0开始递增）和反向（从-1开始递减）两种，在切片时，也可以混合使用

 No description has been provided for this image


序列中的元素可以通过序号进行访问:

```
In [ ]: idCard = '520125197907167551'
        idCard[16]

In [ ]: idCard[-2]
```

分片操作

如果把序号放在每个元素前，分片操作可以形象的用下图来理解:

 No description has been provided for this image

从上图可以看出，分片所得到的内容包含了冒号左边序号指向的字符“f”，但是没有包括冒号右边序号指向的字符“是”。

```
In [ ]: idCard[6:10]

In [ ]: s = list('abcdef')
        s[:3]

In [ ]: d = (1,2,3,4,5,6,7)
        d[-3:]
```

步长

在分片时，除了可以指定左边界和右边界的序号外（或者不指定而使用缺省值），还可以增加第三个值，即步长:

s[开始:结束:步长]

```
In [ ]: s = list(range(20))
        s[1:10:2]

In [ ]: "abcdefg"[:-1]
```

序列类型有12个通用的操作符和函数:

操作符	描述
x in s	如果x是s的元素，返回True，否则返回False
x not in s	如果x不是s的元素，返回True，否则返回False
s + t	连接s和t
s * n 或 n * s	将序列s复制n次
s[i]	索引，返回序列的第i个元素
s[i: j]	分片，返回包含序列s第i到j个元素的子序列（不包含第j个元素）
s[i: j: k]	步骤分片，返回包含序列s第i到j个元素以k为步数的子序列
len(s)	序列s的元素个数（长度）
min(s)	序列s中的最小元素
max(s)	序列s中的最大元素
s.index(x, i[, j])	序列s中从i开始到j位置中第一次出现元素x的位置
s.count(x)	序列s中出现x的总次数

+或*操作

- + 连接两个相同的序列类型
- * 重复序列类型

```
In [ ]: "abc" + "cde"
```

```
In [ ]: list("abc") + [6,7,8]
```

```
In [ ]: (1, 2, 3) + (5,)
```

```
In [ ]: tuple([1, 2, 3]) + [6, 5, 4]
```

```
In [ ]: "abc" * 3
```

```
In [ ]: "abc" * "def"
```

```
In [ ]: [4, 5, 6] * 3
```

`int`, `float`, `str`, `boolean`, `tuple` 都是不可变数据类型。

```
In [ ]: 10 = 8
```

```
In [ ]: "swufe"[2] = "c"
```

```
In [ ]: (1, 2, 3)[2] = 6
```

```
In [ ]: True = 123
```

`list`, `set`, `dictionary` 都是可变数据类型。

```
In [ ]: s = [1, 2, 3]
s[2] = 10
s
```

```
In [ ]: s = {7, 8, 9}
s.add(20)
s
```

```
In [ ]: d = {"a":20}
d['a'] = 10
d['b'] = 30
d
```

每种类型都有对应的函数，将可以转换的其他数据类型转换成该类型：

- `int()`
- `float()`
- `str()`
- `list()`
- `tuple()`
- `set()`
- `dict()`

5.3 组合数据类型

《观看在线视频》

`list`, `tuple`, `set`, `dictionary` 是由多个元素构成，元素之间用逗号隔开，并且元素可以是其他各种类型，所以又称为组合数据类型。

新增元素

- list: `append()`, `extend()`, `insert()`
- tuple: 没有
- set: `add()`
- dictionary: 直接给不存在的key赋值

```
In [ ]: s = [1,2,3]
s.append("abc")
s
```

```
In [ ]: s = [1,2,3]
s.extend("abc")
s
```

```
In [ ]: s = [1,2,3]
s.insert(1, "abc")
s
```

```
In [ ]: s = {1,2,3}
s.add("abc")
```

```
In [ ]: d = {"a":10}
d["b"] = 20
d
```

删除元素

- list: `remove()`, `del`, `clear()`
- tuple: 没有
- set: `remove()`, `discard()`, `pop()`, `clear()`
- dictionary: `del`, `pop()`, `popitem()`, `clear()`

下面的列子，请注意索引值（下标）和元素的区别

```
In [ ]: s = [1,2,3]
del s[1]
s
```

```
In [ ]: s = {1,2,3}
s.remove(2)
s
```

```
In [ ]: d = {"a":2, "b":3}
del d["b"]
d
```

`len()` 函数

- 对于 `str`, `list`, `tuple`, `set` 来说，得到的是元素的个数（长度）；
- 对于 `dictionary` 来说，得到的是 `key` 的个数（字典中键值对的个数）。

```
In [ ]: len([1, 2, 3])
```

```
In [ ]: len({"a":1, "b":2})
```

`in` 和 `not in`

- 对于 `str`, `list`, `tuple`, `set` 来说，判断一个值是否在字符串，列表，元组和集合中
- 对于 `dictionary` 来说，判断一个值是否在字典的 `key` 当中，等同于 `keys()`；如果要判断一个值是否在字典的值中，使用 `values()`

```
In [ ]: 'a' in ('a','b','c')
```

```
In [ ]: 'wu' not in 'swufe'
```

```
In [ ]: 'a' in {'a':1,'b':2}
```

```
In [ ]: 1 not in {"a":1,"b":2}.values()
```

创建组合数据类型：

- 空列表： `[]` , `list()`
- 空元组： `()` , `tuple()`
- 空集合： ~~`{}`~~，应该用 `set()`
- 空字典： `{}` , `dict()`

5.4 字符串str

《观看在线视频》

5.4.1 基本概念

在Python中，字符串是用两个双引号 `" "` 或者单引号 `' '` 括起来的一个或多个字符。

注意区别字符串和变量，下面一个是变量，一个是字符串：

- `swufe`
- `'swufe'`

当字符串中有引号时，可以使用另外一类引号将字符串括起来，也可以使用转义字符：

```
In [ ]: sayHello = '他说："你好啊"。'  
sayHello
```

转义字符还有：

- `\n`
- `\t`
- `\\`

等

字符串前面可以使用`r`或者`f`。

对于字符串类型，可以通过 `str` 将其他类型的数据转换为字符串。

```
In [ ]: str(10.8), str(100), str(123e+10)
```

5.4.2 用户输入函数: `input()`

函数等待用户在键盘上输入一些文本，并按下回车键。这个函数的值为一个字符串，即用户输入的文本。

```
In [ ]: myInputVal = input("请输入你的姓名: ")
```

5.4.3 打印函数: `print()`

`print()` 函数将括号内的值显示在屏幕上。并且默认换行：

```
In [ ]: print("西南财经大学")  
print(123456)
```

如果打印后不想换行，可以使用下面的方法：

```
In [ ]: print("Hello World!", end="")  
print("西南财经大学", end="_")  
print(123456)
```


5.4.4 字符串表达式函数: eval()

`eval()` 函数用来执行一个字符串表达式，并返回表达式的值。

```
In [ ]: eval('10 + 8')
```

```
In [ ]: x = 10
eval('x ** 2 + 8')
```

5.4.5 字符串方法 `upper()`、`lower()`、`isupper()` 和 `islower()`

`upper()` 和 `lower()` 字符串方法返回一个新字符串，其中原字符串的所有字母都被相应地转换为大写或小写。字符串中非字母字符保持不变。

```
In [ ]: sayHello = 'Hello world!'
sayHello.upper()
```

```
In [ ]: sayHello = 'Hello world!'
sayHello.lower()
```

```
In [ ]: sayHello = 'Hello world!'
sayHello.isupper()
```

```
In [ ]: sayHello = 'Hello world!'
sayHello.islower()
```

