程序结构

目标

- 掌握模块基本概念
- 掌握模块的使用方法
- 掌握包的概念及应用
- 掌握模块、包的编写、制作、发布

1. 模块 Module

1.1 定义

Python 模块(Module)，是一个 Python 文件，以 .py 结尾，包含了 Python 对象定义和Python语句。

- 本质是一个Python程序源文件
- 包含了Python定义的类、函数、对象及语句

1.2 作用

- 实现代码重用
- 划分系统命名空间
- 实现共享数据或服务

1.3 导入

1、import

(1) 语法：

import 模块名

import 模块名 as 别名

(2) 作用：将模块整体导入到当前模块中

(3) 使用：模块名.成员

2、from import

(1) 语法：

from 模块名 import 成员名

from 模块名 import 成员名 as 别名

from 模块名 import *

(2) 作用：将模块内的成员导入到当前模块作用域中

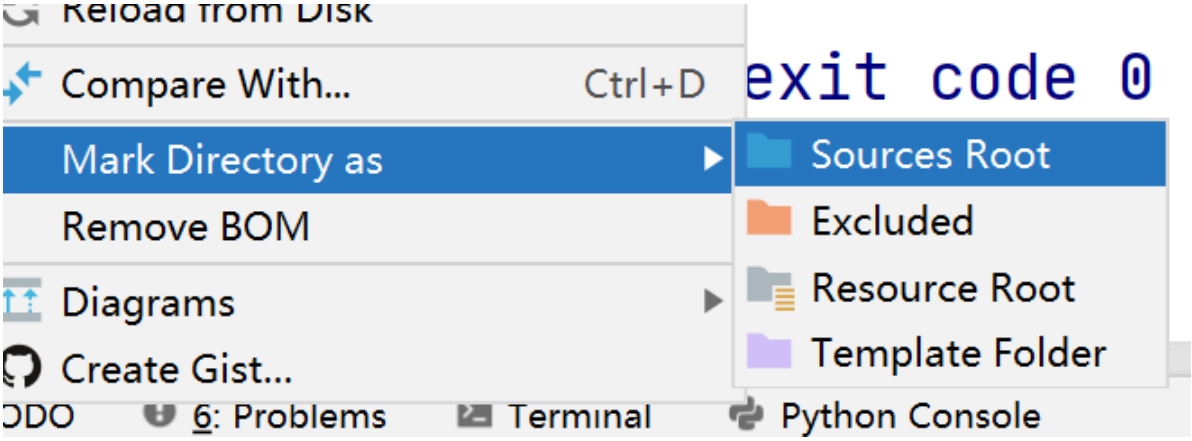
(3) 使用：直接使用成员名

```
"""
    module01.py
"""

def func01():
    print("module01 - func01执行喽")


def func02():
    print("module01 - func02执行喽")
```

```
# 导入方式1：import 模块名
# 使用：模块名.成员
# 原理：创建变量名记录文件地址,使用时通过变量名访问文件中成员
# 适用性：适合面向过程(全局变量、函数)
import module01

module01.func01()

# 导入方式2.1：from 文件名 import 成员
# 使用：直接使用成员
# 原理：将模块的成员加入到当前模块作用域中
# 注意：命名冲突
# 适用性：适合面向对象(类)

from module01 import func01

def func01():
    print("demo01 - func01")

func01() # 调用的是自己的func01


# 导入方式2.2：from 文件名 import *
from module01 import *

func01()
func02()
```

> 模块导入报错解决：
>
> 目录名右键-->Mark Directory as-->Sources Root
>
> ctrl+q：可以快速查看文档
>
> alt+enter：自动导模块

模块导入不进去的解决方法：



练习1：

创建2个模块module_exercise.py与exercise.py

将下列代码粘贴到module_exercise模块中，并在exercise中调用。

```
data = 100

def func01():
    print("func01执行喽")

class MyClass:
    def func02(self):
        print("func02执行喽")

    @classmethod
    def func03(cls):
        print("func03执行喽")
```

1.4 模块重载

- 模块被修改或者失效时需要重载
- 模块重载会重新执行模块代码，覆盖当前命名空间
- 模块顶层的赋值语句会覆盖变量原有的引用
- 用于实现应用程序的动态更新
- 通过reload()方法实现模块重载

> reload方法
>
> - 是一个内建函数
> - reload()函数将以前导入过的模块再加载一次
> - 在imp模块中实现，使用时需导入
> - 对已经导入的【模块对象】进行操作

1.5 分类

(1) 内置模块(builtins)，在解析器的内部可以直接使用。

(2) 标准库模块，安装Python时已安装且可直接使用。

(3) 第三方模块（通常为开源），需要自己安装。

(4) 自定义模块，用户自己编写的模块（可以作为其他人的第三方模块）

1.6 自定义模块

模块定义

在Python中，每个Python文件都可以作为一个模块，模块的名字就是文件的名字。也就是说自定义模块名必须要符合标识符命名规则。

模块的自定义有三种：

> 自定义模块与当前代码同目录
>
> 在Python代码中指定模块位置
>
> 通过环境变量指定，python模块所在位置

- 自定义模块与当前代码同目录

```python
# hello.py
# hello模块
def hello():
    return "hello"
```

```python
# 模块定义1
# 使用模块
from hello import hello
str1 = hello()
print(str1)
```

