第5章

组合数据类型

组合数据类型能够将多个同类型或不同类型的数据组织起来，通过单一的表示使数据操作更有序、更容易。根据数据之间的关系，组合数据类型分为三大类，分别是集合类型、序列类型和映射类型。本章将通过具体案例详细介绍列表类型、元组类型、字典类型、集合类型以及迭代，并通过5个综合课业任务对组合数据类型的应用进行实例演示。

【教学目标】

- 了解集合常量并掌握集合的交并差集运算
- 了解列表、元组、字典、集合的区别
- 了解迭代器与可迭代对象
- 掌握列表的常用方法和操作
- 掌握字典的访问与遍历
- 熟练掌握元组的访问与基本操作
- 熟练迭代器的基本操作

【课业任务】

王小明想使用Django＋百度翻译API开发一个智能语音识别与翻译平台，Django是一个开放源代码的Web应用框架，由Python写成。在学习完程序控制结构后，需要使用组合数据类型对数据进行存储操作，现通过5个课业任务来完成。

课业任务5.1 使用列表类型随机分配办公室
课业任务5.2 使用元组推导式遍历出元组中的奇数
课业任务5.3 使用字典数据类型编写用户登录程序
课业任务5.4 使用集合数据类型对重复元素进行判定
课业任务5.5 使用迭代器实现输出斐波那契数列

5.1 列表类型

5.1.1 列表的特点

列表是一个有序序列，属于序列类型，可以通过位置偏移量执行索引和分片操作，具有可变性，列表的长度、元素的值可改变，即可添加或删除列表成员，以及修改列表中的值，列表还可以包含任意类型的对象，如数字、字符串、列表、元组或其他对象。

5.1.2 常用列表方法和操作

1. 列表的创建

可以使用列表常量或list()函数创建列表对象。

案例 5.1 分别使用列表常量和 list()函数创建列表,示例代码如下。

```
# 使用列表常量创建列表
li = [ ]
print(li)
# 使用 list()函数创建列表
print(list(range(10,20,2)))              # 创建数值列表对象
```

执行上述代码,结果如图 5.1 所示。

```
[]
[10, 12, 14, 16, 18]

进程已结束,退出代码0
```

图 5.1 案例 5.1 执行结果

2. 列表的删除

可以使用 del 语句或 clear 语句将整个列表或列表中的元素删除。

案例 5.2 使用 del 语句删除整个列表,示例代码如下。

```
v = ['Python']
del v
print(v)
```

执行上述代码,结果如图 5.2 所示。

案例 5.3 使用 del 语句删除列表中的元素,示例代码如下。

```
v = ['Python','php','C','C++']                    # 创建列表
del v[0]                                          # 删除索引值为 0 的元素
print(v)                                          # 输出
```

执行上述代码,结果如图 5.3 所示。

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

进程已结束,退出代码1
```

图 5.2 案例 5.2 执行结果

```
['php', 'C', 'C++']

进程已结束,退出代码0
```

图 5.3 案例 5.3 执行结果

案例 5.4 使用 clear 语句删除指定列表中的元素,示例代码如下。

```
x = [1,2,3,3,4]
print(x)
# 删除指定元素
x.remove(3)                    # 删除元素 3
print(x)
# 使用 clear 语句
x = [1,2,3,4]
x.clear()                      # 删除全部数据
print(x)                       # 输出
```

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

进程已结束,退出代码0
```

图 5.4 案例 5.4 执行结果

执行上述代码,结果如图 5.4 所示。

3. 添加列表元素

添加单个数据:可使用 append()方法在列表末尾添加单个数据。

添加多个数据:可使用 extend()方法在列表末尾添加

多个数据，参数为可迭代对象。

案例 5.5　使用 append()和 extend()方法添加列表元素，示例代码如下。

```
# 使用 append()方法
x = [1,2]
x.append('python')                    # 添加单个数据
print('添加单个元素',x)
y = [3,4]
# 使用 extend()方法
y.extend(['a','b','c'])               # 列表对象作为参数
print('添加多个元素',y)
y.extend('ege')                       # 字符串作为参数时,每个字符串作为一个数据
print(y)
```

```
添加单个元素 [1, 2, 'python']
添加多个元素 [3, 4, 'a', 'b', 'c']
[3, 4, 'a', 'b', 'c', 'e', 'g', 'e']

进程已结束,退出代码0
```

图 5.5　案例 5.5 执行结果

执行上述代码，结果如图 5.5 所示。

4. 修改列表元素

修改列表中的元素，只需要通过索引获取该元素，然后为其重新赋值即可。

案例 5.6　修改列表中的元素，示例代码如下。

```
x = ['E','G','C']
print('修改前:',x)
x[1] = 'F'                    # 修改列表中索引值为 1 的元素
print('修改后:',x)
```

执行上述代码，结果如图 5.6 所示。

```
修改前: ['E', 'G', 'C']
修改后: ['E', 'F', 'C']

进程已结束,退出代码0
```

图 5.6　案例 5.6 执行结果

5. 统计计算

使用 count()方法可以获取指定列表中元素的出现次数。

使用 index()方法可以获取指定元素在列表中首次出现的位置(即索引)。

案例 5.7　count()方法的应用，示例代码如下。

```
x = ['Python','Php','C#','Python']          # 创建列表
num = x.count('Python')                      # 获取指定元素出现的次数
print('Python 出现的个数:',num)               # 输出次数
```

执行上述代码，结果如图 5.7 所示。

案例 5.8　index()方法的应用，示例代码如下。

```
x=['Html','Python','Java','C']               # 创建列表
position = x.index('Python')                 # 获取指定元素首次出现的索引
print('Python 所在的索引值:',position)        # 输出索引
```

执行上述代码，结果如图 5.8 所示。

```
Python出现的个数:  2

进程已结束,退出代码0
```

图 5.7　案例 5.7 执行结果

```
Python所在的索引值:  1

进程已结束,退出代码0
```

图 5.8　案例 5.8 执行结果

6. 排序

使用 sort()方法可将列表排序,若列表对象全部是数字,则按数字从小到大排序;若列表对象全部是字符串,则按字典顺序排序(先对大写字母排序,再对小写字母排序)。

案例 5.9 sort()方法排序的应用,示例代码如下。

```
# 对数字列表排序
x = [88,25,98,45,20,14,1,6]
x.sort()                    # 使用 sort()方法进行排序
print(x)
# 对字符串列表排序
y = ['abh', 'Abc', 'BBC', 'CBD']
y.sort()                    # 使用 sort()方法进行排序
print(y)
```

