# 《Python编程 从入门到实践》读书笔记(一)

### **1、字符串**

****1.1 大小写转换****

* title()：每个单词的首字母大写
* lower()：全部转换为小写
* upper()：全部转换为大写

****1.2 删除空白****

* strip()：删除首位空白
* lstrip()：删除字符串开头空白
* rstrip()：删除字符串结尾空白

****1.3 引号****  
单引号中可以包含双引号，双引号中可以包含单引号，但是同种引号不能相互包含，因为会被解释器认为是字符串的结尾。  
三个双引号，python中允许使用三个双引号分别作为字符串的开头和结尾，这个就比较强悍了，一对三个双引号中间的部分将原样输出，原样输出的意思是指制表符换行符等也会在输出结果中表现出来，例如：

s = """hello"world # this is a string"""print(s)

上面这行代码的输出结果是酱紫的：

# 这里会有一个换行

hello"

world # this is a string# 这里还会有一个换行

在python中使用三个双引号声明字符串时，字符串中的所有内容都会如实输出，包括各种空白，而且会真实的体现在输出结果中。个人认为可以用在格式化排版输出中，初学，不知道三个双引号会不会有性能问题。

### **2、数字**

****2.1 乘方****  
在python中，乘方是酱紫来表示的：

>>> 3 \*\* 2>>> 9

>>> 3 \*\* 3>>> 27

****2.2 精度问题****  
和javascript一样，python中也存在这么一个问题：

>>> 0.1 + 0.2>>> 0.30000000000000004

所以，同在js中一样，最好不采用下面的方式判断：

if(0.3 == 0.1 + 0.2) {

......

}

上面的计算中，虽然0.1 ＋ 0.2应该是0.3，但是由于精度的问题，上面的等式不成立，所以应避免这种判断方式。

****2.3 与字符串一起计算****  
python中可以使用“＋”来连接字符串，在js中数字可与字符串直接“加”操作来连接，但是在python中这种操作会出现错误，因为python不会对数字进行类型转换，所以会引发类型错误。将数字转换为字符串可以使用str()方法，该方法会将非字符串转换为字符串，用法如下：

age = 5

s = 'your age is:' + str(age)print(s)

****2.4 python2中整数除法运算****  
在python2中，3/2的结果为1而不是1.5，小数部分会被直接丢掉而不是四舍五入，所以在使用时要多加小心。

### **3、列表（数组?）**

****3.1 索引****

python列表中一个有意思的地方是：可以使用－1表示列表中最后一个元素的索引，－2表示列表中倒数第二个元素的索引，依此类推。

python中的列表有一个有意思的规律：所有会改变列表自身结构或者顺序的方法都是在列表自身上调用的，所有只是得到某些信息或者说不会改变列表自身的方法都是将列表作为参数适用的。

****3.2 常用方法****

* append(val)：在列表末尾添加一个值；
* insert(index, val)：在指定位置插入一个值，如果指定位置超出列表长度则添加在列表的最后
* del arr[0]：删除指定位置的元素，如果索引超出列表的范围则会引发错误
* pop(index)：删除任意位置的元素并将其返回，不指定index则删除列表的最后一个值并返回，索引不对同样会引起错误
* remove(val)：根据值来删除列表中的元素，但是只会删除一次，如果有多个重复值则需要多次调用该方法

****3.3 排序方法****

* arr.sort()：该方法会永久改变列表顺序，即将内存中的列表顺序改变，反向排序可向该方法中传递reverse=True
* sorted(arr, reverse=True)：该方法用于临时排序列表，不会改变原来列表的顺序，返回排序后的列表，反向排序同上
* arr.reverse()：方法会将列表顺序反转
* len(arr)：返回列表长度

### **4、生成数值列表**

****4.1 range()方法****  
该方法可以生成指定范围内的一系列数值，并且可以指定步长，例如：

for value in range(1, 10, 2):

print(value);