5.4.6 isX 字符串方法

下面是一些常用的isX 字符串方法：

- `isalpha()` 返回True，如果字符串只包含字母，并且非空；
- `isalnum()` 返回True，如果字符串只包含字母和数字，并且非空；
- `isdecimal()` 返回True，如果字符串只包含数字字符，并且非空；
- `isspace()` 返回True，如果字符串只包含空格、制表符和换行，并且非空；
- `istitle()` 返回True，如果字符串仅包含以大写字母开头、后面都是小写字母的单词。

5.4.7 字符串方法 `startswith()` 和 `endswith()`

`startswith()` 和 `endswith()` 方法返回True，如果它们所调用的字符串以该方法传入的字符串开始或结束。否则，方法返回False。

5.4.8 字符串方法 `join()` 和 `split()`

如果有一个字符串列表，需要将它们连接起来，成为一个单独的字符串，`join()` 方法就很有用。`join()` 方法在一个字符串上调用，参数是一个字符串列表，返回一个字符串。返回的字符串由传入的列表中每个字符串连接而成：

```
In [ ]: ','.join(['管理部', '人事部', '销售部'])
```

`split()` 方法和 `join()` 正好相反，用于将字符串分隔成列表。

```
In [ ]: '管理部,人事部,销售部'.split(',')
```

5.4.9 用 `strip()`、`rstrip()` 和 `lstrip()` 删除空白字符

有时候你希望删除字符串左边、右边或两边的空白字符（空格、制表符和换行符）。

- `strip()` 字符串方法将返回一个新的字符串，它的开头或末尾都没有空白字符。
- `lstrip()` 方法将相应删除左边的空白字符。
- `rstrip()` 方法将相应删除右边的空白字符。

```
In [ ]: sayHello = ' Hello World '
print('-' + sayHello.strip() + '-')
print('-' + sayHello.lstrip() + '-')
print('-' + sayHello.rstrip() + '-')
print('-' + sayHello + '-')
```

5.4.10 用 `replace()` 函数替换

Python `replace()` 方法把字符串中的 `old` (旧字符串) 替换成 `new`(新字符串), 如果指定第三个参数`max`, 则替换不超过 `max` 次。

```
str.replace(old, new[, max])
```

```
In [ ]: s = "this is string example....wow!!! this is really string"
print(s.replace("is", "was"))
print(s.replace("is", "was", 3))
```

5.4.11 字符串 `format()` 方法

字符串 `format()` 方法的基本使用格式是:

```
'...{<参数序号>}...'.format(<逗号分隔的参数列表>)
```

在模板字符串 `'...{ }...'` 中的花括号相对位置 (例如`{}`) 或指定位置 (例如`{1}`) 来制定替换目标及要插入的参数。

在 `format()` 方法中模板字符串的槽除了包括参数序号, 还可以通过冒号:分隔后, 包括格式控制信息。此时, 槽的内部样式如下:

```
{<参数序号>[:<格式控制标记>]}
```

其中, 格式控制标记用来控制参数显示时的格式。格式控制标记包括: `<填充><对齐><宽度>[,<.精度><类型>` 个字段, 这些字段都是可选的, 可以组合使用, 这里按照使用方式逐一介绍:

:	<填充>	<对齐>	<宽度>	,	<.精度>	<类型>
引导符号	用于填充的单个字符	< 左对齐 > 右对齐 ^ 居中对齐	槽的设置 输出宽度	数字的 千位分隔符 只用于整数和浮点数	浮点数 小数部分的精度 或 字符串 的最大输出长度	整数类型 b,c,d,o,x,X 浮点数类型 e,E,f,%

```
In [ ]: s = "Python"
"{0:30}".format(s)
```

```
In [ ]: "{0:>30}".format(s)
```

```
In [ ]: "{0:*^30}".format(s)
```

```
In [ ]: "{0:.2f}".format(12345.67890)
```

5.4.12 `ord()` 和 `chr()` 函数

`ord()` 函数以一个字符 (长度为1的字符串) 作为参数, 返回对应的 ASCII 数值, 或者 Unicode 数值, 如果所给的 Unicode 字符超出了你的 Python 定义范围, 则会引发一个 `TypeError` 的异常。

```
In [ ]: ord('a')
```

`chr()` 用一个整数作参数, 返回一个对应的字符。

```
In [ ]: chr(97)
```

5.5 列表list

《观看在线视频》

5.5.1 用 `sort()` 方法将列表中的值排序

数值的列表或字符串的列表，能用 `sort()` 方法排序。`sort()` 方法当场对列表排序。

```
In [ ]: numberList = [2, 10, 3.14, -20, -50]
        numberList.sort()
        numberList
```

也可以指定 `reverse` 关键字参数为 `True`，让 `sort()` 按逆序排序。

```
In [ ]: numberList.sort(reverse=True)
        numberList
```

`sort()` 方法有一个 `key` 的关键字参数，可以指定排序方法，例如：

```
In [ ]: someWords = ['Alice', 'ants', 'Bob', 'badgers', 'Carol', 'cats']
        someWords.sort(key=len)
        someWords #按照每一项的长度进行排序
```

```
In [ ]: someWords.sort(key=lambda x: x[1])
        someWords #按照每一项的第二个字符进行排序
```

5.5.2 列表的操作

函数或方法	描述
<code>ls[i] = x</code>	替换列表 <code>ls</code> 第 <code>i</code> 数据项为 <code>x</code>
<code>ls[i: j] = lt</code>	用列表 <code>lt</code> 替换列表 <code>ls</code> 中第 <code>i</code> 到 <code>j</code> 项数据（不含第 <code>j</code> 项，下同）
<code>ls[i: j: k] = lt</code>	用列表 <code>lt</code> 替换列表 <code>ls</code> 中第 <code>i</code> 到 <code>j</code> 以 <code>k</code> 为步的数据
<code>del ls[i: j]</code>	删除列表 <code>ls</code> 第 <code>i</code> 到 <code>j</code> 项数据，等价于 <code>ls[i: j]=[]</code>
<code>del ls[i: j: k]</code>	删除列表 <code>ls</code> 第 <code>i</code> 到 <code>j</code> 以 <code>k</code> 为步的数据
<code>ls += lt</code> 或 <code>ls.extend(lt)</code>	将列表 <code>lt</code> 元素增加到列表 <code>ls</code> 中
<code>ls *= n</code>	更新列表 <code>ls</code> ，其元素重复 <code>n</code> 次
<code>ls.append(x)</code>	在列表 <code>ls</code> 最后增加一个元素 <code>x</code>
<code>ls.clear()</code>	删除 <code>ls</code> 中所有元素
<code>ls.copy()</code>	生成一个新列表，复制 <code>ls</code> 中所有元素
<code>ls.insert(i, x)</code>	在列表 <code>ls</code> 第 <code>i</code> 位置增加元素 <code>x</code>
<code>ls.pop(i)</code>	将列表 <code>ls</code> 中第 <code>i</code> 项元素取出并删除该元素
<code>ls.remove(x)</code>	将列表中出现第一个元素 <code>x</code> 删除
<code>ls.reverse()</code>	列表 <code>ls</code> 中元素反转
<code>ls.index(x)</code>	找出某个值第一个匹配项的索引位置

5.5.3 二维列表

列表中的元素可以是另外一个列表，称为二维列表。

```
In [ ]: employees = [{"10932", "张三", "男"}, {"10933", "李四", "女"}, {"10934", "王五", "男"}]
        employees
```

获取列表中的元素：

```
In [ ]: employees[1]
```

```
In [ ]: employees[1][1]
```

5.6 元组tuple

元组和列表的最大区别是，元组不能修改。创建单个元素的元组：

```
In [ ]:
In [ ]: (2,)
In [ ]: (10)+ (8)
In [ ]: (10,)+ (8,)
```

