

[笔记][黑马 Python 之 Python 基础 - 6]

Python

[笔记][黑马 Python 之 Python 基础 - 6]

- 190. 字符串-06-字符串的查找和替换
- 191. 字符串-07-文本对齐方法演练
- 192. 字符串-08-去除空白字符
- 193. 字符串-09-拆分和拼接字符串
- 194. 字符串-10-切片概念和语法以及倒序索引
- 195. 字符串-11-字符串切片演练
- 196. 公共方法-01-内置函数长度、删除、最大、最小、比较
- 197. 公共方法-02-切片
- 198. 公共方法-03-算术运算符及对比列表追加方法
运算符
- 199. 公共方法-04-成员运算符
- 200. 公共方法-05-完整的for循环-for else
- 201. 公共方法-06-利用for else搜索字典列表
- 202. 名片管理-01-明确目标及备课代码演示
- 203-框架搭建-01-框架介绍及系统架构分析
- 204. 框架搭建-02-新建项目准备文件
- 205. 框架搭建-03-用户输入判断和pass关键字
- 206. 框架搭建-04-无限循环保证用户能够重复选择操作
- 207. 框架搭建-05-if嵌套判断用户的具体操作预留代码位置
- 208. 框架搭建-06-cards_main知识点小结
- 209. 框架搭建-07-[扩展]TODO注释及格式
- 210. 框架搭建-08-显示欢迎界面及功能菜单
- 211. 框架搭建-09-准备名片操作函数修改主文件中函数调用
- 212. 数据结构确定-01-确定名片管理系统的数据结构
- 213. 新增名片-01-新增名片功能实现
- 214. 新增名片-02-[扩展]PyCharm技巧重命名变量
- 215. 显示全部-01-遍历列表显示字典明细
- 216. 显示全部-02-判断列表数量，没有名片直接返回
- 217. 查询名片-01-查询功能实现
- 218. 查询名片-02-准备处理名片函数
- 219. 处理名片-01-增加分支判断用户操作类型
- 220. 处理名片-02-删除名片
- 221. 处理名片-03-修改名片
- 222. 处理名片-04-明确细化修改名片的思路，准备新的输入函数

190. 字符串-06-字符串的查找和替换

```
hello_str = 'hello world'

# 1. 判断是否以指定字符串开始
print(hello_str.startswith('Hello'))

# 2. 判断是否以指定字符串结束
print(hello_str.endswith('world'))

# 3. 查找指定字符串
# index 同样可以查找指定的字符串在大字符串中的索引
# index 如果指定的字符串不存在，会报错
# find 如果指定的字符串不存在，会返回-1
print(hello_str.find('llo'))
print(hello_str.find('abc')) # 返回 -1

# 4. 替换字符串
# replace 方法执行完成之后，会返回一个新的字符串
# 注意：不会修改原有字符串的内容
print(hello_str.replace('world', 'python'))

print(hello_str)
```

191. 字符串-07-文本对齐方法演练

两个字符串连着写，比如下面这样，会拼接字符串：

```
'你好''世界'
```

或者

```
'你好'
'世界'
```

```

# 假设：以下内容是从网上抓取的
# 要求：顺序并且居中对齐输出以下内容

poem = ['登鹳雀楼',
        '王之涣',
        '白日依山尽',
        '黄河入海流',
        '欲穷千里目',
        '更上一层楼']

for poem_str in poem:

    print('%s' % poem_str.center(10))

```

结果是：

```

| 登鹳雀楼 |
| 王之涣   |
| 白日依山尽 |
| 黄河入海流 |
| 欲穷千里目 |
| 更上一层楼 |

```

效果并不好，因为 `center` 使用英文空格进行居中


`.center(10))`

```

str
def center(self,
            width: int,
            fillchar: str) -> str

```

`S.center(width[, fillchar]) -> str`
 Return S centered in a string of length width. Padding is done using the specified fill character (default is a space)

 < Python 3.6 >

可以传入第二个参数 `fillchar`，使用全角空格

```
# 假设：以下内容是从网上抓取的
# 要求：顺序并且居中对齐输出以下内容

poem = ['登鹳雀楼',
        '王之涣',
        '白日依山尽',
        '黄河入海流',
        '欲穷千里目',
        '更上一层楼']

for poem_str in poem:

    print('|%s|' % poem_str.center(10, ' '))
```

这样的输出结果就很整齐了

```
|      登鹳雀楼      |
|      王之涣        |
|      白日依山尽    |
|      黄河入海流    |
|      欲穷千里目    |
|      更上一层楼    |
```

```
# 假设：以下内容是从网上抓取的
# 要求：顺序并且居中对齐输出以下内容

poem = ['登鹳雀楼',
        '王之涣',
        '白日依山尽',
        '黄河入海流',
        '欲穷千里目',
        '更上一层楼']

for poem_str in poem:

    # print('|%s|' % poem_str.center(10, ' '))
    # print('|%s|' % poem_str.ljust(10, ' '))
    print('|%s|' % poem_str.rjust(10, ' '))
```

192. 字符串-08-去除空白字符

lstrip
rstrip
strip

```
# 假设：以下内容是从网上抓取的
# 要求：顺序并且居中对齐输出以下内容

poem = ['登鹳雀楼',
        '王之涣',
        '白日依山尽\n',
        '\t黄河入海流',
        '欲穷千里目',
        '更上一层楼']

for poem_str in poem:
    # 先使用 strip 方法去除字符串中的空白字符
    # 再使用 center 方法居中显示文本
    print('|%s|' % poem_str.strip().center(10, ' '))

    # print('|%s|' % poem_str.ljust(10, ' '))
    # print('|%s|' % poem_str.rjust(10, ' '))
```

