

目 录

一 Python3.....	1
第 1 章 基本数据类型.....	1
1.1 数值类型.....	1
1.1.1 整数类型.....	1
1.1.2 浮点数类型.....	2
1.1.3 复数类型.....	4
1.2 字符串类型.....	5
1.2.1 字符串的表示.....	5
1.2.2 字符串的格式化.....	7
1.2.3 字符串操作.....	10
1.3 列表.....	10
1.3.1 列表的定义.....	10
1.3.2 列表的表示.....	10
1.3.3 列表的基本操作.....	11
1.3.4 列表的高阶操作.....	11
1.4 元组.....	12
1.4.1 元组的定义.....	12
1.4.2 元组的表示.....	12
1.5 集合.....	13
1.5.1 集合的定义.....	13
1.5.2 集合的操作.....	13
1.6 字典.....	14
1.6.1 字典的定义.....	14
1.6.2 字典的表示.....	14
1.6.3 字典的高级操作.....	15
第 2 章 控制与循环.....	16
2.1 if 语句.....	16
2.1.1 If-elif-else 结构.....	16
2.1.2 Python 中“switch-case”的实现.....	17
2.2 循环语句.....	18
2.2.1 for 循环与 while 循环.....	18
第 3 章 列表推导式.....	18
第 4 章 函数.....	19
4.1 函数的结构.....	19
4.1.1 函数的定义.....	19
4.2 函数的参数.....	20
4.2.1 传参的几种方式.....	20
4.2.2 传递任意数量或类型的参数.....	21
4.3 匿名函数.....	22
4.4 高阶函数.....	23
4.4.1 排序函数.....	23
4.4.2 map 函数.....	25

4.4.3 reduce 函数.....	26
4.4.4 filter 函数.....	26
第 5 章 时间和日期.....	27
5.1 datetime 模块.....	27
5.1.1 datedelta 类型.....	27
5.1.2 date 类型.....	28
5.1.3 datetime 类型.....	28
5.1.4 datetime 的应用.....	28
5.2 将字符串-日期的转化.....	29
5.2.1 datetime 对象转化为想要字符串.....	29
5.2.2 将字符串转化为 datetime 对象.....	30
第 6 章 多线程.....	31
6.1 关于 Python 进程.....	31
6.1.1 全局解释器锁.....	31
6.1.2 启动和执行 Python 线程.....	31
6.1.3 判断进程是否启动.....	32
二 Mysql.....	32
第 1 章 表连接.....	34
1.1 内连接.....	34
1.2 外连接.....	35
1.2.1 左外连接.....	35
1.2.2 右外连接.....	36
1.2.3 全连接.....	37
1.3 交叉连接.....	38
第 2 章 聚合函数.....	38
2.1 COUNT()函数.....	38
2.2 AVG()函数.....	39
2.3 SUM()函数.....	39
2.4 MAX()函数与 MIN()函数.....	39
第 3 章 子查询.....	40
3.1 带有 IN 谓词的子查询.....	40
3.2 带有比较运算符的子查询.....	41
3.3 带有 ANY(SOME)或 ALL 谓词的子查询.....	41
3.4 带有 EXISTS 谓词的子查查询.....	42

第 1 章 基本数据类型

1.1 数值类型

1.1.1 整数类型

Python3 中，整数类型为 `int`，无长整型和短整型之分。整数默认情况下为十进制整数，但也有二进制、八进制和十六进制。

(1) 进制转化

二进制前缀为 `0b` 或 `0B`，可用内建函数 `bin()` 转化；八进制前缀为 `0o` 或 `0O`，可由函数 `oct()` 转化；十六进制前缀为 `0x` 或 `0X`，可由 `hex()` 函数转化。

```
Python 3.6.5 (default, Apr  4 2018, 23:33:46)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> x = 123
>>> x
123
>>> bin(x)
'0b1111011'
>>> oct(x)
'0o173'
>>> hex(x)
'0x7b'
```

若不希望有 `0x`、`0b`、`0o` 等前缀，可使用 `format()` 函数。

```
>>> x
123
>>> format(x, 'b' )
'1111011'
>>> format(x, 'x')
'7b'
>>> format(-x, 'o')
'-173'
>>> |
```

注意：若要将其他字符串形式的整数转化为整数，可使用 `int` 函数在配合相应的进制数（若不带进制数则默认转化为十进制整数）。

```
>>> c = '101110'
>>> int(c)
101110
>>> int(c, 16)
1052944
>>> int(c, 8)
33352
>>> int(c, 2)
46
```

(2) 整数运算

Python 中可对整数执行加(+)减(-)乘(*)除(/)运算, `P` 传参的几种方式
python 中两个乘号表示乘方运算。在终端中 Python 根据运算顺序直接返回表达式运

算结果。

```
>>> 2 + 3
5
>>> 3 - 5
-2
>>> 8 * 7
56
>>> 9 / 5
1.8
>>> 2 **32
4294967296
>>> 2 + 8 / 4
4.0
>>> (2 + 8) / 4
2.5
>>> type(2 + 8 / 4)
<class 'float'>
>>> 10 // 3
3
```

总结：1.python 中一旦设计到整除运算，结果自动转为浮点类型。2.python 中除号 '/' 与 C 语言中做整除运算不同，他直接得到一个浮点数，若欲求整除运算，可使用 "//" 来操作。

1.1.2 浮点数类型

Python 中将带小数的数字称为浮点数。浮点数的小数点可出现在数字的任何位置。

(1) 浮点数的运算

浮点数的运算与整数相似，可以进行加减乘除甚至取余数运算。

```
>>> 0.1 + 0.4
0.5
>>> 0.3 - 0.4
-0.10000000000000003
>>> 0.5 * 4
2.0
>>> 0.6 / 6
0.09999999999999999
>>> 4.5 % 0.6
0.30000000000000016
>>> |
```

注意：浮点数的运算会产生一点误差，如 $0.1 + 0.2 = 0.30000000000000004$ ，浮点数的运算结果包含的小数并不能确定，这是因为受到 IEEE754 浮点数算术标准等的影响。对于这种情况下牺牲一定性能，使用 Decimal 模块来避免。

```
>>> 0.1 + 0.2
0.30000000000000004
>>> from decimal import Decimal
>>> 0.1 + 0.2
0.30000000000000004
>>> a = Decimal('0.1')
>>> b = Decimal('0.2')
>>> a + b
Decimal('0.3')
>>> print(a+b)
0.3
>>> |
```

(2) 分数的运算

Python 中常通过 Fraction 模块来处理分数

```
>>> from fractions import Fraction
>>> a = 1/4
>>> b = Fraction(5,4)
>>> print(b)
5/4
>>> print(a+b)
1.5
>>> print(b**2)
25/16
```

(3) 对数值格式化处理

a) 数值取整

简单的取整操作，可采用内建的函数 round(value,ndigits)处理。

代码：

```
#保留两位小数
print(round(1.23333,2))
#保留一位小数
print(round(2.3366,1))
#python的就近偶数原则
print(round(1.5))
print(round(2.5))
'''当ndigits为负数时，round()会取整到相应的整数位'''
#取整到个位
print(round(532,-1))
#取整到千位
print(round(236452,-3))
```

结果：

```
/home/ubuntu/PycharmProjects/uesofheapq/venv/bin/python /
1.23
2.3
2
2
530
236000
```

b) 数值做格式化输出

对单独的数值做格式化处理（包括控制位数，对齐，包含千分位分隔符，采用科学计数法等），均可采用 format()函数。

代码：

```

x = 1234.56789
#控制x的输出为两位小数
print(format(x,'0.2f'))
#控制x的输出为四位小数
print(format(x,'0.4f'))

#给x增加千位分隔符
print(format(x,'0,.4f'))

#x输出为科学计数法格式,将'f'替换成'e'或'E'即可
print(format(x,'0.2e'))

```

结果:

```

/home/ubuntu/PycharmProjects/uesofheapq/venv/bin/python /
1234.57
1234.5679
1,234.5679
1.23e+03

```

1.1.3 复数类型

(1) 复数的表示

Python 中通过 `complex(real,imag)` 类来定义一个复数,其中 `real` 为实部,`imag` 为虚部,也可用浮点数加上后缀 `j` 来制定复数。

代码:

```

a = 4 + 5j
b = complex(5,7)
print(a,b)
#查看a的数据类型
print(type(a))
#通过'.'运算符调用实部虚部
print(b.imag,b.real)

```

结果:

```

/home/ubuntu/PycharmProjects/uesofheapq/venv/bin/python
(4+5j) (5+7j)
<class 'complex'>
7.0 5.0

```

(2) 复数的运算

与整数相识,复数也可进行加减乘除运算,也可通过 `abs()` 方法来求出复数的模。

代码:


```

a = 4 + 3j
b = complex(5,7)

print(a,b)
print(a+b)
print(a-b)
print(a*b)
print(a/b)
print("a的模值为" + str(abs(a)))

```

结果:

```

/home/ubuntu/PycharmProjects/uesofheapq/venv/bin/python
(4+3j) (5+7j)
(9+10j)
(-1-4j)
(-1+43j)
(0.5540540540540542-0.17567567567567569j)
a的模值为5.0

```

1.2 字符串类型

1.2.1 字符串的表示

(1) 字符串的创建

字符串是 Python 中最常用的数据类型。我们可以使用引号('或")来创建字符串。Python 不支持单字符类型，单字符在 Python 中也是作为一个字符串使用。

```

>>> str1 = 'I love tou'
>>> str2 = "I love tou"
>>> print(str1==str2)
True
>>> 'T' == "T"
True
>>> |

```

(2) 三引号字符串

python 三引号（单引号双引号皆可）允许一个字符串跨多行，字符串中可以包含换行符、制表符以及其他特殊字符。三引号长出现在类的定义中，用作函数的解释注释。

```

>>> para_str = """这是一个多行字符串的实例
多行字符串可以使用制表符
TAB ( \t )。
也可以使用换行符 [ \n ]。
""" ... ..
>>> print(para_str)
这是一个多行字符串的实例
多行字符串可以使用制表符
TAB (    )。
也可以使用换行符 [
]。

```

(3) 转义字符

在需要在字符中使用特殊字符时，python 用反斜杠(\)转义字符。

转义字符	描述
\(在行尾时)	续行符
\\	反斜杠符号
'\'	单引号
'\"'	双引号
\a	响铃
\b	退格(Backspace)
\e	转义
\000	空
\n	换行
\v	纵向制表符
\t	横向制表符
\r	回车
\f	换页
\oyy	八进制数，yy代表的字符，例如：\o12代表换行
\xyy	十六进制数，yy代表的字符，例如：\x0a代表换行
\other	其它的字符以普通格式输出

