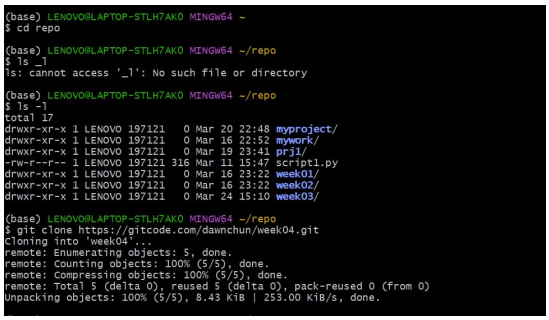
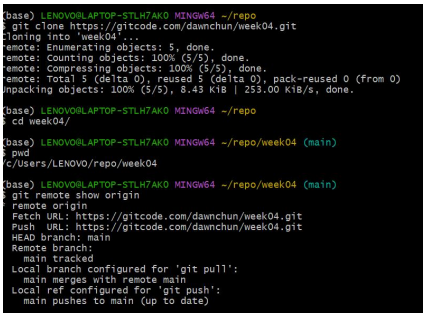
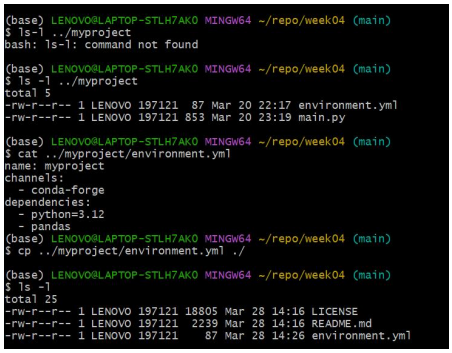
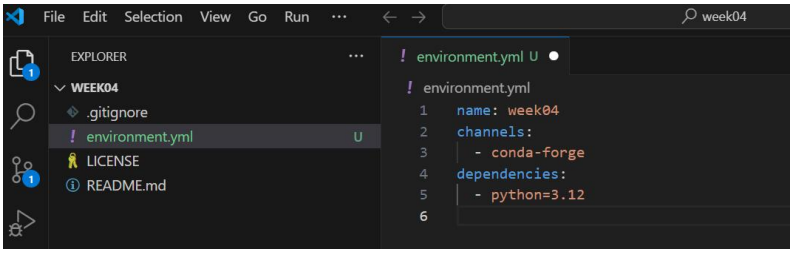
1. Fork 第 04 周打卡 仓库至你的名下，然后将你名下的这个仓库 Clone 到你的本地计算机

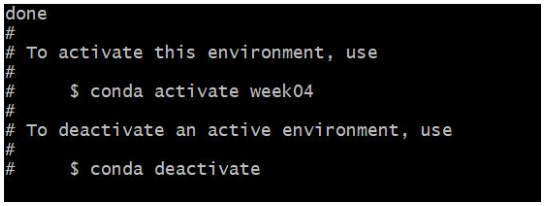




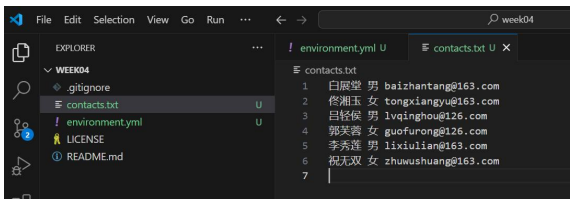
2. 用 VS Code 打开项目目录，新建一个 environment.yml 文件，指定安装 Python 3.12，然后运行 conda env create 命令创建 Conda 环境

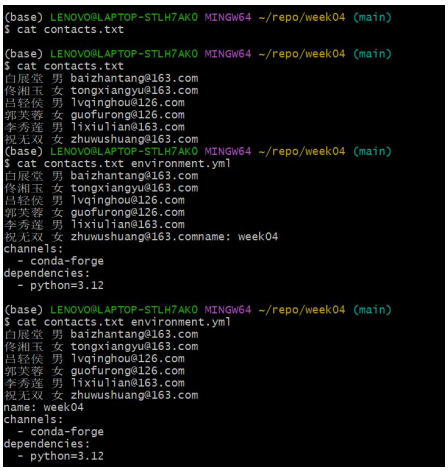




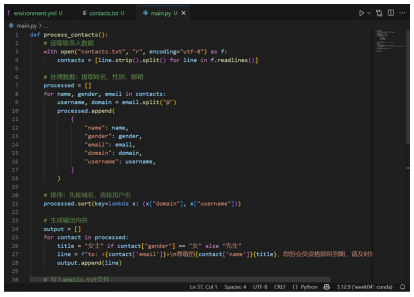


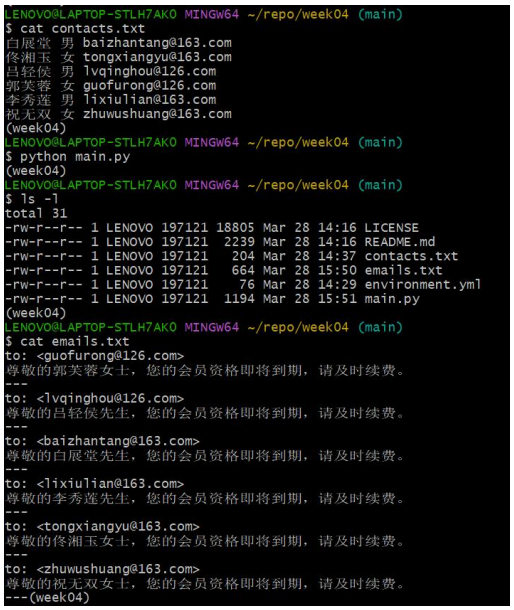
3. 新建一个 contacts.txt 文件，每行写一个联系人，每个联系人都包含姓名、性别、邮箱三个字段，用空格分隔

