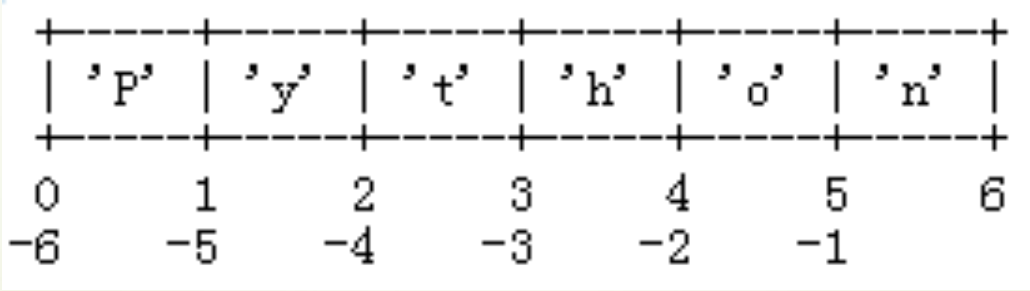
第3课　数据结构1

Python序列概述

- 列表
- 元组
- 字典
- 集合
- 高级数据结构

Python序列概述

- Python序列类似于其他语言中的数组，但功能要强大很多！
- Python中常用的序列结构有列表、元组、字符串，字典、集合，以及range等对象也支持很多类似的操作。
- 列表、元组、字符串支持双向索引，第一个元素下标为0，第二个元素下标为1，以此类推；最后一个元素下标为-1，倒数第二个元素下标为-2，以此类推。



列表list

Finite, ordered, mutable sequence of elements.

- 列表是Python中内置**有序可变序列**，列表的所有元素放在一对中括号“[]”中，并使用逗号分隔开；
- 当列表元素增加或删除时，列表对象自动进行扩展或收缩内存，保证元素之间没有缝隙；
- 在Python中，**一个列表中的数据类型可以各不相同**，可以同时分别为整数、实数、字符串等基本类型，甚至是列表、元组、字典、集合以及其他自定义类型的对象。

```
[10, 20, 30, 40]

['crunchy frog', 'ram bladder', 'lark vomit']

['spam', 2.0, 5, [10, 20]]

[['file1', 200,7], ['file2', 260,9]]
```

列表list

Finite, ordered, mutable sequence of elements.

- 什么时候使用列表？
  - ✓ 需要包含非齐次元素的集合时
  - ✓ 需要对元素进行排序时
  - ✓ 需要修改集合或添加新元素到集合时
  - ✓ 不需要按自定义的值来索引元素时
  - ✓ 元素不一定是唯一的

列表创建

- 使用“=”直接将一个列表赋值给变量即可创建列表对象
- 可以使用括号符号创建空列表或初始化列表

```
mylist1 = [] # Creates an empty list
mylist2 = [expression1, expression2, ...]
mylist3 = [expression for variable in sequence]
```

- 前两个直接**显示列表**内容，最后一个示例是**列表推导式**（list comprehension）。
- 也可以使用list()函数将元组、range对象、字符串或其他类型的可迭代对象类型的数据转换为列表。

```
mylist1 = list()
mylist2 = list(sequence)
mylist3 = list(expression for variable in sequence)
```

列表创建

- 使用“=”直接将一个列表赋值给变量即可创建列表对象
- 可以使用括号符号创建空列表或初始化列表

```
>>> a_list = ['a', 'b', 'mpilgrim', 'z', 'example']
>>> a_list = []                                    #创建空列表
```

- 也可以使用list()函数将元组、range对象、字符串或其他类型的可迭代对象类型的数据转换为列表。

```
>>> a_list = list((3,5,7,9,11))
>>> list(range(1,10,2))
[1, 3, 5, 7, 9]
>>> list('hello world')
['h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd']
>>> x = list()                                     #创建空列表
```

创建列表

- 注意！

```
# mylist1 and mylist2 point to the same list
>>> mylist1 = mylist2 = []
>>> id(mylist1)
4546619224
>>> id(mylist2)
4546619224


# mylist3 and mylist4 point to the same list
>>> mylist3 = []
>>> mylist4 = mylist3
>>> id(mylist3)
4546396032
>>> id(mylist4)
4546396032
```

列表删除

- 当不再使用时，使用del命令删除整个列表。

```
>>> del a_list
>>> a_list
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    a_list
NameError: name 'a_list' is not defined
```