实例：


```

#当\字符位于行位时作为续行符,用于连接较长的字符串
message = "My name is "\
    "Bob."
print(message)
#\与',",\等相连时,表示所连接的字符本身
words = "I\'m learning Python."
print(words)
"""转义字符常用于windows系统中的文件路径"""
address = "c:\\User\\郭远鹏\\Desktop\\"
print(address)

```

结果:

```

/home/ubuntu/PycharmProjects/uesofheapq/venv/bin/python
My name is Bob.
I'm learning Python.
c:\User\郭远鹏\Desktop\

```

在字符串前添加 `r` 时表示为原始字符串,所有的字符不考虑转义字符的影响,只当普通字符处理,从而不受换行符等的影响。

```

>>> str1 = "c:\nowhere"
>>> str2 = r"c:\nowhere"
>>> print(str1)
c:
owhere
>>> print(str2)
c:\nowhere

```

1.2.2 字符串的格式化

`Format` 是 `python2.6` 新增的一个格式化字符串的方法,相比较与快被淘汰的 `%` 方法, `format` 更为灵活,可处理包括字符串,浮点数等多种数据类型,他的填充方式和对齐方式十分强大。

(1) 填充

a) 通过位置顺序填充字符串

代码:

```

#通过位置顺序填充
print('Hello {0}, I am {1}'.format('Cauchy','Bob'))
print('Hello {}, I am {}'.format('Bob','Cauchy'))

```

结果:

```

/home/ubuntu/PycharmProjects/uesofheapq/venv/bin/python
Hello Cauchy, I am Bob
Hello Bob, I am Cauchy

```

b) 通过关键词来填充

代码:

```
#通过key顺序填充(可打乱顺序)
print('Hello {name1}, I am {name2}'.format(name2='Cauchy',name1='Bob'))
```

结果:

```
/home/ubuntu/PycharmProjects/uesofheapq/venv/bin/python
Hello Bob, I am Cauchy
```

c) 通过下标填充(通常用于列表处理)

代码:

```
#通过列表下标(可打乱顺序)
names = ['Bob', 'Cauchy', 'Lucy']
print('Hello {0[1]}, I am {0[0]}'.format(names))
print('Hello {names[2]}, I am {names[1]}'.format(names_=names))
```

结果:

```
/home/ubuntu/PycharmProjects/uesofheapq/venv/bin/python
Hello Cauchy, I am Bob
Hello Lucy, I am Cauchy
```

d) 通过字典的 key(用于字典)

代码:

```
#通过字典的key
names = {
    'name1': 'Bob',
    'name2': 'Tom',
    'name3': 'Cauchy'
}
print('Hello {0[name1]}, I am {0[name3]}'.format(names))
print('Hello {names[name2]}, I am {names[name3]}'.format(names_=names))
```

结果

```
/home/ubuntu/PycharmProjects/uesofheapq/venv/bin/python
Hello Bob, I am Cauchy
Hello Tom, I am Cauchy
```

注意: 与字典中通过相对应 key 访问 value 不同, format 中括号中无需对 key 添加引号。

(2) 格式转化

数字	格式	输出	描述
3.1415926	{:.2f}	3.14	保留小数点后两位
3.1415926	{:+.2f}	3.14	带符号保留小数点后两位
-1	{:+.2f}	-1	带符号保留小数点后两位
2.71828	{:.0f}	3	不带小数
1000000	{:,}	1,000,000	以逗号分隔的数字格式
0.25	{:.2%}	25.00%	百分比格式
1000000000	{:.2e}	1.00E+09	指数记法
25	{0:b}	11001	转换成二进制
25	{0:d}	25	转换成十进制
25	{0:o}	31	转换成八进制
25	{0:x}	19	转换成十六进制

具体操作可参考 1.1.2 浮点数的格式化处理

(3) 对齐与填充

数字	格式	输出	描述
4	{:0>2}	40	数字补零 (填充左边, 宽度为 2)
4	{:x<4}	4xxx	数字补 x (填充右边, 宽度为 4)
4	{:x^4}	X4xx	两边补 x (宽度为 4)
12	{:10}	12	右对齐(默认, 宽度为 10)
24	{:<10}	24	左对齐(宽度为 10)
48	{:^10}	48	中间对齐(宽度为 10)

代码:

```
x = 4
print('{:0<2}'.format(x))
print('{:x<4}'.format(x))
print('{:x^4}'.format(x))
print('{:10}'.format(3*x))
print('{:<10}'.format(6*x))
print('{:^10}'.format(12*x))
```

结果:

```
/home/ubuntu/PycharmProjects/uesofheapq/venv/bin/python
40.
4xxx.
x4xx.

      12.
24      .
      48      .
```

1.2.3 字符串操作

代码:

```
#字符串的连接
conversion_ = "What day is it today?"
conversion += "\nIt's Sunday."
print(conversion)

print("-"*50)
#字符串的乘法
msg = "重要的事情说三遍 "
print(msg*3)

print("-"*50)
#字符串的匹配与查找(startswith(),endswith(),find(),)
text = "All things in their being are good for something."
print(text.startswith('all')) #检查字符串的开头是否正确(区分大小写)
print(text.endswith("something")) #检查字符串的结尾是否正确
print(text.find('something')) #查找特定字符串是否存在并返回这个字符串的首字母的索引,若不存在则返回-1
print(text.find('all'))
print(text.replace('All','No')) #将字符串中存在的字符串做替换
```

结果:

```
/home/ubuntu/PycharmProjects/Demo/venv/bin/python
What day is it today?
It's Sunday.

重要的事情说三遍 重要的事情说三遍 重要的事情说三遍

False
False
39
-1
No things in their being are good for something.
```

1.3 列表

1.3.1 列表的定义

列表是由一系列按**特定顺序**排列的元素组成。与集合的无序和互异性不同,列表的各元素之间无任何关系,可重复出现多次且顺序一定。

1.3.2 列表的表示

Python 中用[]来表示一个列表,列表中为各个元素,元素之间以逗号间隔。Python 的列表元素十分灵活,可以时字符串、数字、字典,元组甚至列表中嵌套列表。

代码:


```

#简单申明几个列表变量
names = ['Bob','Tom','Cauchy','Lucy']
students = [{'name':'Tom','age': 19,'location':'HuBei'},
             {'name':'Cauchy','age': 20,'location':'JiangXi'}]

numbers = list(range(1,10))#通过range()函数生出一系列有规律的数字,再由list()函数列表化
value_square = [value**2 for value in range(0,10)] #一个简单的列表解析

#列表的访问
print(numbers)#直接print出列表
print(value_square)

for student in students:#通过for循环遍历列表元素
    print(student['name'] + ', who aged ' + str(student['age']) +
          ', is from ' + student['location'] + '.')

print(names[0],names[2])#通过索引直接访问列表元素

```

结果:

```

/home/ubuntu/PycharmProjects/Demo/venv/bin/python
[1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
Tom, who aged 19, is from HuBei.
Cauchy, who aged 20, is from JiangXi.
Bob Cauchy

```

1.3.3 列表的基本操作

代码:

```

#列表的添加
names = []#申明一个空列表
names.append('Tom')#通过append()从列表末尾添加元素
names.append('Bob')
names.insert(0,'Cauchy')#通过insert()可通过指定索引来插入元素
names.insert(1,'Mike')
print(names)

#列表的删除
value = 2
del names[value]_# del语句可通过指定列表的元素索引来删除相应的元素
print(names)
guy = names.pop(value)#pop()方法可通过指定列表元素的下标来删除指定,
print(guy,names) #默认在不带参数的情况下弹出最后一位,pop()方法返回弹出的元素值
names.remove('Mike')#remove方法可在不知道元素具体的情况下,根据元素的值来删除元素
print(names)

```

结果:

```

/home/ubuntu/PycharmProjects/Demo/venv/bin/python
['Cauchy', 'Mike', 'Tom', 'Bob']
['Cauchy', 'Mike', 'Bob']
Bob ['Cauchy', 'Mike']
['Cauchy']

```

1.3.4 列表的高阶操作

代码:

```

#列表切片
subjects = ['Maths', 'English', 'Chinese', 'PE', 'Arts', 'OS']
"""通过对已有的列表指定其起始索引与终点索引来切片得到新的列表。[:]表示切片所有的元素(可用以复制列表)
与列表的索引相同，切片只截取到终点索引前一位"""
major_subjects = subjects[0:4]
elective_subjects = subjects[-2:] #当索引值为负数(-n)时表示倒数第n个值
print("major subject: {0},elective subjects: {1}".format(major_subjects,elective_subjects))
#分解列表元素
#对于可迭代对象如列表,可将一系列变量配合*来分解迭代对象,提取有效信息
record = "Cauchy,cauchyguo@gmail.com,Chongqing,17723084945,cs"
name, mail, *hehe, depart = record.split(',')
print(name,mail,depart)
#列表加法与乘法
"""列表和字符串一样支持加法与乘法"""
numbers = list(range(1,6))
nums1 = numbers + numbers
nums2 = numbers * 2
print(nums1,nums1 == nums2)
print(list(set(nums1))) #对于列表去除重复,可以用set()函数转化为集合再通过list()转化为列表
#通过Counter类来统计列表元素的出现次数
from collections import Counter
sentences = "jdaiojdoaojdaiojdoajdoeqwqajisijiaioeokd"
words = [s for s in sentences] #将字符串分解成单字符组成的列表
objectC = Counter(words) #建立一个Counter类
top_three = objectC.most_common(3) #调用Counter类中most_common()方法查找频率最高的三个元组并返回列表元组
print(top_three)

```

结果:

```

/home/ubuntu/PycharmProjects/Demo/venv/bin/python /home/ubuntu/PycharmProjects/Demo/venv/bin/main.py
major subject: ['Maths', 'English', 'Chinese', 'PE'],elective subjects: ['Arts', 'OS']
Cauchy cauchyguo@gmail.com cs
[1, 2, 3, 4, 5, 1, 2, 3, 4, 5] True
[1, 2, 3, 4, 5]
[('j', 9), ('o', 9), ('d', 6)]

Process finished with exit code 0

```

1.4 元组

1.4.1 元组的定义

列表适合用以储存可变化的数据集,因为列表非常方便修改。而不可变的列表被称为元组。列表使用方括号而元组使用圆括号,除此外,元组的操作与列表基本相似。

1.4.2 元组的表示

代码:

```

#元组的创建
a_tuple = (4,5,6,7)
#元组的访问
print(a_tuple)
print("a_tuple[2] =",a_tuple[2]) #通过索引直接访问元组元素
for value in a_tuple:
    print(str(value) + '\t',end='') #通过for循环遍历元组元素
print('\n' + '-'*10)
#元组的不可修改性
try: #使用try-except来检测元组能否修改
    a_tuple[2] = 0
except TypeError:
    print("a tuple one doesn't support modification")
else:
    print("a_tuple[2] =",a_tuple[2])

