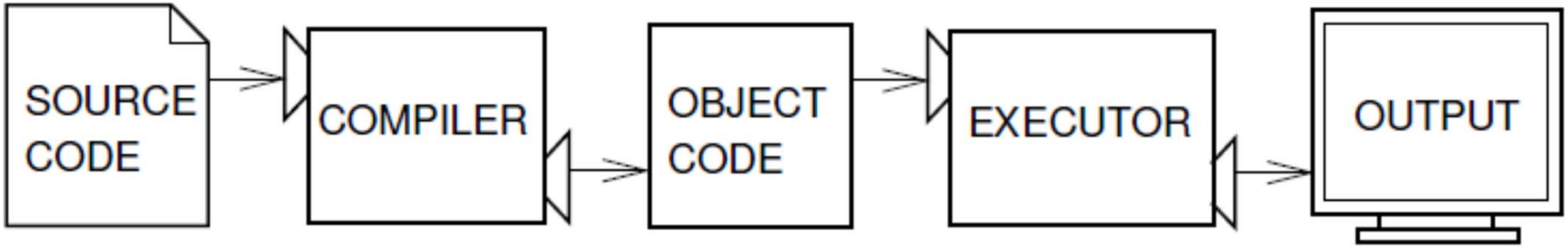
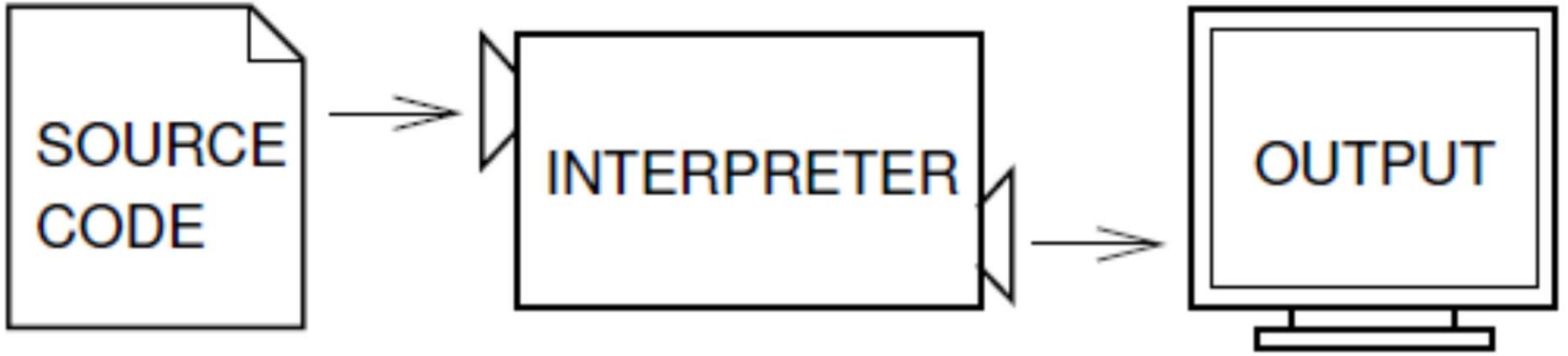
第1章　基础知识

1.0 Python是一种怎样的语言？(1/2)

- 跨平台、开源、免费的解释型高级动态编程语言





1.4 Python基础知识

- 对象模型
- 变量
- 数字
- 字符串
- 运算符与表达式
- 内置函数
- 对象删除
- 输入输出
- 模块

1.4.1 Python的对象模型(1/2)

- 对象：python中处理的每样“东西”都是对象
- 内置对象：可直接使用

  如 数字、字符串、列表、del等；
- 非内置对象：需要导入模块才能使用

  如 sin(x)，random( )等。

1.4.1 Python的对象模型(2/2)

- 常用内置对象

对象类型	示例
数字	1234，3.14，3+4j
字符串	'swfu', "I'm student", '''Python '''
日期	**2012-08-25**
列表	**[1, 2, 3]**
字典	**{1:'food',2:'taste',3:'import'}**
元组	**(2, -5, 6)**
文件	f=open('data.dat', 'r')
集合	**set('abc'), {'a', 'b', 'c'}**
布尔型	True, False
空类型	None
编程单元类型	函数、模块、类

1.4.2 Python变量(1/12)

- 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
```

创建了整型变量x，并赋值为3

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

创建了字符串变量x，并赋值为'Hello world.'

1.4.2 Python变量(2/12)

Python属于强类型编程语言

解释器会根据赋值或运算来自动推断变量类型

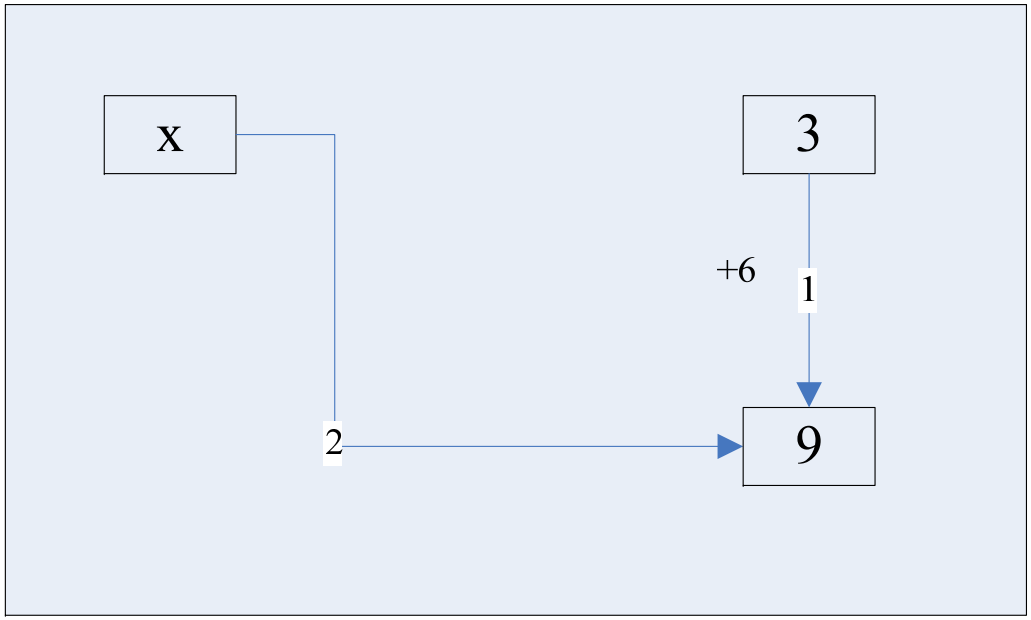
- 不同类型支持的运算也不完全一样
- 使用变量时需要程序员自己确定所进行的运算是否合适
- 同一个运算符对不同类型数据操作的含义和结果也不一样

1.4.2 Python变量(9/12)

- Python采用基于值的内存管理方式

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


```
>>> x += 6
```



1.4.2 Python变量(10/12)

- Python如果为不同变量赋值为相同值，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)
10417624
>>> y = 3
>>> id(y)
10417624
>>> y = 5
>>> id(y)
10417600
>>> id(x)
10417624
```