193. 字符串-09-拆分和拼接字符串

方法	说明
string.partition(str)	把字符串 string 分成一个 3 元素的元组 (str前面, str, str后面)
string.rpartition(str)	类似于 partition() 方法，不过是从右边开始查找
string.split(sep="", maxsplit)	以 str 为分隔符拆分 string，如果 num 有指定值，则仅分隔 num + 1 个子字符串，str 默认包含 '\r', '\t', '\n' 和空格
string.splitlines()	按照行 ('\r', '\n', '\r\n') 分隔，返回一个包含各行作为元素的列表
string.join(seq)	以 string 作为分隔符，将 seq 中所有的元素（的字符串表示）合并为一个新的字符串

string.split(sep="", maxsplit) 默认使用空白字符分割，否则以 str 指定值分割，分割 num 次

string.splitlines() 按照行分割

string.join(seq) 以 string 为分隔符，把 seq 所有元素拼接成字符串

```

# 假设：以下内容是从网络上抓取的
# 要求：
# 1. 将字符串中的空白字符全部去掉
# 2. 再使用 " " 作为分隔符，拼接成一个整齐的字符串
poem_str = '登鹳雀楼\t王之涣\t白日依山尽\t\n黄河入海流\t\t欲穷千里目\t\n更上一层楼'
print(poem_str)

# 1. 拆分字符串
poem_list = poem_str.split()
print(poem_list)

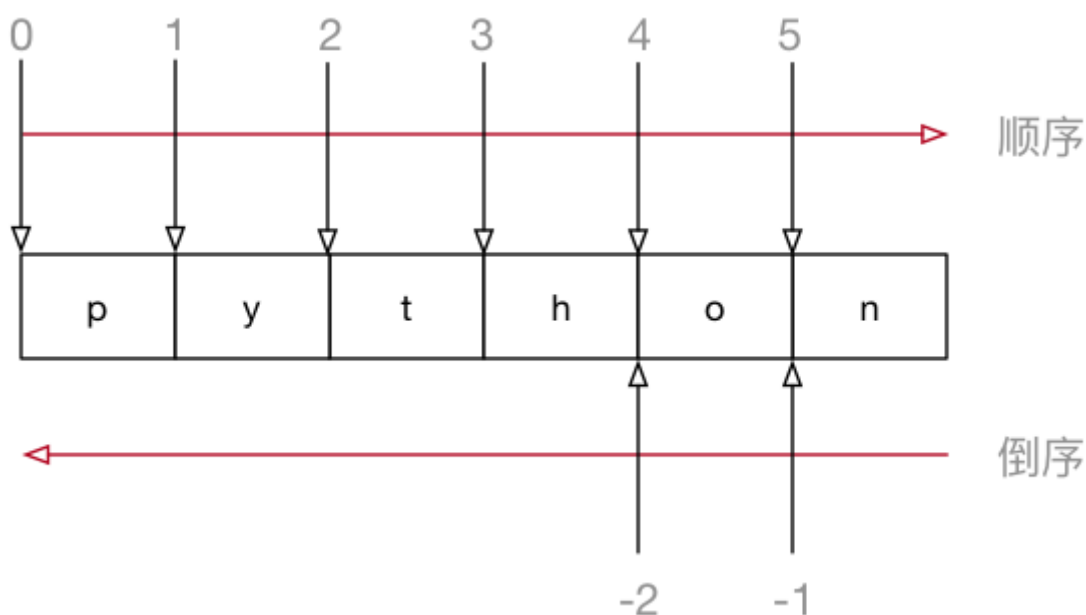
# 2. 合并字符串
result = ' '.join(poem_list)
print(result)

```

194. 字符串-10-切片概念和语法以及倒序索引

切片 方法适用于 字符串、列表、元组

- 切片 使用 索引值 来限定范围，从一个大的 字符串 中 切出 小的 字符串
- 列表 和 元组 都是 有序 的集合，都能够 通过索引值 获取到对应的数据
- 字典 是一个 无序 的集合，是使用 键值对 保存数据



字符串[开始索引:结束索引:步长]

注意：

1. 指定的区间属于 **左闭右开** 型 [开始索引, 结束索引) , 即 开始索引 <= 范围 < 结束索引
 - 从 **起始** 位开始, 到 **结束** 位的前一位 结束 (**不包含结束位本身**)
2. 从头开始, **开始索引** 数字可以省略, 冒号不能省略
3. 到末尾结束, **结束索引** 数字可以省略, 冒号不能省略
4. 步长默认为 **1** , 如果连续切片, **数字和冒号都可以省略**

索引的顺序和倒序

- 在 Python 中不仅支持 **顺序索引**, 同时还支持 **倒序索引**
- 所谓倒序索引就是 **从右向左** 计算索引
 - 最右边的索引值是 **-1**, 依次递减

195. 字符串-11-字符串切片演练

演练需求

- 截取从 2 ~ 5 位置 的字符串
- 截取从 2 ~ **末尾** 的字符串
- 截取从 **开始** ~ 5 位置 的字符串
- 截取完整的字符串
- 从开始位置, 每隔一个字符截取字符串
- 从索引 1 开始, 每隔一个取一个
- 截取从 2 ~ **末尾 - 1** 的字符串
- 截取字符串末尾两个字符
- 字符串的逆序 (面试题)

答案

```
num_str = "0123456789"

# 1. 截取从 2 ~ 5 位置 的字符串
print(num_str[2:6])

# 2. 截取从 2 ~ `末尾` 的字符串
print(num_str[2:])

# 3. 截取从 `开始` ~ 5 位置 的字符串
print(num_str[:6])

# 4. 截取完整的字符串
print(num_str[:])

# 5. 从开始位置, 每隔一个字符截取字符串
print(num_str[::2])

# 6. 从索引 1 开始, 每隔一个取一个
```

```

print(num_str[1::2])

# 倒序切片
# -1 表示倒数第一个字符
print(num_str[-1])

# 7. 截取从 2 ~ `末尾 - 1` 的字符串
print(num_str[2:-1])

# 8. 截取字符串末尾两个字符
print(num_str[-2:])

# 9. 字符串的逆序（面试题）
print(num_str[::-1])