执行上述代码,结果如图 5.9 所示。

说明:若列表中元素包含多种类型,使用 sort()方法排序则会出错。

案例 5.10 sort()方法与 reverse 结合使用进行排序,示例代码如下。

```
# 从大到小排序
x = [78,5,45,8]
x.sort(reverse = True)
print(x)
```

执行上述代码,结果如图 5.10 所示。

```
[1, 6, 14, 20, 25, 45, 88, 98]
['Abc', 'BBC', 'CBD', 'abh']

进程已结束,退出代码1
```

图 5.9 案例 5.9 执行结果

```
[78, 45, 8, 5]

进程已结束,退出代码0
```

图 5.10 案例 5.10 执行结果

5.1.3 列表推导式

列表推导式是 Python 构建列表的一种快捷方式,可以使用简洁的代码创建一个列表,相当于用 for 循环创建列表的简化版。

案例 5.11 使用列表推导式和 for 循环创建列表的对比,示例代码如下。

```
# for 循环
list_a = list()
for a in range(5):
    list_a.append(a)
print("使用 for 循环:",list_a)
# 列表推导式
list_b = [b for b in range(5)]
print("使用列表推导式:",list_b)
```

执行上述代码,结果如图 5.11 所示。

在面对一些比较复杂的循环中,我们可以使用列表推导式去解决,列表推导式比常规循环运行速度更快、更简洁。

案例 5.12　列表推导式的应用，示例代码如下。

```
# 常规循环
result = [ ]                                    # 创建空列表
for x in list(range(3,4)):
    for y in list(range(3,4)):
        for z in list(range(3,4)):
            result.append(x ** z ** y)  # 进行 3 次循环，求出 x 的 z 次方的 y 次方
print(result)
#使用列表推导式
result1 = [x ** z ** y for x in list(range(3,4)) for y in list(range(3,4)) for z in list(range(3,4))]
print(result1)
```

执行上述代码，结果如图 5.12 所示。

```
使用for循环：  [0, 1, 2, 3, 4]
使用列表推导式：  [0, 1, 2, 3, 4]

进程已结束，退出代码0
```

图 5.11　案例 5.11 执行结果

```
[7625597484987]
[7625597484987]

进程已结束，退出代码0
```

图 5.12　案例 5.12 执行结果

5.2　元组类型

5.2.1　元组的特点

元组是 Python 的一种常用数据类型，与列表类似，也是一个序列类型。不同之处在于元组的元素不能修改；表达形式上，元组使用圆括号，列表使用方括号。

元组的优点是占用内存小、处理速度快、具有不可变性（不可修改元素）。

5.2.2　创建和删除元组

1. 创建元组

在 Python 中可以用括号或 tuple()函数创建元组，可以输入多个对象，把输入的对象作为元素加入元组中。

案例 5.13　使用括号创建元组，示例代码如下。

```
tup0 = ()
tup1 = (4,9,7,78)
tup2 = ("a","c","b")
# 这种情况可以不使用括号
tup3 = "A","B","C"
print(tup0)
print(tup1)
print(tup2)
print(type(tup3))                    # 使用 type()函数查看类型
```

执行上述代码，结果如图 5.13 所示。

当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。

案例 5.14　创建只有一个元素的元组，示例代码如下。

```
tuple1 = ('a',)              # 创建只有一个元素的元组时,必须带有逗号
tuple2 = ('a')               # 创建只有一个元素的元组时,不带逗号会变成 str 类型
print(type(tuple1))          # 输出
print(type(tuple2))          # 输出
```

执行上述代码,结果如图 5.14 所示。

```
()
(4, 9, 7, 78)
('a', 'c', 'b')
<class 'tuple'>

进程已结束,退出代码0
```

图 5.13　案例 5.13 执行结果

```
<class 'tuple'>
<class 'str'>

进程已结束,退出代码0
```

图 5.14　案例 5.14 执行结果

2. 删除元组

在 Python 中,对于已经创建的元组,不再使用时,可以使用 del 语句进行删除,但是不能删除元组中某个元素。

案例 5.15　使用 del 语句删除元组,示例代码如下。

```
tuple1 = (8,7,1)             # 创建元组
del tuple1                   # 使用 del 语句删除元组
print(tuple1)                # 输出
```

执行上述代码,结果如图 5.15 所示。

```
NameError: name 'tuple1' is not defined

进程已结束,退出代码1
```

图 5.15　案例 5.15 执行结果

5.2.3　元组的访问与操作

元组访问分别为索引访问和切片操作。

元组的操作分别为连接操作、成员关系操作、比较运算操作、计数操作。

1. 索引访问

使用索引可以访问元组中指定位置的元素。

案例 5.16　索引访问的应用,示例代码如下。

```
# 正索引
tuple = ("Python","list","dict")
print('tuple 索引值为 1 是:',tuple[1])          # 取出索引值为 1 的元素

# 负索引
tuple = ("Python","list","dict")
print("tuple 索引值为 - 1 是:",tuple[ - 1])     # 取出索引值为 - 1 的元素
```

执行上述代码,结果如图 5.16 所示。

```
tuple索引值为1是:  list
tuple索引值为1是:  dict

进程已结束,退出代码0
```

图 5.16　案例 5.16 执行结果

2. 切片操作

元组中的切片取值遵守“左闭右开”的规则。例如，当切片为[9：15]时，则取不到索引为 15 的值。

案例 5.17　切片操作的应用，示例代码如下。

```
tuple0 = (99,88,77,66,55,44,33)          # 创建元组
tuple1 = tuple0[1:5]                     # 左闭右开
print(tuple1)                            # 输出
```

执行上述代码，结果如图 5.17 所示。

```
(88, 77, 66, 55)

进程已结束,退出代码0
```

图 5.17　案例 5.17 执行结果

3. 连接操作

元组中的元素值是不可修改的，但是可以对元组进行连接组合。

(1) 使用 * 符号进行重复拼接操作。

案例 5.18　使用 * 符号进行连接，示例代码如下。

```
A = ('php','good')
print(A * 2)
```

执行上述代码，结果如图 5.18 所示。

(2) 使用＋符号进行重复拼接操作。

案例 5.19　使用＋符号进行拼接操作，示例代码如下。

```
tuple1 = (1,2,3,4)
tuple2 = ("Hello 啊","5678","Hi","9,10,11,12")
print(tuple1 + tuple2)
```

