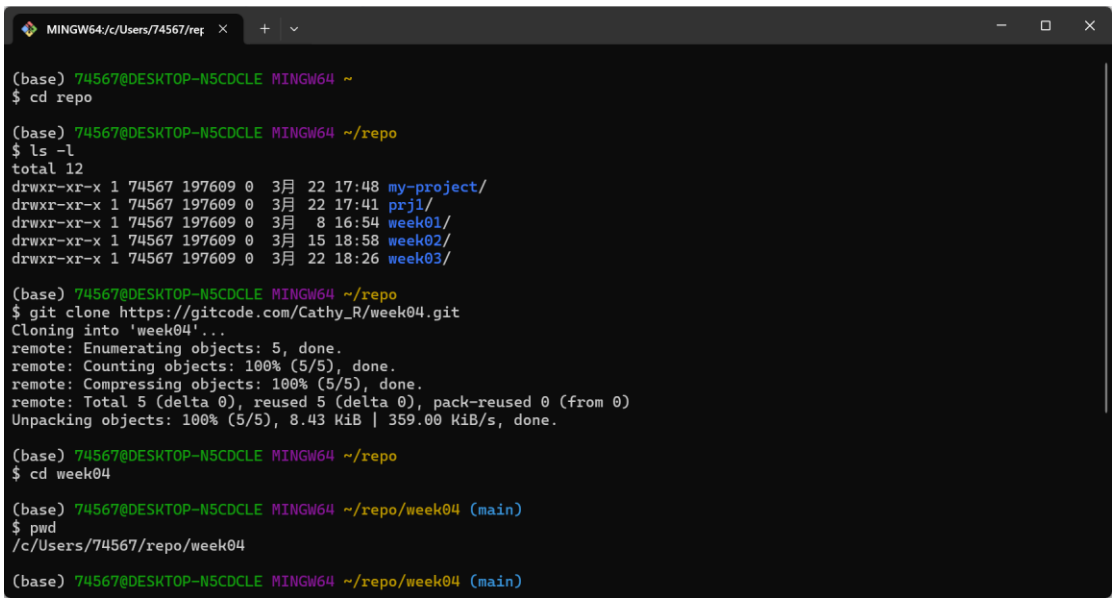
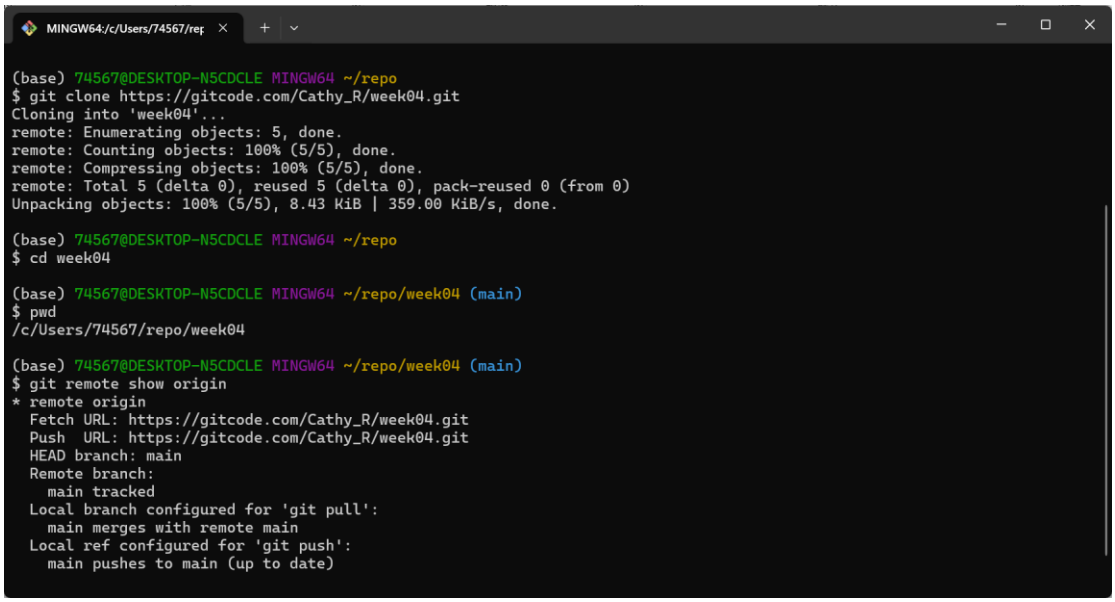
