Python基础(一)

姜城

2015/3

目录

- Python标准数据类型
- 输入输出
- 控制结构：条件与循环
- 异常处理
- 函数定义

1 Python标准数据类型

Python标准数据类型

- 标准原子数据类型
  - 整型int
  - 浮点型float
  - 布尔型bool
- 变量和赋值过程
- 标准容器数据类型
  - 列表list
  - 字符串string
  - 元组tuple
  - 集合set
  - 字典dict

整型和浮点型1/1

- 整型：int
- 浮点型：float
- 算数运算符：
  - 加法+
  - 减法-
  - 乘法*
  - 除法/
  - 地板除//
  - 取余%
  - 乘方**

```
>>> 2+3*4
14
>>> (2+3)*4
20
>>> 7/3
2
>>> 7.0/3
2.3333333333333335
>>> 7//3
2
>>> 7.0//3
2.0
>>> 3%6
3
>>> 2**100
1267650600228229401496703205376L
```

布尔型1/2

- True/False
- 逻辑操作符
  - 与 and（C语言中的“&&”）
  - 或 or（C语言中的“||”）
  - 非 not（C语言中的“!”）

```
>>> not (False or True)
False
>>> False or (not False)
True
```

	A-True B-True	A-True B-False	A-False B-False	A-False B-False
A and B	True	False	False	False
A or B	True	True	True	False
not A	False		True	

布尔型2/2

- bool对象可以作为条件判断的结果

Operation	Meaning
<	strictly less than
<=	less than or equal
>	strictly greater than
>=	greater than or equal
==	equal
!=	not equal
is	object identity
is not	negated object identity

```
>>> 5==10
False
>>> 10>5
True
>>> 5>=5
True
>>> (5>=1) and (5<=10)
True
```

变量与赋值过程1/2

- Python中的变量名以字母或下划线开始，大小写敏感，可以是任意长度
- 给变量取一个有意义的名字是编程的良好习惯，方便自己和他人阅读理解
- 当一个名字第一次被用于赋值符号（=）左边时，一个python变量就被创建
- 赋值过程实际上是让变量保持实际数据的一个引用，而不是持有数据本身，该思想类似于C语言中的指针概念

变量与赋值过程2/2

```
>>> theSum=0
>>> theSum
0
>>> id(theSum)
33841968L
>>> theSum=theSum+1
>>> theSum
1
>>> id(theSum)
33841944L
>>> theSum=True
>>> theSum
True
>>> id(theSum)
505967328L
```

Figure 3: Variables Hold References to Data Objects

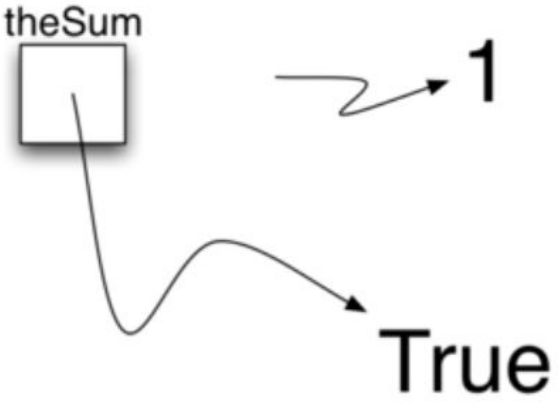


Figure 4: Assignment Changes the Reference

这是python的一个动态特征，同一个变量可以指向不同的数据类型

列表1/4

- Python的内建容器数据类型：
  - 序列容器：list，string，tuple
  - 散列容器：set，dict
- 列表list是0个或多个python数据对象引用的有序容器，用“[]”表示，元素之间用逗号分隔
- 列表是异质容器，即一个列表中可以存放不同类型的数据