  - **优点**：不需要额外的配置，在同目录下，直接和使用内部模块一样 导入即可使用。
  - **缺点**：模块存在的意义在提高代码的复用性，使多个程序可以共享代码，如果每个模块都要单独放在相应的程序下，与出发点有点背道而驰，所以不建议大家使用此方法

- 在Python代码中指定模块位置

```python
# 在指定的目录下创建模块
# E:\standard\Python_core\module 下定义一个模块run.py

# run.py
def world():
    print("在Python代码中指定模块位置")

```

```python
# Demo中使用run模块中的world函数
import sys
sys.path.append("E:\standard\Python_core\module")
import run
run.world()
```

  - **优点**：公共模块可以统一管理，无需放在项目所在目录下，提高的复用性。
  - **缺点**：引用前，需制定模块所在位置。
- 通过环境变量指定，python模块所在位置

  把路径添加到环境变量中，直接可以调用

  - **优点**：代码复用，无需在代码中制定模块路径，无多余代码，和使用标准库一样
  - **缺点**：移植性差，不同的电脑需要配置环境变量

1.6.1 模块单元测试

目的：保证每个模块作为一个单元能正确运行

```python
# 自定义数学计算模块
# math_cal.py

pi = 3.1415926

def add(a,b): return a+b
def sub(a,b): return a-b
def mul(a,b): return a*b
def div(a,b):
    return a/b if b!=0 else 'ZeroDevisionError'

# 模块单元测试
if __name__ == "__main__":
    print(div(52,3))
```

1.6.2 数据隐藏

```python
# 自定义数学计算模块
# math_cal.py

# 针对[from x import * ]导入时，隐藏__all__之外的标识符
__all__ = [ 'add', 'sub', 'mul']

# 针对[from x import * ]导入时，隐藏_开头的标识符
_pi = 3.1415926

```

__all__

如果一个模块文件中有`__all__`变量，当使用`from xxx import *`导入时，只能导入这个列表中的元素。

1.7 time模块

```python
# 时间
# 人类时间：从公元元年      2023年9月10日 14:30:02
# 机器时间：从1970年元旦 到现在经过的秒数

import time
import locale
locale.setlocale(locale.LC_ALL, 'en')   # 设置了整个程序的地区(locale)为英语('en')
locale.setlocale(locale.LC_CTYPE, 'chinese')  # 将字符类型(locale category)设置为中文('chinese')

# 时间戳：表达机器时间
print(time.time())  # 1618986872.6075494
print(time.localtime())
print("************************************************************")

# 时间元组：表达人类时间
# 格式：(年,月,日,时,分,秒,星期,一年的第几天,与夏令时偏移)
tuple_time = time.localtime()
print(tuple_time[1])  # 获取月份
print(tuple_time[-3])  # 获取星期
print("************************************************************")

# 时间戳 --> 时间元组
print(time.localtime(1618986872.6075494))
# 时间元组 --> 时间戳
print(time.mktime(tuple_time))
print("************************************************************")

# 时间元组 --> str
# 语法：字符串 = time.strftime(格式,时间元组)
print(time.strftime("%y/%m/%d %H:%M:%S", tuple_time))
print(time.strftime("%Y年%m月%d日 %H:%M:%S", tuple_time))
print("************************************************************")

# str --> 时间元组
# 语法：时间元组 = time.strptime(字符串类型的时间,格式)
# 注意：时间与格式必须匹配
print(time.strptime("2021年04月21日 14:57:20", "%Y年%m月%d日 %H:%M:%S"))
```

练习1：定义函数，根据年月日，计算星期。

输入：2020 9 15

输出：星期二

练习2：定义函数，根据生日(年月日)，计算活了多天。

输入：2010 1 1

输出：从2010年1月1日到现在总共活了3910天

提示：

2. 包 package

为了更好地管理多个模块源文件，Python 提供了包的概念。

从物理上看，包就是一个文件夹，在该文件夹下包含了一个 `__init__.py` 文件，该文件夹可用于包含多个模块源文件；

从逻辑上看，包的本质依然是模块；

2.1 概念

- 包用于管理模块，本质是用于存放python代码的目录
- 包一般由功能相似的代码模块组成
- 包内可以包含子包、子包中的子包（嵌套）
- 每个包的定义需编写 `__init__.py`

> `__init__.py` 文件：
>
> - 包被首次导入时执行 `__init__.py`
> - 作用是将文件夹变为一个Python模块
> - 作用是初始化包
> - 可以是空文件，通常也为空文件，也可以为它增加其他功能
> - `__init__.py` 中还有一个重要的变量， `__all__`，它用来将模块全部导入。
>
> ```
> # __init__.py
> __all__ = ['os', 'sys', 're', 'urllib']
>
> # 这时就会把注册在__init__.py文件中__all__列表中的模块和包导入到当前文件中来。
> ```

