

# # " " " 三单双反引号多行注释

## 1. Python的格式框架

蜂考 bilibili

### 3) 续行符

Python提供“续行符”将单行代码分割为多行表达。续行符由反斜杠（\）符号表达。

```
>>>print(\n    "Hello"\n    "World!"\n)\nHello World!
```

注意：续行符后不能存在空格，续行符后必须直接换行。续行符不仅可以用于单行代码较长的情况，也适合对代码进行多行排版以增加可读性的情况。

## 保留字大小写敏感

## 2. 保留字

蜂考 bilibili

保留字 (keyword)，也称关键字，指被编程语言内部定义并保留使用的标识符。程序员编写程序不能命名与保留字相同的标识符。

每种程序设计语言都有一套保留字，保留字一般用来构成程序整体框架、表达关键值和具有结构性的复杂语义等。

Python 3.x 版本共有35个保留字，如表所示，按照字母顺序排列。与其他标识符一样，Python 的保留字也是大小写敏感的。例如，True是保留字，但true则不是保留字。

### Python的 35 个保留字列表

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

### (2) 变量的命名

①变量名可以包括字母、数字和下划线，但是数字不能作为开头字符；

例：test1是有效变量名，而1test则是无效变量名

②系统关键字不能做变量名使用；

例：and、break等都是系统关键字，不能作为变量名使用

③Python的变量名区分大小写。

例：test和Test是两个不同的变量

提示：Python 3.x默认使用UTF-8编码，变量名中允许包含中文，如“测试”是一个有效的变量名

### ②同时赋给多个变量

语法格式：变量1, 变量2, ..., 变量 n = 值1, 值2, ..., 值n

例：name, age = 'Lucy', 18

题1. 下列标识符中，合法的是（ ）。

A. helloWorld

B. 2ndObj

C. hello#world

D. -helloworld

## 4. 基本数据类型

蜂考 bili

例：整数1010，其各种进制的数据分别如下：

十进制：1010 #不加任何前缀的为十进制整数

十六进制：0x3F2 #加前缀0x为十六进制整数

八进制：0o1762 #前缀0o为八进制整数

二进制：0b00111110010 #前缀0b为二进制整数

0 x 八  
b =

(2) 浮点型（float），没有取值范围，可正可负，一个浮点数可以表示为带小数点的一般形式，也可以采用科学计数法表示。浮点数只有十进制形式。

例：浮点数123.456，两种表达方式如下：

一般形式：123.456

科学计数法：1.23456e2

对比如 C/C++

十进制(Decimal)：123

八(Octal)：以0开头

十六(Hexadecimal)：以0x或0X开头

二(Binary) C无

C++ | 4 → 0b / 0B

0b / 0B

<complex> 文件

4.11

complex <double>

实虚  
C(C A, B)

### (3) 复数类型 (complex)

复数由实部和虚部组成，每一部分都是一个浮点数，其书写方法：a+bj或a+bJ

其中，a和b是两个数字，j或J是虚部的后缀，即a是实部、b是虚部。

在生成复数时，也可以使用complex函数，其语法格式：complex(real, imag)

其中，real为实部值，imag为虚部值，返回值为real+imag\*j。

## 4. 基本数据类型

### 2) 字符串类型 (string)

Python语言中只有用于保存字符串的String类型，而没有用于保存单个字符的数据类型。

#### (1) 字符串的定义

所有使用成对的引号括起来的任何字符，都叫字符串。

#### (2) 字符串的表示方法

由一对单引号 '' 或双引号 "" 表示，仅表示单行字符串

由一对三单引号或三双引号表示，可表示多行字符串

例：

>>>a = "Hello World!" #双引号

>>>b = 'Hello World!' #单引号

### (3) 字符串的拼接

Python中用于拼接字符串的运算符是加号 (+)

字符串的重复使用运算符是乘号 (\*)

例：

>>>a = "Hello" #双引号

>>>b = 'World!' #单引号

>>>c = a + b #字符串拼接

>>>c

"Hello World!"