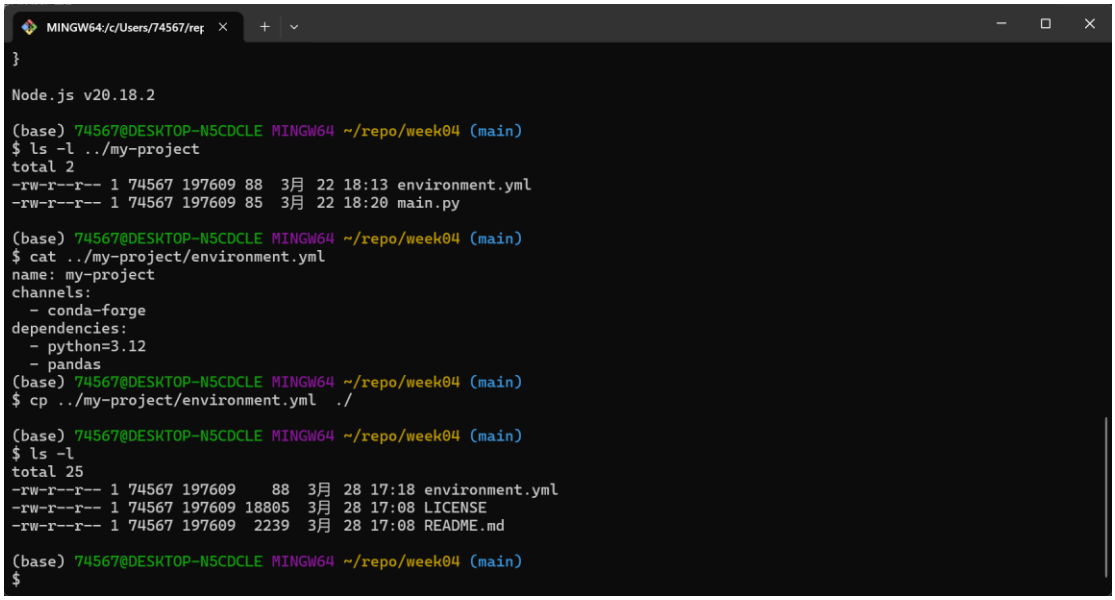
第四周学习报告