结果：1, 3, 5, 7, 9

该方法并不会将生成的一系列数值作为列表返回，如果在终端中直接调用range(1,5)则python解析器会原样放回，所以如果要将range()方法生成的一系列数值转换成列表，可以使用list()方法，该方法接收一个参数，并将其转换为列表，该参数必须为可以迭代的对象，否则python解析器会报错。

****4.2 列表解析****  
如果我们要生成一个数值列表，该列表的值是1到4的2次方，我们可以使用下面的方法：

var squares = []for v in range(1,5):

square = v \*\* 2

squares.append(square)

结果：[1, 4, 9, 16]

有没有更优雅的方式呢？有，那就是列表解析了，直接上代码：

squares = [value \*\* 2 for value in range(1, 5)]

结果：[1, 4, 9, 16]

****4.3 简单数值列表统计****

* min(arr)：取最小值
* max(arr)：取最大值
* sum(arr)：求和

### **5、切片**

所谓的切片就是使用列表的一部分，即列表的子集，切片的使用方法很简单，例如我们有一个列表arr = [1, 2, 3, 4, 5]，我们要取其中的第二个到第四个元素，可以用下面这种方法：

arr[1 : 4]

结果：[2, 3, 4]

切片还有很多花式用法，例如：

arr[:4] # 取第一个到第四个元素

arr[1:] # 取第二个到最后一个元素（索引从0开始，0表示第一个元素，所以是第二个到最后一个）

arr[:] # 取第一个到最后一个元素

arr[-3:] # 取倒数第三个到最后一个（还记得负数索引吗？）

### **6、元组**

我们使用[]和list()方法生成的列表的元素都是可以修改的，但是元组的元素是不可修改的，即其元素的值是不能修改的，有点像c语言中的常量，元组的声明像下面这样：

arr = (1, 3, 5);

元组也是可以迭代的，像下面这样：

for v in arr:

print(v)

结果：1 3 5

元组变量也是可以修改的，这里说的元祖变量是指指向元组的那个变量，就是这里的arr了：

arr = (1, 3, 5)print(arr)> 结果：1 3 5

arr = (2, 4, 6)print(arr)> 结果：2 4 6

但是元组的每个元素都不能重新赋值，否则python解释器会报错。元组支持切片，但是没有append(), pop()等方法，在元组上调用这些方法python解释器会返回错误。

### **7、in和not in关键字**

in关键字可以用来判断一个值是否在列表中存在，存在返回True，否则返回false，例：

arr = [1, 3, 5,7]1 in arr

结果：True

not in关键字用来判断一个值是否不再列表中，如果不在列表中返回True，否则返回false。

# 《Python编程 从入门到实践》读书笔记(二)

### **1、python中的与或**

python表达式中的与或不使用符号“&&”和“||”来表示，而是使用“and”和“or”来表示，例：

if 3 < 4 and 5 < 6:

print('True')if 3 < 4 or 5 < 6:

print('True')

### **2、if语句**

****2.1 if-elif-else****  
与javascript中的else if不同，python中的else if写作elif，使用方法如下：

age = 18;if age < 18:

print('age < 18')elif age == 18:

print('age == 18')else:

print('age > 18')

结果：'age == 18'

****2.2 判断****  
与js中的if判断语句一样，当if后的变量的值为空的字符串、空的列表、0等值时，if语句的判断结果是False。例如：

arr = [];if arr:

print('True')else:

print('False')

结果：False

### **3、python2和python3中range()方法的区别**

在python2中，当我们调用range()这个 方法的时候，返回的是一个列表，例如：

>>> range(5)>>> [1, 2, 3,4]>>> xrange(5)>>> xrange(5)

而xrange()方法返回的不是数组，而是一个生成器，所以当我们需要构造一个大范围的数据范围时，xrange()方法的效率要比range()的效率高很多，因为range()方法的实质就是迭代，而且会在内存中真实的构造出一个包涵整个范围的列表，所以效率和性能都很差。所以在python2中在构建大范围的数据范围用于for循环的时候建议使用xrange()方法。

