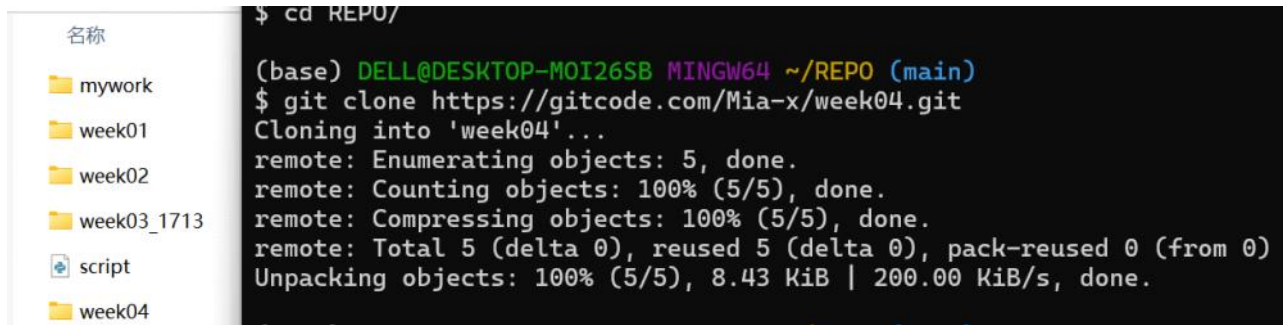


第四周-python数据类型

2025年4月3日 14:32

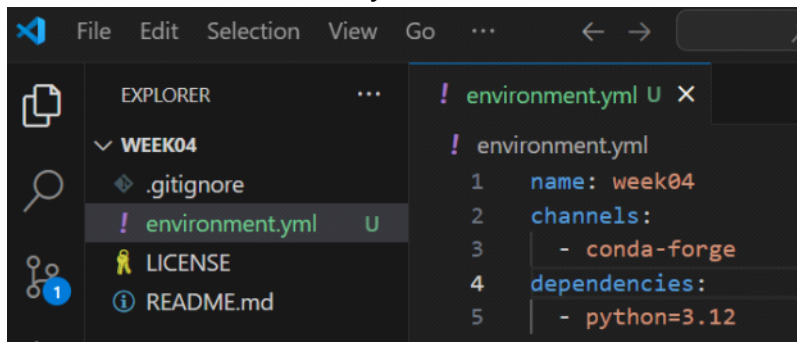
1、Fork [第04周打卡](#) 仓库至你的名下，然后将你名下的这个仓库 Clone 到你的本地计算机



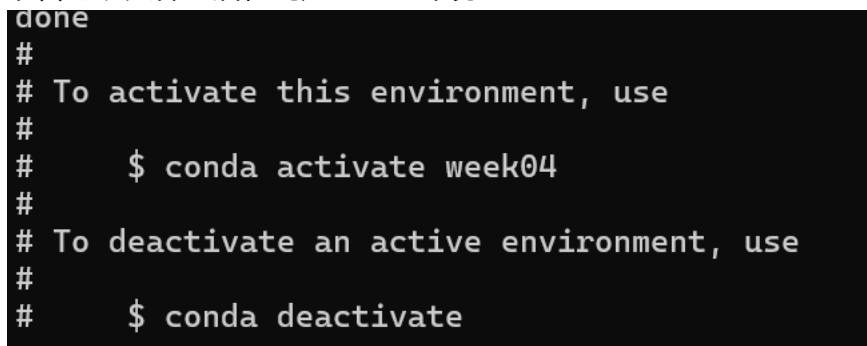
然后可以使用 “git remote show origin” 查看远程仓库的地址

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

使用 “touch environment.yml” 命令创建该文件



准备好该文件之后，创建choda环境 “conda env create”



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。要求输出是先按邮箱域名排序 (126.com 排在 163.com 之前)，然后再按邮箱用户名排序 (guofurong 排在 lvqinghou 之前)。

变成能够实现批量化的工作，通过“读取、计算和写入”三步走，完成任务。

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

将作业要求粘贴到deepseek中，要求它给出代码

终端创建environment文件夹教程

```
bash
吕松谋 男 lvqinghou@126.com
郭芙蓉 女 guofurong@126.com
李秀莲 男 lixiulian@163.com
祝无双 女 zhuwushuang@163.com" > contacts.txt
```

