nsd1912-py01-day05

列表

- 列表属于容器、可变、顺序类型

```
>>> l1 = [10, 20, 1, 3, 8, 10]
>>> l1[2]
1
>>> l1[2] = 100
>>> l1
[10, 20, 100, 3, 8, 10]
>>> l1[3:5]
[3, 8]
>>> l1[3:5] = [1, 3, 5, 7, 9]
>>> l1
[10, 20, 100, 1, 3, 5, 7, 9, 10]
```

- 列表方法

```
>>> l1.append(20)   # 追加
>>> l2 = l1.copy()  # 拷贝l1的数据给l2，它们使用不同的内存空间
>>> l2
[10, 20, 100, 1, 3, 5, 7, 9, 10, 20]
>>> l2.clear()
>>> l2
[]
>>> l1
[10, 20, 100, 1, 3, 5, 7, 9, 10, 20]
>>> l1.count(20)   # 统计列表中20的个数
2
>>> l1.extend([10, 20, 30])  # 批量添加数据
>>> l1
[10, 20, 100, 1, 3, 5, 7, 9, 10, 20, 10, 20, 30]
>>> l1.index(100)  # 100的下标
2
>>> l1.insert(2, 50)  # 在下标为2的位置插入50
>>> l1
[10, 20, 50, 100, 1, 3, 5, 7, 9, 10, 20, 10, 20, 30]
>>> l1.pop()  # 默认弹出最后一项
30
>>> l1
[10, 20, 50, 100, 1, 3, 5, 7, 9, 10, 20, 10, 20]
>>> l1.pop(2)  # 弹出下标为2的项目
50
>>> l1
[10, 20, 100, 1, 3, 5, 7, 9, 10, 20, 10, 20]

>>> l1.remove(20)   # 移除列表中第一个20
>>> l1.reverse()    # 翻转列表
>>> l1.sort()       # 升序排列
>>> l1
[1, 3, 7, 9, 10, 10, 20]
>>> l1.sort(reverse=True)  # 降序排列
>>> l1
[20, 10, 10, 9, 7, 3, 1]
```

对象的方法，就是函数。有些函数有返回值，有些函数没有返回值。

比如l1.pop()有返回值，可以把返回值赋值给一个变量；l1.remove()没有返回值，默认返回None。

```
>>> a = l1.pop()
>>> l1
[10, 1, 3, 5, 7, 9, 10, 20]
>>> a
10
>>> b = l1.remove(5)
>>> l1
[10, 1, 3, 7, 9, 10, 20]
>>> b
>>> print(b)
None
```

元组

- 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
- 单元素元组必须在元素后面加上逗号

```
>>> t1 = (10)
>>> type(t1)
<class 'int'>
>>> t1
10
>>> t2 = (10,)
>>> type(t2)
<class 'tuple'>
>>> len(t2)
1
```

- 元组的方法

```
>>> t3 = (10, 20, 10, 30, 10, 40)
>>> t3.count(10)  # 统计10出现的次数
3
>>> t3.index(20)  # 返回20的下标
1
```

练习：模拟栈结构

1. 发呆。思考程序的运行过程

```
[root@localhost day05]# python3 stack.py
(0) 压栈
(1) 出栈
(2) 查询
(3) 退出
请选择(0/1/2/3): abc
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:
(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): 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: world
(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): 2
[]
(0) 压栈
(1) 出栈
(2) 查询
(3) 退出
请选择(0/1/2/3): 1
栈是空的
(0) 压栈
(1) 出栈
(2) 查询
(3) 退出
请选择(0/1/2/3): 3
Bye-bye
```

2. 分析有哪些功能，并按照一定的规则调用函数，编写程序框架

```
def push_it():
    "用于将数据压入栈"

def pop_it():
    "用于将数据从栈弹出"

def view_it():
    "用于查询"

def show_menu():
    "用于显示菜单，根据用户的选择，不断调用其他函数 "

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

3. 编写每个函数的代码

```
stack = []

def push_it():
    "用于将数据压入栈"
    data = input('数据: ').strip()  # 去除字符串两端空格
    if data:  # 如果字符串非空
        stack.append(data)
    else:
        print('\033[31;1m请输入非空白字符串 \033[0m')

def pop_it():
    "用于将数据从栈弹出"
    if stack:  # 如果列表非空
        print('\033[34;1m从列表中，弹出了： %s\033[0m' % stack.pop())
    else:
        print('\033[31;1m栈是空的\033[0m')

def view_it():
    "用于查询"
    print('\033[32;1m%s\033[0m' % stack)

def show_menu():
    "用于显示菜单，根据用户的选择，不断调用其他函数 "
    # 将函数存入到字典，函数后面不要加 ()，如果加()是把函数的返回值存入字典
    funcs = {'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:
        xz = input(prompt).strip()
        if xz not in ['0', '1', '2', '3']:
            print('\033[31;1m无效的选择，请重试。 \033[0m')
            continue

        if xz == '3':
            print('Bye-bye')
            break

        funcs[xz]()  # 在字典中取出函数并调用

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

> Important：请大家及早修炼出一双透视眼。要透过变量名看到它的值，要透过函数名看到它代表的代码，透过函数调用看到它的返回值。

```
def func1():
    print('Hello func1')

def func2():
    print('Welcome func2')

if __name__ == '__main__':
    funcs1 = {'0': func1, '1': func2}
    funcs2 = {'0': func1(), '1': func2()}
    print(funcs1)
    print(funcs2)
```