执行上述代码，结果如图 5.19 所示。

```
('php', 'good', 'php', 'good')

进程已结束,退出代码0
```

图 5.18　案例 5.18 执行结果

```
(1, 2, 3, 4, 'Hello啊', '5678', 'Hi', '9,10,11,12')

进程已结束,退出代码0
```

图 5.19　案例 5.19 执行结果

4. 成员关系操作

成员关系操作主要是 in 和 not in 运算符操作。对于 in 运算符，如果元组中包含给定的元素，则返回 True，否则返回 False；对于 not in 运算符，如果指定元素不在元组中，则返回 True，否则返回 False。

案例 5.20　in 和 not in 运算符的应用，示例代码如下。

```
tuple0 = (65,"five","nice")
print("five"in tuple0)
print("中国"not in tuple0)
print("65"in tuple0)
print("nice"not in tuple0)
```

执行上述代码，结果如图 5.20 所示。

5. 比较运算操作

元组之间可以通过>、<、>=、<=等运算符进行比较运算,数字类型的比较是从两个元组的第 1 个元素开始比较大小的,如 t4=(1,2),t5=(2,2),则 t5>t4;字符串类型的比较是以 26 个英文字母的顺序作为比较方式,字母顺序越靠后越大,另外,小写字母大于大写字母。

```
True
True
False
False

进程已结束,退出代码0
```

图 5.20 案例 5.20 执行结果

案例 5.21 比较元组的操作,示例代码如下。

```
t0 = (1,2,3)
t1 = (1,3,2)
print(t0 < t1)                    # 输出布尔值,判断对错
t2 = ('a','b')
t3 = ('b','b')
print(t2 < t3)                    # 输出布尔值,判断对错
```

```
True
True

进程已结束,退出代码0
```

图 5.21 案例 5.21 执行结果

执行上述代码,结果如图 5.21 所示。

说明:不同类型的数据不能进行比较,否则会抛出异常。

6. 计数

可以使用内置函数对元组进行计数,如 len()、max()、min()、sum()等。

1) len()函数

使用 len()函数可以获取元组长度。

案例 5.22 使用 len()函数获取元组长度,示例代码如下。

```
A = ("中国","伟大","复兴","正向我们走来")
print(len(A))
```

执行上述代码,结果如图 5.22 所示。

```
4

进程已结束,退出代码0
```

图 5.22 案例 5.22 执行结果

2) max()函数

使用 max()函数可以获取元组中元素的最大值。

案例 5.23 使用 max()函数获取元组中元素的最大值,示例代码如下。

```
a = (48,1000,79,100)
print("最大值为:",max(a))
```

```
最大值为:  1000

进程已结束,退出代码0
```

图 5.23 案例 5.23 执行结果

执行上述代码,结果如图 5.23 所示。

3) min()函数

使用 min()函数可以获取元组中元素的最小值。

案例 5.24 使用 min()函数获取元组中元素的最小值,示例代码如下。

```
a = (48,1000,79,100)
print("最小值为:",min(a))
```

执行上述代码,结果如图 5.24 所示。

4）sum()函数

顾名思义，sum()函数作为 Python 内置函数，可以对迭代器中的所有元素求和。

案例 5.25　sum()函数求和，示例代码如下。

```
a = (48,1000,79,100)
print("总数为:",sum(a))
```

执行上述代码，结果如图 5.25 所示。

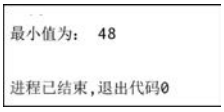

图 5.24　案例 5.24 执行结果

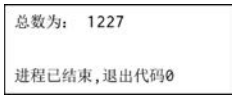

图 5.25　案例 5.25 执行结果

5.2.4　元组推导式

在元组中，使用元组推导式可以快速生成一个元组，其表现形式和列表推导式类似，只是将列表推导式中的“[ ]”改为“()”。

案例 5.26　使用元组推导式生成元组，示例代码如下。

```
import random                          # 导入 random 库
tuple0 = (random.randint(10,30) for i in range(5))
print(tuple0)                          # 此时输出为一个生成器
#print(tuple(tuple0))                  # 可以使用 tuple()函数转换为元组类型进行输出
print(tuple0.__next__())               # 使用 next()函数返回下一个元素
print(tuple0.__next__())
print(tuple0.__next__())
print(tuple0.__next__())
print(tuple0.__next__())
print(tuple(tuple0))                   # 使用 next()函数访问完后,tuple0 变为空元组
```

执行上述代码，结果如图 5.26 所示。

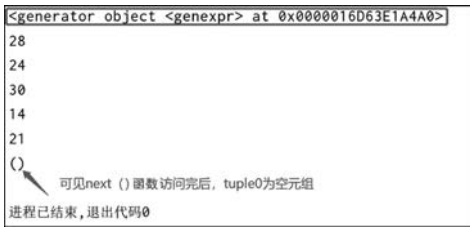

图 5.26　案例 5.26 执行结果

说明：若想再使用该生成器对象，必须重新创建一个生成器，因为遍历后的原生成器对象已经不存在了。

5.3　字典类型

5.3.1　字典的定义

在 Python 中，字典是另一种可变容器模型，可存储任意类型对象，如字符串、数字、元组等其他容器模型。因为字典是无序的，所以不支持索引和切片。

字典中数据以键-值对(key-value)的形式存在,键和值之间用冒号隔开,字典之间元素用逗号隔开,包含在“{}”中。

字典的特点如下。

(1) 字典是通过键而不是索引来读取。

(2) 字典是可变的。

(3) 字典是任意对象的无序集合。

(4) 字典中的键是唯一的。

(5) 字典中的键是不可变的。

5.3.2 常用字典方法和操作

1. 创建字典

字典包含两部分,即“键”和“值”,并且在键和值之间用冒号隔开(一定要注意是英文状态下的冒号),相邻的两个字典用逗号隔开,所有字典元素放在“{}”中。

案例 5.27 赋值创建字典,示例代码如下。

```
# 创建空字典
dict = {}
print(dict)
# 赋值创建字典
stu_info = {'num':'1998','name':'王小明','sex':'男','age':18}          # 创建字典
print(stu_info)                                                        # 查看字典
```

执行上述代码,结果如图 5.27 所示。

案例 5.28 通过确定的键-值对创建字典,示例代码如下。

```
# 第 1 种形式
list = [('ID',1),('name',2),('num',3)]                    # 创建列表
dic1 = dict(list)                                         # 使用 dict()函数转换为字典
print(dic1)                                               # 输出字典
# 第 2 种形式
dic0 = dict(a = 99,b = 45,c = 1.1)
print(dic0)
```

