桂林理工大学
GUILIN UNIVERSITY OF TECHNOLOGY

第一章

基础知识

主讲：姜建武

测绘地理信息学院

第一章 Python语言基础

- 1.0 Python是这样一种语言
- 1.1 如何选择Python版本
- 1.2 Python安装与简单使用
- 1.3使用pip管理第三方包
- 1.4运算符、表达式和内置对象
- 1.5Python代码规范
- 1.6Python文件名
- 1.7Python脚本的“__name__”属性
- 1.8编写自己的包和模块
- 1.9Python快速入门
- 1.10 The Zen of Python

1.0 Python是这样一种语言

- Python是一门跨平台、开源、免费的**解释型高级动态编程语言**，支持伪编译将Python源程序转换为**字节码**来优化程序和提高运行速度，支持使用py2exe、pyinstaller或cx_Freeze工具将Python程序**转换为二进制可执行文件**。

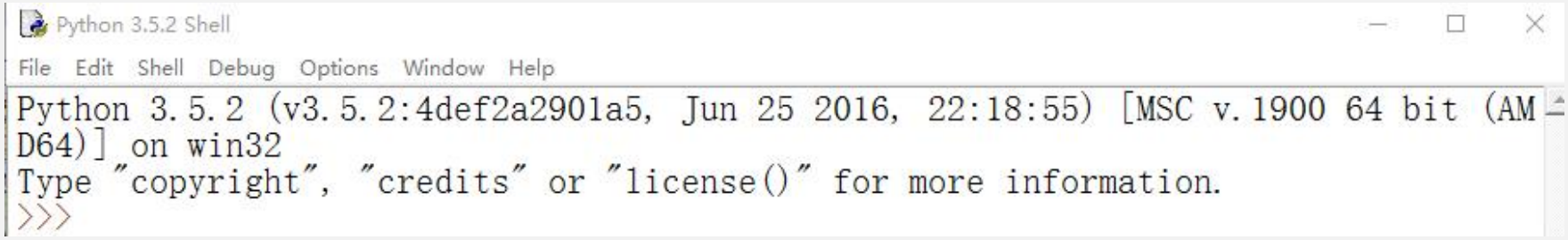
- Python支持**命令式编程**、**函数式编程**，完全支持**面向对象程序设计**，语法简洁清晰，拥有大量的几乎支持所有领域应用开发的成熟**扩展库**。

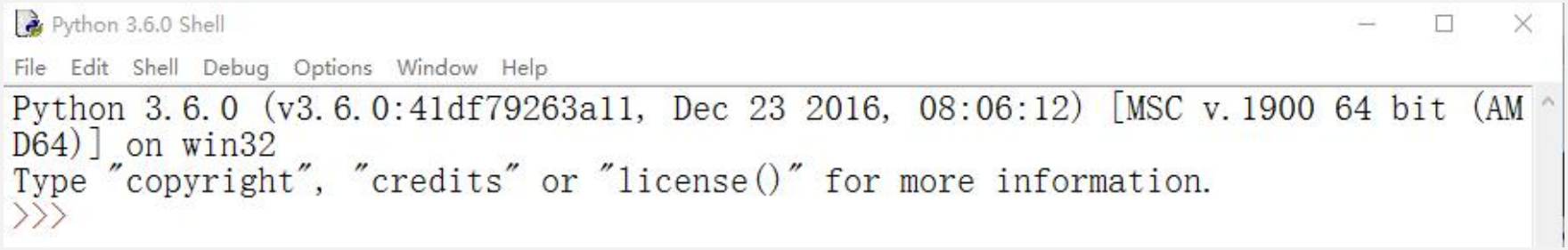
- **胶水语言**：可以把多种不同语言编写的程序融合到一起实现无缝拼接，更好地发挥不同语言和工具的优势，满足不同应用领域的需求。

1.1如何选择Python版本

3.x：必然的趋势，已经开始全面普及

多版本共存与切换简便方法：更改系统环境变量path





1.1如何选择Python版本

安装好Python以后，在“开始”菜单中启动“IDLE（Python GUI）”即可启动Python解释器并可以看到当前安装的Python版本号。

三个大于号“>>>”是Python的提示符，可以在提示符后面输入要执行的语句。

1.2 Python安装与简单使用

- 几个重要网址
  - https://www.python.org/
  - https://www.python.org/doc/
  - http://bugs.python.org/
  - https://hackerone.com/python
  - http://stackoverflow.com/questions/tagged/python

1.2 Python安装与简单使用

- （默认编程环境：IDLE
- 其他常用开发环境：
  - Eclipse+PyDev
  - pyCharm
  - wingIDE
  - Eric
  - PythonWin
  - Anaconda

```
with open(fn) as fp:
    for line in csv.reader(fp):
        if line:
            print(*line)
```