>>>d = a \* 2 #字符串重复使用

>>>d

"HelloHello"

C中：Strcat (str1, str2)

C++中：<string>

str1 + str2

str1.append(str2);

自动管理内存

值得一提的是，布尔类型可以当做整数来对待，即True相当于整数值1，False相当于整数值0

因此，下边这些运算都是可以的：

例：

```
>>> False+1  
1  
>>> True+1  
2
```

#### 4) 数据类型转换

Python已经为我们提供了多种可实现数据类型转换的函数，如表所示。

函数	作用
<code>int(x)</code>	将 <code>x</code> 转换成整数类型
<code>float(x)</code>	将 <code>x</code> 转换成浮点数类型
<code>complex(real, [, imag])</code>	创建一个复数
<code>str(x)</code>	将 <code>x</code> 转换为字符串
<code>repr(x)</code>	将 <code>x</code> 转换为表达式字符串
<code>eval(str)</code>	计算在字符串中的有效Python表达式，并返回一个对象
<code>chr(x)</code>	将整数 <code>x</code> 转换为一个字符
<code>ord(x)</code>	将一个字符 <code>x</code> 转换为它对应的整数值
<code>hex(x)</code>	将一个整数 <code>x</code> 转换为一个十六进制字符串
<code>oct(x)</code>	将一个整数 <code>x</code> 转换为一个八进制的字符串



蜂考



## 5. 运算符

### 1) 算术运算符

算术运算符即数学运算符，用来对数字进行数学运算，比如加减乘除。下表列出了Python支持所有基本算术运算符。

Python常用算术运算符			
运算符	说明	实例	结果
<code>+</code>	加	<code>12.45 + 15</code>	27.45
<code>-</code>	减	<code>4.56 - 0.26</code>	4.3
<code>*</code>	乘	<code>5 * 3.6</code>	18.0
<code>/</code>	除法（和数学中的规则一样）	<code>7 / 2</code>	3.5
<code>//</code>	整除（只保留商的整数部分）	<code>7 // 2</code>	3
<code>%</code>	取余，即返回除法的余数	<code>7 % 2</code>	1
<code>**</code>	幂运算/次方运算，即返回x的y次方	<code>2 ** 4</code>	16，即 24

### Python扩展赋值运算符

运算符	说明	用法举例	等价形式
<code>=</code>	最基本的赋值运算	<code>x = y</code>	<code>x = y</code>
<code>+=</code>	加赋值	<code>x += y</code>	<code>x = x + y</code>
<code>-=</code>	减赋值	<code>x -= y</code>	<code>x = x - y</code>
<code>*=</code>	乘赋值	<code>x *= y</code>	<code>x = x * y</code>
<code>/=</code>	除赋值	<code>x /= y</code>	<code>x = x / y</code>
<code>%=</code>	取余数赋值	<code>x %= y</code>	<code>x = x % y</code>
<code>**=</code>	幂赋值	<code>x **= y</code>	<code>x = x ** y</code>
<code>//=</code>	取整数赋值	<code>x //= y</code>	<code>x = x // y</code>
<code>&amp;=</code>	按位与赋值	<code>x &amp;= y</code>	<code>x = x &amp; y</code>
<code> =</code>	按位或赋值	<code>x  = y</code>	<code>x = x   y</code>
<code>^=</code>	按位异或赋值	<code>x ^= y</code>	<code>x = x ^ y</code>
<code>&lt;=</code>	左移赋值	<code>x &lt;= y</code>	<code>x = x &lt;&lt; y</code> ，这里的y指的是左移位数
<code>&gt;=</code>	右移赋值	<code>x &gt;= y</code>	<code>x = x &gt;&gt; y</code> ，这里的y指的是右移位数

C/C++不支持

左移 位数  
 $x \ll y$   $\ll$  10  
右移 位数  
 $x \gg y$   $\gg$  10

位向左移动一位  
 $x \ll 1$   $\ll$  1

## Python比较运算符汇总

位数升高扩大

比较运算符	说明
>	大于, 如果>前面的值大于后面的值, 则返回 True, 否则返回 False。
<	小于, 如果<前面的值小于后面的值, 则返回 True, 否则返回 False。
=	等于, 如果==两边的值相等, 则返回 True, 否则返回 False。
>=	大于等于(等价于数学中的 $\geq$ ), 如果>=前面的值大于或者等于后面的值, 则返回 True, 否则返回 False。
<=	小于等于(等价于数学中的 $\leq$ ), 如果<=前面的值小于或者等于后面的值, 则返回 True, 否则返回 False。
!=	不等于(等价于数学中的 $\neq$ ), 如果!=两边的值不相等, 则返回 True, 否则返回 False。 说白了 != 不相等
is	判断两个变量所引用的对象是否相同, 如果相同则返回 True, 否则返回 False。
is not	判断两个变量所引用的对象是否不相同, 如果不相同则返回 True, 否则返回 False。 蜂 FEN