```

196. 公共方法-01-内置函数长度、删除、最大、最小、比较

Python 内置函数

Python 包含了以下内置函数：

函数	描述	备注
len(item)	计算容器中元素个数	
del(item)	删除变量	del 有两种方式
max(item)	返回容器中元素最大值	如果是字典，只针对 <code>key</code> 比较
min(item)	返回容器中元素最小值	如果是字典，只针对 <code>key</code> 比较
cmp(item1, item2)	比较两个值，-1 小于/0 相等/1 大于	Python 3.x 取消了 cmp 函数

注意

- **字符串** 比较符合以下规则：“0” < “A” < “a”

`del` 的两种方式

```

>>> a = [1, 2, 3]
>>> del a[1]
>>> a
[1, 3]
>>> del(a[1])

```



```
>>> a
[1]
>>> del(a)
>>> a
Traceback (most recent call last):
  File "<input>", line 1, in <module>
NameError: name 'a' is not defined
```

python 3 取消了 `cmp`，可以使用比较运算符来进行比较
字符串，列表，元组都可以比较大小
但是字典不能比较大小

197. 公共方法-02-切片

切片

描述	Python 表达式	结果	支持的数据类型
切片	"0123456789"[:-2]	"97531"	字符串、列表、元组

- 切片 使用 索引值 来限定范围，从一个大的 字符串 中 切出 小的 字符串
- 列表 和 元组 都是 有序 的集合，都能够 通过索引值 获取到对应的数据
- 字典 是一个 无序 的集合，是使用 键值对 保存数据

198. 公共方法-03-算术运算符及对比列表追加方法

运算符

运算符	Python 表达式	结果	描述	支持的数据类型
+	[1, 2] + [3, 4]	[1, 2, 3, 4]	合并	字符串、列表、元组
*	['Hi!'] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	重复	字符串、列表、元组
in	3 in (1, 2, 3)	True	元素是否存在	字符串、列表、元组、字典

not in	4 not in (1, 2, 3)	True	元素是否不存在	字符串、列表、元组、字典
= == < <=	(1, 2, 3) < (2, 2, 3)	True	元素比较	字符串、列表、元组

注意

- `in` 在对 **字典** 操作时，判断的是 **字典的键**
- `in` 和 `not in` 被称为 **成员运算符**

注意列表的 `append` 和 `extend` 方法的区别。

199. 公共方法-04-成员运算符

成员运算符用于 **测试** 序列中是否包含指定的 **成员**

运算符	描述	实例
in	如果在指定的序列中找到值返回 True，否则返回 False	<code>3 in (1, 2, 3)</code> 返回 <code>True</code>
not in	如果在指定的序列中没有找到值返回 True，否则返回 False	<code>3 not in (1, 2, 3)</code> 返回 <code>False</code>

注意：在对 **字典** 操作时，判断的是 **字典的键**

200. 公共方法-05-完整的for循环-for else

完整的 for 循环语法

- 在 `Python` 中完整的 `for` 循环 的语法如下：

```
for 变量 in 集合:

    循环体代码

else:
    没有通过 break 退出循环，循环结束后，会执行的代码
```

```
for num in [1, 2, 3]:
    print(num)
    if num == 2:
        break
else:
    # 如果循环体内部使用了break退出循环
    # else下方的代码就不会被执行
    print('会执行吗? ')

print('循环结束')
```

201. 公共方法-06-利用for else搜索字典列表

应用场景

- 在 **迭代遍历** 嵌套的数据类型时，例如 **一个列表包含了多个字典**
- 需求：要判断 某一个字典中 是否存在 指定的 值
 - 如果 **存在**，提示并且退出循环
 - 如果 **不存在**，在 **循环整体结束** 后，希望 **得到一个统一的提示**

```
students = [
    {'name': '阿土'},
    {'name': '小美'}
]

# 在学员列表中搜索指定的姓名
find_name = '张三'

for stu_dict in students:
    print(stu_dict)
    if stu_dict['name'] == find_name:
        print('找到了 %s' % find_name)
        # 如果已经找到，应该直接退出循环，而不再遍历后续的元素
        break
else:
    # 如果希望在搜索列表时，所有的字典检查之后，都没有发现需要搜索的目标
    # 还希望得到一个统一的提示！
    print('抱歉，没有找到 %s' % find_name)
print('循环结束')
```

【!】我一般都是采用一个 `flag` 标志，如果找到就设置为 `True`，否则为 `False`
然后在 `for` 循环后面用 `if` 判断是否找到
其实可以用 `for else` 来处理

202. 名片管理-01-明确目标及备课代码演示

目标

综合应用已经学习过的知识点：

- 变量
- 流程控制
- 函数
- 模块

开发 名片管理系统

系统需求

1. 程序启动，显示名片管理系统欢迎界面，并显示功能菜单

```
*****
欢迎使用【名片管理系统】V1.0

1. 新建名片
2. 显示全部
3. 查询名片

0. 退出系统
*****
```

2. 用户用数字选择不同的功能
3. 根据功能选择，执行不同的功能
4. 用户名片需要记录用户的 **姓名、电话、QQ、邮件**
5. 如果查询到指定的名片，用户可以选择 **修改** 或者 **删除** 名片

步骤

1. 框架搭建
2. 新增名片

3. 显示所有名片
4. 查询名片
5. 查询成功后修改、删除名片
6. 让 Python 程序能够直接运行

203-框架搭建-01-框架介绍及系统架构分析

目标

- 搭建名片管理系统 **框架结构**
 1. **准备文件**，确定文件名，保证能够 **在需要的位置** 编写代码
 2. 编写 **主运行循环**，实现基本的 **用户输入和判断**

