

🔗 楼+之Python实战第10期 (/courses/1190)

列表、元组、集合与字典

简介

本节实验将学习 Python 3 中非常常用的四种数据结构：列表、元组、集合与字典。

知识点

- 列表的概念与操作
- 元组的概念与操作
- 集合的概念与操作
- 字典的概念与操作

列表

list (列表) 是一种有序的数据集合。

举例说明，在交互式环境中输入下面的内容，其中 `courses` 就是一个列表：

```
>>> courses = ['Linux', 'Python', 'Vim', 'C++']
>>> courses.append('PHP')
>>> courses
['Linux', 'Python', 'Vim', 'C++', 'PHP']
```

首先我们建立了一个列表 `courses`。然后调用列表的方法 `courses.append('PHP')` 添加元素 `PHP` 到列表末尾。你可以看到元素字符串 `PHP` 已经添加到列表的末端了。

列表中的索引类似 C 语言中数组的访问索引，可以通过索引访问到每一个列表的元素，第一个元素的索引为 0，最后一个元素的索引可以使用 -1 进行标示，这一点与上一节中的字符串的索引完全相同。

```
楼+之Python实战第10期 (/courses/1190)
>>> courses[0]
'Linux'
>>> courses[-1]
'PHP'
>>> courses[-2]
'C++'
>>> courses[9]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

超出索引的最大数字范畴，会出现越界，抛出 `IndexError` 异常，回忆下上一节的异常的内容。

如何知道列表中元素的数量呢，可以使用 `len()`：

```
>>> len(courses)
5
```

列表操作

上面的例子中我们初步接触到列表的最基本操作 `append()`，列表是有序的，所以 `append()` 就是在列表的末尾添加新的元素。

有些时候我们需要将数据插入到列表的任何位置，这时我们可以使用列表的 `insert()` 方法。

```
>>> courses.insert(0, 'Java')
>>> courses
['Java', 'Linux', 'Python', 'Vim', 'C++', 'PHP']
>>> courses.insert(1, 'Ruby')
>>> courses
['Java', 'Ruby', 'Linux', 'Python', 'Vim', 'C++', 'PHP']
```

列表方法 `count(s)` 会返回列表元素中 `s` 的数量。我们来检查一下 `Java` 这个元素在列表中出现了多少次。

```
>>> courses.count('Java')
1
```

如果你想要在列表中移除任意指定值，你需要使用 `remove()` 方法。

```
>>> courses.remove('Java')
>>> courses
['Ruby', 'Linux', 'Python', 'Vim', 'C++', 'PHP']
```

注意：如果 `Java` 出现多次，则只有第一个 `'Java'` 元素会被清除。

另外一种删除元素的方法是使用 `del` 关键字，这个关键字可以删除列表指定位置的元素，需要使用到列表中要删除元素的索引：(来自Python实战第10期 (/courses/1190))

```
>>> courses
['Ruby', 'Linux', 'Python', 'Vim', 'C++', 'PHP']
>>> del courses[-1]
>>> courses
['Ruby', 'Linux', 'Python', 'Vim', 'C++']
>>> courses.append('PHP')
>>> courses
['Ruby', 'Linux', 'Python', 'Vim', 'C++', 'PHP']
```

列表是有顺序的，我们在执行所有的列表操作的过程中都要时刻记住这一点，有序的列表可以进行反转：

```
>>> courses
['Ruby', 'Linux', 'Python', 'Vim', 'C++', 'PHP']
>>> courses.reverse()
>>> courses
['PHP', 'C++', 'Vim', 'Python', 'Linux', 'Ruby']
```

如果我们有二个列表，想合并到一起，一种方法是将其中的一个列表合并到另外一个列表的末尾位置，可以使用 `extend()`：

