### 6内置函数及操作

## 6.1字符串的相关操作

#### 6.1.1索引与切片

在python中，一个字符串是一个字符序列，可以通过索引的方法访问字符串中的一个字符：

str1[index]

下标索引是基于0的，范围0~len(str1)-1

str1 = 'abcdefg'  
for i in range(0, len(str1) - 1):  
 print('索引：', i, '字符：', str1[i])

运行结果：

索引： 0 字符： a  
索引： 1 字符： b  
索引： 2 字符： c  
索引： 3 字符： d  
索引： 4 字符： e  
索引： 5 字符： f

python也可以使用负数作为索引，它是相对于字符串最后一个字母的位置

str1 = 'abcdefg'  
print(str1[-1])  
print(str1[-2])

运行结果：

g  
f

切片操作

列表可以允许选择下标上元素的值从而用于截取操作str1[start:end]，返回列表中的片段值

str1 = 'hello world'  
print(str1[:6])  
print(str1[2:])  
print(str1[5:7])  
print(str1[::-1])

运行结果：

hello   
llo world  
 w  
dlrow olleh

#### 6.1.2字符串函数

|  |  |
| --- | --- |
| 函数 | 作用 |
| capitalize() | 字符串第一个字母大写 |
| upper() | 字符串所有字母大写 |
| lower() | 字符串所有字母小写 |
| swapcase() | 字符串大小写互换 |
| replace(old,new) | 使用新字符串替换旧字符串所有出现的位置 |
| find(s1) | 返回s1这这个字符串中的下标，不存在返回-1 |
| count(s1) | 返回s1在字符串中的次数 |
| isalnum() | 该字符串中字符是字母数字且至少有一个字符，返回True |
| isupper() | 该字符串中所有字符为大写且至少有一个字符返回True |
| isspace() | 如果该字符串中包含空格返回True |
| endswith(s1) | 该字符串以s1结尾返回True |
| startswith(s1) | 该字符串以s1开始返回True |
| lstrip() | 返回去掉前端空格的字符串 |
| rstrip() | 返回去掉后端空格的字符串 |
| strip() | 返回去掉两端空格的字符串 |
| split(s1) | 通过指定分隔符s1对字符串进行切片，返回为切片后的字符列表 |

# 首字母大写  
test\_1 = 'this Is test code '  
test\_1=test\_1.capitalize()  
print(test\_1)  
# 每一个单词首字母大写  
test\_1 = 'this Is test code '  
test\_1=test\_1.title()  
print(test\_1)  
# 所有单词大写  
test\_1 = 'this Is test code '  
test\_1=test\_1.upper()  
print(test\_1)  
# 小写  
test\_1 = 'this Is test code '  
test\_1=test\_1.lower()  
print(test\_1)  
# 大小写互换  
test\_1 = 'this Is test code '  
test\_1=test\_1.swapcase()  
print(test\_1)

运行结果：

This is test code   
This Is Test Code   
THIS IS TEST CODE   
this is test code   
THIS iS TEST CODE

# 统计某个字符串出现的次数(词频)  
str = '午夜笛 笛声残 偷偷透 透过窗'  
str1 = str.count('笛')  
print(str1)  
# 查找 当前词汇第一个字出现的位置  
str = '午夜笛 笛声残 偷偷透 透过窗'  
str1 = str.find('偷')  
print(str1)  
# 检测是否以指定的字符开头 endswith 结尾  
str = '午夜笛 笛声残 偷偷透 透过窗'  
str1 = str.startswith('午')  
print(str1)

运行结果：

2  
8  
True

# 检测都是大写  
str = 'abcdABCD'  
str1 = str.isupper()  
print(str1)  
# 检测字符串是否由数字，字母文字等组成 isalpha 字符  
test\_3 = ' !@#$%^&\*'  
str1 = test\_3.isalnum()  
print(str1)  
  
