第2章 运算符、表达式与内置对象

2.1 Python常用内置对象

- 对象是python语言中最基本的概念，在python中处理的一切都是对象。
- python中有许多内置对象可供编程者使用，内置对象可直接使用，如数字、字符串、列表、元组、字典、集合以及内置函数sum()、max()、sorted()、min()等。
- 非内置对象需要导入模块才能使用，如正弦函数sin(x)，随机数产生函数random( )等。

2.1 Python常用内置对象

常用内置对象

对象类型	类型名称	示例	简要说明
数字	int float complex	1234 3.14, 1.3e5 3+4j	整数**大小没有限制**，内置支持复数及其运算
字符串	str	'swfu' "I'm student" '''Python ''', r'abc', R'bcd'	使用单引号、双引号、三引号作为定界符，以字母r或R引导表示原始字符串
字节串	bytes	b'hello world'	以字母b引导，可以使用单引号、双引号、三引号作为定界符
列表	list	[1, 2, 3] ['a', 'b', ['c', 2]]	所有元素放在一对方括号中，元素之间使用逗号分隔，其中的元素可以是任意类型
字典	dict	{1:'food', 2:'taste', 3:'import'}	所有元素放在一对大括号中，元素之间使用逗号分隔，**元素形式为“键:值”**
元组	tuple	(2, -5, 6) (3,)	**不可变**，所有元素放在一对圆括号中，元素之间使用逗号分隔，**如果元组中只有一个元素的话，后面的逗号不能省略**
集合	set	{'a', 'b', 'c'}	所有元素放在一对大括号中，元素之间使用逗号分隔，**元素必须可哈希且唯一**

2.1 Python常用内置对象

续表

对象类型	类型名称	示例	简要说明
布尔型	bool	True, False	逻辑值，关系运算符、成员测试运算符、同一性测试运算符组成的表达式的值一般为True或False
空类型	NoneType	None	空值
异常	Exception ValueError TypeError		Python内置大量异常类，分别对应不同类型的异常
文件		f = open('data.dat', 'rb')	open是Python内置函数，使用指定的模式打开文件，返回文件对象
迭代器对象		生成器对象、zip对象、enumerate对象、map对象、filter对象等等	具有惰性求值的特点，其中的元素只能使用一次
可调用对象	function method class	函数和方法使用def定义 类使用class定义	可调用对象包括函数、方法、lambda表达式、类、实现了特殊方法__call__()的类的对象
模块	module		模块用来集中存放函数、类、常量或其他对象

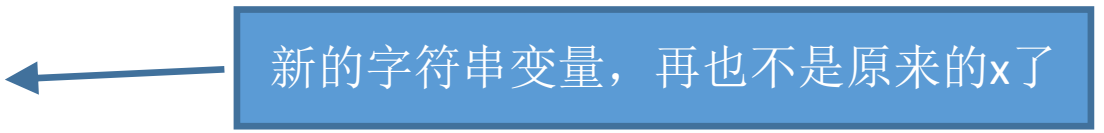
2.1.1 常量与变量

- 在Python中，不需要事先声明变量名及其类型，直接赋值即可创建各种类型的对象变量。这一点适用于Python任意类型的对象。

例如语句


```
>>> x = 3
```

创建了整型变量x，并赋值为3，再例如语句



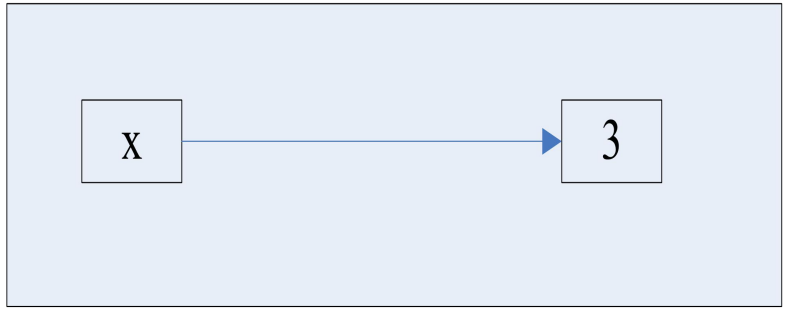

```
>>> x = 'Hello world.'
```

创建了字符串变量x，并赋值为'Hello world.'。

2.1.1 常量与变量

- 赋值语句的执行过程是：首先把等号右侧表达式的值计算出来，然后在内存中寻找一个位置把值存放进去，最后创建变量并指向这个内存地址。

```
>>> x = 3
```



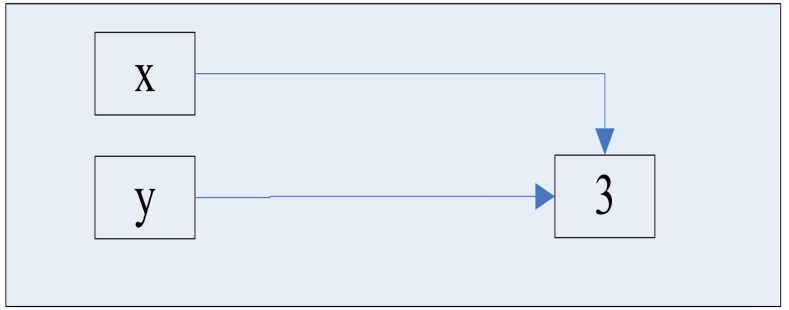
- Python中的变量并不直接存储值，而是存储了值的内存地址或者引用，这也是变量类型随时可以改变的原因。

