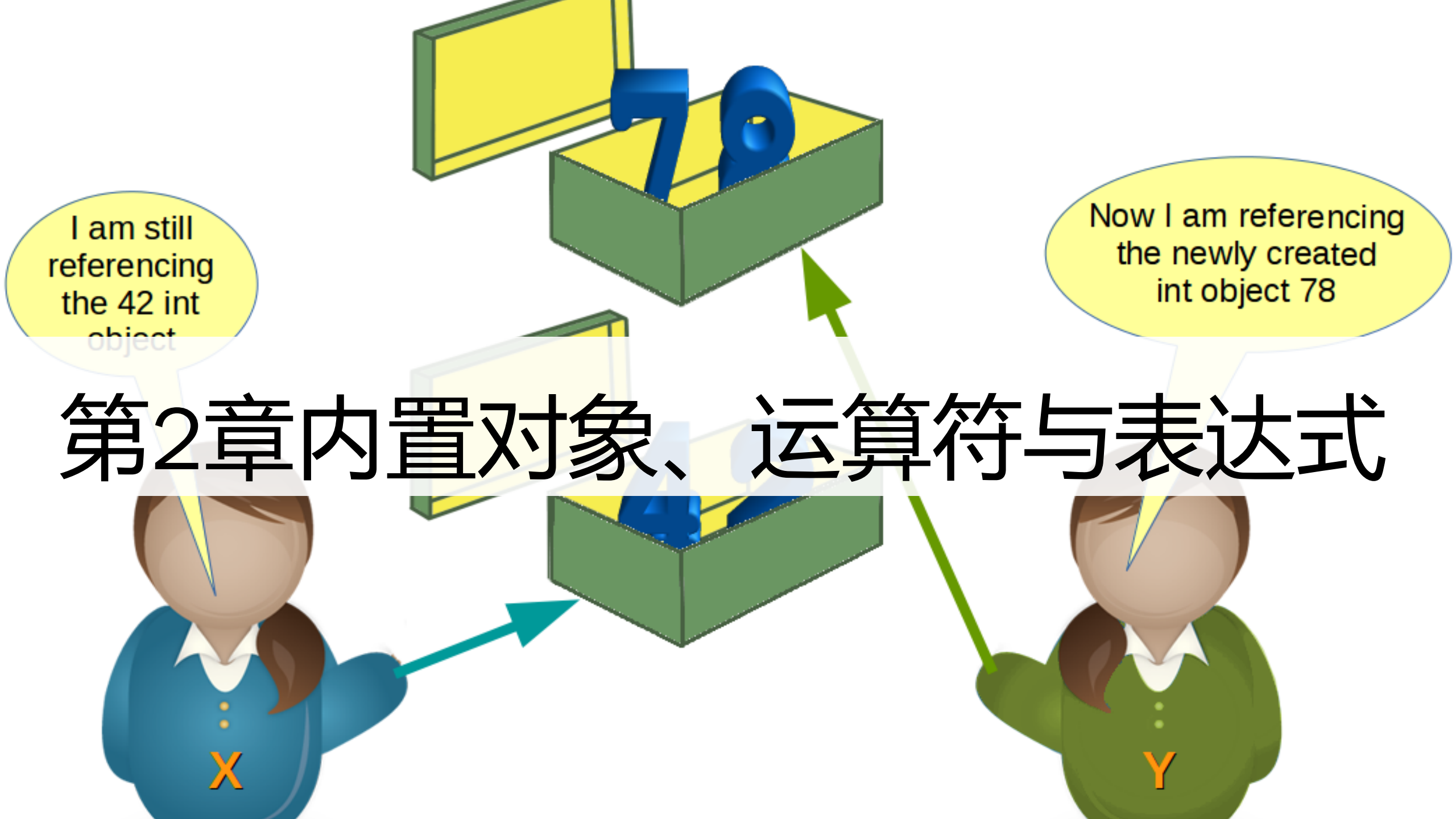


第2章内置对象、运算符与表达式

第2章 万丈高楼平地起: 运算符、表达式与内置对象

- 2.1 Python常用内置对象
  - 2.1.1 常量与变量
  - 2.1.2 数字
  - 2.1.3 字符串与字节串
  - 2.1.4 列表、元组、字典、集合
  - 其他可迭代对象
  - 布尔型

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完美的支持面向对象编程。一个抽象的例子：

➢ 抽象出一个类(class)：矩阵(matrix)

$$a\_matrix = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

➢ a_matrix是matrix的一个对象，它由数据和行为构成

➢ a_matrix的数据：矩阵元素的值、矩阵的行数和列数、....，使用a_matrix.shape等访问这些数据

➢ a_matrix的行为：矩阵的转置、删除矩阵的一行、....，使用a_matrix.transpose()等执行这些行为

2.1 Python常用内置对象

- 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、列表等。
- 非内置对象需要导入模块才能使用，如numpy中的array对象。

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	数字大小没有限制，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引导的表示原始字符串
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** frozenset	{'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，**元素不允许重复**;另外，set是可变的，而frozenset是不可变的

2.1 Python常用内置对象

续表

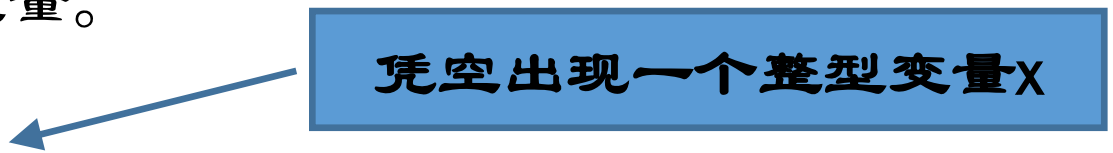
对象类型	类型名称	示例	简要说明
布尔型	**bool**	True, False	逻辑值，关系运算符、成员测试运算符、同一性测试运算符组成的表达式的值一般为True或False
空类型	NoneType	None	空值
异常	Exception ValueError TypeError		Python内置大量异常类，分别对应不同类型的异常
文件		f = open('data.dat', 'rb')	open是Python内置函数，使用指定的模式打开文件，返回文件对象
其他可迭代对象		生成器对象、range对象、zip对象、enumerate对象、map对象、filter对象等等	具有**惰性求值**的特点，除range对象之外，其他对象中的元素只能看一次
编程单元		函数（使用def定义） 类（使用class定义） 模块（类型为module）	类和函数都属于**可调用对象**，模块用来集中存放函数、类、常量或其他对象

第2章 万丈高楼平地起: 运算符、表达式与内置对象

- 2.1 Python常用内置对象
  - **2.1.1 常量与变量**
  - 2.1.2 数字
  - 2.1.3 字符串与字节串
  - 2.1.4 列表、元组、字典、集合
  - 其他可迭代对象
  - 布尔型