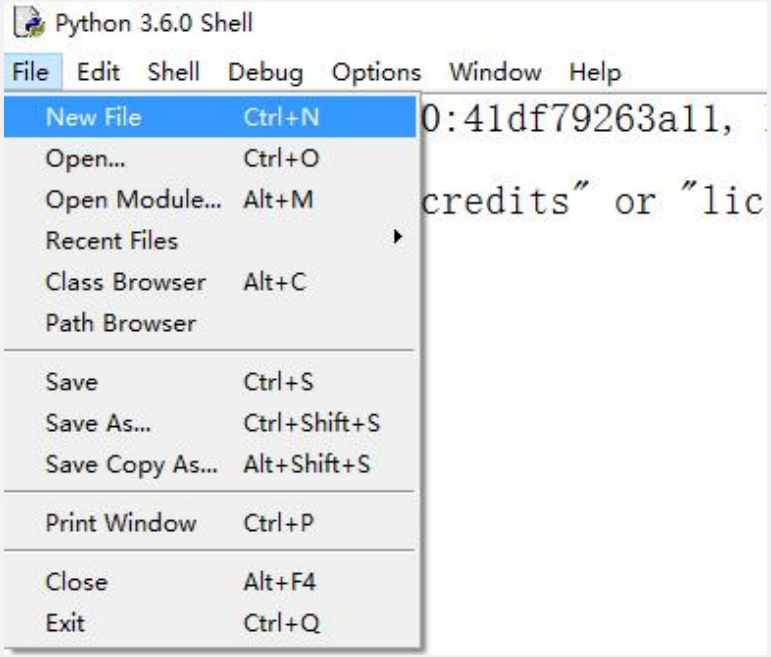
1.2 Python安装与简单使用

- 在IDLE中，如果使用交互式编程模式，那么直接在提示符“>>>”后面输入相应的命令并回车执行即可，如果执行顺利的话，马上就可以看到执行结果，否则会抛出异常。

```
>>> 3+5
8
>>> import math
>>> math.sqrt(9)
3.0
>>> 3*(2+6)
24
>>> 2/0
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    2/0
ZeroDivisionError: integer division or modulo by zero
```

1.2 Python安装与简单使用

- 在IDLE界面中使用菜单“File”==>“New File”创建一个**程序文件**，输入代码并保存为.py或.pyw文件。



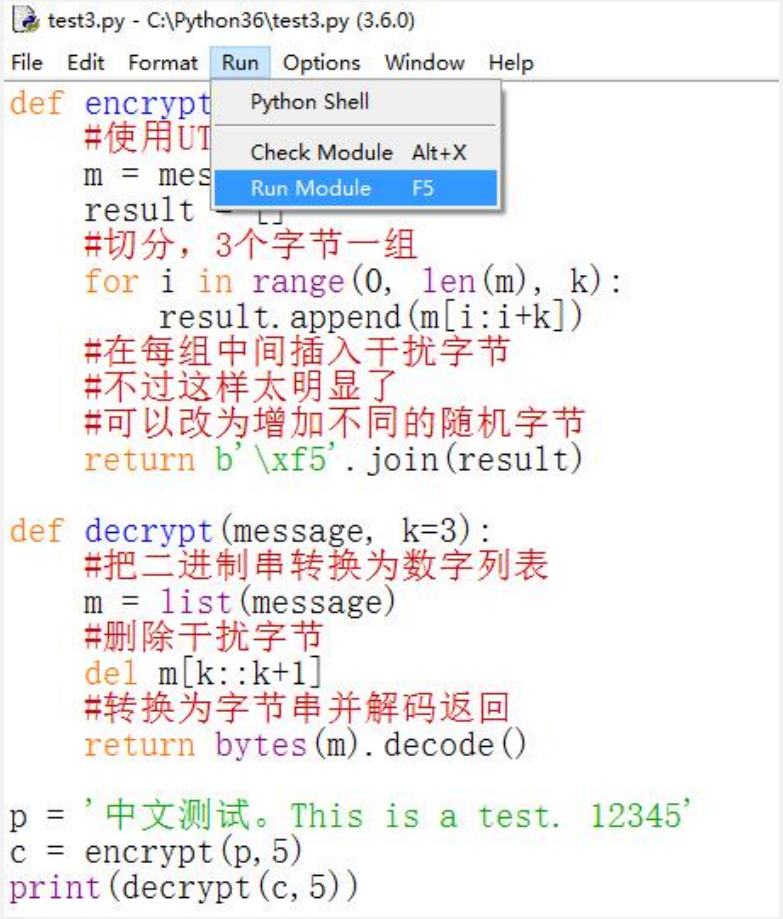
1.2 Python安装与简单使用

- 使用菜单

  “Run”==>“Check Module”来检查程序中是否存在语法错误。

- ，或者使用菜单

  “Run”==>“Run Module”运行程序，程序运行结果将直接显示在IDLE交互界面上。



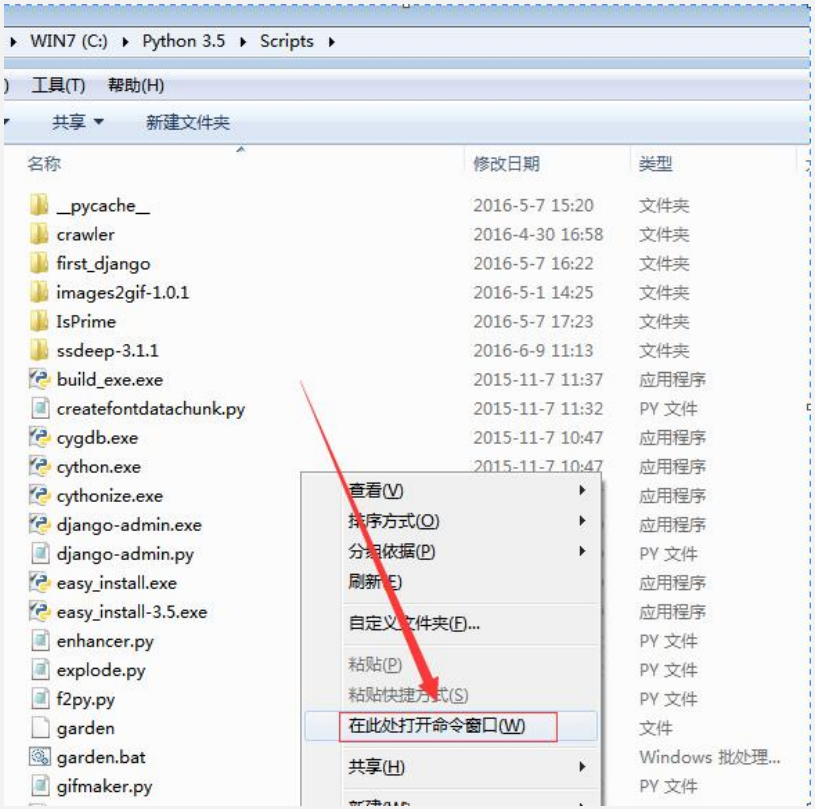
1.2 Python安装与简单使用

- 在有些情况下可能需要在命令提示符环境中运行Python程序文件。在“开始”菜单的“附件”中单击“命令提示符”，然后执行Python程序。假设有程序HelloWorld.py内容如下。

```
def main():
    print('Hello world')
main()
```

