Python Day 4

Dienstag, 15. September 2020 21:48

列表

简单数据类型

- 整型<class 'int'>
- 浮点型<class 'float'>
- 布尔型<class 'bool'>

容器数据类型

- 列表<class 'list'>
- 元组<class 'tuple'>
- 字典<class 'dict'>
- 集合<class 'set'>
- 字符串<class 'str'>

1. 列表的定义

列表是有序集合，没有固定大小，能够保存任意数量任意类型的 Python 对象，语法为 [元素1, 元素2, ..., 元素n]。

- 关键点是「中括号 []」和「逗号 ,」
- 中括号 把所有元素绑在一起
- 逗号 将每个元素一一分开

2. 列表的创建

- 创建一个普通列表

【例子】

```
In [1]: x = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
        print(x, type(x))
        # ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday'] <class 'list'>

        x = [2, 3, 4, 5, 6, 7]
        print(x, type(x))
        # [2, 3, 4, 5, 6, 7] <class 'list'>
```

```
['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday'] <class 'list'>
[2, 3, 4, 5, 6, 7] <class 'list'>
```

- 利用 `range()` 创建列表

【例子】

```
In [2]: x = list(range(10))
        print(x, type(x))
        # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] <class 'list'>
```

[元素

```
x = list(range(1, 11, 2))
print(x, type(x))
# [1, 3, 5, 7, 9] <class 'list'>

x = list(range(10, 1, -2))
print(x, type(x))
# [10, 8, 6, 4, 2] <class 'list'>
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9] <class 'list'>
[1, 3, 5, 7, 9] <class 'list'>
[10, 8, 6, 4, 2] <class 'list'>
```

- 利用推导式创建列表
  【例子】

In [3]:
```
x = [0] * 5
print(x, type(x))
# [0, 0, 0, 0, 0] <class 'list'>

x = [0 for i in range(5)]
print(x, type(x))
# [0, 0, 0, 0, 0] <class 'list'>

x = [i for i in range(10)]
print(x, type(x))
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] <class 'list'>

x = [i for i in range(1, 10, 2)]
print(x, type(x))
# [1, 3, 5, 7, 9] <class 'list'>

x = [i for i in range(10, 1, -2)]
print(x, type(x))
# [10, 8, 6, 4, 2] <class 'list'>

x = [i ** 2 for i in range(1, 10)]
print(x, type(x))
# [1, 4, 9, 16, 25, 36, 49, 64, 81] <class 'list'>

x = [i for i in range(100) if (i % 2) != 0 and (i % 3) == 0]
print(x, type(x))

# [3, 9, 15, 21, 27, 33, 39,
```

```
[0, 0, 0, 0, 0] <class 'list'>
[0, 0, 0, 0, 0] <class 'list'>
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9] <class 'list'>
[1, 3, 5, 7, 9] <class 'list'>
[10, 8, 6, 4, 2] <class 'list'>
[1, 4, 9, 16, 25, 36, 49, 64, 81] <class 'list'>
[3, 9, 15, 21, 27, 33, 39, 45, 51, 57, 63, 69, 75, 81, 87, 93, 99] <class 'list'>
```

注意：
由于list的元素可以是任何对象，因此列表中所保存的是对象的指针。即使保存一个简单[1,2,3]，也有3个指针和3个整数对象。
x = [a] * 4操作中，只是创建4个指向list的引用，所以一旦a改变，x中4个a也会随之改变
【例子】

In [4]:
```
x = [[0] * 3] * 4
print(x, type(x))
# [[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]] <class 'list'>

x[0][0] = 1
print(x, type(x))
# [[1, 0, 0], [1, 0, 0], [1, 0, 0], [1, 0, 0]] <class 'list'>

a = [0] * 3
x = [a] * 4
print(x, type(x))
# [[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]] <class 'list'>

x[0][0] = 1
print(x, type(x))
# [[1, 0, 0], [1, 0, 0], [1, 0, 0], [1, 0, 0]] <class 'list'>
```

```
[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]] <class 'list'>
[[1, 0, 0], [1, 0, 0], [1, 0, 0], [1, 0, 0]] <class 'list'>
[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]] <class 'list'>
[[1, 0, 0], [1, 0, 0], [1, 0, 0], [1, 0, 0]] <class 'list'>
```

的

。

- 创建一个混合列表

【例子】

In [5]:
```
mix = [1, 'lsgo', 3.14, [1, 2, 3]]
print(mix, type(mix))
# [1, 'lsgo', 3.14, [1, 2, 3]] <class 'list'>
```

```
[1, 'lsgo', 3.14, [1, 2, 3]] <class 'list'>
```

