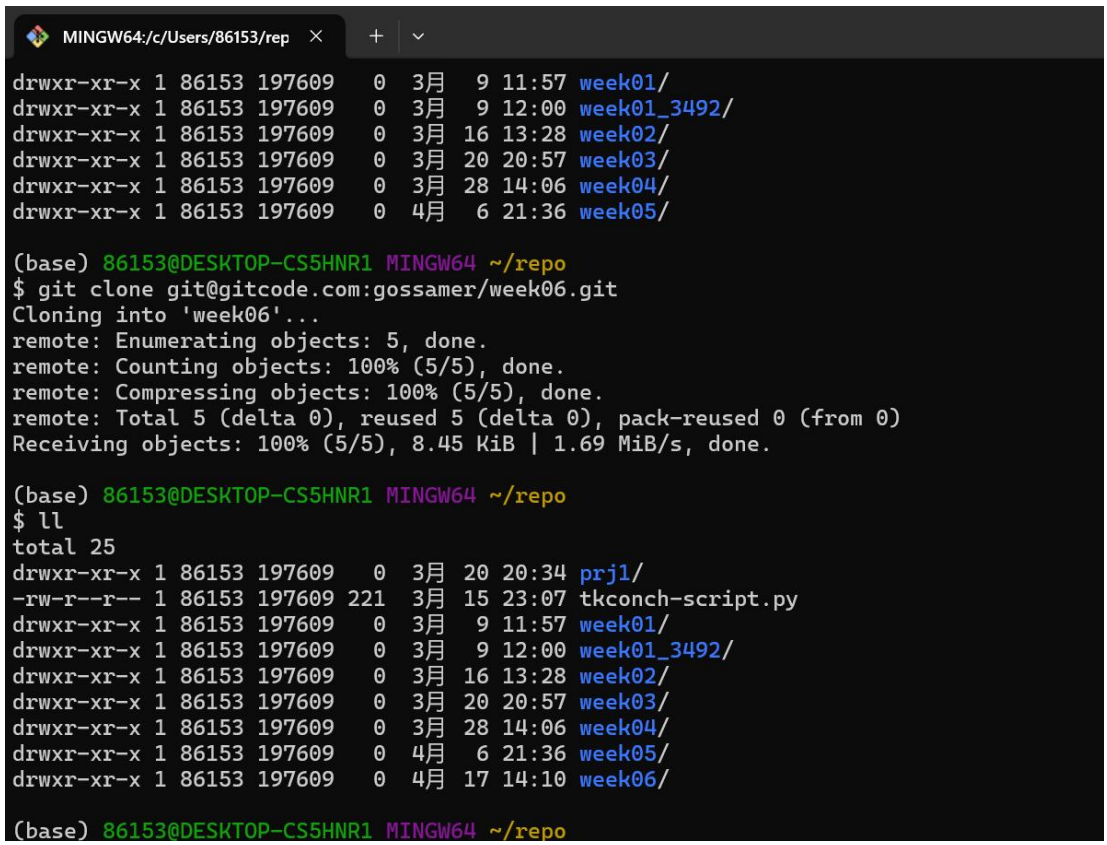
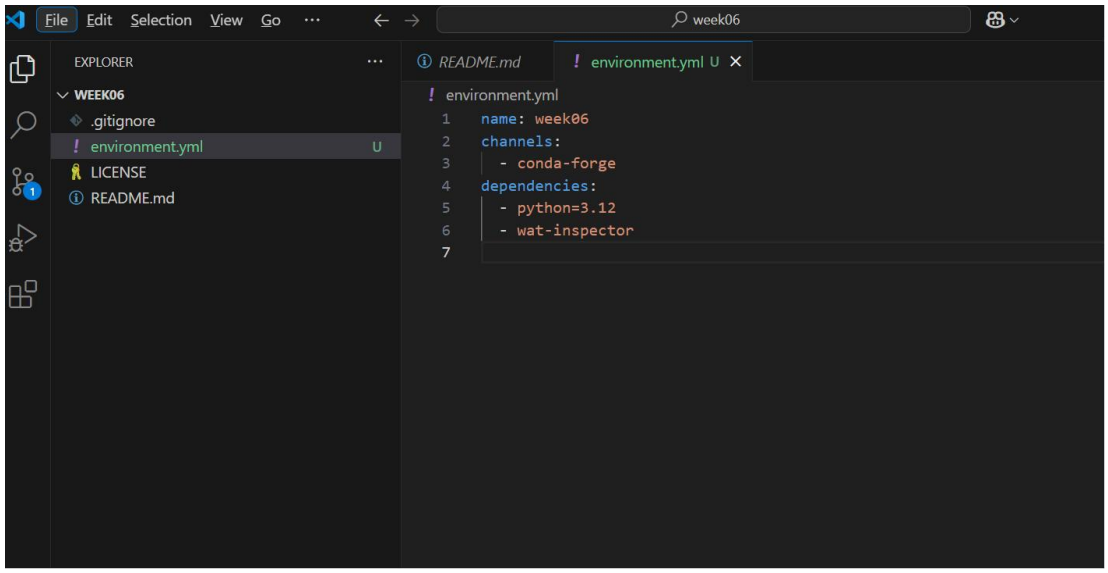
第六周学习报告