多重赋值，不仅列表可用，元组也可以：

```
In [ ]: x, y = [10,20]
        print(x)
        print(y)
In [ ]: a, b = ("西南财经大学","电子科技大学")
        print(a)
        print(b)
```

5.7 集合set

《观看在线视频》

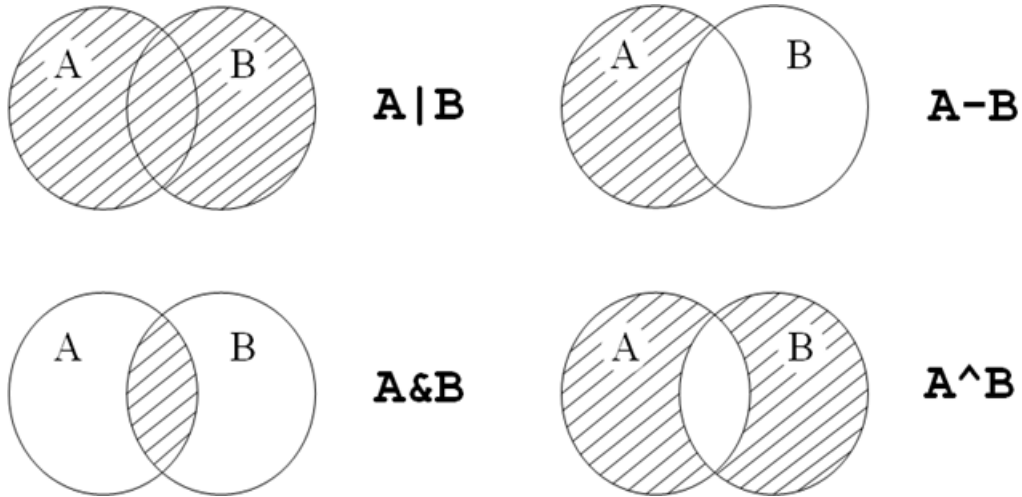
集合含0个或多个数据项的无序组合。集合中元素不可重复，元素类型只能是固定数据类型，例如：整数、浮点数、字符串、元组等，列表、字典和集合类型本身都是可变数据类型，不能作为集合的元素出现。

集合类型的操作符、函数或方法

集合类型有10个操作符：

操作符	描述
$S - T$	返回一个新集合，包括在集合S中但不在集合T中的元素
$S -= T$	更新集合S，包括在集合S中但不在集合T中的元素
$S \& T$	返回一个新集合，包括同时在集合S和T中的元素
$S \&= T$	更新集合S，包括同时在集合S和T中的元素。
$S \wedge T$	返回一个新集合，包括集合S和T中元素，但不包括同时在其中的元素
$S \wedge= T$	更新集合S，包括集合S和T中元素，但不包括同时在其中的元素
$S T$	返回一个新集合，包括集合S和T中所有元素
$S = T$	更新集合S，包括集合S和T中所有元素
$S \leq T$	如果S与T相同或S是T的子集，返回True，否则返回False，可以用 $S < T$ 判断S是否是T的真子集
$S \geq T$	如果S与T相同或S是T的超集，返回True，否则返回False，可以用 $S > T$ 判断S是否是T的真超集

上述操作符表达了集合类型的4种基本操作，交集 (&)、并集 (|)、差集 (-)、补集 (^)，操作逻辑与数学定义相



同：

集合类型有10个操作函数或方法：

函数或方法	描述
S.add(x)	如果数据项x不在集合S中，将x增加到s
S.clear()	移除S中所有数据项
S.copy()	返回集合S的一个拷贝
S.pop()	随机返回集合S中的一个元素，如果S为空，产生KeyError异常
S.discard(x)	如果x在集合S中，移除该元素；如果x不在，不报错
S.remove(x)	如果x在集合S中，移除该元素；不在产生KeyError异常
S.isdisjoint(T)	如果集合S与T没有相同元素，返回True
len(S)	返回集合S元素个数
x in S	如果x是S的元素，返回True，否则返回False
x not in S	如果x不是S的元素，返回True，否则返回False

```
In [ ]: stock_name_set1 = {'建设银行', '浦发银行', '招商银行', '农业银行'}
stock_name_set2 = {'工商银行', '中国银行', '招商银行', '农业银行'}
stock_name_set1 | stock_name_set2

In [ ]: stock_name_set1 - stock_name_set2

In [ ]: stock_name_set1 & stock_name_set2

In [ ]: stock_name_set1 ^ stock_name_set2
```

5.8 字典dict

《观看在线视频》

字典数据(dictionary)类型提供了一种灵活的访问和组织数据的方式。像列表一样，“字典”是许多值的集合。但不像列表的下标，*字典的索引可以使用许多不同数据类型，不只是整数。字典的索引被称为“键”，键及其关联的值称为“键-值*”对。

Python语言中的字典可以通过大括号({ })建立，建立模式如下：

```
{<键1>:<值1>, <键2>:<值2>, ... , <键n>:<值n>}
```

其中，键和值通过冒号连接，不同键值对通过逗号隔开。

键通常是字符串，但它还可以是Python中其他任意的不可变类型：布尔型、整型、浮点型、元组、字符串，以及其他一些不可变的类型。字典本身是可变的，因此可以增加、删除或修改其中的键值对。字典的键必须保证互不相同。

5.8.1 访问字典值：

字典最主要的用法是查找与特定键相对应的值，这通过索引符号来实现。

一般来说，字典中键值对的访问模式如下，采用中括号格式：

<值> = <字典变量>[<键>]

字典中对某个键值的修改可以通过中括号的访问和赋值实现：

```
In [ ]: d = {"name": "zhang", "age": 20}
        d["name"]

In [ ]: d["zhang"]

因为字典是不排序的，所以不能像列表那样切片。

In [ ]: d["name": "age"]
```

5.8.2. 字典类型的操作

函数和方法	描述
dic.keys()	返回所有的键信息
dic.values()	返回所有的值信息
dic.items()	返回所有的键值对
dic.get(k,d)	键存在则返回相应值，否则返回默认值
dic.setdefault(k,d)	键存在则返回相应值，否则返回默认值，并插入k和d构成的键值对
dic.pop(k,d)	键存在则返回相应值，同时删除键值对，否则返回默认值
dic.popitem()	随机从字典中取出一个键值对，以元组(key, value)形式返回
dic.clear()	删除所有的键值对
del dic[key]	删除字典中某一个键值对
key in d	如果键在字典中返回True，否则返回False

```
In [ ]: d = {"name": "zhang", "age": 20}
        d.keys()

In [ ]: d.values()

In [ ]: d.items()

In [ ]: d.get("age", 10)

In [ ]: d.get("gender")

In [ ]: d.get("gender", "male")
```

5.8.3 字典的格式化字符串

如果数据存储在字典中，可以通过字典的键来指定占位符替换的值。这种方式更加清晰。

```
In [ ]: stock_info = {'code': '600000', 'name': '浦发银行', 'price': 11.54}
        print('{name} 的代码是 {code}，收盘价为 :{price:.2f}'.format(**stock_info))
```

5.9 布尔类型

《观看在线视频》

“布尔”数据类型只有两种值： True 和 False 。

5.9.1 比较运算符

“比较操作符”比较两个值，求值为一个布尔值。下表列出了比较操作符：

运算符	举例	结果	功能说明
<	10 < 8	False	小于
>	10 > 8	True	大于
<=	10 <= 8	False	小于等于
>=	10 >=10	True	大于等于
==	10 == 8	False	等于
!=	10 != 8	True	不等于

```
In [ ]: 1 > 10
```

```
In [ ]: 5 <= 8
```