## Python逻辑运算符及功能

逻辑运算符	含义	基本格式	说明
and	逻辑与运算	a and b	当a和b两个表达式都为真时, a and b 的结果才为真, 否则为假。
or	逻辑或运算	a or b	当a和b两个表达式都为假时, a or b 的结果才是假, 否则为真。
not	逻辑非运算	not a	如果a为真, 那么not a的结果为假; 如果a为假, 那么not a的结果为真。相当于对a取反。

## 5) 位运算符

位运算是指对二进制数进行逐位运算。

位运算符	说明	使用形式	详解
&	按位与	a & b	如果a和b对应位均为1, 则结果中该位为1, 否则为0。
	按位或	a   b	如果a和b对应位均为0, 则结果中该位为0, 否则为1。
^	按位异或	a ^ b	如果a和b对应位不同, 则结果中该位为1, 否则为0。
~	按位取反	~a	如果a某位为1, 则结果中该位为0, 否则为1。
<<	按位左移	a << b	将a左移b位(右侧补0)
>>	按位右移	a >> b	将a右移b位(左侧补0)

## 6) 身份运算符

身份运算符用于比较两个对象的存储单元。

运算符	描述	实例
is	is 是判断两个标识符是不是引用自一个对象	x is y, 类似 id(x) == id(y), 如果引用的是同一个对象则返回 True, 否则返回 False
is not	is not 是判断两个标识符是不是引用自不同对象	x is not y, 类似 id(a) != id(b)。如果引用的不是同一个对象则返回结果 True, 否则返回 False。

注: `id()` 函数用于获取对象内存地址。

## 7) 成员运算符

用来识别某一元素是否包含在变量中，这个变量可以是字符串、列表、元组

运算符	描述	实例	
in	如果在指定的序列中找到值返回 True, 否则返回 False。	x在y序列中, 如果x在y序列中返回True。	
not in	如果在指定的序列中没有找到值返回True, 否则返回 False。	x不在y序列中, 如果x不在y序列中返回True。	
运算符说明	Python运算符	结合性	优先级顺序 高 ↑ ↓ 低
小括号	( )	无	
乘方	**	右	
按位取反	~	右	
符号运算符	+ (正号)、- (负号)	右	
乘除	*、/、//、%	左	
加减	+、-	左	
位移	>>、<<	左	
按位与	&	右	
按位异或	^	左	
按位或		左	
比较运算符	==、!=、>、>=、<、<=	左	
is 运算符	is、is not	左	
in 运算符	in、not in	左	
逻辑非	not	右	
逻辑与	and	左	
逻辑或	or	左	
逗号运算符	exp1, exp2	左	

蜂考

## 6. 输入输出语句

### 1) 输入函数

#### (1) input() 函数

功能：接收标准输入数据（即从键盘输入），返回字符串类型。

语法格式：

```
str = input(tipmsg)
```

说明：str表示一个字符串类型的变量，input会将读取到的字符串放入str中。

tipmsg表示提示信息，它会显示在控制台上，告诉用户应该输入什么样的内容；如果不写tipmsg，就不会有任何提示信息。

```
例：name=input("请输入你的姓名：") #输入“张三”
```

```
print(name)
```



## (2) eval() 函数

功能：计算字符串所对应的表达式的值，返回表达式的计算结果

语法格式：eval(expression)

说明：expression是字符串类型的参数，对应一个有效的Python表达式。

例：

```
r=eval(input("请输入一个有效的表达式："))
print(r)
```

运行结果：输入3+5，则输出8；输入3，则输出3

## (3) print() 函数

功能：将各种类型的数据（字符串、整型、浮点型、列表等）输出到屏幕上。

语法格式：print(object)