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。




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

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

- Python中对象自带类型，变量x只是对象的一个标签，找到这个标签就可以使用数据。

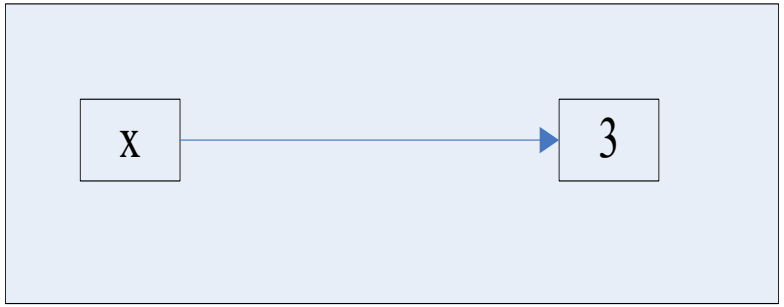
C语言中变量带类型，是容器。

python: x = 3 #给3这个值贴一个标签x
C: x = 3 #给x这个盒子里装入3

2.1.1 常量与变量

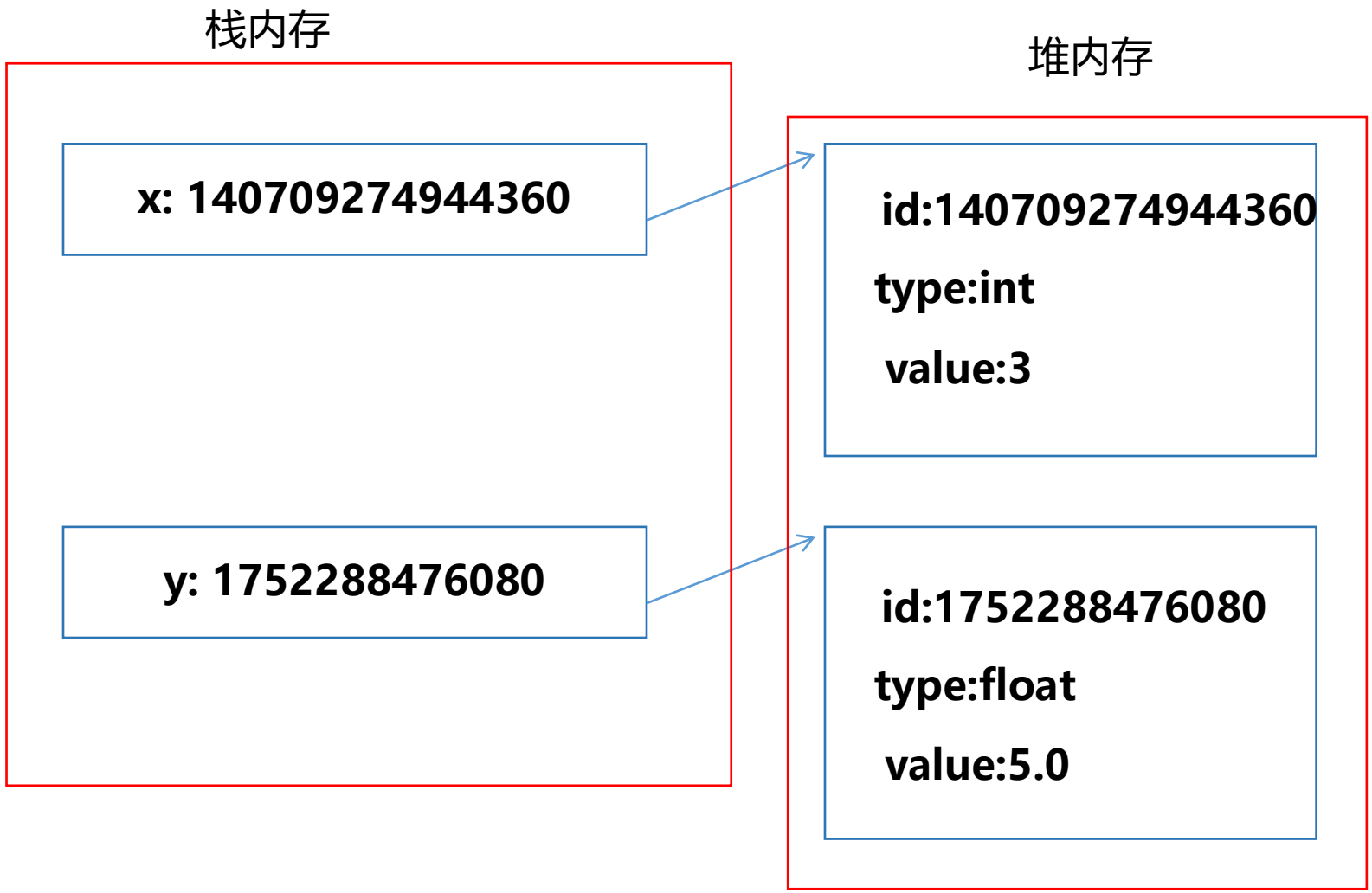
- 赋值语句的执行过程是：首先把等号右侧表达式的值计算出来，然后在内存中寻找一个位置把值存放进去，最后创建变量并**指向**这个内存地址。

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



- Python采用的是**基于值的内存管理方式**：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，因此变量类型随时可以改变（**动态类型**）。

2.1.1 常量与变量


栈内存
堆内存
x: 140709274944360
y: 1752288476080
id:140709274944360
type:int
value:3
id:1752288476080
type:float
value:5.0

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：

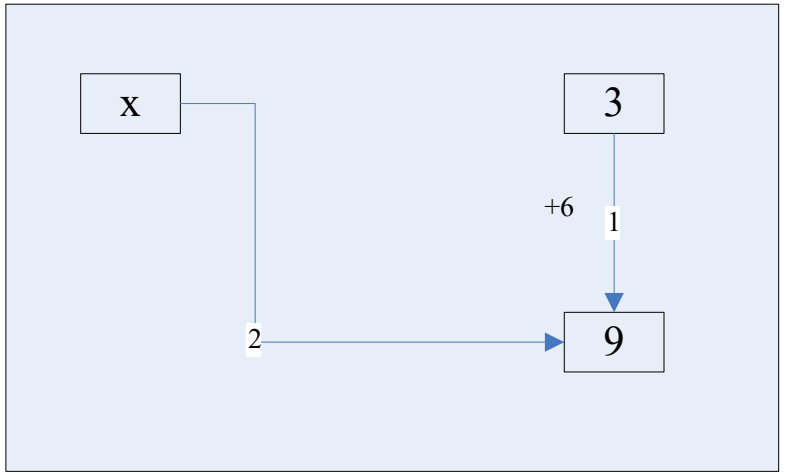
```
>>> x = 3
>>> id(x)
1786684560
>>> y = x
>>> id(y)
1786684560
```

id()函数用来查看对象的内存地址

```
id(obj, /)
    Return the identity of an object.
```

- 接着上面的代码再继续执行下面的代码：

```
>>> x = x+6
>>> id(x)
1786684752
```



2.1.1 常量与变量

❖Python属于**强类型编程语言**，Python解释器会根据所赋的值来自动**推断**变量类型，可以使用内置函数type()返回变量类型。

```
>>> x = 3
>>>type(x)
<class 'int'>
>>> x = 'Hello world.'
>>> type(x)
<class 'str'>
>>> type(type(x))
<class 'type'>
```

#查看变量类型

```
class type(object)
|  type(object) -> the object's type
```

```
>>> isinstance(x, int)
False
>>> isinstance(x, str)
True
```

#测试对象是否是某个类型的实例

```
isinstance(obj, class_or_tuple, /)
    Return whether an object is an instance of a class or of a
subclass thereof.
```

查内置函数: dir(__builtins__)

https://docs.python.org/zh-cn/3/library/functions.html

2.1.1 常量与变量

- Python语言规定，变量名由字母、数字和下划线组成，且不允许以数字开头。在使用变量名时应注意以下几点：

✓ 变量名**必须**以字母或下划线开头，但以下划线开头的变量在Python中有特殊含义；

✓ 变量名中**不能**有空格以及标点符号；

✓ **不能**使用关键字作变量名；

✓ 变量名对英文字母的**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作变量名，**这将会改变其类型和含义**；

```
>>> id(3)
1667343520
>>> id = '3724....'
>>> id(3)
TypeError: 'str' object is not callable
>>> id = __builtins__.id
#可恢复id的内置函数属性
>>> id(3)
```

合法变量名　　student_1、addNumber、_num

不合法变量名　　3number、3 num、2_student、class、math

2.1.1 常量与变量

常见的变量名命名方式有以下两种。

- **下划线命名法。**用下划线分割小写字母段或者大写字母段，例如my_name、my_age、GlOBAL_NAME等。
- **驼峰式命名法，包括小驼峰法和大驼峰法。**其中小驼峰法是指首字母小写，其他单词的首字母大写，例如myName、myAge、myStudentCount等。大驼峰法又称帕斯卡命名法，是指首字母大写的多个单词，例如MyName、MyAge、MyStudentCount等。