执行上述代码,结果如图 5.28 所示。

```
{}
{'num': '1998', 'name': '王小明', 'sex': '男', 'age': 18}

进程已结束,退出代码0
```

图 5.27 案例 5.27 执行结果

```
{'ID': 1, 'name': 2, 'num': 3}
{'a': 99, 'b': 45, 'c': 1.1}

进程已结束,退出代码0
```

图 5.28 案例 5.28 执行结果

案例 5.29 使用 zip()函数进行映射创建字典,示例代码如下。

```
dic = dict(zip('ABC',[1,2,3])) # 通过 zip()函数进行映射,然后用 dict()函数进行转换为字典
print(dic)                     # 输出字典
```

执行上述代码,结果如图 5.29 所示。

2. 删除字典

使用 del 语句可以删除字典或删除字典中指定键-值对，也可以使用 clear()函数删除整个字典。

```
{'A': 1, 'B': 2, 'C': 3}

进程已结束,退出代码0
```

图 5.29　案例 5.29 执行结果

案例 5.30　使用 del 语句删除字典和字典中指定键-值对，示例代码如下。

```
dic = dict(zip('ABC',[1,2,3]))    # 通过 zip()函数进行映射,然后用 dict()函数转换为字典
print(dic)
del dic["B"]                      # 删除 dic 中键为"B"的元素
print(dic)
del dic                           # 删除整个字典 dic
print(dic)                        # 此处打印会抛出异常,因为 dic 已经被删除
```

执行上述代码，结果如图 5.30 所示。

```
{'A': 1, 'B': 2, 'C': 3}
{'A': 1, 'C': 3}        可看到B元素已经被删除
Traceback (most recent call last):
  File "C:/Users/86136/PycharmProjects/pythonProject7/main.py'
    print(dic)
NameError: name 'dic' is not defined        证明dic已经被删除
```

图 5.30　案例 5.30 执行结果

3. 访问字典

在字典中，我们可以访问字典的键，输出对应的值；也可以使用 print()函数将字典内容直接输出，与访问列表、元组类似。

案例 5.31　使用 print()函数访问字典，示例代码如下。

```
dict1 = {"小红":12,"Alan":13,"puls":14}
print(dict1)
```

执行上述代码，结果如图 5.31 所示。

案例 5.32　通过键访问字典的值，示例代码如下。

```
dict1 = {"小红":12,"Alan":13,"puls":14}
print("Alan 的年龄:",dict1["Alan"])
```

执行上述代码，结果如图 5.32 所示。

```
{'小红': 12, 'Alan': 13, 'puls': 14}

进程已结束,退出代码0
```

图 5.31　案例 5.31 执行结果

```
Alan的年龄: 13

进程已结束,退出代码0
```

图 5.32　案例 5.32 执行结果

说明：在使用 print()函数访问字典时，如果指定的键不存在，会抛出异常。

案例 5.33　使用字典对象的 get()方法获取指定键的值，示例代码如下。

```
dict1 = {"小红":12,"Alan":13,"puls":14}
print("Alan 的年龄:",dict1.get('Alan'))              # 将字典中 Alan 的值输出
```

执行上述代码,输出结果与案例 5.32 相同。

4. 遍历字典

(1) 使用 items()方法遍历字典的键-值对。

items()方法可以获取字典的键-值对列表,语法格式如下。

```
dictionary.items()
```

案例 5.34 使用 items()方法遍历字典的键-值对,示例代码如下。

```
# 通过 for 循环用 items()方法进行遍历
dict1 = {"php":'88',"html":'69',"sql":'99'}
for item in dict1.items():
    print(item)
```

执行上述代码,结果如图 5.33 所示。

案例 5.35 使用 items()方法单独遍历字典中的键或值,示例代码如下。

```
# 遍历字典中的键
dict1 = {"php":'88',"html":'69',"sql":'99'}
for key, value in dict1.items():                    # 通过 for 循环用 items()方法进行遍历
    print(key)
print('----------------------------------')
# 遍历字典中的值
dict2 = {"php":'88',"html":'69',"sql":'99'}
for key, value in dict2.items():                    # 通过 for 循环用 items()方法进行遍历
    print(value)
```

执行上述代码,结果如图 5.34 所示。

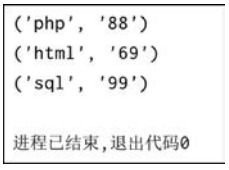

图 5.33 案例 5.34 执行结果

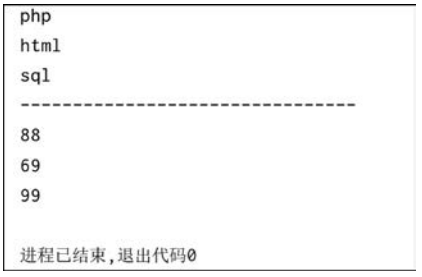

图 5.34 案例 5.35 执行结果

(2) 使用 keys()和 values()函数遍历字典的键和值。

案例 5.36 使用 keys()和 values()函数分别遍历字典的键和值,示例代码如下。

```
# 通过 for 循环用 keys()函数对字典的键进行遍历
dict1 = {"age":'88',"num":'69',"ID":'99'}
for item in dict1.keys():
    print(item)
# 通过 for 循环用 values()函数对字典的值进行遍历
dict1 = {"age":'88',"num":'69',"ID":'99'}
for item in dict1.values():
    print(item)
```

执行上述代码，结果如图 5.35 所示。

5. 添加字典

(1) 根据键-值对添加。

若键已存在，则覆盖新的键-值对；若当键不存在，则会新增键-值对。

案例 5.37　根据键-值对添加字典，示例代码如下。

```
dict1 = {"name":'王大明',"age":19}                    # 创建字典
# 添加字典中 age 的值，会直接覆盖原来的值
dict1['name'] = 'Alan'
# 由于原字典中没有 sex 的键-值对，会新建一个键-值对
dict1['sex'] = '男'
print(dict1)
```

执行上述代码，结果如图 5.36 所示。

```
age
num
ID
88
69
99

进程已结束，退出代码0
```

图 5.35　案例 5.36 执行结果

```
{'name': 'Alan', 'age': 19, 'sex': '男'}

进程已结束，退出代码0
```

图 5.36　案例 5.37 执行结果

(2) 使用 update()方法添加。

使用 update()方法将新字典中所有键-值对全部添加到旧字典对象上，语法格式如下。

```
dictionary1.update(dictionary2)
```

