Python Day 6

Mittwoch, 16. September 2020 14:08

集合

Python 中set与dict类似，也是一组key的集合，但不存储value。由于key不能重复，所以set中，没有重复的key。
注意，key为不可变类型，即可哈希的值。
【例子】

```
In [112]: num = {}
          print(type(num))  # <class 'dict'>
          num = {1, 2, 3, 4}
          print(type(num))  # <class 'set'>
```

```
<class 'dict'>
<class 'set'>
```

1. 集合的创建

- 先创建对象再加入元素。
- 在创建空集合的时候只能使用s = set()，因为s = {}创建的是空字典。

【例子】

```
In [113]: basket = set()
          basket.add('apple')
          basket.add('banana')
          print(basket)  # {'banana', 'apple'}
```

```
{'banana', 'apple'}
```

- 直接把一堆元素用花括号括起来{元素1, 元素2, ..., 元素n}。
- 重复元素在set中会被自动被过滤。

【例子】

```
In [114]: basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
          print(basket)  # {'banana', 'apple', 'pear', 'orange'}
```

```
{'pear', 'orange', 'banana', 'apple'}
```

- 使用set(value)工厂函数，把列表或元组转换成集合。

【例子】

```
In [115]: a = set('abracadabra')
          print(a)
          # {'r', 'b', 'd', 'c', 'a'}

          b = set(("Google", "Lsgogroup", "Taobao", "Taobao"))
          print(b)
          # {'Taobao', 'Lsgogroup', 'Google'}

          c = set(["Google", "Lsgogroup", "Taobao", "Google"])
          print(c)
```

，在

```
# {'Taobao', 'Lsgogroup', 'Google'}
```

```
{'b', 'r', 'a', 'c', 'd'}
{'Taobao', 'Google', 'Lsgogroup'}
{'Taobao', 'Google', 'Lsgogroup'}
```

【例子】去掉列表中重复的元素

In [116]:
```
lst = [0, 1, 2, 3, 4, 5, 5, 3, 1]

temp = []
for item in lst:
    if item not in temp:
        temp.append(item)

print(temp)  # [0, 1, 2, 3, 4, 5]

a = set(lst)
print(list(a))  # [0, 1, 2, 3, 4, 5]
```

```
[0, 1, 2, 3, 4, 5]
[0, 1, 2, 3, 4, 5]
```

从结果发现集合的两个特点：无序 (unordered) 和唯一 (unique)。

由于 `set` 存储的是无序集合，所以我们不可以为集合创建索引或执行切片(slice)操作，也没有键(keys)可用来获取集合中元素的值，但是可以判断一个元素是否在集合中。

2. 访问集合中的值

- 可以使用len()内建函数得到集合的大小。

【例子】

In [117]:
```
s = set(['Google', 'Baidu', 'Taobao'])
print(len(s))  # 3
```

```
3
```

- 可以使用 `for` 把集合中的数据一个个读取出来。

【例子】

In [7]:
```
s = set(['Google', 'Baidu', 'Taobao'])
for item in s:
    print(item)

# Baidu
# Google
# Taobao
```

```
Baidu
Taobao
Google
```

- 可以通过 `in` 或 `not in` 判断一个元素是否在集合中已经存在

【例子】

In [119]:
```
s = set(['Google', 'Baidu', 'Taobao'])
print('Taobao' in s)  # True
print('Facebook' not in s)  # True
```

```
True
True
```

3. 集合的内置方法

- set.add(elmnt)用于给集合添加元素，如果添加的元素在集合中已存在，则不执行任何操作。

【例子】

In [120]:
```
fruits = {"apple", "banana", "cherry"}
fruits.add("orange")
print(fruits)
# {'orange', 'cherry', 'banana', 'apple'}

fruits.add("apple")
print(fruits)
# {'orange', 'cherry', 'banana', 'apple'}
```

```
{'cherry', 'orange', 'banana', 'apple'}
{'cherry', 'orange', 'banana', 'apple'}
```