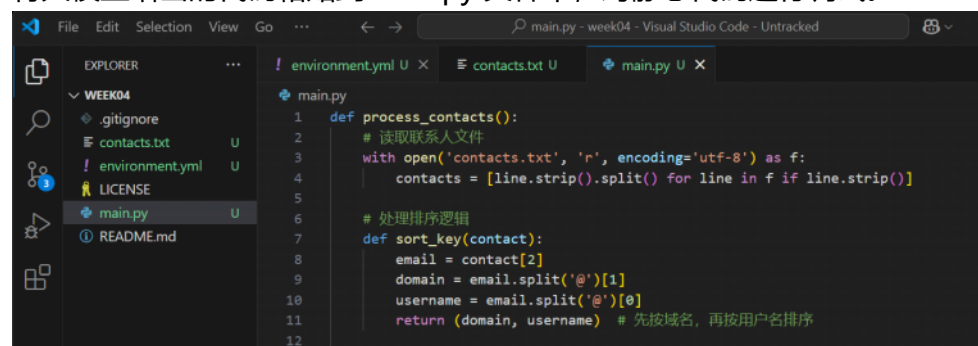
2. 编写 main.py 处理程序:

```
python
def process_contacts():
    # 读取联系人文件
    with open('contacts.txt', 'r', encoding='utf-8') as f:
        contacts = [line.strip().split() for line in f if line.strip()]

    # 处理排序逻辑
    def sort_key(contact):
        email = contact[2]
        domain = email.split('@')[1]
        username = email.split('@')[0]
        return (domain, username) # 先按域名，再按用户名排序

    sorted_contacts = sorted(contacts, key=sort_key)
```

将大模型给出的代码粘贴到main.py 文件中，对静态代码进行调试。



首先更改一下conda环境，将原来的base解释器切换到week04的解释器下。

```
if __name__ == '__main__':
```

minerals... Screen Reader Optimized Ln 31, Col 19 Spaces: 4 UTF-8 CRLF {} Python 3.12.9 (week04: conda)

使用终端运行main.py文件中的代码，生成了emails.txt，使用ls -l命令查看，结果如图：

```
(week04)
DELL@DESKTOP-MOI26SB MINGW64 ~/repo/week04 (main)
$ python main.py
(week04)
DELL@DESKTOP-MOI26SB MINGW64 ~/repo/week04 (main)
$ ls -l
total 31
-rw-r--r-- 1 DELL 197121 18805 Apr  3 14:50 LICENSE
-rw-r--r-- 1 DELL 197121  2239 Apr  3 14:50 README.md
-rw-r--r-- 1 DELL 197121   202 Apr  3 15:19 contacts.txt
-rw-r--r-- 1 DELL 197121   666 Apr  3 16:13 emails.txt
-rw-r--r-- 1 DELL 197121    73 Apr  3 15:07 environment.yml
-rw-r--r-- 1 DELL 197121  1121 Apr  3 16:13 main.py
```

然后使用cat命令查看该文件的内容，确为题目要求，并且排序正确

```
(week04)
DELL@DESKTOP-M0I26SB MINGW64 ~/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: <lixuilian@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。
---
```

AI可以根据我们的要求输出可以运行出预期结果的代码，但我们必须要理解每一行代码是做什么的，我们要掌握AI，而不是被替代。

6、进入调试

使用命令“python -m pdb main.py”进入调试模式，然后在（pdb）提示符下练习使用 l（显示代码）、n（执行当前行）、p（打印表达式）、s（步入调用）、pp（美观打印）、c（继续执行）等命令显示代码，如图所示

```
$ python -m pdb main.py
> c:\users\dell\repo\week04\main.py(1)<module>()
-> def process_contacts():
(Pdb) l
1  -> def process_contacts():
2      # 读取联系人文件
3      with open('contacts.txt', 'r', encoding='utf-8') as f:
4          contacts = [line.strip().split() for line in f if line.st
5
6      # 处理排序逻辑
7      def sort_key(contact):
8          email = contact[2]
9          domain = email.split('@')[1]
10         username = email.split('@')[0]
11         return (domain, username) # 先按域名，再按用户名排序
(Pdb)
```

n执行当前行，显示当前执行到第29行，执行l可现实当前行上下几行，继续执行ll，可显示所有代码。

```
(Pdb) n
> c:\users\dell\repo\week04\main.py(29)<module>()
-> if __name__ == '__main__':
(Pdb)
```

