nsd_1908_py01_day05

列表

- 属于可变、容器、顺序类型

```
>>> alist = [1, 8, 54, 10, 3, 1]
>>> alist[2]=80
>>> alist[2:4] = [1, 3, 5, 10]
>>> alist
[1, 8, 1, 3, 5, 10, 3, 1]
>>> alist[3:3] = [10, 20]
>>> alist
[1, 8, 1, 10, 20, 3, 5, 10, 3, 1]

# 列表的方法
>>> alist.append(10)   # 追加
>>> alist.insert(1, 10)  # 在下标为1的位置插入数字10
>>> alist.count(10)  # 统计列表中10的个数
>>> alist.extend([100, 200, 300])  # 向列表中加入3项
>>> alist.pop()  # 移除并返回列表的最后一项
300
>>> alist.pop(2)  # 移除并返回下标为2的项
8
>>> alist.remove(10)  # 删除列表中出现的第一个10
>>> alist.index(5)  # 返回5的下标
4
>>> alist.sort()  # 升序排列
>>> alist.reverse()  # 翻转列表
>>> clist = alist.copy()  # 将alist的值赋值给clist，它们使用不同的地址空间
>>> clist.clear()  # 清空clist
```

元组

- 属于容器、不可变、顺序类型
- 元组相当于是不可变的列表

```
>>> atuple = (10, 20, 30, 20, 40, 20, 8)
>>> atuple.count(20)  # 统计20出现的次数
3
>>> atuple.index(30)  # 返回30的下标
2
```

- 单元素元组，必须在元素后面加逗号

```
>>> a = (10)
>>> type(a)
<class 'int'>
>>> a
10
>>> a = (10,)
>>> type(a)
<class 'tuple'>
>>> len(a)
1
```

列表练习

1. 发呆。思考程序的运行方式

```
(0) 压栈
(1) 出栈
(2) 查询
(3) 退出
请选择(0/1/2/3): 10
无效的选择，请重试。
(0) 压栈
(1) 出栈
(2) 查询
(3) 退出
请选择(0/1/2/3): 2
[]
(0) 压栈
(1) 出栈
(2) 查询
(3) 退出
请选择(0/1/2/3): 0
数据:
输入为空
(0) 压栈
(1) 出栈
(2) 查询
(3) 退出
请选择(0/1/2/3): 0
数据: hello
(0) 压栈
(1) 出栈
(2) 查询
(3) 退出
请选择(0/1/2/3): 0
数据: world
(0) 压栈
(1) 出栈
(2) 查询
(3) 退出
请选择(0/1/2/3): 2
```

```
['hello', 'world']
(0) 压栈
(1) 出栈
(2) 查询
(3) 退出
请选择(0/1/2/3): 1
从栈中，弹出了: hello
(0) 压栈
(1) 出栈
(2) 查询
(3) 退出
请选择(0/1/2/3): 1
从栈中，弹出了: world
(0) 压栈
(1) 出栈
(2) 查询
(3) 退出
请选择(0/1/2/3): 1
栈已经为空
(0) 压栈
(1) 出栈
(2) 查询
(3) 退出
请选择(0/1/2/3): 2
[]
(0) 压栈
(1) 出栈
(2) 查询
(3) 退出
请选择(0/1/2/3): 3
Bye-bye
```

2. 思考程序有哪些功能，将功能写成功能函数。
3. 编写主程序代码，按照一定的规则调用函数。

```
def push_it():
    '用于实现压栈功能'

def pop_it():
    '用于实现出栈功能'

def view_it():
    '用于实现查询功能'

def show_menu():
    '用于在屏幕上打印菜单，根据用户选择，调用相关功能函数'


if __name__ == '__main__':
    show_menu()
```

4. 完成每个具体的函数

字典

- 属于容器、可变、映射
- 字典的key不能重复。赋值时，key不存在则向字典加新值；key存在则修改值
- 字典的key只能是不可变对象

```
>>> adict = dict(['ab', ('name', 'tom'), ['age', 20]])
>>> adict
{'a': 'b', 'name': 'tom', 'age': 20}

>>> dict([('name', 'tom'), ('age', 20), ('email', 'tom@tedu.cn')])
{'name': 'tom', 'age': 20, 'email': 'tom@tedu.cn'}

>>> {}.fromkeys(['tom', 'bob', 'jerry'], 20)
{'tom': 20, 'bob': 20, 'jerry': 20}

>>> 'tom' in adict  # tom是字典的key吗？
False
>>> 'name' in adict
True

>>> for key in adict:
...   print(key, adict[key])
>>> adict['email'] = 'tom@tedu.cn'
>>> adict
{'a': 'b', 'name': 'tom', 'age': 20, 'email': 'tom@tedu.cn'}
>>> adict['age'] = 22
>>> adict
{'a': 'b', 'name': 'tom', 'age': 22, 'email': 'tom@tedu.cn'}

# 字典方法
>>> adict.get('name')  # 最常用的方法，通过key取出value
'tom'
>>> print(adict.get('phone'))  # 如果key不在字典中，默认返回None
None
>>> adict.get('phone', 'not found')  # key不在字典中，返回值可以自行指定
'not found'
>>> adict.get('name', 'not found')
'tom'

# adict.clear()和adict.copy()与列表的相关方法一样
>>> adict.keys()  # 取出所有的key
dict_keys(['a', 'name', 'age', 'email'])
>>> list(adict.keys())
['a', 'name', 'age', 'email']
>>> adict.values()  # 取出所有的value
dict_values(['b', 'tom', 22, 'tom@tedu.cn'])
>>> adict.items()  # 将字典每一项以(key, val)形式返回
dict_items([('a', 'b'), ('name', 'tom'), ('age', 22), ('email', 'tom@tedu.cn')])
>>> list(adict.items())
```

```
[('a', 'b'), ('name', 'tom'), ('age', 22), ('email', 'tom@tedu.cn')]
>>> adict.pop('a')  # 通过key弹出value
'b'
>>> adict
{'name': 'tom', 'age': 22, 'email': 'tom@tedu.cn'}
```

