python基础10_包和PEP8代码规范和案例

一、包

1.1 包的作用

- 将有关联的模块文件组织起来的文件夹，这个文件夹中有一个`__init__`.py.
- 有效避免模块名称冲突问题，提高程序的结构性和可维护性

1.2. 包的创建

- 右键 -> new -> python package -> 包名
- 右键 -> new -> python file -> 模块名

1.3. 使用 `import 包名.模块名` 能够导入包中的模块

```
"""
导入格式:      import 包名.模块名
                包名就是文件夹名      模块名就是文件名字
使用格式:      包名.模块名.工具    (类名、函数、变量)
"""
import msg.sendmsg
msg.sendmsg.send_test()
```

1.3 使用 `from 包名.模块名 import...`能够导入模块中的符号

```
"""
导入格式:      from 包名.模块名 import 所需的工具
使用格式:      工具    (类名、函数、变量)
"""
from msg.sendmsg import send_test
# from msg.sendmsg import *  # __all__列举的标识符
send_test()
```

1.4 包中`__init__.py`文件的作用

- 包被导入时，会执行 `__init__.py`文件的内容
- init文件作用: 控制包的**导入行为，管理模块文件**

- `__init__.py:`

```
print("__init__文件被执行了, 负责包中模块管理")

# 导入模块
# 方法1:
# from msg import recvmsg
# from msg import sendmsg

# 方法2:
# 下面的.代表当前文件夹: 既是msg
# from . import recvmsg
# from . import sendmsg

# 导入模块中的工具:
from msg.recvmsg import recv_msg
from msg.sendmsg import send_msg
```

- 代码

```
# 1. 导入包, 自动执行__init__.py文件的内容

"""
导包格式: import 包名
使用格式: 包名.模块.工具
"""
```

```
import msg

# __init__.py中导入模块的使用方法:
# msg.sendmsg.send_msg()
# msg.recvmsg.recv_msg()

# __init__.py中导入工具的使用方法:
msg.send_msg()
msg.recv_msg()
```

二、【了解】PEP8代码规范

- 自动格式化快捷键: ctrl + alt + L
- 文档地址：https://www.python.org/dev/peps/pep-0008/
- 中文文档：http://zh-google-styleguide.readthedocs.io/en/latest/google-python-styleguide/python_style_rules/

三、【应用】学生名片管理系统-面向对象版

3.1. 需求分析-抽象学生类-增删改查

```
"""
1. 以前存储
    user_list = [{'name':'mike', 'age':34, 'tel':'110'},
                 {'name':'yoyo', 'age':24, 'tel':'120'}]

2. user_list = [对象1, 对象2]
    2.1 对象类设计, 学生类: 属性有name, age, tel
"""


class Student(object):
    def __init__(self, _name, _age, _tel):
        # 添加属性,并赋予初始值
        self.name = _name
        self.age = _age
        self.tel = _tel

    def __str__(self):
```

```
        # 返回对象属性信息字符串
        return f"我叫{self.name}, 我今年{self.age}岁, 我的电话时{self.tel}"


"""
增加学生:
# 1. 定义空列表
# 2. 创建对象
# 3. 列表追加对象
"""
user_list = []
name = input("请输入姓名: ")
age = int(input("请输入年龄: "))
tel = input("请输入电话: ")

# 使用类创建对象 对象 = 类名(形参, ....)
stu = Student(name, age, tel)
user_list.append(stu)

stu = Student("tom", 20, "123")
user_list.append(stu)

stu = Student("mike", 21, "1234")
user_list.append(stu)
```

```
# 如果列表中是对象，直接打印列表，打印的是对象的地址，不能打印对象的属性
# [<__main__.Student object at 0x000001EE4B7E4910>]
# print(user_list)

"""
遍历打印信息：
# 1. 遍历从列表取出对象
# 2. 打印的是对象.属性
"""
for user_obj in user_list:
    print(user_obj)

"""
查询某人：
# 1. 遍历从列表取出对象
# 2. 判断对象.name和查找的用户是否相等
# 3. 如果相等，打印对象的相应属性信息，然后break
# 4. for的else，提示没有此人
"""
search_name = input("请输入需要查询的学生姓名：")
for user_obj in user_list:
    if user_obj.name == search_name:
        print("查询到学生信息：", user_obj)
```

```
            break
    else:
        print("查无此人")


"""
修改某人:
# 1. 通过enumerate(user_list)带序号遍历: 索引、对象
# 2. 判断对象.name和修改的用户是否相等
# 3. 修改内容, 然后break
# 4. for的else, 提示没有此人
"""
update_name = input("请输入需要修改的学生姓名: ")

for user_obj in user_list:
    if user_obj.name == update_name:
        user_obj.name = input("请输入新的姓名: ")
        user_obj.age = input("请输入新的年龄: ")
        user_obj.tel = input("请输入新的电话: ")
        break
else:
```

```
    print("查无此人")

for user_obj in user_list:
    print(user_obj)


"""
删除某人:
# 1. 通过enumerate(user_list)带序号遍历: 索引、对象
# 2. 判断对象.name和删除的用户是否相等
# 3. 通过 del 列表[索引] 删除元素, 然后break
# 4. for的else, 提示没有此人
"""
del_name = input("请输入需要删除的学生姓名: ")
for i, user_obj in enumerate(user_list):
    if user_obj.name == del_name:
        # del user_obj  # 这样删除,只是删除了user_obj这个引用,不会删除user_list[i]
        del user_list[i]
        break
else:
    print("查无此人")

for user_obj in user_list:
    print(user_obj)
```