2.1.1 常量与变量

- 在Python中，允许多个变量指向同一个值，例如：

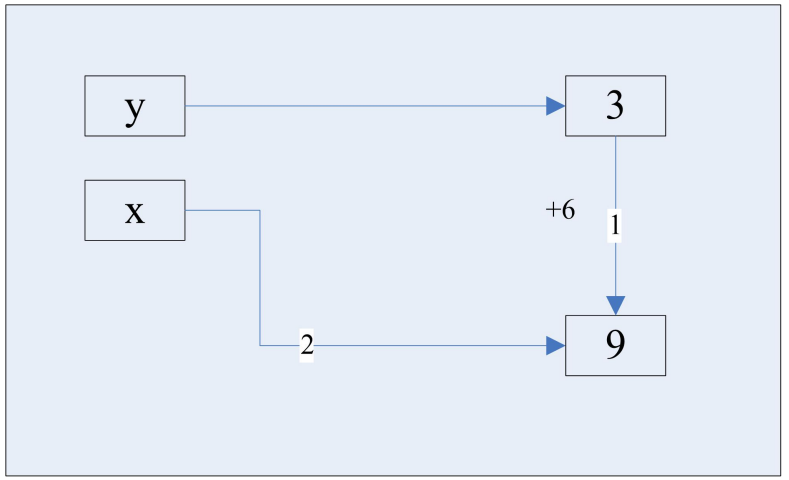
```
>>> x = 3
>>> id(x)
1786684560
>>> y = x
>>> id(y)
1786684560
```

id()函数用来查看对象的内存地址



- 接着上面的代码再继续执行下面的代码：

```
>>> x += 6
>>> id(x)
1786684752
>>> y
3
>>> id(y)
1786684560
```



2.1.1 常量与变量

- Python采用的是基于值的内存管理方式，如果为不同变量赋值为相同值（交互模式下仅适用于-5至256的整数和短字符串），这个值在内存中只有一份，多个变量指向同一块内存地址。

```
>>> x = 3
>>> id(x)
140720461453136
>>> y = 3
>>> id(y)
140720461453136
>>> x = [1, 1, 1, 1]
>>> id(x[0]) == id(x[1])
True
```

2.1.1 常量与变量

- 交互模式下同一个语句中直接赋值的任意大整数变量具有同样的内存地址，但不适用于表达式。

```
>>> x, y = 2235111150895389301249381342232951352130031828640647616754238557881163071896851147531724784530814582097445922881737078370567562092223091453615905548102178073175097305670733077683179094016, 2235111150895389301249381342232951352130031828640647616754238557881163071896851147531724784530814582097445922881737078370567562092223091453615905548102178073175097305670733077683179094016
>>> x is y
True
>>> x, y = 666 ** 6, 666 ** 6
>>> x is y
True
>>> x, y = 666**66, 666**66          # 表达式的值太大时不适用，与直接赋值整数不同
>>> x is y
False
```

2.1.1 常量与变量

- 同一个程序中的任意大整数具有同样的内存地址，但不适用于超大值的表达式。

```
x = 99999999999999999999999999999999999999999999999
y = 99999999999999999999999999999999999999999999999
print(id(x) == id(y))
print(x is y)
x = 9999 ** 9
y = 9999 ** 9
print(id(x) == id(y))
print(x is y)
x = 9999 ** 99
y = 9999 ** 99
print(id(x) == id(y))
print(x is y)
```

```
True
True
True
True
False
False
```

2.1.1 常量与变量

❖Python属于**强类型编程语言**，Python解释器会根据赋值或运算来自动推断变量类型。Python还是一种**动态类型语言**，变量的类型也是可以随时变化的。

```
>>> x = 3
>>> print(type(x))
<class 'int'>
>>> x = 'Hello world.'
>>> print(type(x))                    # 查看变量类型
<class 'str'>
>>> x = [1,2,3]
>>> print(type(x))
<class 'list'>
>>> isinstance(3, int)                # 测试对象是否是某个类型的实例
True
>>> isinstance('Hello world', str)
True
```

2.1.1 常量与变量

- 在定义变量名、函数名、类名、方法名等标识符时，需要注意以下问题：

✓ **必须**以汉字、英文字母或下划线开头，但以下划线开头的变量在Python中有特殊含义；

✓ **不能**包含空格以及标点符号（括号、引号、逗号、斜线、反斜线、冒号、句号、问号等等）；

✓ **不能**使用关键字，可以导入keyword模块后使用print(keyword.kwlist)查看所有Python关键字；

✓ 对英文字母的**大小写敏感**，例如student和Student是不同的变量；

✓ **不建议**使用系统内置的模块名、类型名或函数名以及已导入的模块名及其成员名，这将会改变其类型和含义，可以通过dir(__builtins__)查看所有内置模块、类型和函数。

```
>>> id(3)
1667343520
>>> id = '3724....'
>>> id(3)
TypeError: 'str' object is not callable
```

2.1.2 数字

- Python支持任意大的整数，具体可以大到什么程度仅受内存大小的限制。
- 由于精度的问题，对于实数运算可能会有一定的误差，应尽量避免在实数之间直接进行相等性测试，而是应该以二者之差的绝对值是否足够小作为两个实数是否相等的依据，或者使用标准库函数math.isclose()测试两个实数是否足够接近。
- 在数字的算术运算表达式求值时会进行隐式的类型转换，如果存在复数则都变成复数，如果没有复数但是有实数就都变成实数，如果都是整数则不进行类型转换。

2.1.2 数字

```
>>> 9999 ** 99              # 这里**是幂乘运算符，等价于内置函数pow()
99014835352672348760226312475328262557055952889579105732432652912179483789405351346442217682691643393258692438667776624403200162375682140043297505120882020498009873555270384136230466997051069124380021820284037432937880069492030979195418511779843432959121215910629869993866990806757337472433120894242554489391091007320504903165678922088956073296292622630586570659359491789627675639684851490098999
>>> 0b10101                 # 二进制整数，以0b开头，每位数字只能是0或1
21
>>> 0o777                   # 八进制整数，以0o开头，每位数字只能是0-7
511
>>> 0xffffff                # 十六进制整数，以0x开头，每位数字可以是0-9或a-f
                            # 其中a表示10、b表示11、c表示12、d表示13、e表示14、f表示15
16777215
```