1.2 Python安装与简单使用

- 可以在资源管理器中切换至相应的文件夹并直接进入命令提示符环境。



1.2 Python安装与简单使用

- 在IDLE环境下，除了撤销（Ctrl+Z）、全选（Ctrl+A）、复制（Ctrl+C）、粘贴（Ctrl+V）、剪切（Ctrl+X）等常规快捷键之外，其他比较常用的快捷键如下表所示。

快捷键	功能说明
Alt+p	浏览历史命令（上一条）
Alt+n	浏览历史命令（下一条）
Ctrl+F6	重启Shell，之前定义的对象和导入的模块全部失效
F1	打开Python帮助文档
Alt+/	自动补全前面曾经出现过的单词，如果之前有多个单词具有相同前缀，则在多个单词中循环选择
Ctrl+]	缩进代码块
Ctrl+[	取消代码块缩进
Alt+3	注释代码块
Alt+4	取消代码块注释。
Tab	补全单词

1.3 使用pip管理第三方包

- pip工具常用命令

pip命令示例	说明
pip download SomePackage[==version]	下载扩展库的指定版本，不安装
pip freeze [> requirements.txt]	以requirements的格式列出已安装模块
pip list	列出当前已安装的所有模块
pip install SomePackage[==version]	在线安装SomePackage模块的指定版本
pip install SomePackage.whl	通过whl文件离线安装扩展库
pip install package1 package2 ...	依次（在线）安装package1、package2等扩展模块
pip install -r requirements.txt	安装requirements.txt文件中指定的扩展库
pip install --upgrade SomePackage	升级SomePackage模块
pip uninstall SomePackage[==version]	卸载SomePackage模块的指定版本

1.4.1 Python的对象模型

- 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、del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。

1.4.1 Python的对象模型

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,)	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

1.4.1 Python的对象模型

续表

对象类型	类型名称	示例	简要说明
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
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
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

1.4.2 Python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.'。

1.4.2 Python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
```

1.4.2 Python变量

❖如果变量出现在赋值运算符或复合赋值运算符（例如+=、*=等等）的左边则表示创建变量或修改变量的值，否则表示引用该变量的值，这一点同样适用于使用下标来访问列表、字典等可变序列以及其他自定义对象中元素的情况。

```
>>> x = 3          #创建整型变量
>>> print(x**2)
9
>>> x += 6         #修改变量值
>>> print(x)       #读取变量值并输出显示
9
>>> x = [1,2,3] #创建列表对象
>>> x[1] = 5       #修改列表元素值
>>> print(x)       #输出显示整个列表
[1, 5, 3]
>>> print(x[2]) #输出显示列表指定元素
3
```

1.4.2 Python变量

- **字符串和元组属于不可变序列**，不能通过下标的方式来修改其中的元素值，试图修改元组中元素的值时会抛出异常。

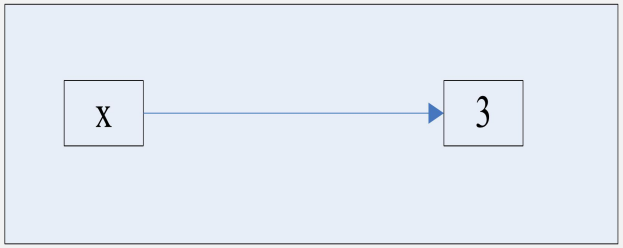
```
>>> x = (1,2,3)
>>> print(x)
(1, 2, 3)

>>> x[1] = 5
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    x[1] = 5
TypeError: 'tuple' object does not support item
 assignment
```

1.4.2 Python变量

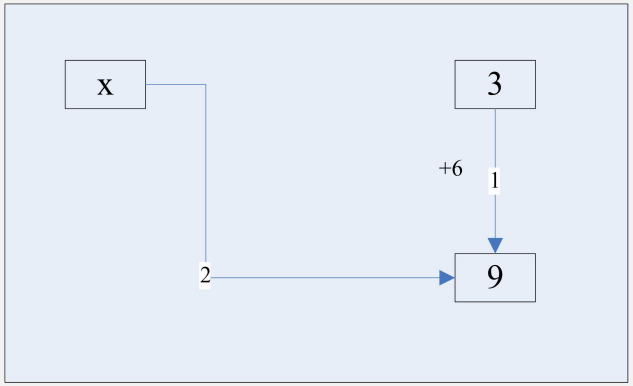
- 在Python中，允许多个变量指向同一个值，例如：

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



- 然而，当为其中一个变量修改值以后，其内存地址将会变化，但这并不影响另一个变量，例如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
```



1.4.2 Python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，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
>>> x = [1, 1, 1, 1]
>>> id(x[0]) == id(x[1])
True
```

1.4.2 Python变量

- Python具有**自动内存管理功能**，对于没有任何变量指向的值，Python自动将其删除。Python会跟踪所有的值，并自动删除不再有变量指向的值。因此，Python程序员一般情况下不需要太多考虑内存管理的问题。

- 尽管如此，显式使用del命令删除不需要的值或显式关闭不再需要访问的资源，仍是一个好的习惯，同时也是一个优秀程序员的基本素养之一。

1.4.2 Python变量

- 在定义变量名的时候，需要注意以下问题：

✓ 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（括号、引号、逗号、斜线、反斜线、冒号、句号、问号等等）；

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

✓ **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，可以通过dir(__builtins__)查看所有内置模块、类型和函数；

✓ 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。

1.4.3 数字

- 数字是**不可变**对象，可以表示任意大小的数字。