- 创建一个空列表

【例子】

In [6]:
```
empty = []
print(empty, type(empty))  # [] <class 'list'>
```

```
[] <class 'list'>
```

列表不像元组，列表内容可更改 (mutable)，因此附加 (`append` , `extend`)、插入 (`insert`)、删除 (`remove` , `pop`) 这些操作都可以用在它身上。

3. 向列表中添加元素

- `list.append(obj)` 在列表末尾添加新的对象，只接受一个参数，参数可以是任何数据型，被追加的元素在 list 中保持着原结构类型。

【例子】

In [7]:
```
x = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
x.append('Thursday')
print(x)
# ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Thursday']

print(len(x))  # 6
```

```
['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Thursday']
6
```

此元素如果是一个 list，那么这个 list 将作为一个整体进行追加，注意 `append()` 和 `extend()` 的区别。

【例子】

In [8]:
```
x = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
x.append(['Thursday', 'Sunday'])
print(x)
# ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', ['Thursday', 'Sunday']]

print(len(x))  # 6
```

```
['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', ['Thursday', 'Sunday']]
6
```

- `list.extend(seq)` 在列表末尾一次性追加另一个序列中的多个值（用新列表扩展原来的列表）

【例子】

In [9]:
```
x = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
x.extend(['Thursday', 'Sunday'])
print(x)
# ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Thursday', 'Sunday']

print(len(x))  # 7
```

```
['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Thursday', 'Sunday']
7
```

严格来说 append 是追加，把一个东西整体添加在列表后，而 extend 是扩展，把一个东西里的所有元素添加在列表后。

- list.insert(index, obj) 在编号 index 位置插入 obj。

【例子】

In [10]:
```
x = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
x.insert(2, 'Sunday')
print(x)
# ['Monday', 'Tuesday', 'Sunday', 'Wednesday', 'Thursday', 'Friday']

print(len(x))  # 6
```

```
['Monday', 'Tuesday', 'Sunday', 'Wednesday', 'Thursday', 'Friday']
6
```

4. 删除列表中的元素

- list.remove(obj) 移除列表中某个值的第一个匹配项

【例子】

In [11]:
```
x = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
x.remove('Monday')
print(x)  # ['Tuesday', 'Wednesday', 'Thursday', 'Friday']
```

```
['Tuesday', 'Wednesday', 'Thursday', 'Friday']
```

- `list.pop([index=-1])` 移除列表中的一个元素（默认最后一个元素），并且返回该元素的值

【例子】

In [12]:
```
x = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
y = x.pop()
print(y)  # Friday

y = x.pop(0)
print(y)  # Monday

y = x.pop(-2)
print(y)  # Wednesday
print(x)  # ['Tuesday', 'Thursday']
```

```
Friday
Monday
Wednesday
['Tuesday', 'Thursday']
```

`remove` 和 `pop` 都可以删除元素，前者是指定具体要删除的元素，后者是指定一个索引。

- `del var1[, var2 ……]` 删除单个或多个对象。

【例子】

如果知道要删除的元素在列表中的位置，可使用 `del` 语句。

In [13]:
```
x = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
del x[0:2]
print(x)  # ['Wednesday', 'Thursday', 'Friday']
```

```
['Wednesday', 'Thursday', 'Friday']
```

如果你要从列表中删除一个元素，且不再以任何方式使用它，就使用 `del` 语句；如果你要在删除元素后还能继续使用它，就使用方法 `pop()` 。

5. 获取列表中的元素

- 通过元素的索引值，从列表获取单个元素，注意，列表索引值是从0开始的。
- 通过将索引指定为-1，可让Python返回最后一个列表元素，索引 -2 返回倒数第二个列表元素，以此类推。

【例子】

In [14]:
```
x = ['Monday', 'Tuesday', 'Wednesday', ['Thursday', 'Friday']]
print(x[0], type(x[0]))  # Monday <class 'str'>
print(x[-1], type(x[-1]))  # ['Thursday', 'Friday'] <class 'list'>
print(x[-2], type(x[-2]))  # Wednesday <class 'str'>
```

```
Monday <class 'str'>
['Thursday', 'Friday'] <class 'list'>
Wednesday <class 'str'>
```