- `set.update(set)` 用于修改当前集合，可以添加新的元素或集合到当前集合中，如果添加的元素在集合中已存在，则该元素只会出现一次，重复的会忽略。

【例子】

In [121]:
```
x = {"apple", "banana", "cherry"}
y = {"google", "baidu", "apple"}
x.update(y)
print(x)
# {'cherry', 'banana', 'apple', 'google', 'baidu'}

y.update(["lsgo", "dreamtech"])
print(y)
# {'lsgo', 'baidu', 'dreamtech', 'apple', 'google'}
```

```
{'google', 'banana', 'cherry', 'apple', 'baidu'}
{'apple', 'dreamtech', 'lsgo', 'google', 'baidu'}
```

- `set.remove(item)` 用于移除集合中的指定元素。如果元素不存在，则会发生错误。

【例子】

In [122]:
```
fruits = {"apple", "banana", "cherry"}
fruits.remove("banana")
print(fruits)  # {'apple', 'cherry'}
```

```
{'cherry', 'apple'}
```

由于 set 是无序和无重复元素的集合，所以两个或多个 set 可以做数学意义上的集合操作。

- `set.intersection(set1, set2)` 返回两个集合的交集。
- `set1 & set2` 返回两个集合的交集。
- `set.intersection_update(set1, set2)` 交集，在原始的集合上移除不重叠的元素。

【例子】

In [67]:
```
a = set('abracadabra')
b = set('alacazam')
print(a)  # {'r', 'a', 'c', 'b', 'd'}
print(b)  # {'c', 'a', 'l', 'm', 'z'}

c = a.intersection(b)
print(c)  # {'a', 'c'}
print(a & b)  # {'c', 'a'}
print(a)  # {'a', 'r', 'c', 'b', 'd'}

a.intersection_update(b)
print(a)  # {'a', 'c'}
```

```
{'b', 'r', 'a', 'c', 'd'}
{'l', 'a', 'c', 'z', 'm'}
{'a', 'c'}
{'a', 'c'}
{'b', 'r', 'a', 'c', 'd'}
{'a', 'c'}
```

- `set.union(set1, set2)` 返回两个集合的并集。
- `set1 | set2` 返回两个集合的并集。

- `set.union(set1, set2)` 返回两个集合的并集。
- `set1 | set2` 返回两个集合的并集。

【例子】

In [68]:
```
a = set('abracadabra')
b = set('alacazam')
print(a)  # {'r', 'a', 'c', 'b', 'd'}
print(b)  # {'c', 'a', 'l', 'm', 'z'}

print(a | b)
# {'l', 'd', 'm', 'b', 'a', 'r', 'z', 'c'}

c = a.union(b)
print(c)
# {'c', 'a', 'd', 'm', 'r', 'b', 'z', 'l'}
```

```
{'b', 'r', 'a', 'c', 'd'}
{'l', 'a', 'c', 'z', 'm'}
{'l', 'b', 'r', 'a', 'c', 'z', 'd', 'm'}
{'l', 'b', 'r', 'a', 'c', 'z', 'd', 'm'}
```

- `set.difference(set)` 返回集合的差集。
- `set1 - set2` 返回集合的差集。
- `set.difference_update(set)` 集合的差集，直接在原来的集合中移除元素，没有返回值。

【例子】

In [69]:
```
a = set('abracadabra')
b = set('alacazam')
print(a)  # {'r', 'a', 'c', 'b', 'd'}
print(b)  # {'c', 'a', 'l', 'm', 'z'}

c = a.difference(b)
print(c)  # {'b', 'd', 'r'}
print(a - b)  # {'d', 'b', 'r'}

print(a)  # {'r', 'd', 'c', 'a', 'b'}
a.difference_update(b)
print(a)  # {'d', 'r', 'b'}
```

```
{'b', 'r', 'a', 'c', 'd'}
{'l', 'a', 'c', 'z', 'm'}
{'d', 'b', 'r'}
{'d', 'b', 'r'}
{'b', 'r', 'a', 'c', 'd'}
{'b', 'r', 'd'}
```

- `set.symmetric_difference(set)` 返回集合的异或。
- `set1 ^ set2` 返回集合的异或。
- `set.symmetric_difference_update(set)` 移除当前集合中在另外一个指定集合相同的元素，并将另外一个指定集合中不同的元素插入到当前集合中。