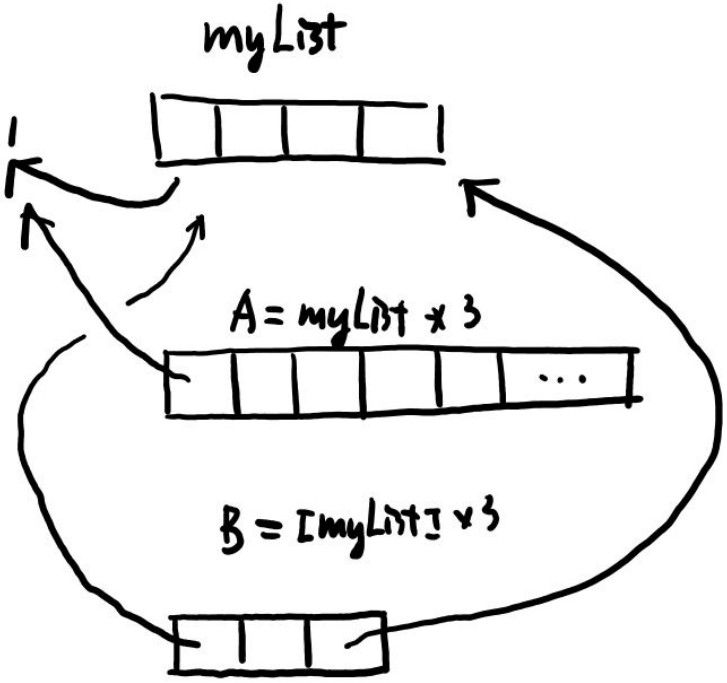
```
>>> [1, 3, True, "How"]
[1, 3, True, 'How']
>>> myList=[1, 3, True, "How"]
>>> myList
[1, 3, True, 'How']
```

列表2/4

- 列表是序列的，它支持许多操作
- 这些操作对于其他任何python系列都适用
  - []：索引
  - +：连接
  - *：重复
  - in：某一项是否在序列中
  - len()：获取序列的长度
  - [:]：切片操作

```
>>> myList=[1, 2, "How", True, [3, 4]]
>>> myList[1]
2
>>> myList[4][0]
3
>>> myList=myList+["are"]
>>> myList
[1, 2, 'How', True, [3, 4], 'are']
>>> 1 in myList
True
>>> 3 in myList
False
>>> [3, 4] in myList
True
>>> myList[1:3]
[2, 'How']
>>> myList[:3]
[1, 2, 'How']
>>> myList[5:]
['are']
>>> myList[4:-1]
[[3, 4]]
```

```
>>> myList=[0]*6 # 常用于初始化
>>> myList
[0, 0, 0, 0, 0, 0]
>>> myList=[1, 2, 3, 4]
>>> A=myList*3
>>> A
[1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]
>>> myList[0]=5
>>> myList
[5, 2, 3, 4]
>>> A
[1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]
>>> B=[myList]*3 # B持有三份相同的引用
>>> B
[[5, 2, 3, 4], [5, 2, 3, 4], [5, 2, 3, 4]]
>>> myList[0]=10
>>> myList
[10, 2, 3, 4]
>>> B
[[10, 2, 3, 4], [10, 2, 3, 4], [10, 2, 3, 4]]
```



列表3/4

- 列表本身实现了许多方法，常用于构建数据结构
- 方法的调用通过“.”实现
  - aList.append(item)：在列表末尾添加一个新项
  - aList.insert(i, item)：在列表的某个位置插入一个项
  - aList.pop()：移除并返回列表的最后一项
  - aList.pop(i)：移除并返回列表的第i项
  - aList.sort()：对列表进行排序
  - aList.reverse()：反转列表
  - aList.index(item)：返回列表中第一个等于item项的索引
  - aList.count(item)：返回列表中有多少项的值等于item
  - aList.remove(item)：删除列表中第一个值等于item的项

```
>>> myList=[1024, 3, True, 6.5]
>>> myList.append(False)
>>> myList
[1024, 3, True, 6.5, False]
>>> myList.insert(2, 4.5)
>>> myList
[1024, 3, 4.5, True, 6.5, False]
>>> myList.pop()
False
>>> myList
[1024, 3, 4.5, True, 6.5]
>>> myList.pop(3)
True
>>> myList
[1024, 3, 4.5, 6.5]
>>> myList.sort()
>>> myList
[3, 4.5, 6.5, 1024]
>>> myList.reverse()
>>> myList
[1024, 6.5, 4.5, 3]
>>> myList.index(4.5)
2
```