# 检测字符串是否由空白字符组成  
test\_4 = '\n\r'  
str1 = test\_4.isspace()  
print(str1)  
  
# 字符串的切割split() 返回值是列表  
str = '1234\_55\n55\_98766'  
str1 = str.split('\_')  
print(str1)

运行结果：

False  
False  
True  
['1234', '55\n55', '98766']

# 去掉两侧空格  
html = ' <span class="pl">又名:</span>&nbsp;Death And All His Friends '  
str1 = html.strip()  
print(str1)

运行结果：

<span class="pl">又名:</span>&nbsp;Death And All His Friends

#### 6.1.3Format函数

python存在格式化的函数format()，它增强了字符串的格式化功能，使用{}代替之前的%

# 不设置指定位置，按默认顺序  
info = '我叫{},今年{}，我来自{}'.format('张三', '22', '哈尔滨')  
print(info)

运行结果：

我叫张三,今年22，我来自哈尔滨

# 设置指定位置  
name = '李四'  
age = 22  
where = '哈尔滨'  
info = '我叫{0},今年{2}，我来自{1}'.format(name, age, where)  
print(info)

运行结果：

我叫李四,今年哈尔滨，我来自22

# 通过关键字的方式传递参数  
name = '李四'  
age = 22  
where = '哈尔滨'  
info = '我叫{name},今年{age}，我来自{where}'.format(name='王五', age='23', where='哈尔滨')  
print(info)

运行结果：

我叫王五,今年23，我来自哈尔滨

# 通过下标的方式传递参数  
info = ['张三', 20, '哈尔滨学院']  
info\_str = '{0[0]},{0[1]},{0[2]}'.format(info)  
print(info\_str)

运行结果：

张三,20,哈尔滨学院

也可以用来格式化数字

# 精度与类型f  
info = '{:.2f}'.format(3.1495926)  
print(info)  
# 进制类型  
# b--2 d--10 o--8 x--16  
info = '{:b}'.format(17)  
print(info)  
info = '{:,}'.format(200000000)  
print(info)

运行结果：

3.15  
10001  
200,000,000

## 6.2列表操作

#### 6.2.1索引与切片

列表中元素可以使用索引来访问：

list[index]

列表元素索引是从0开始，范围为0~len(list)-1

list1 = [1, 2, 3, 4, 5, 6, 7, 8, 9]  
# 取出第一个值  
print(list1[0])  
# 取出第二个值  
print(list1[1])  
# 取出第四个个值  
print(list1[3])

运行结果：

1  
2  
4

切片操作

列表可以允许选择下标上元素的值从而用于截取操作list[start:end]，返回列表中的片段值

list1 = [1, 2, 3, 4, 5, 'a', 'b', 'c']  
# 获取整个列表  
print(list1[:])  
# 获取列表开头到结束索引之前  
print(list1[:6])  
# 获取开始索引到列表末尾的数据组成的列表  
print(list1[0:])  
# 获取开始索引和结束索引之间的数据组成的列表  
print(list1[2:7])  
# 获取开始索引和结束索引之间的数据按照间隔值获取  
print(list1[0:9:2])

运算结果：

[1, 2, 3, 4, 5, 'a', 'b', 'c']  
[1, 2, 3, 4, 5, 'a']  
[1, 2, 3, 4, 5, 'a', 'b', 'c']  
[3, 4, 5, 'a', 'b']  
[1, 3, 5, 'b']

如果开始值大于结束值（start>=end），那么list[start:end]返回一个空值，若是end>len(list)

则会使用该列表长度代替end

# start > end  
print(list1[4:2])  
# end > len(list1)  
print(list1[5:9])

运行结果：

[]  
[6, 7]

#### 6.2.2创建列表

list类定义列表，可以用来创建列表

# 创建列表  
list1 = list() # 创建一个空列表  
list2 = list([1, 2, 3, 4]) # 整形列表  
list3 = list(['a', 'b', 'c', 'd']) # 字符串列表  
list4 = list(range(1, 5))  
list5 = list('abcd')