2.2 定义包

1. 创建一个文件夹，该文件夹的名字就是该包的包名。
2. 在该文件夹内添加一个 __init__.py 文件即可。

> 包的本质就是模块，因此导入包和导入模块的语法完全相同

```
# 先新建一个包文件夹，然后在该文件夹中添加一个 __init__.py 文件
# __init__.py
'''
这是学习包的第一个示例
'''
print('this is first_package')
```

```python
# 使用上面定义好的包
# 导入bao包（模块）
import bao

print('==========')
# 输出了包的说明文档
print(bao.__doc__)
# 输出了包的类型
print(type(bao))
# 输出了包本身
print(bao)# 输出说明了导入包的本质就是加载并执行该包下的 __init__.py 文件
```

2.3 发布

打包

使用`setuptools`打包重点在于`setup.py`文件编写。这个文件主要用来描述你的项目信息，好让setuptools 打包工具来帮你打包项目。

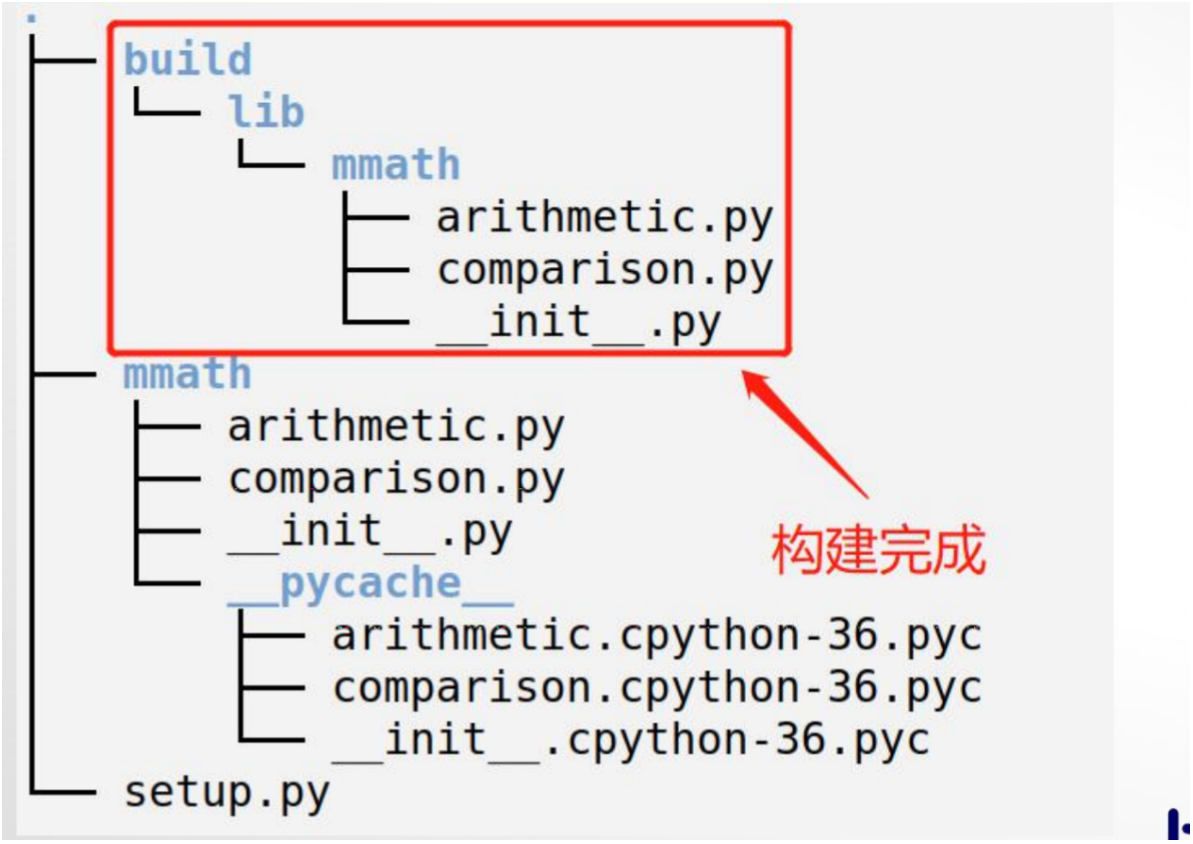
以下图的目录结构为例

编写`setup.py`文件：

```python
from distutils.core import setup
setup(
    name = '文件名',
    version = '.01',
    discription = '这是一个测试包',
    author = 'Briup',
    py_modules = [
        # 包下的python文件
        'mmath.arithmetic‘,
        'mmath.comparison'
    ]
)
```

发布

python setup.py build　　构建模块



python setup.py sdist　　生成发布压缩包

2.4 安装和使用

> 命令：
>
> - ls　　查看当前文件夹中的压缩文件
> - tar -vxf 压缩文件名　　解压文件
> - cd 解压后的文件名　　进入解压文件
> - python setup.py install --prefix=安装路径　　安装包