

例 5.1 列表的元素访问与列表切片示例

```
list1=[1,"ok",2,3,"python",["a","b","c"]]
print(list1[4],list1[-2])    #结果: python python

print(list1[:2],list1[2:])    #结果: [1, 'ok'] [2, 3, 'python', ['a', 'b', 'c']]
print(list1[:-4],list1[-4:])  #结果: [1, 'ok'] [2, 3, 'python', ['a', 'b', 'c']]
print(list1[::-1])           #结果: [['a', 'b', 'c'], 'python', 3, 2, 'ok', 1]

print(list1[-1][1:])         #结果: ["b","c"]

list1[1]='book'    #改变列表元素的值。列表是可改变对象。
print(list1)#结果: [1, 'book', 2, 3, 'python', ['a', 'b', 'c']]

del list1[1]
print(list1)       #结果: [1, 2, 3, 'python', ['a', 'b', 'c']]
```

###例 5.2 列表的元素访问与列表切片示例

```
list3=[1,2,3]
list4=['a','b','c']
print(list3+list4)    #结果: [1, 2, 3, 'a', 'b', 'c']
print(list3*3)        #结果: [1, 2, 3, 1, 2, 3, 1, 2, 3]
print(3 in list3)     #结果: True
print('a' not in list4) #结果: False
```

###例 5.3 列表的函数操作举例

```
list1 = ['Google', 'Runoob']
list2 = [100,300, 200]

list1.append('Baidu')
print ("append('Baidu'):", list1)  #['Google', 'Runoob', 'Baidu']
list1.append(list2)
print ("append(list2):", list1)    #['Google', 'Runoob', 'Baidu', [100, 300, 200]]

list1.extend('Baidu')
print ("extend('Baidu'):", list1)  #['Google', 'Runoob', 'Baidu', [100, 300, 200], 'B', 'a', 'i', 'd', 'u']
list1.extend(list2)
print ("extend(list2):", list1)
#['Google', 'Runoob', 'Baidu', [100, 300, 200], 'B', 'a', 'i', 'd', 'u', 100, 300, 200]

list1.remove(list2)
print ("remove(list2):", list1)  #['Google', 'Runoob', 'Baidu', 'B', 'a', 'i', 'd', 'u', 100, 300, 200]

x=list1.pop()
print ("pop():", list1)  #['Google', 'Runoob', 'Baidu', 'B', 'a', 'i', 'd', 'u', 100, 300]
print(x)#200
```

```

print("clear",list1.clear()) #None。注意：该函数直接作用于原列表。函数返回 None
print("clear():", list1)    #[]

list1=list2.copy()
print ("copy():", list1)    #[100, 300, 200]

print("list1.reverse():",list1.reverse()) #None。注意：该函数直接作用于原列表。函数返回 None
print("list1:",list1)#[200, 300, 100]

print("list1.sort():",list1.sort())#None。注意：该函数直接作用于原列表。函数返回 None
print("list1:",list1)#[100, 200, 300]

list1.insert(1,['ab',"cd"]) #[100, ['ab', 'cd'], 200, 300]
print("insert():",list1)
print("index():",list1.index(200)) #2

```

###例 5.4 列表的函数操作异常举例（错误做法）

```

list1=[1,5,4,3,-2,6]
for x in list1.sort():
    print(x**2)

```

###例 5.4 列表的函数操作异常举例(正确做法)

```

list1=[1,5,4,3,-2,6]
list1.sort()
for x in list1:
    print(x**2,end=" ")

```

###例 5.5 元组的定义和操作

tup1 = ('Google', 'Runoob', 1997, 2017)#一般用小括号括起来
tup2 = 1, 2, 3, 4, 5 #输入或赋值时也可以不用小括号#元组输出的时候总是有小括号

```
tup3 = ("a", "b", "c", "d")
```

#创建空元组

```
tup4 = ()
```

```
tup5=tuple()
```

```
print("空元组:",tup4,tup5) #空元组: () ()
```

#元组中只包含一个元素时

```
tup6 = (50) #这是一个整数加了括号而已，不是元组
```

```
tup7=(50,) #这才是创建一个元素的元组的唯一正确方式。需要在元素后面添加逗号。
```

```
tup8=tuple([50])#tuple(50)/tuple(50,)都是错的。
```

```
print("类型：",type(tup6),type(tup7),type(tup8))#类型： <class 'int'> <class 'tuple'> <class 'tuple'>
```

