

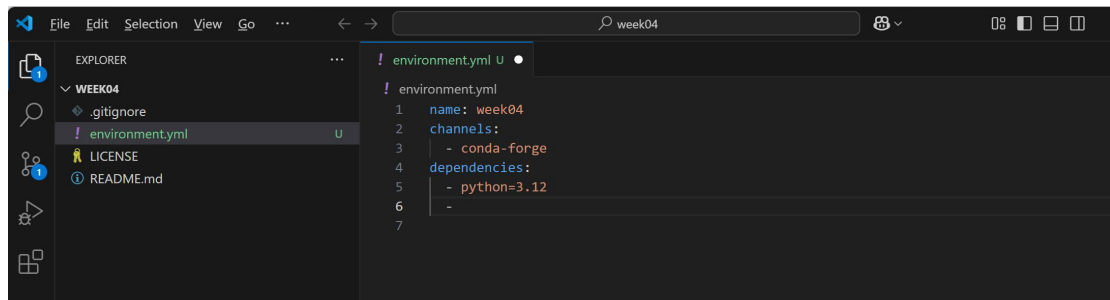
金融编程与计算 week04 作业

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

```
lenovo@LAPTOP-UDBV5M8K MINGW64 ~/repo
$ cd week04/
(base)
lenovo@LAPTOP-UDBV5M8K MINGW64 ~/repo/week04 (main)
$ pwd
/c/Users/lenovo/repo/week04
(base)
lenovo@LAPTOP-UDBV5M8K MINGW64 ~/repo/week04 (main)
$ git remote show origin
* remote origin
Fetch URL: https://gitcode.com/xixixining/week04.git
Push URL: https://gitcode.com/xixixining/week04.git
HEAD branch: main
Remote branch:
main tracked
Local branch configured for 'git pull':
main merges with remote main
Local ref configured for 'git push':
main pushes to main (up to date)
(base)
lenovo@LAPTOP-UDBV5M8K MINGW64 ~/repo/week04 (main)
$
```

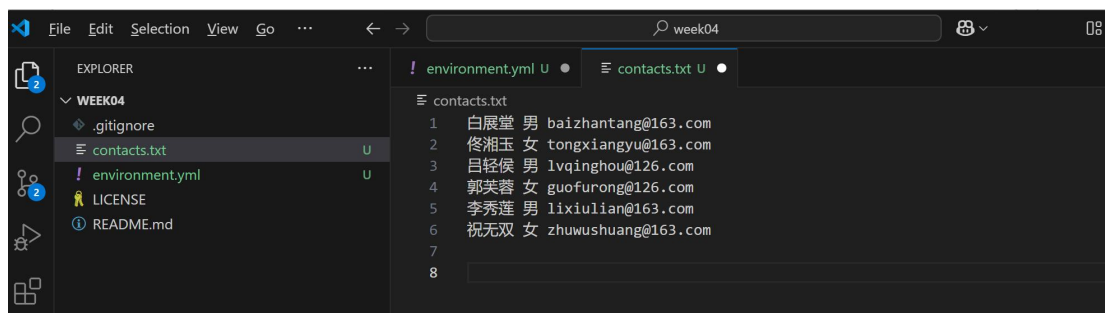
2、用 VS Code 打开项目目录，新建一个 environment.yml 文件，指定安装 Python 3.12，然后运行 conda env create 命令创建 Conda 环境

```
lenovo@LAPTOP-UDBV5M8K MINGW64 ~/repo/week04 (main)
$ ls -l ../myproject
total 4
-rw-r--r-- 1 lenovo 197121 1866 3月 20 10:50 environment.yml
(base)
lenovo@LAPTOP-UDBV5M8K MINGW64 ~/repo/week04 (main)
$ cat ../myproject/enviroment.yml
cat: ../myproject/enviroment.yml: No such file or directory
(base)
lenovo@LAPTOP-UDBV5M8K MINGW64 ~/repo/week04 (main)
$ cp ../myproject/environment.yml ./
(base)
lenovo@LAPTOP-UDBV5M8K MINGW64 ~/repo/week04 (main)
$
```



```
lenovo@LAPTOP-UDBV5M8K MINGW64 ~/repo/week04 (main)
$ conda env create
C:\Users\lenovo\anaconda3\Lib\argparse.py:2006: FutureWarning: 'remote_definition' is deprecated and will be removed in
25.9. Use 'conda env create --file=URL' instead.
  action(self, namespace, argument_values, option_string)
Retrieving notices: ...working... done
Warning: you have pip-installed dependencies in your environment file, but you do not list pip itself as one of your con
da dependencies. Conda may not use the correct pip to install your packages, and they may end up in the wrong place. P
lease add an explicit pip dependency. I'm adding one for you, but still nagging you.
CondaValueError: prefix already exists: C:\Users\lenovo\anaconda3\envs\prj1
```

