03-Python列表

列表介绍

- 列表由一系列特定顺序(sequence)排列的元素组成。
- 在Python中，用方括号[ ]表示列表，用逗号分隔其中的元素。
- 最后一个元素后面的逗号会被忽略。

In [6]:
```
bicycles = ['trek', 'cannondale', 'redline', 'specialized',]
print(bicycles)
```

```
['trek', 'cannondale', 'redline', 'specialized']
```

python列表没有类型限制，列表中可以存放任意类型的元素

In [3]:
```
elements = [3, 'hello', 2.5, True, 'world', ]
print(elements)
```

```
[3, 'hello', 2.5, True, 'world']
```

利用索引访问列表元素

In [2]:
```
numbers = [1, 2, 3, 4, 5]

print(numbers[0])
print(numbers[4])

# 可以使用符数作为索引，-1表示最后一个元素
print(numbers[-1])
print(numbers[-5])
```

```
1
5
5
1
```

通过索引查找或者修改元素

In [1]:
```
motorcycles = ['honda', 'yamaha', 'suzuki', ]
motorcycles[0] = 'ducati'
print(motorcycles)
```

```
['ducati', 'yamaha', 'suzuki']
```

在末尾附加元素：append方法

In [2]:
```
motorcycles = ['honda', 'yamaha', 'suzuki', ]
motorcycles.append('ducati')
print(motorcycles)
```

```
['honda', 'yamaha', 'suzuki', 'ducati']
```

在列表中插入元素： insert方法

In [3]:
```
motorcycles = ['honda', 'yamaha', 'suzuki', ]
motorcycles.insert(0, 'ducati')
print(motorcycles)
```

```
['ducati', 'honda', 'yamaha', 'suzuki']
```

使用del语句删除元素

In [4]:
```
motorcycles = ['honda', 'yamaha', 'suzuki', ]
del motorcycles[0]
print(motorcycles)
```

```
['yamaha', 'suzuki']
```

del语句也可以用来删除其他简单变量

In [1]:
```
message = 'Hello'
print(message)
del message
print(message)
```

```
Hello

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[1], line 4
      2 print(message)
      3 del message
----> 4 print(message)

NameError: name 'message' is not defined
```

pop语句删除列表末尾的元素并返回元素值

In [2]:
```
motorcycles = ['honda', 'yamaha', 'suzuki', ]
print(motorcycles)

popped_motorcycle = motorcycles.pop()
print(motorcycles)
print(popped_motorcycle)
```

```
['honda', 'yamaha', 'suzuki']
['honda', 'yamaha']
suzuki
```

pop语句删除列表任意位置的元素

In [4]:
```
motorcycles = ['honda', 'yamaha', 'suzuki', ]
first_owned = motorcycles.pop(0)
print(f'The first motorcycle I owned was a {first_owned.title()}.')
print(motorcycles)
```

```
The first motorcycle I owned was a Honda.
['yamaha', 'suzuki']
```

remove方法根据值删除列表中的元素

In [5]:
```
motorcycles = ['honda', 'yamaha', 'suzuki', 'ducati']
print(motorcycles)
motorcycles.remove('ducati')
print(motorcycles)
```

```
['honda', 'yamaha', 'suzuki', 'ducati']
['honda', 'yamaha', 'suzuki']
```

如果列表中有相同的值，remove方法删除第一个匹配的值

In [7]:
```
motorcycles = ['honda', 'yamaha', 'suzuki', 'honda']
motorcycles.remove('honda')
print(motorcycles)
```

```
['yamaha', 'suzuki', 'honda']
```

使用sort()方法对列表进行永久性(in place)排序

In [8]:
```
cars = ['bmw', 'audi', 'toyota', 'subaru', ]
cars.sort()
print(cars)
```

```
['audi', 'bmw', 'subaru', 'toyota']
```

In [9]:
```
cars.sort(reverse=True)
print(cars)
```

```
['toyota', 'subaru', 'bmw', 'audi']
```