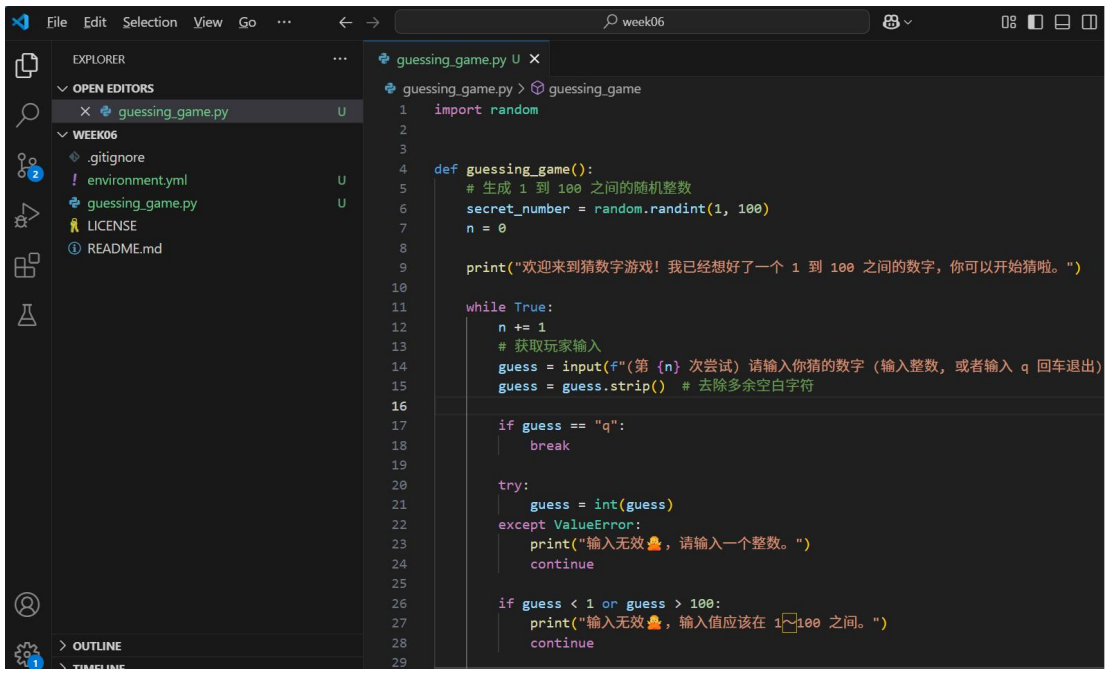
第 6 周 Python 代码组织 (初级)

1. Fork [第 06 周打卡](https://gitcode.com/cueb-fintech/week06) 仓库至你的名下，然后将你名下的这个仓库 Clone 到你的本地计算机



2. 用 VS Code 打开项目目录，新建一个 `environment.yml` 文件，指定安装 Python 3.12，然后运行 `conda env create` 命令创建 Conda 环境