在Python中，也可以像数学中一样表示范围比较：

```
In [ ]: 5 < 10 and 5 > 3
```

可以写成：

```
In [ ]: 3 < 5 < 10
```

5.9.2 布尔运算

Python中3个布尔操作符（`and`、`or` 和 `not`）用于比较布尔值。像比较操作符一样，它们将这些表达式求值为一个布尔值。

`and` 和 `or` 操作符总是接受两个布尔值（或表达式），所以它们被认为是“二元”操作符。

和 `and` 和 `or` 不同，`not` 操作符只作用于一个布尔值（或表达式）。

- `and` :如果两个布尔值都为 `True`，`and` 操作符就将表达式求值为 `True`，否则求值为 `False`。
- `or` :只要有一个布尔值为真，`or` 操作符就将表达式求值为 `True`。如果都是 `False`，所求值为 `False`。
- `not` :求值为相反的布尔值。

```
In [ ]: 1 == 2 and 1 == 1
```

```
In [ ]: 1 == 1 and 5 > 3
```

```
In [ ]: 1 == 1 or 3 != 3
```

```
In [ ]: not 1 == 1
```

6. Python控制结构

程序的基本结构包括：顺序、选择、循环结构

6.1 代码块

《观看在线视频》

一些代码行可以作为一组，放在“代码块”中。可以根据代码行的缩进，知道代码块的开始和结束。代码块有3条规则。

- (1) 缩进增加时，代码块开始。
- (2) 代码块可以包含其他代码块。
- (3) 缩进减少为零，或减少为外面包围代码块的缩进，代码块就结束了。

Python语言采用严格的“缩进”来表明程序的格式框架。缩进指每一行代码开始前的空白区域，用来表示代码之间的包含和层次关系。

Python 对于缩进的数量没有严格规定，比较常见的是使用 4 个空格或者 1 个制表符。但最好不要在同一段代码中混合使用空格和制表符，这样可能会引起语法错误。

1个缩进 = 4个空格

缩进是Python语言中表明程序框架的唯一手段。

在写代码过程中，遇到冒号:就代表新的语句块开始，也就是需要缩进，例如：

`if, elif, else, for, while, def, try, except, finally, with` 等关键字的语句后面

6.2 分支结构

分支结构是程序根据条件判断结果而选择不同向前执行路径的一种运行方式，包括单分支结构和二分支结构。由二分支结构会组合形成多分支结构。

6.2.1 if 语句

`if` 语句的子句（也就是紧跟 `if` 语句的语句块），将在语句的条件为 `True` 时执行。如果条件为 `False`，子句将跳过。在Python中，`if` 语句包含以下部分：

- `if` 关键字；
- 条件（即求值为 `True` 或 `False` 的表达式）；
- 冒号；
- 在下一行开始，缩进的代码块（称为`if`子句）

```
if condition:
    statements
```

```
In [ ]: idCard = "520125197907167551"
genderFlag = int(idCard[16])
if genderFlag % 2 == 1:
    print("男")
print("不管真假，都会打印我。")
```

6.2.2 带 `else` 的 `if` 语句

`if` 子句后面有时候也可以跟着 `else` 语句。只有 `if` 语句的条件为 `False` 时，`else` 子句才会执行。在英语中，`else` 语句读起来可能是：“如果条件为真，执行这段代码。否则，执行那段代码”。`else` 语句不包含条件，在代码中，`else` 语句中包含下面部分：

- `else` 关键字；
- 冒号；
- 在下一行开始，缩进的代码块（称为 `else` 子句）。

```
if condition:
    statements1
else:
    statements2
```

```
In [ ]: idCard = "520125197907167551"
genderFlag = int(idCard[16])
if genderFlag % 2 == 1:
    print("男")
else:
    print("女")
```

```
In [ ]: idCard = "520125197907167551"
genderFlag = int(idCard[16])
if genderFlag % 2 == 1:
    print("男")
```



```
else:
    print("女")
print("不管真假，都会打印我。")
```

6.2.3 带 elif 的 if 语句

虽然只有 `if` 或 `else` 子句会被执行，但有时候可能你希望，“许多”可能的子句中有一个被执行。`elif` 语句是“否则如果”，总是跟在 `if` 或另一条 `elif` 语句后面。它提供了另一个条件，仅在前面的条件为 `False` 时才检查该条件。在代码中，`elif` 语句总是包含以下部分：

- `elif` 关键字；
- 条件（即求值为 `True` 或 `False` 的表达式）；
- 冒号；
- 在下一行开始，缩进的代码块（称为 `elif` 子句）

```
if condition1:
    statements1
elif condition2:
    statements2
else:
    statements3
```

```
In [ ]: taxIncome = 120000.00
if taxIncome <= 36000:
    taxRate = 0.03
elif taxIncome <= 144000:
    taxRate = 0.1
elif taxIncome <= 300000:
    taxRate = 0.2
elif taxIncome <= 420000:
    taxRate = 0.25
elif taxIncome <= 660000:
    taxRate = 0.30
elif taxIncome <= 960000:
    taxRate = 0.35
else:
    taxRate = 0.4
print(taxRate)
```

6.2.4 什么是真值（True）

控制语句的开始部分通常是“条件表达式”。分支语句通过条件表达式确定选择执行的代码块；循环语句通过条件表达式决定是否执行循环体中的代码块。通常来说，条件表达式的结果为一个布尔值，即 `True` 或者 `False`，例如关系运算或者逻辑运算的表达式。控制语句根据条件是 `True` 还是 `False` 来决定做什么。

当表达式的最终值是下表中的某个值时，会被认作是假值：

对象类型	值	对象类型	值
布尔	<code>False</code>	null 类型	<code>None</code>
整型	<code>0</code>	浮点型	<code>0.0</code>
空字符串	<code>"</code>	空列表	<code>[]</code>
空元组	<code>()</code>	空字典	<code>{}</code>
空集合	<code>set()</code>		

除了上表所列出来的值，其他的值都会被认作是真值。

6.2.5 三元表达式

Python中的三元表达式可以将 `if-else` 语句放到一行里。语法如下：

```
True_expression if condition else False_expression
```

真值表达式或假值表达式可以是任何Python代码。它和下面的代码效果相同：

```
if condition:
    a = True_expression
else:
    a = False_expression
```

```
In [ ]: numInIDCard = 6
        '男' if numInIDCard % 2 == 1 else '女'
```

6.3 while 循环语句

《观看在线视频》

利用 `while` 语句，可以让一个代码块一遍又一遍的执行。*只要`while` 语句的条件为`True`，`while` 子句中的代码就会执行*。在代码中，`while` 语句总是包含下面几部分：

- 关键字；
- 条件（求值为 `True` 或 `False` 的表达式）；
- 冒号；
- 从新行开始，缩进的代码块（称为 `while` 子句）。

Python当中，while实现循环的方法如下：

```
while condition:
    statements
```

`while` 语句看起来和`if` 语句类似。不同之处是它们的行为。`if` 子句结束时，程序继续执行 `if` 语句之后的语句。但在 `while` 子句结束时，程序执行跳回到 `while` 语句开始处。`while` 子句常被称为“while 循环”，或就是“循环”。例如下面的代码：

```
In [ ]: userCounter = 0
        while userCounter < 10:
            userCounter += 1
            print("第{}个用户".format(userCounter))
```

`while` 语句也有一种使用保留字 `else` 的扩展模式,如果 `while` 循环正常结束（没有使用 `break` 跳出），程序将进入到可选的 `else` 段：

```
while condition:
    statements1
else:
    statements2
```

```
In [ ]: s, idx = "SWUFE", 0
        while idx < len(s):
            print("循环中: " + s[idx])
            idx += 1
        else:
            print("循环正常结束")
```