使用sorted()函数对列表进行临时排序

In [13]:
```
cars = ['bmw', 'audi', 'toyota', 'subaru', ]
print(f"The original list:\n {cars}\n")
print(f"The sorted list:\n {sorted(cars)}\n")
print(f"The original list:\n {cars}\n")
```

```
The original list:
 ['bmw', 'audi', 'toyota', 'subaru']

The sorted list:
 ['audi', 'bmw', 'subaru', 'toyota']

The original list:
 ['bmw', 'audi', 'toyota', 'subaru']
```

倒着打印列表：使用reverse()方法

In [14]:
```
cars = ['bmw', 'audi', 'toyota', 'subaru', ]
print(cars)
cars.reverse()
print(cars)
```

```
['bmw', 'audi', 'toyota', 'subaru']
['subaru', 'toyota', 'audi', 'bmw']
```

确定列表的长度： 使用len()函数

In [15]:
```
len(cars)
```

Out[15]: 4

很多其他数据类型和数据结构都可以使用len()函数

In [16]:
```
len('hello world')
```

Out[16]: 11

操作列表

遍历整个列表

In [17]:
```
magicians = ['alice', 'david', 'carolina', ]
for magician in magicians:
    print(magician)
```

```
alice
david
carolina
```

在Python语言中，使用 ： 表示代码块的开始，使用缩进来表示代码块

In [18]:
```
magicians = ['alice', 'david', 'carolina', ]
for magician in magicians:
    print(f"{magician.title()}, that was a great trick!")
    print(f"I can't wait to see your next trick, {magician.title()}.\n")

print("Thank you, everyone. That was a great magic show!")
```

```
Alice, that was a great trick!
I can't wait to see your next trick, Alice.

David, that was a great trick!
I can't wait to see your next trick, David.

Carolina, that was a great trick!
I can't wait to see your next trick, Carolina.

Thank you, everyone. That was a great magic show!
```

使用 range() 函数创建数值列表

In [19]:
```
for value in range(1, 5):
    print(value)
```

```
1
2
3
4
```

使用range()函数产生1到10的偶数，range()函数的第三个参数表示步长

In [21]:
```
for value in range(2, 11, 2):
    print(value)
```

```
2
4
6
8
10
```

range()函数的步长可以是负数

In [29]:
```
for value in range(5, 0, -1):
    print(value, end=' ')
```

```
5 4 3 2 1
```

使用range()函数创建数字列表

In [30]:
```
numbers = list(range(1, 6))
print(numbers)
```

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

对数字列表进行统计计算

In [31]:
```
digits = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0, ]
min(digits)
```

Out[31]: 0

In [35]:
```
max(digits)
```

Out[35]: 9

In [34]:
```
sum(digits)
```

Out[34]: 45

列表解析 (List Comprehension)

In [36]:
```
#列表解析格式：[ 变量表达式  for  变量  in  列表       ]                                    ]
squares =    [ value**2  for  value  in range(1, 11)]
print(squares)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

列表切片 (Slice)

In [37]:
```
players = ['charles', 'martina', 'michael', 'florence', 'eli', ]
print(players[0:3])  # 切片的长度等于3-0=3
```

```
['charles', 'martina', 'michael']
```

In [38]:
```
print(players[1:4])
```

```
['martina', 'michael', 'florence']
```

In [39]:
```
print(players[:4])
```

```
['charles', 'martina', 'michael', 'florence']
```

In [40]:
```
print(players[2:])
```

```
['michael', 'florence', 'eli']
```

In [41]:
```
print(players[-3:])
```

```
['michael', 'florence', 'eli']
```

切片的步长

In [47]:
```
numbers = list(range(1, 11))
print(numbers[1:10:2])
```