###例 5.6 元组的操作、索引和切片

```
tup1 = ('Google', 'Runoob', 1997, 2017)
tup2 = 1, 2, 3, 4, 5
#与字符串和列表类似，元组有+和*操作，有成员判断操作。
print(tup1+tup2)      #('Google', 'Runoob', 1997, 2017, 1, 2, 3, 4, 5)
print(tup2*3)         #(1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5)
print(1997 in tup1)    #True
print("1997" in tup1)  #False
#与字符串和列表类似，元组的索引和切片操作原理相同
print(tup1[-2])        #1997
print(tup1[1:2])        #('Runoob',)
print(tup1[1:])         #('Runoob', 1997, 2017)
print(tup1[::-1])       #(2017, 1997, 'Runoob', 'Google')
```

###例 5.7 元组的修改和删除

```
t1 = ('Google', 'Runoob', 1997, 2017);
t2 = (1, 2, 3, 4, 5 );
```

```
#元组中的元素值是不允许修改的
#t1[0]='huawei' #error
```

#既然元组不能修改，那么其中的元素值也就不允许删除，但我们可以使用 del 语句删除整个元组

```
del t2
#print(t2) #error
```

###例 5.8 元组的函数操作

```
t=(0,1, 2, 3, 4, 5, 6,7,8,5,5)
#注意，max min len sum 函数是 python 的内建函数，直接调用。
print(max(t)) # 8
print(sum(t)) # 46
#元组的内建函数，只有 count()和 index()方法
print(t.count(5)) # 3
print(t.index(5)) # 5
#####前面讲过的列表的函数如下#####
# 1 list.append(obj) 在列表末尾添加新的对象。
# 2 list.count(obj) 统计某个元素在列表中出现的次数。
# 3 list.extend(seq) 在列表末尾一次性追加另一个序列中的多个值
# 4 list.index(obj) 从列表中找出某个值第一个匹配项的索引位置。
# 5 list.insert(index, obj) 将对象插入列表。
# 6 list.pop(obj=list[-1]) 移除列表中的一个元素（默认最后一个），返回该元素
# 7 list.remove(obj) 移除列表中某个值的第一个匹配项。
```

```
# 8 list.reverse() 将列表中的元素反向，作用于原列表。
# 9 list.sort([func]) 对原列表进行排序，作用于原列表。
# 10 list.clear() 清空列表，作用于原列表。
# 11 list.copy() 复制列表。
```

###例 5.9 字典的定义

#一个简单的字典例子如下：

```
dict0 = {'百度': '123', '华为': '456', '阿里': '789'} #直接使用大括号定义
dict1 = dict(百度=123, 华为=456, 阿里=789) #使用 dict 函数，参数为多个参数形式
dict2 = dict([("百度", 123), ("华为", 456), ("阿里", 789)]) #或者使用值成对的元组或者列表
print(dict1) # {'百度': 123, '华为': 456, '阿里': 789}
print(dict2) # {'百度': 123, '华为': 456, '阿里': 789}
#字典的键必须是唯一的，如果定义的时候出现同名的情况，则以后一个为准
dict3 = {'百度': '123', '华为': '456', '阿里': '789', '百度': 'baidu'}
print(dict3) # {'百度': 'baidu', '华为': '456', '阿里': '789'}

#字典是不可变对象，如下创建了两个字典对象
dict4 = {'abc': 456 }
print(id(dict4)) # 1925584103824
dict4 = {'abc': 123 }
print(id(dict4)) # 1925584339016。地址跟 dict4 = {'abc': 456 }不一样,不是同一个对象

#虽然字典是不可变对象，但是字典里的值是可变的
dict4['abc']=789 # 这才是改变字典的某个值
print(id(dict4)) # 1925584339016。地址与 dict4 = {'abc': 123 }一样，是同一个对象
```

例 5.10 字典的访问与修改

```
dict3 = {'百度': '123', '华为': '456', '阿里': '789', '百度': 'baidu'}
print(dict3['百度']) #baidu#通过键名访问字典
#print(dict3['naike'])#KeyError: 'naike'#不能访问一个不存在的键

dict3['Honda']=100#给不存在的键赋值，可以添加键值对
print(dict3)#{'百度': 'baidu', '华为': '456', '阿里': '789', 'Honda': 100}

dict3['百度']=123#修改字典的键值对
print(dict3)#{'百度': 123, '华为': '456', '阿里': '789', 'Honda': 100}

del dict3['Honda']#删除字典元素
print(dict3)#{'百度': 123, '华为': '456', '阿里': '789'}
del dict3 ##删除字典
```