列表4/4

- range
- 原型：
  - range(stop)：生成[0, stop)的整数序列
  - range(start, stop[, step])：生成[start, stop)之间，步长为step的整数序列；step可以是负值，如果不设置，则默认为1

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(1, 10)
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(1, 10, 3)
[1, 4, 7]
>>> range(10, 1, -1)
[10, 9, 8, 7, 6, 5, 4, 3, 2]
```

字符串1/3

- 字符串string是0个或者多个字符的序列容器
- 使用引号包围，可以是单引号，也可以是双引号
- 适用于序列容器的操作也适用于字符串

```
>>> "David"
'David'
>>> myName="David"
>>> myName[3]
'i'
>>> myName*2
'DavidDavid'
>>> len(myName)
5
```

字符串2/3

- 字符串也有属于自己的方法
  - aStr.center(w)：返回一个字符串，w长度，原字符串居中
  - aStr.count(item)：返回原字符串中出现item的次数
  - aStr.ljust(w)：返回一个字符串，w长度，原字符串居左
  - aStr.rjust(w)：返回一个字符串，w长度，原字符串居右
  - aStr.lower()：返回一个字符串，全部小写
  - aStr.upper()：返回一个字符串，全部大写
  - aStr.find(item)：查询item，返回第一个匹配的索引位置
  - aStr.split(schar)：以schar为分隔符，将原字符串分割，返回一个列表

```
>>> myName
'David'
>>> myName.upper()
'DAVID'
>>> myName.lower()
'david'
>>> myName.center(10)
'  David   '
>>> myName.ljust(10)
'David     '
>>> myName.rjust(10)
'     David'
>>> myName.find("v")
2
>>> myName.split("v")
['Da', 'id']
>>> id(myName)
39371480L
>>> id(myName.upper())
39371280L
```

字符串3/3

- 字符串和列表的一个主要区别
  - 列表是可变的
  - 字符串是不可变的
  - 该区别导致的一个结果是调用列表的方法会改变原列表，而字符串的方法将产生一个新的字符串

```
>>> myName
'David'
>>> myName[0]="X"

Traceback (most recent call last):
  File "<pyshell#15>", line 1, in <module>
    myName[0]="X"
TypeError: 'str' object does not support item
assignment
```

元组1/1

- 元组tuple和列表list类似，也是异质数据序列容器，区别是，tuple是不可变数据类型
- 元组使用小括号括起来，元素之间使用逗号隔开
- 同样，它可以使用任何适用于序列类型的操作

```
>>> myTuple=(2, True, "How")
>>> myTuple
(2, True, 'How')
>>> len(myTuple)
3
>>> myTuple[0]
2
>>> myTuple*3
(2, True, 'How', 2, True, 'How', 2, True, 'How')
>>> myTuple[0:2]
(2, True)
>>> myTuple[0]="Wrong"
Traceback (most recent call last):
  File "<pyshell#28>", line 1, in <module>
    myTuple[0]="Wrong"
TypeError: 'tuple' object does not support item
assignment
```

集合1/3

- set是0个或多个数据的无序散列容器
- set不允许多重记录，书写方式是用大括号括起来，元素之间用逗号分隔
- set是异质的
- set是可变的