```
>>> new_courses = ['BigData', 'Cloud']
>>> courses.extend(new_courses)
>>> courses
['PHP', 'C++', 'Vim', 'Python', 'Linux', 'Ruby', 'BigData', 'Cloud']
```

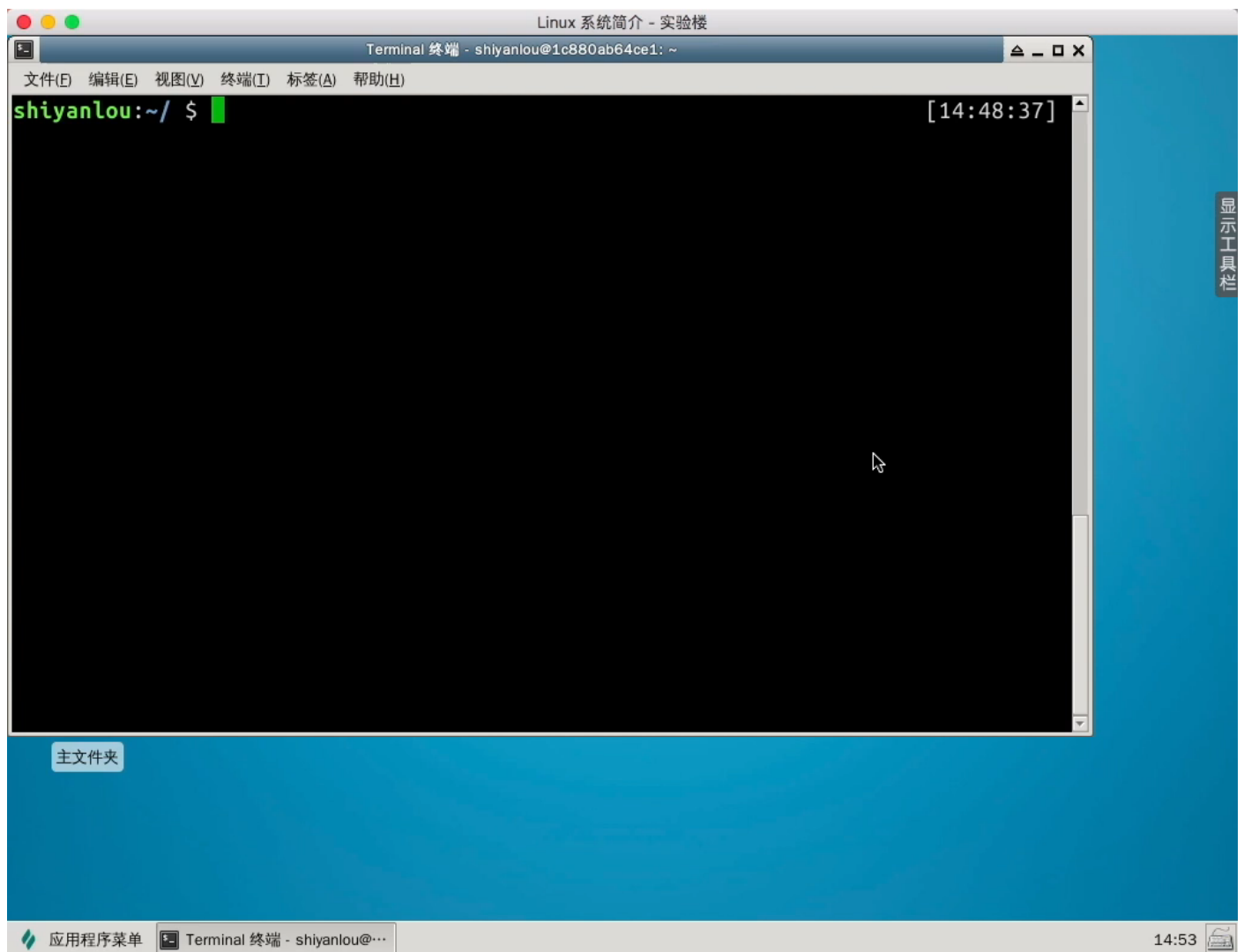
给列表排序，我们使用列表的 `sort()` 方法，排序的前提是列表的元素是可比较的，例如数字是按照大小进行排序，而字符串则会选择按照字母表的顺序进行排序，在我们的课程列表的例子中，我们先使用该函数默认的排序方法，是按照字母表顺序：

```
>>> courses
['PHP', 'C++', 'Vim', 'Python', 'Linux', 'Ruby', 'BigData', 'Cloud']
>>> courses.sort()
>>> courses
['BigData', 'C++', 'Cloud', 'Linux', 'PHP', 'Python', 'Ruby', 'Vim']
```

列表也可以使用 `pop()` 函数返回最后的一个元素，`pop()` 在返回元素的同时也会删除这个元素，传入一个参数 `i` 即 `pop(i)` 会将第 `i` 个元素弹出：