6.4 for 循环语句

在条件为 `True` 时，`while` 循环就会继续循环（这是它的名称的由来）。但如果你想让一个代码块执行固定次数，该怎么办？可以通过 `for` 循环语句和 `range()` 函数来实现。

在代码中，`for` 语句看起来像 `for i in range(5):` 这样，总是包含以下部分：

- `for` 关键字；
- 一个变量名；
- `in` 关键字；
- 调用 `range()` 方法，最多传入3 个参数；
- 冒号；
- 从下一行开始，缩退的代码块（称为`for`子句）。

Python中，for循环实现方法如下：

```
for item in iterable_obj:
    statements
```

其中，item又称为循环变量，可以是任何合法的变量名称。

可遍历的数据结构包括：

- `range`，将range中的每一个数字（不包括最后一个数字）赋值给item进行循环体运算。
- `str`，将字符串中的每个字符赋值给item进行循环体运算。
- `list`，将列表list中的每个元素赋值给item进行循环运算。
- `tuple`，将元组tuple中的每个元素赋值给item进行循环运算。
- `set`，将集合set中的每一个元素赋值给item进行循环运算。
- `dictionary`，将字典dict中的每一个键key赋值给item进行循环运算。
- `file`，将文件中的每一行赋值给item进行循环运算。

```
In [ ]: total = 0
        for num in range(101):
            total = total + num
        print(total)
```

```
In [ ]: for item in "This Is A Test Sentence.":
        if item >= "A" and item <="Z":
            print(item)
```

```
In [ ]: result = ""
        for item in ['西南财经大学', '电子科技大学', '四川大学']:
            result = result + item[2]
        result
```

```
In [ ]: for item in tuple('abc'):
        print(item)
```

```
In [ ]: result = 0
        for item in {"作业1", "作业2", "作业3"}:
            result += int(item[2])
        print(result)
```

```
In [ ]: result = 0
        d = {"作业1":11, "作业2":15, "作业3":10}
        for item in d:
            result += d[item]
        print(result)
```

```
In [ ]: with open("Documents/employees.txt", encoding='utf8') as f:
        for item in f:
            print(item.strip())
```

range() 的开始、停止和步长参数

某些函数可以用多个参数调用，参数之间用逗号分开，`range()`就是其中之一。这让你能够改变传递给 `range()` 的整数，实现各种整数序列，包括从0以外的值开始。

```
In [ ]: for i in range(1, 6):
        print(i)
```

`range()`函数也可以有第三个参数。前两个参数分别是起始值和终止值，第三个参数是“步长”。步长是每次迭代后循环变量增加的值。

```
In [ ]: for i in range(0, 10, 2):
        print(i)
```

在为 `for` 循环生成序列数据方面，`range()` 函数很灵活。举例来说，甚至可以用负数作为步长参数，让循环计数逐渐减少，而不是增加。

```
In [ ]: for i in range(5, -1, -1):
        print(i)
```

`for` 语句也有一种使用保留字 `else` 的扩展模式,如果 `for` 循环正常结束（没有使用 `break` 跳出），程序将进入到可选的 `else` 段：

```
for item in iterable_obj:
    statements1
else:
    statements2
```

```
In [ ]: s, idx = "SWUFE", 0
for idx in range(len(s)):
    print("循环中: " + s[idx])
else:
    print("循环正常结束")
```

6.5 break 语句

`break` 用来跳出最内层 `for` 或 `while` 循环，脱离该循环后程序从循环后代码继续执行。

```
In [ ]: nameStr = "swufe"
#nameStr = "abcde"
for c in nameStr:
    if c == 'u':
        print("在{}中找到u了.".format(nameStr))
        break
else:
    print("{}中没有u字母.".format(nameStr))
```

每个 `break` 语句只有能力跳出当前层次循环。

```
In [ ]: for c in "swufe":
    for i in range(10):
        print(c, end="")
        if c=='u':
            break
```

6.6 continue 语句

像 `break` 语句一样，`continue` 语句用于循环内部。如果程序执行遇到 `continue` 语句，就会马上*跳回到循环开始处，重新对循环条件求值*（这也是执行到达循环末尾时发生的事情）。

试试下面的用户验证：

```
In [ ]: while True:
    print('请输入用户名: ')
    name = input()
    if name != 'zhangsan':
        continue
    print('Hello, zhangsan. 密码是什么? (学校简称.)')
    password = input()
    if password == 'swufe':
        break
print('用户验证成功.')
```

6.7 并列遍历：zip 函数

《观看在线视频》

Python 中提供了一个非常有用的函数：zip 函数。这个函数可以将这些序列并排的元素配对成元组后，组成一个新的可迭代对象。其语法格式如下：

`zip(*iterables)`

```
In [ ]: stock_code = ('600000', '600036', '600048', '000001')
stock_name = ('浦发银行', '招商银行', '保利地产', '平安银行')
close_price = [11.54, 36.84, 16.61, 14.49]
```

```
for item in zip(stock_code, stock_name, close_price):
    print('{1} 的代码是: {0}, 收盘价: {2:.2f} 元'.format(item[0], item[1], item[2]))
```

6.8 简单循环的替身：map 函数

在 Python 中，可以利用 map 函数简化这类代码。其语法格式如下：

```
map(func, *iterables)
```

map 函数有两个参数，func 是一个函数，也就是对序列中元素进行的操作。例如，例 4-8 的代码可以进行如下改写：

```
In [ ]: amt_str = input(' 请输入销售数量 ( 用逗号隔开 ): ')
amt_str_list = amt_str.split(',')
amt_int_map_obj = map(int, amt_str_list)
print(' 总销售数量为: {} 件 '.format(sum(amt_int_map_obj)))
```

6.9 序号和元素都需要时应用 enumerate 函数

在 Python 中，通过 enumerate 函数可以将代码写得更加简洁。

enumerate 函数的语法格式如下：

```
enumerate(iterable, start=0)
```

```
In [ ]: goods_list = [['华为笔记本', 1000], ['联想笔记本', 800], ['苹果笔记本', 600]]
for idx, goods in enumerate(goods_list):
    print('第{}名{}销量: {}台'.format(idx + 1, goods[0], goods[1]))
```

```
In [ ]:
```

6.10 选择函数filter

与 map 函数类似，filter 函数也可以接受一个返回结果为布尔值的函数和可迭代对象作为实参。其作用是对可迭代对象中每一个元素都应用到传入的函数中，并将函数返回为 True 的元素添加到结果中，即对可迭代对象中的元素进行过滤。

利用 filter 函数筛选出列表中上涨的股票信息。

```
In [ ]: stock_list = [['招商银行', '0.0124'], ['兴业银行', '0.0111'],
                    ['中国银行', '-0.0078'], ['宁波银行', '0.0036'],
                    ['浦发银行', '0.0000'], ['工商银行', '-0.0071']]
stock_filter = filter(lambda x: float(x[1]) > 0, stock_list)
```

```
In [ ]: type(stock_filter)
```

```
In [ ]: stock_list = list(stock_filter)
stock_list
```

7. 推导式

《观看在线视频》

推导式是从可迭代对象中快速简洁地创建数据类型的一种方法。它使得你用优美简短的代码就能实现循环甚至条件判断。

利用推导式可以得到列表、字典、集合以及生成器。

7.1 列表推导式

列表推导式会产生一个新的列表，其语法形式如下所示：

```
[expr for item in iterable_obj]
```

或

```
[expr for item in iterable_obj if condition]
```

```
In [ ]: amt_str = input('请输入销售数量(用逗号隔开):')
amt_str_list = amt_str.split(',')
amt_int_list = [int(amt) for amt in amt_str_list if '-' not in amt]
print('总销售数量为: {}件'.format(sum(amt_int_list)))
```

