day05

变量

全局变量

在函数外面定义的变量，从它定义的开始位置，一直到程序结束，都可见可用。

一般来说，变量的值自始至终都不需要变化，可以设置为全局变量。

局部变量

在函数内部定义的变量，只能在本函数内部使用。

生成随机字符串

```
>>> import string
>>> import random
>>> all_chs = string.ascii_letters + string.digits
>>> result = [random.choice(all_chs) for i in range(8)]
>>> result
['V', '1', '1', 'R', '1', '7', 'm', '4']
>>> ''.join(result)
'V11R17m4'
>>> '-'.join(result)
'V-1-1-R-1-7-m-4'
>>> '##'.join(result)
'V##1##1##R##1##7##m##4'
```

原始字符串

作用：取消转义行为

```
>>> win_path = 'c:\temp\newdir'
>>> print(win_path)  # \t成为tab，\n成为换行
c:  emp
ewdir
>>> wpath = r'c:\temp\newdir'
>>> print(wpath)
c:\temp\newdir
>>> wpath
'c:\\temp\\newdir'
```

字符串方法

```
>>> s1 = ' \tHello World!  '
>>> s1.strip()  # 删除两端空白字符
'Hello World!'
>>> s1.lstrip()
'Hello World!  '
>>> s1.rstrip()
' \tHello World!'
>>> s2 = 'hao123'
>>> s2.center(30)
'            hao123            '
>>> s2.center(30, '*')
'************hao123************'
>>> s2.ljust(30, '#')
'hao123########################'
>>> s2.rjust(30, '#')
'########################hao123'
>>> s2.replace('h', 'H')  # 替换
'Hao123'
>>> s2.upper()
'HAO123'
>>> 'HAO123'.lower()
'hao123'
>>> s2.islower()  # 字母都是小写的吗
True
>>> s2.isdigit()  # 所有字符都是数字吗？
False
>>> s2.startswith('ab')   # 是以ab开头吗？
False
>>> s2.endswith('123')  # 是以123结尾吗？
True
>>> s1.count('l')  # 统计l出现的次数
3
>>> s1.index('l')  # 第一个l的下标
4
```

列表

容器、可变、顺序

```
>>> alist = [10, 80, 20, 60]
>>> alist.append(50)  # 追加
>>> alist.extend([15, 100])  # 将序列对象扩展到alist中
>>> alist
[10, 80, 20, 60, 50, 15, 100]
>>> alist.remove(20)  # 删除元素
>>> alist.index(60)  # 取出60的下标
2
>>> alist.reverse()  # 反转
>>> alist.insert(2, 60)  # 将60插入到下标为2的位置
>>> alist
[100, 15, 60, 50, 60, 80, 10]
>>> alist.sort()  # 升序排列
>>> alist.sort(reverse=True)  # 降序
>>> alist.count(60)  # 统计60出现的次数
2
>>> alist.pop()  # 默认弹出最后一项
10
>>> alist.pop(2)  # 弹出下标为2的项目
60
>>> alist.clear()  # 清空列表
```

元组

容器、不可变、顺序

```
>>> atuple = (100, 80, 60, 50, 15)
>>> atuple.count(30)
0
>>> atuple.index(50)
3
# 单元素元组，必须在结尾加逗号，否则不是元组
>>> a = (10)
>>> type(a)
<class 'int'>
>>> a
10
>>> a = (10,)
>>> type(a)
<class 'tuple'>
>>> a
(10,)
>>> len(a)
1
```

方法的返回值

列表等对象，它们的方法就是一个函数。函数有可能有返回值，也可能没有，没有返回值，默认返回None。

```
>>> alist = [100, 80, 60, 50, 15, 50]
>>> a = alist.remove(50)
>>> alist
[100, 80, 60, 15, 50]
>>> print(a)
None
>>> b = alist.pop()  # pop有返回值
>>> alist
[100, 80, 60, 15]
>>> b
50
```

> *注意：变量没有print()语句，在交互解释器中可以显示它的值，但是作为程序文件运行，没有print()是不会打印值的*

用列表模拟栈

1. 首先，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): 2
[]
(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): 2
['hello']
(0) 压栈
(1) 出栈
(2) 查询
(3) 退出
请选择(0/1/2/3): 1
```

```
从列表中弹出了: hello
(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): 1
空列表
(0) 压栈
(1) 出栈
(2) 查询
(3) 退出
请选择(0/1/2/3): 3
Bye-bye
```

2. 分析需要多个功能，将功能写成功能函数

```
def push_it():


def pop_it():


def view_it():


def show_menu():