而在python3.x的版本中删除了xrange()方法，当我们调用range()方法时返回值如下：

>>> range(5)>>> range(5)

可见python3.x的版本对range()方法进行了优化，返回的不再是列表而是构造器。

### **4、字典**

****4.1 与js中对象的区别****  
基本上跟js中的对象差不多，不过也有不同的地方，主要体现在下面几点：

* 字典中的键必须是字符串的形式声明d = {'name':'d'}
* 不能使用.来获取字典中对应键的值，只能使用方括号的形式
* 删除字典中的键值对使用del关键字del d[name]

****4.2 遍历字典****

4.2.1 items()方法  
字典的items方法会返回字典每个键值对为一个元素组成的列表，例如在python3.x中：

>>> d = {'name':'d', 'age':18}>>> d.items() # 结果为dict\_items([('age',18), ('name', 'd')])

>>> for k, v in d.items():>>> print(k+str(v));

这里的k和v分别代表字典中每一个键值对中的键和值，如果需要同时使用字典中的键和值时可以使用items()方法。

4.2.2 keys()方法  
字典的keys()方法会返回字典的所有键为元素组成的列表，在python3中：

>>> d = {'name':'d', 'age': 18}>>> d.keys() # 结果为dict\_keys(['name', 'age'])

>>> for k in d.keys():>>> print(k)

4.2.3 values()方法  
字典的values()方法会返回字典的所有值为元素组成的列表，用法同keys()方法，这里不再赘述。

4.2.4 直接使用for in遍历  
对于字典的遍历可以直接使用for in方法，例如:

>>> d = {'name':'d', 'age': 18}>>> for k in d:>>> print(k)

python中的for in与javascript中的用法相同，上述代码中的k表示每个键值对中的键，可以使用键和字典名字获取对应的值。

4.2.5 其他  
在对字典进行遍历对时候键值对被遍历对顺序和其在字典中的顺序可能不是一一对应的，因为python关心的是键值对的关系而不是键值对的排列顺序，在使用for in遍历字典的时候可以通过下面的方法对键进行排序：

>>> for k in sorted(d.keys()):>>> some options...

### **4.3 嵌套**

python中的嵌套包括：列表中嵌套字典，字典中嵌套列表和字典中嵌套字典，这个比较普通，不再赘述。列表中的列表应该不叫嵌套，在python中应该称为多维列表。

**三种区别：**

**列表：[ ] 元祖：( ) 字典：{ }**

# 《Python编程 从入门到实践》读书笔记(四)

### **1、输入**

python3中获取终端输入内容的方法是input()方法，该方法接收一个参数，用于在终端输出提示信息，返回终端输入的内容，终端输入的所有内容都会被当成是字符串返回。可以通过下面的方法获取终端的输入并存储在变量中：

message = input('please input something')

在python2.x中，获取用户输入的方法为raw\_input()，该方法的使用方式与python3.x中的input的使用方式相同。而在python2.x中也存在一个input方法，该方法的作用类似于js中的eval()方法，获取终端的输入后会尝试将终端输入的代码当作python代码来运行，这个地方在不同版本的python中应注意区别对待。

### **2、函数**

****2.1 定义****  
python中的定义函数使用关键字def，例：

def foo(name, age):

print('name: ' + name)

print('age: ' + str(age))

****2.2 位置实参和关键字实参****  
在传递实参时，我们可以使用位置实参或者关键字实参，所谓的位置实参就是传递参数的顺序要和函数定义中形参的顺序一致，否则形参接收到的实参可能不是预期的实参了。例如上面例子定义的foo()函数，第一个形参是name，第二个形参是age，如果我们使用位置实参foo('Tom', 18)，则‘Tom’会传递给name，18会传递给age，这和预期的一样，而如果我们调用时foo(18, 'Tom')，则18会传递给name，‘Tom’会传递给age，和预期不一样。所以：****位置实参对实参的顺序有严格要求，顺序不对可能会引起不必要的错误。****