```

结果:

```

/home/ubuntu/PycharmProjects/Demo/venv/bin/python
(4, 5, 6, 7)
a_tuple[2] = 6
4 5 6 7
_____
a tuple one doesn't support modification

```

1.5 集合

1.5.1 集合的定义

Python 的集合 **set** 与其他语言相似，是一个无序，不重复的元素集合，基本关系包括关系测试和消除重复元素。集合支持交、差等数学运算。

1.5.2 集合的操作

代码：

```

#集合的创建
a = set(['a', 'b', 'c', 'd']) #创建一个数值集合
string = 'apple'
s = set(string) #创建一个字符集合
#集合的性质
for c in a:
    print(c, end=' ') #集合支持for-in遍历,但是输出不确定(集合的无序性)
print()
print(len(string), len(s)) #集合自动消除重复元素,集合的唯一性
print('-'*30)
#集合的操作
a.add('x') # add()添加一项
print("a添加一项后：" + a)
a.remove('x') #remove()移除一项
print("a移除一项后：" + a)
s.update(['m', 'n']) #update()添加多项
print("s一次添加多项后：" + s)
s.clear() #clear()将集合清零
print(s)
s = set(string)
print('-'*30)
#集合的数学运算
print('s:' + s + '\t' + 'a:' + a)
print('s - a =' + s - a) #求集合的差集
print('s | a =' + s | a) #求集合的并集
print('s ^ a =' + s ^ a) #求集合的对称差集(同时不在两集合中的元素集合)
print('s & a =' + s & a) #求两集合交集

```

结果：


```

/home/ubuntu/PycharmProjects/Demo/venv/bin/python /home/
bdca
5 4
-----
a添加一项后: {'b', 'a', 'c', 'x', 'd'}
a移除一项后: {'b', 'a', 'c', 'd'}
s一次添加多项后: {'a', 'e', 'n', 'l', 'p', 'm'}
set()
-----
s: {'p', 'l', 'e', 'a'}      a: {'b', 'a', 'c', 'd'}
s - a = {'p', 'l', 'e'}
s | a = {'b', 'a', 'e', 'c', 'd', 'p', 'l'}
s ^ a = {'b', 'e', 'c', 'd', 'p', 'l'}
s & a = {'a'}

```

1.6 字典

1.6.1 字典的定义

在 Python 中，字典就是一系列键-值对。每一个键都与一个值相关联，可以使用键来访问相关联的值，与键相关联的值可以是数字、字符串、列表甚至字典。

1.6.2 字典的表示

代码:

```

student = {'name': 'Cauchy', 'ID': '2016212074', 'dept': 'CS'} #建立一个字典对象
#字典的访问
print(student['name']) #通过键访问值
for key, value in student.items(): #items()方法返回一个键值对元组列表,通过for-in 即可遍历字典的内容
    print(key, value) #keys, values等方法与items相似
#键值对的添加与修改
student['sex'] = 'man' #通过指明键来对字典进行添加和修改
student['dept'] = 'IS'
print(student)

#字典键值对的删除
del student['sex'] #使用del即可将字典的键及其对应的值删除
print(student)

```

结果:

```

/home/ubuntu/PycharmProjects/Demo/venv/bin/python /home/ubuntu/PycharmProjects/Demo/venv/bin/main.py
Cauchy
name Cauchy
ID 2016212074
dept CS
{'name': 'Cauchy', 'ID': '2016212074', 'dept': 'IS', 'sex': 'man'}
{'name': 'Cauchy', 'ID': '2016212074', 'dept': 'IS'}

```

注意：字典中的键值对是无序的，通过 for-in 的方式遍历时结果不确定。若想通过指定的顺序来遍历，可通过列表储存键的顺序，在通过列表的方式来遍历，如下图，除此之外，也可使用 collections 中 OrderedDict 类(详见 1.6.3)。

发现：本来在初学 Python 字典时使用的是 Python3.5.2，后来在做本实验时用的 Python3.6.5，发现在 3.6 中，字典的遍历始终是有序的，且顺序与创建时的顺序相关。原来 3.6 添加了新的特性，更换了 dict 的实现方式，使字典顺序保留，同时节省了 20%-25% 的内存，但为了兼容之前版本还是介绍了处理字典无序的方法。详见：<https://docs.python.org/3/whatsnew/3.6.html#whatsnew36-compactdict>

```

ubuntu@Lenovo:~$ python3.5
Python 3.5.2 (default, Nov 23 2017, 16:37:01)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> user = {
...     'user_name': 'Cauchy',
...     'first_name': 'Tom',
...     'last_name': 'Kobe',
... }
>>> for key,value in user.items():
...     print(key,value)
...
user_name Cauchy
last_name Kobe
first_name Tom
>>> lists = ['user_name', 'first_name', 'last_name']
>>> for list in lists:
...     print(list,user[list])
...
user_name Cauchy
first_name Tom
last_name Kobe

```

1.6.3 字典的高级操作

代码:

```

#通过OrderedDict模块使字典有序
from collections import OrderedDict
user = OrderedDict()
user['user_name'] = 'Cauchy'      #OrderedDict内部维护了一个双向链表,会根据元素添加进来的顺序排列键的位置
user['first_name'] = 'Bob'        #但是这种做法几乎使字典的存储空间翻倍
user['last_name'] = 'Kobe'
for key, value in user.items():
    print(key,value)
print('-'*20)
#通过defaultdict模块使字典的键映射多个值(列表list或集合set)
from collections import defaultdict
d = defaultdict(list)
d['a'].append([1,2,4]) #添加多个值到'a'键
d['b'].append(5)
for key,value in d.items():
    print(key,value)
e = defaultdict(set)
e['a'].add(1)
e['a'].add(1) #集合的唯一性
e['b'].add(3)
print(e)
#字典推导式
"""字典的keys()方法返回key-view对象,由此可对keys()对象做类集合运算,交并差对称等运算"""
print('-'*20)
a = {2:2,3:3,4:4}
b = {1:2,2:3,3:3}
print(a.keys()&b.keys()) #结果返回一个集合包含两者的交集
c = {key:a[key] for key in a.keys() & b.keys()} #通过字典推导式可得a中与b都存在的键的新字典
print(c)

```

结果:

```

/home/ubuntu/PycharmProjects/Demo/venv/bin/python
user_name Cauchy
first_name Bob
last_name Kobe
-----
a [[1, 2, 4]]
b [5]
defaultdict(<class 'set'>, {'a': {1}, 'b': {3}})
-----
{2, 3}
{2: 2, 3: 3}

```

第2章 控制与循环

2.1 if 语句

2.1.1 If-elif-else 结构

与 C 语言等相比较，Python 采用了相似的 if 语句结构。不同的是，if 之后的判断语句时 elif(等效于 C 中的 else if)，且判断的语句无须括号，只需末尾添加“:”即可，也无须大括号包裹语句块，简单通过缩进次数来实现分块。

代码：

```

msg = "How old are you?" #输入提示
msg += "\nage:"
age = int(input(msg)) #将输入的字符整型化
#多重if-elif-else判断语句
if 0 < age and age < 18:
    print("You are a minor.")
elif 18 <= age and age < 35:
    print("You are a adult.")
elif 35 <= age and age < 60:
    print("You are a middle-aged person.")
elif age >= 60:
    print("You are the old.")
else: #当输入的数据不在有效范围内,坚持使用else以加强健壮性
    print("You are strange man.")

```

结果：

age = 15

```

/home/ubuntu/PycharmProjects/Demo/venv/bin/python
How old are you?
age: 15
You are a minor.

```

age = 60

```
/home/ubuntu/PycharmProjects/Demo/venv/bin/python /f
How old are you?
age:60
You are the old.
```

age = -10

```
/home/ubuntu/PycharmProjects/Demo/venv/bin/python
How old are you?
age:-10
You are strange man.
```

2.1.2 Python 中“switch-case”的实现

当处理多重匹配时，在 C、java 中常使用 switch-case 语句。从官方 Tutorial 文档([Why isn't there a switch or case statement in Python?](#))

中得知，我们可采用字典来解决。

代码：

```
def func0():
    print('This is the default function.')
def func1():
    print('This is function 1.')
def func2():
    print('This is function 2')
def func3():
    print('This is function 3')

functions = {
    0:func0,
    1:func1,
    2:func2,
    3:func3,
}

menu = 'which function do you want to run\n' #生成人性化菜单
menu += '1 -> func1\n2 -> func2\n3 -> func3\n'
menu += 'choice:'
choice = int(input(menu)) #将字典的键数值化
#字典的get方法通过传递的第一个参数查找并返回指定的键的值,若不存在则返回default参数的值(default默认为none)
functions.get(choice,func0)()
```

结果：

```
/home/ubuntu/PycharmProjects/Demo/venv/bin/python
which function do you want to run
1 -> func1
2 -> func2
3 -> func3
choice:1
This is function 1.
```



```

/home/ubuntu/PycharmProjects/Demo/venv/bin/python
which function do you want to run
1 -> func1
2 -> func2
3 -> func3
choice: 100
This is the default function.