2.1.2 数字

```
>>> 0.3 + 0.2                          # 实数相加
0.5
>>> 0.4 - 0.1                          # 实数相减，结果稍微有点偏差
0.30000000000000004
>>> 0.4 - 0.1 == 0.3                   # 应尽量避免直接比较两个实数是否相等
False
>>> abs(0.4-0.1 - 0.3) < 1e-6          # 这里1e-6表示10的-6次方
True
>>> import math
>>> math.isclose(0.4-0.1, 0.3)         # 使用标准库函数判断两个实数是否足够接近
True
```

2.1.2 数字

- Python内置支持复数类型及其运算，并且形式与数学上的复数完全一致。

```
>>> x = 3 + 4j                          # 使用j或J表示复数虚部
>>> y = 5 + 6j
>>> x + y                               # 支持复数之间的加、减、乘、除以及幂乘等运算
(8+10j)
>>> x * y
(-9+38j)
>>> abs(x)                              # 内置函数abs()可用来计算复数的模
5.0
>>> x.imag                              # 虚部
4.0
>>> x.real                              # 实部
3.0
>>> x.conjugate()                       # 共轭复数
(3-4j)
>>> 3 + 4j.imag
7.0
>>> (3 + 4j).imag
4.0
```

2.1.2 数字

- Python 3.6.x之后的版本支持在数字中间位置使用单个下划线对数字分组来提高可读性，类似于数学上使用逗号作为千位分隔符。
- 在Python数字中单个下划线可以出现在中间任意位置，但不能出现开头和结尾位置，也不能使用多个连续的下划线。

```
>>> 1_000_000
1000000
>>> 1_2_3_4
1234
>>> 1_2 + 3_4j
(12+34j)
>>> 1_2.3_45
12.345
```

2.1.2 数字

- Python标准库fractions中的Fraction对象支持分数及其运算。

```
>>> from fractions import Fraction
>>> x = Fraction(3, 5)      # 创建分数对象
>>> y = Fraction(3, 7)
>>> x
Fraction(3, 5)
>>> x ** 2                  # 幂运算
Fraction(9, 25)
>>> x.numerator             # 查看分子
3
>>> x.denominator           # 查看分母
5
>>> x + y                   # 支持分数之间的四则运算，自动进行通分
Fraction(36, 35)
>>> x / y                   # 分数之间的除法
Fraction(7, 5)
>>> x * 3                   # 支持与实数的四则运算
Fraction(9, 5)
>>> sum([Fraction(3,5), Fraction(5,6)])          # 支持内置函数sum()
Fraction(43, 30)
>>> sum([Fraction(3,5), Fraction(5,6)]) / 2
Fraction(43, 60)
```

2.1.2 数字

- 标准库decimal中提供的Decimal类实现了更高精度实数的运算。

```
>>> from decimal import Decimal, getcontext
>>> 1 / 3                                  # 内置实数类型，精度有限
0.3333333333333333
>>> 1 / 7
0.14285714285714285
>>> Decimal(1) / Decimal(3)                # 高精度实数运算
Decimal('0.33333333333333333333333333')
>>> Decimal(1) / Decimal(7)
Decimal('0.14285714285714285714285714')
>>> sum([Decimal('0.5'), Decimal('1.3'), Decimal('3.14')]) / 3     # 支持内置函数sum()
Decimal('1.6466666666666666666666667')
>>> getcontext().prec = 50                 # 设置更高精度，有效数字位数
>>> Decimal(1) / Decimal(3)
Decimal('0.33333333333333333333333333333333333333333333333333')
>>> Decimal(1) / Decimal(7)
Decimal('0.14285714285714285714285714285714285714285714285714')
```

2.1.2 数字

```
>>> getcontext().prec = 2                    # 有效数字位数的设置只在计算时起作用
>>> Decimal('.234567')
Decimal('0.234567')
>>> Decimal('.234567') + Decimal('0')
Decimal('0.23')
>>> Decimal('2.5') / Decimal('2')
Decimal('1.2')
>>> getcontext().rounding = 'ROUND_HALF_UP'    # 修改四舍五入的方式
>>> Decimal('2.5') / Decimal('2')
Decimal('1.3')
```

2.1.3 字符串与字节串

- 在Python中，没有字符常量和变量的概念，只有字符串类型的常量和变量，单个字符也是字符串。使用单引号、双引号、三单引号、三双引号作为定界符（delimiter）来表示字符串，并且不同的定界符之间可以互相嵌套。
- Python 3.x全面支持中文，中文和英文字母都作为一个字符对待，甚至可以使用中文作为变量名。
- 除了支持使用加号运算符连接字符串以外，Python字符串还提供了大量的方法支持格式化、查找、替换、排版等操作。

2.1.3 字符串与字节串

```
>>> x = 'Hello world.'                          # 使用单引号作为定界符
>>> x = "Python is a great language."           # 使用双引号作为定界符
>>> x = '''Tom said, "Let's go."'''             # 不同定界符之间可以互相嵌套
>>> print(x)
Tom said, "Let's go."
>>> x = 'good ' + 'morning'                     # 连接字符串
>>> x
'good morning'
>>> x = 'good ''morning'                        # 连接字符串，仅适用于字符串常量
>>> x
'good morning'
>>> x = 'good '
>>> x = x'morning'                              # 不适用于字符串变量
SyntaxError: invalid syntax
>>> x = x + 'morning'                           # 字符串变量之间的连接可以使用加号
>>> x
'good morning'
```