例 5.11 字典的函数操作

#创建和浅复制

```

names=['zhang','wang','li']
values=60
d=dict.fromkeys(names,values) #d= {'zhang': 60, 'wang': 60, 'li': 60}
print('d=',d)
print('d.copy()=',d.copy())      #d.copy()= {'zhang': 60, 'wang': 60, 'li': 60}

#更新和设置
dict0 = {'fu': 80, 'yu': 90, 'zhang':95}
d.update(dict0)    #用 dict0 更新 d
print(d)           #{'zhang': 95, 'wang': 60, 'li': 60, 'fu': 80, 'yu': 90}
score=d.setdefault('li',100)#若 li 不存在则设置键值对： li-100。本例存在，仅返回 60
print(score,d['li']) #60 60
score=d.setdefault('xxx',100) #xxx'不存在则添加键值对： 'xxx'-100
print(d)    #{'zhang': 95, 'wang': 60, 'li': 60, 'fu': 80, 'yu': 90, 'xxx': 100}

#####按 key 取值#####
d={'zhang':88,'wang':99,'li':77}
print(d['zhang']) # 88----如果键不存在，报错
print(d.get('zzz')) # None----返回指定的键对应的值，如果键不存在，返回 None
print(d.get('zzz',100)) # 100---如果键不存在，返回 100。键 zzz 不会添加到集合中去
print(d)    #{'zhang': 88, 'wang': 99, 'li': 77}
#再次看一下 setdefault()方法，与 get()方法比较
print(d.setdefault('zzz',200)) #200----如果键不存在于字典中，将会添加该键值对
print(d)    #{'zhang': 88, 'wang': 99, 'li': 77, 'zzz': 200}
#同列表一样，可以通过给一个不存在的键赋值的方式，动态添加键值对
d['jr']=300
print(d)    #{'zhang': 88, 'wang': 99, 'li': 77, 'zzz': 200, 'jr': 300}

#####删除键值对#####
d={'zhang':88,'wang':99,'li':77}
print(d.pop('wang')) #99
print(d)             #{'zhang': 88, 'li': 77}
print(d.popitem())   #('li', 77)
print(d)             #{'zhang': 88}

```

###例 5.12 字典的函数操作——遍历

```

#####遍历#####
d={'zhang':88,'wang':99,'li':77}
keys=d.keys()
values=d.values()
items=d.items()
print(keys)    #dict_keys(['zhang', 'wang', 'li'])
print(values)  #dict_values([88, 99, 77])
print(items)   #dict_items([('zhang', 88), ('wang', 99), ('li', 77)])
# print(items[0])#error, 不能通过 items[0]、keys[0]、values[0]形式访问

```

```
#因为他们是惰性对象，参考 range 对象
print(*keys)#zhang wang li----解包运算
print(sum(values))#264
```

```
for key in keys:#迭代键，也可写为 for key in d:，直接遍历字典，就当做遍历其 keys
    print(key,end=',')#zhang,wang,li,
print("")
```

```
#字典转换为列表，也是直接输出 keys 的列表，相当于 list(d.keys())
print(list(d))#list(d): ['zhang', 'wang', 'li']
print(list(d.keys()))#list(d.keys()): ['zhang', 'wang', 'li']
```

```
for key,value in items:#迭代键值对
    print(f'{key}={value}',end=' ')#zhang=88 wang=99 li=77
```

###例 5.13 字典的应用——统计字符串不同字母的个数(方法 1)

```
words = input("输入字符串: ")
word_dict={}
for word in words:
    word_dict.setdefault(word,0)
    word_dict[word]+=1
print(word_dict)
```

###例 5.13 字典的应用——统计字符串不同字母的个数(方法 2)

```
words = input("输入字符串: ")
word_dict=dict.fromkeys(words,0)
print(word_dict)
for word in words:
    word_dict[word]+=1
print(word_dict)
```

###例 5.14 集合的创建

```
#####使用花括号直接创建可变集合#####
s1={1:1,2:2,3:3}#这是字典
s2={1,2,3,2}      #这是可变集合，注意和字典定义的区别
print(type(s1),s1) #<class 'dict'> {1: 1, 2: 2, 3: 3}
print(type(s2),s2) #<class 'set'> {1, 2, 3}
#####使用 set 函数创建可变集合#####
s0=set() #使用 set 函数创建空字典
s3=set([1,2,3])
s4=set((1,2,3,2))
s5=set({'zhang':90,'wang':80}) #{'zhang':90,'wang':80}是一个字典
print(s3) # {1, 2, 3}
```

```

print(s4)  #{1, 2, 3}
print(s5)  #{'wang', 'zhang'}, 用键值作为集合的元素
#####使用 frozenset 函数创建不可变集合#####
s6=frozenset([1,2,3])
s7=frozenset((1,2,3,2))
print(type(s6))#<class 'frozenset'>
print(s7)      #frozenset({1, 2, 3})

