**17Python\_05列表和元组+字典+集合**

**一 列表和元组-列表**

**1.1 创建及访问列表**

**• 列表是有序、可变的数据类型**

**• 列表中可以包含不同类型的对象**

**• 列表可以由[]或工厂函数list()创建**

**• 支持下标及切片操作**

type(listname[下标x]) #查看list中下标为x的元素的数据类型

**1.2 更新列表**

• 通过下标只能更新值,不能使用下标添加新值

>>> alist = [10, 8, 20, 365, 23, 4]

>>> alist[0]=100 #更新下标为0的元素的值为100

>>> alist

[100, 8, 20, 365, 23, 4]

>>> alist[1:3] #切片,取出下标为1到3(不含)的字列表

[8, 20]

>>> alist[1:3] = [99, 98 ,97, 96] #替换下标为1到3(不含)为后续的值

>>> alist

[100, 99, 98, 97, 96, 365, 23, 4]

**1.3 列表內建函数**

**list.append(obj)** 将obj作为一个元素追加到列表末尾

**list.count(obj)** 返回一个对象obj在列表中出现的次数

**list.extend(seq)** 把可迭代序列seq中的每项作为元素追加到列表末尾

**list.index(obj)** 返回obj对象的下标

**list.insert(index,obj)** 在索引量为index的位置插入对象obj

**list.reverse()** 原地翻转列表

**list.sort()** 列表排序

>>> alist.append(1000) #向列表末尾追加元素1000

>>> alist

[100, 99, 98, 97, 96, 365, 23, 4, 1000]

>>> alist.append((1, 2)) #将序列对象作为1个元素追加到列表末尾

>>> alist

[100, 99, 98, 97, 96, 365, 23, 4, 1000, (1, 2)]

>>> alist.extend((3,4)) #将序列对象中的每一项作为一个元素追加到列表

>>> alist

[100, 99, 98, 97, 96, 365, 23, 4, 1000, (1, 2), 3, 4]

>>> alist.remove((1,2)) #删除列表中的元素

>>> alist

[100, 99, 98, 97, 96, 365, 23, 4, 1000, 3, 4]

>>> alist.index(100) #返回元素100的下标

0

>>> alist.reverse() #翻转列表

>>> alist

[4, 3, 1000, 4, 23, 365, 96, 97, 98, 99, 100]

>>> alist.insert(2, 9) #在下标为2的位置插入元素9

>>> alist

[4, 3, 9, 1000, 4, 23, 365, 96, 97, 98, 99, 100]

>>> alist.sort() #升序排序

>>> alist

[3, 4, 4, 9, 23, 96, 97, 98, 99, 100, 365, 1000]

>>> alist.sort(reverse=True) #降序排序

>>> alist

[1000, 365, 100, 99, 98, 97, 96, 23, 9, 4, 4, 3]

>>> alist.count(4) #统计元素4在列表中出现的次数

2

>>> alist.pop() #弹出下标为-1(缺省或默认)的元素,有返回值

3

>>> alist

[1000, 365, 100, 99, 98, 97, 96, 23, 9, 4, 4]

>>> alist.pop(2) #弹出下标为2的元素,有返回值,而remove无返回值

100

>>> alist

[1000, 365, 99, 98, 97, 96, 23, 9, 4, 4]

>>> alist.index(9,0,-1) #从下标为0开始,到下标-1结束,取出元素9的下标

7

>>> blist = alist.copy() #将alist的值复制出来,赋值给blist

>>> blist

[1000, 365, 99, 98, 97, 96, 23, 9, 4, 4]

>>> blist.clear() #清空blist

>>> blist

[]

**二 列表和元组-元组**

**2.1 创建元组**

**• 通过()或工厂函数tuple()创建元组**

**• 元组是有序的、不可变类型**

**• 与列表类似,作用于列表的操作,绝大数也可以作用于元组**

**元组是静态的列表,除元组内元素不能修改外,其他操作同列表**

**2.2 单元素元组**

**• 如果一个元组中只有一个元素,那么创建该元组的时候,需要加上一个逗号**

atu = (10, 3, 20)

atu.count(100)