3. 创建一个 `guessing_game.py` 文件，复制粘贴以下代码，运用 `pdb` 调试器理解其运行流程：

```python
import random
```

```
def guessing_game():
    # 生成 1 到 100 之间的随机整数
    secret_number = random.randint(1, 100)
    n = 0

    print("欢迎来到猜数字游戏！我已经想好了一个 1 到 100 之间的数字，你可以开始猜啦。")

    while True:
        n += 1
        # 获取玩家输入
        guess = input(f"(第 {n} 次尝试) 请输入你猜的数字 (输入整数, 或者输入 q 回车退出): ")
        guess = guess.strip()  # 去除多余空白字符

        if guess == "q":
            break

        try:
            guess = int(guess)
        except ValueError:
```

```
            print("输入无效 ， 请输入一个整数。")
            continue

        if guess < 1 or guess > 100:
            print("输入无效 ， 输入值应该在 1～100 之
间。")
            continue

        if guess == secret_number:
            print("恭喜你 ， 猜对了！ ")
            break

        if guess < secret_number:
            print("猜的数字太小了，再试试 。")
            continue

        if guess > secret_number:
            print("猜的数字太大了，再试试 。")
            continue

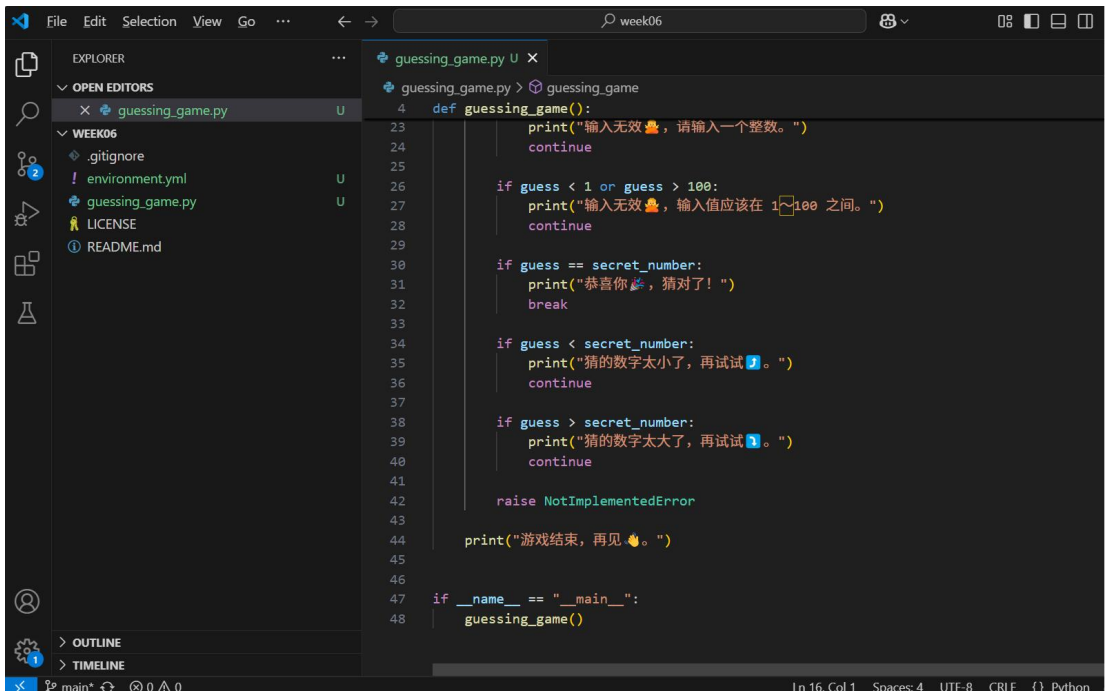
        raise NotImplementedError
```