列表元素的增加

append(item)：将元素添加到现有列表中，在列表**尾部**添加元素。

```
>>> mylist = [1, 5, 3, 6, 2, 4]
>>> mylist.append(9)
>>> mylist
[1, 5, 3, 6, 2, 4, 9]
```

extend(list)：添加**其他列表**中的所有项到列表的尾部

```
>>> mylist = [1, 5, 3, 6, 2, 4]
>>> mylist.extend([7,10])
>>> mylist
[1, 5, 3, 6, 2, 4, 7, 10]
```

列表元素的增加

- Python采用的是**基于值的自动内存管理**方式，当为对象修改值时，并不是真的直接修改变量的值，而是使变量指向新的值。

```
>>> a=[1,2]
>>> id(a)
4450612240
>>> a=[1,2,3]
>>> id(a)
4450612800
```

```
>>> a=[1,2]
>>> id(a)
4450612240
>>> a.append(3)
>>> id(a)
4450612240
```

列表元素的增加

- 列表中包含的是元素值的**引用**，而不是直接包含元素值。
- 如果是通过下标来修改序列中元素的值或通过可变序列对象自身提供的方法来增加和删除元素时，序列对象在内存中的**起始地址是不变的**，仅仅是被改变值的元素地址发生变化，也就是所谓的“**原地操作**”。

```
>>> id(a)
4326739912
>>> a.append(3)
>>> id(a)
4326739912
>>>
a.extend([5,6,7])
>>> id(a)
4326739912
```

列表元素的增加

insert(*pos,item*): 使用列表对象的insert()方法将元素添加至列表的指定位置。

```
>>> mylist = [1, 5, 3, 6, 2, 4]
>>> mylist.insert(2,7)
>>> mylist
[1, 5, 7, 3, 6, 2, 4]
```

- 应尽量从列表尾部进行元素的增加与删除操作。

✓ 列表的insert()可以在列表的任意位置插入元素，但由于列表的自动内存管理功能，insert()方法会涉及到插入位置之后所有元素的移动，这会影响处理速度。

✓ 课后实验：比较insert()和append()的速度（导入time模块，使用time.time()记录开始/结束时间）。

列表元素的增加

- 使用乘法来扩展列表对象，将列表与整数相乘；
- 生成一个新列表，新列表是原列表中元素的重复。

```
>>> aList = [3,5,7]
>>> bList = aList
>>> id(aList)
4326752840
>>> id(bList)
4326752840

>>> aList = aList*3
>>> aList
[3, 5, 7, 3, 5, 7, 3, 5, 7]
>>> bList
[3,5,7]
>>> id(aList)
4326739912
>>> id(bList)
4326752840
```

列表元素的增加

- 当使用*运算符将包含列表的列表重复并创建新列表时，并不创建元素的复制，而是创建已有对象的引用。因此，当修改其中一个值时，相应的引用也会被修改。

```
>>> x = [[None] * 2] * 3
>>> x
[[None, None], [None, None], [None,
None]]

>>> x[0][0] = 5
>>> x
[[5, None], [5, None], [5, None]]

>>> x = [[1,2,3]] * 3
>>> x[0][0] = 10
>>> x
[[10, 2, 3], [10, 2, 3], [10, 2, 3]]
```

递归列表

```
# 因为，列表中的元素不一定同类型
# 那么，如果自己添加自己呢？
# 递归列表！
x = [1, 2]
x.append(x)

x               # => [1, 2, [...]]
x[2]            # => [1, 2, [...]]
x is x[2]       # => True
id(x)           # => 4546619152
id(x[2])        # => 4546619152

x[1]=4          # => x=[1, 4, [...]]
x[2]            # => x[2]=[1, 4, [...]]
```

列表元素的删除

- 使用del命令删除列表中的指定位置上的元素。

```
>>> a_list = [3,5,7,9,11]
>>> del a_list[1]
>>> a_list
[3, 7, 9, 11]
```

- 使用列表的pop()方法删除并返回指定（默认为最后一个）位置上的元素，如果给定的索引超出了列表的范围则抛出异常。

```
>>> a_list = list((3,5,7,9,11))
>>> a_list.pop(1)
5
>>> a_list
[3, 7, 9 ,11]
```