```

2.2 循环语句

2.2.1 for 循环与 while 循环

代码:

```

#for循环
for value in range(0,11,2): #0-10之间的偶数
    print(value, ' ', end='')
print()
print('-'*20)

#while循环
Test = True
lists = []
while Test: #在while语句中通常设置一个标签，通过修改标签的真假的决定退出循环
    words = input("Input a word('quit' to leave)\nwords:")
    if words.lower() != 'quit': #字符串的lower方法使字符串的字符全部改为小写与之相反的upper方法
        lists.append(words)
        continue #continue语句一旦执行之间跳过单次循环的剩余语句直接执行下一次循环
    else:
        Test = False #此句效果等价与break,间接的退出了循环
for list in lists:
    print(list, ' ', end='')

```

结果:

```

/home/ubuntu/PycharmProjects/Demo/venv/bin/python
0 2 4 6 8 10
-----
Input a word('quit' to leave)
words: a
Input a word('quit' to leave)
words: ab
Input a word('quit' to leave)
words: abc
Input a word('quit' to leave)
words: quiT
a ab abc
Process finished with exit code 0

```

第3章 列表推导式

推导式是 Python 中的一独有特性。推导式是可以从一个数据序列构建另一个新的数据序列的结构体。推导式有列表推导式，集合推导式，字典推导式三种。

代码:

```

#列表推导式 使用[]生成列表
odds = [x for x in range(1,30) if x % 2 is 1] #列表推导式
print(odds,type(odds)) #查看类型
#列表推导式支持多重for循环
a = [[1,2,3],[3,4,5],[5,6,7],[7,8,9]]
nums = [x**2 for i in a for x in i]
print(nums)
#模拟掷骰子实验
from random import randint
from collections import Counter
experiment = [randint(1,6) for i in range(10000)] #生成一万次的随机实验
nums_count = Counter(experiment) #使用Counter来统计实验
results = nums_count.most_common(6)
for r in sorted(results,key=lambda x:x[0]):
    print(r[0],':',r[1],':{:.2%}'.format(float(r[1]/10000)))
#字典推导式 使用{}生成字典
names = ['Bob','Tom','Lucy','Cauchy']
height = [172,163,152,175]
student_h = {x:y for x in names for y in height}
print(student_h)

```

结果:

```

/home/ubuntu/PycharmProjects/Demo/venv/bin/python /home/ubuntu/PycharmProjects,
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29] <class 'list'>
[1, 4, 9, 16, 25, 36, 49, 64, 81]
1 : 1667 16.67%
2 : 1629 16.29%
3 : 1729 17.29%
4 : 1639 16.39%
5 : 1679 16.79%
6 : 1657 16.57%
{'Bob': 175, 'Tom': 175, 'Lucy': 175, 'Cauchy': 175}

```

注意: 使用()并不会生成元组, 而会得到一个生成器对象

代码:

```

Python 3.6.5 (default, Apr 4 2018, 23:33:46)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> x = (x for x in range(1,10))
>>> type(x)
<class 'generator'>
>>> print(x)
<generator object <genexpr> at 0x7f0aa3245308>
>>> set(x)
{1, 2, 3, 4, 5, 6, 7, 8, 9}
>>> |

```

第4章 函数

4.1 函数的结构

4.1.1 函数的定义

函数是带名字的代码块, 用以完成具体的工作。具体结构如下:

代码:

```

#函数代码块以def关键词开头,后接函数标识符名称和圆括号()
#圆括号为传递的参数,参数分为形参(language)与实参("java")
def hello_world(language): #冒号起始
    """a function to greet"""#函数的第一行语句为文档字符串,存放函数说明
    print("Hello, " + language + " world!")
    return #return结束函数并选择性地返回值给传递方,不带表达式则返回None

languages = ['java','c','python','JavaScript']
for language in languages:
    hello_world(language.title()) #函数的调用

```

结果:

```

/home/ubuntu/PycharmProjects/Demo/venv/bin/python
Hello, Java world!
Hello, C world!
Hello, Python world!
Hello, Javascript world!

```

4.2 函数的参数

函数参数分为实参与形参,函数通过传递参数来调用函数,函数在调用中也可包含实参。

4.2.1 传参的几种方式

代码:

```

#通过位置传递参数
def greet(name,time,sentence):
    return "Good "+time+", "+name+'. '+sentence
print(greet("Bob","morning","Nice to meet you.))
print("参数位置改变后:")
print(greet("morning","Bob","Nice to meet you.)) #通过函数括号内参数的顺序来匹配参数。当参数位置交换时,结果出人意料
print("-"*35)

#通过指定关键字传递参数
"""指定关键字值可以无需考虑调用函数时参数的位置"""
print(greet(time='Afternoon',sentence="Are you ok?",name='Cauchy'))
print("-"*35)

#编写函数时指定默认值
def greet_plus(name,time='evening',sentence="I miss you."):
    return ("Good "+time+", "+name+'. '+sentence)
print(greet_plus("Tom"))
"""对于存在默认值的参数,若无匹配的参数传递则使用默认的值,见下:"""
print(greet_plus("Tom") == greet_plus("Tom",'evening','I miss you.))

```

结果:


```

/home/ubuntu/PycharmProjects/Demo/venv/bin/python /h
Good morning, Bob. Nice to meet you.
参数位置改变后：
Good Bob, morning. Nice to meet you.
-----
Good Afternoon, Cauchy. Are you ok?
-----
Good evening, Tom. I miss you.
True

Process finished with exit code 0

```

总结，函数参数的传递常有 **a** 通过顺序位置。**b** 通过关键字指定。**c** 通过默认值。三种，三种方式可结合使用。如对于某些较为固定的值可指定默认参数。默认参数最好放在末尾或者更改时使用关键字修改，因为若默认值参数的位置靠前，再通过顺序位置传参时，跳过默认参数会把参数当做默认项传递过去。

4.2.2 传递任意数量或类型的参数

(1) 可接受任意数量位置参数的函数

对于一些参数数量未定的函数如求和函数，可在函数中添加*开头的参数，使python创建了一个空元组以存储所有传递过来的位置参数。

代码：

```

def sum1(first,*rest):
    print(type(rest)) #rest是一个元组
    sum = first
    print("first = " + str(first))
    for other in rest:
        sum += other
    return sum
print("Sum = " + str(sum1(1,2,3,4,5,6)))

```

结果：

```

/home/ubuntu/PycharmProjects/Demo/venv/bin/python
<class 'tuple'>
first = 1
Sum = 21

```

(2) 可接受任意数量关键字参数的函数

对于任意数量指定了关键字的参数，可在函数定义中添加一个**开头的参数(实为字典类型)，他将传递过来的关键字及其实参封装为一个字典。

代码：

```
def information(first_name,last_name,**others):
    print(type(others)) ***型变量为字典类型
    man = {}
    man['first_name'] = first_name
    man['last_name'] = last_name
    for key,value in others.items(): ##可迭代
        man[key] = value
    return man

print(imformation('albert','einstein',sex='male',location='princeton',field='physics'))
```

结果:

```
/home/ubuntu/PycharmProjects/Demo/venv/bin/python /home/ubuntu/PycharmProjects/Demo/venv/bin/main.py
<class 'dict'>
{'first_name': 'albert', 'last_name': 'einstein', 'sex': 'male', 'location': 'princeton', 'field': 'physics'}
```

总结: 根据参数传递方式不同的任意数量(位置传递或关键字传递), 可分别采用添加*或**参数来接受参数。同时还可以根据*参数的特性选择性的接受关键字参数, 如 (def recv(maxsize,*_,block))中, 函数只接受来自指定关键字的参数, 过滤了其他无用的参数。

4.3 匿名函数

当我们需要一个短小的回调函数为如同排序操作所用时, 可使用更为简短的 lambda 表达式而非 def 语句。lambda 表达式创建了一个匿名函数, 同时生成了一个函数对象。Lambda 表达式形式: lambda arguments:expression, 等效于:

```
def <lambda>(arguments):
    return expression
```

代码:

```
add = lambda x,y:x + y
print(add(5,6))
print(add('hello','world'))
#lambda常用于sort排序的关键词
students = [
    {'name': 'bob', 'grades': 79},
    {'name': 'tom', 'grades': 80},
    {'name': 'cauchy', 'grades': 84},
]
print("按成绩高低排序")
for student in sorted(students, key=lambda student: student['grades'], reverse=True):
    print(student) ##lambda后的student为列表的元素, 冒号后为表达式返回的结果, key的实参
print("按名字首字母排序")
for student in sorted(students, key=lambda student: student['name']):
    print(student)
```

结果:

```

/home/ubuntu/PycharmProjects/Demo/venv/bin/python
11
helloworld
按成绩高低排序
{'name': 'cauchy', 'grades': 84}
{'name': 'tom', 'grades': 80}
{'name': 'bob', 'grades': 79}
按名字首字母排序
{'name': 'bob', 'grades': 79}
{'name': 'cauchy', 'grades': 84}
{'name': 'tom', 'grades': 80}

```

总结: **lambda** 表达式十分方便简洁, 其主体是一个表达式而不是代码块, 他只能指定一条单独的表达式, 即表达式的结果为等效函数的返回值, 这意味着一般函数中所能用到的多行语句, 条件分支, 迭代, 异常处理等无法使用, 并且 **lambda** 也不同访问自有参数列表之外或全局命名空间内的参数。

4.4 高阶函数

4.4.1 排序函数

Python 列表中 **sort()** 方法可对列表永久排序, 也有内置的全局方法 **sorted()** 对可迭代的序列生成新的临时序列。两者都相同的关键词 **key** 和 **reverse**。函数铜鼓接受一个可用来传递的可调用对象的参数 **key**, 而该对象会返回待排序对象中的某些值, **sort** 则利用这些值来比较, 排序。一般情况下 **sort** 根据参数对对象元素按从小到大或字母从小到大排序, **reverse** 关键词默认为 **False**, 若传递参数 **reverse=True**, 则更具 **key** 反向排序。

代码:

```

students = [
    {'name': 'bob', 'grades': 79},
    {'name': 'tom', 'grades': 80},
    {'name': 'cauchy', 'grades': 84},
    {'name': 'mike', 'grades': 84}
]
def list_students(students, sort_key=''):
    print("Sort by", sort_key)
    print('\tname\tgrades')
    for student in students:
        print('\t{:<8} {}'.format(student['name'].title(), student['grades']))
# 若想通过学生的名字字母永久顺序排序, 使用list()
students.sort(key=lambda student: student['name'])
list_students(students, "name")
# 若想对学生按照分数临时从高到低排序(做菜单式输出)
list_students(sorted(students, key=lambda student: student['grades'], reverse=True), "grades")
# 检查列表实际有无变化
list_students(students) # 结果与第一次输出相同
# 可同时指定多个key, 比如按分数, 姓名排序
list_students(sorted(students, key=lambda student: (student['grades'], student['name']),
                    reverse=True), "grades and name")

```

结果:

```
/home/ubuntu/PycharmProjects/Demo/venv/bin/python
Sort by name :
  name    grades
  Bob      79
  Cauchy   84
  Mike     84
  Tom      80
Sort by grades :
  name    grades
  Cauchy   84
  Mike     84
  Tom      80
  Bob      79
Sort by :
  name    grades
  Bob      79
  Cauchy   84
  Mike     84
  Tom      80
Sort by grades and name :
  name    grades
  Mike     84
  Cauchy   84
  Tom      80
  Bob      79
```

总结，对于排序中 `key` 的参数常使用 `lambda`，除此之外也可使用 `operator` 模块中的 `itemgetter` 函数，通常 `itemgetter()` 在字典列表的操作中速度快一些，用的也多一些。`Itemgetter` 可直接接受字典的键的值传递给 `key`。