```
>>> c = courses.pop()
['BigData', 'C++', 'Cloud', 'Linux', 'PHP', 'Python', 'Ruby', 'Vim']
>>> c
'Vim'
>>> courses
['BigData', 'C++', 'Cloud', 'Linux', 'PHP', 'Python', 'Ruby']
>>> courses.pop()
'Ruby'
>>> courses.pop()
'Python'
>>> courses
['BigData', 'C++', 'Cloud', 'Linux', 'PHP']
>>> courses.pop(0)
'BigData'
>>> courses
['C++', 'Cloud', 'Linux', 'PHP']
```

列表操作视频：



- 注：视频中关于列表的 remove 方法讲解有误，以课程文档为准。
- 拓展阅读 《Python 官方针对列表的介绍文档》
(<https://docs.python.org/3/tutorial/datastructures.html#more-on-lists>)

练习题：使用列表存储数据

楼+之Python实战第10期 (/courses/1190)

本节将考察之前学习的列表、命令行参数、逻辑判断、循环、字符串相关知识点，请按照题目要求完成，点击 [下一步](#) 系统将自动检测结果。

在 `/home/shiyanlou` 目录下创建代码文件 `listtest.py`：

```
$ cd /home/shiyanlou/  
$ touch listtest.py
```

在这个文件中，我们需要实现以下需求：

1. 执行程序可以输入多个命令行参数，程序将参数分为两类，长度小于等于3的和长度大于3的，然后将分类后的参数分两行打印输出。

例如：

```
$ cd /home/shiyanlou  
$ python3 listtest.py shiyan hi louplus py 123  
hi py 123  
shiyan louplus
```

注意输出的内容不包含程序的名称，即应该从第一个参数开始处理，所以循环中使用的是 `sys.argv[1:]`。

程序完成后，点击 [下一步](#)，系统将自动检测完成结果。

元组

`tuple`（元组）是一种特殊的列表，不同点是元组一旦创建就不能修改，上述的所有会修改列表内容的操作例如 `sort()`、`append()` 等对于元组都不再适用：

```
>>> courses = ('C++', 'Cloud', 'Linux', 'PHP')  
>>> courses  
( 'C++', 'Cloud', 'Linux', 'PHP')  
>>> courses[0]  
'C++'  
>>> courses.sort()  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
AttributeError: 'tuple' object has no attribute 'sort'  
>>> del courses[0]  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: 'tuple' object doesn't support item deletion
```

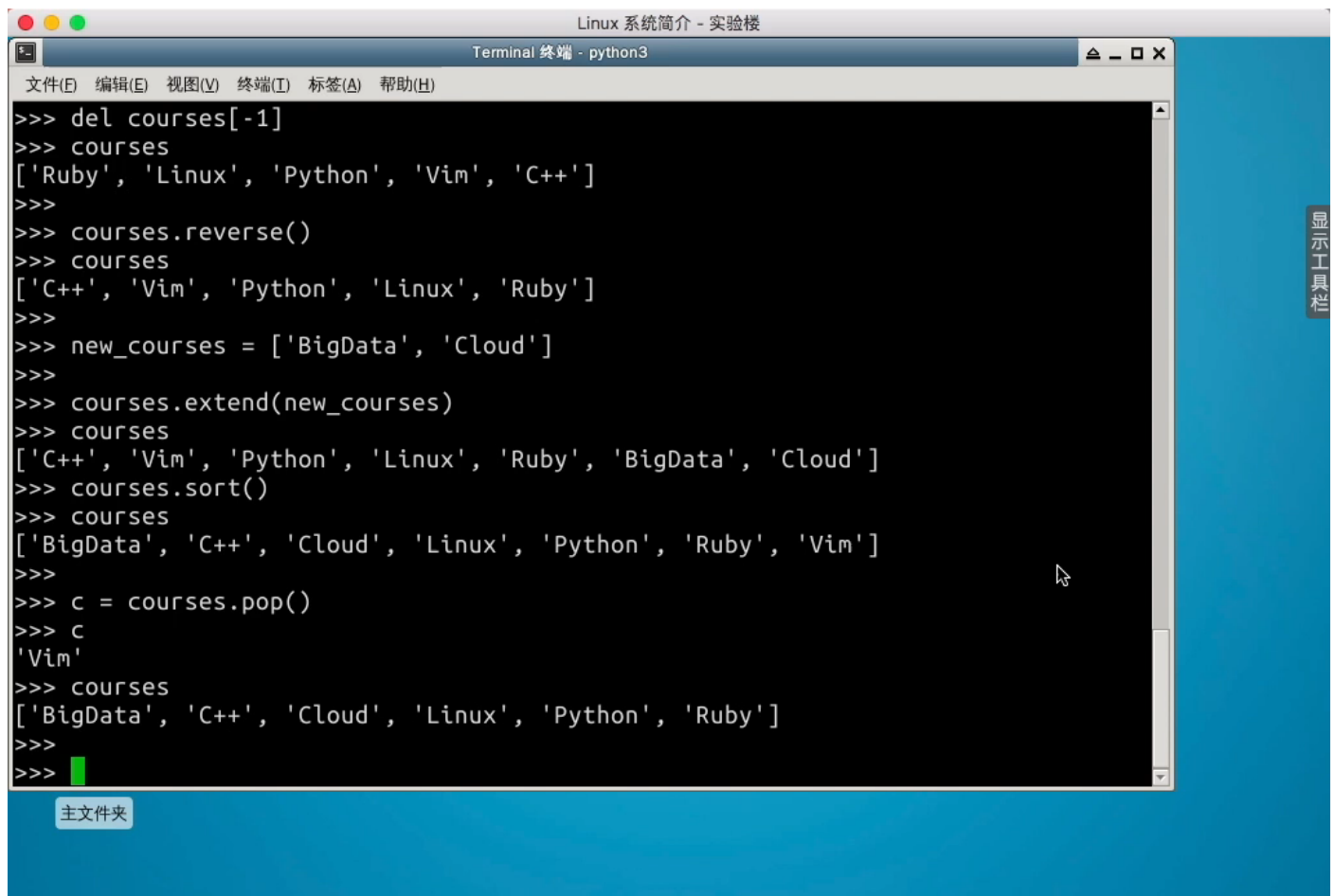
在编写程序的时候，元组比列表更安全，如果是只读的数据，尽可能使用元组，另外务必在使用过程中时刻记住元组是不可修改的，但是元组中如果包含可变的数据元素，这些数据元素是可以修改的，例如元组中包含一个列表，这个列表的内容是可以修改的：