```
>>> mySet={3, 6, "cat", 4.5, False}
>>> mySet
set([False, 4.5, 3, 6, 'cat'])
>>> mySet=set()
>>> mySet
set([])
```

集合2/3

- set不是序列容器
- 集合操作
  - len()：返回集合的元素个数
  - in：判断一个元素是否在集合内
  - |：返回两个集合的并集
  - &：返回两个集合的交集
  - -：返回差集
  - <=：判断第二个集合是否为第一个集合的超集

集合3/3

- set的方法
  - union(otherset)：同“|”操作
  - intersection(otherset)：同“&”操作
  - difference(otherset)：同“-”操作
  - insubset(otherset)：同“<=”操作
  - add(item)：添加一个项
  - remove(item)：移除一个项
  - pop()：任意移除一个元素
  - clear()：清空所有元素

```
>>> mySet
set([False, 4.5, 3, 6, 'cat'])
>>> yourSet={99, 3, 100}
>>> yourSet
set([99, 3, 100])
>>> mySet.union(yourSet)
set([False, 4.5, 3, 100, 6,
'cat', 99])
>>> mySet|yourSet
set([False, 4.5, 3, 100, 6,
'cat', 99])
>>> mySet.intersection(yourSet)
set([3])
>>> mySet&yourSet
set([3])
>>> mySet.difference(yourSet)
set([False, 4.5, 6, 'cat'])
>>> mySet-yourSet
set([False, 4.5, 6, 'cat'])
```

```
>>> {99, 3}.issubset(yourSet)
True
>>> {99, 3}<=yourSet
True
>>> mySet.add("How")
>>> mySet
set([False, 4.5, 3, 6, 'cat',
'How'])
>>> mySet.remove(4.5)
>>> mySet
set([False, 3, 6, 'cat',
'How'])
>>> mySet.clear()
>>> mySet
set([])
```

字典1

- 字典dict是key-value键值对的容器，key在字典中必须唯一
- 字典用大括号括起来，每对元素为key:value，元素之间用逗号分隔
- 可以通过键来访问字典中的值，也可以添加一个新的键值对

```
>>> myDict={1:"How", "name":"JC"}
>>> myDict
{1: 'How', 'name': 'JC'}
>>> myDict=dict()
>>> myDict
{}
```

```
>>> myDict={"A":1, "B":"name", 10:4.5}
>>> myDict # 注意，是无序的
{'A': 1, 10: 4.5, 'B': 'name'}
>>> myDict["A"]
1
>>> myDict[10]
4.5
>>> myDict["C"]=False
>>> myDict
{'A': 1, 10: 4.5, 'B': 'name', 'C': False}
>>> myDict["A"]="How"
>>> myDict
{'A': 'How', 10: 4.5, 'B': 'name', 'C': False}
>>> len(myDict)
4
>>> "A" in myDict
True
```

字典2

- 字典方法
  - keys()：返回一个list，包含所有键
  - values()：返回一个list，包含所有值
  - items()：返回一个list，包含所有键值对
  - get(k)：返回键为k的值，如果不存在，返回None
  - get(k, alt)：返回键为k的值，如果不存在，返回alt

```
>>> myDict
{'A': 'How', 10: 4.5, 'B': 'name', 'C': False}
>>> myDict.keys()
['A', 10, 'B', 'C']
>>> myDict.values()
['How', 4.5, 'name', False]
>>> myDict.items()
[('A', 'How'), (10, 4.5), ('B', 'name'), ('C',
False)]
>>> myDict.get("A")
'How'
>>> myDict.get("X")
>>> myDict.get("X", "NoValue")
'NoValue'
```

2 输入输出

输入输出

- 输入
  - input()
- 输出
  - print()
- 格式化字符串
- 文件操作
  - file()

输入1/1

- 在程序运行过程中需要与用户进行交互以获取数据或提供结果
- Python没有创建对话框的方法，而是有更加简单的函数可供调用
- 输入函数input()
  - 原型：input([prompt])
  - 可以接收一个字符串为参数作为提示语句
  - 该函数对用于的输入内容进行解析，并返回一个解析结果
- 如果不需要进行解析，则可以使用raw_input()

