在 Python 中，**模块（Module）** 和**包（Package）** 是组织代码的核心机制，用于解决 “代码复用” 和 “大型项目管理” 问题。简单来说：

- 模块是 “单个 Python 文件”（.py）（符合Python一切皆模块理念），包含函数、类、变量等；
- 包是 “包含多个模块的目录”，通过目录结构组织相关模块，让代码更有条理。

模块

学习目标：

1. 知道什么是模块
2. 学会导入模块的几种不同的方式
3. 知道模块的搜索路径
4. 了解模块的相关属性
   - `__name__`
   - `__file__`
   - `__all__`

什么是模块

一个以`.py`为后缀的 Python 文件就是一个模块。例如`calc.py`，里面可以定义函数、类、变量等，供其他文件调用。

```python
# calc.py（一个简单的模块）
pi = 3.14159  # 模块变量

def add(a, b):  # 模块函数
    return a + b

class Calculator:  # 模块类
    def multiply(self, a, b):
        return a * b
```

注意事项：

1. 每一个以扩展名 py 结尾的 Python 源代码文件都是一个 **模块**
2. **模块名** 同样也是一个 **标识符**，需要符合标识符的命名规则

在模块中定义的 **全局变量**、**函数**、**类** 都是提供给外界直接使用的 **工具**

模块 就好比是 **工具包**，要想使用这个工具包中的工具，就需要先 **导入** 这个模块

为什么需要模块

- **代码复用**：写一次代码，在多个文件中导入使用，避免重复编写
- **命名隔离**：不同模块可以有同名函数 / 类（通过 “模块名。功能” 访问，避免冲突）
- **逻辑拆分**：将不同功能的代码放在不同模块，结构更清晰（如 “数据处理模块”“绘图模块”）

如何导入模块

通过 `import` 语句导入模块，根据需求选择不同方式：

编号	导入方式	语法示例	说明
方式1	导入整个模块	`import 模块名`	使用时需加模块名前缀（`模块名.功能`）
方式2	导入模块并取别名	`import 模块名 as 别名`	简化模块名（`别名.功能`）
方式3	导入模块中的特定功能	`from 模块名 import 功能1, 功能2`	直接使用功能名（无需前缀）
方式4	导入特定功能并取别名	`from 模块名 import 功能名 as 别名`	直接使用别名
方式5	导入模块中的所有功能	`from 模块名 import *`	不推荐（可能导致命名冲突）

Calculator.py

```python
# 变量
zero = 0

# 函数
def sum1(*args):
    result = 0
    for i in args:
        result += i
    return result

# 类
```

```python
class User:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def print_info(self):
        print(f"{self.name}同学的年龄是:{self.age}")
```

方式一： import 模块名 → 导入整个模块，使用的时候直接用 `模块名.功能名`

```python
# 导入整个模块
# 可以使用模块中的所有功能
import Calculator

# 使用模块中的所有功能 [模块名.功能名]
print(Calculator.zero)                          # 0
print(Calculator.sum(1,2,3,4))                  # 10
user = Calculator.User('张三',20)
user.print_info()                               # 张三同学的年龄是:20
```

方式二： import 模块名 as 别名 → 导入整个模块并取别名，使用的时候直接使用

`别名.功能名`

```python
# 导入整个模块并取别名
import Calculator as Cal

# 使用模块中的所有功能 [别名.功能名]
print(Cal.zero)                             # 0
print(Cal.sum(1,2,3,4))                 # 10
user = Cal.User('张三',20)
user.print_info()                           # 张三同学的年龄是:20
```

方式三：from 模块名 import 功能1，功能2... → 直接使用模块中的功能名

```python
# 导入模块中的功能
from Calculator import zero,sum1

# 使用模块中的所有功能 [功能名]
print(zero)                             # 0
print(sum(1,2,3,4))                 # 10

# 没有导入的功能不能使用 name 'User' is not defined
# user = User('张三',20)
# user.print_info()                     # 张三同学的年龄是:20
```

方式4： from 模块名 import 功能名 as 别名 → 直接使用别名

```python
# 导入模块中的功能
# 导入整个模块
from Calculator import zero as cal_zero, sum as cal_sum

# 使用模块中的所有功能 [别名]
print(cal_zero)                              # 0
print(cal_sum(1,2,3,4))                 # 10

# 没有导入的模块不能使用 name 'User' is not defined
# user = User('张三',20)
# user.print_info()                          # 张三同学的年龄是:20
```