7.2 字典推导式

与列表类似，字典也可以使用推导式来生成。字典推导式利用每次迭代收集表达式的键和值结果，并将该键值对添加到新的字典中。语法形式如下：

```
{key_expression : value_expression for expression in iterable}
```

或

```
{key_expression : value_expression for expression in iterable if condition}
```

```
In [ ]: stock_code = ['600000', '600036', '600048', '000001']
close_price = [11.54, 36.84, 16.61, 14.49]
stock_dic = {code: price for code, price in zip(stock_code, close_price) if code.startswith('60')}
stock_dic
```

7.3 集合推导式

与字典推导式类似，集合推导式使用的也是花括号“{}”。不同的是，集合推导式在 for 关键字前的表达式结果是一个元素，而不是键值对。语法形式如下：

```
{expression for item in iterable}
```

或

```
{expression for item in iterable if condition}
```

```
In [ ]: stock_code = ['600000', '600036', '600048', '000001']
sh_stock_code = {'sh_' + item for item in stock_code if item.startswith('60')}
sh_stock_code
```

8. 函数

《观看在线视频》

Python中，可以使用函数来定义重复的功能。函数是一段具有特定功能的、可重用的语句组，用函数名来表示并通过函数名进行完成功能调用。

函数也可以看作是一段具有名字的子程序，可以在需要的地方调用执行，不需要在每个执行地方重复编写这些语句。每次使用函数可以提供不同的参数作为输入，以实现对不同数据的处理；函数执行后，还可以反馈相应的处理结果。

8.1 def 语句和参数

如果调用 `print()` 或 `len()` 函数，你会传入一些值，放在括号之间，在这里称为“参数”

Python定义一个函数使用def保留字，语法形式如下：

```
def name(arg1, arg2, ...,argN):
    statements
    return value
```

函数命名规范和变量命名一样（必须使用字母或者下划线_开头，仅能含有字母、数字和下划线）。

```
In [ ]: def myFunc(x, y):
        return x + y

myFunc(10, 2)
```

8.2 返回值和 return 语句

如果调用 `len()` 函数，并向它传入像 'Hello' 这样的参数，函数调用就求值为整数 5。这是传入的字符串的长度。一般来说，函数调用求值的结果，称为函数的“返回值”。

用 `def` 语句创建函数时，可以用 `return` 语句指定应该返回什么值。`return` 语句包含以下部分：

- `return` 关键字；
- 函数应该返回的值或表达式。

如果在 `return` 语句中使用了表达式，返回值就是该表达式求值的结果。

```
In [ ]: def echo(anything):  
        return anything + ' ' + anything  
  
echo("你好")
```

```
In [ ]: def myFunc(x, y):  
        if x > y:  
            return x  
        else:  
            return y  
myFunc(10, 2)
```

```
In [ ]: myFunc(5, 20)
```

8.3 位置参数传值

Python 处理参数的方式要比其他语言更加灵活。其中，最熟悉的参数类型是位置参数，传入参数的值是按照顺序依次复制过去的。

```
In [ ]: def myMinus(num1, num2):  
        return num1 - num2  
myMinus(100, 20)
```

8.4 关键字参数传值

为了避免位置参数带来的混乱，调用参数时可以指定对应参数的名字，甚至可以采用与函数定义不同的顺序调用：

```
In [ ]: myMinus(num1=100, num2=20)
```

```
In [ ]: myMinus(num2=20, num1=100)
```

8.5 指定默认参数值

当调用方没有提供对应的参数值时，你可以指定默认参数值。这个听起来很普通的特性实际上特别有用：

```
In [ ]: def myMod(x, y=2):  
        return x % y
```

```
In [ ]: myMod(15, 4)
```

```
In [ ]: myMod(15)
```

8.6 定义任意数量参数

8.6.1 使用*收集位置参数

当参数被用在函数内部时，星号将一组可变数量的位置参数集成参数值的元组。在下面的例子中 `args` 是传入到函数 `print_args()` 的参数值的元组：

```
In [ ]: def mySum(*args):
        result = 0
        for item in args:
            result += item
        return result

mySum(1,2,3,4,5)
```

8.6.2 使用**收集位置参数

针对形参的关键字参数赋值形式，利用 Python 定义函数时，在形参前面加上双星号 ****** 来定义收集关键字参数的形参。此时形参是字典类型。

```
In [ ]: def empDetailInfo(**emp_info):
        if 'name' not in emp_info.keys():
            print('必须有姓名信息。')
        else:
            print(emp_info['name'] + '的信息如下: ')
            print('部门: ' + emp_info.get('department', ''))
            print('电话号码: ' + emp_info.get('phone', ''))
            print('电子邮箱: ' + emp_info.get('email', ''))
        empDetailInfo(name='张珊', department='管理', email='zhangs@163.com')
        print('-' * 30)
        empDetailInfo(name='', department='财务', phone='186123456789')
```

8.7 解包参数

在调用函数时，实参也可以使用 ***** 和 ****** 语法。此时不是收集参数，正好相反，实参前加上 ***** 或 ****** 执行的是参数解包。通常来说，在列表、元组等类型的实参值前加上 *****，将这些类型的元素解包成位置参数的形式；在字典类型的实参值前加上 ******，将字典的元组解包成关键字参数的形式。

```
In [ ]: def empDetailInfo(name, department='', phone='', email=''):
        if name and len(name) > 0:
            print(name + '的信息如下: ')
            print('部门: ' + department)
            print('电话号码: ' + phone)
            print('电子邮箱: ' + email)
        else:
            print('必须有姓名信息。')
```

当调用者的数据存储于列表中时，可以通过在列表前加上 ***** 对列表解包来实现位置参数形式的调用。

```
In [ ]: emp_info1 = ['张明', '管理', '18612345683', 'zhangming@qq.com']
empDetailInfo(*emp_info1)
```

当调用者的数据存储于字典中时，可以通过在字典前加上 ****** 对字典解包来实现关键字参数形式的调用。

```
In [ ]: emp_info2 = {'name': '张珊', 'department': '管理', 'email': 'zhangs@163.com'}
empDetailInfo(**emp_info2)
```

8.8 匿名函数：lambda 函数

Python 中，**lambda** 函数是用一个语句表达的匿名函数。可以用它来代替小的函数。

匿名函数可以赋值给变量来使用，如下：

```
lambda <args>: <expression>
```

lambda 函数与正常函数一样，等价于下面形式：

```
def name(<args>):
    return <expression>
```

简单说，lambda 函数用于定义简单的、能够在一行内表示的函数，返回一个函数类型，实例如下：

```
In [ ]: f = lambda x, y : x + y
        print(type(f))
```



```
print(f(10, 20))
```

```
In [ ]: def myAdd(x, y):  
        return x + y  
        print(myAdd(10, 20))
```

通常，使用实际的函数（例如myAdd()）会比使用 `lambda` 更清晰明了。但是，当需要定义很多小的函数以及记住它们的名字时，`lambda` 会非常有用。`lambda` 函数通常也用于作为其他函数的参数，体会一下下面的这个例子：

```
In [ ]: myList = ['apple', 'watermelon', 'orange', 'lemon']  
myList.sort(key=lambda x:x[1])  
print(myList)
```

8.8 局部和全局作用域

在被调用函数内赋值的变元和变量，处于该函数的“局部作用域”。在所有函数之外赋值的变量，属于“全局作用域”。处于局部作用域的变量，被称为“*局部变量*”。处于全局作用域的变量，被称为“全局变量*”。一个变量必是其中一种，不能既是局部的又是全局的。