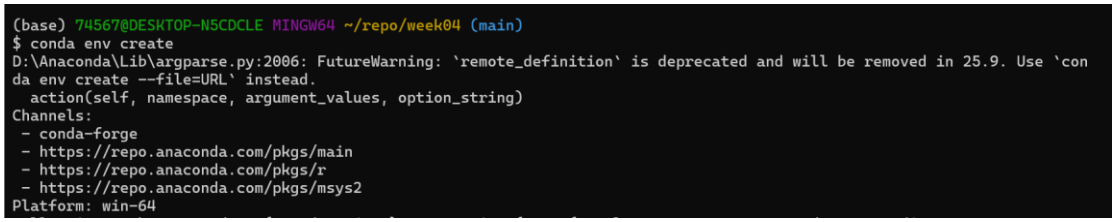
1. Fork 第 04 周打卡仓库至你的名下，然后将你名下的这个仓库 Clone 到你的本地计算机。



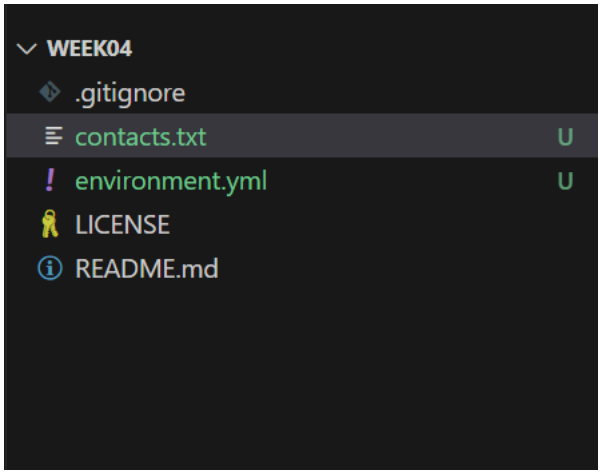


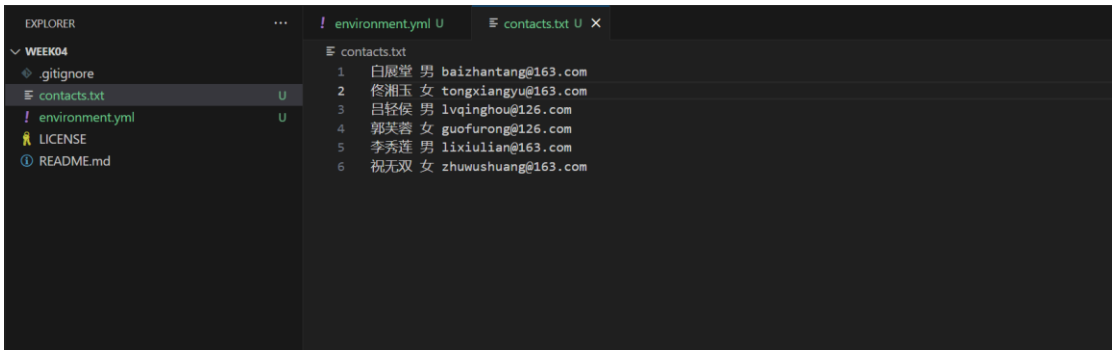
2. 用 VS Code 打开项目目录，新建一个 environment.yml 文件，指定安装 Python 3.12，然后运行 conda env create 命令创建 Conda 环境。





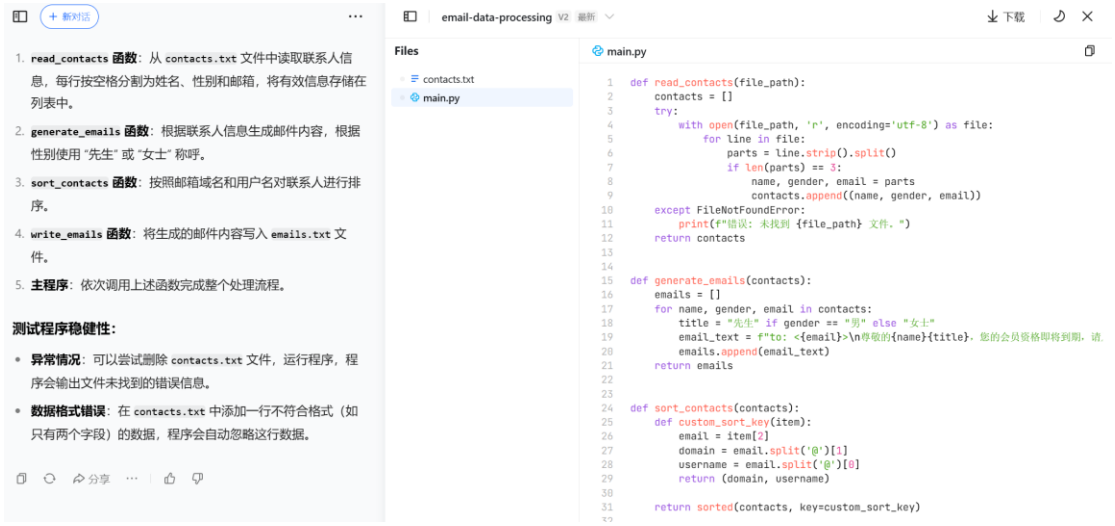
3. 新建一个 contacts.txt 文件，每行写一个联系人，每个联系人都包含姓名、性别、邮箱三个字段，用空格分隔。