P命令为显示表达式，冒号之后的所有意为定义了一个函数

```

(Pdb) l
1      def process_contacts():
2          # 读取联系人文件
3          with open('contacts.txt', 'r', encoding='utf-8') as f:
4              contacts = [line.strip().split() for line in f if line]
5
6          # 处理排序逻辑
7          def sort_key(contact):
8              email = contact[2]
9              domain = email.split('@')[1]
10             username = email.split('@')[0]
11             return (domain, username) # 先按域名，再按用户名排序
12
13         sorted_contacts = sorted(contacts, key=sort_key)
14
15         # 生成邮件内容
16         email_template = "to: <{email}>\n尊敬的{name}{title}, 您的会员资格即"
17
18         with open('emails.txt', 'w', encoding='utf-8') as f:
19             for contact in sorted_contacts:
20                 name, gender, email = contact
21                 title = '女士' if gender == '女' else '先生'
22                 f.write(email_template.format(
23                     email=email,
24                     name=name,
25                     title=title
26                 ))
27                 f.write('\n') # 确保每个联系人之间有空行
28
29     -> if __name__ == '__main__':
30         process_contacts()
31

```

所以可以用p查看process_contacts，运行之后会显示变量的一个内存地址

```

(Pdb) p process_contacts
<function process_contacts at 0x000000210918C3880>

```

如果使用p查看sort_key会显示错误，原因在于这是一个函数，而p命令是打印变量的表达式，对于函数可以直接输入函数名，按enter，此时会显示函数的内存地址。若仍然出现如下情况：

```

(Pdb) sort_key
*** NameError: name 'sort_key' is not defined
(Pdb) p sort_key(['张三', '男', 'zhangsan@example.com'])
*** NameError: name 'sort_key' is not defined

```

原因在于sort_key的定义未在当前调试位置的作用域中，所以需要使用s命令步入sort_key的函数内部，直接输入函数名回车才可以显示其内存地址。

```

31
[EOF]
(Pdb) s
> c:\users\dell\repo\week04\main.py(7)process_contacts()
-> def sort_key(contact):
(Pdb) s
> c:\users\dell\repo\week04\main.py(13)process_contacts()
-> sorted_contacts = sorted(contacts, key=sort_key)
(Pdb) sort_key
<function process_contacts.<locals>.sort_key at 0x000000149F29C3C40>
(Pdb)

```

s表示步入，即步入到函数内部中，n执行当前行在第29行，s执行后显示在30行。

```

(Pdb) s
> c:\users\dell\repo\week04\main.py(30)<module>()
-> process_contacts()
(Pdb)

```

老师的示例中，n为17行，s运行之后为26行的原因在于，n执行之后定义这个函数，冒号后的部分为一个函数，n只是把这个里边的这些代码包装起来，作为一个函数的对象放在内存里边就

完了，它并不运行。我也通过deepseek了解了这两个命令之间的区别：

对比总结

命令	含义	是否进入函数内部	适用场景
n	下一步 (跳过)	✗ 否	快速跳过无需调试的函数调用。
s	步入	✓ 是	需要深入调试函数内部逻辑。

命令l表示查看当前运行行（箭头所示行）的上下5行

```
(Pdb) n
> c:\users\dell\repo\week04\main.py(16)process_contacts()
-> email_template = "to: <{email}>\n尊敬的{name}{title}, 您的会员资格即将到期, 请
(Pdb) l
11         return (domain, username) # 先按域名, 再按用户名排序
12
13     sorted_contacts = sorted(contacts, key=sort_key)
14
15     # 生成邮件内容
16 -> email_template = "to: <{email}>\n尊敬的{name}{title}, 您的会员资格即将到期, 请
17
18     with open('emails.txt', 'w', encoding='utf-8') as f:
19         for contact in sorted_contacts:
20             name, gender, email = contact
21             title = '女士' if gender == '女' else '先生'
(Pdb)
```

命令l可以确定你要查看的范围，例如“l 5,7”即查看5-7行的命令

```
(Pdb) l 5,7
5
6         # 处理排序逻辑
7         def sort_key(contact):
(Pdb)
```

