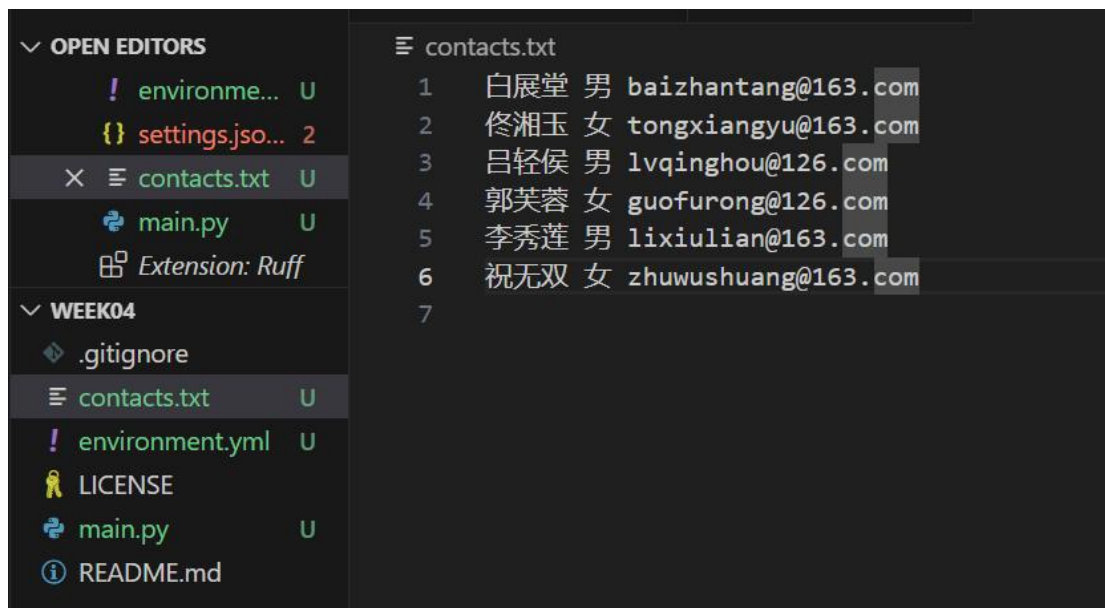


第四周主题：建议项目“车间”，运用 AI 生成静态代码并运用，学习 Python 核心概念

1. 创建环境

```
! environment.yml
1  name: week04
2  channels:
3    - conda-forge
4  dependencies:
5    - python=3.12
6
```

新建 contact.txt

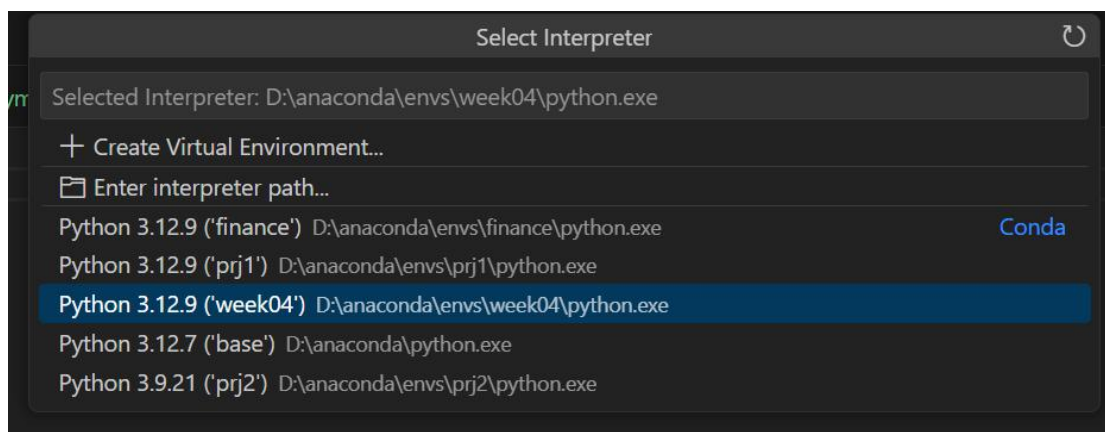


The screenshot shows the VS Code interface. On the left, the Explorer sidebar displays the project structure under 'WEEK04', including files like '.gitignore', 'contacts.txt', 'environment.yml', 'LICENSE', 'main.py', and 'README.md'. The 'contacts.txt' file is open in the editor, showing a list of contacts with their names, genders, and email addresses.

```
contacts.txt
1  白展堂 男 baizhantang@163.com
2  佟湘玉 女 tongxiangyu@163.com
3  吕轻侯 男 lvqinghou@126.com
4  郭芙蓉 女 guofurong@126.com
5  李秀莲 男 lixiulian@163.com
6  祝无双 女 zhuwushuang@163.com
7
```

注：cat 是 conact 的缩写，后可以跟多个文件，为了防止多个显示的文件时出现第一个文件最后一行和第二个文件最后一行并行出现，建议在输入文本时最后空一行

在 VS code 中设置当前所在环境：1.可以点右下角，再选中需要的环境/或者 Ctrl+shift+P 在搜索栏中输入 inter interpreter path（每个项目一个车间）



The screenshot shows the 'Select Interpreter' dialog box in VS Code. It lists several Python interpreters available in the environment. The interpreter 'Python 3.12.9 ('week04') D:\anaconda\envs\week04\python.exe' is selected and highlighted in blue. Other options include 'Python 3.12.9 ('finance')', 'Python 3.12.9 ('prj1')', 'Python 3.12.7 ('base')', and 'Python 3.9.21 ('prj2')'.