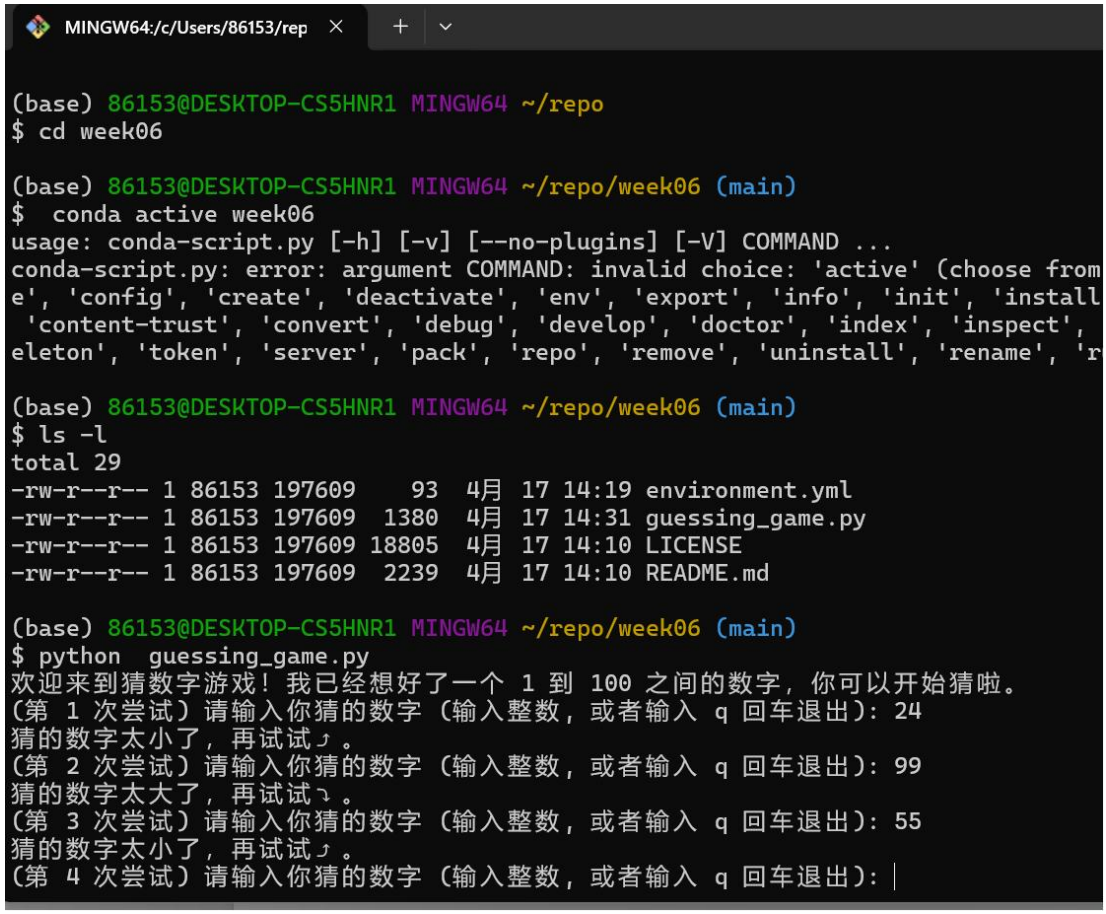
print("游戏结束，再见 。")

if __name__ == "__main__":

guessing_game()

```


```



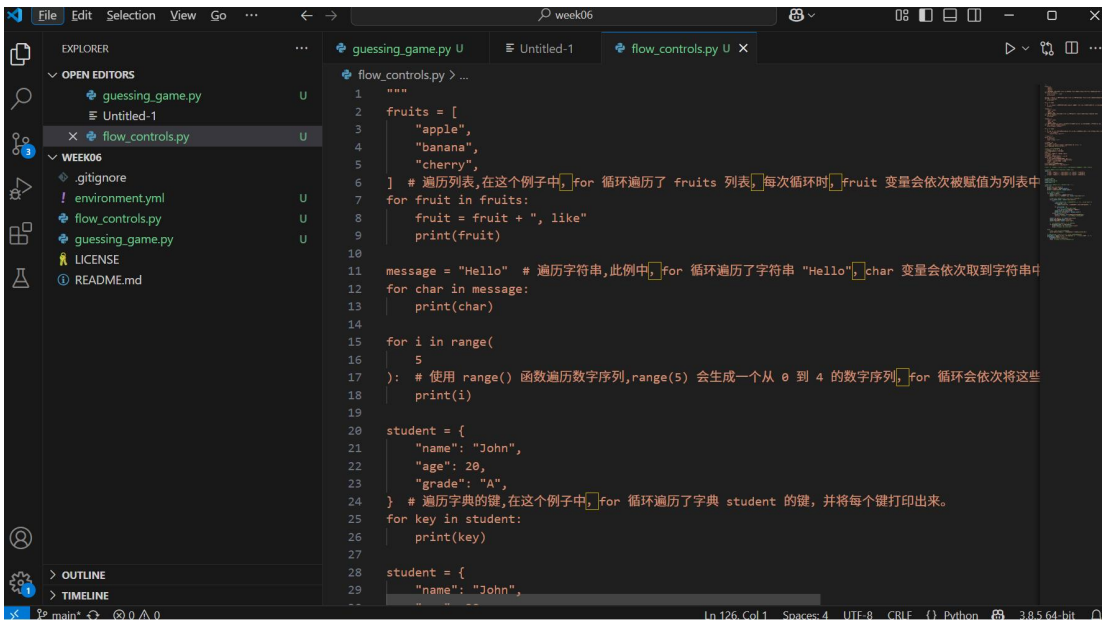
4. 创建一个 `flow_controls.py` 文件，让豆包 (或 DeepSeek 等任何大模型) 生成例子，尝试运行，体会理解以下 Python 流程控制语句：