第2章 万丈高楼平地起: 运算符、表达式与内置对象

- 2.1 Python常用内置对象
  - 2.1.1 常量与变量
  - **2.1.2 数字**
  - 2.1.3 字符串与字节串
  - 2.1.4 列表、元组、字典、集合
  - 其他可迭代对象
  - 布尔型

2.1.2 数字

- Python支持任意大的整数
- 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。
- 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

2.1.2 数字

```
>>> 9999 ** 99                    #这里**是幂乘运算符


>>> 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
```

```
>>> type(99)
<class 'int'>

>>> type(0.4)
<class 'float'>
```

2.1.2 数字

```
#浮点数的精度：约16位有效数字

>>> x = 1.0
>>> y = 1.0 + 2.0e-17
>>> y - x > 1.0e-17
False

>>> x = 1.0
>>> y = 1.0 + 2.0e-16
>>> y - x > 1.0e-16
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
(-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)
```

```
>>> type(4j)
<class 'complex'>
>>> type(4J)
<class 'complex'>

imag和real为对象x的属性(attribute)
conjugate()为对象x的方法(method)
```

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

>>> x ** 3                      #复数幂运算
(-117+44j)

>>> x ** (1/2)
(2+1j)

>>> import math
>>> math.log(x)    #不支持
TypeError: must be real number, not complex

>>> import numpy as np
>>> np.log(x)
np.complex128(1.6094379124341003+0.9272952180016122j)
```

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.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 ** 2                 #支持幂运算
Fraction(9, 25)

>>> str(x)                 #类型转换
'3/5'
>>> float(x)               #类型转换
0.6
```

```
>>> import fractions
>>> dir(fractions)
```

2.1.2 数字

- 标准库fractions中提供的Decimal类实现了更高精度实数的运算。

```
>>> from fractions import Decimal
>>> 1 / 9                    #内置的实数类型
0.1111111111111111

>>> Decimal(1/9)             #高精度实数，但是不准确
Decimal('0.111111111111111104943205418749130330979824066162109375')

>>> Decimal(1)/Decimal(9)  #高精度，且准确
Decimal('0.111111111111111111111111111111')
```

2.1.2 数字

- 标准库random可产生各种随机数

```
>>> import random

>>> random.randrange(0, 10)
5   #[0, 10)之间的随机整数

>>> random.random()
0.9124818648228413  #[0, 1)之间的随机浮点数

>>> random.uniform(-1, 1)
-0.10770901677038269  #连续均匀分布

>>> sum(random.random() for i in range(10000))/10000
0.504904980528766    #[0,1)之间的随机数的平均值

>>> random.randint(0,10)
6   #[0, 10]之间的随机整数 #包括右端数
```

```
>>> import random
>>> dir(random)
```

https://docs.python.org/zh-cn/3/library/random.html

Practice

- Python“基于值的内存管理方式”的含义？

✓ 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；变量类型随时可以改变

- Python内置的三种数字类型（type）是什么？

✓ int, float, complex

- 为变量x赋值x = 3+4j，x.real和x.imag的类型(type)是什么？

✓ float

第2章 万丈高楼平地起: 运算符、表达式与内置对象

- 2.1 Python常用内置对象
  - 2.1.1 常量与变量
  - 2.1.2 数字
  - **2.1.3 字符串与字节串**
  - 2.1.4 列表、元组、字典、集合
  - 其他可迭代对象
  - 布尔型

2.1.3 字符串与字节串

- 在Python中，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。

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
'goodmorning'
```

2.1.3 字符串与字节串

```
>>> x = 'good morning'
>>> y = 'good'
>>> x - y    #不支持
TypeError: unsupported operand type(s) for -: 'str' and 'str'

>>> x * 2
'good morninggood morning'


>>>   '0'*100    #可以很容易得到100个0的字符串
'0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000'


>>> x ** 2
TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'int'
```

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**则得到str字符串。

```
>>> a_bytes = '工大'.encode('utf-8')
>>> a_bytes
b'\xe5\xb7\xa5\xe5\xa4\xa7'
#\x代表16进制
>>> type(a_bytes)
<class 'bytes'>
>>> a_bytes.decode('utf-8')
'工大‘
>>> '工大'.encode().decode()
'工大‘ #‘utf-8’是默认，可不写
>>> len('工大')
2
>>> len(a_bytes)
```

```
len(obj, /)
    Return the number of items in a container.
```

第2章 万丈高楼平地起: 运算符、表达式与内置对象

- 2.1 Python常用内置对象
  - 2.1.1 常量与变量
  - 2.1.2 数字
  - 2.1.3 字符串与字节串
  - **2.1.4 列表、元组、字典、集合**
  - 其他可迭代对象
  - 布尔型

2.1.4 列表、元组、字典、集合

列表：打了激素的数组

```
>>> x_list = [1, 2, ‘工大’]                #创建列表对象
```

元组：轻量级列表

```
>>> x_tuple = (1,0.2,'a')                  #创建元组对象
```

字典：反映对应关系的函数映射

```
>>> x_dict = {'a':97, 'b':98, 'c':99}     #创建字典对象
```

集合：元素之间不允许重复

```
>>> x_set = {1, 2, 3}                     #创建集合对象
```

2.1.4 列表、元组、字典、集合

	列表(list)	元组(tuple)	字典(dict)	集合(set)
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（不可改变，例：数字、字符串、元组）**	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
元素查找速度	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

■对象“不可变”(可哈希)的含义?

- python为每个对象分配一块内存空间，“不可变”意味着一旦生成该对象，为分配的内存空间填好数据，就不可以对这块内存空间做任何改变。
- 分配给元组的内存空间实际存储的是其他对象的地址；元组不可变只意味着不能再让元组中的元素指向新的对象，也不能再增加和减少元组中的元素，但可对元组中的对象做原地操作。

■不可变对象：int, float, complex, bool, **str**, bytes, **tuple**

■可变对象：list, dict, set

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, 0.2, 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
```

```
>>> a = (1,2,[1,2])
>>> a[2]
[1, 2]
>>> a[2][0] = 3      #元组不可变，但可对元组的元素做原地操作
>>> a
(1, 2, [3, 2])
```

第2章　万丈高楼平地起：运算符、表达式与内置对象

- 2.1 Python常用内置对象
  - 2.1.1 常量与变量
  - 2.1.2 数字
  - 2.1.3 字符串与字节串
  - 2.1.4 列表、元组、字典、集合
  - **其他可迭代对象**
  - **布尔型**
  - **日期与时间**

其他可迭代对象(iterator)

■可迭代对象支持成员测试运算：item in x或可以遍历的对象

- generator(生成器)对象
- range对象（内置函数range的返回值）
- zip对象（内置函数zip的返回值）
- enumerate对象（内置函数enumerate的返回值）
- reversed对象（内置函数reversed的返回值）
- （了解）map对象（内置函数map的返回值）
- （了解）filter对象（内置函数filter的返回值）

布尔型

```
>>> type(True)
<class 'bool'>

>>> bool(0)          #0、空的序列和None对应于False
False
>>> bool([])
False
>>> bool('')
False
>>> bool(None)
False

>>> bool(1)
True
>>> bool('a')
True
```

bool(x)	返回与x等价的布尔值 True或False

日期与时间

- 标准库time的time()返回当前时间，以秒为单位，时间零点为1970-1-1 8:00
- time()函数可用于测量程序的运行时间

```
import time   #dir(time)

# 获取当前时间戳（秒数，从1970年1月1日开始）
start_time = time.time()

# 让程序暂停3秒
time.sleep(3)

# 获取3秒后的时间戳
end_time = time.time()

print("开始时间:", start_time)
print("结束时间:", end_time)
print("实际耗时:", end_time - start_time, "秒")
```

```
开始时间: 1700000000.123456
结束时间: 1700000003.126789
实际耗时: 3.003333 秒
```