0

atu.index(3)

1

>>> btu = ('hello') #无逗号,单元素定义的不是元组

>>> btu

'hello'

>>> type(btu)

<class 'str'>

>>> ctu = ('hello',) #有逗号,单元素定义的才是元组

>>> ctu

('hello',)

>>> type(ctu)

<class 'tuple'>

**案例1:用列表构建栈结构**

1. 栈是一个后进先出的结构; 2. 编写一个程序,用列表实现栈结构; 3. 需要支持压栈、出栈、查询功能

stack.py

stack = []

def push\_it():

# 读取用户输入，非空内容追加到列表，否则打印提示

data = input('数据: ').strip()

if data: # 如果data非空

stack.append(data)

else:

print('输入内容为空。')

def pop\_it():

if stack:

print('从栈中，弹出: %s' % stack.pop())

else:

print('空栈')

def view\_it():

print(stack)

def show\_menu():

cmds = {'0': push\_it, '1': pop\_it, '2': view\_it}

prompt = """(0) 压栈

(1) 出栈

(2) 查询

(3) 退出

请选择(0/1/2/3): """

while 1:

# 将用户输入字符去除两端空白字符后，赋值给choice

choice = input(prompt).strip()

if choice not in ['0', '1', '2', '3']:

print('无效的输入，请重试。')

continue

if choice == '3':

print('\nBye-bye')

break

cmds[choice]()

if \_\_name\_\_ == '\_\_main\_\_':

show\_menu()

**三 字典-字典基本操作**

**容器 可变 映射**

**3.1 创建字典**

• 通过{}操作符创建字典

• 通过dict()工厂方法创建字典

• 通过fromkeys()创建具有相同值的默认字典

字典的key不能重复,key必须是不可变类型

adict = dict(['ab',('name','bob'),['age',20]])

adict

{'a': 'b', 'name': 'bob', 'age': 20}

bdict = {}.fromkeys(['bob','tom','ailce'],7) #创建具有相同值的字典

bdict

{'bob': 7, 'tom': 7, 'ailce': 7}

**3.2 访问字典**

**• 字典是映射类型,意味着它没有下标,访问字典中的值需要使用相应的键**

>>> for key in adict:

... print(key,adict[key])

...

a b

name bob

age 20

>>> '%(name)s is %(age)d years old' % adict #s表示字符串,d表数字

'bob is 20 years old'

**3.3 更新字典**

• 通过键更新字典

– 如果字典中有该键,则更新相关值

– 如果字典中没有该键,则向字典中添加新值

>>> adict['age'] = 22 #更新值

>>> adict ['email'] = ['bob@tedu.cn'](mailto:'bob@tedu.cn') #添加新键值

>>> adict

{'a': 'b', 'name': 'bob', 'age': 22, 'email': 'bob@tedu.cn'}

>>> 'bob' in adict bob是字典adict的key吗?

False

>>> 'name' in adict name是字典adict的key吗?

True

>>> len(adict)

4

**3.4 删除字典**

• 通过del可以删除字典中的元素或整个字典

>>>　adict

{'name': 'bob', 'age': 22, 'email': 'bob@tedu.cn', 'qq': '123456'}

>>> del adict['name']

>>> adict

{'age': 22, 'email': 'bob@tedu.cn', 'qq': '123456'}

>>> blist

[]

>>> del blist

>>> blist #报错：name 'blist' is not defined

• 使用内部方法clear()可以清空字典,清空后字典为空

>>> alist

[1000, 365, 99, 98, 97, 96, 23, 9, 4, 4]

>>> alist.clear()

>>> alist

[]