运行结果：

[]  
[1, 2, 3, 4]  
['a', 'b', 'c', 'd']  
[1, 2, 3, 4]  
['a', 'b', 'c', 'd']

也可以使用简便方法

list1 = []  
list2 = [1, 2, 3, 4]  
list3 = ['a', 'b', 'c', 'd']

列表中元素使用逗号隔开

元素中既可以存在相同类型的元素，也可以存在不同类型的函数

list1 = [1, 2, 'a', 'b']

运行结果

[1, 2, 'a', 'b']

#### 6.2.3普通操作

使用 + 、\* 和 in 、not in 可以分别对列表和列表元素进行操作，

+，可以用来组合列表

list1 = [1, 2]  
list2 = [3, 4]  
list3 = list1 + list2

运行结果：

[1, 2, 3, 4]

\*，可以对列表中元素进行复制

list4 = list1 \* 3

运行结果：

[1, 2, 1, 2, 1, 2]

list3 是对list1 和list2 的连接，成为一个新的列表，list4 是将list1 中的元素复制3次

in 和 not in 用来判断值是不是在列表中，返回的是布尔值

list1 = [1, 4, 9, 5, 7, 'a', 'r', 's', 'b']  
2 in list1  
Out[4]: False  
2 not in list1  
Out[5]: True  
'a' in list1  
Out[7]: True  
'a' not in list1  
Out[8]: False

使用python内嵌函数可以用于列表操作

len函数返回列表元素个数

max/min分别返回列表的最大值最小值

sum函数返回列表元素之和

random.shuffle函数随意排列列表中元素

list1 = [1, 2, 3, 4, 5, 6, 7, 8, 9]  
print(len(list1))  
print(max(list1), min(list1))  
print(sum(list1))  
random.shuffle(list1)  
print(list1)

运行结果：

9  
9 1  
45  
[2, 4, 3, 9, 7, 8, 5, 6, 1]

#### 6.2.4列表遍历

python列表元素是可迭代的，可以使用for循环，存在两种方式，一使用列表的索引，二直接取元素值

list1 = [1, 2, 3, 4, 5, 6, 7, 8, 9]  
for i in range(0, len(list1)):  
 print('下标：', i, '元素值：', list1[i])  
print('------------------')  
for u in list1:  
 print('元素值:', u)

运行结果：

下标： 0 元素值： 1  
下标： 1 元素值： 2  
下标： 2 元素值： 3  
下标： 3 元素值： 4  
下标： 4 元素值： 5  
下标： 5 元素值： 6  
下标： 6 元素值： 7  
下标： 7 元素值： 8  
下标： 8 元素值： 9  
------------------  
元素值: 1  
元素值: 2  
元素值: 3  
元素值: 4  
元素值: 5  
元素值: 6  
元素值: 7  
元素值: 8  
元素值: 9

#### 6.2.5列表的相关函数

|  |  |
| --- | --- |
| 列表函数 | 作用 |
| append(x:object) | 把元素x添加到列表结尾 |
| count(x:object) | 返回列表中x的个数 |
| extend(L:list) | 把L列表中所有元素追加到列表中 |
| index(x:object) | 返回x第一次出现在列表中的下标 |
| insert(index:int,x:object) | 将元素x插入到列表指定下标处 |
| pop(i) | 删除给定位置处元素，并返回它，i可选，无指定，那么删除最后一个元素 |
| remove(x:object) | 删除列表第一次出现的x |
| reverse() | 将列表所有元素倒序 |
| sort() | 对列表元升序排序 |

list1 = [1, 2, 1, 4, 5, 6, 7, 8, 9]  
list1.append(10)  
list1  
Out[4]: [1, 2, 1, 4, 5, 6, 7, 8, 9, 10]  
list1.count(1)  
Out[5]: 2  
list2=[11,12]  
list1.extend(list2)  
list1  
Out[8]: [1, 2, 1, 4, 5, 6, 7, 8, 9, 10, 11, 12]  
list1.index(4)  
Out[9]: 3  
list1.insert(2,3)  
list1  
Out[11]: [1, 2, 3, 1, 4, 5, 6, 7, 8, 9, 10, 11, 12]  
list1.pop(3)  
Out[12]: 1  
list1.pop()  
Out[13]: 12  
list1.reverse()  
list1  
Out[15]: [11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]  
list1.sort()  
list1  
Out[17]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