2.1.3 字符串与字节串

- 对str类型的字符串调用其encode()方法进行编码得到bytes字节串，对bytes字节串调用其decode()方法并指定正确的编码格式进行解码得到str字符串。

```
>>> type('Hello world')                          # 默认字符串类型为str
<class 'str'>
>>> type(b'Hello world')                         # 在定界符前加上字母b表示字节串
<class 'bytes'>
>>> 'Hello world'.encode('utf8')                 # 使用utf8编码格式进行编码
b'Hello world'
>>> 'Hello world'.encode('gbk')                  # 使用gbk编码格式进行编码
b'Hello world'
>>> '付国'.encode('utf8')                      # 对中文进行编码
b'\xe8\x91\xa3\xe4\xbb\x98\xe5\x9b\xbd'
>>> _.decode('utf8')                             # 单下划线表示最后一个正确计算的表达式的值
'董付国'
>>> '付国'.encode('gbk')
b'\xb6\xad\xb8\xb6\xb9\xfa'
>>> _.decode('gbk')                              # 对bytes字节串进行解码
'付国'
```

2.1.4 列表、元组、字典、集合

	列表	元组	字典	集合
类型名称	list	tuple	dict	set
定界符	方括号[]	圆括号()	大括号{}	大括号{}
是否可变	是	否	是	是
是否有序	是	是	否	否
是否支持下标	是（使用序号作为下标）	是（使用序号作为下标）	是（使用“键”作为下标）	否
元素分隔符	逗号	逗号	逗号	逗号
对元素形式的要求	无	无	键:值	必须可哈希
对元素值的要求	无	无	“键”必须可哈希	必须可哈希
元素是否可重复	是	是	“键”不允许重复，“值”可以重复	否
成员测试速度	非常慢	很慢	非常快	非常快
新增和删除元素速度	尾部操作快 其他位置慢	不允许	快	快

2.1.4 列表、元组、字典、集合

```
>>> x_list = [1, 2, 3]                          # 创建列表对象
>>> x_tuple = (1, 2, 3)                         # 创建元组对象
>>> x_dict = {'a':97, 'b':98, 'c':99}           # 创建字典对象
>>> x_set = {1, 2, 3}                           # 创建集合对象
>>> print(x_list[1])                            # 使用下标访问指定位置的元素
2
>>> print(x_tuple[1])                           # 元组也支持使用序号作为下标
2
>>> print(x_dict['a'])                          # 字典对象的下标是“键”
97
>>> 3 in x_set                                  # 成员测试
True
```

2.2 Python运算符与表达式

- Python是面向对象的编程语言，在Python中一切都是对象。对象由数据和行为两部分组成，而行为主要通过方法来实现，通过一些特殊方法的重写，可以实现运算符重载。运算符也是表现对象行为的一种形式，不同类的对象支持的运算符有所不同，同一种运算符作用于不同的对象时也可能会表现出不同的行为，这正是面向对象程序设计中“多态”的体现。
- 在Python中，单个常量或变量可以看作最简单的表达式，使用除赋值运算符之外的其他任意运算符和函数调用连接的式子也属于表达式。

2.2 Python运算符与表达式

- 运算符优先级遵循的规则为：成员访问运算符、下标运算符、算术运算符优先级最高，其次是位运算符、成员测试运算符、关系运算符、逻辑运算符等，算术运算符遵循“先乘除，后加减”的基本运算原则。
- 虽然Python运算符有一套严格的优先级规则，但是强烈建议在编写复杂表达式时使用圆括号来明确说明其中的逻辑来提高代码可读性。

2.2 Python运算符与表达式

运算符	功能说明
:=	赋值运算，Python 3.8新增，俗称海象运算符
lambda [parameter]: expression	用来定义lambda表达式，功能相当于函数，parameter相当于函数参数，可以没有；expression表达式的值相当于函数返回值
value1 if condition else value2	用来表示一个二选一的表达式，其中value1、condition、value2都为表达式，如果condition的值等价于True则整个表达式的值为value1的值，否则整个表达式的值为value2的值
or	“逻辑或”运算符，以exp1 or exp2为例，如果exp1的值等价于True则返回exp1的值，否则返回exp2的值
and	“逻辑与”运算符，以exp1 and exp2为例，如果exp1的值等价于False则返回exp1的值，否则返回exp2的值

2.2 Python运算符与表达式

not	“逻辑非”运算符，对于表达式not x，如果x的值等价于True则返回False，否则返回True
in、not in is、is not <、<=、>、>=、==、!=	成员测试，表达式x in y的值当且仅当y中包含元素x时才会为True； 测试两个对象是否为同一个对象的引用。如果两个对象是同一个对象的引用，那么它们的内存地址相同； 关系运算，用于比较大小，作用于集合时表示测试集合的包含关系； 这三组运算符具有相同的优先级
|	“按位或”运算，集合并集
^	“按位异或”运算，集合对称差集
&	“按位与”运算，集合交集
<<、>>	左移位、右移位

2.2 Python运算符与表达式

+ -	算术加法，列表、元组、字符串合并与连接； 算术减法，集合差集
* @ / // %	算术乘法，序列重复； 矩阵乘法，修饰器； 真除； 整除； 求余数，字符串格式化
+x -x ~x	正号 负号，相反数 按位求反
**	幂运算，指数可以为小数，例如3**0.5表示计算3的平方根
[] . ()	下标，切片； 属性访问，成员访问； 函数定义或调用，修改表达式计算顺序，声明多行代码为一个语句
[]、()、{}	定义列表、元组、字典、集合，列表推导式、生成器表达式、字典推导式、集合推导式

2.2.1 算术运算符

（1）+运算符除了用于算术加法以外，还可以用于列表、元组、字符串的连接，但不支持不同类型的对象之间相加或连接。

```
>>> [1, 2, 3] + [4, 5, 6]                    # 连接两个列表
[1, 2, 3, 4, 5, 6]
>>> (1, 2, 3) + (4,)                         # 连接两个元组
(1, 2, 3, 4)
>>> 'abcd' + '1234'                          # 连接两个字符串
'abcd1234'
>>> 'A' + 1                                  # 不支持字符与数字相加，抛出异常
TypeError: Can't convert 'int' object to str implicitly
>>> True + 3                                 # Python内部把True当作1处理
4
>>> False + 3                                # 把False当作0处理
3
```