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

3. 编写主程序代码
4. 填写函数代码

字典

容器、可变、映射

创建字典

```
>>> adict = {'name': 'bob', 'age': 25}
>>> dict(['ab', ('name', 'alice'), ['age', '20']])
{'a': 'b', 'name': 'alice', 'age': '20'}
>>> {}.fromkeys(['bob', 'tom', 'jerry'], 'male')
{'bob': 'male', 'tom': 'male', 'jerry': 'male'}
```

访问字典

```
>>> adict
{'name': 'bob', 'age': 25}
>>> 'bob' in adict   # bob是字典的key吗？
False
>>> 'name' in adict
True
>>> for key in adict:
...     print('%s: %s' % (key, adict[key]))
...
name: bob
age: 25
>>> '%(name)s is %(age)s years old.' % adict
'bob is 25 years old.'
```

更新字典

字典的key不能重复。在进行赋值的时候，如果key不存在，则向字典加入新值；如key已存在，则更新。

```
>>> adict
{'name': 'bob', 'age': 25}
>>> adict['email'] = 'bob@tedu.cn'
>>> adict
{'name': 'bob', 'age': 25, 'email': 'bob@tedu.cn'}
>>> adict['age'] = 23
>>> adict
{'name': 'bob', 'age': 23, 'email': 'bob@tedu.cn'}
```

del方法

> *del可以删除可种对象，如删除变量，列表中的某一项、字典中的某一项*

```
>>> del adict['email']
>>> adict
{'name': 'bob', 'age': 23}
```

字典的key

- 字典的key必须是不可变类型：数字、字符串、元组
- 通过hash函数判断是否可变

```
>>> hash(10)
10
>>> hash('abc')
-4683219747325187051
>>> hash((1, 2))
3713081631934410656
>>> hash([1, 2])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

字典的方法

```
>>> adict.get('name')
'bob'
>>> print(adict.get('qq'))
None
>>> adict.get('name', 'not found')  # 字典中有name，返回val
'bob'
>>> adict.get('qq', '123456')  # 字典中没有qq，返回123456
'123456'

>>> adict.update({'qq': '135265234', 'phone':
'1508899007766'})
>>> adict
{'name': 'bob', 'age': 23, 'qq': '135265234', 'phone':
'1508899007766'}
>>> adict.pop('phone')
'1508899007766'
>>> adict.keys()
dict_keys(['name', 'age', 'qq'])
>>> adict.values()
dict_values(['bob', 23, '135265234'])
>>> adict.items()
dict_items([('name', 'bob'), ('age', 23), ('qq',
'135265234')])
```

集合

集合是数学上的概念，要求集合中的所有元素都是唯一的。

集合元素是唯一的、不可变的、无序的。集合就像是一个没有值的字典。

```
>>> aset = set('abc')
```

```
>>> aset
{'b', 'a', 'c'}
>>> bset = set('bcd')
>>> bset
{'b', 'c', 'd'}
>>> aset & bset   # 交集
{'b', 'c'}
>>> aset | bset   # 并集
{'c', 'b', 'a', 'd'}
>>> aset - bset   # 差补，只在aset中包含的元素
{'a'}
>>> bset - aset
{'d'}
```

集合方法

```
>>> aset.update(['d', 'e'])
>>> aset.add('hello')
>>> aset.remove('hello')
>>> aset
{'e', 'c', 'b', 'a', 'd'}
>>> bset
{'b', 'c', 'd'}
>>> aset.issuperset(bset)  # aset是bset的超集吗？
True
>>> bset.issubset(aset)  # bset是aset是aset的子集吗？
True
>>> aset.union(bset)  # aset | bset
{'c', 'b', 'a', 'd', 'e'}
>>> aset.intersection(bset)  # aset & bset
{'b', 'c', 'd'}
>>> aset.difference(bset)  # aset - bset
{'a', 'e'}
```

比较两个文件，将/tmp/mima中存在的，但是在/tmp/passwd中不存在的取出来。

```
[root@room8pc16 day05]# cp /etc/passwd /tmp/
[root@room8pc16 day05]# cp /etc/passwd /tmp/mima
[root@room8pc16 day05]# vim /tmp/mima  # 修改
```

```
>>> with open('/tmp/passwd') as f1:
...     aset = set(f1)
...
>>> with open('/tmp/mima') as f2:
...     bset = set(f2)
...
>>> cset = bset - aset
>>> bset - aset
{'hello world\n', 'my test\n'}
>>> with open('/tmp/diff.txt', 'w') as fobj:
...     fobj.writelines(cset)
```

```
>>> import random
>>> nums = [random.randint(1, 10) for i in range(20)]
>>> set(nums)
{1, 2, 3, 4, 5, 6, 7, 9, 10}
>>> list(set(nums))
[1, 2, 3, 4, 5, 6, 7, 9, 10]
##################################
>>> result = []
>>> for i in nums:
...     if i not in result:
...         result.append(i)
...
>>> result
[2, 10, 9, 3, 4, 1, 6, 5, 7]
```