切片的通用写法是 `start : stop : step`

- 情况 1 - "start :"
- 以 `step` 为 1 (默认) 从编号 `start` 往列表尾部切片。

【例子】

In [15]:
```
x = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
print(x[3:])  # ['Thursday', 'Friday']
print(x[-3:])  # ['Wednesday', 'Thursday', 'Friday']
```

```
['Thursday', 'Friday']
['Wednesday', 'Thursday', 'Friday']
```

- 情况 2 - ": stop"
- 以 `step` 为 1 (默认) 从列表头部往编号 `stop` 切片。

【例子】

In [16]:
```
week = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
print(week[:3])  # ['Monday', 'Tuesday', 'Wednesday']
print(week[:-3])  # ['Monday', 'Tuesday']
```

```
['Monday', 'Tuesday', 'Wednesday']
['Monday', 'Tuesday']
```

- 情况 3 - "start : stop"
- 以 `step` 为 1 (默认) 从编号 `start` 往编号 `stop` 切片。

【例子】

In [17]:
```
week = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
print(week[1:3])  # ['Tuesday', 'Wednesday']
print(week[-3:-1])  # ['Wednesday', 'Thursday']
```

```
['Tuesday', 'Wednesday']
['Wednesday', 'Thursday']
```

- 情况 4 - "start : stop : step"
- 以具体的 `step` 从编号 `start` 往编号 `stop` 切片。注意最后把 `step` 设为 -1，相当于将列表反向排列。

【例子】

In [18]:
```
week = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
print(week[1:4:2])  # ['Tuesday', 'Thursday']
print(week[:4:2])  # ['Monday', 'Wednesday']
print(week[1::2])  # ['Tuesday', 'Thursday']
print(week[::-1])
# ['Friday', 'Thursday', 'Wednesday', 'Tuesday', 'Monday']
```

```
['Tuesday', 'Thursday']
['Monday', 'Wednesday']
['Tuesday', 'Thursday']
['Friday', 'Thursday', 'Wednesday', 'Tuesday', 'Monday']
```

- 情况 5 - " : "
- 复制列表中的所有元素（浅拷贝）。

【例子】

In [19]:
```
eek = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
print(week[:])
# ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
```

```
['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
```

【例子】浅拷贝与深拷贝

【例子】浅拷贝与深拷贝

In [20]:
```
list1 = [123, 456, 789, 213]
list2 = list1
list3 = list1[:]

print(list2)  # [123, 456, 789, 213]
print(list3)  # [123, 456, 789, 213]
list1.sort()
print(list2)  # [123, 213, 456, 789]
print(list3)  # [123, 456, 789, 213]

list1 = [[123, 456], [789, 213]]
list2 = list1
list3 = list1[:]
print(list2)  # [[123, 456], [789, 213]]
print(list3)  # [[123, 456], [789, 213]]
list1[0][0] = 111
print(list2)  # [[111, 456], [789, 213]]
print(list3)  # [[111, 456], [789, 213]]
```

```
[123, 456, 789, 213]
[123, 456, 789, 213]
[123, 213, 456, 789]
[123, 456, 789, 213]
[[123, 456], [789, 213]]
[[123, 456], [789, 213]]
[[111, 456], [789, 213]]
[[111, 456], [789, 213]]
```

6. 列表的常用操作符

- 等号操作符：==
- 连接操作符 +
- 重复操作符 *
- 成员关系操作符 in、not in

「等号 ==」，只有成员、成员位置都相同时才返回True。

列表拼接有两种方式，用「加号 +」和「乘号 *」，前者首尾拼接，后者复制拼接。

【例子】

In [21]:
```
list1 = [123, 456]
list2 = [456, 123]
list3 = [123, 456]

print(list1 == list2)  # False
print(list1 == list3)  # True

list4 = list1 + list2  # extend()
print(list4)  # [123, 456, 456, 123]
```

```
list5 = list3 * 3
print(list5)  # [123, 456, 123, 456, 123, 456]

list3 *= 3
print(list3)  # [123, 456, 123, 456, 123, 456]

print(123 in list3)  # True
print(456 not in list3)  # False
```

```
False
True
[123, 456, 456, 123]
[123, 456, 123, 456, 123, 456]
[123, 456, 123, 456, 123, 456]
True
False
```

前面三种方法（append, extend, insert）可对列表增加元素，它们没有返回值，是直接修改了原数据对象。 而将两个list相加，需要创建新的 list 对象，从而需要消耗额外的内存，特别是当 list 较大时，尽量不要使用 “+” 来添加list。

7. 列表的其它方法

list.count(obj) 统计某个元素在列表中出现的次数
【例子】

In [22]:
```
list1 = [123, 456] * 3
print(list1)  # [123, 456, 123, 456, 123, 456]
num = list1.count(123)
print(num)  # 3
```

```
[123, 456, 123, 456, 123, 456]
3
```