代码：


```

from operator import itemgetter
students = [
    {'name': 'bob', 'grades': 79},
    {'name': 'tom', 'grades': 80},
    {'name': 'cauchy', 'grades': 84},
    {'name': 'mike', 'grades': 84}
]

def list_students(students, sort_key=''):
    print("Sort by", sort_key)
    print('\tname\tgrades')
    for student in students:
        print('\t{:<8} {}'.format(student['name'].title(), student['grades']))

#若想通过学生的名字字母永久顺序排序，使用list()
students.sort(key=itemgetter('name'))
list_students(students, "name")
#若想对学生按照分数临时从高到低排序(做菜单式输出)
list_students(sorted(students, key=itemgetter('grades'), reverse=True), 'grades')
#检查列表实际有无变化
list_students(students) #结果与第一次输出相同

```

结果:

```

/home/ubuntu/PycharmProjects/Demo/venv/bin/python
Sort by name :
    name    grades
    Bob      79
    Cauchy   84
    Mike     84
    Tom      80
Sort by grades :
    name    grades
    Cauchy   84
    Mike     84
    Tom      80
    Bob      79
Sort by :
    name    grades
    Bob      79
    Cauchy   84
    Mike     84
    Tom      80

```

4.4.2 map 函数

`map()`会根据提供的函数对指定序列做映射。`map()`接受一个函数参数，然后对后续的参数列表中每一个元素依次调用函数，返回每次函数返回值的新列表，生成一个 `map` 对象(`python2` 中为一个列表，`python3` 中为迭代器)，注意，这个函数可以为 `lambda` 表达式，`map` 常与 `lambda` 结合使用。

代码:

```

#求列表元素的平方
nums = list(range(0,9))
nums_square = map(lambda x:x**2,nums)
print(type(nums_square)) #生成一个map对象
print(list(nums_square)) #对map对象使用list生成列表
#提取列表中开头字母为'San'的元素
citys = ["San Jose", "San Francisco", "Santa Fe", "Houston"]
def find_san(city):
    if city.startswith('San'):
        return city
san_city = list(map(find_san,citys))
print(san_city)

```

结果:

```

/home/ubuntu/PycharmProjects/Demo/venv/bin/python
<class 'map'>
[0, 1, 4, 9, 16, 25, 36, 49, 64]
['San Jose', 'San Francisco', 'Santa Fe', None]

Process finished with exit code 0

```

4.4.3 reduce 函数

reduce 函数对参数列表中元素进行累积操作,函数通常对一个数据集合做如下操作:用传给 reduce 中的函数(有两个参数),先对集合中的第一二个元素进行操作,再将得到的结果与下一个元素进行操作,最后得到结果。

代码:

```

from functools import reduce
from random import randint #导入随机数模块
#对数值列表求和
nums = list(range(1,10))
print(nums)
sum = reduce(lambda x,y:x+y,nums)
print(sum)

random = set()
for x in range(0,15): #生成是个随机数组
    random.add(randint(0,100))
print("集合: ",random,"长度: ",str(len(random)))
#找出集合中的最大数
max = reduce(lambda x,y:x if x > y else y,random) #对集合中的元素进行遍历比较
print("最大数: ",str(max))

```

结果:

```

/home/ubuntu/PycharmProjects/Demo/venv/bin/python /home/ubuntu/Pycha
[1, 2, 3, 4, 5, 6, 7, 8, 9]
45
集合: {0, 64, 38, 6, 73, 10, 43, 18, 19, 84, 52, 57, 27} 长度: 13
最大数: 84

```

注意: 在 python3 中 reduce 模块不在作为内置函数的存在,而是存放在 functools 模块中。

4.4.4 filter 函数

`filter()`函数用于过滤序列，筛选掉不符合条件的元素，返回符合条件元素。该函数接受两个参数，第一个为函数，第二个为可迭代对象，将对象内的每一个元素作为参数传递给函数进行判定，然后返回 `True` 或 `False`，最后生成一个 `filter` 对象(`python2` 中为一个列表，`python3` 中为迭代器)。

代码:

```
from random import randint #导入随机数模块
#找出偶数
nums = list(range(-6,3))
odds = list(filter(lambda x:x % 2 == 0,nums))
print(nums,odds)
#从集合中找出质数
def find_prime_number(x):
    for i in range(2,int(x/2)): #质数判断
        if x % i == 0:
            return False
        return True
if name == 'main':
    random = set()
    for x in range(0, 15): #生成是个随机数组
        random.add(randint(0, 100))
    print("集合:", random, "长度:", str(len(random)))
    _ = filter(find_prime_number,random)
    print(type(_)) #filter函数生成一个filter对象
    prime_nums = set(_)
    print("质数:", prime_nums)
```

结果:

```
/home/ubuntu/PycharmProjects/Demo/venv/bin/python /home/ubuntu/PycharmPro
[-6, -5, -4, -3, -2, -1, 0, 1, 2] [-6, -4, -2, 0, 2]
集合: {64, 37, 38, 41, 10, 42, 44, 74, 17, 50, 51, 18, 30} 长度: 13
<class 'filter'>
质数: {41, 51, 37, 17}
```

第5章 时间和日期

5.1 datetime 模块

5.1.1 datetime 类型

`datetime` 对象表示一个时间间隔，常用于不同时间单位间的换算。`Datetime` 可直接手工创建实例，也可通过 `datetime`，`date` 等相减得到。

代码:

```
from datetime import timedelta,datetime
a = timedelta(days=2,hours=4)
b = timedelta(hours=5)
c = a - b
print(c,type(c)) # datetime类型可由datetime类型相加减得到
print(c.days,c.total_seconds()) #datetime类型有days,seconds等属性,total_seconds则为所有单位换算为秒的总数值
d = datetime.now()
e = d + a
print(type(e)) # datetime类型与datetime类型相加减得到datetime类型对象
```

结果:


```

/home/ubuntu/PycharmProjects/Demo/venv/bin/python
1 day, 23:00:00 <class 'datetime.timedelta'>
1 169200.0
<class 'datetime.datetime'>

```

5.1.2 date 类型

`date` 对象只能表示日期，精确度最高为天，创建 `date` 实例只需要传递 `year, month, day` 三个参数，`month, day` 默认为 `none`，且 `date` 无 `tzinfo` 属性(时区)

代码:

```

from datetime import date
birthday = date(1998, 1, 5)
today = date.today() #date中today方法根据time.time()返回当前时间信息
print(today)
print(birthday)
delta = today - birthday
print(delta, type(delta)) #date对象相减得到timedelta对象

```

结果:

```

/home/ubuntu/PycharmProjects/Demo/venv/bin/python
2018-04-21
1998-01-05
7411 days, 0:00:00 <class 'datetime.timedelta'>

```

5.1.3 datetime 类型

`datetime` 相较于 `date` 对象，表示的时间更为精确，最高可达 `microsecond`(微秒，百万分之一秒)，除此之外还具备 `tzinfo` 属性，可表示具体时区(默认为 `none`，可通过 `utcnow()` 得到当前的 UTC 时间(世界标准时间))

代码:

```

from datetime import datetime
today = datetime.today() #today方法返回当前时间并精确到微秒
print(today)
utctime = today.utctime()
delta = today - utctime #北京时间一般为utc + 8小时
print(delta, type(delta)) #相减得到timedelta对象

```

结果:

```

/home/ubuntu/PycharmProjects/Demo/venv/bin/python
2018-04-21 21:17:03.984136
7:59:59.999956 <class 'datetime.timedelta'>

```

5.1.4 datetime 的应用

代码

```

#计算类似上周x的日期的问题
from datetime import datetime, timedelta
weekdays = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
             'Saturday', 'Sunday'] #申明星期一个列表
def get_previous_byday(dayname, start_date=None):
    if start_date == None: #若输入的其实时间为空,则通过today()获取当前时间
        start_date = datetime.today()
    day_num = start_date.weekday() #取得起始时间的星期对应的下标(周一~周日-->0~6)
    day_num_target = weekdays.index(dayname) #根据列表通过index()获取相应weekday对应的索引下标
    days_ago = (7 + day_num - day_num_target) % 7 #通过取余的方法来求得今天与上周x相隔的天数
    if days_ago == 0: #若取余得7,正好说明相隔7天
        days_ago = 7
    target_date = start_date - timedelta(days=days_ago) #起始日期减去datetime类型的时间间隔得到上周x的datetime对象
    return target_date #将datetime返回

if __name__ == '__main__':
    today = datetime.today()
    print("今天日期: ", today.strftime('%m-%d,%A')) #按格式输出日期
    last_Monday = get_previous_byday('Monday')
    print("上周一时间: ", last_Monday.strftime('%m-%d,%A'))
    delta = datetime.today() - last_Monday
    print("相隔: ", delta.days, "天") #调用函数,得到日期

```

结果:

```

/home/ubuntu/PycharmProjects/Demo/venv/bin/python
今天日期: 04-22,Sunday
上周一时间: 04-16,Monday
相隔: 6 天

```

5.2 将字符串-日期的转化

5.2.1 datetime 对象转化为想要字符串

若想改变 `datetime` 的输出形式,将其转化为一个字符串,`datetime` 中 `strftime()` 方法通过多种格式化占位符来修改日期格式,常见格式占位符如下:

%Y	完整的四位数字的年份
%y	去掉世纪的年份,两位数字(00-99)
%m	月份(01-12)
%d	一个月的第几天(01-31)
%H	一天中的第几个小时(24小时制,00-23)
%M	分钟数(00-59)
%S	秒数(00-59)
%A	日期对应的星期(英文,Monday等)
%B	日期对应的月份(英文,September等)

代码:

```

from datetime import datetime
today = datetime.today()
print(today)
birthday = datetime(1998,1,5)
print(birthday)
#datetime中的strftime方法将按照给定的占位符将datetime对象转化
today_form = today.strftime('%y-%m-%d,%H:%M,%A')
birthday_form = birthday.strftime('%A %B %d, %Y')
print(today_form,type(today_form)) #strftime返回一个字符串类型
print(birthday_form)

```

结果:


```
/home/ubuntu/PycharmProjects/Demo/venv/bin/python
2018-04-21 22:06:01.146791
1998-01-05 00:00:00
18-04-21,22:06,Saturday <class 'str'>
Monday January 05, 1998
```

5.2.2 将字符串转化为 datetime 对象

对于字符串形式的时间数据处理，常将其转化为 **datetime** 对象。与上文相似，**datetime** 模块中的 **strptime()** 方法，可根据相对应的格式化占位符提取字符中的相应信息，生成 **datetime** 对象。

代码：

```
from datetime import datetime
text = '1998-01-05'
birthday = datetime.strptime(text, '%Y-%m-%d') #将字符串按照其特有的格式将有效信息提取，生成datetime对象
print(birthday, type(birthday))
print(datetime.strftime(birthday, '%A, %B %d, %Y')) #datetime对象转化为字符串类型
```

结果：

```
/home/ubuntu/PycharmProjects/Demo/venv/bin/python
1998-01-05 00:00:00 <class 'datetime.datetime'>
Monday, January 05, 1998
```