1.4.2 Python变量(11/12)

- Python自动内存管理功能，自动删除没有任何变量指向的值

  Python跟踪所有的值，自动删除不再有变量指向的值

- Python程序员一般情况下不需要太多考虑内存管理的问题

  显式使用del命令删除不需要的值

  显式关闭不再需要访问的资源

优秀程序员的基本素养

1.4.3 数字(1/4)

- 数字：python中最常用的不可变对象
- 可以表示任意大小的数字

```
>>> a=99999999999999999999999999999
>>> a*a
999999999999999999999999999998000000000000000000000000001L
>>> a**3
999999999999999999999999999997000000000000000000000000029999999999999999999999999999999L
```

- Python的IDEL交互界面可以当做简便计算器来使用

1.4.4 字符串(1/3)

- 用单引号、双引号或三引号括起来的符号系列称为字符串
- 单引号、双引号、三单引号、三双引号可以互相嵌套，用来表示复杂字符串。

  ’abc’、’123’、’中国’、″Python″
- 字符串属于不可变序列
- 空串表示为’’或 ″″
- 三引号’’’或″″″表示的字符串可以换行
- 三引号’’’或″″″可以在程序中表示较长的注释

1.4.5 操作符和表达式(1/6)

运算符示例	功能说明
x+y	算术加法，列表、元组、字符串合并
x-y	算术减法，集合差集
x*y	乘法，序列重复
x/y	除法（在Python 3.x中叫做真除法）
x//y	求整商
-x	相反数
x%y	余数（对实数也可以进行余数运算），字符串格式化
x**y	幂运算
x<y; x<=y; x>y; x>=y	大小比较（可以连用），集合的包含关系比较
x==y; x!=y	相等（值）比较，不等（值）比较
x or y	逻辑或（只有x为假才会计算y）
x and y	逻辑与(只有x为真才会计算y)
not x	逻辑非
x in y; x not in y	成员测试运算符
x is y; x is not y	对象实体同一性测试（地址）
|、^、&、<<、>>、~	位运算符
&、|、^	集合交集、并集、对称差集

1.4.6 常用内置函数(2 /5)

all(iterable)	如果对于可迭代对象中所有元素x都有bool(x)为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
bin(x)	把数字x转换为二进制串
callable(object)	测试对象是否可调用。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
chr(x)	返回ASCII编码为x的字符
cmp(x, y)	比较大小，如果x<y则返回负数，如果x==y则返回0，如果x>y则返回正数。Python 3.x不再支持该函数
eval	计算字符串中表达式的值并返回
filter	返回序列中使得函数值为True的那些元素，如果函数为None则返回那些值等价于True的元素。如果序列为元组或字符串则返回相同类型结果，其他则返回列表

1.4.6 常用内置函数(3/5)

hex(x)	把数字x转换为十六进制串
id(obj)	返回对象obj的标识（地址）
input([提示内容字符串])	接收键盘输入的内容，返回字符串。
int(x[, d])	返回数字的整数部分，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(object, class-or-type-or-tuple)	测试对象是否属于指定类型的实例
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、字符串等类型的对象
list([x])、 set([x])、tuple([x])、dict([x])	把对象转换为列表、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
map(函数,序列)	将单参数函数映射至序列中每个元素，返回结果列表

1.4.6 常用内置函数(4 /5)

open(name[, mode[, buffering]])	以指定模式打开文件并返回文件对象
ord(s)	返回1个字符s的编码
range([start,] end [, step])	返回一个等差数列（Python 3.x中返回一个range对象），不包括终值
reduce(函数,序列)	将接收2个参数的函数以累积的方式从左到右依次应用至序列中每个元素，最终返回单个值作为结果
reversed(列表或元组)	返回逆序后的迭代器对象
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
str(obj)	把对象obj转换为字符串
sorted(列表[, cmp[, key[reverse]]]	返回排序后的列表。Python 3.x中的sorted()方法没有cmp参数
zip(seq1 [, seq2 [...]])	返回[(seq1[0], seq2[0] ...), (...)]形式的列表

1.4.7　对象的删除(1/3)

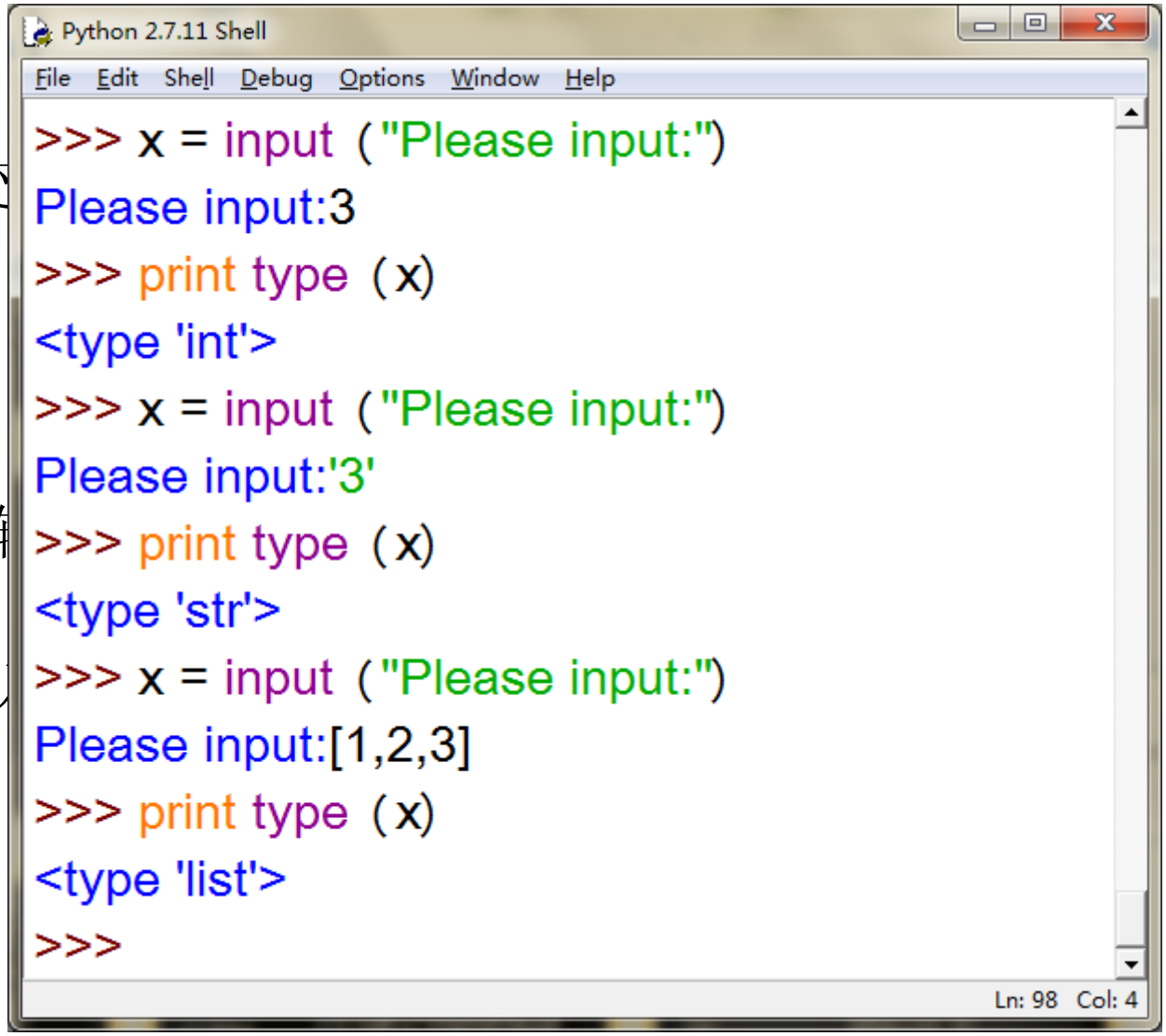
- Python自动内存管理功能:
  - 解释器跟踪所有值，没有变量指向的自动删除
  - 自动内存管理不保证及时释放内存
  - 显式释放申请的资源是程序员的好习惯和素养
- del命令
  - 显式删除对象
  - 解除与值之间的指向关系