```
[2, 4, 6, 8, 10]
```

步长可以为负数

In [44]:
```
print(numbers[9:0:-2])
```

```
[10, 8, 6, 4, 2]
```

In [48]:
```
print(numbers[-1:-10:-2])
```

```
[10, 8, 6, 4, 2]
```

复制列表

In [58]:
```
my_food = ['pizza', 'falafel', 'carrot cake', ]
# 不可以这样复制列表
# friend_food = my_food

friend_food = my_food[:]
friend_food.append('cannoli')
my_food.append('ice cream')

print(my_food)
print(friend_food)
print(id(my_food))
print(id(friend_food))
```

```
['pizza', 'falafel', 'carrot cake', 'ice cream']
['pizza', 'falafel', 'carrot cake', 'cannoli']
2887549032640
2887549033920
```

使用extend方法扩展列表,也可以使用 + 连接两个列表

In [5]:
```
numbers = list(range(1, 11))

# extend方法扩展了原有列表
numbers.extend([11, 12, 13])
print(numbers)

# 使用加号连接两个列表，然后创建了新的列表连接
numbers = numbers + [14, 15]

print(numbers)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
```

使用双端队列deque

In [54]:
```
from collections import deque
dq = deque(range(10))
dq.append(11)
dq.appendleft(-1)  # appendleft效率比insert(0, -1)高
print(dq)

dq.pop()
dq.popleft()
print(dq)
```

```
deque([-1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11])
deque([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [72]:
```
dq.index(5)
```

Out[72]:
```
6
```

In [55]:
```
dq.extend([11, 12, 13])
dq.extendleft([-1, -2, -3])
print(dq)
```

```
deque([-3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13])
```

In [57]:
```
dq.rotate(1)  # 向右旋转1位
print(dq)

dq.rotate(-4)  # 向左旋转1位
print(dq)
```

```
deque([12, 13, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11])
deque([-1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, -3, -2])
```

元组（Tuple）

In [59]:
```
dimensions = (200, 50)
print(dimensions[0])
print(dimensions[1])
```

```
200
50
```

遍历元组

In [64]:
```
for dimension in dimensions:
    print(dimension)
```

```
200
50
```

元组中数据是不可以修改的

In [60]:
```
dimensions[0] = 250
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[60], line 1
----> 1 dimensions[0] = 250

TypeError: 'tuple' object does not support item assignment
```

只包含一个元素的元组，必须在元素后面加上逗号

In [67]:
```
m_t = (3, )
m_t
```

Out[67]:
```
(3,)
```

元组的不可修改是相对的，如果元组包含了列表元素，该列表元素是可以被修改的

In [6]:
```
my_tuple = (1, 2, [1, 2, 3])
print(my_tuple)
my_tuple[2].append(4)
print(my_tuple)
my_tuple[2].extend([5, 6])
print(my_tuple)
my_tuple[2] = my_tuple[2] + [7, 8]
print(my_tuple)
```

```
(1, 2, [1, 2, 3])
(1, 2, [1, 2, 3, 4])
(1, 2, [1, 2, 3, 4, 5, 6])
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[6], line 7
      5 my_tuple[2].extend([5, 6])
      6 print(my_tuple)
----> 7 my_tuple[2] = my_tuple[2] + [7, 8]
      8 print(my_tuple)

TypeError: 'tuple' object does not support item assignment
```

