第四周

一、概念

（一）Workbook 是工作簿，指一个 EXCEL 表

（二）进程：

CPU 和内存在运用的时候，都是以进程为单元进行处理，而且要相互隔离来执行这一段计算的任务。例如：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。

（三）CPU：

操作系统的一个重要的工作就是要把这个 CPU 和内存要同时分享给多个进程来使用

二、如何用 Python 工作

1.把 environment.yml 复制粘贴到当前文件夹

2.然后 code .打开 vs code，输入安装包，保存

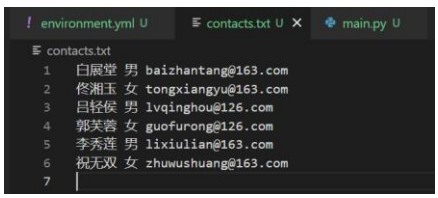
3.终端输入 conda env create

4. conda activate week04

5.code .(右下角调成 week04)

6.vs code 里创建 main.py，把老师的变成要求发给豆包，把豆包代码结果复制到 vs code 里的 main.py 里

vs code 里创建 contacts.txt，把老师的变成要求里的一些文本，复制到 vs code 里的 contacts.txt 里



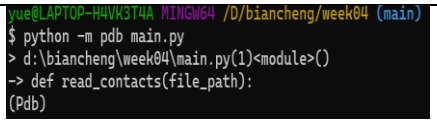
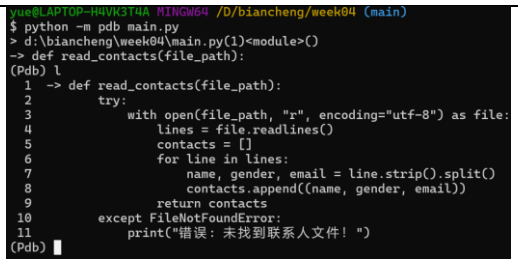
7. 运行程序 python main.py

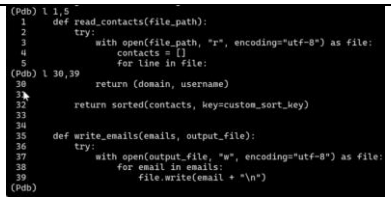
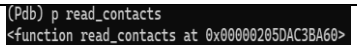
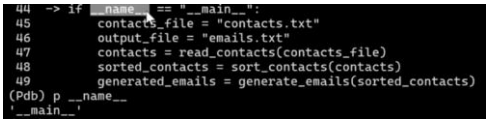
查看输出结果：cat emails.txt

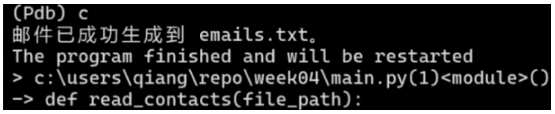
调试：

1 删除 emails.文件:rm emails.txt

2. 启动调试器: python -m pdb main.py（pdb 调试器；先启动调试器，调试模式 debug mode，准备执行 main.py 里的代码）

终端代码光标跳到开头	Ctrol+A	
终端代码光标跳到结尾	Ctrol+E	
新内容：验证概念，学会使用调试器 pdb		
准备调试器，准备用调试器运行 main.py 里的代码，（pdb）表示进入 pdb 语法	python -m pdb main.py	
	(pdb) 提示符下的常用代码，如下	
	l (显示代码)	 `-> def read_contacts(file_path):`：显示箭头指向的：即将运行，还没有运行的代码（目前到了代码的第一行） ll:显示所有代码 l . :当前这个箭头上下上面加五行，下面加五行,显示上下五行内容 l 1,5：显示代码的第 1 行到第五行

		
next	n (执行当前行)	 ：执行到了第 15 行。 再打 l:显示局部代码；打 ll:显示全部代码   注意：为什么一下到 15 行？因为 def 后边的冒号，缩进了，def 后面是一个语句，可以认为 read contacts 是一个变量
重复上一步操作：n	打完 n 后直接按回车	
print	p (打印表达式)	 P+变量: 就是问问这个变量是什么意思，一般会输出，变量的值（参数）  __name__这个变量的值是__main__  两个 Python 里本来就有的变量: Type ： 类型 Len:长度
Step in	s (步入调用)	n: 从这个房门就走过去(运行房门里的东西), 走到下一个房门 s:它就会走到这个门里面去，在这个门里边。