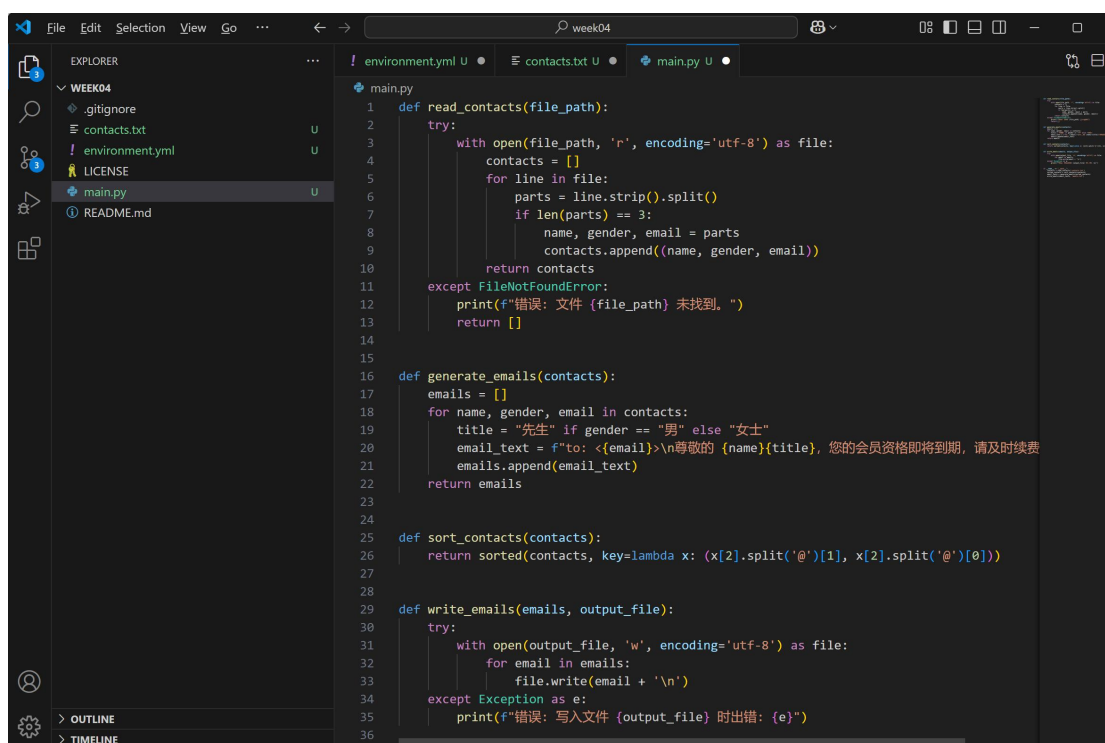
3、新建一个 **contacts.txt** 文件，每行写一个联系人，每个联系人都包含姓名、性别、邮箱三个字段，用空格分隔。



The screenshot shows a VS Code editor window with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'WEEK04' with files: .gitignore, contacts.txt, environment.yml, LICENSE, and README.md. The code editor shows the content of 'contacts.txt' with 8 lines of contact information, each containing a name, gender, and email address separated by spaces.

```
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
8
```