```
>>> a=99999999999999999999999999999
>>> a*a
999999999999999999999999999998000000000000000000000000000001
>>> a**3
999999999999999999999999999997000000000000000000000000000002999999999999999999999999999999999
```

- Python的IDEL交互界面可以当做简便计算器来使用。

```
>>> ((3**2) + (4**2)) ** 0.5
5.0
```

1.4.3 数字

- Python中的整数类型可以分为：
  - **十进制整数**如，0、-1、9、123
  - **十六进制整数**，需要16个数字0、1、2、3、4、5、6、7、8、9、a、b、c、d、e、f来表示整数，必须以0x开头，如0x10、0xfa、0xabcdef
  - **八进制整数**，只需要8个数字0、1、2、3、4、5、6、7来表示整数，必须以0o开头，如0o35、0o11
  - **二进制整数**，只需要2个数字0、1来表示整数，必须以0b开头如，0b101、0b100

1.4.3 数字

- 浮点数又称小数

15.0、0.37、-11.2、1.2e2、314.15e-2

1.4.3 数字

- Python内置支持复数类型。

```
>>> a = 3+4j
>>> b = 5+6j
>>> c = a+b
>>> c
(8+10j)
>>> c.real          #查看复数实部
8.0
>>> c.imag          #查看复数虚部
10.0
>>> a.conjugate()  #返回共轭复数
(3-4j)
>>> a*b             #复数乘法
(-9+38j)
>>> a/b             #复数除法
(0.6393442622950819+0.03278688524590165j)
```

1.4.4 字符串

- 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"、'"Tom said, "Let's go""'

- 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，支持排版较为复杂的字符串；三引号还可以在程序中表示较长的注释。

1.4.4 字符串

- 字符串合并

```
>>> a = 'abc' + '123'          #生成新对象
```

- 字符串格式化

```
>>> a = 3.6674
>>> '%7.3f' % a
'  3.667'
>>> "%d:%c"%(65,65)
'65:A'
>>> """My name is %s, and my age is %d""" % ('Dong Fuguo',39)
'My name is Dong Fuguo, and my age is 39'
```

1.4.4 字符串

- 常用转义字符

转义字符	含义	转义字符	含义
\b	退格，把光标移动到前一列位置	\\	一个斜线\
\f	换页符	\’	单引号’
\n	换行符	\”	双引号”
\r	回车	\ooo	3位八进制数对应的字符
\t	水平制表符	\xhh	2位十六进制数对应的字符
\v	垂直制表符	\uhhhh	4位十六进制数表示的Unicode字符

1.4.4 字符串

- 转义字符用法

```
>>> print('Hello\nWorld')              #包含转义字符的字符串
Hello
World
>>> print('\101')                      #三位八进制数对应的字符
A
>>> print('\x41')                      #两位十六进制数对应的字符
A
>>> print('我是\u8463\u4ed8\u56fd')  #四位十六进制数表示的Unicode字符
我是董付国
```

1.4.4 字符串

- 字符串界定符前面加字母r表示原始字符串，其中的特殊字符不进行转义，但字符串的最后一个字符不能是\。原始字符串主要用于正则表达式、文件路径或者URL的场合。

```
>>> path = 'C:\Windows\notepad.exe'
>>> print(path)                    #字符\n被转义为换行符
C:\Windows
otepad.exe
>>> path = r'C:\Windows\notepad.exe'  #原始字符串，任何字符都不转义
>>> print(path)
C:\Windows\notepad.exe
```

1.4.5 运算符和表达式

Python运算符与功能

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或内存地址是否相同
|、^、&、<<、>>、~	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

1.4.5 运算符和表达式

- +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
```

1.4.5 运算符和表达式

- *运算符不仅可以用于数值乘法，还可以用于列表、字符串、元组等类型，当列表、字符串或元组等类型变量与整数进行“*”运算时，表示对内容进行重复并返回重复后的新对象。

```
>>> 2.0 * 3                          #浮点数与整数相乘
6.0
>>> (3+4j) * 2                       #复数与整数相乘
(6+8j)
>>> (3+4j) * (3-4j)                  #复数与复数相乘
(25+0j)
>>> "a" * 10                         #字符串重复
'aaaaaaaaaa'
>>> [1,2,3] * 3                      #列表重复
[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> (1,2,3) * 3                      #元组重复
(1, 2, 3, 1, 2, 3, 1, 2, 3)
```

1.4.5 运算符和表达式

- Python中的除法有两种，“/”和“//”分别表示除法和整除运算，并且Python 2.x和Python 3.x对“/”运算符的解释也略有区别。在Python 3.5.2中运算结果如下：

```
>>> 3/5
0.6
>>> 3//5
0
>>> 3.0/5
0.6
>>> 3.0//5
0.0
>>> 13//10
1
>>> -13//10
-2
```

1.4.5 运算符和表达式

- 上面的表达式在Python 2.7.12中运算结果如下：

```
>>> 3/5
0
>>> 3//5
0
>>> 3.0/5
0.6
>>> 3.0//5
0.0
>>> 13//10
1
>>> -13//10
-2
```

1.4.5 运算符和表达式

- %运算符除去可以用于字符串格式化之外，还可以对整数和浮点数计算余数。但是由于浮点数的精确度影响，计算结果可能略有误差。

```
>>> 3.1%2
1.1
>>> 6.3%2.1
2.0999999999999996
>>> 6%2
0
>>> 6.0%2
0.0
>>> 6.0%2.0
0.0
>>> 5.7%4.8
0.9000000000000004
```