注意：方法 **strptime()** 有纯 python 代码编写，效率较低。若在已知时间字符串的准确格式情况下要解析大量的日期，可自行编写函数提取。例如

在“sitka_weather_07-2014.csv”文档中储存有 2014 年每一天的气候信息，如下：

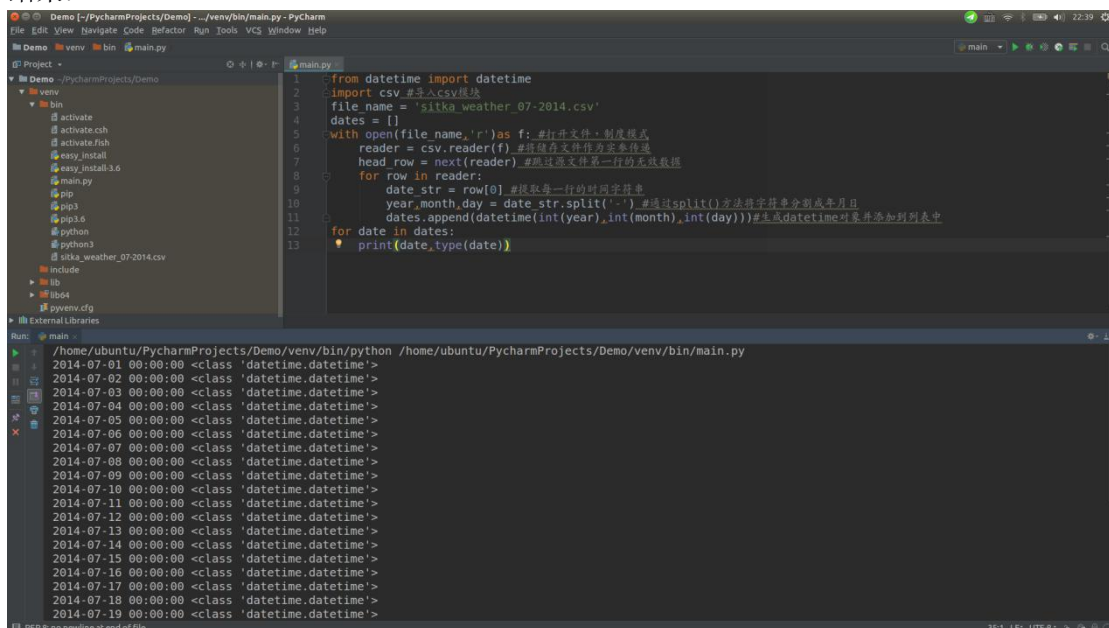
```
sitka_weather_07-2014.csv (~/Desktop/python/Python编程从入门到实践/源代码文件/chapter_16) - gedit
Open Save
AKDT,Max TemperatureF,Mean TemperatureF,Min TemperatureF,Max Dew PointF,MeanDew PointF,Min
DewpointF,Max Humidity, Mean Humidity, Min Humidity, Max Sea Level PressureIn, Mean Sea Level
PressureIn, Min Sea Level PressureIn, Max VisibilityMiles, Mean VisibilityMiles, Min
VisibilityMiles, Max Wind SpeedMPH, Mean Wind SpeedMPH, Max Gust SpeedMPH,PrecipitationIn,
CloudCover, Events, WindDirDegrees
2014-7-1,64,56,50,53,51,48,96,83,58,30.19,30.00,29.79,10,10,10,7,4,,0.00,7,,337
2014-7-2,71,62,55,55,52,46,96,80,51,29.81,29.75,29.66,10,9,2,13,5,,0.14,7,Rain,327
2014-7-3,64,58,53,55,53,51,97,85,72,29.88,29.86,29.81,10,10,8,15,4,,0.01,6,,258
2014-7-4,59,56,52,52,51,50,96,88,75,29.91,29.89,29.87,10,9,2,9,2,,0.07,7,Rain,255
2014-7-5,69,59,50,52,50,46,96,72,49,29.88,29.82,29.79,10,10,10,13,5,,0.00,6,,110
2014-7-6,62,58,55,51,50,46,80,71,58,30.13,30.07,29.89,10,10,10,20,10,29,0.00,6,Rain,213
2014-7-7,61,57,55,56,53,51,96,87,75,30.10,30.07,30.05,10,9,4,16,4,25,0.14,8,Rain,211
2014-7-8,55,54,53,54,53,51,100,94,86,30.10,30.06,30.04,10,6,2,12,5,23,0.84,8,Rain,159
2014-7-9,57,55,53,56,54,52,100,96,83,30.24,30.18,30.11,10,7,2,9,5,,0.13,8,Rain,201
2014-7-10,61,56,53,53,52,51,100,90,75,30.23,30.17,30.03,10,8,2,8,3,,0.03,8,Rain,215
2014-7-11,57,56,54,56,54,51,100,94,84,30.02,30.00,29.98,10,5,2,12,5,,1.28,8,Rain,250
2014-7-12,59,56,55,58,56,55,100,97,93,30.18,30.06,29.99,10,6,2,15,7,26,0.32,8,Rain,275
2014-7-13,57,56,55,58,56,55,100,98,94,30.25,30.22,30.18,10,5,1,8,4,,0.29,8,Rain,291
2014-7-14,61,58,55,58,56,51,100,94,83,30.24,30.23,30.22,10,7,0,16,4,,0.01,8,Fog,307
2014-7-15,64,58,55,53,51,48,93,78,64,30.27,30.25,30.24,10,10,10,17,12,,0.00,6,,318
2014-7-16,61,56,52,51,49,47,89,76,64,30.27,30.23,30.16,10,10,15,6,,0.00,6,,294
2014-7-17,59,55,51,52,50,48,93,84,75,30.16,30.04,29.82,10,10,6,9,3,,0.11,7,Rain,232
2014-7-18,63,56,51,54,52,50,100,84,67,29.79,29.69,29.65,10,10,7,10,5,,0.05,6,Rain,299
2014-7-19,60,57,54,55,53,51,97,88,75,29.91,29.82,29.68,10,9,2,9,2,,0.00,8,,292
2014-7-20,57,55,52,54,52,50,94,89,77,29.92,29.87,29.78,10,8,2,13,4,,0.31,8,Rain,155
2014-7-21,69,60,52,53,51,50,97,77,52,29.99,29.88,29.78,10,10,10,13,4,,0.00,5,,297
2014-7-22,63,59,55,56,54,52,90,84,77,30.11,30.04,29.99,10,10,10,9,3,,0.00,6,Rain,240
2014-7-23,62,58,55,54,52,50,87,80,72,30.10,30.03,29.96,10,10,10,8,3,,0.00,7,,230
2014-7-24,59,57,54,54,52,51,94,84,78,29.95,29.91,29.89,10,9,3,17,4,28,0.06,8,Rain,207
2014-7-25,57,55,53,55,53,51,100,92,81,29.91,29.87,29.83,10,8,2,13,3,,0.53,8,Rain,141
2014-7-26,57,55,53,57,55,54,100,96,93,29.96,29.91,29.87,10,8,1,15,5,24,0.57,8,Rain,216
2014-7-27,61,58,55,55,54,53,100,92,78,30.10,30.05,29.97,10,9,2,13,5,,0.30,8,Rain,213
2014-7-28,59,56,53,57,54,51,97,94,90,30.06,30.00,29.96,10,8,2,9,3,,0.61,8,Rain,261
2014-7-29,61,56,51,54,52,49,96,89,75,30.13,30.02,29.95,10,9,3,14,4,,0.25,6,Rain,153
2014-7-30,61,57,54,55,53,52,97,88,78,30.31,30.23,30.14,10,10,8,8,4,,0.08,7,Rain,160
2014-7-31,66,58,50,55,52,49,100,86,65,30.31,30.29,30.26,10,9,3,10,4,,0.00,3,,217
```

已知每行的日期格式为 YYYY-MM-DD 形式，要提取日期，可编写如下代码：

代码：

```
from datetime import datetime
import csv #导入csv模块
file_name = 'sitka_weather_07-2014.csv'
dates = []
with open(file_name, 'r') as f: #打开文件，制度模式
    reader = csv.reader(f) #将储存文件作为实参传递
    head_row = next(reader) #跳过源文件第一行的无效数据
    for row in reader:
        date_str = row[0] #提取每一行的时间字符串
        year, month, day = date_str.split('-') #通过split()方法将字符串分割成年月日
        dates.append(datetime(int(year), int(month), int(day))) #生成datetime对象并添加到列表中
for date in dates:
    print(date, type(date))
```

结果：



```
2014-07-01 00:00:00 <class 'datetime.datetime'>
2014-07-02 00:00:00 <class 'datetime.datetime'>
2014-07-03 00:00:00 <class 'datetime.datetime'>
2014-07-04 00:00:00 <class 'datetime.datetime'>
2014-07-05 00:00:00 <class 'datetime.datetime'>
2014-07-06 00:00:00 <class 'datetime.datetime'>
2014-07-07 00:00:00 <class 'datetime.datetime'>
2014-07-08 00:00:00 <class 'datetime.datetime'>
2014-07-09 00:00:00 <class 'datetime.datetime'>
2014-07-10 00:00:00 <class 'datetime.datetime'>
2014-07-11 00:00:00 <class 'datetime.datetime'>
2014-07-12 00:00:00 <class 'datetime.datetime'>
2014-07-13 00:00:00 <class 'datetime.datetime'>
2014-07-14 00:00:00 <class 'datetime.datetime'>
2014-07-15 00:00:00 <class 'datetime.datetime'>
2014-07-16 00:00:00 <class 'datetime.datetime'>
2014-07-17 00:00:00 <class 'datetime.datetime'>
2014-07-18 00:00:00 <class 'datetime.datetime'>
2014-07-19 00:00:00 <class 'datetime.datetime'>
```

第 6 章 多线程

6.1 关于 Python 进程

Python 支持多种不同的并发编程方法，包括多进程、加载子进程以及各种涉及生成器函数的技巧。

6.1.1 全局解释器锁

全局解释器锁(简称 GIL)，是计算机程序设计语言解释器用于同步线程的一种机制，它使得任何时刻仅有一个线程在执行。Python 中使用最广泛的是用 C 实现的 CPython 解释器，由于 GIL 的存在，使得 CPython 不能并发编程，即 Python 线程的执行模式被限制在任何时刻只允许解释器运行一个线程。基于此，不宜使用 Python 线程来处理计算密集型的任务，我们尝试在多个 CPU 核心上实行并行处理。Python 线程更适合 I/O 处理以及涉及阻塞操作的并行执行任务。