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，例如



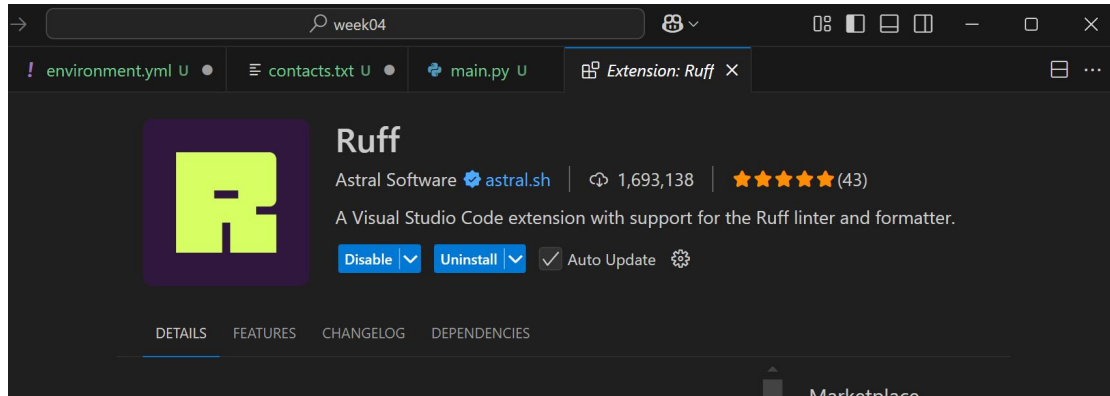
The screenshot shows a VS Code editor window with a file explorer on the left and a code editor on the right. The file explorer shows the same project as before, but now with 'main.py' added. The code editor shows the content of 'main.py' with Python code that reads 'contacts.txt', processes the data, and writes the output to 'emails.txt'.

```
1 def read_contacts(file_path):
2     try:
3         with open(file_path, 'r', encoding='utf-8') as file:
4             contacts = []
5             for line in file:
6                 parts = line.strip().split()
7                 if len(parts) == 3:
8                     name, gender, email = parts
9                     contacts.append((name, gender, email))
10            return contacts
11    except FileNotFoundError:
12        print(f"错误: 文件 {file_path} 未找到。")
13        return []
14
15
16 def generate_emails(contacts):
17     emails = []
18     for name, gender, email in contacts:
19         title = "先生" if gender == "男" else "女士"
20         email_text = f"to: <{email}>\n尊敬的 {name}{title}, 您的会员资格即将到期, 请及时续费"
21         emails.append(email_text)
22     return emails
23
24
25 def sort_contacts(contacts):
26     return sorted(contacts, key=lambda x: (x[2].split('@')[1], x[2].split('@')[0]))
27
28
29 def write_emails(emails, output_file):
30     try:
31         with open(output_file, 'w', encoding='utf-8') as file:
32             for email in emails:
33                 file.write(email + '\n')
34     except Exception as e:
35         print(f"错误: 写入文件 {output_file} 时出错: {e}")
36
```

5、可以将以上“任务要求”的文本，复制粘贴到大模型（比如豆包、DeepSeek）里，请 AI 来帮助编写程序初稿

6、AI 回复的只是静态代码，而且可能含有错误，所以我们必须在 Conda 环境里运行代码，逐行调试，检查每一行代码的运行都符合我们的期望（越是初学者越应该慢慢调试、检查、试验，借此学习）

在 VS Code 扩展商店里安装 Ruff 扩展，按照文档配置 Ruff，实现在保存 .py 文件时能够自动规范化 Python 代码：帮助我们 text editor，就这个文本编辑器，编写这个 python 的脚本的时候，可以给我们自动做规范。



运行 `python main.py` 命令（作用是启动 Python 解释器，执行 main.py 里的代码直至结束 (EOF) 或报错 (Exception)），检查运行结果是否符合预期。