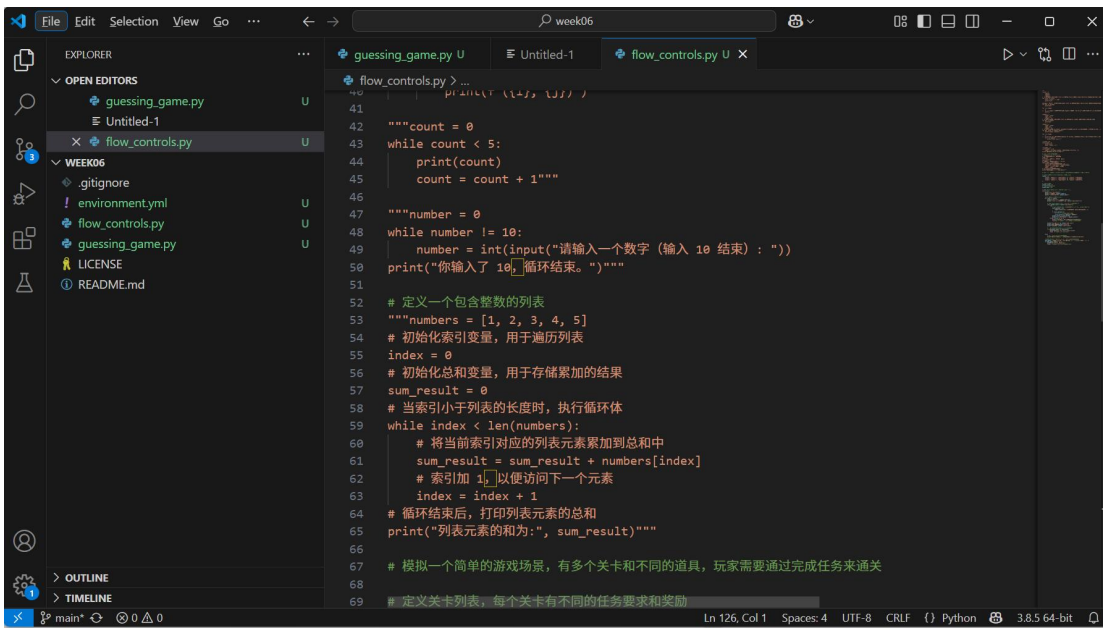
- `for` 迭代循环 (iteration loop)
- `while` 条件循环 (conditional loop)
- `break` 打断跳出循环
- `continue` 跳至下一轮循环
- `for...else` 循环未被打断的处理
- `if` 条件分支
- `if...elif[...elif]` 多重条件分支