```
>>> new_courses = ('Linux', ['BigData1', 'BigData2', 'BigData3'], 'Vim')
>>> new_courses[1]
['BigData1', 'BigData2', 'BigData3']
>>> new_courses[1].append('BigData4')
>>> new_courses
('Linux', ['BigData1', 'BigData2', 'BigData3', 'BigData4'], 'Vim')
```

最后，需要提醒下如果要创建只有一个元素的元组，是不可以直接使用括号中一个元素的，需要在元素值后面跟一个逗号：

```
>>> courses = ('Linux')
>>> courses
'Linux'
>>> type(courses)
<class 'str'>
>>> courses = ('Linux',)
>>> courses
('Linux',)
>>> type(courses)
<class 'tuple'>
```

元组操作视频：



楼+之Python实战第10期 (/courses/1190)

应用程序菜单 Terminal 终端 - python3

15:03

- 拓展阅读 《Python 官方针对元组的介绍文档》
(<https://docs.python.org/3/tutorial/datastructures.html#tuples-and-sequences>)

集合

set (集合) 是一个无序不重复元素的数据集，对比列表的区别首先是无序的，不可以使用索引进行顺序的访问，另外一个特点是不能够有重复的数据。

项目开发中，集合主要用在数据元素的去重和测试是否存在。集合还支持一些数学上的运算，例如：union (联合)，intersection (交)，difference (差) 和 symmetric difference (对称差集)。

创建集合的方法比较简单，使用大括号或者 set 函数，需要注意空的集合不能够使用 {} 创建，只能使用 set 函数，因为 {} 创建的是一个空的字典：

```
>>> courses = set()
>>> type(courses)
<class 'set'>
>>> courses = {'Linux', 'C++', 'Vim', 'Linux'}
>>> courses
{'Linux', 'Vim', 'C++'}
```

上面的代码示例中可以看到，多余的 Linux 字符串已经被自动去除。

集合还可以直接由字符串与 set 函数进行创建，会将字符串拆散为不同的字符，并去除重复的字符：

```
>>> nameset = set('shiyancelou.com')
>>> nameset
{'c', 'o', '.', 'm', 'u', 'h', 's', 'a', 'n', 'i', 'y', 'l'}
```

集合操作

上一节的例子中我们了解到集合去重的功能，如何进行测试判断是否存在呢？可以使用 in：

```
>>> 'Python' in courses
True
>>> 'Python' in courses
False
>>> 'Python' not in courses
True
```

注意 `not in` 是一个判断 `Python` 是否不在集合中的操作。`in` 操作也适用于列表和元组。

可以使用 `add()` 向集合中增加元素，使用 `remove()` 从集合中删除元素，如果元素不存在则抛出 `KeyError` 异常：

```
>>> courses
{'Linux', 'Vim', 'C++'}
>>>
>>> courses.add('Python')
>>> 'Python' in courses
True
>>> courses
{'Linux', 'Python', 'Vim', 'C++'}
>>> courses.remove('Python')
>>> 'Python' in courses
False
>>> courses.remove('Python')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'Python'
```

现在我们尝试两个集合的运算：