2. 启动 week04 中的代码环境（要确定文件夹中有 environment.yml 文件）

```

(base) lqy0929@LAPTOP-CDK7DEUF MINGW64 ~
$ cd /d/repo/week04

(base) lqy0929@LAPTOP-CDK7DEUF MINGW64 /d/repo/week04 (main)
$ pwd
/d/repo/week04

(base) lqy0929@LAPTOP-CDK7DEUF MINGW64 /d/repo/week04 (main)
$ ls -l
total 30
-rw-r--r-- 1 lqy0929 197121 204 3月 30 22:18 contacts.txt
-rw-r--r-- 1 lqy0929 197121 74 3月 28 17:11 environment.yml
-rw-r--r-- 1 lqy0929 197121 18805 3月 28 16:42 LICENSE
-rw-r--r-- 1 lqy0929 197121 1108 3月 30 22:55 main.py
-rw-r--r-- 1 lqy0929 197121 2239 3月 28 16:42 README.md

(base) lqy0929@LAPTOP-CDK7DEUF MINGW64 /d/repo/week04 (main)
$ conda activate week04
(week04)

```

查看大模型给出的代码

```

$ cat main.py
# 读取contacts.txt文件内容
contacts = []
with open("contacts.txt", "r", encoding="utf-8") as f:
    for line in f:
        line = line.strip()
        if line:
            name, gender, email = line.split()
            contacts.append({"name": name, "gender": gender, "email": email})

# 按邮箱域名和用户名排序
sorted_contacts = sorted(
    contacts, key=lambda x: (x["email"].split("@")[1], x["email"].split("@")[0])
)

# 生成输出内容
output_lines = []
for i, contact in enumerate(sorted_contacts):
    # 根据性别确定称呼
    title = "女士" if contact["gender"] == "女" else "先生"
    # 添加邮件内容行
    output_lines.append(f"to: <{contact['email']}>")
    output_lines.append(
        f"尊敬的{contact['name']}{title}, 您的会员资格即将到期, 请及时续费。"
    )
    # 如果不是最后一个联系人, 添加分隔符
    if i != len(sorted_contacts) - 1:
        output_lines.append("----")

# 写入emails.txt文件

```

运行 python main.py

```

lqy0929@LAPTOP-CDK7DEUF MINGW64 /d/repo/week04 (main)
$ ls -l
total 31
-rw-r--r-- 1 lqy0929 197121 204 3月 30 22:18 contacts.txt
-rw-r--r-- 1 lqy0929 197121 659 3月 31 08:29 emails.txt
-rw-r--r-- 1 lqy0929 197121 74 3月 28 17:11 environment.yml
-rw-r--r-- 1 lqy0929 197121 18805 3月 28 16:42 LICENSE
-rw-r--r-- 1 lqy0929 197121 1108 3月 30 22:55 main.py
-rw-r--r-- 1 lqy0929 197121 2239 3月 28 16:42 README.md
(week04)
lqy0929@LAPTOP-CDK7DEUF MINGW64 /d/repo/week04 (main)
$ cat emails.txt
to: <guofurong@126.com>
尊敬的郭芙蓉女士，您的会员资格即将到期，请及时续费。
---
to: <lvqinghou@126.com>
尊敬的吕轻侯先生，您的会员资格即将到期，请及时续费。
---
to: <baizhantang@163.com>
尊敬的白展堂先生，您的会员资格即将到期，请及时续费。
---
to: <lixiulian@163.com>
尊敬的李秀莲先生，您的会员资格即将到期，请及时续费。
---
to: <tongxiangyu@163.com>
尊敬的佟湘玉女士，您的会员资格即将到期，请及时续费。
---
to: <zhuwushuang@163.com>
尊敬的祝无双女士，您的会员资格即将到期，请及时续费。(week04)

```

3. 大模型下的 AI

1. 大模型的确已经能够按照我们的要求生成文章、代码，而且无疑还在更加快速地变得更加强大。但人类的分工在无限深化，人类的定制化需求永无止境，永远需要我们人类自己来解决更多的创新性问题。是的，**创新**，因为不创新的常规问题都可以自动化了。所以我们不要以为“大模型都可以写代码了，我们就不用再学习编程了”，相反，我们不仅要学（学已经变得很容易），而且还要能创新，学习的压力反而更大了。真正创新的文章、代码是不一样的，是有灵魂的。尽管大模型能够生成，但在这里我写下的每一个字，我都不希望是由大模型生成的，我都希望由我自己来写。我所编写的每一行代码，我都更愿意由我自己来写，大模型顶多作为一个助手（Copilot），一个学习的帮手，我还是希望最终由我亲手掌控，因为我做的是做前所未有的创新工作。——这就是我所预见的人类和大模型（AI）相处的哲学关系。