【例子】

In [70]:
```
a = set('abracadabra')
b = set('alacazam')
print(a)  # {'r', 'a', 'c', 'b', 'd'}
print(b)  # {'c', 'a', 'l', 'm', 'z'}

c = a.symmetric_difference(b)
print(c)  # {'m', 'r', 'l', 'b', 'z', 'd'}
print(a ^ b)  # {'m', 'r', 'l', 'b', 'z', 'd'}

print(a)  # {'r', 'd', 'c', 'a', 'b'}
a.symmetric_difference_update(b)
print(a)  # {'r', 'b', 'm', 'l', 'z', 'd'}
```

```
{'b', 'r', 'a', 'c', 'd'}
{'l', 'a', 'c', 'z', 'm'}
{'l', 'b', 'z', 'r', 'd', 'm'}
{'l', 'b', 'z', 'r', 'd', 'm'}
{'b', 'r', 'a', 'c', 'd'}
{'l', 'b', 'r', 'z', 'd', 'm'}
```

- `set.issubset(set)` 判断集合是不是被其他集合包含，如果是则返回 True，否则返回 False。
- `set1 <= set2` 判断集合是不是被其他集合包含，如果是则返回 True，否则返回 False。

【例子】

In [71]:
```
x = {"a", "b", "c"}
y = {"f", "e", "d", "c", "b", "a"}
z = x.issubset(y)
print(z)  # True
print(x <= y)  # True

x = {"a", "b", "c"}
y = {"f", "e", "d", "c", "b"}
z = x.issubset(y)
print(z)  # False
print(x <= y)  # False
```

```
True
True
False
False
```

- `set.issuperset(set)` 用于判断集合是不是包含其他集合，如果是则返回 True，否则返回 False。
- `set1 >= set2` 判断集合是不是包含其他集合，如果是则返回 True，否则返回 False。

【例子】

In [72]:
```
x = {"f", "e", "d", "c", "b", "a"}
y = {"a", "b", "c"}
z = x.issuperset(y)
print(z)  # True
print(x >= y)  # True

x = {"f", "e", "d", "c", "b"}
y = {"a", "b", "c"}
```

```
print(z)  # False
print(x >= y)  # False
```

```
True
True
False
False
```

- `set.isdisjoint(set)` 用于判断两个集合是不是不相交，如果是返回 True，否则返回 False。

【例子】

In [73]:
```
x = {"f", "e", "d", "c", "b"}
y = {"a", "b", "c"}
z = x.isdisjoint(y)
print(z)  # False

x = {"f", "e", "d", "m", "g"}
y = {"a", "b", "c"}
z = x.isdisjoint(y)
print(z)  # True
```

```
False
True
```

4. 集合的转换

【例子】

In [74]:
```
se = set(range(4))
li = list(se)
tu = tuple(se)

print(se, type(se))  # {0, 1, 2, 3} <class 'set'>
print(li, type(li))  # [0, 1, 2, 3] <class 'list'>
print(tu, type(tu))  # (0, 1, 2, 3) <class 'tuple'>
```

```
{0, 1, 2, 3} <class 'set'>
[0, 1, 2, 3] <class 'list'>
(0, 1, 2, 3) <class 'tuple'>
```

5. 不可变集合

Python 提供了不能改变元素的集合的实现版本，即不能增加或删除元素，类型名叫frozenset。需要注意的是frozenset仍然可以进行集合操作，只是不能用带有update的方法。

- frozenset([iterable]) 返回一个冻结的集合，冻结后集合不能再添加或删除任何元素。

【例子】

In [75]:
```
a = frozenset(range(10))  # 生成一个新的不可变集合
print(a)
# frozenset({0, 1, 2, 3, 4, 5, 6, 7, 8, 9})

b = frozenset('lsgogroup')
print(b)
# frozenset({'g', 's', 'p', 'r', 'u', 'o', 'l'})
```

```
frozenset({0, 1, 2, 3, 4, 5, 6, 7, 8, 9})
frozenset({'l', 'g', 'r', 'u', 'o', 's', 'p'})
```