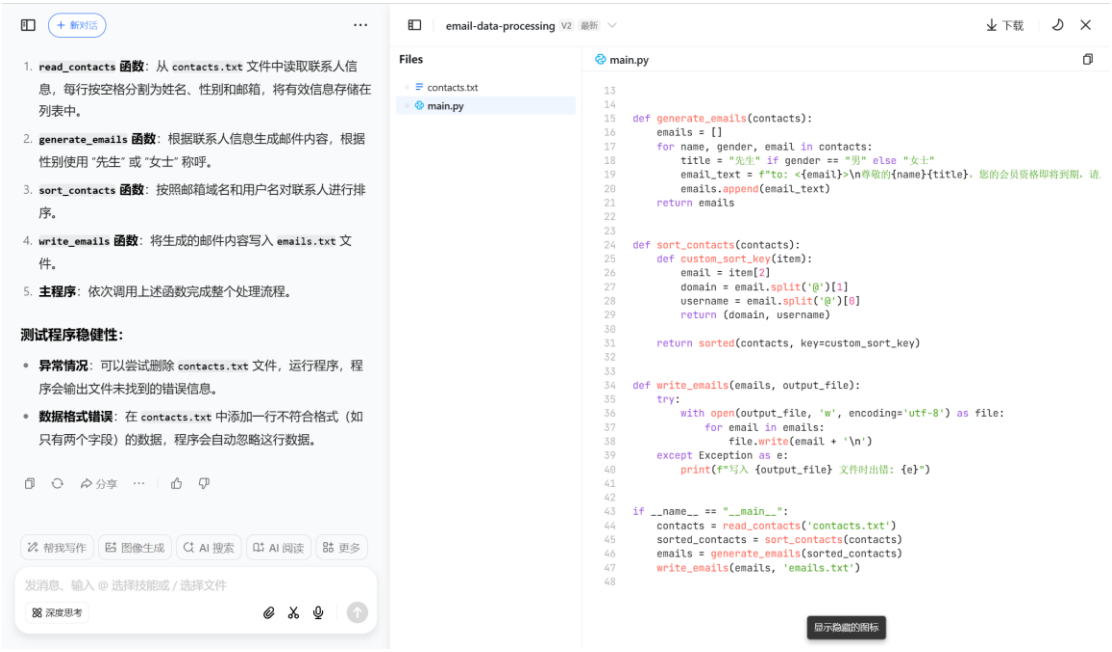




4. 新建一个 main.py 文件，里面写 Python 代码，要求读取 contacts.txt 文件的内容，进行数据处理后，输出一个 emails.txt 文件。

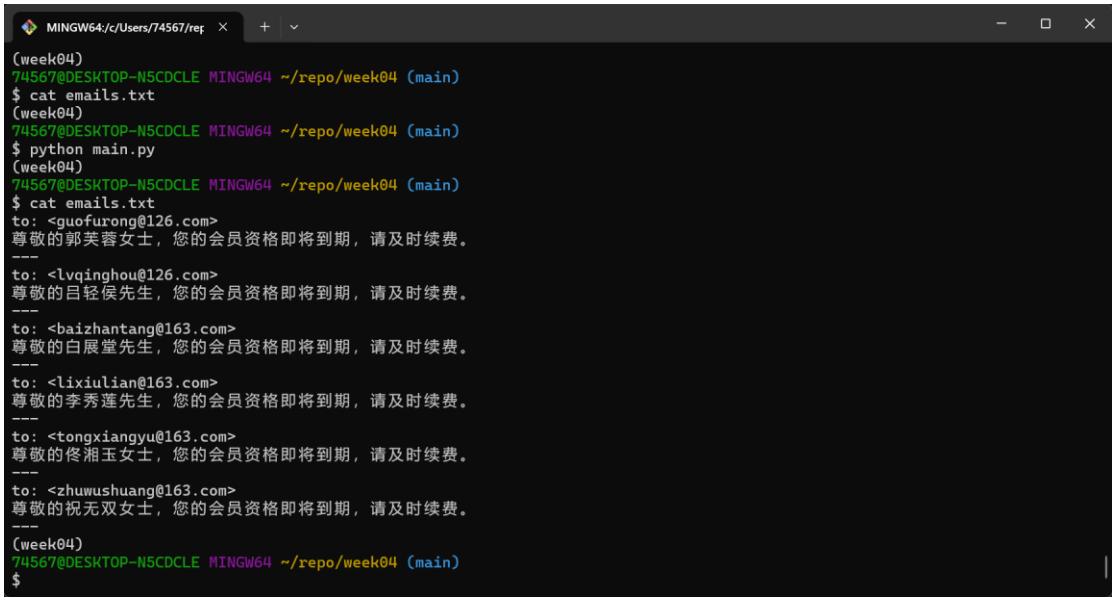
5. 可以将以上“任务要求”的文本，复制粘贴到大模型（比如豆包、DeepSeek）里，请 AI 来帮助编写程序初稿