`list.index(x[, start[, end]])` 从列表中找出某个值第一个匹配项的索引位置

【例子】

In [23]:
```
list1 = [123, 456] * 5
print(list1.index(123))  # 0
print(list1.index(123, 1))  # 2
print(list1.index(123, 3, 7))  # 4
```

```
0
2
4
```

`list.reverse()` 反向列表中元素

【例子】

In [24]:
```
x = [123, 456, 789]
x.reverse()
print(x)  # [789, 456, 123]
```

```
[789, 456, 123]
```

list.sort(key=None, reverse=False) 对原列表进行排序。

- 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）。
- 该方法没有返回值，但是会对列表的对象进行排序。

【例子】

In [25]:
```
x = [123, 456, 789, 213]
x.sort()
print(x)
# [123, 213, 456, 789]

x.sort(reverse=True)
print(x)
# [789, 456, 213, 123]


# 获取列表的第二个元素
def takeSecond(elem):
    return elem[1]


x = [(2, 2), (3, 4), (4, 1), (1, 3)]
x.sort(key=takeSecond)
print(x)
# [(4, 1), (2, 2), (1, 3), (3, 4)]

x.sort(key=lambda a: a[0])
print(x)
# [(1, 3), (2, 2), (3, 4), (4, 1)]
```

```
[123, 213, 456, 789]
[789, 456, 213, 123]
[(4, 1), (2, 2), (1, 3), (3, 4)]
[(1, 3), (2, 2), (3, 4), (4, 1)]
```

元组

「元组」定义语法为：(元素1, 元素2, ..., 元素n)

- 小括号把所有元素绑在一起
- 逗号将每个元素一一分开

1. 创建和访问一个元组

- Python 的元组与列表类似，不同之处在于tuple被创建后就不能对其进行修改，类似字符串。
- 元组使用小括号，列表使用方括号。
- 元组与列表类似，也用整数来
- 对它进行索引 (indexing) 和切片 (slicing)。

【例子】

In [26]:
```
t1 = (1, 10.31, 'python')
t2 = 1, 10.31, 'python'
print(t1, type(t1))
# (1, 10.31, 'python') <class 'tuple'>

print(t2, type(t2))
# (1, 10.31, 'python') <class 'tuple'>

tuple1 = (1, 2, 3, 4, 5, 6, 7, 8)
print(tuple1[1])  # 2
print(tuple1[5:])  # (6, 7, 8)
print(tuple1[:5])  # (1, 2, 3, 4, 5)
tuple2 = tuple1[:]
print(tuple2)  # (1, 2, 3, 4, 5, 6, 7, 8)
```

```
(1, 10.31, 'python') <class 'tuple'>
(1, 10.31, 'python') <class 'tuple'>
2
(6, 7, 8)
(1, 2, 3, 4, 5)
(1, 2, 3, 4, 5, 6, 7, 8)
```

- 创建元组可以用小括号 ()，也可以什么都不用，为了可读性，建议还是用 ()。
- 元组中只包含一个元素时，需要在元素后面添加逗号，否则括号会被当作运算符使用。

【例子】

```
In [27]: x = (1)
         print(type(x))  # <class 'int'>
         x = 2, 3, 4, 5
         print(type(x))  # <class 'tuple'>
         x = []
         print(type(x))  # <class 'list'>
         x = ()
         print(type(x))  # <class 'tuple'>
         x = (1,)
         print(type(x))  # <class 'tuple'>

         <class 'int'>
         <class 'tuple'>
         <class 'list'>
         <class 'tuple'>
         <class 'tuple'>
```

【例子】

```
In [28]: print(8 * (8))  # 64
         print(8 * (8,))  # (8, 8, 8, 8, 8, 8, 8, 8)

         64
         (8, 8, 8, 8, 8, 8, 8, 8)
```

【例子】创建二维元组。

```
In [29]: x = (1, 10.31, 'python'), ('data', 11)
         print(x)
         # ((1, 10.31, 'python'), ('data', 11))

         print(x[0])
         # (1, 10.31, 'python')
         print(x[0][0], x[0][1], x[0][2])
         # 1 10.31 python

         print(x[0][0:2])
         # (1, 10.31)

         ((1, 10.31, 'python'), ('data', 11))
         (1, 10.31, 'python')
         1 10.31 python
         (1, 10.31)
```

