随堂笔记\_LESSON01\_包、模块

产品名称：牟文斌

联系方式：15682808270

QQ：1007821300

**课程模式**

上午

1. 当天课程安排
2. 课前分享(技术相关) + 1~3个人的作业讲解
   1. 自我介绍[介绍自己的 优点 1分钟]
3. 课程内容-知识点回顾
4. 新课内容-(1,2,3,4,5...)
5. 课间练习+思考

下午

1. 颈椎操
2. 课前分享(可以和技术无关)
3. 新课内容{中午练习项目-讲解}
4. 课间练习+思考
5. 当然的课程总结
6. 晚自习任务

课程资料

1. 随堂笔记.docx (WPS|MicrosoftOffice)-> PDF
2. 项目开发过程中，需要哪些文档？
   1. **需求分析文档**{描述用户**需**要的功能，和要**求**的限制}
      1. 产品经理、项目经理、开发总监、开发组长、开发人员
   2. **详细设计文档**{描述软件整体架构的文档}
      1. 技术总监**(架构师)**[编写]->产品经理+项目经理-审核
         1. 技术总监->详细设计[技术细节实现]
         2. 项目经理->任务分配->开发组长
         3. 任务细化-> 开发人员
      2. 工作经验【技术经验、行业经验】
         1. **技术经验：会哪些技术[广度、深度]、解决过的问题**
         2. 行业经验：对现实生活中某个行业的业务的熟悉度[医疗、银行、通信..]
      3. 单元测试[开发人员操作]
         1. 开发完一个功能，测试功能的可用性和正确性
         2. 单元测试过程-> 附注在详细设计文档中[对应的章节下]

|  |
| --- |
| num1 = input(“enter number :”)  num2 = input(“enter number2:”)  print(“please waiting, calculations..”)  res = num1 + num2  print(res) |
| 可用吗？是否会报错？~不会报错  正确吗？业务上~不正确 |
| 1  1  11 |

* 1. 详细测试文档【测试工程师】
     1. 测试的分类
        1. 白盒测试
           1. 单元测试【开发人员、测试人员】
           2. 集成测试【测试人员】
           3. 系统测试【测试人员】
        2. 黑盒测试
           1. 完全不熟悉代码、不熟悉底层操作的业务人员进行测试
     2. 提交系统备案
  2. 部署运维手册
     1. 开发人员+测试人员
     2. 软件使用说明书
  3. 软件交付
  4. 软件运维阶段
  5. 软件下线

100%

10W

需求调研-> 需求分析-> 详细设计-> 代码开发-> 详细测试-> 部署上线-> 运维-> 下线

[ 30% ] [ 15% ][ %5 ] [ %5 ] [ 45% ]

[ 2W ][ 8W ]

# 主要内容

1. 包和模块
   1. 包和模块的引入方式
   2. 绝对引入方式
   3. 相对引入方式
   4. 包和模块的定义
   5. 自定义包和模块
   6. 包的发布
      1. 本地发布
      2. 网络发布
   7. python中的main方法
      1. \_\_name\_\_ == “\_\_main\_\_”

# 课程内容

1. 包和模块-概述(Overview)

一般情况下，一个成熟的、企业级的软件，代码量都会很多！

一般情况下，都是由多人协同开发完成的！

在一个项目中，就会存在大量的python文件，文件中包含大量的python代码

**问题：按照什么样的标准，划分不同的文件？**

**按照什么样的标准，将不同的代码存放在同一个文件中？**

**飞机大战**

当一个需求功能逐渐增加的时候，一个文件中的代码就会无限制增加

要修改一行代码-> 直接影响了后面几十行~几百行代码的的正确性。

要修改一个功能添加一行代码-> 找不到在什么位置应该添加[嵌套太多]