关键字实参是指在调用函数传递参数的时候带上形参的名字作为关键字，例如foo(name="Tom", age=18)，关键字实参对传递参数的顺序没有要求，因为python会根据关键字将对应的值传递给对应的形参，前提是关键字没有写错。

****2.3 默认值****  
python中的函数支持参数默认值，即可以给一个或多个形参指定默认值，当函数被调用时如果给该形参传递了实参则使用传递的实参，否则使用该形参的默认值。

python中规定有默认值的形参必须在形参列表的最后声明，否则python解释器会报错。我们可以借用默认值形参来指定可选的形参，我们可以给可选形参一个空字符串作为默认值，然后在函数内部使用if语句判断该形参的Boolean值，如果if语句中为True，说明肯定传递了对应的实参；如果为false，说明没有传递该可选形参。

def foo(name, age, nickname=''):

"""有昵称则打印昵称，否则打印名字"""

if nickname:

print(nickname)

else:

print(name)

****2.4 值传递和地址传递****  
保存基本数据类型的变量在作为参数传递给函数时，是值传递；而像列表、字典作为实参传递给函数时，是地址传递，即如果在函数中对地址传递的参数进行修改时，会永久修改该参数的值，所以在地址传递时应注意是要修改原来的数据还是修改副本。

对于列表，可使用切片的方式获取列表的副本，避免修改原始数据。但是在操作大范围的数据时，应避免使用切片制作副本的方式，因为使用切片制作副本会占用内存和花费时间来生成新的列表。数据特别大时会消耗更多的内存和时间，效率很低。

### **3、模块**

为了方便管理和使用方法，我们可以将具有相似功能的一组方法放到一个模块(module)中，然后在使用的地方通过import关键字将需要的整个模块或者模块中的部分函数导入。

module.py：

def foo():

print('This is a module function.')

main.py:

import module

def main():

module.foo()

导入模块的几种方法：

* import module\_name：这种方法会将整个模块都导入，调用时module\_name.foo()，module\_name不能省略
* import module\_name as m：给模块取个别名，调用时方便：m.foo()
* from module\_name import fun\_1, fun\_2：导入模块中指定的方法，调用时可直接通过方法名字调用，不用关心模块名字：fun\_1()
* from module\_name import fun\_1 as f1：给方法指定方法指定别名，可通过别名直接调用：f1()
* from module\_name import \*：可将模块中的所有方法导入，不过不推荐这么做，这样有可能会将引入模块的文件中同名的方法覆盖掉

### **4、类**

****4.1 定义方法****  
定义每个方法都需要设置第一个参数self，python在运行时会自动传入一个self，如果定义的方法不包含self这个形参，在调用时python会报错。

class Dong():

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def sit(self):

print(self.name + ' is sitting')

****4.2 继承****

python3:

class Car():

def \_\_init\_\_(self, name):

self.name = name

class Tesla(Car):

def \_\_init\_\_(self, name, model):

# 继承父类的全部属性和方法并初始化

super().\_\_init\_\_(name)

self.model = model

python2:

class Car(object):

def \_\_init\_\_(self, name):

self.name = name

class Tesla(Car):

def \_\_init\_\_(self, name, model):

# 继承父类的全部属性和方法并初始化

super(Tesla, self).\_\_init\_\_(name)

self.model = model

python2中继承时需要注意：函数super()需要两个实参：子类的名字和对象self，这两个实参必不可少；同时，务必在父类定义的括号中指定object。

****4.3 导入****  
类的导入同函数的导入，同样支持模块别名和类别名，使用方法和函数导入的使用方法相同，详见本文的第三条。

# 《Python编程 从入门到实践》读书笔记(四)

### **1、文件读取**

with open('xx.txt') as file\_object:

contents = file\_object.read()