2.2.1 算术运算符

（2）*运算符除了表示算术乘法，还可用于列表、元组、字符串这几个序列类型与整数的乘法，表示序列元素的重复，生成新的序列对象。字典和集合不支持与整数的相乘，因为其中的元素是不允许重复的。

```
>>> True * 3
3
>>> False * 3
0
>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> (1, 2, 3) * 3
(1, 2, 3, 1, 2, 3, 1, 2, 3)
>>> 'abc' * 3
'abcabcabc'
```

2.2.1 算术运算符

```
>>> data = [[1], [2], [3]]
>>> data_new = data * 3          # 对元素的引用进行重复
>>> data_new
[[1], [2], [3], [1], [2], [3], [1], [2], [3]]
>>> data[0] is data_new[0] is data_new[3] is data_new[6]
                                 # 新列表中元素与旧列表中元素具有相同的引用
True
```

2.2.1 算术运算符

（3）运算符/和//在Python中分别表示算术除法和算术求整商（floor division）。

```
>>> 3 / 2                          # 数学意义上的除法
1.5
>>> 15 // 4                        # 如果两个操作数都是整数，结果为整数
3
>>> 15.0 // 4                      # 如果操作数中有实数，结果为实数形式的整数值
3.0
>>> -15 // 4                         # 向下取整
-4
```

2.2.1 算术运算符

（4）%运算符可以用于整数或实数的求余数运算，还可以用于字符串格式化，但是这种用法并不推荐。

```
>>> 789 % 23                    # 余数
7
>>> -17 % 4
3
>>> '%c, %d' % (65, 65)         # 把65分别格式化为字符和整数
'A, 65'
>>> '%f,%s' % (65, 65)          # 把65分别格式化为实数和字符串
'65.000000,65'
```

2.2.1 算术运算符

（5）**运算符表示幂乘：

```
>>> 3 ** 2                          # 3的2次方，等价于pow(3, 2)
9
>>> pow(3, 2, 8)                    # 等价于(3**2) % 8
1
>>> 9 ** 0.5                        # 9的0.5次方，平方根
3.0
>>> (-9) ** 0.5                     # 可以计算负数的平方根，得到复数
(1.8369701987210297e-16+3j)
```

2.2.2 关系运算符

- Python关系运算符可以连用，其含义与我们日常的理解完全一致。使用关系运算符的一个最重要的前提是，操作数之间必须可比较大小。例如把一个字符串和一个数字进行大小比较是毫无意义的，所以Python也不支持这样的运算。

```
>>> 1 < 3 < 5                          # 等价于1 < 3 and 3 < 5
True
>>> 3 < 5 > 2
True
>>> 1 > 6 < 8
False
>>> 1 > 6 < math.sqrt(9)               # 具有惰性求值或者逻辑短路的特点
False
>>> 1 < 6 < math.sqrt(9)               # 还没有导入math模块，抛出异常
NameError: name 'math' is not defined
>>> import math
>>> 1 < 6 < math.sqrt(9)
False
```

2.2.3 成员测试运算符in与同一性测试运算符is

- 成员测试运算符in用于成员测试，即测试一个对象是否为另一个对象的元素。

```
>>> 3 in [1, 2, 3]                    # 测试3是否存在于列表[1, 2, 3]中
True
>>> 5 in range(1, 10, 1)              # range()是用来生成指定范围整数的内置函数
True
>>> 'abc' in 'abcdefg'                # 测试是否为子字符串
True
>>> for i in (3, 5, 7):               # 循环，元素遍历
    print(i, end='\t')
3       5       7
```

2.2.3 成员测试运算符in与同一性测试运算符is

- 同一性测试运算符is用来测试两个对象是否是同一个，如果是则返回True，否则返回False。如果两个对象是同一个，二者具有相同的内存地址。

```
>>> 3 is 3
True
>>> x = [300, 300, 300]
>>> x[0] is x[1]                    # 基于值的内存管理，同一个值在内存中只有一份
True
>>> x = [1, 2, 3]
>>> y = [1, 2, 3]
>>> x is y                          # 上面形式创建的x和y不是同一个列表对象
False
>>> x[0] is y[0]
True
```

2.2.4 位运算符与集合运算符

```
>>> 3 << 2                    # 把3左移2位
12
>>> 3 & 7                     # 位与运算
3
>>> 3 | 8                     # 位或运算
11
>>> 3 ^ 5                     # 位异或运算
6
```

位与运算：1&1=1、1&0=0&1=0&0=0

位或运算：1|1=1|0=0|1=1、0|0=0

位异或运算：1^1=0^0=0、1^0=0^1=1

左移位时右侧补0，每左移一位相当于乘以2

移位时左侧补0，每右移一位相当于整除以2

2.2.4 位运算符与集合运算符

- 集合的交集、并集、对称差集等运算借助于位运算符来实现，而差集则使用减号运算符实现（注意，并集运算符不是加号）。

```
>>> {1, 2, 3} | {3, 4, 5}                # 并集，自动去除重复元素
{1, 2, 3, 4, 5}
>>> {1, 2, 3} & {3, 4, 5}                # 交集
{3}
>>> {1, 2, 3} - {3, 4, 5}                # 差集
{1, 2}
>>> {1, 2, 3} ^ {3, 4, 5}                # 对称差集
{1, 2, 4, 5}
```