204. 框架搭建-02-新建项目准备文件

文件准备

1. 新建 `cards_main.py` 保存 **主程序功能代码**
 - 程序的入口
 - 每一次启动名片管理系统都通过 `main` 这个文件启动
2. 新建 `cards_tools.py` 保存 **所有名片功能函数**
 - 将对名片的 **新增、查询、修改、删除** 等功能封装在不同的函数中

205. 框架搭建-03-用户输入判断和pass关键字

```
action_str = input('请选择希望执行的操作：')
print('您选择的操作是【%s】' % action_str)

# 1, 2, 3 针对名片的操作
if action_str in ['1', '2', '3']:
    pass

# 0 退出系统
elif action_str == '0':
    # 如果在开发程序时，不希望立刻编写分支内部的代码
```

```
# 可以使用 pass 关键字，表示一个占位符，能够保证程序的代码结构正确！
# 程序运行时，pass 关键字不会执行任何的操作！

pass

# 其他内容输入错误，需要提示用户
else:
    print('您输入的不正确，请重新选择')
```

206. 框架搭建-04-无限循环保证用户能够重复选择操作

```
# 无限循环，由用户主动决定什么时候推出循环！
while True:
    action_str = input('请选择希望执行的操作：')
    print('您选择的操作是【%s】' % action_str)

    # 1, 2, 3 针对名片的操作
    if action_str in ['1', '2', '3']:
        pass

    # 0 退出系统
    elif action_str == '0':
        # 如果在开发程序时，不希望立刻编写分支内部的代码
        # 可以使用 pass 关键字，表示一个占位符，能够保证程序的代码结构正确！
        # 程序运行时，pass 关键字不会执行任何的操作！
        # pass
        print('欢迎再次使用【名片管理系统】')
        break

    # 其他内容输入错误，需要提示用户
    else:
        print('您输入的不正确，请重新选择')
```

207. 框架搭建-05-if嵌套判断用户的具体操作预留代码位置

```
# 无限循环，由用户主动决定什么时候推出循环！
while True:
    # 显示功能菜单
```

```
action_str = input('请选择希望执行的操作：')
print('您选择的操作是【%s】' % action_str)

# 1, 2, 3 针对名片的操作
if action_str in ['1', '2', '3']:
    # 新增名片
    if action_str == '1':
        pass

    # 显示全部
    elif action_str == '2':
        pass

    # 查询名片
    elif action_str == '3':
        pass

# 0 退出系统
elif action_str == '0':
    # 如果在开发程序时，不希望立刻编写分支内部的代码
    # 可以使用 pass 关键字，表示一个占位符，能够保证程序的代码结构正确！
    # 程序运行时，pass 关键字不会执行任何的操作！
    # pass
    print('欢迎再次使用【名片管理系统】')
    break

# 其他内容输入错误，需要提示用户
else:
    print('您输入的不正确，请重新选择')
```

208. 框架搭建-06-cards_main知识点小结

字符串判断

```
if action in ["1", "2", "3"]:
```

```
if action == "1" or action == "2" or action == "3":
```

1. 使用 `in` 针对 **列表** 判断，避免使用 `or` 拼接复杂的逻辑条件
2. 没有使用 `int` 转换用户输入，可以避免 **一旦用户输入的不是数字**，导致程序运行出错

pass

- `pass` 就是一个空语句，不做任何事情，一般用做占位语句
- 是为了保持程序结构的完整性

无限循环

- 在开发软件时，如果 **不希望程序执行后 立即退出**
- 可以在程序中增加一个 **无限循环**
- **由用户来决定** 退出程序的时机

209. 框架搭建-07-[扩展]TODO注释及格式

```
# 无限循环，由用户主动决定什么时候推出循环！
while True:
    # TODO(jpch89@outlook.com) 显示功能菜单

    action_str = input('请选择希望执行的操作：')
    print('您选择的操作是【%s】' % action_str)

    # 1, 2, 3 针对名片的操作
    if action_str in ['1', '2', '3']:
        # 新增名片
        if action_str == '1':
            pass

        # 显示全部
        elif action_str == '2':
            pass

        # 查询名片
        elif action_str == '3':
            pass

    # 0 退出系统
    elif action_str == '0':
        # 如果在开发程序时，不希望立刻编写分支内部的代码
        # 可以使用 pass 关键字，表示一个占位符，能够保证程序的代码结构正确！
        # 程序运行时，pass 关键字不会执行任何的操作！
        # pass
        print('欢迎再次使用【名片管理系统】')
```


break

```
# 其他内容输入错误，需要提示用户
else:
    print('您输入的不正确，请重新选择')
```

TODO 注释

- 在 `#` 后跟上 `TODO`，用于标记需要去做的工作

```
# TODO(作者/邮件) 显示系统菜单
```

210. 框架搭建-08-显示欢迎界面及功能菜单

完成之后要删除 `TODO` 注释

`cards_main.py`

```
import cards_tools

# 无限循环，由用户主动决定什么时候推出循环！
while True:
    # 显示功能菜单
    cards_tools.show_menu()
    action_str = input('请选择希望执行的操作：')
    print('您选择的操作是【%s】' % action_str)

    # 1, 2, 3 针对名片的操作
    if action_str in ['1', '2', '3']:
        # 新增名片
        if action_str == '1':
            pass

        # 显示全部
        elif action_str == '2':
            pass

        # 查询名片
        elif action_str == '3':
            pass

    # 0 退出系统
```

```

elif action_str == '0':
    # 如果在开发程序时，不希望立刻编写分支内部的代码
    # 可以使用 pass 关键字，表示一个占位符，能够保证程序的代码结构正确！
    # 程序运行时，pass 关键字不会执行任何的操作！
    # pass
    print('欢迎再次使用【名片管理系统】')
    break

# 其他内容输入错误，需要提示用户
else:
    print('您输入的不正确，请重新选择')

```

cards_tools.py

```

def show_menu():
    """
    显示菜单
    :return: None
    """
    print('*' * 50)
    print('欢迎使用【名片管理系统】V 1.0')
    print("")
    print("1. 新增名片")
    print("2. 显示全部")
    print('3. 搜索名片')
    print('')
    print('0. 退出系统')
    print('*' * 50)