列表元素的删除

- 使用列表对象的remove()方法删除首次出现的指定元素，如果列表中不存在要删除的元素，则抛出异常。

```
>>> a_list = [1,2,2,1]
>>> a_list.remove(1)
>>> a_list
[2, 2, 1]
>>> a_list.remove(3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: list.remove(x): x not in list
```

列表元素的删除：小测

```
>>> x = [1,2,1,2,1,2,1,2,1]
>>> for i in x:
        if i == 1:
            x.remove(i)
>>> x
[2, 2, 2, 2]
```

```
>>> x = [1,2,1,2,1,1,1]
>>> for i in x:
        if i == 1:
            x.remove(i)
>>> x

[2, 2, 1] #???
```

- 在删除列表元素时，Python会自动对列表内存进行收缩并移动列表元素以保证所有元素之间没有空隙。

- 每当删除一个元素之后，该元素位置后面所有元素的索引就都改变了。

列表元素的删除：小测

- 在删除列表元素时，Python会自动对列表内存进行收缩并移动列表元素以保证所有元素之间没有空隙。

- 每当删除一个元素之后，该元素位置后面所有元素的索引就都改变了。

```
x=[1,2,1,2,1,1,1]
print(x)
index = 0;
for i in x:
    print("current number:",i,", index:",index)
    print()

    index+=1

    if i==1:
        x.remove(i)

    print(x)
```

```
[1, 2, 1, 2, 1, 1, 1]
current number: 1 , index: 0

[2, 1, 2, 1, 1, 1]
current number: 1 , index: 1

[2, 2, 1, 1, 1]
current number: 1 , index: 2

[2, 2, 1, 1]
current number: 1 , index: 3

[2, 2, 1]
```

列表元素的删除：小测

```
>>> x = [1,2,1,2,1,2,1,2,1]
>>> for i in x:
        if i == 1:
            x.remove(i)
>>> x
[2, 2, 2, 2]
```

```
>>> x = [1,2,1,2,1,1,1]
>>> for i in x:
        if i == 1:
            x.remove(i)
>>> x

[2, 2, 1] #???
```

- 参考正解

```
>>> x = [1,2,1,2,1,1,1]
>>> for i in range(len(x)-1,-1,-1):
        if x[i]==1:
            del x[i]
```

列表元素的访问

- 如果所需元素的索引是已知的，则可以简单地使用括号符号索引到列表中。列表里第一个元素的“位置”或者“索引”是从“0”开始，第二个元素的则是“1”，以此类推。如果指定下标不存在，则抛出异常。

```
numbers = [2, 3, 5, 7, 11]
# 访问已知索引的元素
numbers[0] # => 2
numbers[-1] # => 11
```

- 如果索引未知，可以使用index（）方法查找项的**第一个**索引。如果找不到项，将引发异常。

```
>>> numbers.index(5)
2
>>> numbers.index(10)
Traceback (most recent call last):
  File "<stdin>", line 1, in
<module>
ValueError: 10 is not in list
```

列表元素的计数

- 使用列表对象的count()方法统计指定元素在列表对象中出现的次数

```
>>> aList
[3, 4, 5, 5.5, 7, 9, 11, 13, 15, 17]
>>> aList.count(7)
1
>>> aList.count(0)
0
>>> aList.count(8)
0
```

列表成员判断

- 使用 “in” 来判断一个值是否存在于列表中，返回结果为“True”或“False”。

```
>>> aList
[3, 4, 5, 5.5, 7, 9, 11, 13, 15, 17]
>>> 3 in aList
True
>>> 18 in aList
False

>>> bList = [[1], [2], [3]]
>>> 3 in bList
False                    #? ? ?
>>> 3 not in bList
True                     #? ? ?
>>> [3] in bList
True
```

切片 Slicing

- 切片是Python序列的重要操作之一，适用于列表、元组、字符串、range对象等类型。
- 切片使用2个冒号分隔的3个数字来完成
  - ✓ 第一个数字表示切片开始位置（默认为0）
  - ✓ 第二个数字表示切片截止（**但不包含**）位置（默认为列表长度）
  - ✓ 第三个数字表示切片的步长（默认为1），当步长省略时可以顺便省略最后一个冒号

```
mylist[start:end]  # items start to end-1
mylist[start:]     # items start to end of the array
mylist[:end]       # items from beginning to end-1
mylist[:]          # a copy of the whole array

mylist[start:end:step] # start to end-1, by step
```