		Eg:即将运行 If 命令，还没运行  现在打 n:运行 if 并跳到下一行 contacts 再打 s:因为 read contacts 是一个上面已经运行过一次的函数，此时输入 S 就会  进入 read contacts 这个函数里面去；但如果此时输入 n,那就会跑一遍这个函数，并箭头之向函数的下一行。
	pp (美观打印)	 ‘白展堂’叫一个字符串， （）叫一个元祖 【】叫一个列表 上面的例子：三个字符串构成一个元祖，六个元祖构成一个列表
	c (继续执行)	C 一般会把所有代码运行到底，然后自动回到调试器的第一行 
	q 就是退出	退出 pdb 调试器

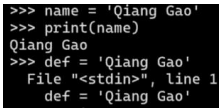
三、查询安装包

法一：package

conda-forge | community-driven packaging for conda

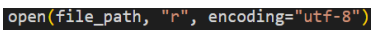
四、Python 基本概念（Python 语法）

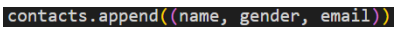
<table>
<tr>
<td></td>
<td>保留字
(reserved key words)</td>
<td>


保留字：蓝色的字
注意：
1 保留字有自己特殊的含义，如 for 是循环，def 是定义
2 保留字不能用作定义变量名


</td>
</tr>
<tr>
<td></td>
<td>语句 (statement)和表达式(expression)</td>
<td>

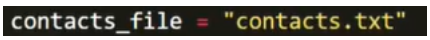

大语句：def
嵌套子语句：try 子语句、with 子语句

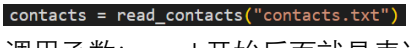
表达式：语句里面含有表达式




再比如：


If 语句中的


这个条件判断，就是表达式
前面的变量名__name__是表达式
后面的 main 也是表达式
这一整条也是表达式


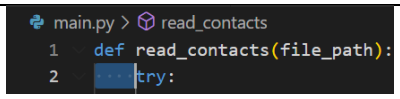
是一个赋值语句，赋值语句必须有等号，后面就是表达式


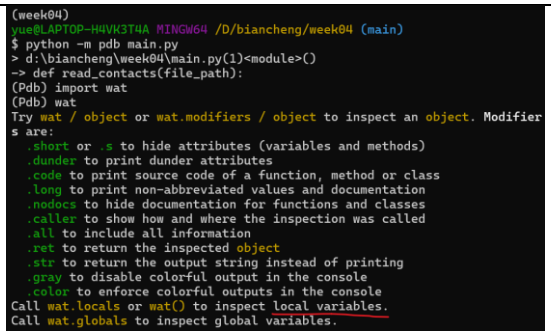
调用函数：read 开始后面就是表达式

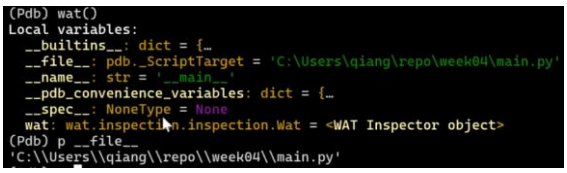

</td>
</tr>
</table>

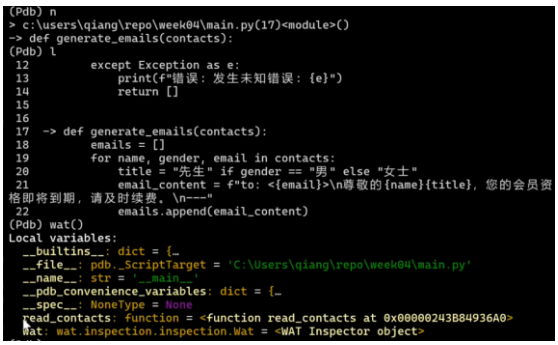
<table>
<tr><td></td><td></td><td>一个表达式自己也可以构成一个语句</td></tr>
<tr><td></td><td>变量、字符串</td><td>


白色：变量名
橙色：字符串
 ：转译
绿色：函数（可调用）


</td></tr>
<tr><td></td><td>缩进（indent）</td><td>


</td></tr>
<tr><td></td><td>局部变量（local variable）、
全局变量（global variable）、
LEGB 规则</td><td>