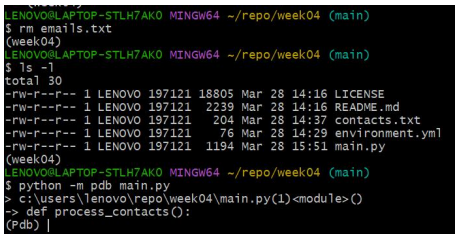




4. 新建一个 main.py 文件，里面写 Python 代码，要求读取 contacts.txt 文件的内容，进行数据处理后，输出一个 emails.txt 文件







在 Python 调试器(pdb)中，`n`和`s`是两个最常用的单步执行命令，它们的区别可以用一个简单的比喻来解释：

`n` (next) - 下一步

- **通俗解释**：像"快进"一样执行当前行代码，**不进入**函数内部
- **场景**：当你只想看当前函数的执行过程，不想深入被调用的函数细节时
- **示例**：

  ```python
  def greet(name):
      print(f"Hello, {name}")  # 用 n 会直接执行完这行，不会进入 print 函数内部

  greet("Alice")  # 用 n 会直接执行完整个函数调用
  ```

`s` (step) - 步入
- **通俗解释**：像"显微镜"一样**进入**当前行调用的函数内部
- **场景**：当你想深入查看某个函数的具体实现时
- **示例**：
  ```python
  def greet(name):
      print(f"Hello, {name}")  # 用 s 会进入 print 函数的实现

  greet("Alice")  # 用 s 会进入 greet 函数内部
  ```

实际调试演示
假设调试以下代码：
```python
def cook():
    print("加热食材")  # 断点停在这里
    add_seasoning()

def add_seasoning():
    print("加盐")
    print("加胡椒")

cook()
```
在 pdb 中：
1. 使用`n`时：
   - 会直接执行完`print("加热食材")`和`add_seasoning()`两行
   - 看不到`add_seasoning`函数内部的执行过程

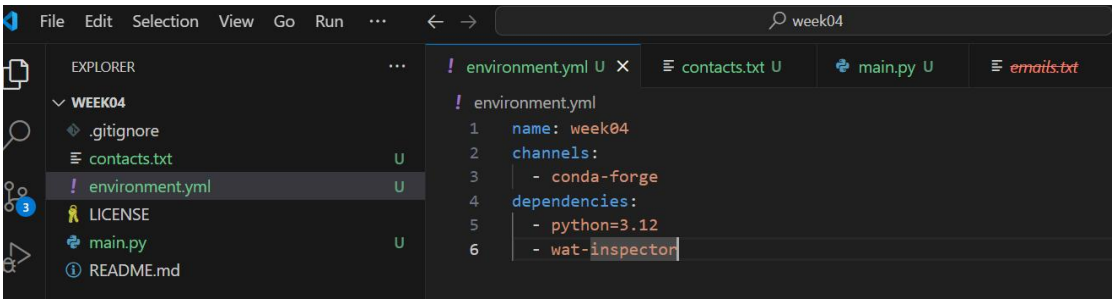
2. 使用`s`时：
   - 执行`print("加热食材")`时会进入 print 函数的底层实现
   - 执行到`add_seasoning()`时会进入该函数内部，看到`加盐`和`加胡椒`的执行过程

记忆技巧

- `n`像"鸟瞰"(只看表面)
- `s`像"潜水"(深入内部)
- 当不确定用哪个时，先用`s`进入函数，如果发现不需要深入，再用`c`跳出继续执行

这两个命令配合使用，可以像控制电影播放一样精确控制调试过程：用`s`进入细节镜头，用`n`快速跳过已知正常的片段。

q 退掉




```
(Pdb) wat
*** NameError: name 'wat' is not defined
(Pdb) import wat
(Pdb) wat
Try wat / object or wat.modifiers / object to inspect an object. Modifiers are:
  .short or .s to hide attributes (variables and methods)
  .dunder to print dunder attributes
  .code to print source code of a function, method or class
  .long to print non-abbreviated values and documentation
  .nodocs to hide documentation for functions and classes
  .caller to show how and where the inspection was called
  .all to include all information
  .ret to return the inspected object
  .str to return the output string instead of printing
  .gray to disable colorful output in the console
  .color to enforce colorful outputs in the console
Call wat.locals or wat() to inspect local variables.
Call wat.globals to inspect global variables.

(Pdb) wat()
Local variables:
  __builtins__: dict = {…
  __file__: pdb._ScriptTarget = 'C:\Users\LENOVO\repo\week04\main.py'
  __name__: str = '__main__'
  __pdb_convenience_variables: dict = {…
  __spec__: NoneType = None
  wat: wat.inspection.inspection.Wat = <WAT Inspector object>
(Pdb)
```

Python 变量寻找时的 LEGB 规则