其中，object是要输出的数据

例：

```
print("Hello World!") #输出 "Hello World!"
print(10) #输出 "10"
print(3.5) #输出 "3.5"
print([1, 3, 5, 'list']) #输出 "[1, 3, 5, 'list']"
print({1:'A', 2:'B', 3:'C', 4:'D'}) #输出 {1: 'A', 2: 'B', 3: 'C', 4: 'D'}
```

蜂考 bilibili

## 1. Python的程序结构

Python的基本控制语法结构分为：顺序结构、分支结构和循环结构。

### 1) 顺序结构

最简单的控制结构，即按照语句的顺序从上往下依次执行（默认）

### 2) 分支结构

也称选择结构，表示程序运行中会根据特定条件的具体值和判断结果来选择其中一个分支执行。

实现分支结构的语句只有一种：if

### 3) 循环结构

表示在程序运行中，会根据特定条件的具体值和判定结果选择重复执行某一段代码。Python中

实现循环结构的方式有两种：for、while。

## 3. 分支结构

蜂考 bilibili

Python中实现分支结构的语句为if，主要有三种形式：if语句、if...else语句和  
if...elif...else语句。

### 1) if语句

最简单的分支结构是由条件语句if和语句块构成。

简单if语句的语法格式为：

```
if <条件表达式>:
    <语句块>
```

其中，条件表达式可以是布尔值或变量，也可以是比较表达式或逻辑表达式，如果表达式为True时就执行语句块，否则就跳过“语句块”，继续执行后续的语句。

## 2) 二分支结构: if-else

Python的二分支结构使用 `if-else` 保留字对条件进行判断, 语法格式如下:

```
if <条件表达式>:  
    <语句块 1>  
  
else:  
    <语句块 2>
```

其中, `if`、`else`、`:` 和 `<语句块>` 前的缩进 都是语法的一部分。

`<语句块1>` 在 `if` 中 `<条件表达式>` 满足即为 `True` 时执行, `<语句块 2>` 在 `if` 中 `<条件>` 不满足即 `False` 时执行。简单说, 二分支结构根据条件的 `True` 或 `False` 结果产生两条路径。

二分支结构还有一种更简洁的表达方式, 适合 `<语句块1>` 和 `<语句块 2>` 都只包含简单表达式的情况, 语法格式如下:

```
<表达式 1> if <条件> else <表达式 2>
```

对于简洁表达方式, 要使用表达式而不是语句。

例:

```
#求绝对值  
a= -9  
b= a if a>0 else -a  
print(b)
```

说明: 表达式和语句

表达式是产生或计算新数据值的代码片段, 它并不是完整语句。例如, `45+2` 是表达式, `a=45+2` 则是语句。

## 3) 多分支结构: if...elif...else

Python 的多分支结构使用 `if...elif...else` 保留字对多个相关条件进行判断, 并根据不同条件结果按照顺序选择执行路径, 语法格式如下:

```
if <条件 1>:  
    <语句块 1>  
elif <条件 2>:  
    <语句块 2>  
...  
else:  
    <语句块 N>
```

说明:

- (1) 多分支结构的关键词语句 `if`、`elif`、`else` 的缩进必须保持一致;
- (2) 多分支结构的不同分支的代码块之间的缩进如果不一致, 从语法的角度是允许的, 但是从美观的角度是不好的。

`while` 的功能为: 当条件成立时, 执行循环体, 然后再次检测条件, 如果还成立, 再次执行循环, 直到条件不再成立时跳出循环, 转去执行后面的其他语句。

例:

```
>>>n=0  
>>>while n<10:  
    print(n)  
    n=n+3  
  
0  
3  
6  
9
```

无限循环也有一种使用保留字 else 的扩展模式，使用方式如下：

```
while <条件>:  
    <语句块1>  
else:  
    <语句块 2>
```

在这种扩展模式中，当 while 循环正常执行之后，程序会继续执行 else 语句中内容。else 子句只在循环正常执行后才执行，因此，可以在语句块2中放置评价循环执行情况的语句。

?

else 属于循环内，break 跳出循环，就也跳过了 else

## 4. 循环结构

蜂考

题1. 请阅读下面的程序：

```
count=0  
  
while count<5:  
    print (count, '小于5')  
    if count==2:  
        break  
    count += 1  
  
else:  
    print (count, "不小于5")
```