案例 5.38　使用 update()方法添加字典，示例代码如下。

```
dict1 = {"name":'Alan',"age":17}              # 创建字典
dict2 = {'ID':12345,"nation":'CN'}            # 创建新字典
dict1.update(dict2)                           # 使用 update()方法添加
print(dict1)                                  # 输出
```

执行上述代码，结果如图 5.37 所示。

```
{'name': 'Alan', 'age': 17, 'ID': 12345, 'nation': 'CN'}

进程已结束，退出代码0
```

图 5.37　案例 5.37 执行结果

6. 修改字典

(1) 重新赋值所要修改的键。

案例 5.39　使用[key]方法，与添加字典元素类似，示例代码如下。

```
dict0 = {"name":'Alan',"state":"上海"}
dict0["state"] = '广东'                  # 修改 key 的值
print(dict0)
```

执行上述代码,结果如图 5.38 所示。

(2) 使用 update()方法修改。

案例 5.40 使用 update()方法或解包字典模式修改字典,示例代码如下。

```
dict1 = {"name":'Alan',"state":"广州"}
# 使用关键字参数
dict1.update(name = "Harden",age = 22)
# 解包字典模式,等同于 dict1.update(sex = '女')
dict1.update( **{"sex":'男'})
print(dict1)
```

执行上述代码,结果如图 5.39 所示。

```
{'name': 'Alan', 'state': '广东'}

进程已结束,退出代码0
```

图 5.38 案例 5.39 执行结果

```
{'name': 'Harden', 'state': '广州', 'age': 22, 'sex': '男'}

进程已结束,退出代码0
```

图 5.39 案例 5.40 执行结果

5.3.3 字典推导式

Python 字典推导式是从一个或多个迭代器快速、简洁地创建数据类型的一种方法,它将循环和条件判断结合,从而避免语法冗长的代码,提高代码运行效率。语法格式如下。

```
dict = {表达式 for 迭代变量 in 可迭代对象 [if 条件表达式]}
```

1. 生成指定范围的数值字典

在字典推导式中,可以借助列表、元组、字典、集合以及 range 区间,快速生成符合需求的字典。

案例 5.41 生成 key 长度小于 5 的字典,示例代码如下。

```
dict01 = ['Alan',"Dawei"]
# 以循环变量作为 key,key 长度为 value 值,len(key)作为条件
dict = {key:len(key) for key in dict01 if len(key)< 5}
print(dict)
```

```
{'Alan': 4}

进程已结束,退出代码0
```

图 5.40 案例 5.41 执行结果

执行上述代码,结果如图 5.40 所示。

2. 使用 zip()函数生成字典

案例 5.42 使用 zip()函数生成字典,示例代码如下。

```
list1 = ['Alan','Harden','Dawei','Boys']
list2 = ['19','65','19','42']
dict1 = {key + '年龄':value for key,value in zip(list1,list2)}
print(dict1)
```

执行上述代码,结果如图 5.41 所示。

```
{'Alan年龄': '19', 'Harden年龄': '65', 'Dawei年龄': '19', 'Boys年龄': '42'}

进程已结束,退出代码0
```

图 5.41 案例 5.42 执行结果

5.4　集合类型

集合中的元素无序，每个元素唯一，不存在相同元素；集合元素不可更改，不能是可变数据类型；集合用大括号“{}”表示，元素间用逗号分隔；可使用{}或 set()函数创建集合。

5.4.1　创建集合

使用 set()函数创建集合。

案例 5.43　使用 set()函数创建集合，示例代码如下。

```
Set1 = {7,2,55,3}                    # 赋值创建集合
Li = ['Alan','CBD','NBA','CBA']      # 创建列表
Str1 = ('123abc')                    # 创建字符串
print(type(Set1))                    # 测试集合对象的类型名称
print(set(Li))                       # 用列表常量作为参数创建集合对象
print(set(Str1))                     # 用字符串常量作为参数创建集合对象
```

执行上述代码，结果如图 5.42 所示。

```
<class 'set'>
{'Alan', 'CBD', 'CBA', 'NBA'}
{'c', 'a', '3', '2', '1', 'b'}

进程已结束,退出代码0
```

图 5.42　案例 5.43 执行结果

5.4.2　删除集合

可使用 pop()函数删除指定位置的对象，省略位置时删除集合最后一个对象，同时返回被删除对象；可使用 clear()函数删除集合中的全部数据；可使用 discard()函数删除集合中指定的元素。

案例 5.44　使用 pop()、clear()和 discard()函数删除集合，示例代码如下。

```
x = {'a','b'}
print(x)
x.pop()              # 从集合中随机删除一个元素,并返回该元素
print(x)
x.clear()            # 删除集合中的全部元素
print(x)
# 使用 discard()函数进行删除
y = {1,2,3}
y.discard(1)         # 使用 discard()函数删除指定元素 1 的值
print(y)
```

执行上述代码，结果如图 5.43 所示。

5.4.3　集合的运算

在 Python 中，集合类型与数学中的集合概念保持一致，集合对象支持求长度、判断包含、求差集、求并集、求交集、求对称差和比较等运算。

1. 使用 len()方法计算集合长度

案例 5.45　使用 len()方法计算集合长度，示例代码如下。

```
# 计算集合中元素的个数(集合长度)
x = {'a','bc',455,'Alan'}
print('长度:',len(x))
```

执行上述代码,结果如图 5.44 所示。

```
{'b', 'a'}
{'a'}
set()
{2, 3}

进程已结束,退出代码0
```

图 5.43　案例 5.44 执行结果

```
长度:  4

进程已结束,退出代码0
```

图 5.44　案例 5.45 执行结果

案例 5.46　使用对称差运算符^求 x 和 y 两个集合的对称差,以及对两个集合进行比较运算,示例代码如下。

```
x = {1,'b',8}
y = {1,'a',5,'u'}
# 用属于 x 但不属于 y 以及属于 y 但不属于 x 的元素创建新集合
print('对称差:',x ^ y)
# 判断子集和超集的关系,x 是 y 的子集时返回 True,否则返回 False
print('比较:',x < y)
```

```
对称差:  {'b', 5, 8, 'u', 'a'}
比较:  False

进程已结束,退出代码0
```

图 5.45　案例 5.46 执行结果

执行上述代码,结果如图 5.45 所示。

2. 交、差、并的运算

(1) 差集:用属于 x 但不属于 y 的元素创建新集合。

(2) 并集:用 x 和 y 两个集合中的所有元素创建新集合。

(3) 交集:用同时属于集合 x 和 y 的元素创建新集合。

案例 5.47　求集合 x 与 y 的交集、差集、并集,示例代码如下。

```
x = {3,2,1,'就是我没有呀','hello'}
y = {1,2,3,'hello'}
# 求差集
print(x - y)
# 求并集
print(x | y)
# 求交集
print(x & y)
```