#### 6.2.6列表推导式

列表提供一种创建顺序元素列表的简便方法，一个列表推导式由两个及两个以上的方括号组成，方括号内包含一个for 表达式或if。通俗来说，推导式可以是表达式求值的结果所组成的列表。

# 列表内涵/列表的推导式  
book = ['水浒传', '西游记']  
books = ['《' + i + '》' for i in book]  
print(books)  
  
# 带有判断条件的列表推导式  
books = ['《' + i + '》' for i in book if i == '水浒传'] # 三部分 i：变量处理 for：遍历 if：判断  
print(books)  
  
book = ['水浒传', '西游记']  
writer = ['施耐庵', '吴承恩']  
# 多个例如同时循环的列表推导式 组合 4个  
book\_info = [b + ':' + w for b in book for w in writer if b == '水浒传']  
print(book\_info)

运行结果：

['《水浒传》', '《西游记》']  
['《水浒传》']  
['水浒传:施耐庵', '水浒传:吴承恩']

## 6.3元组操作

#### 6.3.1创建元组

在应用中不对列表中的内容进行修改，那么可以使用元组来防止元素被意外的增删改，除了元组中的元素是固定的，元组跟列表很像

t1 = ()  
t2 = (1, 2, 3)  
t3 = tuple([2 \* i for i in range(1, 4)])  
t4 = tuple('abcdef')  
print(t1)  
print(t2)  
print(t3)  
print(t4)

运行结果：

()  
(1, 2, 3)  
(2, 4, 6)  
('a', 'b', 'c', 'd', 'e', 'f')

#### 6.3.2基本操作

使用 + 、\* 和 in 、not in 可以分别对元组和列元组元素进行操作，

+，可以用来组合元组

t1 = (1, 2)  
t2 = (3, 4)  
t3 = t1 + t2  
print(t3)

运行结果：

*(1, 2, 3, 4)*

\*，可以对元组中元素进行复制

t4 = t1 \* 3  
print(t4)

运行结果：

*(1, 2, 1, 2, 1, 2)*

t3 是对t1 和t2 的连接，成为一个新的元组，t4 是将t1 中的元素复制3次

in 和 not in 用来判断值是不是在元组中，返回的是布尔值

t1 = (1, 4, 9, 5, 7, 'a', 'r', 's', 'b')  
2 in t1  
Out[4]: False  
2 not in t1  
Out[5]: True  
'a' in t1  
Out[7]: True  
'a' not in t1  
Out[8]: False

使用python内嵌函数可以用于元组操作

len函数返回元组元素个数

max/min分别返回元组的最大值最小值

sum函数返回元组元素之和

t1 = (1, 2, 3, 4, 5, 6, 7, 8, 9)  
print(len(t1))  
print(max(t1), min(t1))  
print(sum(t1))

运行结果：

9  
9 1  
45

#### 6.3.3序列操作

#### 6.3.4元组函数

元组中的元素是固定的所以不能增删改，列表中的函数基本不可取

#### 6.3.5元组遍历

python元组元素是可迭代的，可以使用for循环，存在两种方式，一使用元组的索引，二直接取元素值

t1 = (1, 2, 3, 4, 5, 6, 7, 8, 9)  
for i in range(0, len(t1)):  
 print('下标：', i, '元素值：', t1[i])  
print('------------------')  
for u in t1:  
 print('元素值:', u)