关于上述程序的说法中，描述错误的是（ ）。

- A. else 语句会在循环执行完成后运行
- B. 当 count 的值等于 2 时，程序会终止循环
- C. break 语句会跳过 else 语句块执行
- D. else 语句块一定会执行

While 未正常完成退出不执行 else



遍历循环可以理解为从遍历结构中逐一提取元素，放在循环变量中，对于每个所提取的元素执行一次语句块。for 语句的循环执行次数是根据遍历结构中元素个数确定的。

遍历结构可以是字符串、文件、range() 函数或组合数据类型等。

对于字符串，可以逐一遍历字符串的每个字符，基本使用方式如下：

```
for <循环变量> in <字符串变量>:  
    <语句块>
```

```
>>>for c in "Python":
```

```
    print(c)
```

```
P  
y  
t  
h  
o  
n
```



n  
0~n-1

```
>>>for i in range(5):  
    print(i)
```

索引从 0 开始

```
0  
1  
2  
3  
4
```

## 拓展: range() 函数

该函数是Python内置的函数, 用于生成一系列连续的整数, 多用于 for 循环语句中。其语法

格式如下:

`range(start, end, step)`

(start, end) 左闭右开

参数说明:

`start`: 用于指定计数的起始值, 可以省略, 如果省略则从0开始。

`end`: 用于指定计数的结束值(但不包括该值, 如`range(7)`, 则得到的值为0~6, 不包括7), 不能省略。当`range()`函数中只有一个参数时, 即表示指定计数的结束值。

`step`: 用于指定步长, 即两个数之间的间隔, 可以省略, 如果省略则表示步长为1。例如,  
`range(1, 7)` 将得到 1、2、3、4、5、6。

遍历循环还有一种扩展模式, 使用方法如下:

```
for <循环变量> in <遍历结构>:
```

<语句块 1>

```
else:
```

<语句块 2>

当`for`循环正常执行之后, 程序会继续执行`else`语句中内容。`else`语句只在循环正常执行后才执行并结束, 因此, 可以在<语句块 2>中放置评价循环执行情况的语句。

## 5. 循环嵌套

在`while`循环中套用`for`循环的格式如下:

`while` 条件表达式:

`for` 迭代变量 `in` 对象:

循环体2

循环体1

在`for`循环中套用`while`循环的格式如下:

`for` 迭代变量 `in` 对象:

`while` 条件表达式:

循环体2

循环体1

end非关键字, 常用于`print()`中复制输出结尾字符  
可以替换为任意字符串, `end=' '`不换行, `end='--'`表示以--结尾

### 1. 序列

#### 川贝序存储

蜂考

序列是一块用于存放多个值的连续的内存空间, 并且按一定顺序排列, 每个值(称为元素)都分配一个数字, 称为索引或位置。通过该索引可以去取相应的值。

在 Python 中, 序列结构主要有列表、元组、集合、字典和字符串, 对于这些序列结构有以下通用的操作, 其中, 集合和字典不支持索引、切片、相加和相乘操作。

### 1. 序列

#### 1) 索引

索引也可以是负数, 这个索引从右往左计数, 也就是从最后一个元素开始计数, 最后一个元素的索引值为-1

, 倒数第二个元素索引值为-2 , 以此类推。

元素 1	元素 2	元素 3	元素 ...	元素 $n-1$	元素 $n$
------	------	------	--------	----------	--------

$-n$   $-(n-1)$   $-(n-2)$  ... -2 -1 ← 索引(下标)

图2 序列的负数索引

# 1. 序列

## 2) 操作符和函数

序列类型有一些通用的操作符和函数。

操作符	描述
$x \text{ in } s$	如果 $x$ 是 $s$ 的元素, 返回 $\text{True}$ ; 否则返回 $\text{False}$
$x \text{ not in } s$	如果 $x$ 不是 $s$ 的元素, 返回 $\text{True}$ ; 否则返回 $\text{False}$
$s + t$	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
$s[i:j]$	切片, 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)
$s[i:j:k]$	步骤切片, 返回包含序列 $s$ 第 $i$ 到 $j$ 个元素以 $k$ 为步数的子序列
$\text{len}(s)$	序列 $s$ 的元素个数(长度)
$\text{min}(s)$	序列 $s$ 中的最小元素
$\text{max}(s)$	序列 $s$ 中的最大元素
$s.\text{index}(x)$	序列 $s$ 中第一次出现元素 $x$ 的位置
$s.\text{count}(x)$	序列 $s$ 中出现 $x$ 的总次数