（注意要打：import wat 才有下面的东西哦）
画红线的东西：叫局部变量
后两句话：
当你打 wat.local 或 Wat()时，可以查看局部变量
当你打 wat.globals 时，可以查看全局变量


在当前的视野的范围内，它就能够找到这些变量（黄色），那 wat（）就叫局部变量


</td></tr>
</table>

输入 n:往下走一行

再打 wat(),局部变量里面就多了一个变量

```
 44      if __name__ == "__main__":
 45          contacts_file = "contacts.txt"
 46          output_file = "emails.txt"
 47  ->      contacts = read_contacts(contacts_file)
 48          sorted_contacts = sort_contacts(contacts)
 49          generated_emails = generate_emails(sorted_contacts)
 50          write_emails(generated_emails, output_file)
 51          print(f"邮件已成功生成到 {output_file}。")
[EOF]
(Pdb) s
--Call--
> c:\users\qiang\repo\week04\main.py(1)read_contacts()
-> def read_contacts(file_path):
(Pdb) l
  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                      name, gender, email = line.strip().split()
  7                      contacts.append((name, gender, email))
  8                  return contacts
  9          except FileNotFoundError:
 10              print(f"错误：文件 {file_path} 未找到。")
 11              return []
(Pdb) wat()
Local variables:
  file_path: str = 'contacts.txt'
```

打 S 进入到函数里边（进入小房间），再打局部变量 wat(),就只有这个函数里边的变量

注意：

```
with open(output_file, "w", encoding="utf-8") as file:
```

这里的两个橙色是两个形参，形参天然就是局部变量

局部变量 VS 全局变量：

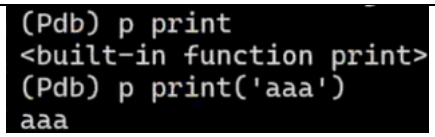
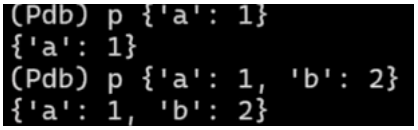
```
def read_contacts(file_path):
    try:
        with open(file_path, "r", encoding="utf-8") as file:
            lines = file.readlines()
            contacts = []
            for line in lines:
                name, gender, email = line.strip().split()
                contacts.append((name, gender, email))
            return contacts
    except FileNotFoundError:
        print("错误：未找到联系人文件！")
        return []


def generate_emails(contacts):
    sorted_contacts = sorted(
        contacts, key=lambda x: (x[2].split("@")[1], x[2].sp
    )
    emails = []
    for name, gender, email in sorted_contacts:
        title = "先生" if gender == "男" else "女士"
        email_text = f"to: <{email}>\n尊敬的{name}{title}，您
        emails.append(email_text)
    return "\n".join(emails)
```

```
if __name__ == "__main__":
    contacts_file = "contacts.txt"
    output_file = "emails.txt"
    contacts = read_contacts(contacts_file)
    sorted_contacts = sort_contacts(contacts)
    generated_emails = generate_emails(sorted_contacts)
    write_emails(generated_emails, output_file)
    print(f"邮件已成功生成到 {output_file}。")
```

红色：不缩进的变量是全局变量、if 下面的也是全局变量：随时 p+变量，都可以输出正确值

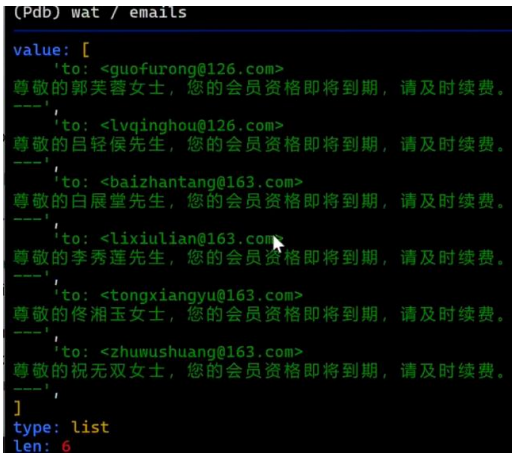
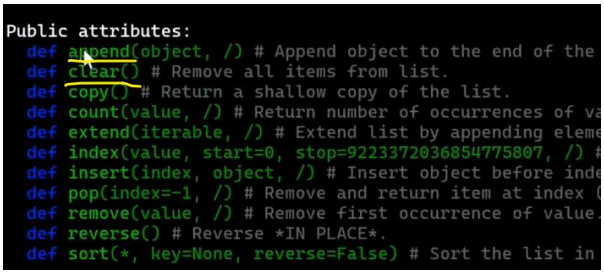
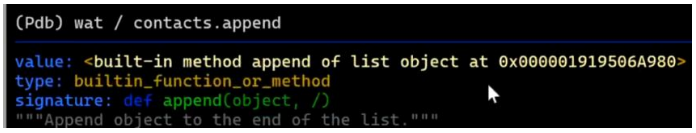
粉色：局部变量，只有进入到 def 语句中去或者某个函数中去，才能 wat()里显示出来，p+变量也能显示出来