执行上述代码,结果如图 5.46 所示。

3. 集合的增、删、改

在集合中,元素的值是不允许修改的,但可以复制集合、为集合添加或删除元素以及修改元素。

案例 5.48　集合的增、删、改,示例代码如下。

```
x = {5,7}
# 复制集合对象
y = x.copy()
print(y)
# 为集合添加一个元素
```

```
x.add('加上我')
print(x)
# 为集合添加多个元素
x.update({'a','b'})
print(x)
# 从集合中删除指定元素
x.remove('加上我')
print(x)
x.discard('a')
print(x)
x.remove('b')
print(x)
```

执行上述代码，结果如图 5.47 所示。

```
{'就是我没有呀'}
{1, 2, 3, 'hello', '就是我没有呀'}
{'hello', 1, 2, 3}

进程已结束,退出代码0
```

图 5.46　案例 5.47 执行结果

```
{5, 7}
{'加上我', 5, 7}
{'加上我', 5, 7, 'b', 'a'}
{5, 7, 'b', 'a'}
{5, 7, 'b'}
{5, 7}

进程已结束,退出代码0
```

图 5.47　案例 5.48 执行结果

说明：集合元素是不可修改的，因此不能将可变对象放入集合中，如集合、列表和字典；但元组可以作为一个元素放入集合中。

5.4.4　冻结集合

Python 提供了一种特殊的集合——冻结集合(frozenset)，即一旦创建就不可以进行修改的集合，可将其作为其他集合的元素。冻结集合除了不能修改之外，其余和集合一样。

案例 5.49　冻结集合的应用，示例代码如下。

```
x = frozenset([9,3,6])                    # 创建冻结集合
print(x)                                  # 输出冻结集合
y = set([4,5])
y.add(x)                                  # 将冻结集合作为元素加入另一个集合
print(y)
x.add(1)                                  # 为冻结集合添加元素会抛出异常
print(x)
```

执行上述代码，结果如图 5.48 所示。

```
frozenset({9, 3, 6})
{4, 5, frozenset({9, 3, 6})}
Traceback (most recent call last):
  File "C:/Users/86136/PycharmProjects/pythonProject7/main.py"
    x.add(1)                              #为冻结集合添加元素会抛出异常
AttributeError: 'frozenset' object has no attribute 'add'
```

图 5.48　案例 5.49 执行结果

5.4.5 列表、元组、字典与集合的区别

列表、元组、字典与集合的区别如表 5.1 所示。

表 5.1 列表、元组、字典与集合的区别

比较内容	列 表	元 组	字 典	集 合
英文名称	list	tuple	dict	set
可否读写	读写	只读	读写	读写
可否重复	是	是	是	否
存储方式	值	值	键-值对	键(不可重复)
是否有序	有序	有序	无序	无序
添加方式	append()函数	只读	d['key']='value'	add()函数
可否切片	是	是	否	否

5.5 迭代

迭代是访问集合元素的一种方式。迭代器是一个可以记住遍历位置的对象。迭代器对象从集合的第 1 个元素开始访问,直到所有元素被访问完结束。迭代器只能向前,不能后退。

5.5.1 迭代器的特点和优势

迭代器具有三大特点,如下所示。

(1) __iter__方法返回迭代器本身。

(2) next()方法返回容器的下一个元素。

(3) 当没有下一个元素时,会抛出 StopIteration 异常。

迭代器具有四大优势,如下所示。

(1) 在处理文件对象时特别有用,因为文件也是迭代器对象。

(2) 不依赖索引取值。

(3) 节约内存(循环过程中,数据不用一次读入)。

(4) 实现惰性计算(需要时再取值计算)。

5.5.2 迭代器的常见基本操作

常见的可迭代对象有字符串(str)、列表(list)、元组(tuple)、字典(dict)、集合(set)、文件。

1. 字符串迭代

(1) 使用 iter()函数进行迭代。

案例 5.50 使用 iter()函数进行迭代,然后使用 next()函数返回下一个元素,示例代码如下。

```
# 使用 iter()函数生成迭代器
str1 = iter('Python')
print(str1)
for item in range(5):
    # 使用 next()函数返回迭代元素,无数据则抛出 StopIteration 异常
    print(next(str1))
```

执行上述代码，结果如图 5.49 所示。

(2) 使用 for 循环进行迭代。

案例 5.51　使用 for 循环对字符串进行迭代，示例代码如下。

```
str1 = "CBD"
# 使用 for 循环对字符串进行迭代
for item in str1:
    print(item)
```

执行上述代码，结果如图 5.50 所示。

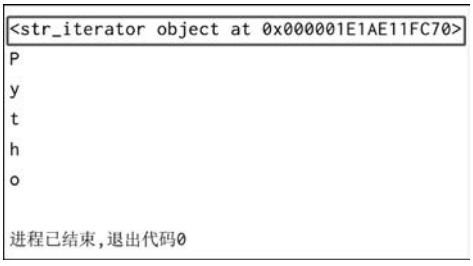

图 5.49　案例 5.50 执行结果

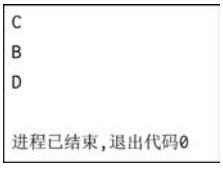

图 5.50　案例 5.51 执行结果

2. 列表迭代

(1) 通过序列项和索引进行迭代。

案例 5.52　通过序列项和索引对列表进行迭代，示例代码如下。

```
# 通过序列项进行迭代
list1 = [8,1,2,0]                          # 创建列表
for item in list1:                          # 对 list1 进行迭代
    print(item)
print('----------------------------')
# 通过项和索引进行迭代
list1 = ['a','b','c']                      # 创建列表
for index,item in enumerate(list1):        # 使用 enumerate()函数进行迭代
    print(index,item)
```

执行上述代码，结果如图 5.51 所示。

(2) 使用 iter()和 next()函数进行迭代。

案例 5.53　使用 iter()和 next()函数对列表进行迭代，示例代码如下。

```
list1 = ['NBA','CBA','Cuba','NCAA'] # 创建列表
list2 = iter(list1)                 # 使用 iter()函数生成迭代器
print(next(list2))                  # next()函数返回下一个元素
print(next(list2))
print(next(list2))
#print(next(list2))                 # 进行第 4 次输出后，列表无数据，会抛出 StopIteration 异常
```