日期与时间

■ 标准库datetime提供了datetime类，以面向对象编程方式处理时间

```
>>> from datetime import datetime  #从内置datetime模块导入datetime类
                                    #import datetime;dir(datetime)

>>> t1 = datetime(year=2019, month=5, day=17) #使用命名参数创建datetime对象
>>> t2 = datetime(year=2019, month=5, day=21) #可用type(t1)测试datetime是类
>>> t1 < t2
True
```

■ 创建datetime对象的一般格式是

```
datetime(year, month, day, hour=0, minute=0, second=0,
microsecond=0,tzinfo=None)

>>> t = datetime.strptime('2019-5-17', '%Y-%m-%d')
>>> t
datetime.datetime(2019, 5, 17, 0, 0)
#将一个格式化的日期字符串，解析并转换成一个 datetime 对象
```

https://docs.python.org/zh-cn/3/library/datetime.html

日期与时间

```
>>> from datetime import datetime #从Python内置的datetime模块中导入datetime类
>>> t = datetime.now()   #调用datetime类的now()方法，创建了一个datetime对象
datetime.datetime(2025, 9, 20, 18, 1, 6, 858877)
>>> t.ctime()   #current time #ctime()方法将datetime对象转换为易读的字符串格式
'Sat Sep 20 18:03:16 2025'
>>> t.isoformat()  #datetime模块中的ISO 8601标准格式
'2025-09-20T18:08:36.464998'

>>> t1 = datetime(2010, 5, 17, hour=8)
>>> t2 = datetime(2020, 2, 8, hour=10)
>>> dt = t2 - t1
>>> dt
datetime.timedelta(3554, 7200)
>>> dt.days
3554
>>> dt.seconds
7200
```

Practice

- [1,2],{1,2},(1,2),{1:2}的类型(type)分别是什么?

✓ list,set,tuple,dict

- 元组和列表最大的不同点？集合和列表最大的不同点?

✓ 列表可变，元组不可变

✓ 列表中元素可重复，集合中的元素不允许重复

- a = (1,[2,1],[1,2])，a[2][1]的值?

✓ 2

习题

2.3 表达式 3**2 + 5 的值为_____。

2.30 多选题：下面属于可哈希对象的有（ ）。

A. 345 B. '0b10001' C.[1, 2, 3] D.{3}

2.32 单选题：下面不是合法变量名的有（ ）。

A. age B. name C. 3_name D. height

2.33 单选题：下面不是合法变量名的有（ ）。

A. width B. area C. for D. door_number

2.37 多选题：下面属于 Python 内置对象的有（ ）。

A. str B. list C. dict D. set

习题

2.39 多选题：下面属于合法变量名的有（ ）。

A. def B. while C. age D. name

2.40 多选题：下面属于合法变量名的有（ ）。

A. 3name B. name_3 C. 姓名 D. _name

2.42 多选题：下面属于合法数字的有（ ）。

A. 1_234_567 B. 1e8 C. 9.8 D. .3

2.58 判断对错：3+4j 不是合法的 Python 表达式。

2.63 判断对错：4j 是合法 Python 数字类型。

2.65 判断对错：在 Python 程序中可以使用 for 作为变量名。

2.66 判断对错：在 Python 程序中可以使用 while 作为变量名。

休息一下：丹尼斯·里奇(C语言之父)小传（自述）

我于1941年9月9日出生在纽约州布朗克斯维尔（Bronxville），后来在哈佛大学读了本科并进一步深造，我的本科专业是物理学，研究生阶段学的是应用数学。我的博士论文（1968年）是关于函数的子递归层次（subrecursive hierarchies）。本科阶段的学习让我明白，以自己的才智还不足以成为一名物理学者，而往计算机方向发展却相当不错。研究生阶段的经历又让我清醒，自己的才智也不足以让我成为算法理论方面的专家。我自己更喜欢过程式语言，而不是函数式语言。

我在1967年加入了贝尔实验室，算是步父亲的后尘，我的父亲Alistair E. Ritchie就在贝尔实验室工作了很长时间。我为BCPL语言写一个编译器。然后，我帮助Ken Thompson创建了Unix操作系统。

在Unix开发的早期，我对Thompson的B语言进行了改造，添加了数据类型和新语法，由此创造了C语言。C是Unix可移植性的技术基础，后来在许多其他环境下也被广泛采用。从手持设备到超级计算机，各类大小的电脑的许多应用开发和系统开发，都用到了C。这门语言后来出现了统一的美国标准和国际标准，并且Stroustrup在此基础上发明了后续的C++。

我获得过的奖励如下：1974年ACM的杰出论文奖（系统和语言方面）；1982年IEEE的Emmanuel Piore奖；1983年贝尔实验室会士荣誉称号；1983年ACM图灵奖；1983年ACM软件系统奖；1989年NEC的C&C基金奖；1990年IEEE的汉明奖。
我在1988年被选为美国工程院院士。1999年4月获得了美国国家技术奖。这些荣誉都是和Ken Thompson一起获得的。附注：像我这样靠Ken的裙带关系获得好处的人还真不少。但除了他的夫人Bonnie T.，我是少有的几个真的看见过他穿大衣（甚至打上黑领带）的人，还见过不止一次。

第2章　万丈高楼平地起：运算符、表达式与内置对象

- 2.2 Python运算符与表达式
  - 2.2.1 算术运算符
  - 2.2.2 关系运算符
  - 2.2.3 成员测试运算符in与同一性测试运算符is
  - 2.2.4 位运算符与集合运算符
  - 2.2.5 逻辑运算符
  - 2.2.6 矩阵乘法运算符@
  - 2.2.7 补充说明

2.2 Python运算符与表达式

- 运算符优先级遵循的规则为：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。

eg. c/2/a/b 写为 c/(2*a*b)更清晰

2.2 Python运算符与表达式

运算符	功能说明
+	算术加法，**列表、元组、字符串合并与连接**，正号
-	算术减法，**集合差集**，相反数
*	算术乘法，**序列重复**
/	真除法
//	**求整商**，但如果操作数中有实数的话，结果为实数形式的整数
%	求余数，字符串格式化
**	**幂运算**
<、<=、>、>=、==、!=	（值）大小比较，**集合的包含关系比较**
or	**逻辑或**
and	**逻辑与**
not	**逻辑非**
in	**成员测试**
is	**对象同一性测试，即测试是否为同一个对象(内存地址是否相同)**
|、^、&、<<、>>、~	位或、位异或、位与、左移位、右移位、位求反
&、|、^	**集合交集、并集、对称差集**
@	**矩阵相乘运算符**

第2章　万丈高楼平地起：运算符、表达式与内置对象

■ 2.2 Python运算符与表达式

- □ **2.2.1 算术运算符**
- □ 2.2.2 关系运算符
- □ 2.2.3 成员测试运算符in与同一性测试运算符is
- □ 2.2.4 位运算符与集合运算符
- □ 2.2.5 逻辑运算符
- □ 2.2.6 矩阵乘法运算符@
- □ 2.2.7 补充说明

2.2.1 算术运算符

运算符	功能说明
+	算术加法，**列表、元组、字符串合并与连接**，正号
-	算术减法，**集合差集**，相反数
*	算术乘法，**序列重复**
/	真除法
//	**求整商**，但如果操作数中有实数的话，结果为实数形式的整数
%	求余数，字符串格式化
**	**幂运算**