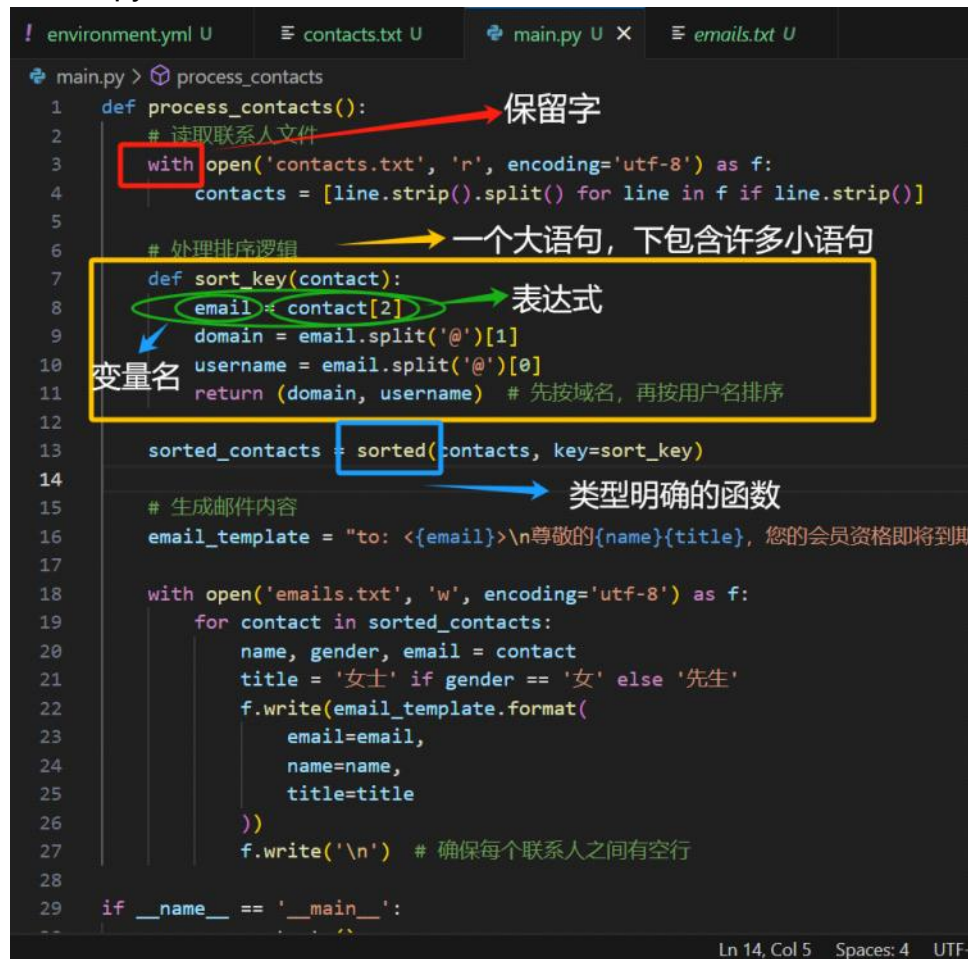
pp为美观打印，P type为打印该变量的类型，p len为打印该变量的长度。type和len都是系统自带的，所以只要进入到python中，这些命令自启动，不需要运行直接可以使用p打印，而其他命令行需得运行过之后才可以打印。使用S命令将循环语句都运行完之后，p sorted_contacts才可以输出按照正确要求排序的结果，如图所示：

```
(Pdb) p sorted_contacts
[['郭芙蓉', '女', 'guofurong@126.com'], ['吕轻侯', '男', 'lvqinghou@126.com'], ['李秀莲', '男', 'lixiulian@163.com'], ['佟湘玉', '女', 'tongxiangyu@163.com']]
(Pdb) pp
*** SyntaxError: invalid syntax
(Pdb) pp sorted_contacts
[['郭芙蓉', '女', 'guofurong@126.com'], ['吕轻侯', '男', 'lvqinghou@126.com'], ['白展堂', '男', 'baizhantang@163.com'], ['李秀莲', '男', 'lixiulian@163.com'], ['佟湘玉', '女', 'tongxiangyu@163.com'], ['祝无双', '女', 'zhuwushuang@163.com']]
(Pdb) p type(sorted_contacts)
<class 'list'>
(Pdb) p len(sorted_contacts)
6
```

c命令是直接运行到结束，调试完成之后q命令为推出调试模式


```
(Pdb) c
The program finished and will be restarted
> c:\users\dell\repo\week04\main.py(1)<module>()
-> def process_contacts():
(Pdb) q
(week04)
DELL@DESKTOP-M0I26SB MINGW64 ~/REPO/week04 (main)
$
```

7、学习python的一些基本概念



Python 语法保留字 (reserved key words)

Python 的保留字（也称为 关键字）是具有特殊含义的标识符，不能用作变量名、函数名或任何其他标识符。

语句 (statement) 和表达式 (expression)：语句里面包含表达式，而表达式是不能包含语句的。表达式可以嵌套。

缩进 (indent)，缩进代表层级，在写代码时要严格缩进，严格对齐。Python语法中是通过缩进对齐来确定子语句的边界

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

可以使用新安装的wat来查看局部变量

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

此时停在代码最开始运行的地方，未运行任何代码，调试器可能默认回退到全局变量显示（如 `__file__`, `process_contacts`）

使用 `n` 往下运行代码，就会显示出更多的局部变量：

```
(Pdb) n
> c:\users\dell\repo\week04\main.py(29)<module>()
-> if __name__ == '__main__':
(Pdb) wat()
Local variables:
  __builtins__: dict = {...
  __file__: pdb._ScriptTarget = 'C:\Users\DELL\REPO\week04\main.py'
  __name__: str = '__main__'
  __pdb_convenience_variables: dict = {...
  __spec__: NoneType = None
  process_contacts: function = <function process_contacts at 0x0000024C56613600>
  wat: wat.inspection.inspection.Wat = <WAT Inspector object>
```