上面的代码中，with的作用是在python确定不再使用文件时自动调用close()方法关闭文件；open()方法的作用是打开指定的文件，前提是文件的路径是正确的；file\_object为读取到到文件对象，该对象有很多方法，例如read()方法和readlines()方法。

file\_object.read()方法会将整个文件读取为一个大字符串；  
file\_object.readlines()方法会返回一个以行为单位生成的列表，每行的内容为列表的一个元素

### **2、文件写入**

with open('xx.txt', 'w') as file\_object:

file\_object.write('file write')

在打开文件时可以指定打开文件的方式：

* w：写入文件，该模式会在打开时清空文件
* r：读取文件
* a：附加写入，该模式会在文件末位附加写入的内容
* r+：读取和写入文件都可以  
  write()方法并不会在写入行都末尾添加换行符，所以如果想要换行的效果，可在需要的位置添加“\n”，其他空白符同样可以添加到字符串中，在写入到文件中时会体现出来。

### **3、异常**

捕获异常的实例代码：

try:

result = int(num1) / int(num2)except ZeroDivisionError:

print('can not division by zero')except UnicodeDecodeError:

passelse:

print(result)

将可能会有异常的代码放在try块中，发生指定异常的处理放在except代码块中，try块中代码正常运行后要执行的代码放在else块中。

python提供了一个pass关键字，用在except块中，在指定异常被抛出时什么都不做，在实际产品开发中可以使用pass隐藏异常。

另，可以通过多个except来捕获多个异常，从而分别作出不同的处理。

### **4、测试**

python自带测试模块和测试类，我们可以通过测试模块提供的类和方法对自己写的函数和类中的方法进行单元测试并编写测试用例。

****4.1 简单的测试用例****

# 必须导入unittest模块import unittestfrom get\_formated\_name import get\_formated\_name as gfn

# 该类必须继承unittest.TestCase类class NameTestCase(unittest.TestCase):

"""name test case"""

# 单元测试方法的名字必须以test开头

def test\_first\_last\_name(self):

formated\_name = gfn('Tom', 'Bat')

self.assertEqual(formated\_name, 'Tom Bat')

def test\_first\_last\_middle\_name(self):

formated\_name = gfn('tom', 'cat', 'dog')

# 调用测试类中的断言方法

self.assertEqual(formated\_name, 'Tom Dog Cat')

# 代码行unittest.main()让Python运行这个文件中的测试

unittest.main()

****4.2 断言方法****  
python中的测试类unittest.TestCase提供了很多断言方法：

| 方法 | 用途 |
| --- | --- |
| assertEqual(a, b) | 核实a == b |
| assertNotEqual(a, b) | 核实a != b |
| assertTrue(a) | 核实a为True |
| assertFalse(a) | 核实a为False |
| assertIn(item, list) | 核实item在list中 |
| assertNotIn(item, list) | 核实item不在list中 |

****4.3 setUp()方法****  
有时候我们测试类中的方法的时候，需要在每个测试方法中都实例化类以获取类的实例来调用其中的方法，每次都实例化会有很多重复代码，所以unittest模块提供了setUp()方法,可以在该方法中给当前的继承unittest.TestCase类的子类添加属性，添加的属性可以在该测试类中的所有方法中访问到，避免重复实例化类。示例：  
survey.py:

class Survey():

"""class Survey"""

def \_\_init\_\_(self, question):

self.question = question

self.responses = []

def show\_question(self):

print(self.question)

def store\_response(self, response):

self.responses.append(response)

def show\_resposes(self):

print('The following is the responses:')

for r in self.repsonses:

print('-' + r)

test\_survey.py:

import unittestfrom survey import Survey as s

class SurverTestCase(unittest.TestCase):

"""survey test case"""

def setUp(self):

self.question = "what's your favorite language:"

self.survey = s(self.question)

def test\_store\_single\_response(self):

self.survey.store\_response('C')

self.assertIn('C', self.survey.responses)

def test\_store\_three\_responses(self):

responses = ['C', 'C#', 'PHP']

for r in responses:

self.survey.store\_response(r)

for r in responses:

self.assertIn(r, self.survey.responses)

unittest.main()