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。

```
>>> [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'
>>>{1,2}  + {3,4}                  #报错，应{1,2}|{3,4}
>>>{1:'a',2:'b'} + {3:'c',4:'d'}       #报错，python3.9后可{1:'a',2:'b'}|{3:'c',4:'d'}

>>> True + 3                       #Python内部把True当作1处理
4
>>> False + 3                      #把False当作0处理
3
```

演示：

```
>>> a = (4,)
>>> type(a)
>>> a = (4)
>>> type(a)
>>> a = [4,]
>>> type(a)
>>> a = [4]
>>> type(a)
>>> a = ()
>>> type(a)
```

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。

```
>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]

>>> 'abc' * 3
'abcabcabc‘
```

演示：

```
>>> [0] * 20
>>> '*' * 50
>>> '=' * 50
>>> '*=' * 50
```

■ 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。

■（了解）运算符*执行的是浅复制（只复制引用）

```
>>> b = [[1,2],3] * 2
>>> b
[[1, 2], 3, [1, 2], 3]
>>> b[0][0] = 0
>>> b
[[0, 2], 3, [0, 2], 3]
```

```
>>> [id(i) for i in b]
[2635893955776, 140711837336424,
2635893955776, 140711837336424]
```

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）。

```
>>> type(4/2)
<class 'float'>
```

```
>>> 3 / 2                    #数学意义上的除法
1.5

>>> 15 // 4                  #向下取整
3

>>> -15//4                   #(-15)//4
-4
```

2.2.1 算术运算符

（4）%运算符用于求余数，（了解）还可以用于字符串格式化，但是这种用法并不推荐。

```
>>> a,b = -7,3
>>> (a//b)*b + a%b == a
True
```

```
>>> 789 % 23                    #余数
7
```

```
>>> '%d' % (65)      #%d 表示将一个值格式化为十进制整数 # %字符串格式化操作符
'65'                  #(65)：这是要插入到格式字符串中的值
```

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
>>> 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
(1.8369701987210297e-16+3j)

>>> (1 + 2j) ** 0.5           #可以计算复数的幂次
(1.272019649514069+0.7861513777574233j)

>>> (1 + 2j) ** (3+4j)
(0.12900959407446697+0.03392409290517014j)
```

Practice

- [[1,2], 3]*2结果是什么？*运算是一种浅复制，浅复制的含义是什么?

✓ [[1,2], 3, [1,2], 3]。

✓ 第二个[1,2]与第一个[1,2]是同一个列表，具有相同的内存地址

习题

2.14 表达式 13 // 4 的值为_____。

2.15 表达式 -13 // 4 的值为_____。

2.16 表达式 -10 % 4 的值为_____。

2.17 表达式 10 % -3 的值为_____。

2.23 单选题：表达式 [1, 2, 3] * 2 的值为（　　）。

A. [1, 2, 3, 1, 2, 3]　　B. [2, 4, 6]

C. [1, 1, 2, 2, 3, 3]　　D. 无法计算

2.24 单选题：表达式 68 // -7 的值为（　　）。

A. -10　　B. 10　　C. 9　　D. -9

习题

2.34 单选题：已知 x = [1, 2] 和 y = [3, 4]，那么 x+y 的结果是（

A. 3　　B. 7　　C. [4, 6]　　D. [1, 2, 3, 4]

2.38 单选题：执行语句 d, _ = divmod(36, 12) 之后，变量 d 的值为（

A. 36　　B. 12　　C. 3　　D. 语法错误无法执

2.43 单选题：已知 x = [1, 2, 3]，那么 x*3 的值为（　　）。

A. 6　　B. 18

C. [3, 6, 9]　　D. [1, 2, 3, 1, 2, 3, 1, 2, 3]

2.56 判断对错：加法运算符可以连接两个字典得到新字典。

2.57 判断对错：加法运算符可以连接两个列表得到新列表。

第2章 万丈高楼平地起: 运算符、表达式与内置对象

- 2.2 Python运算符与表达式
  - 2.2.1 算术运算符
  - **2.2.2 关系运算符**
  - 2.2.3 成员测试运算符in与同一性测试运算符is
  - 2.2.4 位运算符与集合运算符
  - 2.2.5 逻辑运算符
  - 2.2.6 矩阵乘法运算符@
  - 2.2.7 补充说明

2.2.2 关系运算符

运算符	功能说明
<、<=、>、>=、==、 !=	（值）大小比较，集合的包含关系比较

2.2.2 关系运算符

- Python关系运算符最大的特点是可以连用。关系运算符要求操作数之间必须可比较大小。

```
>>> 1 < 3 < 5
True
>>> 3 < 5 > 2
True
>>> 1 > 6 < 8
False
```

■连用的关系运算符具有**惰性求值**或者逻辑短路的特点

```
>>> 1 > 6 < math.sqrt(9)   #左边False，整体就False
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
```

2.2.2 关系运算符

```
>>> 'Hello' > 'world'              #比较字符串大小 #ord('H'):72, ord('w'):119
False
>>> [1, 2, 3] < [1, 2, 4]          #比较列表大小
True
>>> 'Hello' > 3                    #字符串和数字不能比较
TypeError: unorderable types: str() > int()
```

■ 集合的包含关系

```
>>> {1, 2, 3} < {1, 2, 3, 4}       #测试是否子集
True
>>> {1, 2, 3} == {3, 2, 1}         #测试两个集合是否相等
True
>>> {1, 2, 4} > {1, 2, 3}          #集合之间的包含测试
False
```

第2章　万丈高楼平地起：运算符、表达式与内置对象

■ 2.2 Python运算符与表达式

- □ 2.2.1 算术运算符
- □ 2.2.2 关系运算符
- □ **2.2.3 成员测试运算符in与同一性测试运算符is**
- □ 2.2.4 位运算符与集合运算符
- □ 2.2.5 逻辑运算符
- □ 2.2.6 矩阵乘法运算符@
- □ 2.2.7 补充说明

2.2.3 成员测试运算符in与同一性测试运算符is

运算符	功能说明
in	**成员测试**
is	**对象同一性测试，即测试是否为同一个对象(内存地址是否相同)**

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

>>> 'abc' in 'abcdefg'            #子字符串测试
True

>>> for item in [1,2,3]:          #循环，成员遍历
        print(item)
```

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(具有相同的内存地址)，如果是则返回True，否则返回False。

```
>>> x = [1, 2, 3]
>>> y = [1, 2, 3]
>>> x is y      #x和y指向两个不同的列表
False
>>> x == y
True

>>> y = x
>>> x is y
True
```

is运算符在实际编程中最常见的用法：

```
if x is None:
    ...
if x is not None:
    ...
```

- 2.2 Python运算符与表达式

2.2.4 位运算符与集合运算符

运算符	描述
&（位与）	1&1=1、1&0=0&1=0&0=0
|（位或）	1|1=1|0=0|1=1、0|0=0
^（位异或）	1^1=0^0=0、1^0=0^1=1
<<（位左移）	左移位时右侧补0，每左移一位相当于乘以2
>>（位右移）	右移位时左侧补0，每右移一位相当于整除以2
~（位取反）	~1=0，~0=1
&（交集）	
|（并集）	
^（对称差集）	

2.2.4 位运算符与集合运算符

- 位运算符只能用于整数，其内部执行过程为：首先将整数转换为二进制数，然后右对齐，必要的时候左侧补0，按位进行运算，最后再把计算结果转换为十进制数字返回。

2.2.4 位运算符与集合运算符

1．按位与运算符“&”