1.4.8 基本输入输出(1/2)

- 输入通过

  - 返回

  - 可输入



1.4.9 模块的使用(1/4)

- Python默认安装仅包含部分基本或核心模块，用户可以用pip安装大量的扩展
- Python启动时仅加载了很少的一部分模块，需要时由程序员显式地加载（可能需要先安装）其他模块
- 减小运行的压力，仅加载真正需要的模块和功能，且具有很强的可扩展性。

1.8 编写自己的包与模块

- 包可以看做处于同一目录中的模块
- 包的每个目录必须包含一个__init__.py文件
  - 可以是一个空文件，表示该目录是一个包
  - 主要用途：设置__all__变量，包初始化代码

第2章 Python序列

2.0 Python序列的特点

- 序列: 一系列相关的按一定顺序排列的值
- 序列是程序设计中经常用到的数据存储方式
- Python提供的序列类型最丰富，最灵活，功能最强大
- Python序列结构：列表、元组、字典、字符串、集合、range
- 列表、元组、字符串等序列均支持双向索引
  - 第一个元素下标为0，第二个元素下标为1，以此类推；
  - 最后一个元素下标为-1，倒数第二个元素下标为-2，以此类推

2.1 列表(1)

- 列表:
  - 内置可变序列
  - 若干元素的有序集合
  - 列表中的每一个数据称为元素
  - 列表的所有元素放在一对中括号“[”和“]”中，并使用逗号分隔开
- 列表元素增加或删除:
  - 对象自动进行扩展或收缩内存，保证元素之间没有缝隙
- 列表中的数据类型可以各不相同:
  - 可以同时分别为整数、实数、字符串等基本类型
  - 可以是列表、元素、字典、集合以及其他自定义类型的对象

```
[10, 20, 30, 40] ['spam', 2.0, 5, [10, 20]]
['crunchy frog', 'ram bladder', 'lark vomit']
[['file1', 200,7], ['file2', 260,9]]
```

2.1 列表(2)

方法	说明
list.append(x)	将元素x添加至列表尾部
list.extend(L)	将列表L中所有元素添加至列表尾部
list.insert(index, x)	在列表指定位置index处添加元素x
list.remove(x)	在列表中删除首次出现的指定元素
list.pop([index])	删除并返回列表对象指定位置的元素
list.index(x)	返回值为x的首个元素的下标
list.count(x)	返回指定元素x在列表中的出现次数
list.reverse()	对列表元素进行原地逆序
list.sort()	对列表元素进行原地排序

2.1.6 切片操作(1)

- 切片

Python序列的重要操作之一，适用于列表、元组、字符串、range对象等类型。

- 可以使用切片来截取列表中的任何部分
- 可以通过切片来修改和删除列表中部分元素
- 可以通过切片操作为增加元素

2.1.9 列表推导式(1)

- 列表推导式
  - Python程序开发时应用最多的技术之一
  - 使用列表推导式快速生成多个随机数的列表
  - 用非常简洁的方式快速生成满足特定需求的列表，代码具有非常强的可读性

2.1.9 列表推导式(2)

- 实现嵌套列表的平铺
- 列出当前文件夹下所有Python源文件
- 过滤不符合条件的元素

2.2 元组

- 元组和列表类似
  - 属于不可变序列
  - 元组一旦创建，用任何方法都不可以修改其元素
  - 元组的定义方式和列表相同
  - 所有元素是放在一对圆括号“（”和“）”中

2.2.2 元组与列表的区别

- 元组中的数据一旦定义就不允许更改。
- 元组没有append()、extend()和insert()等方法，无法向元组中添加元素；
- 元组没有remove()或pop()方法，也无法对元组元素进行del操作，不能从元组中删除元素。
- 内建的tuple( )函数接受一个列表参数，并返回一个包含同样元素的元组，而list( )函数接受一个元组参数并返回一个列表。从效果上看，tuple( )冻结列表，而list( )融化元组。

2.2.3 元组的优点

- 元组的速度比列表更快。如果定义了一系列常量值，而所需做的仅是对它进行遍历，那么一般使用元组而不用列表。
- 元组对不需要改变的数据进行“写保护”将使得代码更加安全。
- 元组可用作字典键（特别是包含字符串、数值和其它元组这样的不可变数据的元组）。列表永远不能当做字典键使用，因为列表不是不可变的。

2.2.4元组的序列解包

- 可以使用序列解包功能对多个变量同时赋值
- 序列解包对于列表和字典同样有效

2.2.4 生成器推导式

- 生成器推导式使用圆括号，列表推导式使用方括号
- 生成器推导式的结果是一个生成器对象，不是列表或元组
- 生成器对象的元素可根据需要转化为列表或元组
- 可以使用生成器对象的next()方法进行遍历
- 可以直接将其作为迭代器对象来使用。
- 不管用哪种方法访问其元素，当所有元素访问结束以后，如果需要重新访问其中的元素，必须重新创建该生成器对象。

2.3 字典

- 字典是键值对的无序可变集合
- 字典元素的键和值用冒号分隔，元素之间用逗号分隔，所有的元素放在一对大括号“｛”和“｝”中
- 字典中的每个元素包含两部分：键和值，向字典添加一个键的同时，必须为该键增添一个值。
- 字典中的键可以为任意不可变数据
- 字典中的键不允许重复
- globals()返回包含当前作用域内所有全局变量和值的字典
- locals()返回包含当前作用域内所有局部变量和值的字典

2.3.1 字典创建与删除

- 使用=将一个字典赋值给一个变量
- 使用dict利用已有数据创建字典：
- 使用dict根据给定的键、值创建字典
- 以给定内容为键，创建值为空的字典
- 使用del删除整个字典

2.3.2 字典元素的读取

- 以键作为下标可以读取字典元素，若键不存在则抛出异常
- 使用字典对象的get方法获取指定键对应的值，并且可以在键不存在的时候返回指定值。
- 使用字典对象的items方法可以返回字典的键、值对列表
- 使用字典对象的keys方法可以返回字典的键列表
- 使用字典对象的values方法可以返回字典的值列表

2.3.3 字典元素的添加与修改

- 当以指定键为下标为字典赋值时，若键存在，则可以修改该键的值；若不存在，则表示添加一个键、值对
- 使用字典对象的update方法将另一个字典的键、值对添加到当前字典对象