```
>>> set1 = {1,2,3,4}
>>> set2 = {3,4,5,6}
```

`|` 操作，存在 `set1` 中或 `set2` 中的元素，等效于 `union` 操作：

```
>>> set1 | set2
{1, 2, 3, 4, 5, 6}
>>> set1.union(set2)
{1, 2, 3, 4, 5, 6}
```

`&` 操作，返回即在 `set1` 又在 `set2` 的元素：

```
>>> set1 & set2
{3, 4}
```

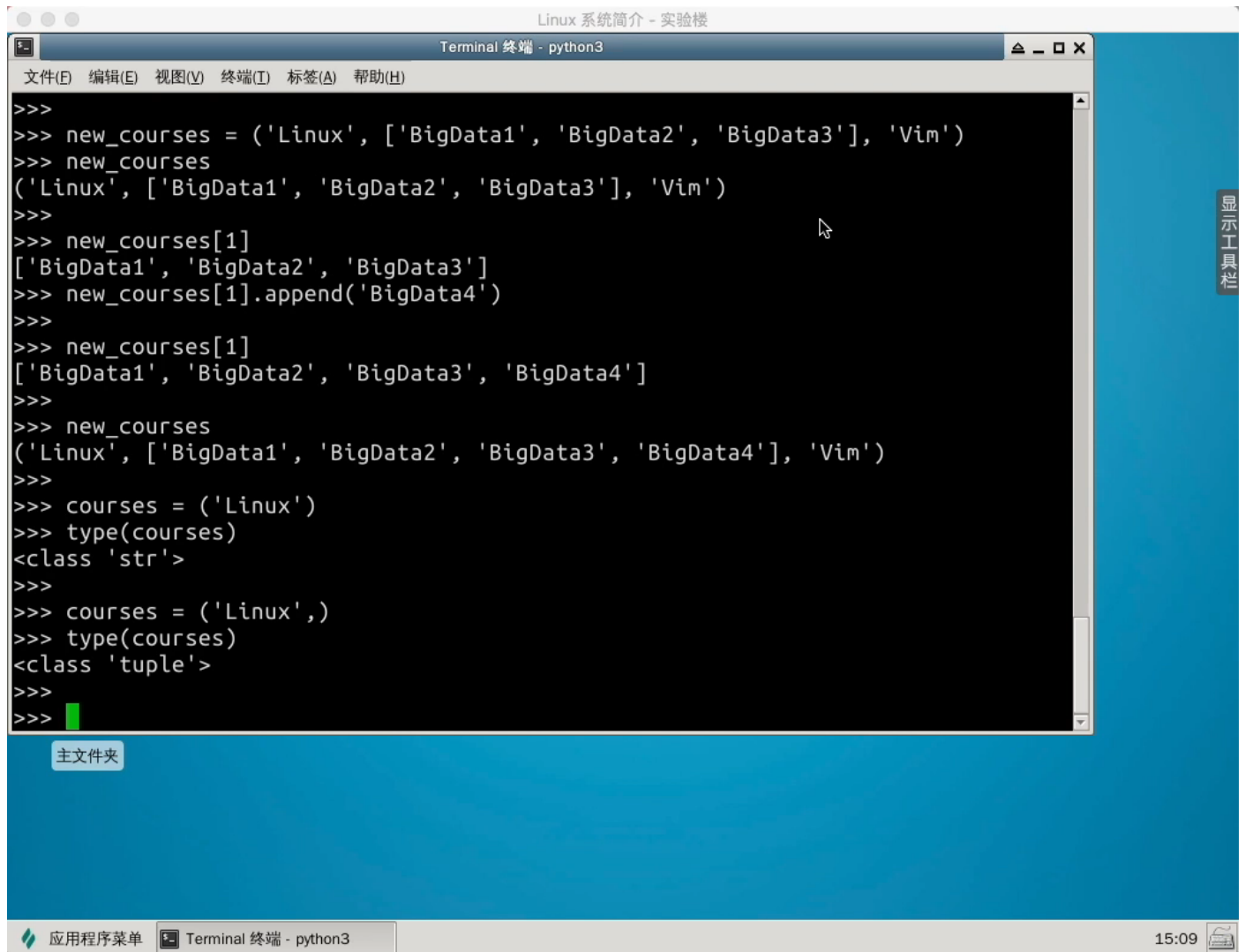
- 返回在 `set1` 不在 `set2` 的元素：

楼之Python实战第10期 (/courses/1190)
{1, 2}

^ 操作，返回只存在两个集合中的元素：