- 全局变量指在函数之外定义的变量，一般没有缩进，在程序执行全过程有效。
- 局部变量指在函数内部使用的变量，仅在函数内部有效，当函数退出时变量将不存在。

局部变量不能在全局作用域内使用

```
In [ ]: n = 1      #n是全局变量  
def func(a, b):  
    c = a * b      #c是局部变量，a和b作为函数参数也是局部变量  
    return c  
s = func("knock~", 2)  
print(s)  
#print(c) #这句会发生错误
```

全局变量可以在局部作用域中读取

```
In [ ]: n = 3      #n是全局变量  
def func(a, b):  
    if n < b:  
        c = a * b      #c是局部变量，a和b作为函数参数也是局部变量  
    else:  
        c = a * n  
    return c  
s = func("knock~", 20)  
#s = func("knock~", 2)  
print(s)
```

名称相同的局部变量和全局变量

要想生活简单，就要避免局部变量与全局变量或其他局部变量同名。但在技术上，在Python 中让局部变量和全局变量同名是完全合法的。看看下面的例子：

```
In [ ]: n = 1      #n是全局变量  
def func(a, b):  
    n = b          #修改n的值  
    c = a * b      #c是局部变量，a和b作为函数参数也是局部变量  
    return c  
s = func("knock~", 2)  
print(s, n)
```

`global` 语句

如果需要在函数内修改全局变量，就使用 `global` 语句。如果希望让func()函数将n当作全局变量，需要在变量n使用前显式声明该变量为全局变量，代码如下：

```
In [ ]: n = 1      #n是全局变量  
def func(a, b):  
    global n  
    n = b          #修改n的值  
    c = a * b      #c是局部变量，a和b作为函数参数也是局部变量
```

```
    return c
s = func("knock~", 2)
print(s, n)
```

Python函数对变量的作用遵守如下原则：

- 简单数据类型变量无论是否与全局变量重名，仅在函数内部创建和使用，函数退出后变量被释放；
- 简单数据类型变量在用global保留字声明后，作为全局变量；
- 对于组合数据类型的全局变量，如果在函数内部没有被真实创建的同名变量，则函数内部可直接使用并修改全局变量的值；
- 如果函数内部真实创建了组合数据类型变量，无论是否有同名全局变量，函数仅对局部变量进行操作。

9. 异常处理机制

《观看在线视频》

Python解释器返回了异常信息，告诉我们错误出现在第1行，是ValueError（值错误），具体原因是int()函数不能将'No'这个无效的字符串转换为十进制的整数。

在异常可能发生的地方添加异常处理程序，对于用户明确错误是一种好方法。

Python异常信息中最重要的部分是异常类型，它表明了发生异常的原因，也是程序处理异常的依据。

Python使用 `try-except` 语句实现异常处理，基本的语法格式如下：

```
try:
    <<<语句块1>>>
except <<<异常类型>>>:
    <<<发生异常执行的语句>>>
else:
    <<<没有发生异常执行的语句>>>
finally:
    <<<不管有没有异常都要执行的语句块>>>
```

```
In [ ]: # 分别输入No和5试试看:
try:
    alp = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    idx = eval(input("请输入一个整数: "))
    print(alp[idx])
except NameError:
    print("输入错误, 请输入一个整数!")
else:
    print("没有发生异常")
finally:
    print("程序执行完毕, 不知道是否发生了异常")
```

10. 文件操作

《观看在线视频》

文件包括两种类型：

- 纯文本文件：“文本文件”只包含基本文本字符，不包含字体、大小和颜色信息。带有.txt 扩展名的文本文件，以及带有.py 扩展名的Python 脚本文件，都是纯文本文件的例子。
- 二进制文件：“二进制文件”是其他非纯文本文件的类型，诸如字处理文档、PDF、图像、电子表格和可执行程序。如果用Notepad 或TextEdit 打开一个二进制文件，它看起来就像乱码。二进制文件直接由比特0和比特1组成，没有统一字符编码，文件内部数据的组织格式与文件用途有关。二进制文件和文本文件最主要的区别在于是否有统一的字符编码

无论文件创建为文本文件或者二进制文件，都可以用“文本文件方式”和“二进制文件方式”打开，打开后的操作不同。

```
In [ ]: textFile = open("Documents/example.txt","rt", encoding='utf8') #t表示文本文件方式
print(textFile.readline())
```

```
textFile.close()
binFile = open("Documents/example.txt", "rb")    #b表示二进制文件方式
print(binFile.readline())
binFile.close()
```

在Python 中，读写文件有3 个步骤：

- (1) 调用 `open()` 函数，返回一个File 对象。
- (2) 调用File 对象的 `read()` 或 `write()` 方法。
- (3) 调用File 对象的 `close()` 方法，关闭该文件。

10.1 用 `open()` 函数打开文件

要用 `open()` 函数打开一个文件，就要向它传递一个字符串路径，表明希望打开的文件。这既可以是绝对路径，也可以是相对路径。`open()` 函数返回一个File对象。

```
open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None,
closefd=True, opener=None)
```

```
In [ ]: textFile = open("Data/2020_01.csv")
```

```
In [ ]: textFile = open("Documents/example.txt", "rt") #t表示文本文件方式
```

这些命令都将以读取纯文本文件的模式打开文件，或简称为“读模式”。当文件以读模式打开时，Python 只让你从文件中读取数据，你*不能*以任何方式写入或修改它。

在Python 中打开文件时，读模式是默认的模式。但如果你不希望依赖于Python 的默认值，也可以明确指明该模式，向 `open()` 传入字符串 `'r'`，作为第二个参数。

调用 `open()` 将返回一个File 对象。File 对象代表计算机中的一个文件，它只是Python 中另一种类型的值，就像你已熟悉的列表和字典。

`open()` 函数提供7种基本的打开模式：

打开模式	含义
'r'	只读模式，如果文件不存在，返回异常FileNotFoundError，默认值
'w'	覆盖写模式，文件不存在则创建，存在则完全覆盖源文件
'x'	创建写模式，文件不存在则创建，存在则返回异常FileExistsError
'a'	追加写模式，文件不存在则创建，存在则在原文件最后追加内容
'b'	二进制文件模式
't'	文本文件模式，默认值
'+'	与r/w/x/a一同使用，在原功能基础上增加同时读写功能

10.2 读取文件内容

既然有了File 对象，就可以开始从它读取内容。如果希望将整个文件的内容读取为一个字符串值，就使用File对象的 `read()` 方法。

```
In [ ]: empFile = open("Documents/employees.txt", encoding='utf8')
empInfo = empFile.read()
empFile.close()
print(empInfo)
```

或者，可以使用 `readlines()` 方法，从该文件取得一个字符串的列表。列表中的每个字符串就是文本中的每一行。请注意每行后面的换行符： `\n`

```
In [ ]: empFile = open("Documents/employees.txt", encoding='utf8')
empInfo = empFile.readlines()
empFile.close()
```

```
print(type(empInfo))
print(empInfo)
```

10.2.1 使用 for 循环逐行读取

在绝大多数情况下，对于文本文件的读取都是逐行读取。文件对象本身是一个可迭代对象。因此，通常情况下，推荐使用 `for` 循环来逐行读取文本文件内容。

```
In [ ]: in_file = open('sales.txt', encoding='utf-8')
        for line in in_file:
            print(line, end='')
        in_file.close()
```

10.2.2 使用列表推导式和 map() 函数逐行处理

文件作为可迭代对象，可以使用列表推导式和 `map()` 函数对每行进行相应处理，生成新的列表。

```
In [ ]: in_file = open('sales.txt', encoding='utf-8')
        lines = [line.rstrip() for line in in_file]
        lines
```

```
In [ ]: in_file.close()
```

```
In [ ]: lines = list(map(lambda line: line.rstrip(), open('sales.txt')))
        lines
```