切片 Slicing

- start或end参数可以是负数，表示从数组末尾开始的计数，而不是从数组开头开始的计数。

```
mylist[-1]     # last item in the array
mylist[-2:]    # last two items in the array
mylist[:-2]    # everything except the last two items
```

- 例子

```
mylist = [34, 56, 29, 73, 19, 62]
mylist[-2]              # 19
mylist[-4::2]           # [29, 19]
mylist[2:-1]            # [29, 73, 19]
```

切片 Slicing

- 更多例子

```
>>> aList = [3, 4, 5, 6, 7, 9, 11, 13, 15, 17]

>>> aList[::-1]                          #逆序的所有元素
[17, 15, 13, 11, 9, 7, 6, 5, 4, 3]

>>> aList[::2]                           #偶数位置，隔一个取一个
[3, 5, 7, 11, 15]
>>> aList[1::2]                          #奇数位置，隔一个取一个
[4, 6, 9, 13, 17]

>>> aList[3::]                           #从下标3开始的所有元素
[6, 7, 9, 11, 13, 15, 17]

>>> aList[0:100:1]                       #前100个元素??
[3, 4, 5, 6, 7, 9, 11, 13, 15, 17]
>>> aList [100:]                             #下标100之后的所有元素??
[]
>>> aList [100]                              #直接使用下标访问会发生越界
IndexError: list index out of range
```

切片 Slicing

- 修改列表内容

```
>>> aList = [3, 5, 7]
>>> aList[len(aList):] = [9]            #在尾部追加元素
>>> aList
[3, 5, 7, 9]
>>> aList[:3] = [1, 2, 3]               #替换前3个元素
>>> aList
[1, 2, 3, 9]
>>> aList[:3] = []                      #删除前3个元素
>>> aList
[9]

>>> aList = list(range(10))
>>> aList
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> aList[::2] = [0]*5                  #替换偶数位置上的元素
>>> aList
[0, 1, 0, 3, 0, 5, 0, 7, 0, 9]
>>> aList[::2] = [0]*3                  #元素个数必须一样多
ValueError: attempt to assign sequence of size 3 to extended slice of size 5
```

切片 Slicing

- 小练习

```
numbers = [1, 2, 3, 4]

>>> numbers[1:-1]
输出？
>>> numbers[::2]
输出？
>>> numbers[1:3:-1]
输出？
>>>
```

切片 Slicing

- 小练习

```
numbers = [1, 2, 3, 4]

>>> numbers[1:-1]
[2, 3]
>>> numbers[::2]
[1, 3]
>>> numbers[1:3:-1]
[]
>>>
```

列表排序

- 使用列表对象的sort方法进行**排序**，支持多种不同的排序方法。

```
>>> aList = [3, 4, 5, 6, 7, 9, 11, 13, 15, 17]
>>> import random
>>> random.shuffle(aList)
>>> aList
[3, 4, 15, 11, 9, 17, 13, 6, 7, 5]
>>> aList.sort()                          #默认是升序排序
>>> aList.sort(reverse = True)            #降序排序
>>> aList
[17, 15, 13, 11, 9, 7, 6, 5, 4, 3]
```

- 使用内置函数sorted对列表进行排序

```
>>> aList
[9, 7, 6, 5, 4, 3, 17, 15, 13, 11]
>>> sorted(aList)                         #升序排序
[3, 4, 5, 6, 7, 9, 11, 13, 15, 17]
```

列表排序

- 使用列表对象的reverse方法将元素逆序

```
>>> aList = [3, 4, 5, 6, 7, 9, 11, 13, 15, 17]
>>> aList.reverse()      #[17, 15, 13, 11, 9, 7, 6, 5, 4, 3]
```

- 使用内置函数reversed方法对列表元素进行逆序排列并返回迭代对象

```
>>> aList = [3, 4, 5, 6, 7, 9, 11, 13, 15, 17]
>>> newList = reversed(aList)                #返回reversed对象
>>> list(newList)                            #把reversed对象转换成列表
[17, 15, 13, 11, 9, 7, 6, 5, 4, 3]
>>> for i in newList:
        print(i, end=' ')                    #这里没有输出内容
                                             #因为list(),迭代对象已遍历结束
>>> newList = reversed(aList)                #重新创建reversed对象
>>> for i in newList:
        print(i, end=' ')
17 15 13 11 9 7 6 5 4 3
```