1.4.5 运算符和表达式

- 关系运算符可以连用

```
>>> 1 < 3 < 5                    #等价于1 < 3 and 3 < 5
True
>>> 'Hello' > 'world'            #比较字符串大小
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()
>>> {1, 2, 3} < {1, 2, 3, 4}     #测试是否子集
```

1.4.5 运算符和表达式

- 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数字的内置函数
True
>>> 'abc' in 'abcdefg'              #子字符串测试
True
>>> for i in (3, 5, 7):             #循环，成员遍历
  print(i, end='\t')
3 5     7
```

1.4.5 运算符和表达式

- 同一性测试运算符（identity comparison）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
```

1.4.5 运算符和表达式

- 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。

```
>>> 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
```

1.4.5 运算符和表达式

- 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}          #对称差集
{1, 2, 4, 5}
>>> {1, 2, 3} - {3, 4, 5}          #差集
{1, 2}
```

1.4.5 运算符和表达式

- 逻辑运算符and和or具有**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
>>> 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
```

1.4.5 运算符和表达式

- Python 3.5增加了一个新的矩阵相乘运算符@

```
>>> import numpy              #numpy是用于科学计算的Python扩展库
>>> x = numpy.ones(3)         #ones()函数用于生成全1矩阵
>>> m = numpy.eye(3)*3        #eye()函数用于生成单位矩阵
>>> m[0,2] = 5                #设置矩阵指定位置上元素的值
>>> m[2, 0] =3
>>> x @ m                     #矩阵相乘
array([ 6.,  3.,  8.])
```

1.4.5 运算符和表达式

- 逗号并不是运算符，只是一个普通分隔符。

```
>>> 'a' in 'b', 'a'
(False, 'a')
>>> 'a' in ('b', 'a')
True
>>> x = 3, 5
>>> x
(3, 5)
>>> 3 == 3, 5
(True, 5)
>>> x = 3+5, 7
>>> x
(8, 7)
```

1.4.5 运算符和表达式

- 在Python中，单个任何类型的对象或常数属于合法表达式，使用运算符连接的变量和常量以及函数调用的任意组合也属于合法的表达式。

```
>>> a = [1,2,3]
>>> b = [4,5,6]
>>> c = a + b
>>> c
[1, 2, 3, 4, 5, 6]
>>> d = list(map(str, c))
>>> d
['1', '2', '3', '4', '5', '6']
>>> import math
>>> list(map(math.sin, c))
[0.8414709848078965, 0.9092974268256817, 0.141120008059867
2, -0.7568024953079282, -0.9589242746631385, -
0.2794154981989258
6]
>>> 'Hello' + ' ' + 'world'
'Hello world'
>>> 'welcome ' * 3
'welcome welcome welcome '
>>> ('welcome,'*3).rstrip(',')+'!'
'welcome,welcome,welcome!'
```

1.4.6 常用内置函数

- 内置函数不需要导入任何模块即可使用
- 执行下面的命令可以列出所有内置函数

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

1.4.6 常用内置函数

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

1.4.6 常用内置函数

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]))	创建不可变的字典对象
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

1.4.6 常用内置函数

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

1.4.6 常用内置函数

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

1.4.6 常用内置函数

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

1.4.6 常用内置函数

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，要求序列x中所有元素必须为数字，允许指定起始值start，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

1.4.6 常用内置函数

- dir()函数可以查看指定模块中包含的所有成员或者指定对象类型所支持的操作。
- help()函数则返回指定模块或函数的说明文档。

1.4.6 常用内置函数

- ord()和chr()是一对功能相反的函数，ord()用来返回单个字符的序数或Unicode码，而chr()则用来返回某序数对应的字符，str()则直接将其任意类型参数转换为字符串。

```
>>> ord('a')
97
>>> chr(65)
'A'
>>> chr(ord('A')+1)
'B'
>>> str(1)
'1'
>>> str(1234)
'1234'
>>> str([1,2,3])
'[1, 2, 3]'
>>> str((1,2,3))
'(1, 2, 3)'
>>> str({1,2,3})
'set([1, 2, 3])'
```

1.4.6 常用内置函数

- max()、min()、sum()这三个内置函数分别用于计算列表、元组或其他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()要求元素支持加法运算，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。

```
>>> import random
>>> a = [random.randint(1,100) for i in range(10)]    #列表推导式
>>> a
[72, 26, 80, 65, 34, 86, 19, 74, 52, 40]
>>> print(max(a), min(a), sum(a))
86 19 548
```

- 如果需要计算该列表中的所有元素的平均值，可以直接这样用：

```
>>> sum(a)*1.0/len(a)                    #Python 2.7.12
54.8
>>> sum(a)/len(a)                        #Python 3.5.2
54.8
```

1.4.6 常用内置函数

- 内置函数max()和min()的key参数可以用来指定比较规则

```
>>> x = ['21', '1234', '9']
>>> max(x)
'9'
>>> max(x, key=len)
'1234'
>>> max(x, key=int)
'1234'
```

1.4.6 常用内置函数

- 内置函数type()和isinstance()可以判断数据类型。

```
>>> 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
>>> isinstance(3, int)                     #判断3是否为int类型的实例
True
>>> isinstance(3j, (int, float, complex))  #判断3是否为int,float或complex类型
True
```

1.4.6 常用内置函数

- 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等类似对象除外）进行翻转（首尾交换）并返回可迭代的reversed对象。

```
>>> 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']
>>> 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
[5, 1, 9, 3, 8, 7, 10, 6, 0, 4, 2]
```

1.4.6 常用内置函数

- 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索引和值的元组。

```
>>> list(enumerate('abcd'))                          #枚举字符串中的元素
[(0, 'a'), (1, 'b'), (2, 'c'), (3, 'd')]
>>> list(enumerate(['Python', 'Greate']))            #枚举列表中的元素
[(0, 'Python'), (1, 'Greate')]
>>> list(enumerate({'a':97, 'b':98, 'c':99}.items()))  #枚举字典中的元素
[(0, ('c', 99)), (1, ('a', 97)), (2, ('b', 98))]
>>> 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)
```