```
>>> set1 ^ set2  
{1, 2, 5, 6}
```

集合操作视频：



The screenshot shows a terminal window titled "Linux 系统简介 - 实验楼" with a "Terminal 终端 - python3" tab. The terminal displays the following Python code and its output:

```
>>>  
>>> new_courses = ('Linux', ['BigData1', 'BigData2', 'BigData3'], 'Vim')  
>>> new_courses  
(('Linux', ['BigData1', 'BigData2', 'BigData3'], 'Vim'))  
>>>  
>>> new_courses[1]  
(['BigData1', 'BigData2', 'BigData3'])  
>>> new_courses[1].append('BigData4')  
>>>  
>>> new_courses[1]  
(['BigData1', 'BigData2', 'BigData3', 'BigData4'])  
>>>  
>>> new_courses  
(('Linux', ['BigData1', 'BigData2', 'BigData3', 'BigData4'], 'Vim'))  
>>>  
>>> courses = ('Linux',)  
>>> type(courses)  
<class 'str'>  
>>>  
>>> courses = ('Linux',)  
>>> type(courses)  
<class 'tuple'>  
>>>  
>>>
```

The terminal window includes a menu bar with options like "文件(F)", "编辑(E)", "视图(V)", "终端(T)", "标签(A)", and "帮助(H)". A vertical toolbar on the right side contains a "显示工具栏" button. The bottom status bar shows "应用程序菜单", "Terminal 终端 - python3", and the time "15:09".

- 拓展阅读 《Python 官方针对集合的介绍文档》
(<https://docs.python.org/3/tutorial/datastructures.html#sets>)

练习题：数据去重

本节将考察之前学习的集合、命令行参数、逻辑判断、循环、字符串相关知识点，请按照题目要求完成，点击 下一步 系统将自动检测结果。

在 /home/shiyanlou 目录下创建代码文件 settest.py：

```
$ cd /home/shiyanlou  
$ touch settest.py
```

在这个文件中，我们需要实现以下需求：

1. 执行程序可以输入多个命令行参数，程序将打印输出去重之后的参数列表

例如：

```
$ cd /home/shiyanlou  
$ python3 settest.py shiyan 123 hi louplus 123 louplus py  
py shiyan 123 louplus hi
```

注意输出的内容不包含程序的名称，即应该从第一个参数开始处理，所以循环中使用的是 `sys.argv[1:]`。

程序完成后，点击 [下一步](#)，系统将自动检测完成结果。

字典

`dict`（字典）是无序的键值对集合。字典中的每一个元素都是一个 `key` 和一个 `value` 的组合，`key` 值在字典中必须是唯一的，因此可以很方便的从字典中使用 `key` 获得其对应的 `value` 的值。

创建字典的时候使用大括号，这一点与集合相同，先前我们已经提到 `{}` 会创建一个空字典，如果非空字典，大括号中的每个元素都是 `key:value` 这样的写法，现在我们创建一个字典保存课程的 ID 和名称，ID 作为 `key`，名称为 `value`：

```
>>> coursesdict = {1:'Linux', 2:'Vim'}  
>>> coursesdict  
{1: 'Linux', 2: 'Vim'}  
>>> coursesdict[1]  
'Linux'  
>>> coursesdict[2]  
'Vim'
```

请注意，字典的 `key` 并不一定只有数字，可以使用各种不同的类型，例如这样的字典也是合法的：

```
testdict = {1:2, 'teststr':'shiyanlou.com', 9:[1,2,3]}
```

在 `testdict` 中，其中一个 `key-value` 对是数字 1 与 2，另外一个是一个字符串，还有一个是数字与列表构成的 `key-value` 对。这些混合在一起使用，尽管看上去毫无意义，但也是可以的。

如果 key 不存在 dict[key] 则会抛出 KeyError，有的时候为了避免这种错误出现，我们会使用 get() 函数获取 key 对应的 value，如果此时 key 不存在则默认返回 None，也可以在 get() 函数中给定一个默认值，如果 key 不存在则返回默认值：

```
>>> coursesdict[2]
'Vim'
>>>
>>> coursesdict[4]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 4
>>>
>>> coursesdict[2]
'Vim'
>>> coursesdict.get(4)
>>> coursesdict.get(2)
'Vim'
>>> coursesdict.get(4, 'default')
'default'
```

同 set 一样，字典也可以使用 dict 函数进行创建，参数是一个包含若干个二元组的元组（比较绕，注意括号的数量）：