序列操作的常用内置函数

- len()：返回列表中的元素个数，同样适用于元组、字典、集合、字符串等。
- max()、 min()：返回列表中的最大或最小元素，同样适用于元组、字典、集合、range对象等。
- sum()：对列表的元素进行求和运算，对非数值型列表运算需要指定start参数，同样适用于元组、range。
- zip()函数返回可迭代的zip对象。
- enumerate(列表):枚举列表元素，返回枚举对象，其中每个元素为包含下标和值的元组。该函数对元组、字符串同样有效。

序列操作的常用内置函数

练习：

```
>>> aList = [1, 2, 3]
>>> bList = [4, 5, 6]
>>> cList = zip(a, b)

>>> list(cList)
```

???

```
>>> for item in enumerate('abcdef'):
        print(item)
```

???

序列操作的常用内置函数

```
>>> aList = [1, 2, 3]
>>> bList = [4, 5, 6]
>>> cList = zip(a, b)

>>> list(cList)

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



>>> for item in enumerate('abcdef'):
        print(item)

(0, 'a')
(1, 'b')
(2, 'c')
(3, 'd')
(4, 'e')
(5, 'f')
```

列表推导式 list comprehension

- 列表推导式（列表解析）使用非常简洁的方式来快速生成满足特定需求的列表，代码具有非常强的可读性。

```
>>> aList = [x*x for x in range(10)]
```

相当于

```
>>> aList = []
>>> for x in range(10):
        aList.append(x*x)
```

```
[function(x) for x in iterable]
[function(x) for x in iterable if cond(x)]
```

列表推导式 list comprehension

```
[function(x) for x in iterable]
```

```
[x ** 2 for x in range(10)]
# => [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
squares = []
for x in range(10):
        squares.append(x**2)
squares # => [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

列表推导式 list comprehension

```
[function(x) for x in iterable if cond(x)]
```

```
>>> squares = [(x, x**2, x**3) for x in range(0,9)
if x % 2 == 0]
>>> squares
[(0, 0, 0), (2, 4, 8), (4, 16, 64), (6, 36, 216),
(8, 64, 512)]
```

```
>> [(i,j) for i in range(5) for j in range(i)]
[(1, 0), (2, 0), (2, 1), (3, 0), (3, 1), (3, 2),
(4, 0), (4, 1), (4, 2), (4, 3)]
```

列表推导式 list comprehension

小练习：

```
>>> [[x*y for x in range(1,5)] for y in range(1,4)]
```

晕……？？？

```
>>> [[x*y for y in range(1,4)] for x in range(1,5)]
```

哭……？？？

列表推导式 list comprehension

小练习：

```
>>> [[x*y for x in range(1,5)] for y in range(1,4)]
[[1, 2, 3, 4], [2, 4, 6, 8], [3, 6, 9, 12]]

>>> [[x*y for y in range(1,4)] for x in range(1,5)]
[[1, 2, 3], [2, 4, 6], [3, 6, 9], [4, 8, 12]]
```

列表推导式 list comprehension

- 列表推导式中可以使用函数或复杂表达式

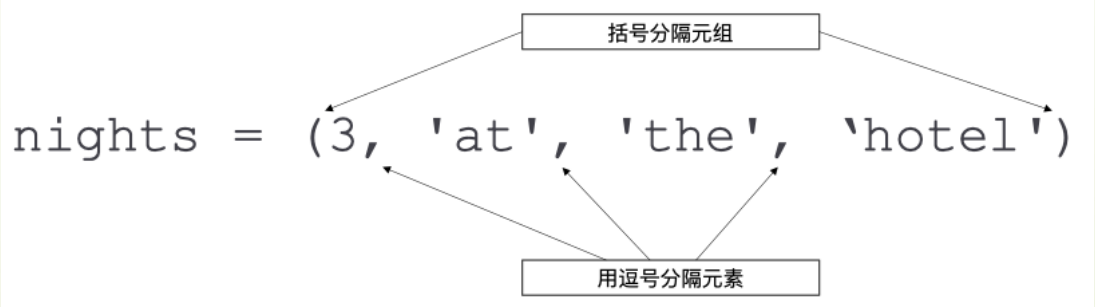
```
def f(v):
    if v%2 == 0:
      v = v**2
    else:
      v = v+1
    return v