• 使用pop()方法可以“弹出”字典中的元素

>>> list(adict.items()) #列出字典的key:value对

[('a', 'b'), ('name', 'bob'), ('age', 22), ('email', 'bob@tedu.cn')]

>>> adict.pop('a') #弹出key为a的key:value对,返回key对应的值

'b'

**3.5 字典操作符**

• 使用字典键查找操作符[],查找键所对应的值

• 使用in和not in判断键是否存在于字典中

**字典最常用的方式是get 根据key取value**

>>> adict.get('age')

22

>>> adict.get('qq')

>>> print(adict.get('qq')) #字典中无qq这个key,默认返回None

None

adict.get('qq','notfound') #字典中无qq这个key,返回设定值notfound

'notfound'

adict.get('name','notfound') #字典中有name这个key,返回keyvalue

'bob'

>>> list(adict.keys()) #列出字典里所有的key

['a', 'name', 'age', 'email']

>>> list(adict.values()) #列出字典里所有的value

['b', 'bob', 22, 'bob@tedu.cn']

list(adict.items()) #列出字典的key:value对

[('a', 'b'), ('name', 'bob'), ('age', 22), ('email', 'bob@tedu.cn')]

>>> adict.pop('a') #弹出key为a的key:value对

'b'

>>> adict.update({'qq':'123456'}) #将数据追加到字典末尾

>>> adict

{'name': 'bob', 'age': 22, 'email': 'bob@tedu.cn', 'qq': '123456'}

**案例2:模拟用户登陆信息系统**

1. 支持新用户注册,新用户名和密码注册到字典中

2. 支持老用户登陆,用户名和密码正确提示登陆成功

3. 主程序通过循环询问进行何种操作,根据用户的选择,执行注册或是登陆操作

import getpass

userdb = {}

def register():

uname = input('username:').strip()

upass = input('password:')

# 如果用户名非空,并且不是字典的key,将用户名:密码键值对写入字典

if uname and (uname not in userdb):

userdb[uname] = upass

print('注册成功')

else:

print('用户名为空或已存在')

def login():

uname = input('username:').strip()

upass = getpass.getpass('password:')

# if (uname in userdb) and (userdb[uname] == upass):

if userdb.get(uname) == upass:

print('登录成功')

else:

print('登录失败')

def show\_menu():

cmds = {'0': register, '1': login}

prompt = '''(0) 注册

(1) 登录

(2) 退出

请选择(0/1/2)'''

while 1:

choice = input(prompt).strip()

if choice not in ['0', '1', '2']:

print('无效的输入,请重新输入.')

continue

if choice == '2':

print('\nBye')

break

cmds[choice]()

if \_\_name\_\_ == '\_\_main\_\_':

show\_menu()

**四 字典-字典相关函数**

**4.1 作用于字典的函数**

• len():返回字典中元素的数目

• hash():本身不是为字典设计的,但是可以判断某个对象是否可以作为字典的键

**4.2 字典內建方法**

• dict.copy():返回字典(深复制)的一个副本

• dict.get(key, default=None):对字典dict中的键key,返回它对应的值value,如果字典中不存在此键,则返回default的值

• dict.setdefault(key, default=None):如果字典中不存在key键,由dict[key]=default为它赋值

• dict.items():返回一个包含字典中(键,值)对元组的列表

• dict.keys():返回一个包含字典中键的列表

• dict.values():返回一个包含字典中所有值的列表

• dict.update(dict2):将字典dict2的键-值对添加到字典dict

**五 集合-集合基础**

**5.1 创建集合**

• 集合是由**不同的元素**组成的集合,集合(set)的成员通常被称作集合元素

• 集合的元素必须是**不可变类型**

• 集合**没有顺序**

• 集合就像是一个**无值的字典**

• 集合有：可变集合set；不可变集合frozenset

**5.2 集合类型操作符**

• 集合支持用in和not in操作符检查成员

• 能够通过len()检查集合大小

• 能够使用for迭代集合成员

• 不能取切片,没有键

>>> s1 = set ('abc')