```

211. 框架搭建-09-准备名片操作函数修改主文件中函数调用

cards_main.py

```

import cards_tools

# 无限循环，由用户主动决定什么时候推出循环！
while True:
    # 显示功能菜单
    cards_tools.show_menu()
    action_str = input('请选择希望执行的操作：')
    print('您选择的操作是【%s】' % action_str)

```

```

# 1, 2, 3 针对名片的操作
if action_str in ['1', '2', '3']:
    # 新增名片
    if action_str == '1':
        cards_tools.new_card()

    # 显示全部
    elif action_str == '2':
        cards_tools.show_all()

    # 查询名片
    elif action_str == '3':
        cards_tools.search_card()

# 0 退出系统
elif action_str == '0':
    # 如果在开发程序时，不希望立刻编写分支内部的代码
    # 可以使用 pass 关键字，表示一个占位符，能够保证程序的代码结构正确！
    # 程序运行时，pass 关键字不会执行任何的操作！
    # pass
    print('欢迎再次使用【名片管理系统】')
    break

# 其他内容输入错误，需要提示用户
else:
    print('您输入的不正确，请重新选择')

```

cards_tools.py

```

def show_menu():
    """显示菜单

    :return: None
    """
    print('*' * 50)
    print('欢迎使用【名片管理系统】V 1.0')
    print("")
    print("1. 新增名片")
    print("2. 显示全部")
    print('3. 搜索名片')
    print('')
    print('0. 退出系统')
    print('*' * 50)

def new_card():
    """ 新增名片

    :return: None

```

```
"""
print('-' * 50)
print('新增名片')

def show_all():
    """ 显示所有名片

    :return: None
    """
    print('-' * 50)
    print('显示所有名片')

def search_card():
    """ 搜索名片

    :return: None
    """
    print('-' * 50)
    print('搜索名片')
```

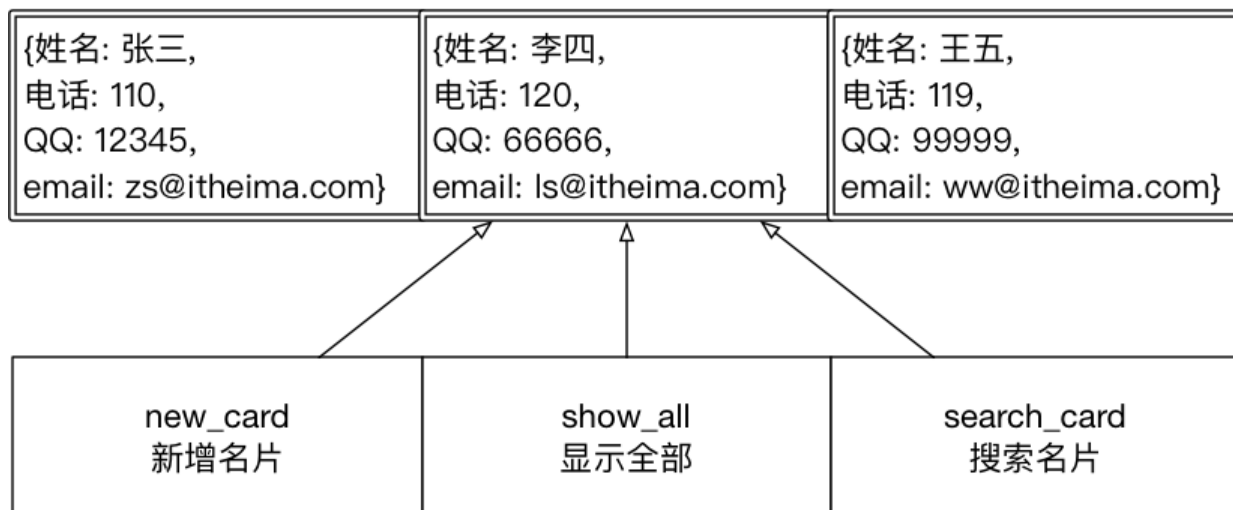
到这里主程序的框架已经搭建完毕，后面只要把开发精力集中在 `cards_tools.py` 模块文件上即可。

212. 数据结构确定-01-确定名片管理系统的数据结构

程序就是用来处理数据的，而变量就是用来存储数据的

- 使用 **字典** 记录 **每一张名片** 的详细信息
- 使用 **列表** 统一记录所有的 **名片字典**

card_list 列表记录所有名片字典



定义名片列表变量

- 在 `cards_tools` 文件的顶部增加一个 **列表变量**

```
# 所有名片记录的列表  
card_list = []
```

注意

- 所有名片相关操作，都需要使用这个列表，所以应该 **定义在程序的顶部**
- 程序刚运行时，没有数据，所以是 **空列表**

213. 新增名片-01-新增名片功能实现

```
def new_card():  
    """ 新增名片  
  
    :return: None  
    """  
    print('-' * 50)  
    print('新增名片')  
  
    # 1. 提示用户输入名片的详细信息  
    name = input('请输入姓名: ')  
    phone = input('请输入电话: ')  
    qq = input('请输入QQ: ')  
    email = input('请输入邮箱: ')
```

```
# 2. 使用用户输入的信息建立一个名片字典
card_dict = {'name': name,
             'phone': phone,
             'qq': qq,
             'email': email}

# 3. 将名片字典添加到列表中
card_list.append(card_dict)
print(card_list)

# 4. 提示用户添加成功
print('添加 %s 的名片成功' % name)
```