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）。

```
lenovo@LAPTOP-UDBV5M8K MINGW64 ~/repo/week04 (main)
$ python -m pdb main.py
> c:\users\lenovo\repo\week04\main.py(1)<module>()
-> def read_contacts(file_path):
(Pdb)

(Pdb) 1
1 -> def read_contacts(file_path):
2     try:
3         with open(file_path, 'r', encoding='utf-8') as file:
4             contacts = []
5             for line in file:
6                 parts = line.strip().split()
7                 if len(parts) == 3:
8                     name, gender, email = parts
9                     contacts.append((name, gender, email))
10            return contacts
11        except FileNotFoundError:
(Pdb)
```

```

(Pdb) n
> c:\users\lenovo\repo\week04\main.py(16)<module>()
-> def generate_emails(contacts):
(Pdb) ll
1     def read_contacts(file_path):
2         try:
3             with open(file_path, 'r', encoding='utf-8') as file:
4                 contacts = []
5                 for line in file:
6                     parts = line.strip().split()
7                     if len(parts) == 3:
8                         name, gender, email = parts
9                         contacts.append((name, gender, email))
10                return contacts
11        except FileNotFoundError:
12            print(f"错误: 文件 {file_path} 未找到。")
13            return []
14
15
16 -> def generate_emails(contacts):
17     emails = []
18     for name, gender, email in contacts:
19         title = "先生" if gender == "男" else "女士"
20         email_text = f"to: <{email}>\n尊敬的 {name}{title}, 您的会
将到期, 请及时续费。 \n---"
21         emails.append(email_text)
22     return emails
23

```

```

(Pdb) p contacts
*** NameError: name 'contacts' is not defined
(Pdb) 

```

```

(Pdb) p read_contacts
<function read_contacts at 0x0000026BF3B73740>
(Pdb) s
> c:\users\lenovo\repo\week04\main.py(25)<module>()
-> def sort_contacts(contacts):
(Pdb) p sort_contacts
*** NameError: name 'sort_contacts' is not defined
(Pdb) ll
20         email_text = f"to: <{email}>\n尊敬的 {name}{title}, 您的会
将到期, 请及时续费。 \n---"
21         emails.append(email_text)
22         return emails
23
24
25 -> def sort_contacts(contacts):
26     return sorted(contacts, key=lambda x: (x[2].split('@')[1], x[
@')[0]))
27
28
29     def write_emails(emails, output_file):
30         try:
(Pdb) 

```

在调试过程中, 利用 `wat-inspector` (第三方软件包, 需要安装) 检查 (inspect) 各种对象 (参考文档)

```
done
#
# To activate this environment, use
#
#     $ conda activate prj1
#
# To deactivate an active environment, use
#
#     $ conda deactivate
```

在调试过程中，观察代码逐步运行的效果，学习理解以下 Python 基本概念 (建议观看下面的录播讲解)

- **Python 语法保留字 (reserved key words):** 在 Python 里，保留字也叫关键字，它们是被 Python 语言赋予特定含义的单词，在编程时不能把这些单词用作变量名、函数名或其他标识符。
- **语句 (statement) 和表达式 (expression):** 表达式 (expression) 定义：表达式是由值、变量、运算符以及函数调用组合而成的一段代码，它会计算出一个结果。表达式可以作为其他表达式的一部分，也能作为语句的一部分。语句 (statement) 定义：语句是 Python 程序中的一个执行单元，它会执行某种操作，但不一定会返回一个值。语句可以改变程序的状态，像变量赋值、控制程序流程等。
- **缩进 (indent):** 它用于明确代码块的界限。
- **局部变量 (local variable)、全局变量 (global variable)、LEGB 规则:** 局部变量 (Local Variable) 定义：局部变量是在函数或代码块内部定义的变量，它的作用域仅限于定义它的函数或代码块。也就是说，在函数外部无法访问这些变量。全局变量 (Global Variable) 定义：全局变量是在函数外部定义的变量，它的作用域是整个程序。全局变量可以在程序的任何地方被访问，但如果要在函数内部修改全局变量的值，需要使用 global 关键字进行声明。LEGB 规则定义：LEGB 规则是 Python 中用于查找变量名的优先级顺序，它代表了不同作用域的层级关系。当 Python 解释器遇到一个变量名时，会按照以下顺序在不同的作用域中查找该变量：**Local (局部作用域):** 函数内部的作用域。当在函数内部使用一个变量时，Python 首先会在该函数的局部作用域中查找。**Enclosing (闭包作用域):** 如果一个函数嵌套在另一个函数内部，那么外部函数的作用域就是内部函数的闭包作用域。当在内部函数中找不到某个变量时，Python 会在闭包作用域中查找。