```
>>> dict_from_tuple = dict(((1, 'Linux'), (2, 'Vim'))))
>>>
>>> dict_from_tuple
{1: 'Linux', 2: 'Vim'}
```

注意，字典也是通过 [] 的方式获取值，但与列表不同的是 [] 中的内容是 key，可以为数字或其他类型，并不是列表的索引。字典是无序的，不能够通过索引进行访问。另外还需要注意字典的 key 必须为不可变的类型，列表是不能够当作 key 的。

向字典中增加元素的方法只需要为字典中某一个 key 进行赋值，这个时候如果 key 已经存在则是更新该 key 对应的 value 值，如果不存在则表示向字典中增加该 key:value：

```
>>> coursesdict[5] = 'Bash'
>>> coursesdict[6] = 'Python'
>>> coursesdict
{1: 'Linux', 2: 'Vim', 5: 'Bash', 6: 'Python'}
```

从字典中删除一个元素，只需要使用 del 删除，如果 key 不存在则抛出 KeyError：

```
>>> del coursesdict[1]
>>> del coursesdict[1]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 1
```

字典中我们可以使用 `items()` 函数获取所有的字典元素，返回得到的是 `dict_items` 类型的对象，这个对象可以使用 `for` 进行遍历，遍历的每个元素都是一个二元组，输入下面的代码的时候，注意 `print` 前的空格需要手动输入4个，这就是前面实验提到过的Python对缩进的要求：

```
>>> for key,value in coursedict.items():
...     print(key,value)
...
2 Vim
5 Bash
6 Python
>>>
```

此外，我们可以使用 `keys()` 和 `values()` 分别只获取字典中的所有 `key` 或 `value` 的列表：

```
>>> coursedict
{2: 'Vim', 5: 'Bash', 6: 'Python'}
>>> coursedict.keys()
dict_keys([2, 5, 6])
>>> coursedict.values()
dict_values(['Vim', 'Bash', 'Python'])
```

这两个返回的类型都可以使用 `for` 进行遍历。

字典中也存在 `pop(key)` 函数，可以返回 `key` 对应的 `value`，并将该 `key:value` 键值对删除：

```
>>> coursedict
{2: 'Vim', 5: 'Bash', 6: 'Python'}
>>> coursedict.pop(2)
'Vim'
>>> coursedict
{5: 'Bash', 6: 'Python'}
```

字典操作视频：





- 拓展阅读 《Python 官方针对字典的介绍文档》
(<https://docs.python.org/3/tutorial/datastructures.html#dictionaries>)

练习题：整理数据

本节将考察之前学习的字典、命令行参数、逻辑判断、循环、字符串相关知识点，请按照题目要求完成，点击 [下一步](#) 系统将自动检测结果。

在 `/home/shiyanlou` 目录下创建代码文件 `dicttest.py`：

```
$ cd /home/shiyanlou/  
$ touch dicttest.py
```

在这个文件中，我们需要实现以下需求：

1. 执行程序可以输入多个命令行参数
2. 每个命令行参数中间都有一个冒号，需要使用字符串的 `split(':')` 进行切分并存储到字典中，存储的时候冒号前面的部分作为 `key`，冒号后面的部分作为该 `key` 对应的 `value`
3. 按照示例的格式要求输出重新处理后的数据，遍历字典，输出 `key` 和 `value`

字符串的 `split()` 是用来使用传入的参数对字符串进行切分的函数，例如 `split(':')` 传入冒号，则会使用冒号对字符串进行切分，例如：

```
>>> str1 = '100:shiyang'  
>>> str1.split(':')  
['100', 'shiyang']  
>>>
```

测试示例输出：

```
楼+Python实战第10期 (/courses/1190)
$ python3 dicttest.py 100:shiyao 101:louplus 102:jack 103:lee
ID:100 Name:shiyao
ID:103 Name:lee
ID:101 Name:louplus
ID:102 Name:jack
```

注意字典是无序的，所以可以不必严格按照参数输入的顺序输出。

程序完成后，点击 [下一步](#)，系统将自动检测完成结果。

数据类型转换

有时候，我们需要对数据内置的类型进行转换，数据类型的转换，只需要将数据类型作为函数名即可。

下表为常用的数据类型转换方法：

方法	描述
<code>int(x, base)</code>	将 x 转换为一个整数
<code>float(x)</code>	将 x 转换为一个浮点数
<code>str(x)</code>	将对象 x 转换为字符串
<code>list(s)</code>	将序列 s 转换为一个列表
<code>tuple(s)</code>	将序列 s 转换为一个元组
<code>set(s)</code>	将序列 s 转换为可变集合
<code>dict(d)</code>	创建一个字典，d 必须是一个序列 (key,value) 元组

举例说明：