3.2. 需求-文件处理需求

```
"""
1. 以前存储
    user_list = [{'name':'mike', 'age':34, 'tel':'110'},
                 {'name':'yoyo', 'age':24, 'tel':'120'}]

2. user_list = [对象1, 对象2]
    2.1 对象类设计, 学生类: 属性有name, age, tel


"""


# 自定义一个学生类
class Student(object):
    def __init__(self, _name, _age, _tel):
        self.name = _name
        self.age = _age
        self.tel = _tel
```

```
    def __str__(self):
        return f"我叫{self.name}, 我今年{self.age}岁, 我的电话时{self.tel}"


"""
增加学生:
# 1. 定义空列表
# 2. 创建对象
# 3. 列表追加对象
"""
# user_list = [
#     Student("rose", 12, "123"),
#     Student("tom", 21, "122"),
#     Student("mike", 22, "333")
# ]

stu1 = Student("rose", 12, "123")
stu2 = Student("tom", 21, "122")
stu3 = Student("mike", 22, "333")
user_list = [
    stu1, stu2, stu3
]
```

```
"""
遍历打印信息：
# 1. 遍历从列表取出对象
# 2. 打印的是对象.属性
"""
print("添加学生信息成功：")
for user_obj in user_list:
    print(user_obj)


"""
写文件：
# 1. 将[对象1， 对象2] 转换为 [{}, {}]
# 2. str([{}, {}])转换后，写入文件
"""
save_list = []
for user_obj in user_list:
    save_dict = {"name": user_obj.name,
"age": user_obj.age, "tel": user_obj.tel}
    save_list.append(save_dict)

with open("stu.txt", "w") as file:
    file.write(str(save_list))

"""
读文件：
```

```
# 1. 从文件读出来的内容是字符串，通过eval转换为[{}, {}]
# 2. 从列表中取出字典，再取出字典的元素，这个元素用新建对象
# 3. 列表追加对象
"""
with open("stu.txt", "r") as file:
    file_data = file.read()
    # eval: 看参数像什么,就转换成什么, 但是不支持自定义类型的转换
    load_list = eval(file_data) # [{},{},{}]

user_list = []
# [{},{},{}] --> [对象, 对象, 对象]
for user_dict in load_list:
    # 使用Student类创建对象user_obj
    user_obj = Student(user_dict['name'], user_dict['age'], user_dict['tel'])
    user_list.append(user_obj)

print("加载文件信息成功: ")
for user_obj in user_list:
    print(user_obj)
```

3.3. 学生类设计

```
class Student(object):

    def __init__(self, _name, _age, _tel):
        self.name = _name
        self.age = _age
        self.tel = _tel

    def __str__(self):
        return f"
{self.name}\t\t{self.age}\t\t{self.tel}"

    def to_dict(self):
        return {"name": self.name, "age":
self.age, "tel": self.tel}
```

3.4. 学生管理系统类设计

```
"""
学生类Student：保存学生的基本信息
1. __init__添加属性name, age, tel
```

```
2. __str__返回实例属性信息
3. to_dict，将属性内容以字典的形势返回

学生管理类ManagerStuSys，管理学生：增删改查学生信息
1. __init__添加属性，专门保存学生对象，属性是列表
2. 设计方法，测试功能使用
3. 类的外面，创建学生管理对象，调用测试功能方法
"""


class Student(object):
    def __init__(self, _name, _age, _tel):
        self.name = _name
        self.age = _age
        self.tel = _tel

    def __str__(self):
        return f"
{self.name}\t\t{self.age}\t\t{self.tel}"

    def to_dict(self):
        return {"name": self.name, "age":
self.age, "tel": self.tel}
```

```
class ManagerStuSys(object):
    def __init__(self):
        self.user_list = []

    def test(self):
        stu = Student("rose", 12, "123")
        self.user_list.append(stu)
        stu = Student("yoyo", 22, "223")
        self.user_list.append(stu)

        for user_obj in self.user_list:
            print(user_obj)
            print(user_obj.to_dict())


ms = ManagerStuSys()
ms.test()
```