执行上述代码，结果如图 5.52 所示。

```
8
1
2
0
--------------------------
0 a
1 b
2 c

进程已结束,退出代码0
```

图 5.51　案例 5.52 执行结果

```
NBA
CBA
Cuba

进程已结束,退出代码0
```

图 5.52　案例 5.53 执行结果

3. 元组迭代

(1) 使用 zip()函数进行迭代。

案例 5.54　使用 zip()函数对元组进行迭代,示例代码如下。

```
tuple0 = ('ID-1','ID-2','ID-3',"ID-4")              # 创建元组
tuple1 = (101,15,79,77)                             # 创建元组
# 使用 zip()函数进行压缩迭代
for name,score in zip(tuple0,tuple1):
    print(name,':',score)
```

执行上述代码,结果如图 5.53 所示。

(2) 使用 iter()和 next()函数进行迭代。

案例 5.55　使用 iter()和 next()函数对元组进行迭代,示例代码如下。

```
tuple0 = ('ID-5','ID-6','ID-7',"ID-8") # 创建元组
tuple1 = iter(tuple0)                  # 使用 iter()函数生成迭代器
print(next(tuple1))                    # 使用 next()函数进行迭代,可迭代对象下一个元素
print(next(tuple1))
print(next(tuple1))
print(next(tuple1))
# 当元组元素迭代完,还使用 next()函数获取下一个元素时,会抛出异常
```

执行上述代码,结果如图 5.54 所示。

```
ID-1 : 101
ID-2 : 15
ID-3 : 79
ID-4 : 77

进程已结束,退出代码0
```

图 5.53　案例 5.54 执行结果

```
ID-5
ID-6
ID-7
ID-8

进程已结束,退出代码0
```

图 5.54　案例 5.55 执行结果

说明:元组也可以使用 for 循环和 enumerate()函数进行迭代。

4. 字典迭代

(1) 使用 iter()和 next()函数进行迭代。

案例 5.56　使用 iter()和 next()函数对字典进行迭代,示例代码如下。

```
dict0 = {'name':'Alan','age':18}
dict1 = iter(dict0)
# 迭代器迭代字典,字典只能返回键
print(next(dict1))
# 可使用 value()函数获取字典的值
dict1 = iter((dict0).values())
print(next(dict1))
# 还可以使用 items()函数迭代获取键-值对
dict1 = iter((dict0).items())
print(next(dict1))
```

执行上述代码,结果如图 5.55 所示。

(2) 使用 for 循环进行迭代。

案例 5.57　使用 for 循环对字典进行迭代,示例代码如下。

```
dict0 = {'name':'Alan','sex':'男'}
# 使用 items()函数迭代字典,获取键-值对
for i in dict0.items():
    print(i)
# 使用 keys()函数迭代字典,获取键
for j in dict0.keys():
    print(j)
# 使用 values()函数迭代字典,获取值
for item in dict0.values():
    print(item)
```

执行上述代码,结果如图 5.56 所示。

```
name
Alan
('name', 'Alan')

进程已结束,退出代码0
```

图 5.55　案例 5.56 执行结果

```
('sex', '男')
name
sex
name
sex

进程已结束,退出代码0
```

图 5.56　案例 5.57 执行结果

5. 集合迭代

(1) 使用 iter()和 next()函数进行迭代。

案例 5.58　使用 iter()和 next()函数对集合进行迭代,示例代码如下。

```
set1 = {5,4,3,2,1}                    # 创建集合
set2 = iter(set1)                     # 使用 iter()函数生成迭代器
print(next(set2))                     # 使用 next()函数返回下一个元素
print(next(set2))
print(next(set2))
print(next(set2))
print(next(set2))
#print(next(set2))                    # 无数据则抛出 StopIteration 异常
```

执行上述代码,结果如图 5.57 所示。

图 5.57　案例 5.58 执行结果

(2) 使用 for 循环进行迭代。

案例 5.59　使用 for 循环对集合进行迭代,示例代码如下。

```
set1 = {'alan','Dawei'}
# 使用 for 循环对集合进行迭代
for i in set1:
    print(i)
```

执行上述代码,结果如图 5.58 所示。

6. 文件迭代

在 Python 中,文件类型对象属于可迭代对象,也属于迭代器。

案例 5.60　使用__next__()函数对文件进行迭代,示例代码如下。

```
file1 = open("D:/test.txt/",encoding = 'utf8')          # 使用 open()函数打开文件
print(file1.__next__())                                 # __next__()函数返回下一个元素
print(file1.__next__())
print(file1.__next__())
#print(file1.__next__())                                # 无数据则抛出 StopIteration 异常
```

执行上述代码,结果如图 5.59 所示。

图 5.58　案例 5.59 执行结果

图 5.59　案例 5.60 执行结果

说明: open()函数第 1 个参数为被读取文件的绝对路径,第 2 个参数为读取模式,以 UTF-8 形式打开文件。

5.6　课业任务

扫一扫

视频讲解

课业任务 5.1　使用列表类型随机分配办公室

【能力测试点】

列表数据类型的应用。

【任务实现步骤】

(1) 打开 PyCharm,新建一个 Python 文件,本课业任务以“任务 5.1”命名。

(2) 任务需求: 导入 random 库,使用列表创建 8 个老师对象和 3 个办公室对象,使用 random 库为老师随机分配办公室,最后进行格式化输出。代码如下。

```
import random
# 1. 准备数据
teachers = ['老师 A', '老师 B', '老师 C', '老师 D', '老师 E', '老师 F', '老师 G', '老师 H']
offices = [[ ], [ ], [ ]]
```

```
# 2. 分配老师到办公室
for name in teachers:
    # 列表追加数据
    num = random.randint(0, 2)
    offices[num].append(name)                    #把随机分配的老师添加到办公室
i = 1
# 3. 进行格式化输出
for office in offices:
    # 打印办公室人数
    print(f'办公室{i}的人数是{len(office)},老师分别是:')
    # 打印老师的名字
    for name in office:
        print(name)
    i += 1
```