1&1=1、1&0=0&1=0&0=0

十进制数	二进制数							
60=	0	0	1	1	1	1	0	0
13=	0	0	0	0	1	1	0	1
60&13=	0	0	0	0	1	1	0	0

其运算结果为60&13=12，二进制数为0000 1100。

二进制位	2^5	2^4	2^3	2^2	2^1	2^0
权重值	32	16	8	4	2	1
60的二进制	1	1	1	1	0	0

总和：32 *1+ 16*1 + 8*1 + 4*1 + 2*0 + 1*0 = 60

2.2.4 位运算符与集合运算符

2．按位或运算符“|”

1|1=1|0=0|1=1、0|0=0

十进制数	二进制数							
60=	0	0	1	1	1	1	0	0
13=	0	0	0	0	1	1	0	1
60|13=	0	0	1	1	1	1	0	1

其运算结果为60|13=61，二进制数为0011 1101。

2.2.4 位运算符与集合运算符

3．按位异或运算符“^”

1^1=0^0=0、1^0=0^1=1

十进制数	二进制数							
60=	0	0	1	1	1	1	0	0
13=	0	0	0	0	1	1	0	1
60^13=	0	0	1	1	0	0	0	1

其运算结果为60^13=49，二进制数为0011 0001。

2.2.4 位运算符与集合运算符

4．按位左移运算符“<<”

十进制数	二进制数							
60=	0	0	1	1	1	1	0	0
60<<2	1	1	1	1	0	0	0	0

其运算结果为60<<2=240，二进制数为1111 0000。

每左移一位相当于乘以2: 240=60*2*2

2.2.4 位运算符与集合运算符

5．按位右移运算符“>>”

十进制数	二进制数							
60=	0	0	1	1	1	1	0	0
60>>2	0	0	0	0	1	1	1	1

其运算结果为60>>2=15，二进制数为0000 1111。

每右移一位相当于除以2: 15=60/2/2

2.2.4 位运算符与集合运算符

6．按位求反运算符“~”

~1=0，~0=1

十进制数	二进制数							
60=	0	0	1	1	1	1	0	0
~60=	1	1	0	0	0	0	1	1
13=	0	0	0	0	1	1	0	1
~13=	1	1	1	1	0	0	1	0

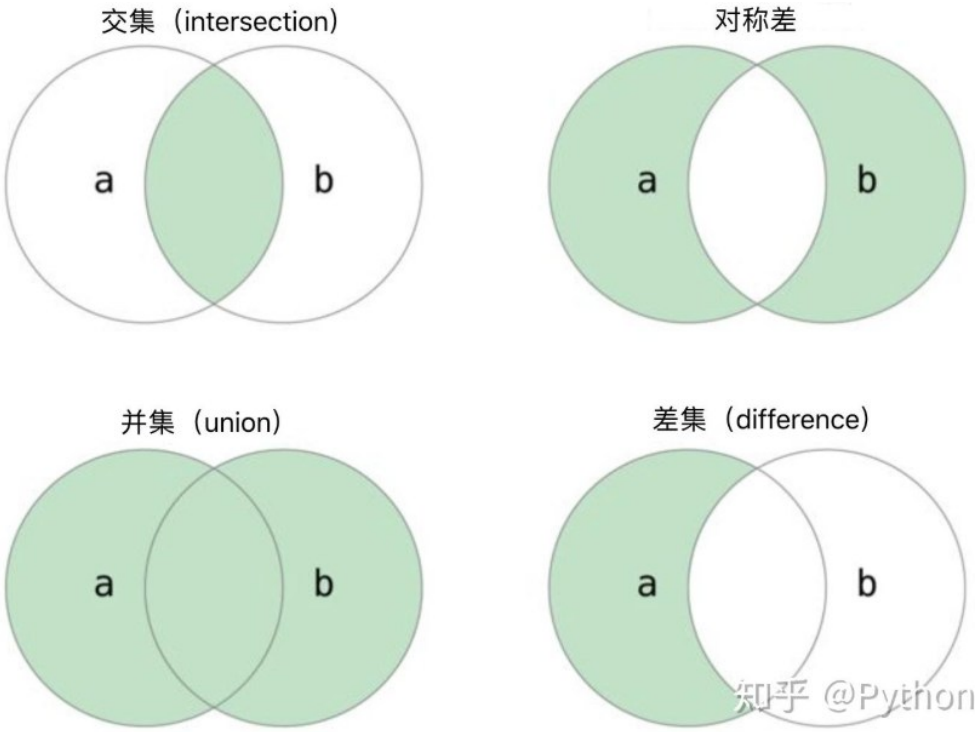
其运算结果为~60 = -61，~13 = -14。

总结：正整数n的位求反运算结果为~n = -n-1

2.2.4 位运算符与集合运算符

- 集合的交集、并集、对称差集等运算符与位运算符相同，差集使用减号运算符（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}          #对称差集=并集-交集
{1, 2, 4, 5}
>>> {1, 2, 3} - {3, 4, 5}          #差集
{1, 2}
>>> {3, 4, 5} - {1, 2, 3}          #差集
{4, 5}
```



习题

2.6 表达式 3 in [1, 2, 3, 4] 的值为_____。

2.7 表达式 3 not in [1, 2, 3, 4] 的值为_____。

2.8 Python 关键字中用来判断两个对象的引用是否相同的是_____。

2.25 单选题：表达式 {1, 2, 3, 4} - {2, 4, 5} 的值为（　　）。

A. {1, 3}　　B. {5}

C. {1, 3, 5}　　D. 错误表达式，无法计算

2.36 单选题：下面运算符中可以用于不同内置类型对象（整数、实数、复数不做区分）之间运算的有（　　）。

A. +　　B. -　　C. *　　D. /

第2章 万丈高楼平地起: 运算符、表达式与内置对象

- 2.2 Python运算符与表达式

  - 2.2.1 算术运算符
  - 2.2.2 关系运算符
  - 2.2.3 成员测试运算符in与同一性测试运算符is
  - 2.2.4 位运算符与集合运算符
  - **2.2.5 逻辑运算符**
  - 2.2.6 矩阵乘法运算符@
  - 2.2.7 补充说明

2.2.5 逻辑运算符

逻辑运算符

表达式1	表达式2	表达式1 and 表达式2	表达式1 or 表达式2	not 表达式1
True	True	True	True	False
True	False	False	True	False
False	False	False	False	True
False	True	False	True	True

运算符	说明	示例	结果
and	与	a and b	如果a的布尔值为True，返回b，否则返回a
or	或	a or b	如果a的布尔值为True，返回a，否则返回b
not	非	not a	a为False，返回True；a为True，返回False

2.2.5 逻辑运算符

■and和or具有惰性求值的特点，只计算必须要计算的值。

■运算符and和or并不一定会返回True或False，而是返回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。

运算符	说明	示例	结果
and	与	a and b	如果a的布尔值为True，返回b，否则返回a
or	或	a or b	如果a的布尔值为True，返回a，否则返回b
not	非	not a	a为False，返回True；a为True，返回False

- 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。

2.2.5 逻辑运算符

```
>>> 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

#惰性求值特点
>>> 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
```

2.2.5 逻辑运算符

```
>>> 3 not in [1, 2, 3]
False
>>> 3 is not 5                    #not的计算结果只能是True或False之一
True


if x is None:
    ...
if x is not None:
    ...
```

第2章 万丈高楼平地起: 运算符、表达式与内置对象

- 2.2 Python运算符与表达式
  - 2.2.1 算术运算符
  - 2.2.2 关系运算符
  - 2.2.3 成员测试运算符in与同一性测试运算符is
  - 2.2.4 位运算符与集合运算符
  - 2.2.5 逻辑运算符
  - **2.2.6 矩阵乘法运算符@**
  - **2.2.7 补充说明**

2.2.6 矩阵乘法运算符@

- 从Python 3.5开始增加了一个新的矩阵相乘运算符@，不过由于Python没有内置的矩阵类型，所以该运算符常与扩展库numpy一起使用。