>>> s2 =set ('bcd')

>>> s3 = set(['tom','jerry','bob','alice'])

>>> s1

{'b', 'c', 'a'}

>>> s2

{'b', 'c', 'd'}

>>> s3

{'tom', 'jerry', 'alice', 'bob'}

>>> set('hello')

{'e', 'o', 'h', 'l'} #只有１个l，集合中的元素不能重复

>>> len(s3)

4

>>> for name in s3: #集合遍历

... print(name)

...

tom

jerry

alice

bob

>>> 'tom' in s3 #成员关系判断

True

>>> 'zs' in s3

False

>>> 'zs' not in s3

True

• |:并集

• &:交集

• -:差补

>>> s1

{'b', 'c', 'a'}

>>> s2

{'b', 'c', 'd'}

>>> s1 & s2 #交集，两个集合中都包含的元素

{'b', 'c'}

>>> s1 | s2 #并集,两个集合中全部的 不重复的元素

{'b', 'c', 'a', 'd'}

>>> s1 - s2 #差集,s1中有,s2中没有

{'a'}

>>> s2 -s1 #差集,s2中有,s1中没有

{'d'}

**六 集合-集合方法**

**6.1 集合的方法**

• set.add():添加元素

• set.update():批量添加元素

• set.remove():移除元素

>>> s3 = set()

>>> s3

set()

>>> s3.add(10)

>>> s3

{10}

>>> s3.add([20,30]) #报错: add是将小括号内的数据当作一个元素添加到集合中,但是[20,30]是列表是可变的,集合要求元素不可变,故报错

>>> s3.update([20,30]) #update是分别将列表中的每个元素添加到集合中

>>> s3

{10, 20, 30}

>>> s3.remove(30) #删除成员30

>>> s3

{10, 20}

>>> s4 = set((10,20,30,40))

s4

{40, 10, 20, 30}

>>> s3.issubset(s4) #s3包含在s4里面吗?(s3是s4的子集吗)

True

>>> s4.issuperset(s3) #s4包含s3吗?(s4是s3的超集吗)

True

>>> s1.intersection(s2) #相当于 s1 & s2

{'b', 'c'}

>>> s1.union(s2) #相当于s1 | s2

{'b', 'c', 'a', 'd'}

>>> s1.difference(s2) #相当于s1 - s2

{'a'}

**七 集合的应用**

**7.1 去重**

>>> from random import randint

>>> nums = [randint(1,20) for i in range(20)] #生成随机数列表

>>> nums

[7, 6, 18, 7, 10, 6, 1, 14, 4, 9, 16, 18, 9, 7, 18, 19, 4, 20, 19, 4]

>>> set(nums) #生成集合,相当于对列表去重;去重后为集合,需再转为列表

{1, 4, 6, 7, 9, 10, 14, 16, 18, 19, 20}

>>> list(set(nums))

[1, 4, 6, 7, 9, 10, 14, 16, 18, 19, 20]

**案例3:比较文件内容**

1. 有两个文件:a.log和b.log

2. 两个文件中有大量重复内容

3. 取出只有在b.log中存在的行

(nsd1906) [root@room9pc01 day05]# cp /etc/passwd /tmp/mima1

(nsd1906) [root@room9pc01 day05]# cp /etc/passwd /tmp/mima2

(nsd1906) [root@room9pc01 day05]# vim /tmp/mima1

#修改mima1内容,使和mima2不同

>>> with open('/tmp/mima1') as f1:

... s1 = set(f1)

...

>>> with open('/tmp/mima2') as f2:

... s2 = set(f2)

...

>>> s2 -s1

{'root:x:0:0:root:/root:/bin/bash\n'}

>>> with open ('/tmp/mima3','w') as f3:

... f3.writelines(s2-s1)

...

(nsd1906) [root@room9pc01 day05]# cat /tmp/mima3

root:x:0:0:root:/root:/bin/bash