```
>>> name=input("Please enter your name: ")
Please enter your name: JC

Traceback (most recent call last):
  File "<pyshell#80>", line 1, in <module>
    name=input("Please enter your name: ")
  File "<string>", line 1, in <module>
NameError: name 'JC' is not defined
>>> name=input("Please enter your name: ")
Please enter your name: "JC"
>>> name
'JC'
>>> type(name)
<type 'str'>
>>> age=input("Please enter your age: ")
Please enter your age: 26
>>> age
26
>>> type(age)
<type 'int'>
```

```
>>> name=raw_input("Please enter your name: ")
Please enter your name: JC
>>> name
'JC'
>>> type(name)
<type 'str'>
>>> age=raw_input("Please enter your age: ")
Please enter your age: 26
>>> age
'26'
>>> type(age)
<type 'str'>
>>> age=int(raw_input("Please enter your age: "))
Please enter your age: 26
>>> age
26
>>> type(age)
<type 'int'>
```

输出1/1

- 输出函数print()
  - 原型：print(*object, sep= ‘ ’ , end= ‘\n’ , file=sys.stdout)
  - object表示输出对象，可以是一个或多个值，之间用逗号分隔
  - sep表示各个输出对象之间的分隔符，默认为空格
  - end表示输出的结束符，默认为换行符
  - file表示输出流，默认为stdout，即标准输出流
  - Python2中的print功能比较简单，如果要在Python2.7中使用上述print()函数，需要加入语句
    - from __future__ import print_function

```
>>> from __future__ import print_function
>>> print("Hello")
Hello
>>> print("Hello", "World")
Hello World
>>> print("Hello", "World", sep="***") #需要指明“sep”
Hello***World
>>> print("Hello", "World", end="***")
Hello World***
```

格式化字符串1/1

- 指在用一个模板来指明一个字符串，其中的单词和空格等会保持原样，而占位符则可以留给变量来填充
- 格式化字符串的书写格式
  - 字符串模板 + % +元组/字典

```
>>> "My name is %s, and I'm %d years old" % ("JC", 26)
"My name is JC, and I'm 26 years old"
>>> "The %(item)s costs %(price)f yuan" % {"item":
"apple", "price":2.5}
'The apple costs 2.500000 yuan'
```

格式化字符串2/2

- 在字符操作中，%称为格式操作符
- 转换字符指明了数据类型

符号	类型
d/i	整型
u	无符号整型
f	浮点型，m.ddddd
e	浮点型，m.dddddde+/-XX
E	浮点型，m.dddddE+/-XX
g	指数比-4小或比5大时使用%e，否则使用%f
c	单字符
s	字符串或者能通过str()转为字符串的数据
%	输入一个%

格式化字符串3/3

- 在%和转换字符之间可以添加格式修饰符，用于指定宽度或者指定小数位数
- 如果右侧是元组，则按顺序填入，如果是字典，则每一个占位符必须提供name修饰符

修饰符类型	说明
num	是该值占据num个字符宽度
-num	num个字符宽度，左对齐
+num	num个字符宽度，右对齐
0num	num个字符宽度，前置“0”
.num	保留num位小数
(name)	从字典中取key为name的值放在该处