Global（全局作用域）：模块级别的作用域。如果在局部作用域和闭包作用域中都找不到变量，Python 会在全局作用域中查找。**Built-in（内置作用域）：**Python 内置的函数和变量所在的作用域。如果在前面三个作用域中都没有找到变量，Python 会在内置作用域中查找。

- **函数 (function) 的定义 (define) 和调用 (call)：**在 Python 里，函数是一段有特定功能、可重复使用的代码块。定义好函数之后，就可以在其他地方调用该函数来执行其功能。函数调用的基本语法是在函数名后面加上括号，如果函数有参数，要在括号内传入相应的参数值。
- **字面值 (literal) (字符串 (str)、整数 (int)、列表 (list)、字典 (dict)、元组 (tuple))：**字符串是由零个或多个字符组成的不可变序列，在 Python 中可以用单引号、双引号或者三引号来表示。整数是没有小数部分的数字，可以是正数、负数或零。在 Python 中，直接写出数字即可表示整数。列表是一种可变的有序序列，用方括号 [] 表示，元素之间用逗号分隔。列表中的元素可以是不同的数据类型。字典是一种无序的键值对集合，用花括号 {} 表示，每个键值对之间用逗号分隔，键和值之间用冒号 : 分隔。字典中的键必须是不可变类型（如字符串、整数、元组等），值可以是任意数据类型。元组是一种不可变的有序序列，用圆括号 () 表示，元素之间用逗号分隔。和列表类似，元组中的元素可以是不同的数据类型。
- **运算符 (operator)：**在 Python 中，运算符是用于执行各种操作的特殊符号。**形参 (parameter)、实参 (argument)、返回值 (return value)：****形参 (Parameter) 定义：**形参也叫形式参数，是在定义函数时，函数名后面括号里指定的变量。形参就像是函数内部使用的占位符，代表了函数期望接收的数据。在函数定义时，形参并没有实际的值，只有在函数被调用时，才会被赋予具体的值。**实参 (Argument) 定义：**实参也叫实际参数，是在调用函数时，传递给函数的具体值。实参是用来给形参赋值的，当函数被调用时，实参的值会被传递给对应的形参。**返回值 (Return Value) 定义：**返回值是函数执行完毕后返回给调用者的结果。在函数体中，可以使用 return 语句来指定函数的返回值。当函数执行到 return 语句时，函数会立即结束，并将 return 后面的值返回给调用者。如果函数没有 return 语句，或者 return 后面没有跟任何值，函数会返回 None。
- **对象 (object)、类型 (type)、属性 (attribute)、方法 (method)：****对象 (Object) 定义：**对象是类的实例。在 Python 里，几乎所有事物都是对象，像数字、字符串、列表等。对象是数据（属性）和操作这些数据的方法的集合。**类型 (Type) 定义：**类型定义了对

对象的特征和行为。每个对象都有一个类型，类型决定了对象可以执行哪些操作以及如何存储数据。Python 提供了内置类型，如 `int`、`str`、`list` 等，同时也允许用户自定义类型（类）。属性（**Attribute**）定义：属性是与对象关联的数据。属性可以是变量，用来存储对象的状态信息。对象的属性可以通过点号（`.`）来访问。方法（**Method**）定义：方法是与对象关联的函数，用于执行特定的操作。方法定义在类中，通过对象来调用。方法可以访问和修改对象的属性。