功能的扩展性太差

代码的维护性太差

* + - 代码 耦合性 太高 --> 解耦合:降低代码之间的耦合度
    - 耦合：依赖

将不同功能的代码，存放在不同的文件中，让每个python文件中的代码较少，多个文件协同工作

模块化开发：将不同功能的代码，按照功能要求划分到不同的文件中

main.py：程序的入口，运行的地方 python main.py

utils.py：飞机大战工具文件，存放各种工具类函数，如事件函数、计时函数、计分函数等等

engine.py：飞机大战流程定义文件，这个文件中定义的游戏的启动、暂停、运行、异常处理等等函数

在同一个文件中，可能会存在大量代码，将不同功能的代码，封装成对应的变量、函数、类型或者方法，方便互相调用执行。

原始代码-> 解耦合 -> 函数-> 解耦合-> 多个文件[互相引入](协同开发)

划分标准：**功能**

1. PYTHON中的包和模块

为了方便整理PYTHON代码

一个成熟的PYTHON项目中，存在多个PYTHON文件

多个PYTHON文件-功能相似的-存放在一个文件夹中

一个PYTHON文件中，会存在多行功能相关的代码

PYTHON中~存放多个PYTHON文件的文件夹~ 定义成 包{package}

文件夹名称：就是包名称

PYTHON中的文件~ 文件 就是一个 模块{module}

文件名称：就是模块名称

包的定义：

默认情况下，PYTHON包~就是一个文件夹

默认包~就是一个普通文件夹[使用较少]

标准包~文件夹中有一个特殊的文件[包声明文件：\_\_init\_\_.py]

模块的定义：

**一个PYTHON模块**，就是一个python 源代码 文件

**.py** python源代码文件

.pyc python可执行二进制文件

模块和包的区别：

包：文件夹，可以包含多个文件

模块：文件~可以包含变量、函数、类型..代码块

两周周期

1 包和模块

2-3 网络编程

4-5 并发编程

6~10 高级编程[函数高级、推导式、迭代器、生成器、装饰器、元类...]

包和模块

**包**：python中一个文件夹，包含了一个包声明模块\_\_init\_\_.py的文件夹，可以统一组织和管理多个python模块

**模块**：python的一个源代码文件就是python模块，模块名称就是文件名称

**包和模块的区别**：包是一个文件夹，可以包含多个文件/模块；模块是一个文件主要包含python代码块(变量、函数、类型、....)

**为什么要有包和模块**：解耦合~通过多个文件协同操作，通过import关键字可以导入模块之外的其他模块，达到代码的文件级别的重复使用。

**解耦合**目标：提高项目功能的扩展性、代码的可维护性。

包和模块的引入

主要通过引入的方式，完成多个模块之间互相协同开发的目标

重点/核心：引入[import]

相对引入[包含了相对路径(./..)]

. : 表示当前文件夹/路径/目录

.. : 表示上一级文件夹/路径/目录

绝对引入

针对包/项目，从根路径直接导入的方式

代码还原：通过代码的方式，熟悉引入的操作

引入模块的操作：[相对引入|绝对引入]

import utils[当前引入的模块可以**独立运行**]

从.site-packages/ PYTHONPATH sys.path **当前路径下** 查询是否有utils模块

备注：pycharm开发工具~不会针对当前路径进行自动搜索，可能会出现误报错的现象，只需要标记{Mark Directory as -> Sources Root}即可。

from . import utils

从**当前**路径下，查询指定的模块[相对于谁是当前路径~一般情况下都是相对于父级文件夹]

引入包的操作：【相对引入|绝对引入】

相对引入：更多的使用在公司内部项目开发时，当前模块只在当前项目中使用。

绝对引入：更多的使用在工具/公共模块开发时，他们要被多个不同的项目使用。

包是一个文件夹~ 引入包--> 相当于 引入包中的\_\_init\_\_.py模块

import modules --> import modules.\_\_init\_\_

包的通配符的操作

就是操作包的\_\_init\_\_.py包声明文件

可以安全的第三方模块

pymysql->第三方模块->提供两种安装方式

1. pip install pymysql
2. 下载pymysql压缩包-> 解压-> python setup.py install
3. pip: package index python
4. pip: python install package
5. **pip install <module\_name> 安装一个第三方模块**
6. **pip uninstall <module\_name> 卸载一个第三方模块**
7. **pip list 查看当前解释器中安装的所有第三方模块**

|  |
| --- |
| **通过pip uninstall 卸载一个模块，一般情况下通过pip install安装的模块肯定能卸载成功。但是通过python setup.py install 这样安装的模块有可能卸载失败~手工进入site-packages/文件夹中，直接删除要卸载模块对应的文件夹即可。** |