2.2.5 逻辑运算符

- 逻辑运算符and、or、not常用来连接多个条件表达式构成更加复杂的条件表达式，并且and和or具有惰性求值或逻辑短路的特点，当连接多个表达式时只计算必须要计算的值。
- 例如表达式“exp1 and exp2”等价于“exp1 if not exp1 else exp2”，而表达式“exp1 or exp2”则等价于“exp1 if exp1 else exp2”。
- 在编写复杂条件表达式时充分利用这个特点，合理安排不同条件的先后顺序，在一定程度上可以提高代码运行速度。
- 另外要注意的是，运算符and和or并不一定会返回True或False，而是得到最后一个被计算的表达式的值，但是运算符not一定会返回True或False。

2.2.5 逻辑运算符

```
>>> 3>5 and a>3                    # 注意，此时并没有定义变量a
False
>>> 3>5 or a>3                     # 3>5的值为False，所以需要计算后面表达式
NameError: name 'a' is not defined
>>> 3<5 or a>3                     # 3<5的值为True，不需要计算后面表达式
True
>>> 3 and 5                        # 最后一个计算的表达式的值作为整个表达式的值
5
>>> 3 and 5>2
True
>>> 3 not in [1, 2, 3]            # 逻辑非运算not
False
>>> 3 is not 5                     # not的计算结果只能是True或False之一
True
>>> not 3
False
>>> not 0
True
```

2.2.7 补充说明

- Python还有赋值分隔符=和+=、-=、*=、/=、//=、**=、|=、^=等大量复合赋值分隔符。例如，x += 3在语法上等价（注意，在功能的细节上可能会稍有区别）于x = x + 3。
- Python不支持++和--运算符，虽然在形式上有时候似乎可以这样用，但实际上是另外的含义，要注意和其他语言的区别。

2.2.7 补充说明

```
>>> i = 3
>>> ++i                                  # 正正得正
3
>>> +(+3)                                # 与++i等价
3
>>> i++                                  # Python不支持++运算符，语法错误
SyntaxError: invalid syntax
>>> --i                                  # 负负得正，等价于-(-i)
3
>>> ---i                                 # 等价于-(-(-i))
-3
>>> i--                                  # Python不支持--运算符，语法错误
SyntaxError: invalid syntax
```

2.3 Python关键字简要说明

- Python关键字只允许用来表达特定的语义，不允许通过任何方式改变它们的含义，也不能用来做变量名、函数名或类名等标识符。
- 在Python开发环境中导入模块keyword之后，可以使用print(keyword.kwlist)查看所有关键字。

2.3 Python关键字简要说明

关键字	含义
False	常量，逻辑假，首字母必须大写
None	常量，空值，首字母必须大写
True	常量，逻辑真，首字母必须大写
and	逻辑与运算
as	在import或except语句中给对象起别名
assert	断言，用来确认某个条件必须满足，可用来帮助调试程序
break	用在循环结构中，提前结束break所在层次的循环
class	用来定义类
continue	用在循环结构中，提前结束本次循环
def	用来定义函数
del	用来删除对象或对象成员
elif	用在选择结构中，表示else if的意思
else	可以用在选择结构、循环结构和异常处理结构中
except	用在异常处理结构中，用来捕获特定类型的异常
finally	用在异常处理结构中，用来表示不论是否发生异常都会执行的代码
for	构造for循环，用来遍历可迭代对象中的所有元素
from	明确指定从哪个模块中导入什么对象，例如from math import sin; 还可以与yield一起构成yield表达式，以及raise...from...表达式

2.3 Python关键字简要说明

关键字	含义
global	声明全局变量
if	用在选择结构中，检查条件是否成立
import	用来导入模块或模块中的对象
in	成员测试
is	同一性测试
lambda	用来定义lambda表达式，类似于函数
nonlocal	用来声明nonlocal变量
not	逻辑非运算
or	逻辑或运算
pass	空语句，执行该语句时什么都不做，常用作占位符
raise	用来显式抛出异常
return	在函数中用来返回值，如果没有指定返回值，表示返回空值None
try	在异常处理结构中用来限定可能会引发异常的代码块
while	用来构造while循环结构，只要条件表达式等价于True就重复执行限定的代码块
with	上下文管理，具有自动管理资源的功能
yield	在生成器函数中用来生成值

2.4.1 类型转换与类型判断

- ord()和chr()是一对功能相反的函数，ord()用来返回单个字符的Unicode码，而chr()则用来返回Unicode编码对应的字符，str()直接将任意类型参数转换为字符串或者把字节串解码为字符串。

```
>>> ord('a')                # 查看指定字符的Unicode编码
97
>>> chr(65)                 # 返回Unicode编码65对应的字符
'A'
>>> chr(ord('A')+1)         # Python不支持字符串和数字之间的加法操作
'B'
>>> chr(ord('国')+1)        # 支持中文
'图'
>>> ''.join(map(chr, (33891, 20184, 22269)))
'董付国'
```

2.4.1 类型转换与类型判断

```
>>> str(b'Python\xe5\xb0\x8f\xe5\xb1\x8b', 'utf8')  # 把字节串解码为字符串
'Python小屋'
>>> str(1234)                                       # 直接变成字符串
'1234'
>>> str([1,2,3])
'[1, 2, 3]'
>>> str((1,2,3))
'(1, 2, 3)'
>>> str({1,2,3})
'{1, 2, 3}'
```

2.4.1 类型转换与类型判断

- 内置类ascii可以把对象转换为ASCII码表示形式，必要的时候使用转义字符来表示特定的字符。

```
>>> ascii('a')
"'a'"
>>> ascii('董付国')
"'\\u8463\\u4ed8\\u56fd'"
>>> eval(_)                                    # 对字符串进行求值
'董付国'
```