214. 新增名片-02-[扩展]PyCharm技巧 重命名变量

批量重命名变量名

右键单击变量名

Refractor

Rename 快捷键：Shift + F6

Rename code occurrences 修改变量名（第一个选项）

修改完毕之后按回车

开发完成之后，可以折叠该函数（功能）的代码

215. 显示全部-01-遍历列表显示字典明 细

```
def show_all():
    """ 显示所有名片

    :return: None
    """
    print('-' * 50)
    print('显示所有名片')

    # 打印表头
    for name in ['姓名', '电话', 'QQ', '邮箱']:
        print(name, end='\t\t')
    print('')

    # 打印分隔线
    print('=' * 50)
```

```
# 遍历名片列表依次输出字典信息
for card_dict in card_list:
    print('%s\t%s\t%s\t%s' % (card_dict['name'],
                              card_dict['phone'],
                              card_dict['qq'],
                              card_dict['email']))
```

216. 显示全部-02-判断列表数量，没有名片直接返回

```
def show_all():
    """ 显示所有名片

    :return: None
    """
    print('-' * 50)
    print('显示所有名片')

    # 判断是否存在名片记录，如果没有，提示用户并且返回
    if len(card_list) == 0:
        print('当前没有任何的名片记录，请使用新增名片功能添加名片！')
        # return 可以返回一个函数的执行结果
        # 下方的代码不会被执行
        # 如果 return 后面没有任何内容，表示会返回到调用函数的位置
        # 并且不返回任何的结果（实际上返回 None）
        return

    # 打印表头
    for name in ['姓名', '电话', 'QQ', '邮箱']:
        print(name, end='\t\t')
    print('')
    # 打印分隔线
    print('=' * 50)

    # 遍历名片列表依次输出字典信息
    for card_dict in card_list:
        print('%s\t%s\t%s\t%s' % (card_dict['name'],
                                  card_dict['phone'],
                                  card_dict['qq'],
                                  card_dict['email']))
```

217. 查询名片-01-查询功能实现

【!】 判断是否相等，简称判等

【!】 `for else` 一般必须要配合 `break` 使用

```
# 记录所有的名片字典
card_list = []

def show_menu():
    """显示菜单

    :return: None
    """
    print('*' * 50)
    print('欢迎使用【名片管理系统】V 1.0')
    print("")
    print("1. 新增名片")
    print("2. 显示全部")
    print("3. 搜索名片")
    print('')
    print('0. 退出系统')
    print('*' * 50)

def new_card():
    """ 新增名片

    :return: None
    """
    print('-' * 50)
    print('新增名片')

    # 1. 提示用户输入名片的详细信息
    name_str = input('请输入姓名: ')
    phone_str = input('请输入电话: ')
    qq_str = input('请输入QQ: ')
    email_str = input('请输入邮箱: ')
    # 2. 使用用户输入的信息建立一个名片字典
    card_dict = {'name': name_str,
                 'phone': phone_str,
                 'qq': qq_str,
                 'email': email_str}
    # 3. 将名片字典添加到列表中
    card_list.append(card_dict)
    print(card_list)
    # 4. 提示用户添加成功
    print('添加 %s 的名片成功' % name_str)
```



```

        card_dict['qq'],
        card_dict['email']))

    # TODO 针对找到的名片记录执行修改和删除的操作
    break

else:
    print('抱歉，没有找到 %s' % find_name)

```

218. 查询名片-02-准备处理名片函数

```

# 记录所有的名片字典
card_list = []

def show_menu():
    """显示菜单

    :return: None
    """
    print('*' * 50)
    print('欢迎使用【名片管理系统】V 1.0')
    print("")
    print("1. 新增名片")
    print("2. 显示全部")
    print("3. 搜索名片")
    print('')
    print('0. 退出系统')
    print('*' * 50)

def new_card():
    """ 新增名片

    :return: None
    """
    print('-' * 50)
    print('新增名片')

    # 1. 提示用户输入名片的详细信息
    name_str = input('请输入姓名: ')
    phone_str = input('请输入电话: ')
    qq_str = input('请输入QQ: ')
    email_str = input('请输入邮箱: ')
    # 2. 使用用户输入的信息建立一个名片字典
    card_dict = {'name': name_str,
                  'phone': phone_str,
                  'qq': qq_str,

```

```

        'email': email_str}

# 3. 将名片字典添加到列表中
card_list.append(card_dict)
print(card_list)

# 4. 提示用户添加成功
print('添加 %s 的名片成功' % name_str)


def show_all():
    """ 显示所有名片

    :return: None
    """
    print('-' * 50)
    print('显示所有名片')

    # 判断是否存在名片记录, 如果没有, 提示用户并且返回
    if len(card_list) == 0:
        print('当前没有任何的名片记录, 请使用新增名片功能添加名片!')
        # return 可以返回一个函数的执行结果
        # 下方的代码不会被执行
        # 如果 return 后面没有任何内容, 表示会返回到调用函数的位置
        # 并且不返回任何的结果 (实际上返回 None)
        return

    # 打印表头
    for name in ['姓名', '电话', 'QQ', '邮箱']:
        print(name, end='\t\t')
    print('')
    # 打印分隔线
    print('=' * 50)

    # 遍历名片列表依次输出字典信息
    for card_dict in card_list:
        print('%s\t%s\t%s\t%s' % (card_dict['name'],
                                  card_dict['phone'],
                                  card_dict['qq'],
                                  card_dict['email']))


def search_card():
    """ 搜索名片

    :return: None
    """
    print('-' * 50)
    print('搜索名片')

    # 1. 提示用户输入要搜索的姓名
    find_name = input('请输入要搜索的姓名: ')

    # 2. 遍历名片列表, 查询要搜索的姓名, 如果没有找到, 需要提示用户

```

```

for card_dict in card_list:
    if card_dict['name'] == find_name:
        print('姓名\t\t电话\t\tQQ\t\t邮箱')
        print('=' * 50)
        print('%s\t\t%s\t\t%s\t\t%s' % (card_dict['name'],
                                       card_dict['phone'],
                                       card_dict['qq'],
                                       card_dict['email']))

        # TODO 针对找到的名片记录执行修改和删除的操作
        deal_card(card_dict)
        break
    else:
        print('抱歉，没有找到 %s' % find_name)

def deal_card(find_dict):
    print(find_dict)