其他第三方模块，团队开发完成之后，提供了两种安装方式

1. 离线安装包
   1. 下载tar.gz压缩包，解压之后执行python setup.py install安装
2. 在线安装
   1. pip install <module\_name>

我们自定义python程序包，同样也可以发布使用上面两种方式，完成异地多人协同开发

image\_validation/ 提交到网络上，让其他人共同参与开发

提供两种安装方式

**离线安装包：可以访问对应的网站，下载离线安装包tar.gz**

**在线安装：pip install image\_validation**

自定义包的网络发布

1. 网络发布的目标地址：<https://pypi.org/>
   1. pypi.org-> python package index . orgnization
2. 注册一个账号[导航栏login登录/register注册]
   1. 注册
   2. 登录邮箱激活
   3. 登录账号，登录网站，查看项目
3. 本地安装twine 项目上传使用的模块
4. 本地打包自己开发的程序包
5. 通过twine uplaod \*\*.tar.gz上传自己的项目
6. 其他开发人员，可以通过离线包/pip安装你上传的项目

包 和 模块

package & modules

包：文件夹，可以包含很多python文件

标准包中，一定要存在一个包声明文件：\_\_init\_\_.py 代表了包本身

模块：python文件，文件名称就是模块名称

文件中可以包含变量、函数、类型等等各种代码

包和模块：代码的重复使用，降低耦合

不同的包/模块，可以被其他模块通过import方式引入使用，提高了代码功能的扩展性和代码的可维护性

import引入：相对引入|绝对引入

区别：相对引入使用了相对路径[.|..]

两种引入方式：

模块的相对引入和绝对引入

包的相对引入和绝对引入

两个使用场景：

内部项目开发：当前包只能被一个项目调用使用

公共模块开发：当前包可能会被多个项目调用使用

总结：import 引入 和 from .. import引入的区别

**自定义包**和模块

自定义包/模块操作步骤

自定义包 的 标准化构建 python setup.py build

自定义包的打包 python setup.py sdist

自定义包的网络发布 twine uplaod sdist/\*.gz

扩展内容：python中的main方法

对比其他语言，其他语言都有一个特殊的程序运行入口：main方法/main函数

python中的最小执行单元：代码块~不存在main方法/函数

为了程序的完善性~模拟了一个main方法的操作

python文件——>python模块-> 模块名称\_\_name\_\_

默认情况下，模块的名称\_\_name\_\_就是文件的名称

如果当前python文件是执行文件[python xxx.py]

-> \_\_name\_\_ = “\_\_main\_\_”

每个Python模块，都有一个魔法属性\_\_name\_\_，指代当前模块的名称

因为存在这样的问题：一个python模块，被其他模块import引入时，会默认执行被引入模块中的所有代码[可能会误执行测试代码造成问题！]

为了保证被引入模块的测试代码不至于在import时执行导致错误，一般情况下，每个模块中的测试执行代码，都需要包含在一个选择判断中[main函数/方法]

if \_\_name\_\_ == “\_\_main\_\_”:

当前模块作为运行模块 要执行的代码

1. 测试代码
2. 程序入口

面试题：能简单说说Python中的main函数/方法吗？

* python中没有所谓的main函数/方法
* python中为了保证测试代码的正确性和程序入口的唯一性，通过选择结构模拟了一个main函数/方法，主要是通过python模块的魔法属性\_\_name\_\_也就是当前模块的名称属性进行操作的，默认情况下名称属性就是当前文件模块的名称，但是如果作为运行模块的话该名称属性的值就会是\_\_main\_\_，所以通过\_\_name\_\_是否是\_\_main\_\_的判断完成了main函数的模拟
* 扩展加分：一般情况下我们在开发时，这个模拟main函数/方法中，会有两个作用，第一个作用用来编写测试代码；第二作用用来规范程序入口。