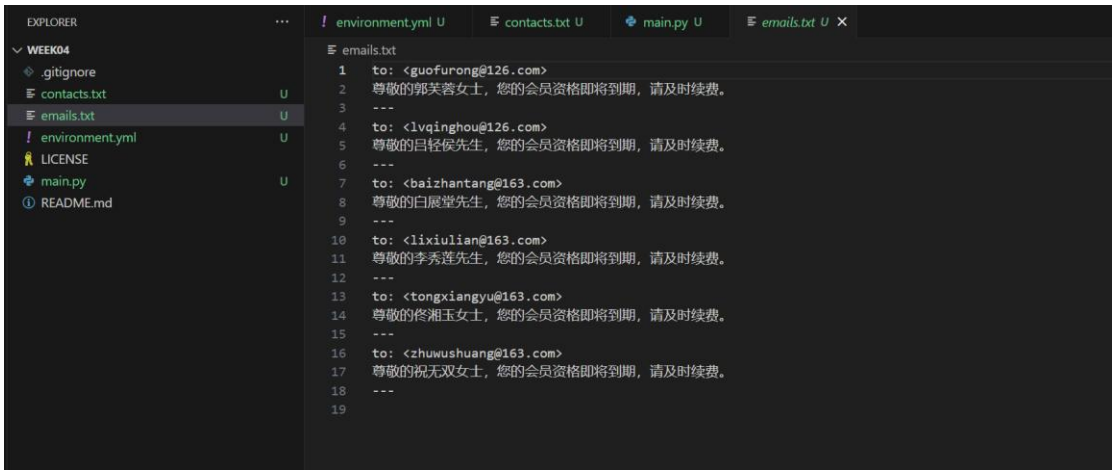




6. AI 回复的只是静态代码，而且可能含有错误，所以我们必须在 Conda 环境里运行代码，逐行调试，检查每一行代码的运行都符合我们的期望（越是初学者越应该慢慢调试、检查、试验，借此学习）

选择环境很重要：ctrl+shift+p 输入 interpreter，选择自己所需要建设的环境。





调试：$ python -m pdb main.py

L: list 显示代码

```
74567@DESKTOP-N5CDCLE MINGW64 ~/repo/week04 (main)
$ python -m pdb main.py
> c:\users\74567\repo\week04\main.py(1)<module>()
-> def read_contacts(file_path):
(Pdb) l
  1  -> def read_contacts(file_path):
  2         contacts = []
  3         try:
  4             with open(file_path, "r", encoding="utf-8") as file:
  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         except FileNotFoundError:
 11             print(f"错误：未找到 {file_path} 文件。")
(Pdb)
```

n: 执行当前行

```
(Pdb) n
> c:\users\74567\repo\week04\main.py(15)<module>()
-> def generate_emails(contacts):
```

p: print 打印表达式

```
(Pdb) p read_contacts
<function read_contacts at 0x0000018C4008F740>
```

s: 步入调用 step in

```
(Pdb) s
> c:\users\74567\repo\week04\main.py(24)<module>()
-> def sort_contacts(contacts):
```

pp: 美观打印

c：（继续执行）等命令

```
(Pdb) c
The program finished and will be restarted
> c:\users\74567\repo\week04\main.py(1)<module>()
-> def read_contacts(file_path):
```