```
>>> price=24
>>> item="banana"
>>> print("The %s costs %d cents" % (item, price))
The banana costs 24 cents
>>> print("The %+10s costs %5.2f cents" % (item, price))
The     banana costs 24.00 cents
>>> print("The %+10s costs %10.2f cents"%(item,price))
The     banana costs      24.00 cents
>>> itemdict = {"item":"banana", "cost":24}
>>> print("The %(item)s costs %(cost)7.1f cents" % itemdict)
The banana costs    24.0 cents
```

文件操作1/2

- 程序中常常需要进行文件的读取和写入
- file()
  - 原型：file(name[, mode])
  - name: 文件名
  - mode: 访问方式，默认为只读模式
    - “r”：读方式打开
    - “w”：写方式打开（必要时清空）
    - “a”：以追加方式打开
    - “r+”/“w+”/“a+”：以读写方式打开，有区别
    - 在后面追加“b”，则表示以二进制模式打开
  - 返回一个file类型，用于对文件的访问

文件操作2/2

- file对象方法
  - close()：关闭文件，在文件读取完成之后记得调用
  - read(size=-1)：读取size个字节，默认读取剩下所有字节
  - readline(size=-1)：读取一行，或返回最大size个字符
  - readlines()：读取所有行并以列表形式返回
  - write(str)：向文件写入字符串str
  - writelines(seq)：向文件写入字符串序列seq

```
>>> f=file("test.txt", "w")
>>> type(f)
<type 'file'>
>>> f.write("function write\n")
>>> strList=["Hello\n", "World\n"]
>>> f.writelines(strList)
>>> f.close()
```

```
function write
Hello
World
```

```
>>> f=file("test.txt", "r")
>>> f.readline()
'function write\n'
>>> strList=f.readlines()
>>> strList
['Hello\n', 'World\n']
```

3 控制结构：条件与循环

控制结构：条件与循环

- 条件结构
- 循环结构
  - while循环结构
  - for循环结构
- 列表解析

条件结构1/2

- 单if语句：

```
if condition:
    …
```

- if-else语句：

```
if condition:
    …
else:
    …
```

- 使用elif：

```
if condition1:
    …
elif condition2:
    …
elif condition3:
    …
else:
    …
```

```
if score >= 90:
    print('A')
else:
    if score >=80:
        print('B')
    else:
        if score >= 70:
            print('C')
        else:
            if score >= 60:
                print('D')
            else:
                print('F')
```

```
if score >= 90:
    print('A')
elif score >=80:
    print('B')
elif score >= 70:
    print('C')
elif score >= 60:
    print('D')
else:
    print('F')
```

条件结构2/2

- Python中没有switch-case结构
- Python中的三元操作符
  - •X if C else Y

    等价于
  - •C/C++语言中的 C ? X : Y
  - •C条件满足时取X值，否则取Y值

```
>>> x=3 if True else 4
>>> x
3
>>> x=3 if (4<=3) else 4
>>> x
4
```

循环结构1/2

- 循环结构包括while循环和for循环
- while语句，当条件为真时执行
- while语句结构如下：

```
while condition:
    …
```

- break和continue的用法同C语言

```
>>> counter=1
>>> while counter<=5:
      print("Hello, Wolrd")
      counter+=1


Hello, Wolrd
Hello, Wolrd
Hello, Wolrd
Hello, Wolrd
Hello, Wolrd
```

循环结构2/2

- for语句用于遍历一个可迭代对象的成员，如列表、字符串等
- for语句结构如下：

```
for item in container:
    …
```

item每一次指向container中的一个元素

- for常与range连用，以遍历range产生的每一个整数

```
>>> for item in range(5):
        print(item*2)
0
2
4
6
8


>>> wordlist = ['cat','dog','rabbit']
>>> letterlist = []
>>> for aword in wordlist:
        for aletter in aword:
                letterlist.append(aletter)
>>> letterlist
['c', 'a', 't', 'd', 'o', 'g', 'r', 'a',
'b', 'b', 'i', 't']
```

列表解析1

- 列表解析是指基于某些处理或选择标准来方便地构建列表
- 结构：
  - [表达式 + for语句]
  - [表达式 + for语句 + if语句]
  - 将满足条件的值用于计算表达式后得到的值加入到列表中

```
>>> sqlist=[]
>>> for x in range(1, 11):
        sqlist.append(x*x)