运行程序,结果如图 5.60 所示。

扫一扫

视频讲解

课业任务 5.2　使用元组推导式遍历出元组中的奇数

【能力测试点】

元组数据类型的应用。

【任务实现步骤】

(1) 打开 PyCharm,新建一个 Python 文件,本课业任务以“任务 5.2”命名。

(2) 任务需求：创建数值元组,使用元组推导式遍历出除以 2 余数不为 0 的元素,使用 tuple()函数把生成器对象转换为元组形式并将结果输出。代码如下。

```
tuple1 = (100,43,323,33,67,88,48,98,12,34,64) # 直接赋值创建元组
result = (i for i in tuple1 if i % 2!= 0)          # 使用元组推导式遍历除以 2 余数不为 0 的元素
print(tuple(result))                              # 使用 tuple()函数把生成对象转换为元组形式输出
```

运行程序,结果如图 5.61 所示。

```
办公室1的人数是1，老师分别是：
老师D
办公室2的人数是3，老师分别是：
老师C
老师F
老师H
办公室3的人数是4，老师分别是：
老师A
老师B
老师E
老师G

进程已结束,退出代码0
```

图 5.60　课业任务 5.1 运行结果

```
(43, 323, 33, 67)

进程已结束,退出代码0
```

图 5.61　课业任务 5.2 运行结果

扫一扫

视频讲解

课业任务 5.3　使用字典数据类型编写用户登录程序

【能力测试点】

字典数据类型的应用。

【任务实现步骤】

(1) 打开 PyCharm,新建一个 Python 文件,本课业任务以“任务 5.3”命名。

(2) 任务需求：创建字典 dic，初始化账号和密码，使用字典的特点，如果输入的账号、密码与 dic 字典中的键-值对对应，则登录成功。代码如下。

```
dic = {'admin':'123456','administrator':'12345678','root':'password'}    # 初始化账号和密码
for i in range(0,3):                    # 输入一次账号密码，循环两次
    x = input("账号:")                  # 账号
    y = input("密码:")                  # 密码
    if x in dic.keys():                 # 账号存在(键存在)
        if dic[x] == y:                 # 密码正确(键对应的值正确)
            print('登录成功')
            break
        else:                           # 密码错误
            print('登录失败')
    else:                               # 账号不存在
        print('登录失败')
```

扫一扫

视频讲解

(3) 运行程序，用不同的输入进行测试，结果如图 5.62 所示。

```
账号：aaaa
密码：123445
登录失败
账号：admin
密码：123456
登录成功

进程已结束，退出代码0
```

图 5.62 课业任务 5.3 运行结果

课业任务 5.4 使用集合数据类型对重复元素进行判定

【能力测试点】

集合数据类型的应用。

【任务实现步骤】

(1) 打开 PyCharm，新建一个 Python 文件，本课业任务以“任务 5.4”命名。

(2) 任务需求：创建一个输入入口，然后使用集合的不可重复性筛选重复的元素，使用 set()方法去重后与原集合长度比较，若相等，则元素重复出现。代码如下。

```
instr = ''
setlen = 0
past = set()
#输入数字退出程序
while instr!= '0':
    instr = input('请输入元素:')              # 若输入 0,则退出程序
    setlen = len(past)                        # 获取添加前集合总长度
    if instr!= '0':
        past.add(instr)                       # 添加到集合中,使用集合的不可重复性
# len(past)是添加后的集合总长度,若添加前后总长度相同,则证明输入的元素已经存在于集合中
        if setlen == len(past):
            print('该元素已存在!')
            break
```

扫一扫

视频讲解

```
请输入元素：王小明
请输入元素：小红
请输入元素：王小明
该元素已存在!

进程已结束，退出代码0
```

图 5.63 课业任务 5.4 运行结果

运行程序，结果如图 5.63 所示。

课业任务 5.5 使用迭代器实现输出斐波那契数列

【能力测试点】

迭代器的应用。

【任务实现步骤】

(1) 打开 PyCharm，新建一个 Python 文件，本课业任务以“任务 5.5”命名。

（2）任务需求：创建一个 Fibs 类，在类中分别定义__init__()函数进行初始化对象、__next__()函数返回下一个元素，以及__iter__()函数返回自身，形成递归，输出数值不大于 10 的斐波那契数列。代码如下。

```
class Fibs(object):
    # 斐波那契数列迭代器
    def __init__(self):
        self.a = 0
        self.b = 1

    # 获取下一个数
    def __next__(self):
        self.a, self.b = self.b, self.a + self.b
        return self.a

    # 返回自身
    def __iter__(self):
        return self
# 调用 Fibs 类
fibs = Fibs()
a = [ ]
for f in fibs:
    # 输出不大于 10 的斐波那契数列
    if f < 10:
        a.append(f)
    else:
        break
print('斐波那契数列:',a)
```

运行程序，结果如图 5.64 所示。

```
斐波那契数列: [1, 1, 2, 3, 5, 8]

进程已结束,退出代码0
```

图 5.64　课业任务 5.5 运行结果

习题 5

1. 选择题

（1）下列选项中不是元组的特点的是(　　)。

A. 占用内存小　　B. 处理速度快

C. 具有不可变性　　D. 无序性

（2）下列选项中不是元组的基本操作的是(　　)。

A. 切片操作　　B. 修改元组元素操作

C. 成员关系操作　　D. 比较运算操作

（3）下列 4 个选项中不是字典的特点的是(　　)。

A. 可变性　　B. 任意对象的无序集合

C. 只能储存列表类型　　D. key 是不可变对象

(4) 下列关于字典操作的描述中错误的是(　　)。

A. del()函数用于删除字典或元素

B. clear()函数用于清空字典中的数据

C. len()函数可以计算字典中键-值对的个数

D. keys()函数可以获取字典的值视图

(5) 下列关于列表解析的优势正确的是(　　)。

A. 语句简洁　　B. 运行速度相对循环要快

C. 空间内存占比大　　D. 效率低

(6) 下列关于 Python 的元组类型的说法中错误的是(　　)。

A. 元组采用逗号和圆括号“()”来表示

B. 元组一旦创建就不能修改

C. 元组中的元素必须是相同类型

D. 一个元组可以作为另一个元组的元素,可以采用多级索引获取信息

(7) 下列不是组合数据类型的是(　　)。

A. 集合类型　　B. 序列类型　　C. 映射类型　　D. 引用类型

2. 简答题

(1) 列表的特点有哪些?

(2) 列表的常用基本操作有哪些?

(3) 字典类型的优点是什么?

(4) 字典与列表相比有什么不同? 什么情况下用字典比用列表好?

(5) 推导式相比循环的优势是什么?

3. 编程题

(1) 使用列表解析式创建 20 以内的偶数列表,计算列表中所有数的和。

(2) 输入 5 个整数,再将其按从大到小顺序输出。

(3) 请用户输入月份,然后返回该月份属于哪个季节。