- 使用del删除字典中指定键的元素
- 使用字典对象的clear方法来删除字典中所有元素
- 使用字典对象的pop 方法删除并返回指定键的元素
- 使用字典对象的popitem方法删除并返回字典中的元素

2.4 集合

- 集合
  - 无序可变集合
  - 使用一对大括号界定
  - 元素不可重复

2.4.1 集合的创建与删除

- 直接将集合赋值给变量
- 使用del删除整个集合
- 使用集合对象的pop()方法
- 使用集合对象的remove()方法
- 使用集合对象的clear()方法清空集合

2.4.2 集合操作

- 交集、并集、差集等运算
- 使用集合快速提取序列中单一元素

第3章 选择与循环

3.1 条件表达式（1）

- 算术运算符：+、-、*、/、//、%、**
- 关系运算符：>、<、==、<=、>=、!=

- 测试运算符：in、not in、is、is not
- 逻辑运算符：and、or、not，注意短路求值
- 位运算符：~、&、|、 ^、 <<、>>

3.1 条件表达式（2）

- 几乎所有的Python合法表达式都可以作为条件表达式，包括含有函数调用的表达式。
- 条件表达式的值只要不是False、0（或0.0、0j等）、空值None、空列表、空元组、空集合、空字典、空字符串、空range对象或其他空迭代对象，Python解释器均认为与True等价

3.1 条件表达式（3）

- 特殊逻辑运算符“and”和“or”：

  短路求值或惰性求值

- “and”例：“表达式1 and 表达式2”

  if“表达式1”=“False”，不论“表达式2”的值是什么，整个表达式的值都是“False”

- 设计条件表达式表示复杂条件时巧用“and”和“or”（如果能够大概预测不同条件失败的概率）提高程序效率，减少不必要的计算与判断

3.1 条件表达式（4）

- Python条件表达式中不允许使用赋值运算符“=”

```
>>> if a=3:
SyntaxError: invalid syntax
>>> if (a=3) and (b=4):
SyntaxError: invalid syntax
```

避免 “==”写作 “=”

3.2 选择结构（1）

- 单分支：

```
if 表达式:
    语句块
```

```
a,b=input('Input two number:')
if a>b:
    a,b=b,a
print a,b
```

3.2 选择结构（2）

- 双分支：

```
if 表达式:
    语句块1
else:
    语句块2
```

3.2 选择结构（3）

- Python双分支表达式：

```
value1 if condition else value2
```

  - condition的值与True等价时，表达式的值为value1
  - 否则表达式的值为value2
  - value1和value2中可以使用复杂表达式如函数调用
  - 双分支结构的表达式具有惰性求值的特点

3.2 选择结构（4）

- Python多分支结构：

```
if 表达式1:
    语句块1
elif 表达式2:
    语句块2
elif 表达式3:
    语句块3
else:
    语句块4
```

关键字elif是else if的缩写

3.2 选择结构（5）

- Python选择结构嵌套：

```
if 表达式1:
    语句块1
    if 表达式2:
        语句块2
    else:
        语句块3
else:
    if 表达式4:
        语句块4
```

缩进正确且一致

3.3 循环结构（1）

- Python基本的循环语句——while语句、for语句
- while循环一般用于循环次数难以提前确定的情况
- for循环一般用于循环次数可以提前确定的情况
- 优先考虑使用for循环
- 循环结构之间可以互相嵌套

3.3 循环结构（2）

- 循环语句语法

```
while 表达式:
    循环体
```

```
for 取值 in 序列或迭代对象:
    循环体
```

3.3 循环结构（3）

- 循环的else子句

循环自然结束时执行else结构中的语句

```
while 表达式:
    循环体
else:
    else子句
```

```
for 取值 in 序列或迭代对象:
    循环体
else:
    else子句
```

3.3 循环结构（4）

- 编写循环语句时，减少循环内部不必要的计算
- 与循环变量无关的代码提取到循环之外
- 使用多重循环嵌套减少内层循环中不必要的计算

- 循环中尽量引用局部变量，查询和访问速度比全局变量略块

3.4 break和continue语句

- break语句在while循环和for循环中使用

  一般放在if选择结构中，使得整个循环提前结束

- continue语句的作用是终止当前循环

  并忽略continue之后的语句，提前进入下一次循环

- 除非break语句让代码更简单或更清晰，否则不要轻易使用

第4章
字符串与正则表达式

4.0 编码标准（2/5）

- GB2312
  - 中国制定的中文编码
  - 使用1个字节表示英语，2个字节表示中文
  - GBK是GB2312的扩充
  - CP936是微软在GBK基础上完成的编码
  - GB2312、GBK和CP936都是使用2个字节表示中文
  - UTF-8使用3个字节表示中文
- Unicode是编码转换的基础

4.1 字符串

- Python中字符串
  - 属于序列类型
  - 支持序列通用方法和切片操作
  - 支持特有的字符串操作方法
  - 字符串属于不可变序列类型

```
>>> testString = 'good'
>>> id(testString)
19046560
>>> testString[0] = 'b'

Traceback (most recent call last):
  File "<pyshell#25>", line 1, in <module>
    testString[0] = 'b'
TypeError: 'str' object does not support item assignment
>>> testString = 'well'
>>> id(testString)
19047808
```

4.1 字符串

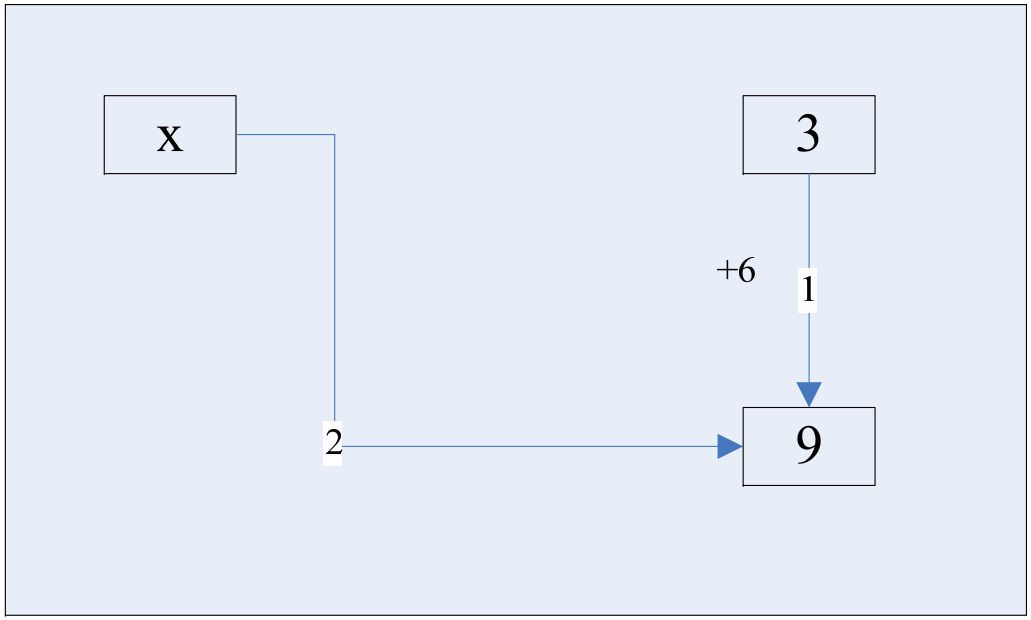
- Python字符串驻留机制
  - 短字符串赋给多个对象时，内存中只有一个副本
  - 长字符串不遵守驻留机制
- 判断一个变量s是否为字符串
  - 使用isinstance（ s, basestring ）