## 2. 列表

### 1) 列表的定义

列表是包含 0 个或多个元素的有序序列, 属于序列类型。

列表可以进行元素增加、删除、替换、查找等操作。列表没有长度限制,

元素类型可以不同, 不需要预定义长度。

列表类型用中括号([ ])表示, 也可以通过 `list(x)` 函数将集合或字符串类型转换成列表类型。`list()` 函数可生成空列表。

```
>>>ls = [ 1010, "1010", [ 1010, "1010"], 1010]
>>>ls
[1010, '1010', [1010,'1010'], 1010]
>>>list('列表可以由字符串生成')
['列', '表', '可', '以', '由', '字', '符', '串', '生', '成']
>>>list()
[]
```

## 2. 列表

### 4) 列表的操作

#### (2) 操作方法

使用切片配合等号(=)可以对列表片段进行修改,修改内容可以不等长。当使用一个列表改变另一个列表值时,Python不要求两个列表长度一样,但遵循“多增少减”的原则。

```
>>>lt=[“1010”, “10.10”, “Python”]  
>>>lt[1:2]=[1010,10.10,0x1010]  
>>>print(lt)  
[‘1010’,1010,10.1,4112, ‘Python’]  
>>>lt[1:4]=[1010]  
[‘1010’,1010, ‘Python’]
```

## 5. 集合

### 1) 创建集合

这个代码片段表明,尽管集合中的元素是不可重复的,但是集合元素在输入时是不受限制的。元素在输入集合后会自动去重。

集合类型有4个操作符,如表所示。

操作符及运算	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	返回一个新集合,包括同时在集合S和T中的元素
S^T	返回一个新集合,包括集合S和T中非共同元素
S T	返回一个新集合,包括在集合S和T中所有元素

## 1. 字符串的常用操作

蜂考

### 1) 字符串的定义

字符串是字符的序列表示,根据字符串的内容多少分为单行字符串和多行字符串。

单行字符串可以由一对单引号('')或双引号("")作为边界来表示,单引号和双引号作用相同。

当使用单引号时,双引号可以作为字符串的一部分;

使用双引号时,单引号可以作为字符串的一部分。

多行字符串可以由一对三单引号(''')或三双引号(" " ")作为边界来表示,两者作用相同。

多行字符串用于大段文本的情况,一般采用变量表示:

```
>>> print('这是“单行字符串”')  
这是“单行字符串”  
>>>print(“这是’ 单行字符串’ ” )  
这是’ 单行字符串’
```

```
s='’这是“多行字符串”的第一行  
这是“多行字符串”的第二行  
’’’  
print(s)
```

# 1. 字符串的常用操作

## 1) 字符串的定义

反斜杠字符(\)是一个特殊字符，在Python字符串中表示“转义”，即该字符与后面相邻的一个字符共同组成了新的含义。

例如:\n表示换行、\\表示反斜杠、\’ 表示单引号、\” 表示双号、\t表示制表符(Tab)等。

如果在字符串中既需要出现单引号又需要出现双引号，则需使用转义符。

反斜杠字符(\)还有一个额外作用：续行。

```
>>>print("这里\n有一个换行")
这里
有一个换行
>>>print("这里\\有一个反斜杠")
这里\有一个反斜杠
>>>print("既需要'单引号'又需要\"双引号\"")
既需要'单引号'又需要"双引号"
```

例：  
if(a>10 and a<100) or \  
(a<-10 and a>-100):  
 print("True")

## 4) 字符串类型的操作

### (2) 字符串处理函数

Python语言提供了一些对字符串处理的内置函数，如表所示。

函数	描述
len(x)	返回字符串x的长度。也可返回其他组合数据类型的元素个数
str(x)	返回任意类型x所对应的字符串形式
chr(x)	返回Unicode编码x对应的单字符
ord(x)	返回单字符x表示的Unicode编码
hex(x)	返回整数x对应十六进制数的小写形式字符串
oct(x)	返回整数x对应八进制数的小写形式字符串