>>> [f(v) for v in [2, 3, 4, -1] if v>0]
[4, 4, 16]
>>> [v**2 if v%2 == 0 else v+1 for v in [2, 3, 4, -1] if v>0]
[4, 4, 16]
```

元组 tuple

Finite, ordered, immutable sequence of elements.

- 元组（Tuples）属于Python中的序列类型，它是任意对象的有序集合，通过“位置”或者“索引”访问其中的元素，它具有可变长度、异构和任意嵌套的特点，
- 与列表不同的是：元组中的元素是不可修改的。
- 什么时候使用tuple？
  - 存储不需要更改元素
  - 在程序中存储关于一个对象的有限的、异构的数据时



创建元组

- 空元组可以用一组空括号创建
- 将序列类型对象传递到tuple()构造函数中
- 可以通过列出逗号分隔的值来初始化

```
>>> t1 = ()
>>> t2 = (1, 2, 3, 4)
>>> t3 =(1,2,"red",[1,2])
>>> t4 = "a", "b", "c", "d"

>>> a = (3,)
#包含一个元素的元组，最后必须多写个逗号！！！
>>> a
(3,)
>>> type(a)
<class 'tuple'>

>>> a = (3)
>>> type(a)
<class 'int'>
```

创建元组

- 使用tuple函数将其他序列转换为元组

```
>>> tuple('abcdefg')                    #把字符串转换为元组
('a', 'b', 'c', 'd', 'e', 'f', 'g')

>>> aList
[-1, -4, 6, 7.5, -2.3, 9, -11]
>>> tuple(aList)                        #把列表转换为元组
(-1, -4, 6, 7.5, -2.3, 9, -11)

>>> s = tuple()                         #空元组
>>> s
()
```

- 使用del可以删除元组对象，但不能删除元组中的元素！

对象的删除 del

- 使用del可以删除元组对象，但不能删除元组中的元素，因为元组属于不可变序列。

```
>>> x=(1,2,3)
>>> del x[1]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object doesn't support item deletion


>>> del x
>>> print(x)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
```

元组

元组也支持切片操作，例如

```
mytuple = ("p","y","t","h","o","n")
```

mytuple[:] 表示取元组mytuple的所有元素；
mytuple[3:] 表示取元组mytuple的索引为3的元素之后的所有元素；
mytuple[0:4:2] 表示元组mytuple的索引为0到4的元素，每隔一个元素取一个。

练习：

```
>>> tuple=(1,5,8,0,0,0,1,9,2,1,2)
>>> tuple[::-4]
```

输出什么？

```
>>> tuple[1:5:3]
```

输出什么？

元组

元组也支持切片操作，例如

mytuple = ("p","y","t","h","o","n")

mytuple[:] 表示取元组mytuple的所有元素；
mytuple[3:] 表示取元组mytuple的索引为3的元素之后的所有元素；
mytuple[0:4:2] 表示元组mytuple的索引为0到4的元素，每隔一个元素取一个。

练习：

```
>>> tuple=(1,5,8,0,0,0,1,9,2,1,2)
>>> tuple[::-4]
(2, 1, 8)

>>> tuple[1:5:3]
(5, 0)
```

元组

元组也支持切片操作，例如
mytuple = "P", "y", "t", "h", "o", "n”

mytuple[:] 表示取元组mytuple的所有元素；
mytuple[3:] 表示取元组mytuple的索引为3的元素之后的所有元素；
mytuple[0:4:2] 表示元组mytuple的索引为0到4的元素，每隔一个元素取一个。

练习：

```
>>> tuple=(1,5,8,0,0,0,1,9,2,1,2)
>>> tuple[::-4]
(2, 1, 8)
>>> tuple[1:5:3]
(5, 0)
```

元组与列表的区别

- 元组中的数据一旦定义就不允许更改。
- 元组没有append()、extend()和insert()等方法，无法向元组中添加元素。
- 元组没有remove()或pop()方法，也无法对元组元素进行del操作，不能从元组中删除元素。
- 从效果上看，tuple( )“冻结”列表，而list( )“融化”元组。

元组例子

- 访问元素：使用括号符号（如mytuple[2]）和支持切片
- 使用len(mytuple) 获取元组的长度
- 元组支持通用的不可变序列类型操作
  - +,*
  - in, not in
  - min(mytuple), max(mytuple), mytuple.index(item),mytuple.count(item)