PYTHON内存

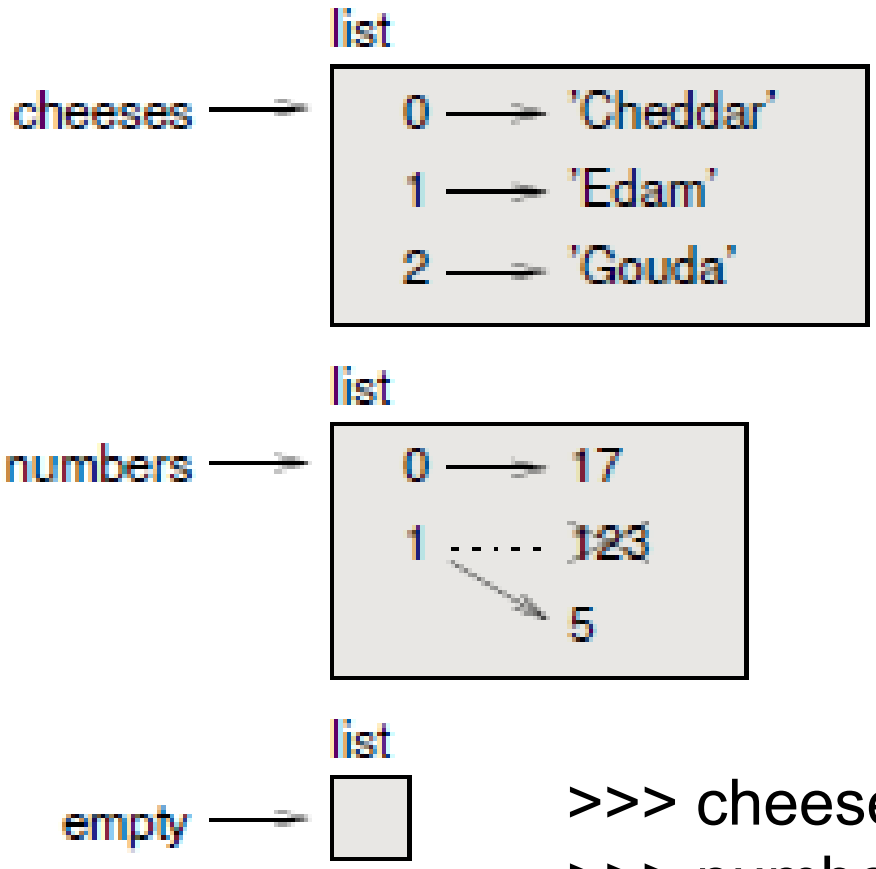
- Python采用基于值的内存管理方式

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


```
>>> x += 6
```



PYTHON内存

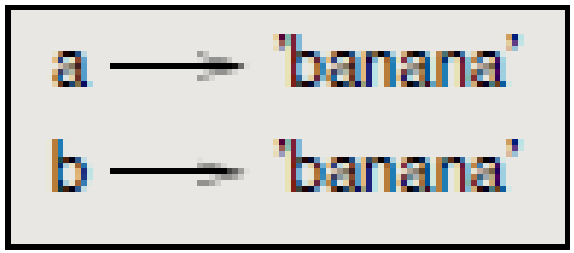



```
>>> cheeses = ['Cheddar', 'Edam', 'Gouda']
>>> numbers = [17, 123]
>>> empty = []
>>> numbers = [17, 123]
>>> numbers[1] = 5
```

PYTHON内存

```
>>> a = [1, 2, 3]
>>> b = [1, 2, 3]
>>> a is b
False
```

```
>>> a = [1, 2, 3]
>>> b = a
>>> a is b
True
```




```
>>> a = [1, 2, 3]
>>> b = a[::]
>>> a is b
False
```

浅拷贝

PYTHON内存

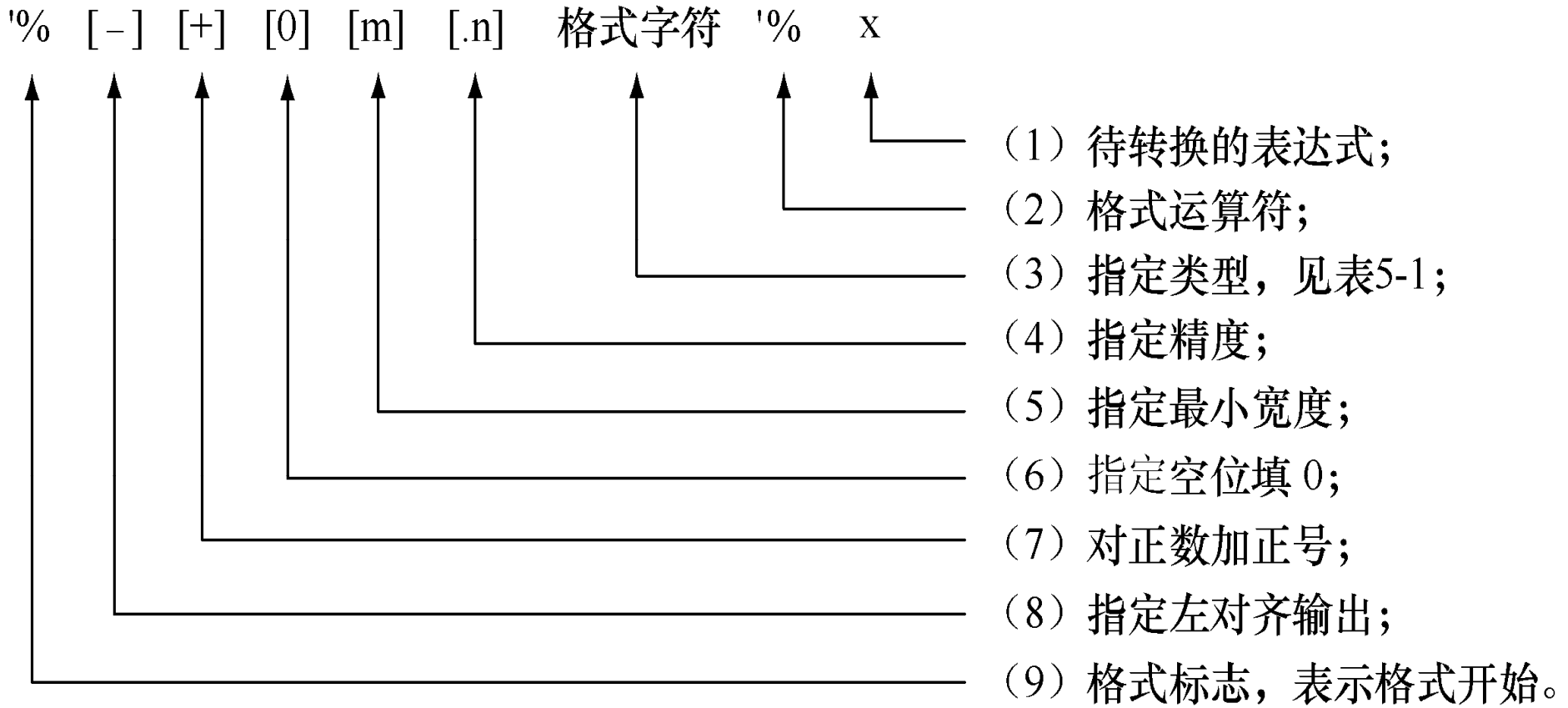
- Python中的对象之间赋值按引用传递
- Python中拷贝对象需要使用标准库中的copy模
  - copy.copy 浅拷贝