2.4.1 类型转换与类型判断

- list()、tuple()、dict()、set()用来把其他类型的数据转换成为列表、元组、字典、集合，或者创建空列表、空元组、空字典和空集合。

```
>>> list(range(5))                      # 把range对象转换为列表
[0, 1, 2, 3, 4]
>>> tuple(_)                            # 一个下划线表示上一次正确的输出结果
(0, 1, 2, 3, 4)
>>> dict(zip('1234', 'abcde'))          # 创建字典
{'1': 'a', '2': 'b', '3': 'c', '4': 'd'}
>>> set('1112234')                      # 创建可变集合，自动去除重复
{'4', '2', '3', '1'}
>>> _.add('5')
>>> _
{'2', '1', '3', '4', '5'}
```

2.4.1 类型转换与类型判断

- 内置函数type()和isinstance()可以用来判断数据类型，常用来对函数参数进行检查，可以避免错误的参数类型导致函数崩溃或返回意料之外的结果。

```
>>> type(3)                                    # 查看3的类型
<class 'int'>
>>> type([3])                                  # 查看[3]的类型
<class 'list'>
>>> type({3}) in (list, tuple, dict)           # 判断{3}是否为list,tuple或dict类型的实例
False
>>> type({3}) in (list, tuple, dict, set)      # 判断{3}是否为list,tuple,dict或set的实例
True
>>> isinstance(3, int)                         # 判断3是否为int类型的实例
True
>>> isinstance(3j, int)
False
>>> isinstance(3j, (int, float, complex))      # 判断3是否为int,float或complex类型
True
```

2.4.2 最值与求和

- max()、min()、sum()这三个内置函数分别用于计算列表、元组或其他包含有限个元素的可迭代对象中所有元素最大值、最小值以及所有元素之和。
- sum()默认（可以通过start参数来改变）支持包含数值型元素的序列或可迭代对象，max()和min()则要求序列或可迭代对象中的元素之间可比较大小。

```
>>> from random import randint
>>> a = [randint(1,100) for i in range(10)]  # 包含10个[1,100]之间随机数的列表
>>> print(max(a), min(a), sum(a))            # 最大值、最小值、所有元素之和
>>> sum(a) / len(a)                          # 平均值
```

2.4.2 最值与求和

- 函数max()和min()还支持default参数和key参数，其中default参数用来指定可迭代对象为空时默认返回的最大值或最小值，key参数用来指定比较大小的依据或规则，可以是函数或lambda表达式。函数sum()还支持start参数，用来控制求和的初始值。

```
>>> max(['2', '111'])                    # 不指定排序规则
'2'
>>> max(['2', '111'], key=len)           # 返回最长的字符串
'111'
>>> print(max([], default=None))         # 对空列表求最大值，返回空值None
None
```

2.4.2 最值与求和

```
>>> from random import randint
>>> lst = [[randint(1,50) for i in range(5)] for j in range(30)]
>>> max(*lst, key=sum)
[47, 40, 47, 23, 36]
>>> max(lst, key=sum)
[47, 40, 47, 23, 36]
>>> max(lst, key=lambda x: x[1])
[41, 45, 19, 39, 33]
>>> sum(range(1, 11))
55
>>> sum(range(1, 11), 5)
60
>>> sum([[1,2], [3], [4]], [])
[1, 2, 3, 4]
>>> sum(2**i for i in range(200))
1606938044258990275541962092341162602522202993782792835301375
>>> int('1'*200, 2)
1606938044258990275541962092341162602522202993782792835301375
>>> int('1'*200, 7)
1743639715219059529169816601969468943303198091695038943325023347339187627904043708629063769
1515606750488442080420910523623438633906139318646917923778899694224395760200000
```

2.4.3 基本输入输出

- input()和print()是Python的基本输入输出函数，前者用来接收用户的键盘输入，后者用来把数据以指定的格式输出到标准控制台或指定的文件对象。不论用户输入什么内容，input()一律返回字符串对待，必要的时候可以使用内置函数int()、float()或eval()对用户输入的内容进行类型转换。

2.4.3 基本输入输出

```
>>> x = input('Please input: ')
Please input: 345
>>> x
'345'
>>> type(x)                          # 把用户的输入作为字符串对待
<class 'str'>
>>> int(x)                           # 转换为整数
345
>>> eval(x)                          # 对字符串求值，或类型转换
345
>>> x = input('Please input: ')
Please input: [1, 2, 3]
>>> x
'[1, 2, 3]'
>>> type(x)
<class 'str'>
>>> eval(x)
[1, 2, 3]
```

2.4.3 基本输入输出

- 内置函数print()用于输出信息到标准控制台或指定文件，语法格式为：

```
print(value1, value2, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

✓sep参数之前为需要输出的内容（可以有任意多个）；

✓sep参数用于指定数据之间的分隔符，默认为空格；

✓end参数用于指定输出完数据之后再输出什么字符；

✓file参数用于指定输出位置，默认为标准控制台，也可以重定向输出到文件。

```
>>> print(1, 3, 5, 7, sep='\t')          # 修改默认分隔符
1       3       5       7
>>> for i in range(10):                  # 修改end参数，每个输出之后不换行
    print(i, end=' ')
0 1 2 3 4 5 6 7 8 9
>>> with open('test.txt', 'a+') as fp:
    print('Hello world!', file=fp)       # 重定向，将内容输出到文件中
```

2.4.4 排序与逆序

- sorted()对列表、元组、字典、集合或其他可迭代对象进行排序并返回新列表，reversed()对可迭代对象（生成器对象和具有惰性求值特性的zip、map、filter、enumerate等迭代器对象除外）进行翻转（首尾交换）并返回可迭代的reversed对象。