序列

在 Python 中，序列类型包括字符串、列表、元组、集合和字典，这些序列支持一些通用的操作，但比较特殊的是，集合和字典不支持索引、切片、相加和相乘操作

作。

1. 针对序列的内置函数

- list(sub) 把一个可迭代对象转换为列表。

【例子】

In [76]:
```
a = list()
print(a)  # []

b = 'I Love LsgoGroup'
b = list(b)
print(b)
# ['I', ' ', 'L', 'o', 'v', 'e', ' ', 'L', 's', 'g', 'o', 'G', 'r', 'o', 'u', 'p']

c = (1, 1, 2, 3, 5, 8)
c = list(c)
print(c)  # [1, 1, 2, 3, 5, 8]
```

```
[]
['I', ' ', 'L', 'o', 'v', 'e', ' ', 'L', 's', 'g', 'o', 'G', 'r', 'o', 'u', 'p']
[1, 1, 2, 3, 5, 8]
```

- `tuple(sub)` 把一个可迭代对象转换为元组。

【例子】

In [77]:
```
a = tuple()
print(a)  # ()

b = 'I Love LsgoGroup'
b = tuple(b)
print(b)
# ('I', ' ', 'L', 'o', 'v', 'e', ' ', 'L', 's', 'g', 'o', 'G', 'r', 'o', 'u', 'p')

c = [1, 1, 2, 3, 5, 8]
c = tuple(c)
print(c)  # (1, 1, 2, 3, 5, 8)
```

```
()
('I', ' ', 'L', 'o', 'v', 'e', ' ', 'L', 's', 'g', 'o', 'G', 'r', 'o', 'u', 'p')
(1, 1, 2, 3, 5, 8)
```

- `str(obj)` 把obj对象转换为字符串

【例子】

In [78]:
```
a = 123
a = str(a)
print(a)  # 123
```

```
123
```

- `len(s)` 返回对象（字符、列表、元组等）长度或元素个数。
  - `s` -- 对象。

【例子】

In [79]:
```
a = list()
print(len(a))  # 0

b = ('I', ' ', 'L', 'o', 'v', 'e', ' ', 'L', 's', 'g', 'o', 'G', 'r', 'o', 'u', 'p')
print(len(b))  # 16

c = 'I Love LsgoGroup'
print(len(c))  # 16
```

```
0
16
16
```

- `max(sub)` 返回序列或者参数集合中的最大值

【例子】

In [80]:
```
print(max(1, 2, 3, 4, 5))  # 5
print(max([-8, 99, 3, 7, 83]))  # 99
print(max('IloveLsgoGroup'))  # v
```

```
5
99
```

v

- `min(sub)` 返回序列或参数集合中的最小值

【例子】

In [81]:
```
print(min(1, 2, 3, 4, 5))  # 1
print(min([-8, 99, 3, 7, 83]))  # -8
print(min('IloveLsgoGroup'))  # G
```

```
1
-8
G
```

- `sum(iterable[, start=0])` 返回序列 `iterable` 与可选参数 `start` 的总和。

【例子】

In [82]:
```
print(sum([1, 3, 5, 7, 9]))  # 25
print(sum([1, 3, 5, 7, 9], 10))  # 35
print(sum((1, 3, 5, 7, 9)))  # 25
print(sum((1, 3, 5, 7, 9), 20))  # 45
```

```
25
35
25
45
```

- `sorted(iterable, key=None, reverse=False)` 对所有可迭代的对象进行排序操作。
  - `iterable` -- 可迭代对象。
  - `key` -- 主要是用来进行比较的元素，只有一个参数，具体的函数的参数就是取自于可迭代对象中，指定可迭代对象中的一个元素来进行排序。
  - `reverse` -- 排序规则，`reverse = True` 降序， `reverse = False` 升序（默认）。
  - 返回重新排序的列表。

