

## 极简版计算机组成原理与操作系统

数据在不通电的情况下可以长期持久地 (persistently) 存储在 **磁盘** (如固态硬盘 SSD、机械硬盘 HDD) 或磁带 (常用于数据备份、长期归档) 里。但在需要呈现 (print、render、show、display、play)、计算加工 (compute、transform、analyze、machine learning、deep learning) 或编解码 (encode、decode) 时, 就需要通电的 **CPU** 和 **内存** (硬件), 在操作系统 (软件) 里以 **进程** (process) 为单元 (相互隔离) 进行处理。例如, Microsoft Word 启动后就是一个进程, 我们在 Word 进程里打开某个 .docx 文档, 将其从磁盘加载 (**读取**) 到内存, 然后在图形界面 (GUI) 里查看和编辑 (**计算**) 内存中的文档, 最后将内存数据保存 (**写入**) 到磁盘。

Python 解释器 (interpreter) (作用: 将代码转换成二进制, 以便 cpu 和内存的识别) 启动后也是一个进程, 她按照流程 (flow) 执行我们准备好的 Python 代码, 根据我们代码的要求, 转告 (即 **调用**, call) 操作系统或其他软件 (即 **依赖项**, dependency), 委托她们替我们执行各种“读取——计算——写入”等工作。

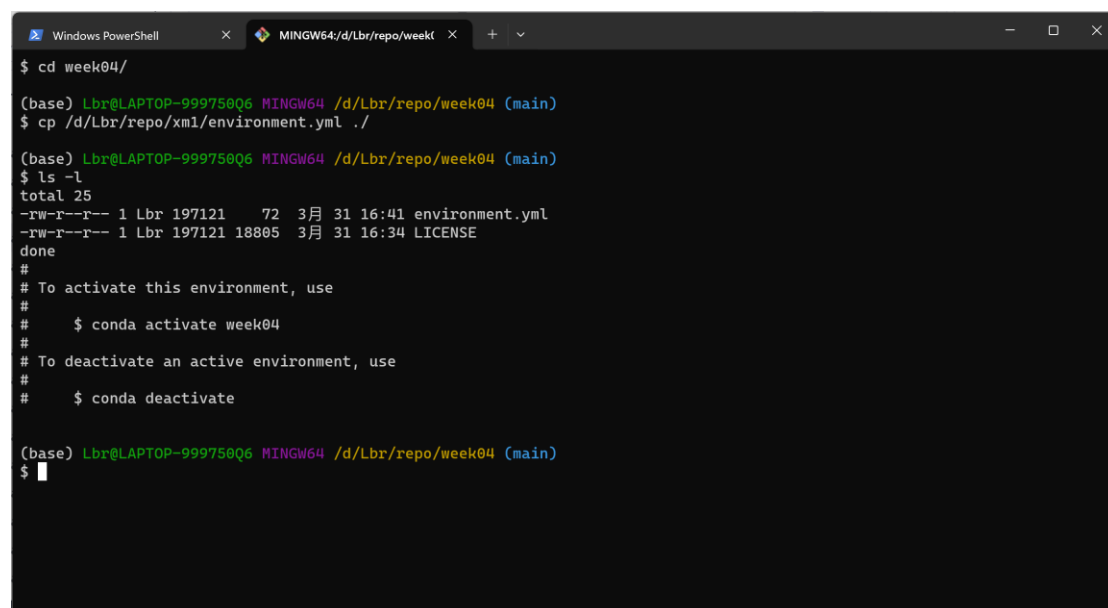
注: 每打一个命令都会启动一个进程, 该进程完成了工作以后就结束

需要注意的东西:

不需要完全理解依赖项内部的工作细节 (黑箱), 只需要清楚每个调用的主体 (即 **对象**, object) 是什么 **类型** (type), 每个调用的输入 (即 **参数**, parameter/argument)、输出 (即 **返回值**, return value) 是什么类型, 以及调用会对内存数据、磁盘文件做什么修改, 就足以支持我们自动批量地完成工作了。