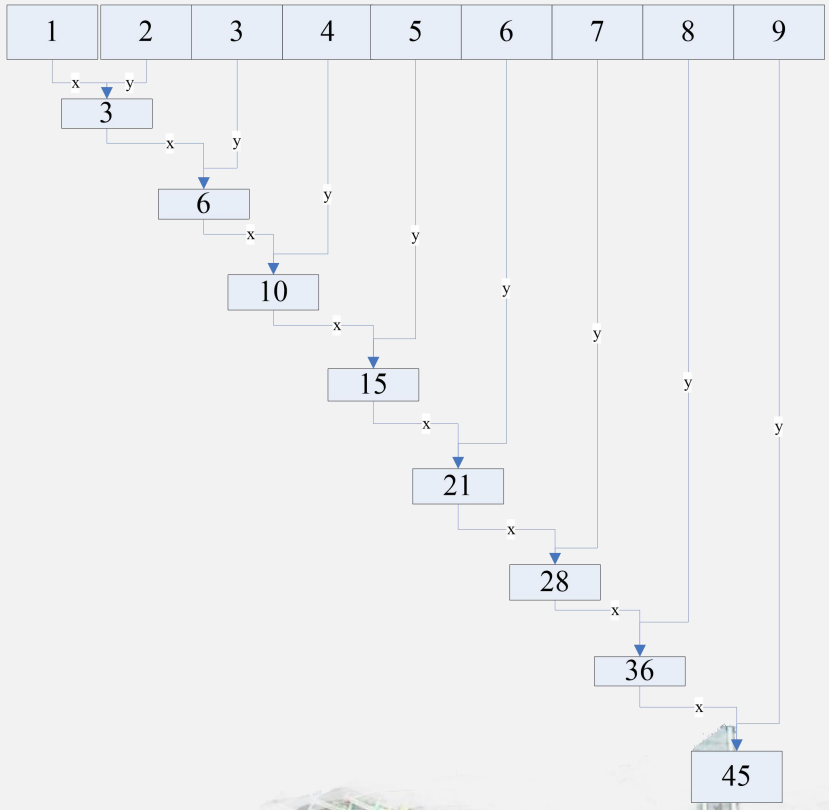
1.4.6 常用内置函数

- 内置函数map()把一个函数func依次映射到序列或迭代器对象的每个元素上，并返回一个可迭代的map对象作为结果，map对象中每个元素是原序列中元素经过函数func处理后的结果。

```
>>> list(map(str, range(5)))              #把列表中元素转换为字符串
['0', '1', '2', '3', '4']
>>> def add5(v):                          #单参数函数
    return v+5
>>> list(map(add5, range(10)))            #把单参数函数映射到一个序列的所有元素
[5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
>>> def add(x, y):                                #可以接收2个参数的函数
    return x+y
>>> list(map(add, range(5), range(5,10)))      #把双参数函数映射到两个序列上
[5, 7, 9, 11, 13]
```

1.4.6 常用内置函数

- 标准库functools中的函数reduce()可以将一个接收2个参数的函数以迭代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，并且允许指定一个初始值。




```
>>> from functools import reduce
>>> seq = list(range(1, 10))
>>> reduce(lambda x, y: x+y, seq)
45
```

1.4.6 常用内置函数

- 内置函数filter()将一个单参数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filter对象，如果指定函数为None，则返回序列中等价于True的元素。

```
>>> seq = ['foo', 'x41', '?!', '***']
>>> def func(x):
    return x.isalnum()                  #测试是否为字母或数字
>>> filter(func, seq)                          #返回filter对象
<filter object at 0x000000000305D898>
>>> list(filter(func, seq))                    #把filter对象转换为列表
['foo', 'x41']
```

1.4.6 常用内置函数

- range()是Python开发中非常常用的一个内置函数，语法格式为range([start,] end [, step])。该函数返回具有**惰性求值**特点的range对象，其中包含**左闭右开区间**[start,end)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end大
[9, 7, 5, 3, 1]
```

1.4.6 常用内置函数

- zip()函数用来把多个可迭代对象中的元素压缩到一起，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]))          #压缩字符串和列表
[('a', 1), ('b', 2), ('c', 3)]
>>> list(zip('123', 'abc', ',.!'))        #压缩3个序列
[('1', 'a', ','), ('2', 'b', '.'), ('3', 'c', '!')]
>>> x = zip('abcd', '1234')
>>> list(x)
[('a', '1'), ('b', '2'), ('c', '3'), ('d', '4')]
```

1.4.7 对象的删除

- 在Python中具有**自动内存管理功能**，Python解释器会跟踪所有的值，一旦发现某个值不再有任何变量指向，将会自动删除该值。尽管如此，自动内存管理或者垃圾回收机制并不能保证及时释放内存。显式释放自己申请的资源是程序员的好习惯之一，也是程序员素养的重要体现之一。

- 在Python中，可以使用del命令来显式删除对象并解除与值之间的指向关系。删除对象时，如果其指向的值还有别的变量指向则不删除该值，如果删除对象后该值不再有其他变量指向，则删除该值。

1.4.7 对象的删除

```
>>> x = [1,2,3,4,5,6]
>>> y = 3
>>> z = y
>>> print(y)
3
>>> del y #删除对象
>>> print(y)
NameError: name 'y' is not defined
>>> print(z)
3
>>> del z
>>> print(z)
NameError: name 'z' is not defined
>>> del x[1] #删除列表中指定元素
>>> print(x)
[1, 3, 4, 5, 6]
>>> del x #删除整个列表
>>> print(x)
NameError: name 'x' is not defined
```