```
>>> import numpy as np          #numpy是用于科学计算的Python扩展库
>>> a = np.ones((2,2))          #ones()函数用于生成全1矩阵，参数表示矩阵大小
>>> a
array([[1., 1.],
       [1., 1.]])
>>> a*a
array([[1., 1.],
       [1., 1.]])
>>> a@a
array([[2., 2.],
       [2., 2.]])
```

2.2.7 补充说明

- Python还有赋值运算符=和+=、-=、*=、/=等大量复合赋值运算符。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

Practice

- 连接(1, 2)和(3, 4)的表达式，合并{1, 2}和{3, 4}的表达式？

✓ (1, 2) + (3, 4)

✓ {1, 2} | {3, 4}

- 哪几种序列支持与整数的乘法？

✓ 列表、元组、字符串。字典和集合中的元素不允许重复

- 表达式的值？
- -10//3, {2, 4}<{2, 3, 4}, (9%3) or 2

✓ -4, True, 2

习题

2.13 已知 x = 3，那么执行语句 x += 6 之后，x 的值为_____。

2.18 表达式 3>5 and a<b 的值为_____。

2.26 单选题：表达式 3 and 5 的值为（ ）。

A. 3　　B. 5　　C. 8　　D. True

2.27 单选题：表达式 3 or 5 的值为（ ）。

A. 3　　B. 5　　C. 8　　D. True

2.45 多选题：下面表达式的值为 True 的有（ ）。

A. 5>3　　B. 3 and 5　　C. 5==3　　D. 3 not in [1,2,5]

2.46 多选题：下面表达式作为条件表达式时表示条件成立的有（ ）。

A. 5>3　　B. 3 and 5　　C. 5==3　　D. 3 not in [1,2,5]

第2章 万丈高楼平地起: 运算符、表达式与内置对象

- **2.3 Python关键字简要说明**
- 2.4 Python常用内置函数用法精要
  - 2.4.1 类型转换与类型判断
  - 2.4.7 range()
  - 2.4.2 最值与求和
  - 2.4.3 基本输入输出
  - 2.4.4 排序与逆序

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。

```
import keyword
print(keyword.kwlist)
```

2.3 Python关键字简要说明

关键字	含义
False	**常量，逻辑假**
None	**常量，空值**
True	**常量，逻辑真**
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	用在循环中，提前结束break所在层次的循环
class	用来定义类
continue	用在循环中，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，用来迭代序列或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**

2.3 Python关键字简要说明

关键字	含义
global	定义或声明全局变量
if	用在选择结构中
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返回值

第2章 万丈高楼平地起: 运算符、表达式与内置对象

- 2.4 Python常用内置函数用法精要
  - 2.4.1 类型转换与类型判断
  - 2.4.7 range()
  - 2.4.2 最值与求和
  - 2.4.3 基本输入输出
  - 2.4.4 排序与逆序

2.4 Python常用内置函数用法精要

- 内置函数（BIF，built-in functions）是Python内置对象类型之一，不需要额外导入任何模块即可直接使用，这些内置对象都封装在内置模块__builtins__之中，用C语言实现并且进行了大量优化，具有非常快的运行速度，推荐优先使用。

- 内置函数dir()可以查看所有内置函数和内置对象(directory)：

```
>>> dir(__builtins__)
```

- help(函数名)可以查看某个函数的用法。

```
>>> help(sum)
Help on built-in function sum in module builtins:

sum(iterable, start=0, /)
    Return the sum of a 'start' value (default: 0) plus an iterable of numbers

    When the iterable is empty, return the start value.
    This function is intended specifically for use with numeric values and may
    reject non-numeric types.
```

可迭代对象(iterable)

- 支持成员测试in运算(item in x)的对象x均为**可迭代对象(Iterable)**（即任何可用for循环遍历的对象）

- 例：list，tuple，dict，set，str等序列； range对象；reversed等迭代器
  - range对象（内置函数range的返回值）
  - reversed对象（内置函数reversed的返回值）

- **(了解)迭代器(iterator)**是遍历器，真正负责遍历的对象(书签)，只能访问一次。
- 例如：数列[1,2,3]是可迭代对象，但是iter([1,2,3])才是迭代器，可用next()调用其元素，一次只能调用一个元素。可用list()一次全部遍历，但只能一次。
- 除iter()外，map(),filtter(),zip(),enumerate()等内置函数也是迭代器
- 迭代器具有惰性求值特点，不事先存储所需要遍历的值，只在迭代至某个元素时才计算它的值

```
In [25]: I=iter([1, 2, 3])

In [26]: next(I)
Out[26]: 1

In [27]: next(I)
Out[27]: 2

In [28]: next(I)
Out[28]: 3
```

```
In [52]: I = iter([1,2,3])

In [53]: list(I)
Out[53]: [1, 2, 3]

In [54]: list(I)
Out[54]: []
```

2.4 Python常用内置函数用法精要

函数	功能简要说明
abs(x)	返回数字x的绝对值或复数x的模
all(iterable)	如果对于可迭代对象中所有元素x都等价于True，也就是对于所有元素x都有bool(x)等于True，则返回True。对于空的可迭代对象也返回True
any(iterable)	只要可迭代对象iterable中存在元素x使得bool(x)为True，则返回True。对于空的可迭代对象，返回False
ascii(obj)	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
bin(x)	把整数x转换为二进制串表示形式
bool(x)	返回与x等价的布尔值True或False
bytes(x)	生成字节串，或把指定对象x转换为字节串表示形式
callable(obj)	测试对象obj是否可调用。类和函数是可调用的，包含__call__()方法的类的对象也是可调用的
compile()	用于把Python代码编译成可被exec()或eval()函数执行的代码对象
complex(real, [imag])	返回复数
chr(x)	返回Unicode编码为x的字符

2.4 Python常用内置函数用法精要

续表1

函数	功能简要说明
delattr(obj, name)	删除属性，等价于del obj.name
dir(obj)	返回指定对象或模块obj的成员列表，如果不带参数则返回当前作用域内所有标识符
divmod(x, y)	返回包含整商和余数的元组((x-x%y)/y, x%y)
enumerate(iterable[, start])	返回包含元素形式为(0, iterable[0]), (1, iterable[1]), (2, iterable[2]), ...的迭代器对象
eval(s[, globals[, locals]])	**计算并返回字符串s中表达式的值**
exec(x)	执行代码或代码对象x
exit()	退出当前解释器环境
filter(func, seq)	返回filter对象，其中包含序列seq中使得单参数函数func返回值为True的那些元素，如果函数func为None则返回包含seq中等价于True的元素的filter对象
float(x)	把整数或字符串x转换为浮点数并返回
frozenset([x]))	创建不可变的集合对象
getattr(obj, name[, default])	获取对象中指定属性的值，等价于obj.name，如果不存在指定属性则返回default的值，如果要访问的属性不存在并且没有指定default则抛出异常

2.4 Python常用内置函数用法精要

续表2

函数	功能简要说明
globals()	返回包含当前作用域内全局变量及其值的字典
hasattr(obj, name)	测试对象obj是否具有名为name的成员
hash(x)	返回对象x的哈希值，如果x不可哈希则抛出异常
help(obj)	返回对象obj的帮助信息
hex(x)	把整数x转换为十六进制串
id(obj)	返回对象obj的标识（内存地址）
input([提示])	显示提示，接收键盘输入的内容，返回字符串
int(x[, d])	返回实数（float）、分数（Fraction）或高精度实数（Decimal）x的整数部分，或把d进制的字符串x转换为十进制并返回，d默认为十进制
isinstance(obj, class-or-type-or-tuple)	测试对象obj是否属于指定类型（如果有多个类型的话需要放到元组中）的实例
iter(...)	返回指定对象的可迭代对象
len(obj)	返回对象obj包含的元素个数，适用于列表、元组、集合、字典、字符串以及range对象和其他可迭代对象

2.4 Python常用内置函数用法精要

续表3

函数	功能简要说明
list([x])、set([x])、tuple([x])、dict([x])	把对象x转换为列表、集合、元组或字典并返回，或生成空列表、空集合、空元组、空字典
locals()	返回包含当前作用域内局部变量及其值的字典
map(func, *iterables)	返回包含若干函数值的map对象，函数func的参数分别来自于iterables指定的每个迭代对象，
max(x)、 min(x)	返回可迭代对象x中的最大值、最小值，要求x中的所有元素之间可比较大小，允许指定排序规则和x为空时返回的默认值
next(iterator[, default])	返回可迭代对象x中的下一个元素，允许指定迭代结束之后继续迭代时返回的默认值
oct(x)	把整数x转换为八进制串
open(name[, mode])	以指定模式mode打开文件name并返回文件对象
ord(x)	返回1个字符x的Unicode编码
pow(x, y, z=None)	返回x的y次方，等价于x ** y或(x ** y) % z

2.4 Python常用内置函数用法精要

续表4

函数	功能简要说明
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)	基本输出函数
quit()	退出当前解释器环境
range([start,] end [, step])	返回range对象，其中包含左闭右开区间[start,end)内以step为步长的整数
reduce(func, sequence[, initial])**	将双参数的函数func以迭代的方式从左到右依次应用至序列seq中每个元素，最终返回单个值作为结果。在Python 2.x中该函数为内置函数，在Python 3.x中需要从functools中导入reduce函数再使用
repr(obj)	返回对象obj的规范化字符串表示形式，对于大多数对象有eval(repr(obj))==obj
reversed(seq)	返回seq（可以是列表、元组、字符串、range以及其他可迭代对象）中所有元素逆序后的迭代器对象

2.4 Python常用内置函数用法精要

续表5

函数	功能简要说明
round(x [, 小数位数])	对x进行四舍五入，若不指定小数位数，则返回整数
sorted(iterable, key=None, reverse=False)	返回排序后的列表，其中iterable表示要排序的序列或迭代对象，key用来指定排序规则或依据，reverse用来指定升序或降序。该函数不改变iterable内任何元素的顺序
str(obj)	把对象obj直接转换为字符串
sum(x, start=0)	返回序列x中所有元素之和，返回start+sum(x)
type(obj)	返回对象obj的类型
zip(seq1 [, seq2 [...]])	返回zip对象，其中元素为(seq1[i], seq2[i], ...)形式的元组，最终结果中包含的元素个数取决于所有参数序列或可迭代对象中最短的那个

- 2.4 Python常用内置函数用法精要

2.4.1 类型转换与类型判断

- （了解）内置函数bin()、oct()、hex()用来将整数转换为二进制、八进制和十六进制形式，这三个函数都要求参数必须为整数。