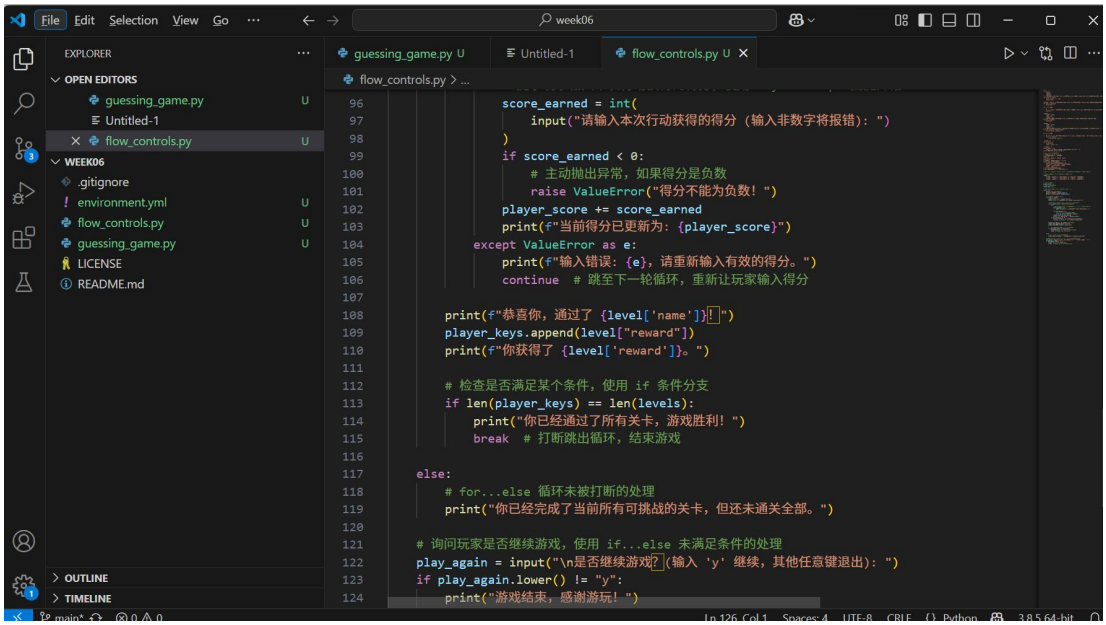
- `if...else` 未满足条件的处理

- `try...except[...except...else...finally]` 捕捉异常的处理

- `raise` 主动抛出异常







5. 创建一个 `mylib.py` **模块** (module)，在里面定义以下函数，再创建一个 `myjob.py` **脚本** (script)，从 `mylib.py` 导入函数并尝试调用：

- 定义函数 `func1`，没有形参，没有返回值
- 定义函数 `func2`，没有形参，有返回值
- 定义函数 `func3`，只有一个 **位置形参** (positional parameter)，先尝试传入 **位置实参** (positional argument) 调用，再尝试传入 **命名实参** (named argument) 调用，再尝试不传实参 (会报错)
- 定义函数 `func4`，只有一个 **命名形参** (named parameter)，先传入 **位置实参** 调用，再传入 **命名实参** 调用，再尝试不传实参 (取默认值)