**Python 本质: 拼接操纵各种对象 (胶水语言)**

**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

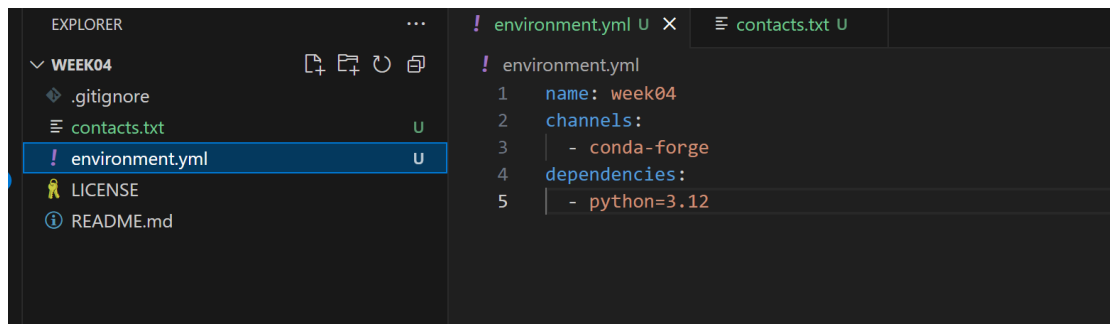


```
Windows PowerShell
MINGW64:/d/Lbr/repo/week04
$ cd week04/

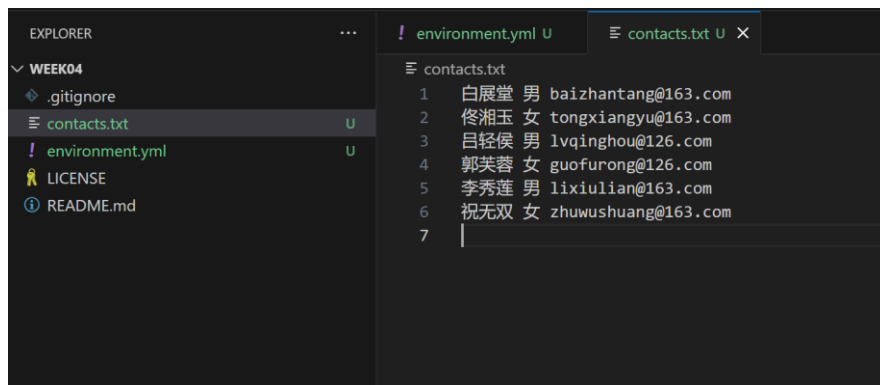
(base) Lbr@LAPTOP-999750Q6 MINGW64 /d/Lbr/repo/week04 (main)
$ cp /d/Lbr/repo/xml/environment.yml ./

(base) Lbr@LAPTOP-999750Q6 MINGW64 /d/Lbr/repo/week04 (main)
$ ls -l
total 25
-rw-r--r-- 1 Lbr 197121    72  3月 31 16:41 environment.yml
-rw-r--r-- 1 Lbr 197121 18805  3月 31 16:34 LICENSE
done
#
# To activate this environment, use
#
#   $ conda activate week04
#
# To deactivate an active environment, use
#
#   $ conda deactivate

(base) Lbr@LAPTOP-999750Q6 MINGW64 /d/Lbr/repo/week04 (main)
$
```



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



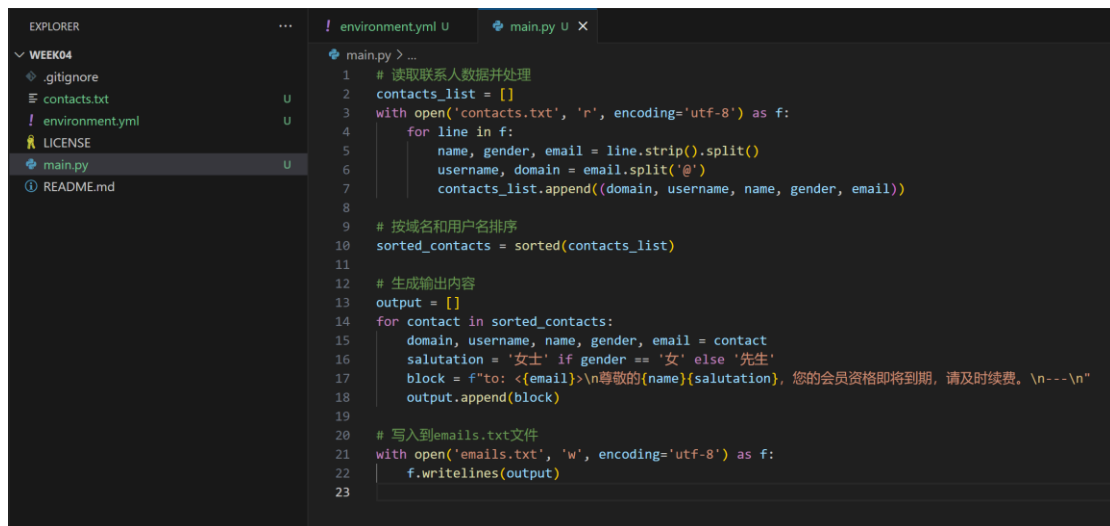
cat (contact) 命令：cat 文件名：读取文件内容      cat 文件 1 文件 2：将文件 1 和 2 拼接起来