## 4) 字符串类型的操作

### (3) 字符串处理方法

方法也是一个函数，只是调用方式不同。采用.运算的形式

表中给出了常用的字符串处理方法，其中str代表一个字符串或字符串变量。

方法	描述
str.lower()	返回字符串str的副本，全部字符小写
str.upper()	返回字符串str的副本，全部字符大写
str.split(sep = None)	返回一个列表，由str根据sep被分割的部分构成，省略sep默认以空格分割
str.count(sub)	返回sub子串出现的次数
str.replace(old, new)	返回字符串str的副本，所有old子串被替换为new
str.center(width, fillchar)	字符串居中函数，fillchar参数可选
str.strip(chars)	从字符串str中去掉在其左侧和右侧chars中列出的字符
str.join(iter)	将iter变量的每一个元素后增加一个str字符串

表中返回字符串的副本指返回一个新的字符串，但不改变原来的变量str。

# 1. 面向对象概述

## 3) 面向对象程序设计的特点

面向对象程序设计具有三大基本特征：封装、继承和多态。

### (1) 封装

封装是面向对象编程的核心思想，将对象的属性和行为封装起来，其载体就是类，类通常会对客户隐藏其实现细节，这就是封装的思想。

### (2) 继承

菱形、平行四边形和梯形等都是四边形。因为四边形与它们具有共同的特征，拥有4条边。平行四边形复用了四边形的属性和行为，同时添加平行四边形特有的属性和行为，如平行四边形的对边平行且相等。

在Python中，可以把平行四边形看作靠继承四边形类产生的类，其中平行四边形的类称为子类，将类似于四边形的类称为父类和超类。

### (3) 多态

将父类对象应用于子类的特征就是多态。

题1. Python类的构造函数是 `__init__`。 (✓)

题2. 定义类时，在一个方法前面使用`@classmethod`进行修饰，则该方法属于类方法。 (✓)

#### 4) 创建类的成员并访问

##### (1) 创建实例方法并访问

所谓实例方法是指在类中定义的函数。该函数是一种在类的实例上操作的函数。同`__init__`方法一样，实例方法的第一个参数必须是`self`，并且必须包含一个`self`参数。

创建实方法的语法格式如下：

```
def funName(self, [list]):  
    block
```

参数说明：

`funName`: 用于指定方法名，一般使用小写字母开头。

`self`: 必要参数，表示类的实例，其名称可以是`self`以外的单词，使用`self`只是一个惯例而已。

`list`: 用于指定除`self`参数以外的参数，各参数间使用逗号“,”进行分隔。

`block`: 方法体，实现的具体功能。

说明：实例方法和Python中的函数的主要区别就是，函数实现的是某个独立的功能，而实例方法是实现类中的一个行为，是类的一部分。

说明：

(1) “`if __name__=='__main__':`”语句的作用就是当运行`myModule`模块时，Python解释器把一个特殊变量`__name__`置为`main`，这时`if`判断成立，就执行`print("sum=", sum(1, 100))`；而当`myModule`模块被导入到其它文件时，`__name__`等于导入模块的模块名即`__name__`置为`myModule`，这时`if`判断不成立，就不会执行`print("sum=", sum(1, 100))`。

(2) 模块名的定义要尽量避免定义的模块名与Python的内置模块（即标准库）重名。

模块名	描述
<code>sys</code>	与Python解释器及其环境操作相关的标准库
<code>time</code>	提供与时何相关的各种函数的标准库
<code>os</code>	提供了访问操作系统服务功能的标准库
<code>calendar</code>	提供与日期相关的各种函数的标准库
<code>urllib</code>	用于读取来自网上（服务器上）的数据的标准库
<code>json</code>	用于使用JSON序列化和反序列化对象
<code>re</code>	用于在字符串中执行正则表达式匹配和替换
<code>math</code>	提供算术运算函数的标准库
<code>decimal</code>	用于进行精确控制运算精度、有效数位和四舍五入操作的十进制运算
<code>shutil</code>	用于进行高级文件操作，如复制、移动和重命名等
<code>logging</code>	提供了灵活的记录事件、错误、警告和调试信息等日志信息的功能
<code>tkinter</code>	使用Python进行GUI编程的标准库