```
>>> x = list(range(11))
>>> import random
>>> random.shuffle(x)                          # 随机打乱顺序
>>> x
[2, 4, 0, 6, 10, 7, 8, 3, 9, 1, 5]
>>> sorted(x)                                  # 以默认规则升序排序
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> sorted(x, key=lambda item:len(str(item)), reverse=True)
                                               # 按转换成字符串以后的长度降序排列
[10, 2, 4, 0, 6, 7, 8, 3, 9, 1, 5]
>>> sorted(x, key=str)                         # 按转换成字符串以后的大小升序排列
[0, 1, 10, 2, 3, 4, 5, 6, 7, 8, 9]
```

2.4.4 排序与逆序

```
>>> x = ['aaaa', 'bc', 'd', 'b', 'ba']
>>> sorted(x, key=lambda item: (len(item), item))
                                                    # 先按长度排序，长度一样的正常排序
['b', 'd', 'ba', 'bc', 'aaaa']
>>> sorted(x, key=len)
['d', 'b', 'bc', 'ba', 'aaaa']
>>> x = [[1,2,3], [2,1,4], [2,2,1]]
>>> sorted(x, key=lambda item: (item[1],-item[2]))
[[2, 1, 4], [1, 2, 3], [2, 2, 1]]
```

2.4.4 排序与逆序

```
>>> game_result = [['Bob', 95.0, 'A'], ['Alan', 86.0, 'C'], ['Mandy', 83.5, 'A'],
                   ['Rob', 89.3, 'E']]
>>> from operator import itemgetter
>>> sorted(game_result, key=itemgetter(2))          # 按子列表第3各元素进行升序排列
[['Bob', 95.0, 'A'], ['Mandy', 83.5, 'A'], ['Alan', 86.0, 'C'], ['Rob', 89.3, 'E']]
>>> sorted(game_result, key=itemgetter(2,0))        # 先按子列表中第3个元素升序，
                                                    # 再按第1个元素升序
[['Bob', 95.0, 'A'], ['Mandy', 83.5, 'A'], ['Alan', 86.0, 'C'], ['Rob', 89.3, 'E']]
>>> sorted(game_result, key=itemgetter(2,0), reverse=True)
[['Rob', 89.3, 'E'], ['Alan', 86.0, 'C'], ['Mandy', 83.5, 'A'], ['Bob', 95.0, 'A']]
>>> list1 = ["what", "I'm", "sorting", "by"]
>>> list2 = ['something', 'else', 'to', 'sort']
>>> pairs = zip(list1, list2)
>>> [item[1] for item in sorted(pairs, reverse=True)]
                                                    # 根据list1的内容对list2进行排序
['something', 'to', 'sort', 'else']
```

2.4.4 排序与逆序

```
>>> reversed(x)                                  # 逆序，返回reversed对象
<list_reverseiterator object at 0x0000000003089E48>
>>> list(reversed(x))                            # reversed对象是可迭代的
['ba', 'b', 'd', 'bc', 'aaaa']
```

2.4.5 枚举

- enumerate()函数用来枚举可迭代对象中的元素，返回可迭代的enumerate对象，其中每个元素都是包含计数和值的元组。

```
>>> list(enumerate('abcd'))                              # 枚举字符串中的元素
[(0, 'a'), (1, 'b'), (2, 'c'), (3, 'd')]
>>> list(enumerate(['Python', 'Great']))                 # 枚举列表中的元素
[(0, 'Python'), (1, 'Great')]
>>> list(enumerate({'a':97, 'b':98, 'c':99}.items())) # 枚举字典中的元素
[(0, ('a', 97)), (1, ('b', 98)), (2, ('c', 99))]
>>> for index, value in enumerate(range(10, 15)):      # 枚举range对象中的元素
    print((index, value), end=' ')
(0, 10) (1, 11) (2, 12) (3, 13) (4, 14)
```

2.4.7 range()

- range()是Python开发中非常常用的一个内置函数，语法格式为range([start,] stop [, step])，有range(stop)、range(start, stop)和range(start, stop, step)三种用法。该函数返回range对象，其中包含左闭右开区间[start,stop)内以step为步长的整数。参数start默认为0，step默认为1。

```
>>> range(5)                          # start默认为0，step默认为1
range(0, 5)
>>> list(_)
[0, 1, 2, 3, 4]
>>> list(range(1, 10, 2))             # 指定起始值和步长
[1, 3, 5, 7, 9]
>>> list(range(9, 0, -2))             # 步长为负数时，start应比stop大
[9, 7, 5, 3, 1]
```

2.4.8 zip()

- zip()函数用来把多个可迭代对象中的元素组合到一起，返回一个可迭代的zip对象，其中每个元素都是包含原来的多个可迭代对象对应位置上元素的元组，如同拉拉链一样。

```
>>> list(zip('abcd', [1, 2, 3]))                # 组合字符串和列表中对应位置的元素
[('a', 1), ('b', 2), ('c', 3)]
>>> list(zip('123', 'abc', ',.!'))              # 处理3个序列
[('1', 'a', ','), ('2', 'b', '.'), ('3', 'c', '!')]
>>> x = zip('abcd', '1234')
>>> list(x)
[('a', '1'), ('b', '2'), ('c', '3'), ('d', '4')]
```

2.5 精彩案例赏析

❖**例2-1** 用户输入一个三位自然数，计算并输出其佰位、十位和个位上的数字。

```
x = input('请输入一个三位数：')
x = int(x)
a = x // 100
b = x // 10 % 10
c = x % 10
print(a, b, c)
```

想一想，还有别的办法吗？

2.5 精彩案例赏析

❖**例2-2** 已知三角形的两边长及其夹角，求第三边长。

```
import math

x = input('输入两边长及夹角（度）: ')
a, b, theta = map(float, x.split())
c = math.sqrt(a**2 + b**2 - 2*a*b*math.cos(theta*math.pi/180))
print('c=', c)
```

2.5 精彩案例赏析

❖**例2-3** 任意输入三个英文单词（使用逗号分隔），按字典顺序输出。

```
s = input('x,y,z=')
x, y, z = sorted(s.split(','))
print(x, y, z)
```