方式五： from 模块名 import * → 直接使用功能名（不推荐）

```python
# 导入对应模块的所有功能
from Calculator import *

# 使用模块中的所有功能 [功能名]
print(zero)                                  # 0
print(sum(1,2,3,4))                    # 10

user = User('张三',20)
user.print_info()                          # 张三同学的年龄是:20
```

注意事项：

1. 导入模块时，每一个模块占一行
2. 导入模块或者功能的代码，应该集中放置在文件开头，方便排查问题
3. 给模块起别名的时候，使用大驼峰命名法
4. 当导入的不同模块中，有同名功能的时候，后导入的会覆盖先导入的
5. 当导入了同名功能的时候，就可以给功能起别名，别名也需要满足标识符的命名规范
6. **起别名的时候，不要和系统的模块（包）文件重名**

模块的搜索路径

当导入模块时，Python 会按以下顺序查找模块文件：

1. 当前执行脚本所在的目录；
2. 系统环境变量 `PYTHONPATH` 指定的目录；
3. Python 安装目录的标准库路径（如 `site-packages` ）。

可以通过 `sys.path` 查看搜索路径：

```python
import sys
print(sys.path)  # 输出模块搜索路径列表
```

模块的name属性

每个模块都有`__name__`属性，用于标识模块的“名称”：

- 当模块**被导入**时，`__name__`的值是模块名（如`calc`）；
- 当模块**自身被执行**时，`__name__`的值是`"__main__"`。

用途：在模块中写“测试代码”，只有当模块自身运行时才执行：

```python
# calc.py 中添加测试代码
if __name__ == "__main__":
    # 只有直接运行calc.py时，才执行以下代码
    print("测试add函数：", add(2, 3))  # 输出：5
    print("测试Calculator类：", Calculator().multiply(2, 3))  # 输出：6
```

模块的其他属性

file属性

Python 中每一个模块都有一个内置属性`__file__`可以 **查看模块**的**完整路径**

```python
import Calculator            # 自定义模块
import random                # 系统模块

print(Calculator.__file__)   # D:\back\cskaoyan\second-project\08-模块与包\Calculator.py
print(random.__file__)       # D:\soft\python\py313\Lib\random.py
```

all属性

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

cal.py

```python
__all__ = ['zero','sum1']

# 变量
zero = 0

```

```python
# 函数
def sum1(*args):
    result = 0
    for i in args:
        result += i
    return result

# 类
class User:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def print_info(self):
        print(f"{self.name}同学的年龄是:{self.age}")
```

导入

```python
# 导入整个模块
from cal import *

# 声明了的导入，没问题
print(zero)                         # 0
print(sum(1,2,3,4))                 # 10

# 没有声明的，导入会报错             # NameError: name 'User' is not defined
# user = User('张三',20)
# user.print_info()
```

包

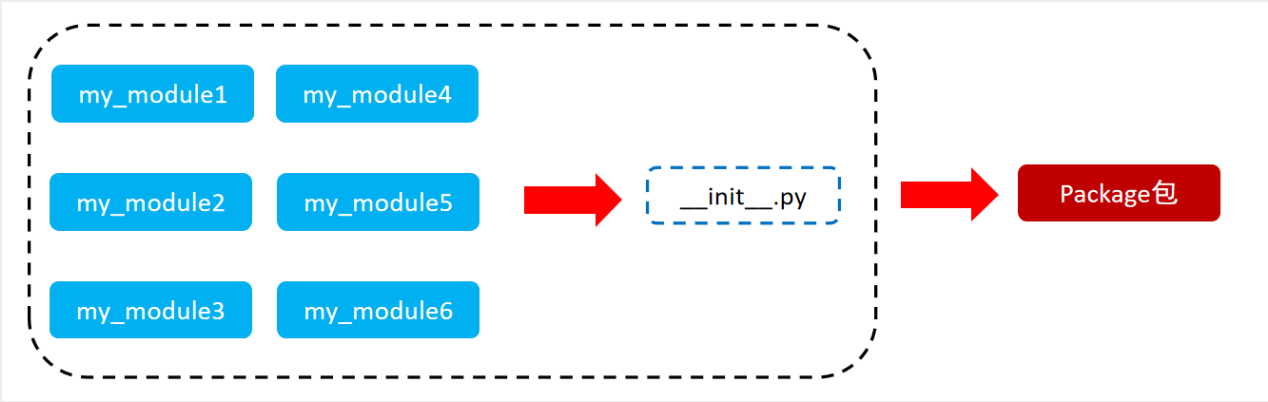
学习目标:

1. 知道包是什么
2. 了解包的基本结构
3. 会制作包与导包

包是什么

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文件

从逻辑上看，包的本质依然是模块



包的基本结构

如上图所示，一个包内包含有：

- `__init__.py` 文件
- 模块文件
- 子包

一个典型的包结构如下（以“学生管理系统”为例）：

```
student_management/   # 顶层包
├── __init__.py       # 包的初始化文件（必须有，可空）
├── core/             # 子包（核心功能）
│   ├── __init__.py
│   ├── student.py    # 学生类模块
│   └── manager.py    # 管理类模块
└── utils/            # 子包（工具功能）
    ├── __init__.py
    └── validator.py # 数据校验模块
```

`__init__.py` 的作用：告诉 Python “这是一个包”，而非普通目录；可以在其中定义包的初始化逻辑（如批量导入模块）。

导入包内的模块

导包和导入模块非常类似，有如下几种方式

- import 包名.模块名
- from 包名 import 指定的模块
- from 包名 import *

快速入门案例：

1. 新建包`my_package`
2. 新建包内模块：`my_module1` 和 `my_module2`
3. 在`my_module1`和`my_module2`中分别添加 say_hello() 函数，并给出不同的方法体
4. 在当前脚本文件中，导入包，并调用对应的函数

文件准备：

`my_module1.py`

```
def say_hello():
    print("my module1...say...")
```

`my_module2.py`

```
def say_hello():
    print("my module2...say...")
```

导包并使用

第一种方式： import 包名.子包名.模块名 → 使用模块中的功能的时候［包名.模块名.功能名］

```
# 第一种方式，导入包中指定的模块
# import 包名.子包名.模块名
import wangdao.my_module1

# 使用包中模块中的函数[包名.模块名.功能名]
wangdao.my_module2.say_hello()
```

第二种方式： from 包名 import * → 使用模块中的功能的时候［模块名.功能名］

注意：当使用import * 导入包中的所有的模块的时候，需要在包中的`__init__.py`文件中指定`__all__`属性来指定那些模块可以导入

```
from . import my_module1
from . import my_module2
```

```
# 第二种方式，导入__init__中配置好的所有模块
from wangdao import *

# 使用 模块名.功能名来调用
my_module1.say_hello()
# my_module2.say_hello()                    # 找不到，因为没有在__all__属性中配置
```

第三种方式：from 包名 import 模块名 → 使用模块中的功能的时候［模块名.功能名］

当然，这里也可以给模块起别名

```
# 第三种方式
from wangdao import my_module2

# 使用模块中的功能
my_module2.say_hello()
```

常用内置模块与第三方包

Python 的强大之处在于丰富的模块和包生态

- 内置模块

  Python 自带，无需安装即可使用。如`os`（文件操作）、`datetime`（时间处理）、`json`（数据序列化）

  ```
  import datetime

  # 获取当前时间
  # 2025-10-20 16:48:24.986831
  print(datetime.datetime.now())
  ```

- 第三方包

  由社区开发，需用`pip`安装（如`requests`（网络请求）、`pandas`（数据分析））

  安装第三方包

  ```
  pip install requests  # 安装第三方包
  ```

  使用第三方包中的模块和功能

  ```
  import requests
  response = requests.get("https://www.baidu.com")  # 使用第三方包
  print(response)
  ```

pip的介绍

`pip`是 Python 官方的第三方包管理工具，用于**安装、卸载、升级、查询**Python 第三方库（如`requests`、`pandas`等），是 Python 生态中不可或缺的工具。几乎所有公开的 Python 第三方包都可以通过`pip`快速获取，极大简化了依赖管理流程。