运行结果：

下标： 0 元素值： 1  
下标： 1 元素值： 2  
下标： 2 元素值： 3  
下标： 3 元素值： 4  
下标： 4 元素值： 5  
下标： 5 元素值： 6  
下标： 6 元素值： 7  
下标： 7 元素值： 8  
下标： 8 元素值： 9  
------------------  
元素值: 1  
元素值: 2  
元素值: 3  
元素值: 4  
元素值: 5  
元素值: 6  
元素值: 7  
元素值: 8  
元素值: 9

#### 6.3.6元组推导式

元组推导式与列表相同，只是方括号变成圆括号，另外元组推导式返回的结果是生成器对象

t1 = (x for x in range(0, 9, 2))  
print(t1)  
print(tuple(t1))

运行结果：

<generator object <genexpr> at 0x00000229B6523C80>  
(0, 2, 4, 6, 8)

## 6.4字典操作

#### 6.4.1创建字典

字典是一个储存键值对的容器对象，可以通过关键字进行快速的获取，删除，更新值。

可以通过一个{}，并将键值对放入，键值通过：隔开，例如：

dict1 = {'name': '张三', 'age': 22}

字典中元素必须是key:value,key必须是可哈希类型，如:数字和字符串，空字典的创立可以是

dict2 = {}

花括号在python中可以用来创建集合和字典，{}表示一个字典，为了创建一个空集合则使用set()

#### 6.4.2字典的基本操作

# 字典的基本操作(增删改查)  
dicts = {'k1': 'v1', 'k2': 'v2'} # 通过键去找值  
# 访问字典的元素  
print(dicts['k1'])  
# 添加字典元素  
dicts['k3'] = 'v3'  
print(dicts)  
# 修改字典元素  
dicts['k1'] = 'v001'  
print(dicts)  
# 删除字典元素  
del dicts['k1']  
print(dicts)

运行结果：

v1  
{'k1': 'v1', 'k2': 'v2', 'k3': 'v3'}  
{'k1': 'v001', 'k2': 'v2', 'k3': 'v3'}  
{'k2': 'v2', 'k3': 'v3'}

#### 6.4.3字典的序列操作

# 成员检测(检测键)  
dicts = {'k1': 1, 'k2': 2, 'k3': 3}  
print('v1' not in dicts)  
# 序列函数(对键做操作)  
  
# len() 计算字典长度  
print(len(dicts))  
# max() 获取字典中最大的键  
print(max(dicts))  
# min() 获取字典中最小的键  
print(min(dicts))

运行结果：

True  
3  
k3  
k1

#### 6.4.4序列函数

#### 6.4.5字典的遍历

# 字典的遍历  
dicts = {'k1': 1, 'k2': 2, 'k3': 3, 'k4': 4, 'k5': 5}  
for key in dicts:  
 print('值', dicts[key])  
  
for key, value in dicts.items():  
 print('key:', key, 'value:', value)  
  
print(dicts.items())

运行结果：

值 1  
值 2  
值 3  
值 4  
值 5  
key: k1 value: 1  
key: k2 value: 2  
key: k3 value: 3  
key: k4 value: 4  
key: k5 value: 5  
dict\_items([('k1', 1), ('k2', 2), ('k3', 3), ('k4', 4), ('k5', 5)])

#### 6.4.6字典的内涵/字典推导式

# 字典的推导式  
# 普通字典的推导式  
# 变量 = {key:value for key value in 字典.items()}  
dicts = {'k1': 1, 'k2': 2, 'k3': 3, 'k4': 4, 'k5': 5}  
res = {key + '+-+': value for key, value in dicts.items()}  
print(res)  
# 带有判断条件的字典推导式  
# 变量 = {key:value for key value in 字典.items() if 判断条件}  
info = {  
 'name': 'zs',  
 'age': 20  
}  
res = {k: v for k, v in info.items() if k == 'name'}  
print(res)