3.4. 主逻辑

```
class Student(object):
    def __init__(self, _name, _age, _tel):
        self.name = _name
        self.age = _age
```

```
        self.tel = _tel

    def __str__(self):
        return f"
{self.name}\t\t{self.age}\t\t{self.tel}"

    def to_dict(self):
        return {"name": self.name, "age":
self.age, "tel": self.tel}


class ManagerStuSys(object):
    def __init__(self):
        self.user_list = []

    # 定义show_menu函数为静态方法：因为该函数不
需要使用实例属性和类属性
    @staticmethod
    def show_menu():
        print('=' * 20)
        print('= 1. 添加学生')
        print('= 2. 查询所有学生')
        print('= 3. 查询某个学生')
        print('= 4. 修改某个学生')
        print('= 5. 删除某个学生')
```

```
        print('= 6. 保存信息')  # 1.1 菜单增加保存信息的提示
        print('= 7. 退出系统')
        print('=' * 20)

    def start(self):
        # 2.1 程序启动时，循环的前面，调用加载数据函数
        # load_info()

        # 1. 死循环
        while True:
            # 调用菜单
            self.show_menu()

            # 2. 用户输入数字
            cmd = input("请输入功能数字: ")

            # 3. 条件选择
            if cmd == "1":
                print('添加学生')
                # add_stu_info()
                # print(user_list) # 打印列表，做测试，看数据
            elif cmd == "2":
                print('查询所有学生')
```

```
                # show_all_stu()
            elif cmd == "3":
                print('查询某个学生')
                # show_one_stu()
            elif cmd == "4":
                print('修改某个学生')
                # update_stu_by_name()
            elif cmd == "5":
                print('删除某个学生')
                # del_stu_by_name()
            elif cmd == '6':  # 1.2 主逻辑增加 名片信息保存到文件 的选择
                print("保存学生信息")
                # save_info()
            elif cmd == "7":
                print('退出循环')
                break
            else:
                print('输入有误，请重新输入')


ms = ManagerStuSys()
ms.start()
```

3.5. 学生信息添加-功能

```
        # 定义新建学生的函数
    def add_stu_info(self):
        """添加学生信息"""
        # 1. 输入用户信息：姓名、年龄、电话
        _name = input('请输入学生姓名：')
        _age = int(input('请输入学生年龄：'))
 # 年龄应该是整型，所有做了int转换
        _tel = input('请输入学生电话：')

        # 2. 通过for遍历，取出某个元素后，这个元素就是对象
        for user_obj in self.user_list:
            # 2.1 对象.name == 用户输入的名字，是否相等，相等则跳出循环
            if user_obj.name == _name:
                print('此用户已经存在，请重来')
                # 2.2 break跳出循环
                break
        else:
            # 3. for中的else 如果用户不存在列表中，添加用户对象到列表
```

```
            # 3.1 创建对象
            new_obj = Student(_name, _age,
_tel)
            # 3.2 追加列表
            # user_list是可变类型，没有重新赋
值，没有改变原来地址，所以不用global声明
            self.user_list.append(new_obj)
```

3.6. 打印所有学生信息-功能

```
    # 显示所有的学生，带序号的
    def show_all_stu(self):
        """显示所有的学生"""
        # 1. 遍历前，打印一些提示信息：序号
姓名   年龄   电话
        # \t一个tab键的空格
        print('序号\t 姓名\t\t年龄\t\t电话')

        # 2. 遍历 for 索引位置，对象 in
enumerate(user_list)：
        for i, user_obj in
enumerate(self.user_list):
            # 2.1 打印一个用户的信息 索引位置
+1，user_dict[‘name’]……
            # print('%d\t\t%s\t\t%d\t\t%s'
% (i + 1, user_obj.name, user_obj.age,
user_obj.tel))
            print(f"{i+1}\t", user_obj)
```

3.7. 查询某一个学生-功能

```
    # 显示某个学生
    def show_one_stu(self):
```

```
        """显示某个学生"""
        # 1. 输入姓名
        _name = input('请输入学生姓名: ')

        # 2. 通过for遍历，取出一个对象user_obj
        for user_obj in self.user_list:
            # 2.1 user_obj.name和输入姓名判断
            if user_obj.name == _name:
                # 2.1.1 如果相等，输出用户信息，退出循环
                print('查询到的用户信息如下: ')
                print(user_obj)
                break
        # 3. for中的else，循环执行完毕，没有break，说明用户不存在，提示一下
        else:
            print('查询的用户不存在')
```