10.2.3 使用 with 语句自动管理

Python 中的 `with` 语句适用于对资源进行访问的场合，确保不管使用过程中是否发生异常都会释放资源，比如文件使用后自动关闭。

```
In [ ]: with open('sales.txt', encoding='utf-8') as sales_file:
        for line in sales_file:
            print(line, end='')
```

```
In [ ]: result = 0
        for line in textFile:
            detail = line.split(',')
            result = result + int(detail[-2])
        print(result)
```

10.3 写入文件

Python 允许你将内容写入文件，方式与 `print()` 函数将字符串“写”到屏幕上类似。但是，如果打开文件时用读模式，就不能写入文件。你需要以“写入纯文本模式”或“添加纯文本模式”打开该文件，或简称为“写模式”和“添加模式”。写模式将覆写原有的文件，从头开始，就像你用一个新值覆写一个变量的值。

将 `'w'` 作为第二个参数传递给 `open()`，以写模式打开该文件。不同的是，添加模式将在已有文件的末尾添加文本。可以认为这类似向一个变量中的列表添加内容，而不是完全覆写该变量。

将 `'a'` 作为第二个参数传递给 `open()`，以添加模式打开该文件。

如果传递给 `open()` 的文件名不存在，写模式和添加模式都会创建一个新的空文件。在读取或写入文件后，调用 `close()` 方法，然后才能再次打开该文件。

Python 提供 3 个与文件内容写入有关的方法：

方法	含义
<code><file>.write(s)</code>	向文件写入一个字符串或字节流
<code><file>.writelines(lines)</code>	将一个元素为字符串的列表写入文件
<code><file>.seek(offset)</code>	改变当前文件操作指针的位置，offset 的值：0：文件开头；1：当前位置；2：文件结尾

```
In [ ]: with open("Documents/result1.txt", "w", encoding='utf8') as resultFile:
        resultFile.write("这是第一行\n")
        resultFile.write("这是第二行第一部分")
        resultFile.write(" ")
        resultFile.write("这时第二行第二部分")
```

11. 第三方库

《观看在线视频》

Python是一门简洁、优雅的语言，丰富的第三方库能让我们很多的编程任务变得更加简单。

11.1 导入模块

Python 程序可以调用一组基本的函数，这称为“内建函数”，包括你见到过的 `print()`、`input()` 和 `len()` 函数。Python 也包括一组模块，称为“标准库”。每个模块都是一个Python 程序，包含一组相关的函数，可以嵌入你的程序之中。例如，`math`模块有数学运算相关的函数，`random` 模块有随机数相关的函数，等等。

在开始使用一个模块中的函数之前，必须用 `import` 语句 导入该模块。在代码中，`import` 语句包含以下部分：

- `import` 关键字；
- 模块的名称；
- 可选的更多模块名称，之间用逗号隔开。

在导入一个模块后，就可以使用该模块中所有很酷的函数。让我们试一试 `random` 模块，它让我们能使用 `random.randint()` 函数。。因为 `randint()` 属于 `random` 模块，必须在函数名称之前先加上 `random.`，告诉python 在 `random` 模块中寻找这个函数。

```
In [ ]: import random
        for i in range(5):
            print(random.randint(1, 10))
```

11.1.1 利用 `import` 模块名

导入整个模块：

```
In [ ]: import math
        math.sqrt(4)
```

11.1.2 利用 `import` 模块名 `as` 名称缩写

导入整个模块的同时给该模块取个较短的别名：

```
In [ ]: import numpy as np
        np.sqrt(4)
```

11.1.3 利用 `import` 模块名.子模块名 `as` 名称缩写

导入某个模块子模块的同时给该子模块取个较短的别名：

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

11.1.4 利用 `from` 模块名 `import` 函数

导入模块中指定函数：

```
In [ ]: from math import sqrt, exp
        sqrt(4)
```

11.1.5 利用 from 模块名.子模块名 import 函数

导入某个模块的子模块中指定函数：

```
In [ ]: from matplotlib.pyplot import plot
```

Windows 下安装非标准库，需要在命令提示符中输入：

pip install 库名

Linux 和 Mac OS X 中则是在终端输入上述命令。

11.2 random库的使用

随机数在计算机应用中十分常见，Python内置的random库主要用于产生各种分布的伪随机数序列。`random` 库采用梅森旋转算法（Mersenne twister）生成伪随机数序列，可用于除随机性要求更高的加解密算法外的大多数工程应用。

使用random库主要目的是生成随机数，因此，只需要查阅该库的随机数生成函数，找到符合使用场景的函数使用即可。这个库提供了不同类型的随机数函数，所有函数都是基于最基本的 `random.random()` 函数扩展而来。

函数	描述
seed(a=None)	初始化随机数种子，默认值为当前系统时间
random()	生成一个[0.0, 1.0)之间的随机小数
randint(a, b)	生成一个[a,b]之间的整数
choice(seq)	从序列类型中随机返回一个元素
shuffle(seq)	将序列类型中元素随机排列，返回打乱后的序列
sample(pop,k)	从pop中随机选取k个元素，以列表类型返回

```
In [ ]: import random
In [ ]: random.random()
In [ ]: random.randint(1, 10)
In [ ]: for i in range(10):
        print(random.choice("abcdef1234567!@#$%"), end="")
```

11.3 math库的使用

math库是Python提供的内置数学函数库，支持整数和浮点数运算。

函数	数学表示	描述
math.ceil(x)	$\lceil x \rceil$	向上取整，返回不小于 x 的最小整数
math.floor(x)	$\lfloor x \rfloor$	向下取整，返回不大于 x 的最大整数
math.sqrt(x)	\sqrt{x}	返回 x 的平方根

```
In [ ]: import math
        math.ceil(10.1)
In [ ]: from math import floor
        floor(10.8)
In [ ]: math.sqrt(16)
In [ ]:
```

12. 程序理解题说明

在大模型时代，程序设计课程开始引入程序理解题，以帮助学生正确解读由大模型编写的代码。这类题目要求学生分析代码的逻辑结构、理解代码的功能，从而能够对代码做出正确的解释和回答相关问题。这样不仅能帮助学生提高阅读代码的能力，还能锻炼逻辑思维和问题解决能力。

程序理解题的结构

- 代码展示：提供一段由大模型生成的代码。
- 问题提出：设置与代码相关的问题，要求学生根据代码进行回答。
- 解答逻辑：引导学生分析代码的结构，推理出正确答案。

示例题

代码展示

```
def is_prime(n):  
    """Checks if a number is a prime number."""  
    if n <= 1:  
        return False  
    for i in range(2, int(n**0.5) + 1):  
        if n % i == 0:  
            return False  
    return True  
  
def prime_count(limit):  
    """Counts the number of prime numbers below a given limit."""  
    count = 0  
    for num in range(2, limit):  
        if is_prime(num):  
            count += 1  
    return count  
  
result = prime_count(10)  
print(result)
```

问题提出

1. 函数 `is_prime` 的作用是什么？
2. 函数 `prime_count` 在参数 `limit` 为10时，返回的值是什么？

解答逻辑

1. `is_prime` 函数作用：
 - `is_prime(n)` 判断一个数字 `n` 是否是质数。质数是指除了1和自身外，无法被其他自然数整除的数字。
 - 如果 `n <= 1`，直接返回 `False`。
 - 循环检查能否找到一个 2 到 \sqrt{n} 范围内的 `i`，能整除 `n`，如果有则 `n` 不是质数。
2. `prime_count` 函数输出分析：
 - `prime_count(limit)` 统计 `limit` 以下的质数个数。
 - 当 `limit` 为10时，质数有 2, 3, 5, 7，共4个。
 - 故 `prime_count(10)` 返回值为4。