>>> sqlist
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
>>> sqlist=[x*x for x in range(1,11)]
>>> sqlist
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
>>> sqlist=[x*x for x in range(1,11) if x%2!=0]
>>> sqlist
[1, 9, 25, 49, 81]
>>> [ch.upper() for ch in "jiang cheng" if ch not in "Monday"]
['J', 'I', 'G', ' ', 'C', 'H', 'E', 'G']
```

4 异常处理

异常处理1/4

- Syntax error：语法错误，表示结构或者语法表达错误，程序不能执行
- Syntax error在Python学习的初期会经常出现

- Logic error：逻辑错误，程序能够执行，但得到错误的结果
  - 除0错误
  - 越界访问错误
  - …
- Logic error导致运行时错误，使得程序终止
- 这些运行时错误称为异常

```
>>> 8/0

Traceback (most recent call last):
  File "<pyshell#58>", line 1, in <module>
    8/0
ZeroDivisionError: integer division or modulo by zero
>>> l=[1, 2]
>>> l[3]

Traceback (most recent call last):
  File "<pyshell#60>", line 1, in <module>
    l[3]
IndexError: list index out of range
```

异常处理2/4

- 使用try-except语句，我们可以对异常进行处理
  - 忽略
  - 处理
  - 停止程序运行
  - …
- 语句结构：

```
try:
    # 可能产生异常的语句
except:
    # 当异常发生时执行的语句
    # 如果异常没有发生，则该段语句将不会执行
```

```
>>> anumber=-3.4
>>>
try:
      # sqrt函数可能抛出异常
      print(math.sqrt(anumber))
except:# except语句捕获异常并处理
      print("Bad Value for square root")
      print("Using absolute value instead")
      print(math.sqrt(abs(anumber)))


Bad Value for square root
Using absolute value instead
1.84390889146
```

异常处理3/4

- 用户可以使用raise语句自己抛出异常

```
>>> anumber=-3
>>>
if anumber<0:
        raise RuntimeError("You cannot use a negative number")
else:
        print(math.sqrt(anumber))

Traceback (most recent call last):
  File "<pyshell#76>", line 2, in <module>
    raise RuntimeError("You cannot use a negative number")
RuntimeError: You cannot use a negative number
```

异常处理4/4

- 断言语句assert
  - 断言语句等价于这样的Python表达式，如果断言成功，则不采取任何措施，否则抛出AssertionError异常
  - AssertionError异常与其他异常一样可以使用try-except捕获，如果没有捕获，则将终止程序
  - assert(expr, info)
    - expr是要进行断言的表达式，表达式结果为False时，触发异常
    - info表示触发异常时附带的信息
- 这里所介绍的异常处理很初步，有兴趣的同学可以课后深入了解

```
>>> assert(1==1)
>>> assert(2+2>=2*2)
>>> assert(range(1, 3)==[1, 2, 3])

Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    assert(range(1, 3)==[1, 2, 3])
AssertionError
>>>
try:
      assert(8/2>5)
except:
      print("Wrong")

Wrong
```

5 函数定义

函数定义1/2

- 通过定义一个函数，我们可以隐藏实现细节，重用代码，并且使得代码功能划分更加清晰
- Python的函数定义需要“def”关键字，一个函数名，一组参数，一个函数体，并可能显式返回值

```
def function(para1, para2, …):
    …
    return …
```

- 一旦return，则跳出该函数体，后面的语句将不再执行
- Python的函数在不同的语句分支可以返回不同类型的数据，也可以在一个语句返回多个数据，此时默认以元组的形式返回

```
>>> def square(n):
      return n**2
>>> square(3)
9
>>> square(square(3))
81

>>> def returnValue():
      return 1, 2 #等同于return (1, 2)
>>> returnValue()
(1, 2)
```

函数定义2/2

- Python的函数参数形式灵活多样，有兴趣的同学可以课后自行了解“位置参数”、“默认参数”、“关键字参数”、“可变长度参数”等内容。