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

environment.yml

```yaml
name: week04
channels:
  - conda-forge
dependencies:
  - python=3.12
  - wat-inspector
```

```
74567@DESKTOP-N5CDCLE MINGW64 ~/repo/week04 (main)
$ cat environment.yml
name: week04
channels:
  - conda-forge
dependencies:
  - python=3.12
  - wat-inspector(week04)
74567@DESKTOP-N5CDCLE MINGW64 ~/repo/week04 (main)
$ conda env update
D:\Anaconda\Lib\argparse.py:2006: FutureWarning: `remote_definition` is deprecated and will be removed in 25.9. Use `con
da env create --file=URL` instead.
  action(self, namespace, argument_values, option_string)
Channels:
 - conda-forge
 - https://repo.anaconda.com/pkgs/main
 - https://repo.anaconda.com/pkgs/r
 - https://repo.anaconda.com/pkgs/msys2
Platform: win-64
Collecting package metadata (repodata.json): done
Solving environment: done
#
# To activate this environment, use
#
#     $ conda activate week04
#
# To deactivate an active environment, use
#
#     $ conda deactivate
```

Python 语法保留字（reserved key words）

```
74567@DESKTOP-N5CDCLE MINGW64 ~/repo/week04 (main)
$ python
Python 3.12.9 | packaged by conda-forge | (main, Mar  4 2025, 22:37:18) [MSC v.1943 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> name ='Yiran Zhang'
>>> print(name)
Yiran Zhang
>>> def ='Yiran Zhang'
  File "<stdin>", line 1
    def ='Yiran Zhang'
        ^
SyntaxError: invalid syntax
>>> which ='Yiran Zhang'
>>> print(which)
Yiran Zhang
>>> which ='Yiran Zhang'
>>> while ='Yiran Zhang'
  File "<stdin>", line 1
    while ='Yiran Zhang'
          ^
SyntaxError: invalid syntax
```

Python 有 35 个保留字，它们是 Python 语言中具有特殊含义的单词，不能将它们用作变量名、函数名或其他标识符。以下是这些保留字的列表：

```plaintext
False      await      else       import     pass
None       break      except     in         raise
True       class      finally    is         return
and        continue   for        lambda     try
as         def        from       nonlocal   while
assert     del        global     not        with
async      elif       if         or         yield
```

这些保留字在 Python 中具有特定的用途和功能，例如 `if`、`else` 和 `elif` 用于条件语句；`for` 和 `while` 用于循环语句；`def` 用于定义函数等。在编写 Python 代码时，应避免使用这些保留字作为标识符。

语句（statement）和表达式（expression）

```python
def read_contacts(file_path):
    contacts = []
    try:
        with open(file_path, "r", encoding="utf-8") as file:
            for line in file:
                parts = line.strip().split()
                if len(parts) == 3:
                    name, gender, email = parts
                    contacts.append((name, gender, email))
    except FileNotFoundError:
        print(f"错误: 未找到 {file_path} 文件。")
    return contacts
```

缩进（indent）

```python
def sort_contacts(contacts):
    def custom_sort_key(item):
        email = item[2]
        domain = email.split("@")[1]
        username = email.split("@")[0]
        return (domain, username)
```