  只拷贝父对象，不拷贝对象内部的子对象。
  - copy.deepcopy 深拷贝

  拷贝对象及其子对象

4.1.1 字符串格式化(1/3)

- 常用格式字符



4.1.2 字符串常用方法(1/12)

- find( )、rfind()

查找一个字符串在另一个字符串指定范围（默认是整个字符串）中首次和最后一次出现的位置，如果不存在则返回-1

- index()、rindex()

返回一个字符串在另一个字符串指定范围中首次和最后一次出现的位置，如果不存在则抛出异常

- count()

返回一个字符串在另一个字符串中出现的次数

4.1.2 字符串常用方法(2/12)

- split()、rsplit()

用来以指定字符为分隔符，将字符串左端和右端开始将其分割成多个字符串，并返回包含分割结果的列表

- partition()、rpartition()

指定字符串为分隔符将原字符串分割为3部分，即分隔符前的字符串、分隔符字符串、分隔符后的字符串，如果指定的分隔符不在原字符串中，则返回原字符串和两个空字符串。

4.1.2 字符串常用方法(5/12)

- 字符串联接join( )

```
>>> li=["apple", "peach", "banana", "pear"]
>>> sep=","
>>> s=sep.join(li)
>>> s
"apple,peach,banana,pear"
```

- 不推荐使用+连接字符串，优先使用join()方法

CompareJoinAndPlusForStringConnection.py

4.1.2 字符串常用方法(9/12)

- strip()、rstrip()、lstrip()
  删除两端、右端或左端的空格或连续的指定字符

```
>>> s=" abc  "
>>> s2=s.strip( )
>>> s2
"abc"
>>> "aaaassddf".strip("a")
"ssddf"
>>> "aaaassddf".strip("af")
"ssdd"
>>> "aaaassddfaaa".rstrip("a")
'aaaassddf'
>>> "aaaassddfaaa".lstrip("a")
'ssddfaaa'
```

4.1.2 字符串常用方法(10/12)

- 内置函数eval()

```
>>> eval("3+4")
7
>>> a = 3
>>> b = 5
>>> eval('a+b')
8
>>> import math
>>> eval('math.sqrt(3)')
1.7320508075688772
>>> eval('help(math.sqrt)')
Help on built-in function sqrt in module math:
sqrt(...)
    sqrt(x)
Return the square root of x.
```

```
>>> eval('aa')
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    eval('aa')
  File "<string>", line 1, in <module>
NameError: name 'aa' is not defined
```

4.1.2 字符串常用方法(11/12)

- 成员判断

```
>>> "a" in "abcde"
True
>>> "j" in "abcde"
False
```

- s.startswith(t)、s.endswith(t)

  判断字符串是否以指定字符串开始或结束

4.1.3 字符串常量

```
>>> import string
>>> string.digits
'0123456789'
>>> string.punctuation
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
>>> string.letters
'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz'
>>> string.printable
'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRS
TUVWXYZ!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~\t\n\r\x0b\x0c'
>>> string.lowercase
'abcdefghijklmnopqrstuvwxyz'
>>> string.uppercase
'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

4.2 正则表达式

- 正则表达式

  字符串处理的有力工具和技术

- 正则表达式原理

  使用预定义模式去匹配一类具有共同特征的字符串

  处理字符串：快速、准确地完成复杂的查找、替换等

- Python中re模块提供了正则表达式操作所需要的功能。

4.2.1 正则表达式元字符（1/3）

.：匹配除换行符以外的任意单个字符

*：匹配位于*之前的0个或多个字符

+：匹配位于+之前的一个或多个字符

|：匹配位于|之前或之后的字符

^：匹配行首，匹配以^后面的字符开头的字符串

$：匹配行尾，匹配以$之前的字符结束的字符串

?：匹配位于?之前的0个或1个字符

\：表示位于\之后的为转义字符

[]：匹配位于[]中的任意一个字符

-：用在[]之内用来表示范围

()：将位于()内的内容作为一个整体来对待

{}：按{}中的次数进行匹配

如果以\开头的元字符与转义字符相同，则需要使用\\，或者使用原始字符串。

4.2.1 正则表达式元字符（2/3）

\b：匹配单词头或单词尾
\B：与\b含义相反

\d：匹配任何数字，相当于[0-9]
\D：与\d含义相反

\s：匹配任何空白字符
\S：与\s含义相反

\w：匹配任何字母、数字以及下划线，相当于[a-zA-Z0-9_]
\W：与\w含义相反

4.2.1 正则表达式元字符（3/3）

- 普通字符串可以匹配自身
- '[pjc]ython' 'python'、'jython'、'cython'
- '[a-zA-Z0-9]'可以匹配一个任意大小写字母或数字
- '[^abc]'可以匹配任意一个除'a'、'b'、'c'之外的字符
- 'python|perl'或'p(ython|erl)'都可以匹配'python'或'perl'
- r'(http://)?(www\.)?python\.org'只能匹配'http://www.python.org'、'http://python.org'、'www.python.org'和'python.org'
- '^http'只能匹配所有以'http'开头的字符串
- (pattern)*：允许模式重复0次或多次
- (pattern)+：允许模式重复1次或多次
- (pattern){m,n}：允许模式重复m~n次

4.2.2 re模块的主要方法

- compile(pattern[,flags]):创建模式对象
- search(pattern,string[,flags]):寻找模式
- match(pattern,string[,flags]):匹配模式
- findall(pattern,string[,flags]):列出模式的所有匹配项
- split(pattern,string[,maxsplit=0]):分割字符串
- sub(pat,repl,string[,count=0]):用repl替换pat的匹配项
- escape(string):将字符串中所有特殊正则表达式字符转义

➢ flags值的说明：

re.I	忽略大小写	re.L	
re.M	多行匹配模式	re.S	使元字符.也匹配换行符
re.U	匹配Unicode字符	re.X	忽略模式中的空格，并可以使用#注释

4.2.3 直接使用re模块方法(1/7)