若使用 `s` 进入某一函数内部，则能看到该函数的局部变量。

```
(Pdb) s
> c:\users\dell\repo\week04\main.py(4)process_contacts()
-> contacts = [line.strip().split() for line in f if line.strip()]
(Pdb) wat()
*** SyntaxError: '(' was never closed
(Pdb) wat()
Local variables:
  f: _io.TextIOWrapper = <_io.TextIOWrapper name='contacts.txt' mode=
(Pdb)
```

全局变量则为函数外部或者顶层模块定义的变量，用 `globals` 命令查看

```
(Pdb) wat.globals
Global variables:
  __builtins__: dict = {...
  __file__: pdb._ScriptTarget = 'C:\Users\DELL\REPO\week04\main.py'
  __name__: str = '__main__'
  __pdb_convenience_variables: dict = {...
  __spec__: NoneType = None
  process_contacts: function = <function process_contacts at 0x0000024C56613600>
  wat: wat.inspection.inspection.Wat = <WAT Inspector object>
(Pdb)
```

通过 `deepseek` 可以了解更多二者的区别：

3. 关键区别

特性	全局变量	局部变量
定义位置	函数外部或模块顶层	函数内部（含参数）
作用域	整个模块	仅限定义它的函数内部
生命周期	程序运行期间有效	函数调用期间有效
跨函数访问	可直接读取，修改需 <code>global</code>	其他函数无法直接访问
内存占用	长期占用内存	函数结束后释放

函数 (function) 的定义 (define) 和调用 (call)

操作	语法示例	说明
定义	<code>def func():</code>	使用 <code>def</code> 关键字
调用	<code>func()</code>	函数名 + 括号
参数	<code>def func(a, b=0, *args):</code>	支持默认参数和可变参数
返回值	<code>return result</code>	可返回任意数据
作用域	<code>global var</code>	局部变量 vs 全局变量

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

运算符 (operator):

以我的代码为例:

```
def process_contacts():
    # 读取联系人文件
    with open('contacts.txt', 'r', encoding='utf-8') as f:
        contacts = [line.strip().split() for line in f if line.strip()]

    # 处理排序逻辑
    def sort_key(contact):
        email = contact[2]
        domain = email.split('@')[1]
        username = email.split('@')[0]
        return (domain, username) # 先按域名, 再按用户名排序

    sorted_contacts = sorted(contacts, key=sort_key)
```

Annotations in the image:

- `def`: 形参 (Parameter)
- `'utf-8'`: 字符串字面值 (String literal)
- `==`: 赋值语句所用, 两个==才是运算符 (Assignment statement, == is the operator)
- `contact`: 字面值 (Literal)
- `sorted`: 返回值 (Return value)
- `key=sort_key`: 实参 (Argument)

```
# 生成邮件内容
email_template = "to: <{email}>\n尊敬的{name}{title}, 您的会员资格即将到期, 请及时续费.\n"

with open('emails.txt', 'w', encoding='utf-8') as f:
    for contact in sorted_contacts:
        name, gender, email = contact
        title = '女士' if gender == '女' else '先生'
        f.write(email_template.format(
            email=email,
            name=name,
            title=title
        ))
    f.write('\n') # 确保每个联系人之间有空行
```

Annotations in the image:

- `if gender == '女' else '先生'`: 运算符 (Operator)
- `f.write`: 名称访问运算符 (Name access operator)
- `sorted_contacts`: 名称访问运算符 (Name access operator)

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

形参是一个抽象的, 而实参是具体的

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

安装的wat可以深度的检查对象, 也可以调用对象

```
type: list
len: 6

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...

(Pdb) p sorted_contacts.append
<built-in method append of list object at 0x0000024C56657740>
(Pdb)
```

使用wat就可以深度访问对象的属性、方法


```
(Pdb) p sorted_contacts.append
<built-in method append of list object at 0x0000024C56657740>
(Pdb) wat / sorted_contacts.append

value: <built-in method append of list object at 0x0000024C56657740>
type: builtin_function_or_method
signature: def append(object, /)
"""Append object to the end of the list."""
```