6.1.2 启动和执行 Python 线程

threading 模块用来在单独的线程中执行的 Python 可调用对象，常创建一个 Thread 实例并为他提供期望执行的可调用对象。

代码：

```

from threading import Thread
from random import randint
import time
def countdown(m):
    n = randint(1,10) #随机生成睡眠时间
    print("进程{}开始执行".format(m))
    while n > 0:
        print('进程{0}正在执行,剩余时间:{1}'.format(m,n))
        n -= 1
        time.sleep(1) #每执行一次循环,休眠一秒
    print("进程{}结束.".format(m))

t1 = Thread(target=countdown,args=(1,)) #建立一个Thread实例,跟踪目标为countdown函数,关键词args接受元组传递参数到函数
t1.start() #当调用start()方法,线程立即开始执行
t2 = Thread(target=countdown,args=(2,))

if t2.is_alive(): #通过is_alive()方法判断线程是否在执行
    print('Alive.')
else:
    print("Has not started.")
t2.start() #只有调用start()方法,Thread才正式开始执行
t2.join() #将t2加入到进程中

```

结果:

```

/home/ubuntu/PycharmProjects/Demo/venv/bin/python
进程1开始执行
Has not started.
进程2开始执行
进程1正在执行,剩余时间:4
进程2正在执行,剩余时间:9
进程1正在执行,剩余时间:3
进程2正在执行,剩余时间:8
进程1正在执行,剩余时间:2
进程2正在执行,剩余时间:7
进程1正在执行,剩余时间:1
进程2正在执行,剩余时间:6
进程1结束.
进程2正在执行,剩余时间:5
进程2正在执行,剩余时间:4
进程2正在执行,剩余时间:3
进程2正在执行,剩余时间:2
进程2正在执行,剩余时间:1
进程2结束.

```

6.1.3 判断进程是否启动

线程由于其不确定性的方式(何时开始执行,何时被打断,何时恢复执行),线程执行完全由操作系统调度。如果一个程序需判断其他线程是否都打执行过程的某个点,根据这个判断来执行后续的操作,就产生了线程同步的问题

二 Mysql

实验时使用的数据表如下:

Tables


```
mysql> use work;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----+
| Tables_in_work |
+-----+
| Course          |
| SC              |
| Student         |
+-----+
3 rows in set (0.00 sec)
```

Table1 Student

```
mysql> desc Student;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| Sno   | char(9)       | NO   | PRI | NULL    |       |
| Sname | char(20)      | YES  | UNI | NULL    |       |
| Ssex  | char(2)       | YES  |     | NULL    |       |
| Sage  | smallint(6)   | YES  |     | NULL    |       |
| Sdept | char(20)      | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)

mysql> select * from Student;
+-----+-----+-----+-----+-----+
| Sno   | Sname | Ssex | Sage | Sdept |
+-----+-----+-----+-----+-----+
| 201215121 | 李勇  | 男   | 20   | CS    |
| 201215122 | 刘晨  | 女   | 19   | CS    |
| 201215123 | 王敏  | 女   | 18   | MA    |
| 201215125 | 张立  | 男   | 19   | IS    |
+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

Table2 Course

```
mysql> desc Course;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| Cno   | char(4)       | NO   | PRI | NULL    |       |
| Cname | char(40)      | NO   |     | NULL    |       |
| Cpno  | char(4)       | YES  | MUL | NULL    |       |
| Credit | smallint(6)  | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> select * from Course;
+-----+-----+-----+-----+
| Cno | Cname          | Cpno | Credit |
+-----+-----+-----+-----+
| 1   | 数据库         | 5    | 4      |
| 2   | 数学           | NULL | 2      |
| 3   | 信息系统       | 1    | 4      |
| 4   | 操作系统       | 7    | 3      |
| 5   | 数据结构       | 7    | 4      |
| 6   | 数据处理       | NULL | 2      |
| 7   | PASCAL语言     | 6    | 4      |
+-----+-----+-----+-----+
7 rows in set (0.00 sec)
```

Table3 SC

```
mysql> desc SC;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| Sno   | char(9)       | NO   | PRI | NULL    |       |
| Cno   | char(4)       | NO   | PRI | NULL    |       |
| grade | smallint(6)  | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> select * from SC;
+-----+-----+-----+
| Sno          | Cno | grade |
+-----+-----+-----+
| 201215121   | 1   | 92    |
| 201215121   | 2   | 85    |
| 201215121   | 3   | 88    |
| 201215122   | 2   | 90    |
| 201215122   | 3   | 80    |
+-----+-----+-----+
5 rows in set (0.00 sec)
```

第 1 章 表连接

当查询同时涉及到两个以上表时，称之为连接查询。链接查询为关系数据库中最主要的查询方式，包括内连接(等值连接、非等值连接，自然连接等)，外连接，和交叉连接。

1.1 内连接

内连接时典型的连接运算使用=或<>之类的比较运算符，内连接通常使用比较运算符根据每个表共有的列的值匹配表的行。内连接包括(等值连接查询、非等值连接查询，自然连接查询)。当比较运算符为‘=’时为等值连接，其他则为非等值连接查询。若在等值连接中把目标列中的重复属性列去掉则为自然连接。

将 Student 表与 SC 表等值连接，MySQL 语句如下：

```
mysql> select Student.*,SC.*
-> from Student,SC
-> where Student.Sno = SC.Sno;
```

Sno	Sname	Ssex	Sage	Sdept	Sno	Cno	grade
201215121	李勇	男	20	CS	201215121	1	92
201215121	李勇	男	20	CS	201215121	2	85
201215121	李勇	男	20	CS	201215121	3	88
201215122	刘晨	女	19	CS	201215122	2	90
201215122	刘晨	女	19	CS	201215122	3	80

5 rows in set (0.02 sec)

考虑到连接表的有重复的 Sno 属性，去除则为自然连接，mysql 语句如下：

```
mysql> select *
-> from Student natural join SC;
```

Sno	Sname	Ssex	Sage	Sdept	Cno	grade
201215121	李勇	男	20	CS	1	92
201215121	李勇	男	20	CS	2	85
201215121	李勇	男	20	CS	3	88
201215122	刘晨	女	19	CS	2	90
201215122	刘晨	女	19	CS	3	80

5 rows in set (0.00 sec)

例如查询选修了 1 号课程且成绩在 90 分以上的所有学生的名字与学号，MySQL 语句如下：

```
mysql> select Student.Sno,Sname
-> from Student,SC
-> where Student.Sno=SC.Sno and SC.Cno='1' and SC.Grade>90;
```

Sno	Sname
201215121	李勇

1 row in set (0.00 sec)

1.2 外连接

外连接不要求连接的两表的每一条记录在对方表中都有一条匹配记录。要保留所有的记录(甚至没有匹配的记录的)的表称为保留表。外连接可依据连接表保留左表，右表或全部表的行而进一步分为左外连接，右外连接和全连接。

1.2.1 左外连接

左外连接又称为左连接，若 A 和 B 两表进行左外连接，那么结果表中将包含"左表"(即表 A)的所有记录，即使那些记录在"右表" B 没有匹配连接条件的匹配。这意味着即使 ON 语句在 B 中的匹配项是 0 条，连接操作还是会返回一条记录，只不过这条记录中来自于 B 的每一列的值都为 NULL。这意味着左外连接会返回左表的所有记录和右表中匹配记录的组合(如果右表中无匹配记录，来自于右表的所有列的值设为 NULL)。如果左表的一行在右表中存在多个匹配行，那么左表的行会复制和右表匹配行

一样的数量, 并进行组合生成连接结果.

例如, 若想以 **Student** 表中的学生为主体列出每个学生的选课情况. 当某学生存在于 **Student** 表却在 **SC** 表中没有选课记录时, 仍想保留该学生并在对应的 **SC** 属性列中填充 **Null**, 此时表便可使用左连接. **MySQL** 语句如下:

```
mysql> select Student.Sno,Sname,Ssex,Sage,Sdept,Cno,Grade
-> from Student left join SC
-> on Student.Sno = SC.Sno;
```

Sno	Sname	Ssex	Sage	Sdept	Cno	Grade
201215121	李勇	男	20	CS	1	92
201215121	李勇	男	20	CS	2	85
201215121	李勇	男	20	CS	3	88
201215122	刘晨	女	19	CS	2	90
201215122	刘晨	女	19	CS	3	80
201215123	王敏	女	18	MA	NULL	NULL
201215125	张立	男	19	IS	NULL	NULL

7 rows in set (0.05 sec)

也可使用 **using** 来去除结果中的重复值, 如下:

```
mysql> select Student.Sno,Sname,Ssex,Sage,Sdept,Cno,Grade
-> from Student left join SC
-> using (Sno);
```

Sno	Sname	Ssex	Sage	Sdept	Cno	Grade
201215121	李勇	男	20	CS	1	92
201215121	李勇	男	20	CS	2	85
201215121	李勇	男	20	CS	3	88
201215122	刘晨	女	19	CS	2	90
201215122	刘晨	女	19	CS	3	80
201215123	王敏	女	18	MA	NULL	NULL
201215125	张立	男	19	IS	NULL	NULL

7 rows in set (0.00 sec)

1.2.2 右外连接

右外连接, 亦简称右连接, 它与左外连接完全类似, 只不过是作连接的表的顺序相反而已. 如果 **A** 表右连接 **B** 表, 那么"右表" **B** 中的每一行在连接表中至少会出现一次. 如果 **B** 表的记录在"左表" **A** 中未找到匹配行, 连接表中来源于 **A** 的列的值设为 **NULL**. 右连接操作返回右表的所有行和这些行在左表中匹配的行(没有匹配的, 来源于左表的列值设为 **NULL**).

例如以 **SC** 表为主想查询每个课程表的情况. 但是不是每个课程都有人选, 若仍想保留未被选择的课程信息, 在对应的 **SC** 表中以 **Null** 填充, 可使用右连接 **MySQL** 语句如下:

```
mysql> select SC.Cno,Sno,Grade,Cname,Cpno,Credit
-> from SC right join Course
-> using (Cno);
```

Cno	Sno	Grade	Cname	Cpno	Credit
1	201215121	92	数据库	5	4
2	201215121	85	数学	NULL	2
3	201215121	88	信息系统	1	4
2	201215122	90	数学	NULL	2
3	201215122	80	信息系统	1	4
NULL	NULL	NULL	操作系统	7	3
NULL	NULL	NULL	数据结构	7	4
NULL	NULL	NULL	数据处理	NULL	2
NULL	NULL	NULL	PASCAL语言	6	4