```
# 不可以编辑tuple里面的元素
fish = (1, 2, "red", "blue")
fish[0] # => 1，可以访问
fish[0] = 'unicorn' # TypeError，不可以更改

# 支持不可变序列类型操作
len(fish) # => 4
fish[:-1] # => (1, 2, 'red')
"red" in fish # => True
```

封包/解包 Packing/unpacking

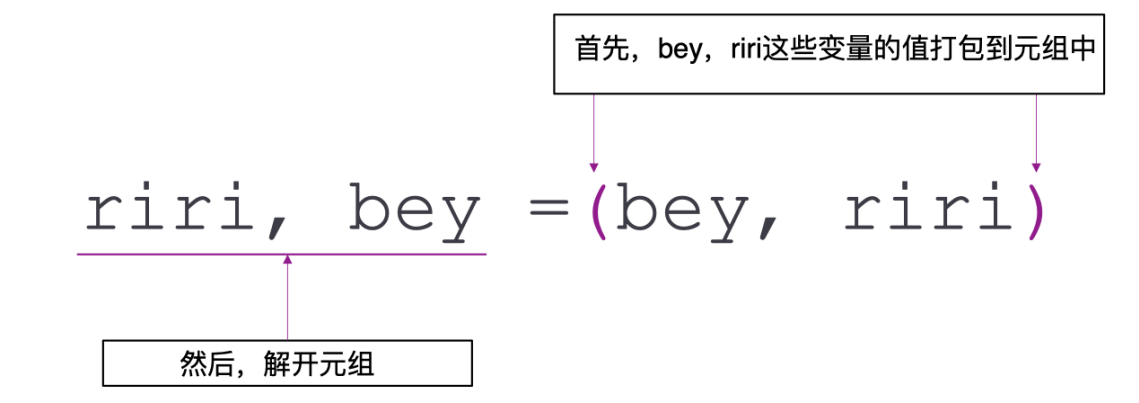
- 元组封包用于将一组项“打包”成元组。我们可以使用Python的多重赋值特性来解包元组。

```
# packing
fruit= 'apple', 'banana', 'orange'
print(fruit) # ('apple', 'banana', 'orange')
type(fruit) # => tuple

# unpacking
x, y, z = fruit
x # => 'apple'
y # => 'banana'
z # => 'orange'
```

解包

- 交换值




```
temp = riri
riri = bey
bey = temp
```

Temporary variable

```
riri = riri ^ bey
bey = riri ^ bey
riri = riri ^ bey
```

Bitwise XOR magic

```
riri, bey = bey, riri
```

Tuple unpacking…!

解包

- 可以使用序列解包功能对多个变量同时赋值

```
>>> v_tuple = (False, 3.5, 'exp')
>>> (x, y, z) = v_tuple                #x=False,y=3.5, z='exp'
>>> x, y, z = v_tuple                  #x=False,y=3.5, z='exp'

>>> x, y, z = range(3)                 #可以对range对象进行序列解包

>>> x, y, z = sorted([1, 3, 2])        #sorted()函数返回排序后的列表

>>> a, b, c = 'ABC'                    #字符串也支持序列解包
```

解包

- 使用序列解包遍历enumerate对象

```
for i, name in enumerate(['Ada', 'Bob', 'Ron']):
        print(i, name)
# 0 Ada
# 1 Bob
# 2 Ron
```

- 函数参数

```
def quick_maths(a, b, c):
        return a + b - c

boom = (2, 2, 1)
quick_maths(*boom) # => 3; Cool! 函数章节将详细学习
```

解包练习

```
>>> aList = [1,2,3]
>>> bList = [4,5,6]
>>> cList = [7,8,9]
>>> dList = zip(aList, bList, cList)
>>> for index, value in enumerate(dList):
        print(index, ':', value)
```

输出什么？

解包练习

```
>>> aList = [1,2,3]
>>> bList = [4,5,6]
>>> cList = [7,8,9]
>>> dList = zip(aList, bList, cList)
>>> for index, value in enumerate(dList):
        print(index, ':', value)
```

```
0 : (1, 4, 7)
1 : (2, 5, 8)
2 : (3, 6, 9)
```