运行结果：

{'k1+-+': 1, 'k2+-+': 2, 'k3+-+': 3, 'k4+-+': 4, 'k5+-+': 5}  
{'name': 'zs'}

# 多个循环的字典内涵  
# 变量={k1+k2: v1+v2 for k1,v1 in 字典1.items() for k2,v2 in 字典2.items() if 判断条件}

#### 6.4.7字典专用函数

|  |  |
| --- | --- |
| 函数 | 作用 |
| clear() | 删除字典所有条目 |
| copy() | 复制新的字典 |
| get(key) | 返回这个键对应的值 |
| items() | 返回一个元组序列，每个元组都是一个条目 |
| keys() | 返回一个关键字序列 |
| pop(key) | 删除这个关键字并删除这个条目 |
| popitem() | 随机选择的键值作为元组并删除这个被选择的条目 |
| update(k=v) | 修改字典中的值 |

# 字典的专用函数  
# 功能：清空字典  
# 格式：字典.clear()  
# 返回值：None  
# 注意：直接改变原有字典  
# info.clear()  
info = {  
 'name': 'zs',  
 'age': 20  
}  
print(info.clear())  
  
# 功能：复制字典  
# 格式：字典.copy()  
# 返回值：新的字典  
info = {  
 'name': 'zs',  
 'age': 20  
}  
res = info.copy()  
print(res)  
  
# 功能：根据键获取指定的值  
# 格式：字典.get(键[默认值])  
# 返回值：值(默认值)  
# 注意:如果键不存在,则使用默认值，如果没有默认值返回None  
  
print(info.get('ages', 18))  
  
# 功能：将字典的键值转化为类似元组的形式方便遍历  
# 格式：字典.items()  
# 返回值:类似于元组的格式  
print(info.items())  
  
# 功能：将字典的所有值组成一个序列  
# 格式：字典.keys()  
# 返回值：列表  
print(info.values())  
  
# pop()  
# 功能：移除字典中指定的元素  
# 格式：字典.pop(键[,默认值])  
# 返回值：被移除的键对应的值  
# 注意：如果键不存在，则报错，如果有默认值就返回默认值  
print(info.pop('ages', 'NULL'))  
  
# 功能：移除字典中的键值对  
# 格式：字典.popitem()  
# 返回值：键值对组成的元组  
# 注意：弹出一个，原字典就少一个，字典为空不能弹出，并报错  
# info.popitem()  
  
# 功能：添加一个元素  
# 格式：字典.setdefault(键，值)  
# 返回值：NONE  
# 注意：添加时键存在不进行任何操作，键不存在则添加  
info.setdefault('school', 'hrbu')  
print(info)  
  
  
# update  
# 功能：修改字典中的值  
# 方式1：字典.update(k=v)  
# 方式2：字典.update({k:v})  
info.update(name='lisi')  
print(info)

运行结果：

None  
{'name': 'zs', 'age': 20}  
18  
dict\_items([('name', 'zs'), ('age', 20)])  
dict\_values(['zs', 20])  
NULL

## 6.5集合

#### 6.5.1创建集合

集合与列表类似，它们都能存储元素，不同于列表来说，集合中的元素是不重复且无序的

可以将元素用一对{}所括起来用于创建一个元素集合，集合中元素使用逗号隔开

s1 = set()  
s2 = {1, 4, 7, 8}  
s3 = set((4, 5, 2, 8)) # 元组转集合  
s4 = set(x \* 3 for x in range(1, 11))  
print(s1)  
print(s2)  
print(s3)  
print(s4)

运行结果：

set()  
{8, 1, 4, 7}  
{8, 2, 4, 5}  
{3, 6, 9, 12, 15, 18, 21, 24, 27, 30}

#### 6.5.2集合的序列操作

使用add(x),remove(x)方法对一个集合添加或删除元素同样也可以通过使用len,min,max,min,sum方法对集合进行操作，in ,not in 来判断一个元素是否在一个集合中

s1 = {1, 4, 7, 2}  
s1.add(8)  
print(s1)  
print(len(s1))  
print(max(s1))  
print(min(s1))  
print(sum(s1))  
print(1 in s1)  
s1.remove(2)  
print(s1)