【例子】

In [83]:
```
x = [-8, 99, 3, 7, 83]
print(sorted(x))  # [-8, 3, 7, 83, 99]
print(sorted(x, reverse=True))  # [99, 83, 7, 3, -8]

t = ({"age": 20, "name": "a"}, {"age": 25, "name": "b"}, {"age": 10, "name": "c"})
x = sorted(t, key=lambda a: a["age"])
print(x)
# [{'age': 10, 'name': 'c'}, {'age': 20, 'name': 'a'}, {'age': 25, 'name': 'b'}]
```

```
[-8, 3, 7, 83, 99]
[99, 83, 7, 3, -8]
[{'age': 10, 'name': 'c'}, {'age': 20, 'name': 'a'}, {'age': 25, 'name': 'b'}]
```

- `reversed(seq)` 函数返回一个反转的迭代器。
  - `seq` -- 要转换的序列，可以是 tuple, string, list 或 range。

【例子】

In [84]:
```
s = 'lsgogroup'
x = reversed(s)
print(type(x))  # <class 'reversed'>
print(x)  # <reversed object at 0x000002507E8EC2C8>
print(list(x))
# ['p', 'u', 'o', 'r', 'g', 'o', 'g', 's', 'l']

t = ('l', 's', 'g', 'o', 'g', 'r', 'o', 'u', 'p')
print(list(reversed(t)))
# ['p', 'u', 'o', 'r', 'g', 'o', 'g', 's', 'l']

r = range(5, 9)
print(list(reversed(r)))
# [8, 7, 6, 5]

x = [-8, 99, 3, 7, 83]
print(list(reversed(x)))
# [83, 7, 3, 99, -8]
```

```
<class 'reversed'>
<reversed object at 0x000001F0517DFD68>
['p', 'u', 'o', 'r', 'g', 'o', 'g', 's', 'l']
['p', 'u', 'o', 'r', 'g', 'o', 'g', 's', 'l']
[8, 7, 6, 5]
[83, 7, 3, 99, -8]
```

- `enumerate(sequence, [start=0])`

【例子】用于将一个可遍历的数据对象(如列表、元组或字符串)组合为一个索引序列，同时列出数据和数据下标，一般用在 for 循环当中。

In [85]:
```
seasons = ['Spring', 'Summer', 'Fall', 'Winter']
a = list(enumerate(seasons))
print(a)
# [(0, 'Spring'), (1, 'Summer'), (2, 'Fall'), (3, 'Winter')]
```

```
b = list(enumerate(seasons, 1))
print(b)
# [(1, 'Spring'), (2, 'Summer'), (3, 'Fall'), (4, 'Winter')]

for i, element in a:
    print('{0},{1}'.format(i, element))
# 0,Spring
# 1,Summer
# 2,Fall
# 3,Winter
```

```
[(0, 'Spring'), (1, 'Summer'), (2, 'Fall'), (3, 'Winter')]
[(1, 'Spring'), (2, 'Summer'), (3, 'Fall'), (4, 'Winter')]
0,Spring
1,Summer
2,Fall
3,Winter
```

- `zip(iter1 [,iter2 [...]])`
  - 用于将可迭代的对象作为参数，将对象中对应的元素打包成一个个元组，然后返回由这些元组组成的对象，这样做的好处是节约了不少的内存。
  - 我们可以使用 `list()` 转换来输出列表。
  - 如果各个迭代器的元素个数不一致，则返回列表长度与最短的对象相同，利用 `*` 号操作符，可以将元组解压为列表。

【例子】

In [86]:
```
a = [1, 2, 3]
b = [4, 5, 6]
c = [4, 5, 6, 7, 8]

zipped = zip(a, b)
print(zipped)  # <zip object at 0x000000C5D89EDD88>
print(list(zipped))  # [(1, 4), (2, 5), (3, 6)]
zipped = zip(a, c)
print(list(zipped))  # [(1, 4), (2, 5), (3, 6)]

a1, a2 = zip(*zip(a, b))
print(list(a1))  # [1, 2, 3]
print(list(a2))  # [4, 5, 6]
```

```
<zip object at 0x000001F0517E38C8>
[(1, 4), (2, 5), (3, 6)]
[(1, 4), (2, 5), (3, 6)]
[1, 2, 3]
[4, 5, 6]
```