```
9 rows in set (0.00 sec)
```

右连接与做链接可以相互转换，上述右连接等效于下：

```
mysql> select SC.Cno,Sno,Grade,Cname,Cpno,Credit
-> from Course left join SC
-> using (Cno);
```

Cno	Sno	Grade	Cname	Cpno	Credit
1	201215121	92	数据库	5	4
2	201215121	85	数学	NULL	2
3	201215121	88	信息系统	1	4
2	201215122	90	数学	NULL	2
3	201215122	80	信息系统	1	4
NULL	NULL	NULL	操作系统	7	3
NULL	NULL	NULL	数据结构	7	4
NULL	NULL	NULL	数据处理	NULL	2
NULL	NULL	NULL	PASCAL语言	6	4

```
9 rows in set (0.00 sec)
```

1.2.3 全连接

全连接是左右外连接的并集，连接表包含被连接的表的所有记录，如果缺少匹配的记录，即以 NULL 填充。

例如查询每个课程的被选课情况，同时查看没有被选上的，以及选上了但是课程表里没有的课程。可用 MySQL 语句：


```
mysql> select SC.Cno,Sno,Grade,Cname,Cpno,Credit
-> from Course full join SC
-> using (Cno);
```

Cno	Sno	Grade	Cname	Cpno	Credit
1	201215121	92	数据库	5	4
2	201215121	85	数学	NULL	2
3	201215121	88	信息系统	1	4
2	201215122	90	数学	NULL	2
3	201215122	80	信息系统	1	4

```
5 rows in set (0.00 sec)
```

1.3 交叉连接

交叉连接(cross join)，又称笛卡尔连接(cartesian join)或叉乘(Product)，它是所有类型的内连接的基础。把表视为行记录的集合，交叉连接即返回这两个集合的笛卡尔积。用于交叉连接的 SQL 代码在 FROM 列出表名，但并不包含任何过滤的连接谓词。例如交叉连接 Student 表与 SC 表，MySQL 语句如下

```
mysql> select *
-> from Student cross join SC;
```

Sno	Sname	Ssex	Sage	Sdept	Sno	Cno	grade
201215121	李勇	男	20	CS	201215121	1	92
201215122	刘晨	女	19	CS	201215121	1	92
201215123	王敏	女	18	MA	201215121	1	92
201215125	张立	男	19	IS	201215121	1	92
201215121	李勇	男	20	CS	201215121	2	85
201215122	刘晨	女	19	CS	201215121	2	85
201215123	王敏	女	18	MA	201215121	2	85
201215125	张立	男	19	IS	201215121	2	85
201215121	李勇	男	20	CS	201215121	3	88
201215122	刘晨	女	19	CS	201215121	3	88
201215123	王敏	女	18	MA	201215121	3	88
201215125	张立	男	19	IS	201215121	3	88
201215121	李勇	男	20	CS	201215122	2	90
201215122	刘晨	女	19	CS	201215122	2	90
201215123	王敏	女	18	MA	201215122	2	90
201215125	张立	男	19	IS	201215122	2	90
201215121	李勇	男	20	CS	201215122	3	80
201215122	刘晨	女	19	CS	201215122	3	80
201215123	王敏	女	18	MA	201215122	3	80
201215125	张立	男	19	IS	201215122	3	80

```
20 rows in set (0.00 sec)
```

第 2 章 聚合函数

聚合函数对一组值执行计算，并返回单个值。聚合函数通常与 select 语句与 group by 子句一起使用。

2.1 COUNT()函数

COUNT()函数统计元组的个数，例如查询 Student 表的学生个数，MySQL 语句如下：


```

+-----+
| count(*) |
+-----+
|         4 |
+-----+
1 row in set (0.00 sec)

```

统计 Course 表的课程数目，MySQL 语句如下：

```

mysql> select count(*) from Course;
+-----+
| count(*) |
+-----+
|         7 |
+-----+
1 row in set (0.00 sec)

```

2.2 AVG()函数

AVG()函数统计一列值的平均值(此列必须为数值型)。例如统计 SC 表中每门课程的平均分，MySQL 语句如下：

```

mysql> select Cno,avg(Grade)
-> from SC
-> group by Cno;
+-----+-----+
| Cno | avg(Grade) |
+-----+-----+
| 1   | 92.0000    |
| 2   | 87.5000    |
| 3   | 84.0000    |
+-----+-----+
3 rows in set (0.00 sec)

```

2.3 SUM()函数

SUM()函数计算一列值的总和(此列必须为数值型)，例如查询学号为 201215121 的学生选修课程的总学分数，MySQL 语句如下：

```

mysql> select Sno,sum(Credit)
-> from SC,Course
-> where Sno='201215121' and SC.Cno = Course.Cno;
+-----+-----+
| Sno      | sum(Credit) |
+-----+-----+
| 201215121 |          10 |
+-----+-----+
1 row in set (0.00 sec)

```

2.4 MAX()函数与 MIN()函数

MAX()函数与 MIN()函数分别计算某列中的最大值与最小值，例如选出 SC 表中每门课程最高分或最低分，MySQL 语句如下：

每门课程最高分

```
mysql> select Student.Sno,Sname,Cno,Grade as Top_Grade
-> from Student,SC
-> where Student.Sno=SC.Sno and Grade in
-> (select max(Grade)
-> from SC
-> group by Cno)
-> order by Cno;
```

Sno	Sname	Cno	Top_Grade
201215121	李勇	1	92
201215122	刘晨	2	90
201215121	李勇	3	88

3 rows in set (0.00 sec)

每门课程最低分

```
mysql> select Student.Sno,Sname,Cno,Grade as lowest_Grade
-> from Student,SC
-> where Student.Sno=SC.Sno and Grade in
-> (select min(Grade)
-> from SC
-> group by Cno)
-> order by Cno;
```

Sno	Sname	Cno	lowest_Grade
201215121	李勇	1	92
201215121	李勇	2	85
201215122	刘晨	3	80

3 rows in set (0.01 sec)

总结: 1.SQL 语句中不能在 **where** 子句中使用聚集函数作为条件表达式, 聚集函数只能用于 **select** 子句和 **group by** 中的 **having** 子句。这是因为 **where** 子句与 **having** 子句的作用对象不同, **where** 子句作用于基本表或视图, 从中选择满足条件的元组。**Having** 短语作用于组, 从中选择满足条件的组。2.当聚集函数遇到空值时, 除 **COUNT(*)**外, 都跳过空值而处理非空值。

第3章 子查询

在 SQL 语言中, 一个 **SELECT-FROM-WHERE** 语句被称为一个查询块。将一个查询块嵌套在另一个查询块的 **WHERE** 子句或 **HAVING** 短语中的条件称之为嵌套查询。其中上层的查询块称为外层查询或父查询, 下层查询块被曾为内层查询或子查询。

3.1 带有 IN 谓词的子查询

在嵌套查询中, 若子查询的结果为一个集合, 则往往使用 **IN** 谓词做连接两个查询块。例如在 2.4 中欲查询每门学科的最高分, 通过子查询 **group by** 将元组按照课程号分组, 并 **select** 出其中每组的最高分, 得到一个高分组集合, 在外层查询块 **where** 中根据判断 **Grade** 是否在这个高分组中, 再做表连接, 最终得到查询结果:

```
mysql> select Student.Sno,Sname,Cno,Grade as Top_Grade
-> from Student,SC
-> where Student.Sno=SC.Sno and Grade in
-> (select max(Grade)
-> from SC
-> group by Cno)
-> order by Cno;
```

Sno	Sname	Cno	Top_Grade
201215121	李勇	1	92
201215122	刘晨	2	90
201215121	李勇	3	88

3 rows in set (0.00 sec)

3.2 带有比较运算符的子查询

带有比较运算符的子查询是父查询与子查询之间用比较运算符之间进行连接。其中子查询返回的时单个值，可以比较运算符进行条件判断。例如找出每个学生低于自己选修课程的平均分的课程。MySQL 语句如下：

```
mysql> select Student.Sno,Sname,Cno
-> from Student,SC x
-> where Student.Sno=x.Sno and Grade <
-> (select avg(Grade)
-> from SC y
-> where y.Sno = x.Sno);
```

Sno	Sname	Cno
201215121	李勇	2
201215121	李勇	3
201215122	刘晨	3

3 rows in set (0.00 sec)

3.3 带有 ANY(SOME)或 ALL 谓词的子查询

子查询若返回单值可以使用比较运算符，但返回多值要用使用 ANY(有些系统为 SOME)或 ALL 等谓词修饰词。也就是说使用 ANY 或 ALL 必须同时使用比较运算符。其语义如下：

>ANY	大于子查询结果中某个值
>=ANY	大于等于子查询结果中某个值
=ANY	等于子查询结果中某个值
<ANY	小于子查询结果中某个值
<=ANY	小于等于子查询结果中某个值
>ALL	大于子查询结果中所有值
>=ALL	大于等于子查询结果中所有值
=ALL	等于子查询结果中所有值
<ALL	小于子查询结果中所有值
<=ALL	小于等于子查询结果中所有值
!=(或<>)ANY	不等于子查询中的某个值
!=(或<>)ALL	不等于子查询中的任意一个值

例如查询非计算机系中的比计算机系中任意一个学生年龄小的学生姓名和年龄。
MySQL 语句如下：

```
mysql> select Sname,Sage
-> from Student
-> where Sage<ANY(
-> select Sage
-> from Student
-> where Sdept='CS')
-> and Sdept <>'CS';
+-----+-----+
| Sname | Sage |
+-----+-----+
| 王敏  | 18   |
| 张立  | 19   |
+-----+-----+
2 rows in set (0.00 sec)
```

又如查询非计算机系中比计算机系所有年龄都小的学生姓名和年龄，MySQL 语句如下：

```
mysql> select Sname,Sage
-> from Student
-> where Sage<ALL(
-> select Sage
-> from Student
-> where Sdept='CS')
-> and Sdept <>'CS';
+-----+-----+
| Sname | Sage |
+-----+-----+
| 王敏  | 18   |
+-----+-----+
1 row in set (0.00 sec)
```

3.4 带有 EXISTS 谓词的子查询

EXISTS 代表存在量词 \exists 。带有 EXISTS 谓词的子查询返回查询数据，只产生逻辑真值 true 和逻辑假值 false。利用 EXISTS 可以判断 $x \in S$, $S \cap R$, $S=R$ 非空等是否正确。例如欲查找所有选修了 1 号课程的学生的姓名。MySQL 代码如下：


```
mysql> select Sname,Sno  
-> from Student  
-> where exists(  
-> select *  
-> from SC  
-> where Sno=Student.Sno and Cno='1');
```

```
+-----+-----+  
| Sname | Sno      |  
+-----+-----+  
| 李勇   | 201215121 |  
+-----+-----+  
1 row in set (0.00 sec)
```