题1. 下面不属于Python标准库的是 ( )。

- A. `os`      B. `pandas`      C. `turtle`      D. `random`



# 1. 异常的定义和分类

异常类型	含义
<code>AssertionError</code>	当 <code>assert</code> 关键字后的条件为假时，程序运行会停止并抛出 <code>AssertionError</code> 异常
<code>AttributeError</code>	当试图访问的对象属性不存在时抛出的异常
<code>IndexError</code>	超出序列范围会引发此异常
<code>KeyError</code>	字典中查找一个不存在的关键字时引发此异常
<code>NameError</code>	尝试访问一个未声明的变量时，引发此异常
<code>TypeError</code>	不同类型数据之间的无效操作
<code>ZeroDivisionError</code>	除法运算中除数为0引发此异常

## 1. 文件打开和关闭

打开模式用于控制使用何种方式打开文件, `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 <code>FileNotFoundException</code> , 默认值
'w'	覆盖写模式，文件不存在则创建，存在则完全覆盖原文件
'x'	创建写模式，文件不存在则创建，存在则返回异常 <code>FileExistsError</code>
'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

## 2. 文件的读写

蜂考

### 1) 读文件

根据打开方式不同, 文件读写也会根据文本文件或二进制打开方式有所不同。

表中给出了Python语言常用的文件读取方法, 以`f`代表文件变量。

方法	含义
<code>f.read(size=-1)</code>	从文件中读入整个文件内容。参数可选, 如果给出, 读入前 <code>size</code> 长度的字符串或字节流
<code>f.readline(size=-1)</code>	从文件中读入一行内容。参数可选, 如果给出, 读入该行前 <code>size</code> 长度的字符串或字节流
<code>f.readlines(hint=-1)</code>	从文件中读入所有行, 以每行为元素形成一个列表。参数可选, 如果给出, 读入 <code>hint</code> 行
<code>f.seek(offset)</code>	改变当前文件操作指针的位置, <code>offset</code> 的值: 0为文件开头; 2为文件结尾

如果文件不大，可以一次性将文件内容读入，保存到程序内部变量中。f.read()是最常  
用的一次性读入文件的函数，其结果是一个字符串。

```
>>> f = open("D:/ bar.txt", "r")
>>> s = f.read()
>>> print(s)
>>> f.close()
```

f.readline()函数用于读取文件中的一行，包含最后的换行符“\n”。

创建的my\_file.txt文件为例，该文件中有如下2行数据：

Python程序设计  
人生苦短，我用Python

```
f = open("my_file.txt")#读取一行数据
byt = f.readline()
print(byt)
```

运行结果：

Python程序设计



f.readlines()也是一次性读入文件的函数，其结果是一个列表，每个元素是文件的一行。

```
f = open("my_file.txt")
byt = f.readlines()
print(byt)
```

运行结果：

```
[ 'Python程序设计\n' , '人生苦短，我用Python\n' ]
```

f.seek()方法能够移动读取指针的位置，f.seek(0)将读取指针移动到文件头部，f.seek(2)将读取指针移动到文件结尾。

从文本文件中逐行读入内容并进行处理是一个基本的文件操作需求。文本文件可以看作由行组成的组合类型，因此，可以使用遍历循环逐行遍历文件，使用方法如下：

```
f = open(<文件路径及名称>, "r")
for line in f:
    #处理一行数据
f.close()
```

提示：逐行遍历文件条件

逐行遍历文件仅针对文本方式，二进制方式打开没有行的概念。

蜂考

## CSV操作

CSV文件又称为逗号分隔值文件，是一种通用的、相对简单的文件格式，用以存储表格数据，包括数字或者字符。CSV是电子表格和数据库中最常见的输入、输出文件格式。

### 1) CSV文件写入

#### (1) csv.writer()

csv模块中的writer类可用于读写序列化的数据，其语法格式如下：

```
writer(csvfile, dialect='excel', **fmtparams)
```

参数说明：

csvfile：必须是支持迭代(Iterator)的对象，可以是文件(file)对象或者列表(list)对象。

dialect：编码风格，默认为excel的风格，也就是使用逗号‘，’分隔。

fmtparam：格式化参数，用来覆盖之前dialect对象指定的编码风格。