- 定义函数 `func5`，接受多个位置形参和命名形参，尝试以位置/命名各种不同方式传入实参，注意位置参数必须排在命名参数之前

- 定义函数 `func6`，在形参列表中使用 `/` 来限定只接受位置实参的形参

- 定义函数 `func7`，在形参列表中使用 `*` 来限定只接受命名实参的形参

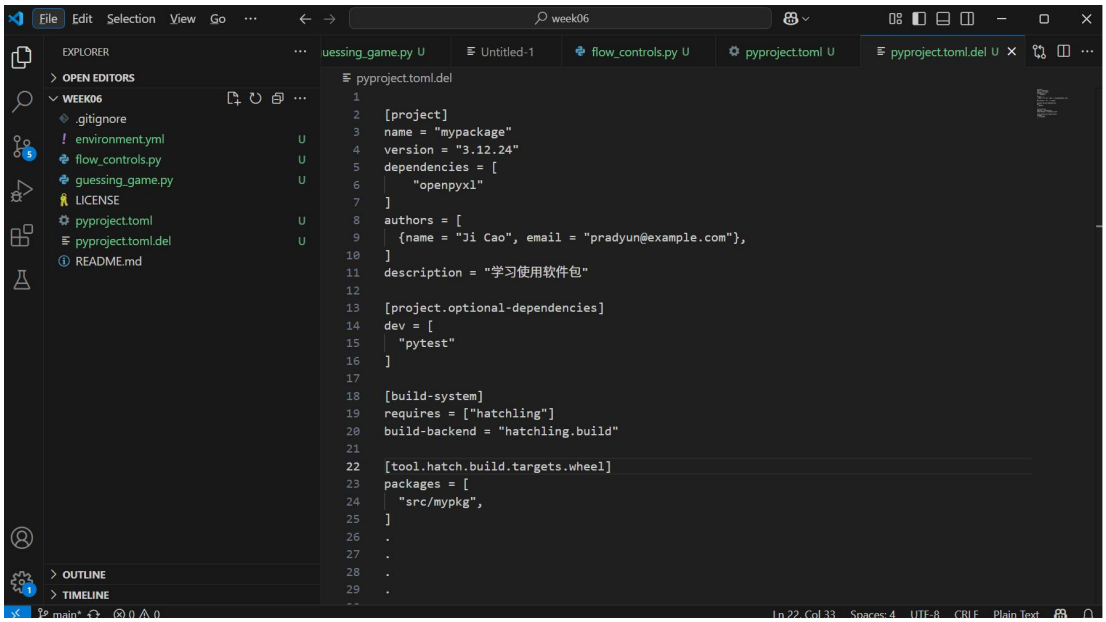
- 定义函数 `func8`，在位置形参的最后，在形参名称前使用 `*` 允许传入任意数量的位置实参 (被打包为元组)

- 定义函数 `func9`，在命名形参的最后，在形参名称前使用 `**` 允许传入任意数量的命名实参 (被打包为字典)

- 定义函数 `func10`，接受两个位置形参，一个命名形参，尝试在调用时使用 `*` 将可迭代对象 (如元组或列表) 自动解包，按位置实参传入

- 定义函数 `func11`，接受一个命名形参，两个命名形参，尝试在调用时使用 `**` 将映射对象 (如字典) 自动解包，按命名实参传入

- 定义函数 `func12`，给函数添加 **内嵌文档** (docstring)，给形参和返回值添加 **类型注解** (type annotation)，提高函数签名的可读性



6. 把 `mylib` 模块转变为 **软件包** (package) 安装进当前的 Conda 环境来使用

- 把 `myjob.py` 脚本移动至 `scripts/myjob.py`，再次尝试运行，会发现 `import mylib` 失败，这是由于 `mylib` 并没有打包成 **软件包** (package) 安装

- 将 `mylib.py` 模块移动至 `src/mypkg/mylib.py`，创建 `src/mypkg/__init__.py` 文件，准备好软件包的源代码

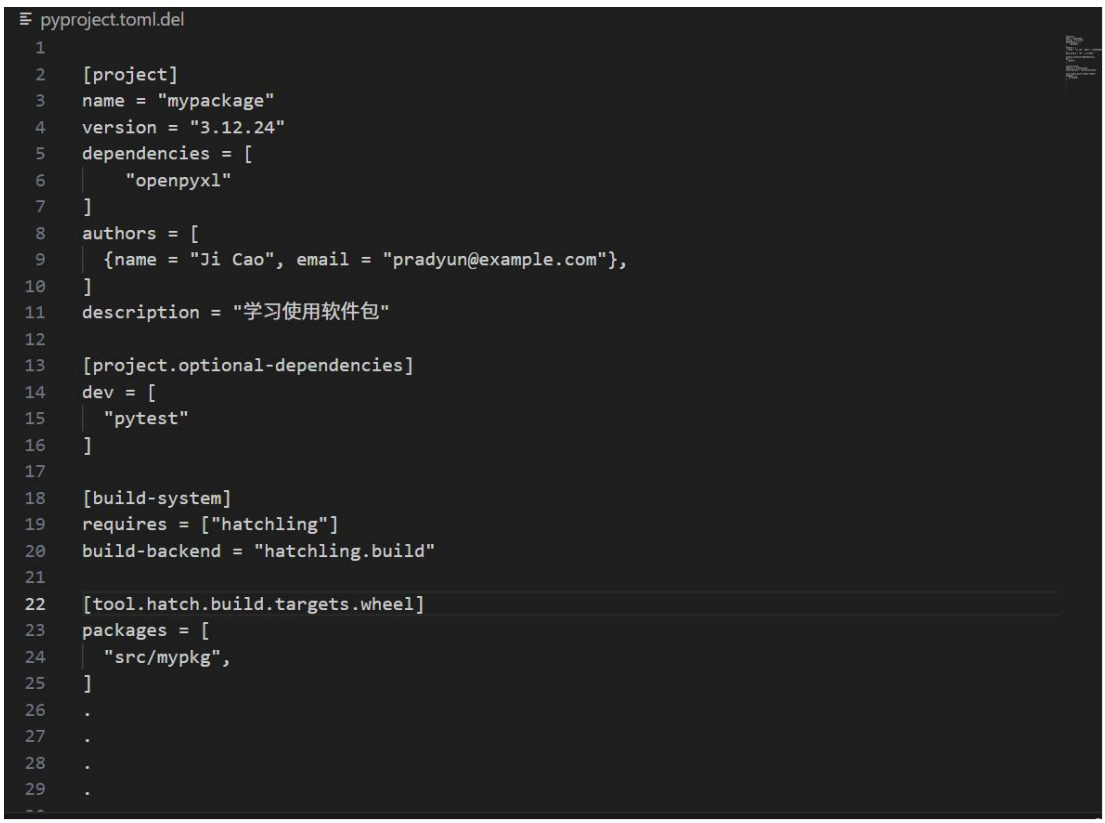
- 创建 `pyproject.toml` 配置文件，按照 [文档](https://packaging.python.org/en/latest/guides/writing-pyproject-toml/) 填写基本的软件包信息