下面的Python程序运行的结果是什么:

```
t = (1, 2, [30, 40])
t[2] += [50, 60]
```

A. t的值 (1, 2, [30, 40, 50, 60])

B. TypeError: 'tuple' object does not support item assignment

C. Neither

D. Both A and B

使用 online Python Tutor (https://pythontutor.com/)可视化Python程序的运行结果

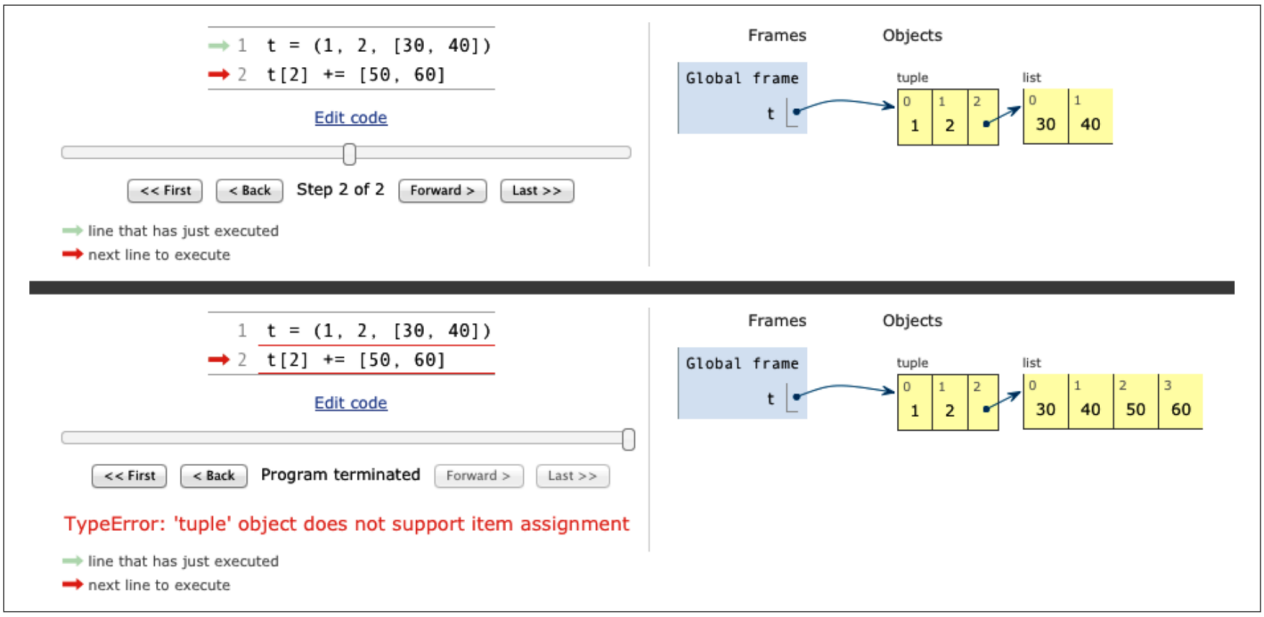

Figure 2-5. Initial and final state of the tuple assignment puzzler (diagram generated by Online Python Tutor).

使用dis查看Python字节码

In [7]:
```
import dis
dis.dis('s[a] += b')
```

```
  1           0 LOAD_NAME                0 (s)
              2 LOAD_NAME                1 (a)
              4 DUP_TOP_TWO
              6 BINARY_SUBSCR
              8 LOAD_NAME                2 (b)
             10 INPLACE_ADD
             12 ROT_THREE
             14 STORE_SUBSCR
             16 LOAD_CONST               0 (None)
             18 RETURN_VALUE
```

关键的三条指令：

- BINARY_SUBSCR
- INPLACE_ADD
- STORE_SUBSCR

0. LOAD_NAME 0 (s)：此指令将 0号 名称 s 的值从局部命名空间加载到堆栈上。
1. LOAD_NAME 1 (a)：此指令将 1号 名称 a 的值从局部命名空间加载到堆栈上。
2. DUP_TOP_TWO：此指令在堆栈上复制顶部的两个元素。
3. BINARY_SUBSCR：此指令执行订阅操作，从下面的对象中检索由堆栈顶部指定的索引处的值。它用于获取 s[a] 的值。
4. LOAD_NAME 2 (b)：此指令将 2号 名称 b 的值从局部命名空间加载到堆栈上。
5. INPLACE_ADD：此指令执行原地加法，将堆栈上的两个顶部元素相加，并将结果存储回堆栈。
6. ROT_THREE：此指令在堆栈上旋转三个元素，有效地将加法的结果移到顶部。
7. STORE_SUBSCR：此指令执行订阅赋值操作，它用于将加法的结果存储在 s[a] 中。
8. LOAD_CONST 0 (None)：此指令将 None 对象加载到堆栈上。
9. RETURN_VALUE：此指令从当前函数返回堆栈顶部的值。