```

219. 处理名片-01-增加分支判断用户操作类型

```

# 记录所有的名片字典
card_list = []

def show_menu():
    """显示菜单

    :return: None
    """
    print('*' * 50)
    print('欢迎使用【名片管理系统】V 1.0')
    print("")
    print("1. 新增名片")
    print("2. 显示全部")
    print("3. 搜索名片")
    print("")
    print('0. 退出系统')
    print('*' * 50)

def new_card():
    """ 新增名片

    :return: None

```



```

def search_card():
    """ 搜索名片

    :return: None
    """
    print('-' * 50)
    print('搜索名片')

    # 1. 提示用户输入要搜索的姓名
    find_name = input('请输入要搜索的姓名: ')

    # 2. 遍历名片列表, 查询要搜索的姓名, 如果没有找到, 需要提示用户
    for card_dict in card_list:
        if card_dict['name'] == find_name:
            print('姓名\t\t电话\t\tQQ\t\t邮箱')
            print('=' * 50)
            print('%s\t\t%s\t\t%s\t\t%s' % (card_dict['name'],
                                           card_dict['phone'],
                                           card_dict['qq'],
                                           card_dict['email']))

            # TODO 针对找到的名片记录执行修改和删除的操作
            deal_card(card_dict)
            break
        else:
            print('抱歉, 没有找到 %s' % find_name)

def deal_card(find_dict):
    print(find_dict)
    action_str = input('请选择要执行的操作 '
                      '[1] 修改 [2] 删除 [0] 返回上级菜单')
    if action_str == '1':
        print('修改名片')
    elif action_str == '2':
        print('删除名片')

```

注意：如果字符串太长，并且该字符串在一对小括号内部，可以直接回车
 PyCharm 会把一个字符串拆分成两个字符串

220. 处理名片-02-删除名片

```

# 记录所有的名片字典
card_list = []

```

```

def show_menu():
    """显示菜单

    :return: None
    """
    print('*' * 50)
    print('欢迎使用【名片管理系统】V 1.0')
    print("")
    print("1. 新增名片")
    print("2. 显示全部")
    print("3. 搜索名片")
    print('')
    print('0. 退出系统')
    print('*' * 50)


def new_card():
    """ 新增名片

    :return: None
    """
    print('-' * 50)
    print('新增名片')

    # 1. 提示用户输入名片的详细信息
    name_str = input('请输入姓名: ')
    phone_str = input('请输入电话: ')
    qq_str = input('请输入QQ: ')
    email_str = input('请输入邮箱: ')
    # 2. 使用用户输入的信息建立一个名片字典
    card_dict = {'name': name_str,
                  'phone': phone_str,
                  'qq': qq_str,
                  'email': email_str}
    # 3. 将名片字典添加到列表中
    card_list.append(card_dict)
    print(card_list)
    # 4. 提示用户添加成功
    print('添加 %s 的名片成功' % name_str)


def show_all():
    """ 显示所有名片

    :return: None
    """
    print('-' * 50)
    print('显示所有名片')

    # 判断是否存在名片记录，如果没有，提示用户并且返回

```

```

if len(card_list) == 0:
    print('当前没有任何的名片记录，请使用新增名片功能添加名片！')
    # return 可以返回一个函数的执行结果
    # 下方的代码不会被执行
    # 如果 return 后面没有任何内容，表示会返回到调用函数的位置
    # 并且不返回任何的结果（实际上返回 None）
    return

# 打印表头
for name in ['姓名', '电话', 'QQ', '邮箱']:
    print(name, end='\t\t')
print('')
# 打印分隔线
print('=' * 50)

# 遍历名片列表依次输出字典信息
for card_dict in card_list:
    print('%s\t%s\t%s\t%s' % (card_dict['name'],
                              card_dict['phone'],
                              card_dict['qq'],
                              card_dict['email']))

def search_card():
    """ 搜索名片

    :return: None
    """
    print('-' * 50)
    print('搜索名片')

    # 1. 提示用户输入要搜索的姓名
    find_name = input('请输入要搜索的姓名: ')

    # 2. 遍历名片列表，查询要搜索的姓名，如果没有找到，需要提示用户
    for card_dict in card_list:
        if card_dict['name'] == find_name:
            print('姓名\t电话\tQQ\t邮箱')
            print('=' * 50)
            print('%s\t%s\t%s\t%s' % (card_dict['name'],
                                      card_dict['phone'],
                                      card_dict['qq'],
                                      card_dict['email']))

            # TODO 针对找到的名片记录执行修改和删除的操作
            deal_card(card_dict)
            break
        else:
            print('抱歉，没有找到 %s' % find_name)

def deal_card(find_dict):
    print(find_dict)

```



```
action_str = input('请选择要执行的操作 '
                  '[1] 修改 [2] 删除 [0] 返回上级菜单')
if action_str == '1':
    print('修改名片')
elif action_str == '2':
    card_list.remove(find_dict)
    print('删除名片成功!')
```

221. 处理名片-03-修改名片

```
def deal_card(find_dict):
    print(find_dict)
    action_str = input('请选择要执行的操作 '
                      '[1] 修改 [2] 删除 [0] 返回上级菜单')
    if action_str == '1':
        find_dict['name'] = input('姓名: ')
        find_dict['phone'] = input('电话: ')
        find_dict['qq'] = input('QQ: ')
        find_dict['email'] = input('邮箱: ')
        print('修改名片成功!')
    elif action_str == '2':
        card_list.remove(find_dict)
        print('删除名片成功!')
```