```
>>> int('12') # int 方法有一个默认参数 base，它的默认值为 10，即将字符串转换为十进制的数值 12
12
>>> int('12.3') # 浮点数无法进行转换，会报错 ValueError
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '12.3'
>>> int('101', 2) # 将第一个参数当做二进制的数值 101，并转换为十进制的数值 5
5
>>> int('4b3cf', 16) # 同上，将第一个参数当作十六进制的 4b3cf，并转换为十进制的 308175
308175
```

楼+之Python实战第10期 (/courses/1190)

```
12.0
>>> float('12')
12.0
>>> float('12.3')
12.3
>>> float('4a3cf') # 同样, 参数无法转换为小数, 则报错 ValueError
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: could not convert string to float: '4a3cf'
```

```
>>> str(['hello', 'shianlou']) # 此方法将任意数据类型转换为字符串
"['hello', 'shianlou']"
>>> str(12.3)
'12.3'
>>> str(('a', 'b', 'c'))
"('a', 'b', 'c')"
>>> str({'name': 'Tom', 'age': 11})
"{'name': 'Tom', 'age': 11}"
```

```
>>> t = ('a', 'b', 'c') # 将元组转换为列表
>>> list(t)
['a', 'b', 'c']
>>> d = {'name': 'Tom', 'age': 11}
>>> list(d) # 将字典的 key 值转换为列表
['name', 'age']
>>> list('hello shianlou') # 将字符串中每个字符作为一个元素存入列表
['h', 'e', 'l', 'l', 'o', ' ', 's', 'h', 'i', 'y', 'a', 'n', 'l', 'o', 'u']
```

```
>>> l = list('abc')
>>> l
['a', 'b', 'c']
>>> tuple(l) # tuple 与 list 方法类似, 将参数对象转换为元组
('a', 'b', 'c')
>>> d = {'name': 'Tom', 'age': 11}
>>> tuple(d)
('name', 'age')
>>> tuple('hello shianlou')
('h', 'e', 'l', 'l', 'o', ' ', 's', 'h', 'i', 'y', 'a', 'n', 'l', 'o', 'u')
```

```
>>> l = list('hello')
>>> set(l) # set 方法将参数转换为集合
{'o', 'l', 'e', 'h'}
>>> d = {'name': 'Tom', 'age': 11}
>>> set(d)
{'name', 'age'}
>>> set(('python', 'louplus', 'python'))
{'python', 'louplus'}
```

实验楼之Python实战第10期 (/courses/1190)

```
>>> dict((( 'name', 'Tom'), ('age', 11)))
{'name': 'Tom', 'age': 11}

>>> d = { 'name': 'Tom', 'age': 11}
>>> d1 = dict(**d) # dict 方法也可以接收关键字参数，即把字典对象作为参数
>>> d1 # 这是一个新的字典
{'name': 'Tom', 'age': 11}
>>> d
{'name': 'Tom', 'age': 11}
>>> d1 == d # 新字典与原字典值相同
True
>>> d1 is d # 但它们不是同一个对象，在内存中各拥有一段空间，仅仅是值相同而已
False
```

总结

本节实验没有需要提交到 Github 代码仓库中的代码，如果你觉得有哪些代码需要保存，可以自行提交。后续较大的示例代码、项目实验及挑战的代码我们都会要求你提交到自己的 Github 中保存。

本节实验中，我们通过一系列的动手操作，熟悉了列表，元组，集合和字典四种最常用的数据集存储方式，在实际的项目开发中，这四种都非常常用。需要在大量的练习中熟悉它们之间的区别以及应用场景。

1. 列表：可修改有序的数据集合
2. 元组：不可修改的有序的数据集合
3. 集合：无序的不重复的数据集合
4. 字典：无序的存储 key:value 键值对的数据集合

请把文档中所有的示例代码都输入一遍，尽可能不要使用复制粘贴，只有这样才能够更加熟悉代码，遇到问题的话仔细对照文档，也可以到QQ群或讨论区寻求帮助和交流。Python 3 语言基础语法并不难，难的是坚持写程序和遇到问题不含糊求根问底的学习态度。

**本课程内容，由作者授权实验楼发布，未经允许，禁止转载、下载及非法传播。*

上一节：挑战：实现个税计算器 (/courses/1190/labs/8519/document)

下一节：函数 (/courses/1190/labs/8521/document)