1.4.7 对象的删除

- del命令无法删除元组或字符串中的元素，只可以删除整个元组或字符串，因为这两者均属于不可变序列。

```
>>> x = (1,2,3)
>>> del x[1]
Traceback (most recent call last):
  File "<pyshell#62>", line 1, in <module>
    del x[1]
TypeError: 'tuple' object doesn't support item deletion
>>> del x
>>> print(x)
Traceback (most recent call last):
  File "<pyshell#64>", line 1, in <module>
    print(x)
NameError: name 'x' is not defined
```

1.4.8 基本输入输出

- 用Python进行程序设计，输入是通过input( )函数来实现的，imput( )的一般格式为：

```
x = input('提示：')
```

该函数返回输入的对象。可输入数字、字符串和其它任意类型对象。

1.4.8 基本输入输出

- 尽管形式一样，Python 2.x和Python 3.x对该函数的解释略有不同。在Python 2.x中，该函数返回结果的类型由输入值时所使用的界定符来决定，例如下面的Python 2.7.12代码：

```
>>> x = input("Please input:")
Please input:3                  #没有界定符，整数
>>> print type(x)
<type 'int'>
>>> x = input("Please input:")
Please input:'3'                #单引号，字符串
>>> print type(x)
<type 'str'>
>>> x = input("Please input:")
Please input:[1,2,3]            #方括号，列表
>>> print type(x)
<type 'list'>
```

1.4.8 基本输入输出

- 在Python 2.x中，还有另外一个内置函数raw_input()也可以用来接收用户输入的值。与input()函数不同的是，raw_input()函数返回结果的类型一律为字符串，而不论用户使用什么界定符。例如：

```
>>> x = raw_input("Please input:")
Please input:[1,2,3]
>>> print type(x)
<type 'str'>
```

1.4.8 基本输入输出

- 在Python 3.x中，不存在raw_input()函数，只提供了input()函数用来接收用户的键盘输入。在Python 3.x中，不论用户输入数据时使用什么界定符，input()函数的**返回结果都是字符串**，需要将其转换为相应的类型再处理。例如下面的Python 3.5.2代码：

```
>>> x = input('Please input:')
Please input:3
>>> print(type(x))
<class 'str'>
>>> x = input('Please input:')
Please input:'1'
>>> print(type(x))
<class 'str'>
>>> x = input('Please input:')
Please input:[1,2,3]
>>> print(type(x))
<class 'str'>
>>> x = raw_input('Please input:')
NameError: name 'raw_input' is not defined
```

1.4.8 基本输入输出

- Python 2.x和Python 3.x的输出方法也不一样。

- ✓在Python 2.x中，使用print语句进行输出

- ✓Python 3.x中使用print()函数进行输出。

1.4.8 基本输入输出

- 默认情况下，Python将结果输出到IDLE或者标准控制台，在输出时也可以进行重定向，例如可以把结果输出到指定文件。在Python 2.7.12中使用下面的方法进行输出重定向：

```
>>> fp = open(r'C:\mytest.txt', 'a+')
>>> print >>fp, "Hello,world"
>>> fp.close()
```

- 而在Python 3.5.2中则需要使用下面的方法进行重定向：

```
>>> fp = open(r'D:\mytest.txt', 'a+')
>>> print('Hello,world!', file = fp)
>>> fp.close()
```

1.4.8 基本输入输出

- 另外一个重要的不同是，对于Python 2.x而言，在print语句之后加上逗号“,”则表示输出内容之后不换行，例如：

```
>>> for i in range(10):
    print i,
0 1 2 3 4 5 6 7 8 9
```

- 在Python 3.x中，为了实现上述功能则需要使用下面的方法：

```
>>> for i in range(10,20):
    print(i, end=' ')
10 11 12 13 14 15 16 17 18 19
```

1.4.9 模块导入与使用

- Python默认安装仅包含部分基本或核心模块，但用户可以安装大量的扩展模块，pip是管理模块的重要工具。
- 在Python启动时，仅加载了很少的一部分模块，在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。
- 可以使用sys.modules.items()显示所有预加载模块的相关信息。

1.4.9 模块导入与使用

❖import 模块名

```
>>>import math
>>>math.sin(0.5)                #求0.5的正弦
>>>import random
>>>x=random.random( )           #获得[0,1) 内的随机小数
>>>y=random.random( )
>>>n=random.randint(1,100)      #获得[1,100]上的随机整数
```

✓可以使用dir函数查看任意模块中所有的对象列表，如果调用不带参数的dir()函数，则返回当前所有名字列表。

✓可以使用help函数查看任意模块或函数的使用帮助。

1.4.9 模块导入与使用

- from 模块名 import 对象名[as 别名] #可以减少查询次数，提高执行速度
- from math import *　　#谨慎使用

```
>>> from math import sin
>>> sin(3)
0.1411200080598672
>>> from math import sin as f #别名
>>> f(3)
0.1411200080598672
```

1.4.9 模块导入与使用

- 在2.x中可以使用reload函数重新导入一个模块，在3.x中，需要使用imp模块的reload函数
- Python首先在当前目录中查找需要导入的模块文件，如果没有找到则从sys模块的path变量所指定的目录中查找。可以使用sys模块的path变量查看python导入模块时搜索模块的路径，也可以向其中append自定义的目录以扩展搜索路径。
- 在导入模块时，会优先导入相应的pyc文件，如果相应的pyc文件与py文件时间不相符，则导入py文件并重新编译该模块。

1.4.9 模块导入与使用

- 导入模块时的文件搜索顺序
  - ✓当前文件夹
  - ✓sys.path变量指定的文件夹
  - ✓优先导入pyc文件

1.4.9 模块导入与使用

- 如果需要导入多个模块，一般建议按如下顺序进行导入：
  - ✓标准库
  - ✓成熟的第三方扩展库
  - ✓自己开发的库

1.5 Python代码规范

（1）缩进

- ✓类定义、函数定义、选择结构、循环结构，行尾的冒号表示缩进的开始
- ✓ python程序是依靠代码块的缩进来体现代码之间的逻辑关系的，缩进结束就表示一个代码块结束了。
- ✓ 同一个级别的代码块的缩进量必须相同。
- ✓一般而言，以4个空格为基本缩进单位