```
>>> import re
>>> text = 'alpha. beta....gamma delta'
>>> re.split('[\. ]+',text)
['alpha', 'beta', 'gamma', 'delta']
>>> re.split('[\. ]+',text,maxsplit=2)#分割2次
['alpha', 'beta', 'gamma delta']
>>> re.split('[\. ]+',text,maxsplit=1)#分割1次
['alpha', 'beta....gamma delta']
>>> pat = '[a-zA-Z]+'
>>> re.findall(pat,text) #查找所有单词
['alpha', 'beta', 'gamma', 'delta']
```

4.2.4 使用正则表达式对象(1/3)

- 使用使用re模块的compile()方法将正则表达式编译生成正则表达式对象,编译后的正则表达式对象可以提高字符串处理速度

- 使用正则表达式对象提供的方法进行字符串处理
  - match(string[, pos[, endpos]]):在字符串开头或指定位置进行搜索，模式必须出现在字符串开头或指定位置
  - search(string[, pos[, endpos]]):搜索;
  - findall(string[, pos[, endpos]]):在字符串中查找所有符合正则表达式的字符串列表。

4.2.5 子模式与match对象(1/2)

- 使()表示一个子模式， ()内的内容作为一个整体出现
- 正则表达式对象的match方法和search方法匹配成功后返回match对象。
- match对象的主要方法

  group()、groups()、groupdict()、start()、end()、span()等等。

4.2.5 子模式与match对象(2/2)

子模式扩展语法：

- (?P<groupname>)：子模式命名
- (?iLmsux)：匹配标志，可为与编译标志含义相同的字母组合
- (?:...)：匹配但不捕获该匹配的子表达式
- (?P=groupname)：命名为groupname的子模式
- (?#...)：表示注释
- (?=…)：用于正则表达式之后，若=后内容出现则匹配
- (?!...)：用于正则表达式之后，若!后的内容不出现则匹配
- (?<=…)：用于正则表达式之前，与(?=…)含义相同
- (?<!...)：用于正则表达式之前，与(?!...)含义相同

第5章
函数的设计和使用

5.1 函数定义(1/4)

- 将可能需要反复执行的代码封装为函数，
- 在需要该段代码功能的地方调用实现代码的复用
- 保证代码的一致性
- 修改函数代码所有调用均受到影响

5.1 函数定义(3/4)

- 定义函数时开头部分的注释可以为用户提供友好的提示和使用帮助。

```
def fib(n):
    '''accept an integer n. return the numbers
    less than n in Fibonacci sequence.'''
    a, b = 1, 1
    while a < n:
        print(a, end=' ')
```

5.2 形参与实参(1/4)

- 函数定义时括弧内为形参，一个函数可以没有形参，但是括弧必须要有，表示该函数不接受参数
- 函数调用时向其传递实参，将实参的值或引用传递给形参
- 在函数内直接修改形参的值不影响实参

5.3 参数类型

- Python函数参数类型：普通参数、默认值参数、关键参数、可变长度参数等等。
- Python定义函数时不需要指定参数的类型，也不需要指定函数的类型，完全由调用者决定，类似于**重载和泛型**
- 函数编写如果有问题，只有在调用时才能被发现，传递某些参数时执行正确，而传递另一些类型的参数时则出现错误

5.3.1 默认值参数(1/4)

def 函数名(形参名=默认值，……)

　　函数体

- 默认值参数必须出现在函数参数列表的最右端
- 任何一个默认值参数右边不能有非默认值参数

```
>>> def f(a=3,b,c=5):
        print a,b,c

SyntaxError: non-default argument follows default argument
>>> def f(a=3,b):
        print a,b

SyntaxError: non-default argument follows default argument
>>> def f(a,b,c=5):
        print a,b,c


>>>
```

5.3.2 关键参数

- 关键参数：调用函数时的传递参数的方式

  通过关键参数传递，实参顺序可以和形参顺序不一致

```
>>> def demo(a,b,c=5):
    print a,b,c
>>> demo(3,7)
3 7 5
>>> demo(a=7,b=3,c=6)
7 3 6
>>> demo(c=8,a=9,b=0)
9 0 8
```

5.4 return语句

- return语句
  - 从一个函数中返回，即跳出函数
  - 从函数中返回一个值
  - 没有return语句，Python以return None结束

```
def maximum( x, y ):
    if x>y:
        return x
    else:
        return y
```

- 调用内置数据类型的方法要注意有否返回值

5.5 变量作用域（2/2）

```
>>> def demo():
    global x
    x = 3
    y =4
    print x,y
>>> x = 5
>>> demo()
3  4
>>> x
3
```

```
>>> y
出错
NameError: name 'y' is not defined
>>> del x
>>> x
出错
NameError: name 'x' is not defined
>>> demo()
3  4
>>> x
3
>>> y
出错
NameError: name 'y' is not defined
```

5.6 lambda表达式（1/4）

- lambda表达式
  - 可以用来声明匿名函数，即没有函数名字的临时使用的小函数
  - 只可以包含一个表达式，且该表达式的计算结果为函数的返回值
  - 不允许包含其他复杂的语句，但在表达式中可以调用其他函数

```
>>> f=lambda x,y,z:x+y+z
>>> print f(1,2,3)
6
```

5.8 高级话题(1/7)

- 内置函数map

  将一个函数作用到一个序列或迭代器对象上

```
>>> map(str,range(5))
['0', '1', '2', '3', '4']
>>> def add5(v):
        return v+5
>>> map(add5,range(10))
[5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

5.8 高级话题(2/7)

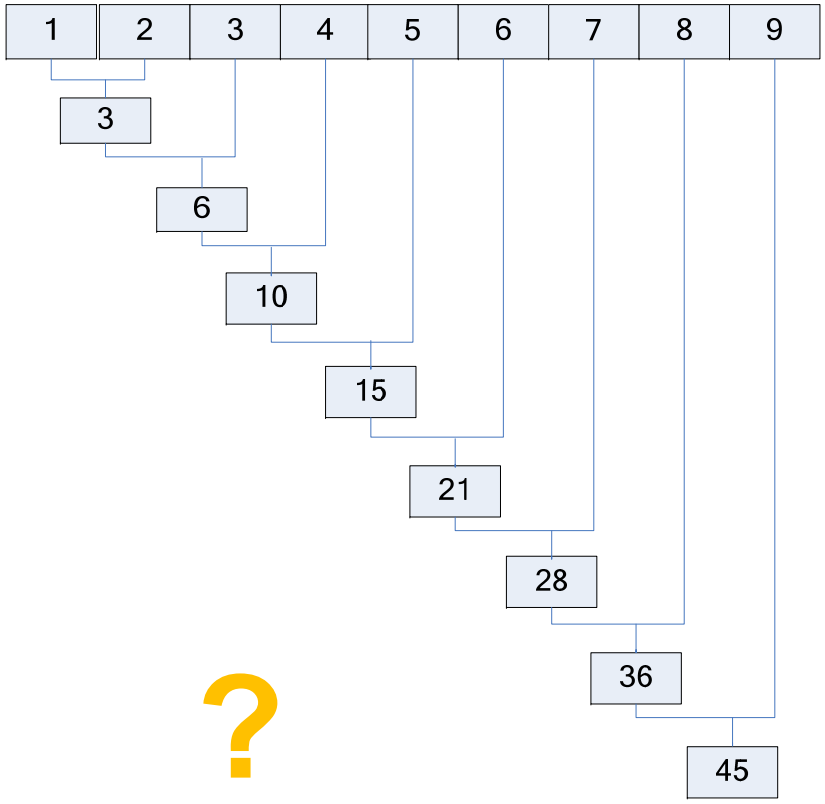
- 内置函数reduce

将一个接受2个参数的函数以累积的方式从左到右依次作用到一个序列或迭代器对象的所有元素上。

```
>>> seq=[1,2,3,4,5,6,7,8,9]
>>> reduce(lambda x,y:x+y, seq)
45

>>> def add(x, y):
    return x + y
>>> reduce(add, range(10))
45

>>> reduce(add,map(str,range(10)))
'0123456789'
```