运行结果：

{1, 2, 4, 7, 8}  
5  
8  
1  
22  
True  
{1, 4, 7, 8}

#### 6.5.3集合的序列函数

|  |  |
| --- | --- |
| 函数 | 作用 |
| pop() | 删除集合中的第一个元素 |
| discard(x) | 删除集合中的元素x |
| clear() | 清空集合 |
| copy() | 复制集合 |

#### 6.5.4集合的遍历

s1 = (1, 2, 3, 4, 5, 6, 7, 8, 9)  
for u in s1:  
 print('元素值:', u)

#### 6.5.5集合的内涵/集合的推导式

普通集合推导式，

变量={i for i in 集合}

带有判断条件的推导式

变量={i for i in 集合 if 条件}

多循环的推导式

变量={i + j for i in 集合1 for j in 集合2}

带有判断条件的多循环推导式

变量={ i + j for i in 集合1 for j in 集合2 if 条件}}

s1 = {1, 3, 5, 7, 9}  
s2 = {2, 4, 6, 8}  
s3 = {i \* 2 for i in s1}  
print(s3)  
s3 = {i \* 2 for i in s1 if i > 2}  
print(s3)  
s4 = {(i, j) for i in s1 for j in s2}  
s5 = {i + j for i in s1 for j in s2} # 可对i,j做相关操作  
print(s4)  
print(s5)  
s6 = {i + j for i in s1 for j in s2 if i > 2 and j < 8}  
print(s6)

运行结果：

{2, 6, 10, 14, 18}  
{10, 18, 6, 14}  
{(3, 4), (5, 4), (9, 2), (9, 8), (1, 6), (7, 4), (5, 6), (3, 6), (9, 4), (1, 2), (1, 8), (7, 6), (3, 2), (5, 2), (3, 8), (5, 8), (9, 6), (1, 4), (7, 2), (7, 8)}  
{3, 5, 7, 9, 11, 13, 15, 17}  
{5, 7, 9, 11, 13, 15}

#### 6.5.6集合的专用函数

|  |  |
| --- | --- |
| 函数 | 作用 |
| difference() | 差集, 获取存在于集合1但是不存在集合2的数据的集合 |
| intersection() | 交集, 获取既存在于集合1又存在集合2的数据的集合 |
| union() | 并集, 将集合1与集合2中所有的数据新建一个集合(去重) |
| difference\_update() | 差集 更新, 获取存在于集合1但是不存在集合2的数据的集合，赋值集合1 |
| intersection\_update() | 交集 更新, 获取既存在于集合1又不存在集合2的数据的集合，赋值集合1 |
| update() | 将两个集合中所有的数据新建一个集合(去重)，赋值 |
| symmetric\_difference() | 对称差集, 将集合1和集合2中不相交的部分取出来组成新的集合 |
| symmetric\_difference\_update() | 对称差集更新，将集合1和集合2中不相交的部分取出来组成新的集合,赋值 |

# 集合间的常规运算  
set1 = {1, 2, 3, 4, 5, 6}  
set2 = {4, 5, 6, 7, 8, 9}  
  
# difference()  
# 功能：差集  
# 格式：集合.difference()  
# 返回值：集合  
# 含义：获取存在于集合1但是不存在集合2的数据的集合  
  
print(set1.difference(set2))  
  
# difference\_update()  
# 功能：差集 更新  
# 格式：集合1.difference\_update(集合2)  
# 返回值：无 直接将结果赋值给集合2  
# 含义：获取存在于集合1但是不存在集合2的数据的集合，赋值集合1  
set1.difference\_update(set2)  
print(set1)  
  
set1 = {1, 2, 3, 4, 5, 6}  
set2 = {4, 5, 6, 7, 8, 9}  
# intersection()  
# 功能：交集  
# 格式：集合1.intersection(集合2)  
# 返回值：集合  
# 含义：获取既存在于集合1又存在集合2的数据的集合  
print(set1.intersection(set2))  
  