局部变量（local variable）、全局变量（global variable）、LEGB 规则

```
74567@DESKTOP-N5CDCLE MINGW64 ~/repo/week04 (main)
$ python -m pdb main.py
> c:\users\74567\repo\week04\main.py(1)<module>()
-> def read_contacts(file_path):
(Pdb) l
  1  -> def read_contacts(file_path):
  2         contacts = []
  3         try:
  4             with open(file_path, "r", encoding="utf-8") as file:
  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         except FileNotFoundError:
 11             print(f"错误：未找到 {file_path} 文件。")
(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
```

```
(Pdb) wat()
Local variables:
  __builtins__: dict = {…
  __file__: pdb._ScriptTarget = 'C:\Users\74567\repo\week04\main.py'
  __name__: str = '__main__'
  __pdb_convenience_variables: dict = {…
  __spec__: NoneType = None
  wat: wat.inspection.inspection.Wat = <WAT Inspector object>
(Pdb) p __file__
'C:\\Users\\74567\\repo\\week04\\main.py'
(Pdb) n
> c:\users\74567\repo\week04\main.py(15)<module>()
-> def generate_emails(contacts):
```

```
(Pdb) wat.globals
Global variables:
  __builtins__: dict = {…
  __file__: pdb._ScriptTarget = 'C:\Users\74567\repo\week04\main.py'
  __name__: str = '__main__'
  __pdb_convenience_variables: dict = {…
  __spec__: NoneType = None
  contacts: list = […
  emails: list = […
  generate_emails: function = <function generate_emails at 0x00000232B7CFF600>
  read_contacts: function = <function read_contacts at 0x00000232B7CFFD80>
  sort_contacts: function = <function sort_contacts at 0x00000232B7D1E200>
  sorted_contacts: list = […
  wat: wat.inspection.inspection.Wat = <WAT Inspector object>
  write_emails: function = <function write_emails at 0x00000232B7D1F880>
```

LEGB 规则总结

Python 变量查找时按照 LEGB 规则的顺序进行：

1. 首先在局部作用域（Local）中查找变量。
2. 若在局部作用域中未找到，就去闭包作用域（Enclosing）中查找。
3. 若在闭包作用域中也未找到，就去全局作用域（Global）中查找。
4. 若在全局作用域中还是未找到，最后去内置作用域（Built-in）中查找。

若在所有作用域中都找不到该变量，就会抛出 `NameError` 异常。

函数（function）的定义（define）和调用（call）

```
(Pdb) wat()
Local variables:
  __builtins__: dict = {…
  __file__: pdb._ScriptTarget = 'C:\Users\74567\repo\week04\main.py'
  __name__: str = '__main__'
  __pdb_convenience_variables: dict = {…
  __spec__: NoneType = None
  contacts: list = […
  generate_emails: function = <function generate_emails at 0x00000232B7CFF600>
  read_contacts: function = <function read_contacts at 0x00000232B7CFFD80>
  sort_contacts: function = <function sort_contacts at 0x00000232B7D1E200>
  wat: wat.inspection.inspection.Wat = <WAT Inspector object>
  write_emails: function = <function write_emails at 0x00000232B7D1F880>
```

```
(Pdb) s
--Call--
> c:\users\74567\repo\week04\main.py(24)sort_contacts()
-> def sort_contacts(contacts):
(Pdb) l
 19             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。\n---"
 20             emails.append(email_text)
 21         return emails
 22
 23
 24  -> def sort_contacts(contacts):
 25         def custom_sort_key(item):
 26             email = item[2]
 27             domain = email.split("@")[1]
 28             username = email.split("@")[0]
 29             return (domain, username)
(Pdb) wat()
Local variables:
  contacts: list = […
```

字面值（literal）（字符串（str）、整数（int）、列表（list）、字典（dict）、元组（tuple））

```
email = item[2]
domain = email.split("@")[1]
username = email.split("@")[0]
```

```
(Pdb) p {'a':1}
{'a': 1}
```

运算符（operator）

```
def sort_contacts(contacts):
    def custom_sort_key(item):
        email = item[2]
        domain = email.split("@")[1]
        username = email.split("@")[0]
        return (domain, username)

    return sorted(contacts, key=custom_sort_key)
```

```
 contacts.append((name, gender, email))
```

形参（parameter）、实参（argument）、返回值（return value）

形参：

```
def sort_contacts(contacts):
    def custom_sort_key(item):
        email = item[2]
        domain = email.split("@")[1]
        username = email.split("@")[0]
        return (domain, username)

    return sorted(contacts, key=custom_sort_key)
```

实参：

```
 contacts = read_contacts("contacts.txt")
```

返回值：

```
  return emails
```

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

```
(Pdb) wat /emails
value: [
    '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。
---',
]
type: list
len: 6

Public attributes:
  def append(object, /) # Append object to the end of the list.
```

```
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.…
```