2. 更新和删除一个元组

【例子】

```
In [30]: week = ('Monday', 'Tuesday', 'Thursday', 'Friday')
         week = week[:2] + ('Wednesday',) + week[2:]
         print(week)  # ('Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday')

         ('Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday')
```

【例子】元组有不可更改 (immutable) 的性质，因此不能直接给元组的元素赋值，但是只要元组中的元素可更改 (mutable)，那么我们可以直接更改其元素，意这跟赋值其元素不同。

```
In [31]: t1 = (1, 2, 3, [4, 5, 6])
         print(t1)  # (1, 2, 3, [4, 5, 6])

         t1[3][0] = 9
         print(t1)  # (1, 2, 3, [9, 5, 6])

         (1, 2, 3, [4, 5, 6])
         (1, 2, 3, [9, 5, 6])
```

3. 元组相关的操作符

- 等号操作符：==
- 连接操作符 +
- 重复操作符 *
- 成员关系操作符 in、not in

「等号 ==」，只有成员、成员位置都相同时才返回True。
元组拼接有两种方式，用「加号 +」和「乘号 *」，前者首尾拼接，后者复制拼接。
【例子】

In [32]:

In [30]:
```
week = ('Monday', 'Tuesday', 'Thursday', 'Friday')
week = week[:2] + ('Wednesday',) + week[2:]
print(week)  # ('Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday')
```

```
('Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday')
```

【例子】元组有不可更改 (immutable) 的性质，因此不能直接给元组的元素赋值，但是只要元组中的元素可更改 (mutable)，那么我们可以直接更改其元素，注意这跟赋值其元素不同。

In [31]:
```
t1 = (1, 2, 3, [4, 5, 6])
print(t1)  # (1, 2, 3, [4, 5, 6])

t1[3][0] = 9
print(t1)  # (1, 2, 3, [9, 5, 6])
```

```
(1, 2, 3, [4, 5, 6])
(1, 2, 3, [9, 5, 6])
```

3. 元组相关的操作符

- 等号操作符：==
- 连接操作符 +
- 重复操作符 *
- 成员关系操作符 in、not in

「等号 ==」，只有成员、成员位置都相同时才返回True。
元组拼接有两种方式，用「加号 +」和「乘号 *」，前者首尾拼接，后者复制拼接。
【例子】

In [32]:
```
t1 = (123, 456)
t2 = (456, 123)
t3 = (123, 456)

print(t1 == t2)  # False
print(t1 == t3)  # True

t4 = t1 + t2
print(t4)  # (123, 456, 456, 123)

t5 = t3 * 3
print(t5)  # (123, 456, 123, 456, 123, 456)

t3 *= 3
print(t3)  # (123, 456, 123, 456, 123, 456)

print(123 in t3)  # True
print(456 not in t3)  # False
```

```
False
True
(123, 456, 456, 123)
(123, 456, 123, 456, 123, 456)
(123, 456, 123, 456, 123, 456)
```

```
True
False
```

4. 内置方法

元组大小和内容都不可更改，因此只有 count 和 index 两种方法。

【例子】

In [33]:
```
t = (1, 10.31, 'python')
print(t.count('python'))  # 1
print(t.index(10.31))  # 1
```

```
1
1
```

- `count('python')` 是记录在元组 `t` 中该元素出现几次，显然是 1 次
- `index(10.31)` 是找到该元素在元组 `t` 的索引，显然是 1

5. 解压元组

【例子】解压（unpack）一维元组（有几个元素左边括号定义几个变量）

In [34]:
```
t = (1, 10.31, 'python')
(a, b, c) = t
print(a, b, c)
# 1 10.31 python
```

```
1 10.31 python
```

【例子】解压二维元组（按照元组里的元组结构来定义变量）

In [35]:
```
t = (1, 10.31, ('OK', 'python'))
(a, b, (c, d)) = t
print(a, b, c, d)
# 1 10.31 OK python
```

```
1 10.31 OK python
```

【例子】如果你只想要元组其中几个元素，用通配符「*」，英文叫 wildcard，在计算机语言中代表一个或多个元素。下例就是把多个元素丢给了 `rest` 变量。

In [36]:
```
t = 1, 2, 3, 4, 5
a, b, *rest, c = t
print(a, b, c)  # 1 2 5
print(rest)  # [3, 4]
```

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

【例子】如果你根本不在乎 rest 变量，那么就用通配符「*」加上下划线「_」。

In [37]:
```
t = 1, 2, 3, 4, 5
a, b, *_ = t
print(a, b)  # 1 2
```

```
1 2
```