5.8 高级话题(3/7)

- 内置函数filter

将一个函数作用到一个序列上，返回该序列中使得该函数返回值为True的那些元素组成的列表、元组或字符串。

```
>>> seq=['foo','x41','?!','***']
>>> def func(x):
        return x.isalnum()
>>> filter(func,seq)
['foo', 'x41']
>>> seq
['foo', 'x41', '?!', '***']

>>> [x for x in seq if x.isalnum()]
['foo', 'x41']
>>> filter(lambda x:x.isalnum(),seq)
['foo', 'x41']
```

5.8 高级话题(4/7)

- Yield语句创建生成器：惰性求值，可迭代

```
>>> a = func1()
>>> a.next()

>>> for i in func1():
        print(i)
        if i>1000:
                break

>>> a=func1()
>>> for i in range(10):
        print(a.next())

>>> for i in range(10):
        print(a.next())
```

第6章
面向对象程序设计

6.1 OOP(1/2)

- 面向对象程序设计
  （Object Oriented Programming，OOP）
  - 针对大型软件设计而提出
  - 软件设计更加灵活
  - 支持代码复用和设计复用
  - 使得代码具有更好的可读性和可扩展性
- 基本原则：程序由多个子程序作用的单元或对象组合而成
- 关键观念：数据以及对数据的操作封装成对象，相同类型的对象分类、抽象后，得出共同的特征而形成了类
- 面向对象程序设计：合理地定义和组织类以及类之间的关系

6.1 OOP(2/2)

- Python真正面向对象的高级动态编程语言
- Python完全支持面向对象的基本功能：

  封装、继承、多态以及对基类方法的覆盖或重写

- Python对象的概念广泛，一切皆对象
- Python对象不一定必须是某个类的实例：

  字符串、列表、字典、元组等内置数据类型

- 类的成员：

  成员属性：用变量形式表示的对象

  成员方法：用函数形式表示的对象

6.1.1 类定义语法(1/2)

- Python类定义
  - class关键字+空格+类名+冒号
  - 类名的首字母一般要大写

```
class Car:
    def infor(self):
        print(" This is a car ")
```

- 类用来实例化对象，通过“对象名.成员” 访问数据和成员

```
>>> car = Car()
>>> car.infor()
 This is a car
```

- 使用内置方法isinstance()来测试对象是否为某个类的实例

```
>>> isinstance(car, Car)
True
>>> isinstance(car, str)
False
```

6.1.1 类定义语法(2/2)

- Python关键字“pass”
- 类似于空语句，使用该关键字来“占位”
- 在类和函数的定义中或者选择结构中为软件升级预留空间

```
>>> class A:
        pass

>>> def demo():
        pass

>>> if 5>3:
        pass
```

6.1.2 self参数

- 类的所有实例方法都必须至少有一个“self”的参数
- “self”参数必须是方法的第一个形参
- “self”参数代表将来要创建的对象本身
- 在类的实例方法中访问实例属性时需要以“self”为前缀
- 在类外部用对象名调用对象方法时并不需要传递self参数
- 在类外部通过类名调用对象方法需要显式为self参数传值

6.1.3 类成员与实例成员

- 类成员：指数据成员，或者广义上的属性
- 属性有两种，一种是实例属性，另一种是类属性
  - 实例属性
    - 一般在构造函数__init__()中定义，使用时以self作为前缀
    - 实例属性属于实例(对象)，只能通过对象名访问
  - 类属性
    - 在类中所有方法之外定义的数据成员
    - 类属性属于类，可以通过类名或对象名访问
- 类的方法中可调用类本身的其他方法，可访问类和对象属性
- **Python中可以动态地为类和对象增加成员**

6.1.4 私有成员与公有成员(1/3)

- 定义类的属性时，如果属性名以两个下划线“__”开头则表示是私有属性，否则是公有属性。
- 私有属性是为了数据封装和保密而设的属性，一般只能在类的成员方法（类的内部）中使用访问，在类的外部不能直接访问，需要通过调用对象的公有成员方法来访问
- 公有属性是可以公开使用的，既可以在类的内部进行访问，也可以在外部程序中使用。
- Python提供了访问私有属性的特殊方式，可用于程序的测试和调试，对于成员方法也具有同样的性质。

6.2 方法

- 公有方法

属于对象，可以访问属于类和对象的成员，公有方法通过对象名直接调用，通过类名来调用属于对象的公有方法，需要显式为该方法“self”参数传递一个对象名，用来确定访问哪个对象的数据成员

- 私有方法

属于对象，私有方法的名字以两个下划线“__”开始，可以访问属于类和对象的成员，私有方法在属于对象的方法中通过“self”调用或在外部通过Python支持的特殊方式来调用

- 静态方法

通过类名和对象名调用，不能直接访问属于对象的成员，只能访问属于类的成员

- 类方法

通过类名和对象名调用，不能直接访问属于对象的成员，只能访问属于类的成员，一般将“cls”作为类方法的第一个参数名称，但也可以使用其他的名字作为参数，并且在调用类方法时不需要为“cls”参数传递值

6.3 属性

- 使用@property或property()声明属性
- Python 2.x和Python 3.x对属性的实现和处理方式不一样，内部实现有较大的差异
- Python 2.x为对象新增数据成员隐藏同名已有属性
- Python 3.x属性实现完整，支持更全面的保护机制

6.4 类常用特殊方法

- Python中类的构造函数是__init__()

一般用来为数据成员设置初值或进行其他必要的初始化工作，在创建对象时被自动调用和执行，可以通过为构造函数定义默认值参数来实现类似于其他语言中构造函数重载的目的。如果用户没有设计构造函数，Python将提供一个默认的构造函数用来进行必要的初始化工作。

- Python中类的析构函数是__del__()

一般用来释放对象占用的资源，在Python删除对象和收回对象空间时被自动调用和执行。如果用户没有编写析构函数，Python将提供一个默认的析构函数进行必要的清理工作。

- 其他方法

__init__()	构造函数
__del__()	析构函数
__add__()	左
__sub__()	-
__mul__()	*
__div__()__truediv__()	/.x使用.x使用
__floordiv__()	整除
__mod__()	%
__pow__()	**
__cmp__()	比较运算
__repr__()	打印、转换
__setitem__()	按照索引赋值
__getitem__()	按照索引获取值
__len__()	计算长度
__call__()	函数调用
__contains__()	测试是否包含某个元素
__eq__()、__ne__()、__lt__()、__le__()、__gt__()	==<>
__str__()	转化为字符串
__lshift__()	<<
__and__()	&
__iadd__()	+=

6.5 继承机制

- 继承的目的是代码复用和设计复用
- 继承关系中，已有的、设计好的类称为父类或基类，新设计的类称为子类或派生类
- 派生类可以继承父类的公有成员，但不能继承其私有成员
- 在派生类中调用基类的方法，可以使用内置函数super()或者通过“基类名.方法名()”的方式实现
- Python支持多继承，如果父类中有相同的方法名，而在子类中使用时没有指定父类名，则Python解释器将从左向右按顺序进行搜索