注：unix 规范文本文件的最后加一个空行

## 静态代码的调试和运行

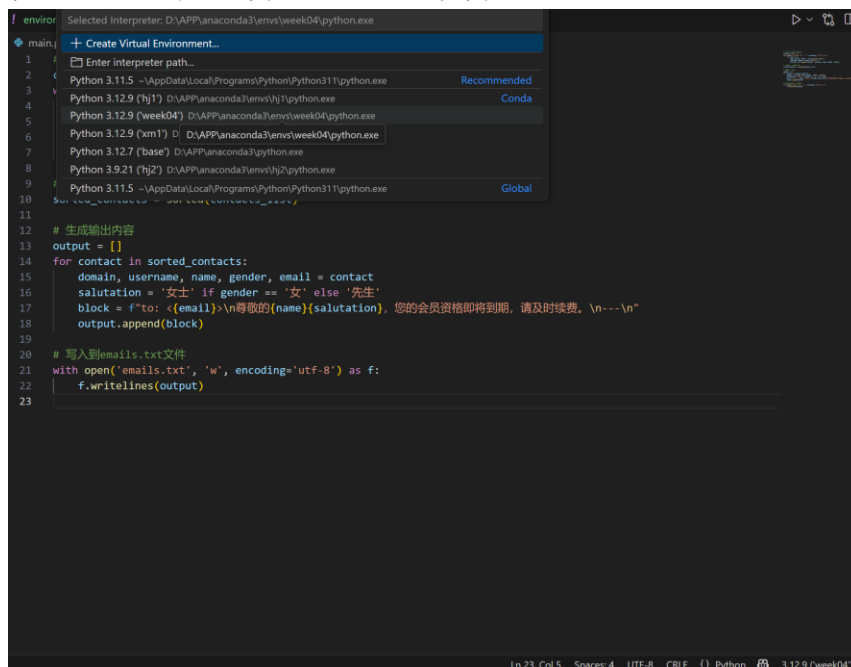
将以上“任务要求”的文本，复制粘贴到大模型（比如豆包、DeepSeek）里，请 AI 来帮助编写程序初稿

注：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



```
1 # 读取联系人数据并处理
2 contacts_list = []
3 with open('contacts.txt', 'r', encoding='utf-8') as f:
4     for line in f:
5         name, gender, email = line.strip().split()
6         username, domain = email.split('@')
7         contacts_list.append((domain, username, name, gender, email))
8
9 # 按域名和用户名排序
10 sorted_contacts = sorted(contacts_list)
11
12 # 生成输出内容
13 output = []
14 for contact in sorted_contacts:
15     domain, username, name, gender, email = contact
16     salutation = '女士' if gender == '女' else '先生'
17     block = f"to: <{email}>\n尊敬的{name}{salutation}, 您的会员资格即将到期, 请及时续费。\\n---\\n"
18     output.append(block)
19
20 # 写入到emails.txt文件
21 with open('emails.txt', 'w', encoding='utf-8') as f:
22     f.writelines(output)
23
```

在 VS code 中更改环境至 week04 的环境



```
12 # 生成输出内容
13 output = []
14 for contact in sorted_contacts:
15     domain, username, name, gender, email = contact
16     salutation = '女士' if gender == '女' else '先生'
17     block = f"to: <{email}>\n尊敬的{name}{salutation}, 您的会员资格即将到期, 请及时续费。\\n---\\n"
18     output.append(block)
19
20 # 写入到emails.txt文件
21 with open('emails.txt', 'w', encoding='utf-8') as f:
22     f.writelines(output)
23
```

启动 python 解释器并运行 main.py

```
Lbr@LAPTOP-999750Q6 MINGW64 /d/Lbr/repo/week04 (main)
$ python main.py
(week04)
```

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)