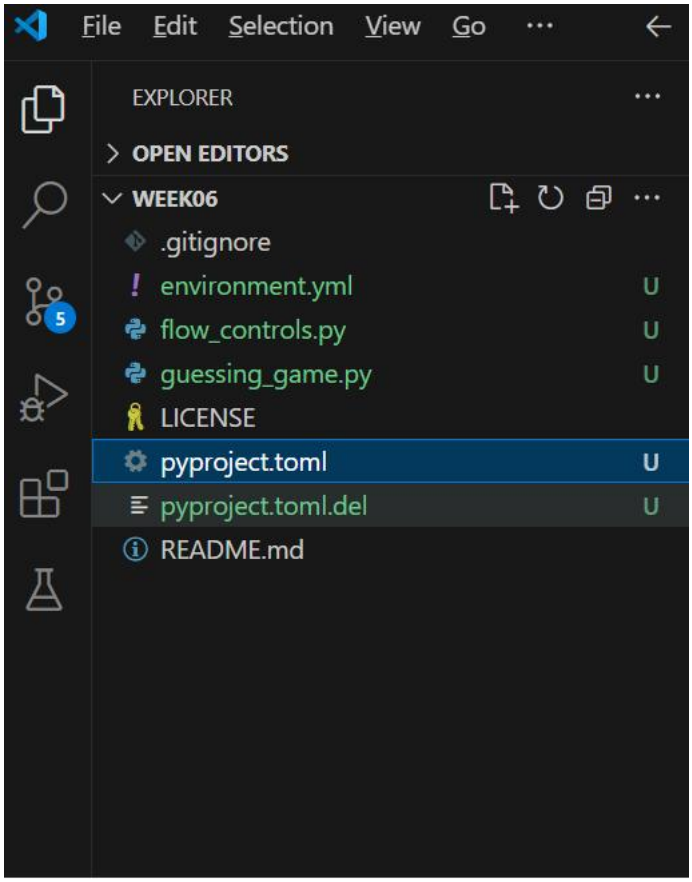
- 在 `pyproject.toml` 配置文件里，按照 [文档](https://hatch.pypa.io/latest/config/build/) 填写软件包的 **

构建** (build) 配置

- 使用 `pip install -e .` 以本地可编辑模式把当前软件包安装进当前 Conda 环境

- 修改 `environment.yml` 文件，使得 `conda env create` 自动安装本地可编辑软件包




File
Edit
Selection
View
Go
EXPLORER
OPEN EDITORS
WEEK06
.gitignore
environment.yml
U
flow_controls.py
U
guessing_game.py
U
LICENSE
pyproject.toml
U
pyproject.toml.del
U
README.md