3.8. 修改某一个学生-功能

```
    def update_stu_by_name(self):
        """更新某个学生信息，根据输入的姓名匹配哪个学生"""
```

```
        # 1. 输入需要修改的用户姓名
        update_name = input('输入需要修改的用户姓名: ')

        # 2. for遍历, 带索引的遍历    i, user_obj  in user_list
        for i, user_obj in enumerate(self.user_list):
            # 2.1 如果user_obj.name和输入用户名字相等
            if user_obj.name == update_name:
                # 2.1.1 重新输入新的姓名、年龄、电话
                _name = input('请输入新的学生姓名: ')
                _age = int(input('请输入新的学生年龄: '))
                _tel = input('请输入新的学生电话: ')
                # 2.1.2 对user_list[i].属性 = 新的name
                # self.user_list[i].name = _name
                # self.user_list[i].age = _age
```

```
                # self.user_list[i].tel =
_tel

                user_obj.name = _name
                user_obj.age = _age
                user_obj.tel = _tel

                # 2.1.3 ……、修改成功打印、
break跳出循环
                print('修改成功')
                break
        # 3. for中的else 输入的用户不在列表
        else:
            print('输入的用户不在列表，请重新输
入')
```

3.9. 删除某一个学生-功能

```
    def del_stu_by_name(self):
        """删除某个学生，根据输入的姓名"""
        # 1. 输入用户姓名
        _name = input('请输入需要删除的名
字: ')
```

```
        # 2. for遍历，带索引的遍历   i, user_obj
        for i, user_obj in enumerate(self.user_list):
            # 2.1 如果user_obj.name和输入用户名字相等
            if user_obj.name == _name:
                # 2.1.1 del 列表[i], 同时break跳出循环
                del self.user_list[i]
                print('删除成功')
                break
        # 3. for中else 输入的用户在列表中，不存在
        else:
            print('用户不在列表中，无法删除')
```

3.10. 保存学生信息到文件-功能

```
    # 1.3 名片信息保存到文件 函数的设计
    def save_info(self):
        """名片信息保存到文件"""
        # 1. 转换[对象1, 对象2, ...] 成为 [{字典}, {字典}, ...]
        temp_list = []
        for user_obj in self.user_list:

 temp_list.append(user_obj.to_dict())

        # 2. 通过with打开文件，自动处理关闭
        with open('stu.txt', 'w') as f:
            # 2.1 将列表转换为字符串后，再往文件中写
            f.write(str(temp_list))
```

3.11. 加载文件信息到学生列表中-功能

```
    def load_info(self):
        """加载数据"""
        # 1.第一种方式判断文件是否存在:
        if os.path.exists('stu.txt'):
            # 2.3 文件存在的时候，加载数据
```

```
            # 2.4 通过with打开文件，只读方式打开文件
            with open('stu.txt', 'r') as f:
                # 2.5 文件变量.read()读取所有内容，返回内容是字符串类型
                content = f.read()
                if content:
                    # 2.7 把读取内容通过eval转换成列表，给全局变量的列表赋值
                    temp_list = eval(content)
                    print('数据成功加载')

                    # 3. 把[字典，字典，...]转换成 [对象，对象，...]
                    for user_dict in temp_list:
                        stu = Student(user_dict['name'], user_dict['age'], user_dict['tel'])

 self.user_list.append(stu)
        else:
            print('文件不存在，无法加载')
```

3.12. 使用捕获异常的方式来判断文件是否存在

```
    def load_info(self):
        """加载数据"""
        # 2. 使用捕获异常的方式来判断文件是否存在
        try:
            # 2.3 文件存在的时候，加载数据
            # 2.4 通过with打开文件，只读方式打开文件
            with open('stu.txt', 'r') as f:
                # 2.5 文件变量.read()读取所有内容，返回内容是字符串类型
                content = f.read()
                if content:
                    # 2.7 把读取内容通过eval转换成列表，给全局变量的列表赋值
                    temp_list = eval(content)

                    # 3. 把[字典, 字典, ...]转换成 [对象, 对象, ...]
                    for user_dict in temp_list:
```

```
                                stu =
Student(user_dict['name'],
user_dict['age'], user_dict['tel'])

 self.user_list.append(stu)
        except FileNotFoundError as e:
            print('文件不存在，无法加载')
        else:
            print('数据成功加载')
```