```

###例 5.15 创建一个包含元素 0-9 的平方的列表

```

s1=[]
for i in range(10):
    s1.append(i*i)
print(s1)#[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
s2=[i*i for i in range(10)]
print(s2)#[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

```

结果:

```

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

```

###例 5.16 列表推导式实例

#快速创建一个包含 1-10 之间所有偶数的列表

```

list0 = [i for i in range(1, 11) if i % 2 == 0]
print(list0)#[2, 4, 6, 8, 10]

```

```

list1 = [2.3,3.4,4.5,5.6,6.7]
print ([int(x) for x in list1]) #list 中所有数转换为整数（割尾巴取整）:[2, 3, 4, 5, 6]
print ([int(x)**2 for x in list1]) #打印 list 中所有数的平方:[4, 9, 16, 25, 36]
print ([int(x)**2 for x in list1 if int(x) % 2 == 0]) #只打印偶数的平方:[4, 16, 36]

```

```

str1='abcde'
tuple1=(1,3,5)
print([x*2 for x in str1])      #['aa', 'bb', 'cc', 'dd', 'ee']
print([x**2 for x in tuple1])   #[1, 9, 25]

```

```

l1 = ['太白金星', 'fdsaf', 'alex', 'sb', 'ab']
l2 = [i.upper() for i in l1 if len(i) > 3]
print(l2)  #['太白金星', 'FDSAF', 'ALEX']

```

```

print([f'python{i}期' for i in range(1, 5)])
#['python1 期', 'python2 期', 'python3 期', 'python4 期']

```

#利用列表推导式将列表中的整数提取出来

```

values =[True, 33, "ll", "kk", 44, 34, -21]

```

```
nums=[i for i in values if type(i) == int]    #type()函数返回数据类型
print(nums)    #[33, 44, 34, -21]
```

###例 5.17 列表推导式效率对比

```
def f1():
    s=[]
    for i in range(1000000):
        s.append(i*i)
def f2():
    s=[i*i for i in range(1000000)]
%time f1() #Wall time: 223 ms
%time f2() #Wall time: 136 ms
```

1.元组推导式：（表达式 for 迭代变量 in 可迭代对象 [if 条件表达式]）

```
#元组的推导式生成的是一个生成器对象(只能“用”一次)
a = [x for x in range(1,10)]    #这是列表推导式
print(a)#[1, 2, 3, 4, 5, 6, 7, 8, 9]
b = (x for x in range(1,10))    #这是元组推导式
print(b)#<generator object <genexpr> at 0x000001C03C0160C0>

for i in b:    #遍历元组推导式
    print(i**2,end=' ') # 1 4 9 16 25 36 49 64 81
for i in b:    #再次遍历元组推导式
    print(i**2,end=' ') #输出为空

b = (x for x in range(1,10))
c=tuple(b)
print(c) #(1, 2, 3, 4, 5, 6, 7, 8, 9)
print(tuple(b))#()
```

2.集合推导式：{表达式 for 迭代变量 in 可迭代对象 [if 条件表达式]}

```
a = [x**2 for x in range(-5,6)]#这是列表推导式
print(a)    #[25, 16, 9, 4, 1, 0, 1, 4, 9, 16, 25]
b = {x**2 for x in range(-5,6)}#这是集合推导式
print(b)    #{0, 1, 4, 9, 16, 25}
for i in b:
    print(i,end=' ') # 0, 1, 4, 9, 16, 25
print("\n 遍历集合推导式结束！ ")
c=tuple(b)
print(c) #(0, 1, 4, 9, 16, 25)
```

3.字典推导式：{表达式 for 迭代变量 in 可迭代对象 [if 条件表达式]} }


```
# 使用字典推导式生成字典
cookies = "id=jy0ui55o-u6f6zd; name=tom; version=1"
cookies = {cookie.split("=")[0]:cookie.split("=")[1] for cookie in cookies.split("; ")}
print(cookies)#{'id': 'jy0ui55o-u6f6zd', 'name': 'tom', 'version': '1'}

# 使用字典推导式交换字典中键值对位置
changed = {value:key for key,value in cookies.items()}
print(changed)#{'jy0ui55o-u6f6zd': 'id', 'tom': 'name', '1': 'version'}

# 使用字典推导式获取字典中 key 值是小写字母的键值对
dict1 = {"a":10,"B":20,"C":True,"D":"hello world","e":"python 教程"}
dict2 = {key:value for key,value in dict1.items() if key.islower()}
print(dict2)#{'a': 10, 'e': 'python 教程'}

# 使用字典推导式将字典中的所有 key 设置为小写
dict3 = {key.lower():value for key,value in dict1.items()}
print(dict3)#{'a': 10, 'b': 20, 'c': True, 'd': 'hello world', 'e': 'python 教程'}
```