222. 处理名片-04-明确细化修改名片的思路，准备新的输入函数

```
def deal_card(find_dict):
    print(find_dict)
    action_str = input('请选择要执行的操作 '
                      '[1] 修改 [2] 删除 [0] 返回上级菜单')
    if action_str == '1':
        find_dict['name'] = input('姓名: ')
        find_dict['phone'] = input('电话: ')
        find_dict['qq'] = input('QQ: ')
        find_dict['email'] = input('邮箱: ')
        print('修改名片成功!')
    elif action_str == '2':
        card_list.remove(find_dict)
        print('删除名片成功!')
```

```
def input_card_info(dict_value, tip_message):  
    # 1. 提示用户输入内容  
  
    # 2. 针对用户的输入进行判断，如果用户输入了内容，直接返回结果  
  
    # 3. 如果用户没有输入内容，返回字典中原有的值  
    pass
```

224. 处理名片-06-增加文档注释、删除TODO标记

当函数开发完成并且通过测试，需要加入文档注释。

快速添加文档注释：`Alt + Enter`，选择第二个

整体移动代码，选中代码，点菜单栏 `Code`，然后 `Move Line Up`

```
# 记录所有的名片字典  
card_list = []  
  
def show_menu():  
    """显示菜单  
  
    :return: None  
    """  
    print('*' * 50)  
    print('欢迎使用【名片管理系统】V 1.0')  
    print("")  
    print("1. 新增名片")  
    print("2. 显示全部")  
    print("3. 搜索名片")  
    print('')  
    print('0. 退出系统')  
    print('*' * 50)  
  
def new_card():  
    """ 新增名片  
  
    :return: None  
    """  
    print('-' * 50)
```

```

print('新增名片')

# 1. 提示用户输入名片的详细信息
name_str = input('请输入姓名: ')
phone_str = input('请输入电话: ')
qq_str = input('请输入QQ: ')
email_str = input('请输入邮箱: ')
# 2. 使用用户输入的信息建立一个名片字典
card_dict = {'name': name_str,
             'phone': phone_str,
             'qq': qq_str,
             'email': email_str}
# 3. 将名片字典添加到列表中
card_list.append(card_dict)
print(card_list)
# 4. 提示用户添加成功
print('添加 %s 的名片成功' % name_str)

```

```

def show_all():
    """ 显示所有名片

    :return: None
    """
    print('-' * 50)
    print('显示所有名片')

    # 判断是否存在名片记录，如果没有，提示用户并且返回
    if len(card_list) == 0:
        print('当前没有任何的名片记录，请使用新增名片功能添加名片！')
        # return 可以返回一个函数的执行结果
        # 下方的代码不会被执行
        # 如果 return 后面没有任何内容，表示会返回到调用函数的位置
        # 并且不返回任何的结果（实际上返回 None）
        return

    # 打印表头
    for name in ['姓名', '电话', 'QQ', '邮箱']:
        print(name, end='\t\t')
    print('')
    # 打印分隔线
    print('=' * 50)

    # 遍历名片列表依次输出字典信息
    for card_dict in card_list:
        print('%s\t%s\t%s\t%s' % (card_dict['name'],
                                card_dict['phone'],
                                card_dict['qq'],
                                card_dict['email']))

```

```

def search_card():

```

```

""" 搜索名片

:return: None
"""

print('-' * 50)
print('搜索名片')

# 1. 提示用户输入要搜索的姓名
find_name = input('请输入要搜索的姓名: ')

# 2. 遍历名片列表, 查询要搜索的姓名, 如果没有找到, 需要提示用户
for card_dict in card_list:
    if card_dict['name'] == find_name:
        print('姓名\t\t电话\t\tQQ\t\t邮箱')
        print('=' * 50)
        print('%s\t\t%s\t\t%s\t\t%s' % (card_dict['name'],
                                       card_dict['phone'],
                                       card_dict['qq'],
                                       card_dict['email']))
        # 针对找到的名片记录执行修改和删除的操作
        deal_card(card_dict)
        break
    else:
        print('抱歉, 没有找到 %s' % find_name)

def deal_card(find_dict):
    """处理查找到的名片

    :param find_dict: 查找到的名片
    """
    print(find_dict)
    action_str = input('请选择要执行的操作 '
                      '[1] 修改 [2] 删除 [0] 返回上级菜单')
    if action_str == '1':
        find_dict['name'] = input_card_info(find_dict['name'], '姓名: ')
        find_dict['phone'] = input_card_info(find_dict['phone'], '电话: ')
        find_dict['qq'] = input_card_info(find_dict['qq'], 'QQ: ')
        find_dict['email'] = input_card_info(find_dict['email'], '邮箱: ')
        print('修改名片成功! ')
    elif action_str == '2':
        card_list.remove(find_dict)
        print('删除名片成功! ')

def input_card_info(dict_value, tip_message):
    """输入名片信息

    :param dict_value: 字典中原有的值
    :param tip_message: 输入的提示文字
    :return: 如果用户输入了内容, 就返回内容, 否则返回字典中原有的值
    """

```

```
# 1. 提示用户输入内容
result_str = input(tip_message)
# 2. 针对用户的输入进行判断, 如果用户输入了内容, 直接返回结果
if len(result_str) > 0:
    return result_str
# 3. 如果用户没有输入内容, 返回字典中原有的值
else:
    return dict_value
```

225. 运行程序-01-增加Shebang符号直接运行Python程序

#!

- **#!** 这个符号叫做 **Shebang** 或者 **Sha-bang**
- **Shebang** 通常在 **Unix** 系统脚本的中 **第一行开头** 使用
- 指明 **执行这个脚本文件** 的 **解释程序**

使用 Shebang 的步骤

1. 使用 **which** 查询 **python3** 解释器所在路径

```
$ which python3
```

2. 修改要运行的 **主 python 文件**, 在第一行增加以下内容

```
#!/usr/bin/python3
```

3. 修改 **主 python 文件** 的文件权限, 增加执行权限

```
$ chmod +x cards_main.py
```

4. 在需要时执行程序即可

```
./cards_main.py
```

完成于 201809302344