字典

- 字典属于容器、可变、映射类型
- 字典的key不能重复
- 字典的key必须是不可变对象

```
>>> d1 = {'name': 'tom', 'name': 'jerry', 'age': 20}
>>> d1
{'name': 'jerry', 'age': 20}
>>> d2 = dict([('name', 'bob'), ('age', 20)])
>>> d2
{'name': 'bob', 'age': 20}
>>> d3 = {}.fromkeys(('tom', 'jerry', 'bob', 'alice'), 18)
>>> d3
{'tom': 18, 'jerry': 18, 'bob': 18, 'alice': 18}
```

- 访问字典

```
>>> d2
{'name': 'bob', 'age': 20}
>>> 'name' in d2
True
>>> for k in d2:
...   print(k, d2[k])
...
name bob
age 20
>>> '%s is %s years old' % (d2['name'], d2['age'])
'bob is 20 years old'
>>> '%(name)s is %(age)s years old' % d2
'bob is 20 years old'
```

- 更新字典：通过key来改变value，如果key已经在字典中了，则修改它的值；如果key不在字典中，则加入新值

```
>>> d2
{'name': 'bob', 'age': 20}
>>> d2['age'] = 22
>>> d2
{'name': 'bob', 'age': 22}
>>> d2['email'] = 'bob@tedu.cn'
>>> d2
{'name': 'bob', 'age': 22, 'email': 'bob@tedu.cn'}
```

- 字典方法

```
# get方法是字典最重要的方法
>>> d2.get('name')  # 通过key取值
'bob'
>>> d2['phone']     # 报错
>>> print(d2.get('phone'))  # 字典没有key返回None
None
>>> d2.get('phone', '110')  # 没有key返回110
'110'
>>> d2.get('age', 'not found')  # key存在，返回value
22

>>> d2.keys()
dict_keys(['name', 'age', 'email'])
>>> list(d2.keys())
['name', 'age', 'email']
>>> list(d2.values())
['bob', 22, 'bob@tedu.cn']
>>> list(d2.items())
[('name', 'bob'), ('age', 22), ('email', 'bob@tedu.cn')]
>>> d2.update({'qq': '123545', 'phone': '13511223344'})
>>> d2
{'name': 'bob', 'age': 22, 'email': 'bob@tedu.cn', 'qq': '123545', 'phone':
'13511223344'}
>>> d2.pop('phone')
'13511223344'
>>> d2
{'name': 'bob', 'age': 22, 'email': 'bob@tedu.cn', 'qq': '123545'}
```

集合

- 集合是一个数学上的概念，它由不同元素构成。
- 集合没有顺序
- 集合元素只能是不可变对象
- 集合就像是一个无值的字典
- 集合分有可变集合set，和不可变集合frozenset

```
>>> s1 = {'tom', 'jerry', 'bob', 'alice'}
>>> s2 = set('abc')
>>> s3 = set('bcd')
>>> s2
{'b', 'a', 'c'}
>>> s3
{'b', 'c', 'd'}
>>> s4 = set([10, 20, 30, 40])
>>> s4
{40, 10, 20, 30}
>>> len(s1)
4
>>> for name in s1:
...   print(name)

>>> s2 & s3   # 交集
{'b', 'c'}
>>> s2 | s3   # 并集
{'d', 'b', 'a', 'c'}
>>> s2 - s3   # 差补，s2中有s3中无
{'a'}

>>> s1.add('zhangsan')  # 增加元素
>>> s1.remove('bob')   # 岫除元素
>>> s1
{'jerry', 'tom', 'zhangsan', 'alice'}
>>> s1.pop()    # 随机弹出一项
'jerry'
>>> s1
{'tom', 'zhangsan', 'alice'}
>>> s1.update(['wangwu', 'zhaoliu'])
>>> s1
{'zhaoliu', 'wangwu', 'tom', 'zhangsan', 'alice'}
>>> s2.union(s3)   # s2 | s3
{'d', 'b', 'a', 'c'}
>>> s2.intersection(s3)  # s2 & s3
{'b', 'c'}
>>> s2.difference(s3)  # s2 - s3
{'a'}
>>> s5 = s2.union(s3)
>>> s2.issubset(s5)    # s2是s5的子集吗
True
>>> s5.issuperset(s2)  # s5是s2的超集吗？
True
```

- 集合经常用于去重

```
>>> from random import randint
>>> nums = [randint(1, 30) for i in range(20)]
>>> nums
[21, 16, 9, 11, 30, 3, 10, 15, 14, 25, 24, 7, 10, 21, 20, 12, 3, 7, 8, 10]
>>> set(nums)
{3, 7, 8, 9, 10, 11, 12, 14, 15, 16, 20, 21, 24, 25, 30}
>>> list(set(nums))

[3, 7, 8, 9, 10, 11, 12, 14, 15, 16, 20, 21, 24, 25, 30]
```

练习：找到b.log中有，a.log中没有的行

```
[root@localhost day05]# cp /etc/passwd /tmp/
[root@localhost day05]# cp /etc/passwd /tmp/mima
[root@localhost day05]# vim /tmp/mima  # 改动文件
>>> with open('/tmp/mima') as f1:
...   s1 = set(f1)
>>> with open('/tmp/passwd') as f2:
...   s2 = set(f2)
>>> s1 - s2
{'wan shang chi shen me?\n', 'hao are you?\n'}
>>> with open('/tmp/result.txt', 'w') as f3:
...   f3.writelines(s1 - s2)
[root@localhost day05]# cat /tmp/result.txt
```