# intersection\_update()  
# 功能：交集 更新  
# 格式：集合1.intersection\_update(集合2)  
# 返回值：无 直接将结果赋值给集合2  
# 含义：获取既存在于集合1又不存在集合2的数据的集合，赋值集合1  
set1.intersection\_update(set2)  
print(set1)  
set1 = {1, 2, 3, 4, 5, 6}  
set2 = {4, 5, 6, 7, 8, 9}  
# union()  
# 功能：并集  
# 格式：集合1.union(集合2)  
# 返回值：集合  
# 含义：将集合1与集合2中所有的数据新建一个集合(去重)  
print(set1.union(set2))  
  
# update()  
# 功能：并集  
# 格式：集合1.update(集合2)  
# 返回值：无 将结果赋值给集合1  
# 含义：将集合1与集合2中所有的数据新建一个集合(去重)，赋值  
set1.update(set2)  
print(set1)

set1 = {1, 2, 3, 4, 5, 6, 7, 8, 9}  
set2 = {3, 4, 5, 6}  
# symmetric\_difference()  
# 对称差集  
# 格式：集合1.symmetric\_difference(集合2)  
# 返回值：集合  
# 含义：将集合1和集合2中不相交的部分取出来组成新的集合  
print(set1.symmetric\_difference(set2))  
  
# symmetric\_difference\_update()  
# 对称差集更新  
# 格式：集合1.symmetric\_difference\_update(集合2)  
# 返回值：集合  
# 含义：将集合1和集合2中不相交的部分取出来组成新的集合,赋值  
set1.symmetric\_difference\_update(set2)  
print(set1)

运行结果：

{1, 2, 3}  
{1, 2, 3}  
{4, 5, 6}  
{4, 5, 6}  
{1, 2, 3, 4, 5, 6, 7, 8, 9}  
{1, 2, 3, 4, 5, 6, 7, 8, 9}

{1, 2, 7, 8, 9}  
{1, 2, 7, 8, 9}

#### 6.5.7冰冻集合的遍历

冰冻集合使用frozenset()函数进行冰冻集合

集合一旦冰冻无法修改，只能做集合的运算交集

s1 = frozenset()  
s2 = frozenset({1, 2, 3, 4})  
s3 = frozenset([4, 5, 6, 7])  
print(s1)  
print(s2)  
print(s3)

运行结果：

frozenset()  
frozenset({1, 2, 3, 4})  
frozenset({4, 5, 6, 7})

遍历：

s1 = frozenset({1, 2, 4, 7, 8, 5, 5, 7})  
for i in s1:  
 print(i)

运行结果：

1  
2  
4  
5  
7  
8

## 6.6实训项目

#### 6.6.1使用python实现线性回归算法

## 6.7课外作业

#### 6.7.1探索泰坦尼克号生存问题

## 6.8本章小结

## 6.9单元测试

## 6.10有趣的知识

#### 6.10.1手写文本图像生成

# 7面向对象

## 7.1类和对象（1）

#### 7.1.1面向对象ObjectOriented

#### 7.1.2面向过程和面向对象的区别

#### 7.1.3类和对象

#### 7.1.4类和对象的关系

#### 7.1.5类的组成

#### 7.1.6类的书写规则

#### 7.1.7实例化对象

#### 7.1.8检测类和对象的成员

#### 7.1.9类和对象成员的操作

## 7.2面向对象的特性

#### 7.2.1封装

#### 7.2.2继承

#### 7.2.3多态

## 7.3类和对象（2）

#### 7.3.1魔术方法

#### 7.3.2装饰器和抽象类

## 7.4项目实训

#### 7.4.1实现商城用户管理系统

#### 7.4.2实现商城商品管理系统

## 7.5课外作业

#### 7.5.1多重继承与C3算法

#### 7.5.2八皇后问题

## 7.6本章小结

## 7.7单元测试

## 7.8有趣的知识点

程序员有趣的注释