“自动化悖论”：人追求自动化，但永远不可能真正达成自动化，永远需要人来使用、发明、氛围编程/文盲程序：AI 下的小规模出错，能否承担？→AI 是老师、助手但永远不能停下自己学习的脚步。

4. 调试器

运行 `python -m pdb main.py` 命令（作用是以调试模式（debug mode）启动 Python 解释器，准备执行 `main.py` 里的代码）

在（pdb）提示符下 `l(显示代码)l 5,10` 可以显示第 5 行和第 10 行

→后面的代码是即将运行的代码

`n` (执行当前行)，直接回车就是可以再次运行 `n`

`p` (打印表达式)可以看某一定义变量具体代表的是什么，`function`（表示函数）

`P type()`可以显示某一个命令的类型，`p len()`：显示长度

注：`p` 可以直接显示内置命令；但是自己定义的变量必须运行过那一行，`p` 才能显示该变量

的定义

、s (步入调用)、pp (美观打印)、c (继续执行)、q (退出调试器)、r (返回)

回<https://docs.python.org/3/library/pdb.html#debugger-commands>

在运行调试器中的出现问题：

```
(Pdb) s
--Call--
> <frozen codecs>(309).__init__()
(Pdb) n
> <frozen codecs>(310).__init__()
```

询问大模型后得知

3. 调试时误入内置函数

- 可能原因：

- 使用 **s** (step) 命令时，意外进入底层模块（如 **codecs**），通常是因为当前执行的代码调用了编码相关的函数（如文件读取时的 **open**）。

- 解决方法：

- 避免使用 **s** 进入底层代码：

- 在调试时使用 **n** (next) 跳过底层函数调用。
- 如果误入底层代码，输入 **c** (continue) 恢复程序运行，直到下一个断点。

5. **wat-inspector**：检查 python 中的对象

6. **python** 核心语句

■ **Python 语法保留字 (reserved key words)**，在 VS code 中显示为红色：在 Python 语法中有特殊含义，不能作为变量名

■ **语句 (statement)**：在 VS code 中可以折叠，大的语句可以嵌套小语句

表达式 (expression)：构成语句的元素，一个表达式可以构成一个语句，表达式也可以嵌套

■ **缩进 (indent)**4 个字符：代表层级，缩进对齐表示子语句的边界在哪里，缩进要求严格的对齐

■ **局部变量 (local variable)**、只有进到语句里面才能反映的变量、

```
(Pdb) wat()
Local variables:
__builtins__: dict = {...}
__file__: pdb._ScriptTarget = 'D:\repo\week04\main.py'
__name__: str = '__main__'
__pdb_convenience_variables__: dict = {...}
__spec__: NoneType = None
contacts: list = []
email: str = 'baizhantang@163.com'
f: _io.TextIOWrapper = <_io.TextIOWrapper name='contacts.txt' mode='r' encoding='utf-8'>
gender: str = '男'
line: str = '白展堂 男 baizhantang@163.com'
name: str = '白展堂'
wat: wat.inspection.inspection.Wat = <WAT Inspector object>
```

local variable 可以用调试器中的 p 进行查看

全局变量 (global variable)


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

LEGB 规则

优先级顺序：Local → Enclosing → Global → Built-in。

关键行为：

如果变量在当前作用域（Local）找到，直接使用。

若未找到，依次向外层作用域（Enclosing、Global）查找。

最后查找内置作用域（Built-in）。

若所有作用域均未找到，抛出 `NameError`。

■ 函数 (function) 的定义 (define) 和调用 (call)：例 `print('aa')` 括号里就是调用

■ 字面值 (literal)：字符串 (str)、整数 (int)、列表 (list)、字典 (dict)、元组 (tuple) 如（引号的字符串””、方括号[]里的、-1）

■ 运算符 (operator)：==、if == else、.（访问运算符）、

■ 形参 (parameter)：调用的时候假装有；实参 (argument)：能找到真正对象、返回值 (return value)

■ 对象 (object)、类型 (type)、属性 (attribute)、方法 (method)

使用 wat /

```
(Pdb) n
> d:\repo\week04\main.py(8)<module>()
-> contacts.append({"name": name, "gender": gender, "email": email})
(Pdb) wat /contacts

value: [
  {
    'name': '白展堂',
    'gender': '男',
    'email': 'baizhantang@163.com',
  },
]
type: list
len: 1

Public attributes:
def append(object, /) # Append object to the end of the list.
def clear() # Remove all items from list.
def copy() # Return a shallow copy of the list.
def count(value, /) # Return number of occurrences of value.
def extend(iterable, /) # Extend list by appending elements from the iterable.
def index(value, start=0, stop=9223372036854775807, /) # Return first index of value...
def insert(index, object, /) # Insert object before index.
def pop(index=-1, /) # Remove and return item at index (default last)...
def remove(value, /) # Remove first occurrence of value...
def reverse() # Reverse *IN PLACE*.
def sort(*, key=None, reverse=False) # Sort the list in ascending order and return None...
```