1.5 Python代码规范

（2）注释

一个好的、可读性强的程序一般包含30%以上的注释。常用的注释方式主要有两种：

- ✓ 以#开始，表示本行#之后的内容为注释
- ✓ 包含在一对三引号'''...'''或"""..."""之间且不属于任何语句的内容将被解释器认为是注释

- ■ 在IDLE开发环境中，可以通过下面的操作快速注释/解除注释大段内容：Format➔Comment Out Region/Uncomment Region

1.5 Python代码规范

（3）每个import只导入一个模块

（4）如果一行语句太长，可以在行尾加上\来换行分成多行，但是更建议使用括号来包含多行内容。

（5）必要的空格与空行

- ✓运算符两侧、函数参数之间、逗号两侧建议使用空格分开。
- ✓不同功能的代码块之间、不同的函数定义之间建议增加一个空行以增加可读性。

（6）适当使用异常处理结构进行容错，后面将详细讲解。

（7）软件应具有较强的可测试性，测试与开发齐头并进。

1.6 Python文件名

- .py：Python源文件，由Python解释器负责解释执行。
- .pyw：Python源文件，常用于图形界面程序文件。
- .pyc：Python字节码文件，无法使用文本编辑器直接查看该类型文件内容，可用于隐藏Python源代码和提高运行速度。对于Python模块，第一次被导入时将被编译成字节码的形式，并在以后再次导入时优先使用“.pyc”文件，以提高模块的加载和运行速度。对于非模块文件，直接执行时并不生成“.pyc”文件，但可以使用py_compile模块的compile()函数进行编译以提高加载和运行速度。另外，Python还提供了compileall模块，其中包含compile_dir()、compile_file()和compile_path()等方法，用来支持批量Python源程序文件的编译。

1.6 Python文件名

- .pyo：优化的Python字节码文件，同样无法使用文本编辑器直接查看其内容。可以使用“python -O -m py_compile file.py”或“python -OO -m py_compile file.py”进行优化编译。Python 3.5不再支持.pyo文件。
- .pyd：一般是由其他语言编写并编译的二进制文件，常用于实现某些软件工具的Python编程接口插件或Python动态链接库。

1.7 Python脚本的“__name__”属性

- 每个Python脚本在运行时都有一个“__name__”属性。如果脚本作为模块被导入，则其“__name__”属性的值被自动设置为模块名；如果脚本独立运行，则其“__name__”属性值被自动设置为“__main__”。例如，假设文件nametest.py中只包含下面一行代码：

```
print(__name__)
```

- 在IDLE中直接运行该程序时，或者在命令行提示符环境中运行该程序文件时，运行结果如下：

```
__main__
```

- 而将该文件作为模块导入时得到如下执行结果：

```
>>> import nametest
nametest
```

1.7 Python脚本的“__name__”属性

- 利用“__name__”属性即可控制Python程序的运行方式。例如，编写一个包含大量可被其他程序利用的函数的模块，而不希望该模块可以直接运行，则可以在程序文件中添加以下代码：

```
if __name__ == '__main__':
    print('Please use me as a module.')
```

- 这样一来，程序直接执行时将会得到提示“Please use me as a module.”，而使用import语句将其作为模块导入后可以使用其中的类、方法、常量或其他成员。

1.8 编写自己的包与模块

- 在包的每个目录中都必须包含一个__init__.py文件，该文件可以是一个空文件，仅用于表示该目录是一个包。

- __init__.py文件的主要用途是设置__all__变量以及所包含的包初始化所需的代码。其中__all__变量中定义的对象可以在使用from …import *时全部正确导入。

1.9 Python快速入门

- 例1-1：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？

1.9 Python快速入门

- 还可以这样写

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

还可以再简单些吗？

1.9 Python快速入门

- 居然可以这样？OMG

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

1.9 Python快速入门

❖例1-2：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)
```

1.9 Python快速入门

❖例1-3：任意输入三个英文单词，按字典顺序输出。

```
s = input('x,y,z=')
x, y, z = s.split(',')
if x > y:
   x, y = y, x
if x > z:
    x, z = z, x
if y > z:
    y, z = z, y
print(x, y, z)
```

- 或直接写为：

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

1.9 Python快速入门

- 例1-4：Python程序框架生成器。

```
import os
import sys
import datetime
head = '# '+'-'*20+'\n'+\
       '# Function description:\n'+\
       '# '+'-'*20+'\n'+\
       '# Author: Dong Fuguo\n'+\
       '# QQ: 306467355\n'+\
       '# Email: dongfuguo2005@126.com\n'+\
       '#'+'-'*20+'\n'
```

1.9 Python快速入门

```
desFile = sys.argv[1]
if os.path.exists(desFile) or not desFile.endswith('.py'):
    print('%s already exist or is not a Python code
file.!'%desFile)
    sys.exit()
fp = open(desFile, 'w')
today = str(datetime.date.today())
fp.write('# -*- coding:utf-8 -*-\n')
fp.write('# Filename: '+desFile+'\n')
fp.write(head)
fp.write('# Date: '+today+'\n')
fp.write('# '+'-'*20+'\n')
fp.close()
```

1.10 The Zen of Python

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.
- Special cases aren't special enough to break the rules.
- Although practicality beats purity.
- Errors should never pass silently.
- Unless explicitly silenced.
- In the face of ambiguity, refuse the temptation to guess.
- There should be one-- and preferably only one --obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.
- Now is better than never.
- Although never is often better than *right* now.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.
- Namespaces are one honking great idea -- let's do more of those!