哈希算法

- 单向加密。只能通过原始数据生成密文，不能反推。
- 原始数据相同，得到的密文必定也是一样的。

```
# tom和jerry的密码都是123456，但是存储的时候，密文不相同。
[root@node4 ~]# grep -e 'tom\|jerry' /etc/shadow
tom:$6$no84OMPb$2.n/F.9lXFiNgfb1d7xfZSDjlQeZpg5GKdZSD1uPWj8aBo./BMC67JAkzAv3kUJmFB7VcpjqD0oF8ElB5O7GT.:18268:0:99999:7:::
jerry:$6$ijip1nNO$Yp9hISNpGRyfQQ9QXNv.nRWzRlPLTXQtcWNF1lxd/3aFE/igBHhy0i6aftdgoY1K5adK4mJEss4fsN3fFNnfj1:18268:0:99999:7:::

# 密文说明
# $算法$salt$通过密码和salt算出来的密文
# 算法如果是md5为1，如果是sha512为6
# salt是随机字符串
>>> import crypt
>>> crypt.crypt('123456', '$6$no84OMPb$')
'$6$no84OMPb$2.n/F.9lXFiNgfb1d7xfZSDjlQeZpg5GKdZSD1uPWj8aBo./BMC67JAkzAv3kUJmFB7VcpjqD0oF8ElB5O7GT.'
>>> crypt.crypt('123456', '$6$ijip1nNO$')
'$6$ijip1nNO$Yp9hISNpGRyfQQ9QXNv.nRWzRlPLTXQtcWNF1lxd/3aFE/igBHhy0i6aftdgoY1K5adK4mJEss4fsN3fFNnfj1'
```

集合

- 数学上，把set称做由不同的元素组成的集合
- 集合元素必须是不可变对象
- 集合元素没有顺序
- 集合元素不能重复
- 集合就像是一个无值的字典

```
>>> aset = set('abc')
>>> bset = set('bcd')
>>> cset = set(['tom', 'jerry', 'bob'])
>>> aset
{'c', 'b', 'a'}
>>> bset
{'c', 'd', 'b'}
>>> cset
{'tom', 'bob', 'jerry'}
>>> for name in cset:
...     print(name)
>>> len(cset)
```

```
3
>>> 'tom' in cset
True
>>> set('hello')
{'l', 'e', 'h', 'o'}


>>> aset & bset  # 交集
{'c', 'b'}
>>> aset | bset  # 并集
{'b', 'a', 'c', 'd'}
>>> aset - bset  # 差补，aset中有，bset中无
{'a'}
>>> bset - aset
{'d'}

>>> cset.add('alice')  # 增加元素
>>> cset
{'tom', 'bob', 'jerry', 'alice'}
>>> cset.remove('bob')  # 删除元素
>>> cset
{'tom', 'jerry', 'alice'}
>>> cset.update(['zhangsan', 'lisi'])  # 批量更新
>>> cset
{'lisi', 'jerry', 'zhangsan', 'tom', 'alice'}
>>> dset = aset | bset
>>> dset
{'b', 'a', 'c', 'd'}
>>> aset.issubset(dset)  # aset是dset的子集吗？
True
>>> dset.issuperset(aset)  # dset是aset的超集吗？
True

>>> aset.union(bset)  # aset | bset
{'b', 'a', 'c', 'd'}
>>> aset.intersection(bset)  # aset & bset
{'c', 'b'}
>>> aset.difference(bset)  # aset - bset
{'a'}

# 集合实现数据去重
>>> from random import randint
>>> nums = [randint(1, 20) for i in range(30)]
>>> nums
[2, 12, 14, 11, 14, 14, 11, 4, 4, 9, 15, 1, 10, 11, 16, 16, 4, 18, 14, 6, 19, 17, 20, 9,
17, 10, 14, 7, 6, 9]
>>> set(nums)
{1, 2, 4, 6, 7, 9, 10, 11, 12, 14, 15, 16, 17, 18, 19, 20}



# 比较两个文件，取出b文件中存在，a中不存在的行
(nsd1908) [root@room8pc16 day05]# cp /etc/passwd /tmp/
```

```
(nsd1908) [root@room8pc16 day05]# cp /etc/passwd /tmp/mima
# 修改mima文件，使其产生重复行，再加几行新内容
(nsd1908) [root@room8pc16 day05]# vim /tmp/mima
>>> with open('/tmp/passwd') as f1:
...     set1 = set(f1)  # 文件中的每一行存到集合中
>>> with open('/tmp/mima') as f2:
...     set2 = set(f2)
>>> set2 - set1  # 取差补
{'hello world!\n', 'how are you?\n'}
>>> with open('/tmp/r.txt', 'w') as f3:
...     f3.writelines(set2 - set1)
```

1.log -> 昨天的日志文件 -> url1.txt

2.log -> 今天的日志文件 -> url2.txt

100 0000 x 100 0000 => 1万亿