启动 python 解释器后, 先要加载 PDB 的模块 (调试器), 再运行这个 main.py

```
(week04)
Lbr@LAPTOP-999750Q6 MINGW64 /d/Lbr/repo/week04 (main)
$ python -m pdb main.py
> d:\lbr\repo\week04\main.py(2)<module>()
-> contacts_list = []
(Pdb)
```

## 理解几个 PDB 常用命令

l: 显示即将运行但是还没有运行的代码（即代码执行到的位置）

ll: 显示全部代码

ll: 显示当前运行位置的代码的上下五行

```
-> contacts_list = []
(Pdb) l
1      # 读取联系人数据并处理
2  -> contacts_list = []
3      with open('contacts.txt', 'r', encoding='utf-8') as f:
4          for line in f:
5              name, gender, email = line.strip().split()
6              username, domain = email.split('@')
7              contacts_list.append((domain, username, name, gender, email))
8
9      # 按域名和用户名排序
10     sorted_contacts = sorted(contacts_list)
11
```

n : 执行当前行

```
(Pdb) n
> d:\lbr\repo\week04\main.py(3)<module>()
-> with open('contacts.txt', 'r', encoding='utf-8') as f:
(Pdb) l
1      # 读取联系人数据并处理
2      contacts_list = []
3  -> with open('contacts.txt', 'r', encoding='utf-8') as f:
4          for line in f:
5              name, gender, email = line.strip().split()
6              username, domain = email.split('@')
7              contacts_list.append((domain, username, name, gender, email))
8
9      # 按域名和用户名排序
10     sorted_contacts = sorted(contacts_list)
```

p: 打印/显示表达式（已经运行过的）

pp : 美观打印

s : 步入调用

q: 退出解释器

```
(Pdb) q
(week04)
Lbr@LAPTOP-999750Q6 MINGW64 /d/Lbr/repo/week04 (main)
$
```

c : 继续执行

```
(Pdb) n
Post mortem debugger finished. The D:\Lbr\repo\week04\main.py will be restarted
> d:\lbr\repo\week04\main.py(2)<module>()
-> contacts_list = []
(Pdb) c
The program finished and will be restarted
> d:\lbr\repo\week04\main.py(2)<module>()
-> contacts_list = []
```

## Python 基本概念

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

作用：可深度检查 python 的对象

保留字：在 python 的语法上面有特殊的含义，不能乱用（例：for 和 which）

```
Lbr@LAPTOP-999750Q6 MINGW64 /d/Lbr/repo/week04 (main)
$ python
Python 3.12.10 | packaged by conda-forge | (main, Apr 16
Type "help", "copyright", "credits" or "license" for mor
>>> for = 'qwe'
      File "<stdin>", line 1
        for = 'qwe'
          ^
SyntaxError: invalid syntax
>>> which = 'qwe'
>>> print(which)
qwe
>>> █
```

注：import keyword, 然后 print keyword keyword list 就可以列出来所有的保留字

语句 (statement)

表达式 (expression)：类似于语言里面的一些词

关系：语句里包含表达式，而表达式不能包含语句

缩进 (indent)：可确定子语句（界定子语句的边界），代表了层级

局部变量 (local variable)：在当前的视野的范围内，能找到的变量（在调用函数后，才产生的变量，取决于运行到的位置）

全局变量 (global variable)：总是能够访问到的变量，任何地方都能反映到全局变量（不缩进）

LEGB 规则

Python 中变量作用域的查找顺序，用于确定程序中某个变量名 (identifier) 的引用来源。

LEGB 是四个作用域层级的缩写，按优先级从高到低依次为：

Local（局部作用域）

定义：当前函数或方法内部定义的变量。

特点：仅在函数执行期间存在，函数执行结束后销毁。

Enclosing（闭包作用域/嵌套函数外层作用域）

定义：嵌套函数中，外层函数（非全局）的作用域。

特点：适用于闭包（closure）场景，内层函数可以访问外层函数的变量。

Global（全局作用域）

定义：在模块（文件）顶层定义的变量。

特点：在整个模块内有效，需用 global 关键字在函数内修改。

Built-in（内置作用域）

定义：Python 内置的变量或函数（如 print, len, list 等）。

特点：当其他作用域找不到变量时，最后查找内置作用域。

Python 在查找变量时，按以下顺序逐级向上搜索：

1. Local → 当前函数内部。
2. Enclosing → 外层嵌套函数（如果存在）。
3. Global → 模块全局作用域。
4. Built-in → Python 内置作用域。