```
>>> bin(555)                    #把数字转换为二进制串
'0b1000101011'
>>> oct(555)                    #转换为八进制串
'0o1053'
>>> hex(555)                    #转换为十六进制串
'0x22b'
```

函数	缩写来源	转换进制	前缀	示例输入	示例输出
bin()	binary	二进制	0b	bin(10)	'0b1010'
oct()	octal	八进制	0o	oct(8)	'0o10'
hex()	hexadecimal	十六进制	0x	hex(16)	'0x10'

2.4.1 类型转换与类型判断

- 内置函数int()用来将其他形式的数字转换为整数，参数可以为整数、实数、分数或合法的数字字符串。

```
>>> int(-3.2)                    #把实数转换为整数
-3
>>> from fractions import Fraction, Decimal
>>> x = Fraction(7, 3)
>>> x
Fraction(7, 3)
>>> int(x)                       #把分数转换为整数
2
>>> x = Decimal(10/3)    >>> x
Decimal('3.333333333333333481363069950020872056484222412109375')
>>> int(x)                       #把高精度实数转换为整数
3

>>> 4.4-1.4
3.0000000000000004
>>>int(4.4-1.4)
3

>>>int('123')      #使用较多
123
```

2.4.1 类型转换与类型判断

- 当参数为数字字符串时，还允许指定第二个参数base用来说明数字字符串的进制，base的取值应为0或2-36之间的整数，其中0表示按数字字符串隐含的进制进行转换。

```
>>> int('0x22b', 16)                #把十六进制数转换为十进制数
555
>>> int('22b', 16)                  #与上一行代码等价
555
>>> int(bin(555), 2)                #二进制与十进制之间的转换
555
>>> int('0b111')                    #非十进制字符串进，必须指定第二个参数
ValueError: invalid literal for int() with base 10: '0b111'
>>> int('0b111', 0)                 #第二个参数0表示使用字符串隐含的进制
7
>>> int('0b111', 6)                 #第二个参数应与隐含的进制一致
ValueError: invalid literal for int() with base 6: '0b111'
>>> int('0b111', 2)
7
>>> int('111', 6)                   #字符串没有隐含进制
                                    #第二个参数可以为2-36之间的数字
43
```

2.4.1 类型转换与类型判断

```
>>> int(2.7)   #int直接舍弃小数部分
2
>>> int(-2.7)
-2
>>> round(-2.7)  #round为内置函数，做四舍五入
-3


>>> import math
>>> math.floor(3.2)   #天花板数
3
>>> math.ceil(3.2)    #地板数
4
```

2.4.1 类型转换与类型判断

- 内置函数float()用来将其他类型数据转换为实数，complex()可以用来生成复数。

```
>>> float(3)                    #把整数转换为实数
3.0
>>> float('3.5')                #把数字字符串转换为实数
3.5
>>> float('inf')                #无穷大，其中inf不区分大小写
inf                             #eg. float(1e1000)
>>> complex(3)                  #指定实部
(3+0j)
>>> complex(3, 5)               #指定实部和虚部
(3+5j)
>>> complex('inf')              #无穷大
(inf+0j)
```

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(万国码)，而chr()则用来返回Unicode编码对应的字符

```
>>> ord('a')          #查看指定字符的Unicode编码
97
>>> chr(65)           #返回数字65对应的字符
'A'
>>> chr(ord('A')+1)    #Python不允许字符串和数字之间的加法操作
'B'
>>> chr(ord('国')+1)   #支持中文
'图'
```

ord -> order
chr -> character

Unicode（中文：统一码、万国码） 是一种国际标准字符编码系统，旨在为世界上所有文字系统的每个字符提供一个唯一的数字标识。Unicode 本身只是字符到码点的映射，实际存储和传输时需要具体的编码方案，UTF-8, UTF-16, UTF-32。
UTF-8,（最常用）：
变长编码：使用1-4个字节表示一个字符
兼容ASCII：ASCII字符在UTF-8中保持不变
存储效率高：英文字符只需1字节，中文通常3字节

2.4.1 类型转换与类型判断

■str()直接将其任意类型参数转换为字符串。

```
>>> 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

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

演示：
>>> list(‘abcd’)

```
>>> list((0,1,2,3,4))              #把元组转换为列表
[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([(1,'a'), (2, 'b'), (3, 'c'), (4, 'd')])   #创建字典
{1: 'a', 2: 'b', 3: 'c', 4: 'd'}

>>> set('1112234')                 #创建集合，自动去除重复
{'4', '2', '3', '1'}
```

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
```

Practice

- 设x是一个浮点数，使用哪三个函数分别得到与其最接近的整数，大于x的最小整数，小于x的最大整数?

✓ round(x), math.ceil(x), math.floor(x)

- list(‘[1,2,3]’)的返回值是?

✓ ['[', '1', ',', '2', ',', '3', ']']

- 如何将 ‘[1,2,3]’ 转换为list?

✓ eval (‘[1,2,3]’)