基本认识

- **作用**：管理 Python 第三方包（非 Python 标准库自带的包），包括下载、安装、更新、删除等；
- **内置性**：Python 3.4+ 和 Python 2.7.9+ 已默认内置 `pip`，安装 Python 时会自动附带（无需单独安装）；
- **调用方式**：在命令行（Windows 的 cmd/PowerShell、Linux/macOS 的终端）中使用 `pip` 或 `pip3` 命令（区分 Python 2 和 Python 3，推荐用 `pip3` 明确指定 Python 3，这是在Ubuntu等Linux下）目前Python2已经基本淘汰，所以windows下pip就是python3。

相关操作

以下操作均在**命令行**中执行，以 Python 3 为例，使用 `pip3` 命令（部分环境中 `pip` 也指向 Python 3，可通过 `pip --version` 确认）。

```
C:\Users\ciggar>pip3 --version
pip 25.2 from D:\soft\python\py313\Lib\site-packages\pip (python 3.13)
```

安装第三方包

安装最新版本

```
pip3 install 包名
```

示例：安装用于网络请求的 `requests` 库

```
pip3 install requests
```

安装指定版本

```
pip3 install 包名==版本号
```

卸载已安装的包

输入命令

```
pip3 uninstall 包名
```

例如：卸载requests

```
pip3 uninstall requests
```

升级已安装的包

将已安装的包升级到最新版本：

```
pip3 install --upgrade 包名  # 或 -U 简写
```

示例：升级 `requests` 到最新版

```
pip3 install -U requests
```

查看已安装的包列表

列出所有已安装的包

```
pip3 list
```

```
C:\Users\ciggar>pip3 list
Package                 Version
----------------------- ---------
asttokens               3.0.0
certifi                 2025.10.5
charset-normalizer      3.4.4
colorama                0.4.6
decorator               5.2.1
executing               2.2.1
idna                    3.11
ipython                 9.5.0
ipython_pygments_lexers 1.1.1
jedi                    0.19.2
matplotlib-inline       0.1.7
parso                   0.8.5
pip                     25.2
prompt_toolkit          3.0.52
pure_eval               0.2.3
Pygments                2.19.2
requests                2.32.5
stack-data              0.6.3
traitlets               5.14.3
urllib3                 2.5.0
wcwidth                 0.2.14
```

查看某个包的详细信息

查询某个已安装包的版本、作者、依赖等信息：

```
pip3 show 包名
```

示例：查看 `requests` 的信息

```
pip3 show requests
```

pip换源

默认情况下，`pip` 从国外的 PyPI 仓库（https://pypi.org/）下载包，国内网络可能较慢。解决方法是**切换到国内镜像源**（如豆瓣、阿里云等），速度会大幅提升。

临时使用国内源

安装包时，通过 `-i` 参数指定镜像源地址：

```
pip3 install 包名 -i 镜像源地址
```

常用国内镜像源：

- 豆瓣：https://pypi.doubanio.com/simple/
- 阿里云：https://mirrors.aliyun.com/pypi/simple/
- 清华大学：https://pypi.tuna.tsinghua.edu.cn/simple/

示例：用豆瓣源安装 `pandas`

```
pip3 install pandas -i https://pypi.doubanio.com/simple/
```

永久设置国内源

一次配置，后续所有 `pip` 操作自动使用国内源，无需每次加 `-i` 参数。

三大系统（windows，Mac，Linux）一句话搞定

pip config set global.index-url https://mirrors.aliyun.com/pypi/simple/

下面的无需看，除非是通过上面命令设置出现异常，再看下面内容：

Windows 系统

1. 在文件资源管理器地址栏输入 `%APPDATA%`，回车进入 `Roaming` 目录；
2. 新建文件夹 `pip`，进入该文件夹后新建文件 `pip.ini`
3. 在 `pip.ini` 中写入以下内容（以清华源为例）

```
[global]
index-url = https://mirrors.aliyun.com/pypi/simple

[install]
trusted-host = https://pypi.tuna.tsinghua.edu.cn
```

Linux/MacOS系统

1 打开终端，执行以下命令创建配置文件：

```
mkdir -p ~/.pip  # 创建.pip目录
vim ~/.pip/pip.conf  # 用vim编辑配置文件（也可用其他编辑器）
```

2 在 `pip.conf` 中写入以下内容（以阿里云为例）：

```
[global]
index-url = https://mirrors.aliyun.com/pypi/simple/
[install]
trusted-host = mirrors.aliyun.com
```