		LEGB 规则：四个作用域的首字母，定义了查找变量的顺序： **LEGB 查找顺序** 当 Python 解释器遇到一个变量引用时，它会按照以下顺序查找该变量的定义： 1. 首先在局部作用域（Local）中查找，如果找到了对应的变量，则使用该变量的值。 2. 如果在局部作用域中没有找到，则在闭包作用域（Enclosing）中查找。 3. 如果在闭包作用域中也没有找到，则在全局作用域（Global）中查找。 4. 如果在全局作用域中仍然没有找到，则在内置作用域（Built-in）中查找。 5. 如果在内置作用域中也没有找到该变量，则会抛出 NameError 异常。
	调用（call）	 调用函数： p +函数变量：只能显示函数的参数 要想调用函数，就要给参数赋值 如：p +函数变量（aaa），相当于给出 x=几，调用函数能得到 f（x）
	字面值（literal)包括： 字符串（str)就是男、 整数（int)、 列表（list)就是方括号、 字典（dict)就是大括号、 元组（tuple)就是小括号	红色：字面值：   {}：字典 'a'：间　　1：值　　'a':1 叫键值对
	运算符（operator）	 一个等号：是赋值语句 两个等号：才是运算符 这里面一共三个运算符：if、==、else 这叫"三目运算符"：由三个表达式构成 if else 就是它中间是一个表达式，然后左边是个表达式，然后右边是个表达式，就是这个 if else，这个运算符，它支持三个表达式。它的规则是先判断中间的表达式是 true 还是 false，判断完了以后，如果是 true，就取左边的这个表达式。如果中间的这个东西是 false 的话，就取右边的这个表达式，最终这个符合的表达式，它最后这个表达式计算出来是什么

		结果就取决于中间的这个条件。这叫做三目运算符 其他运算符：+-*%　和　.（点叫做名称访问运算符） `file.write(email + "\n")` 这个 file 点 write 是在这个 file 对象的这个名称空间里边，我要访问它的名称空间里边的叫 write 的东西。你看 write 我们也没定义是不是，但是你没定义，但 file 这个对象定义了什么叫 write 其他运算符：　()调用函数运算符

	形参（parameter）、实参（argument）、返回值（return value）	形参：
		`def generate_emails(contacts):`
		实参：
		但是等到，你在调用的时候，往里传数的时候，
		就叫做实参。它是能找到真正的对象的（x=几，几就是形参的值）
		`contacts = read_contacts(contacts_file)`
		
		返回值：return 后面的就是返回值
		调用这个函数的时候，它就进到这个房间里面了，在房间里面，可能有一些局部变量搞这搞那搞完了以后，它一旦遇到这个 return 的语句。就把 return 后面的这个对象或者表达式算出来的某一个东西作为返回值就带回去了。
		`def generate_emails(contacts):`
		`    emails = []`
		`    for name, gender, email in contacts:`
		`        title = "先生" if gender == "男"`
		`        email_content = f"to: <{email}>\n`
		`        emails.append(email_content)`
		`    return emails`

	对象（object）　类型（type）、属性（attribute）、方法（method）	python 所管理的内存，内存所管理的那些东西通通都是对象
		wat:用来“深度检查 python 对象”的东西
		`(Pdb) wat()`
		`Local variables:`
		`  emails: list = […`
		`  output_file: str = 'emails.txt'`
		`  wat: wat.inspection.inspection.Wat = <WAT Inspector object>`
		`(Pdb) pp emails`
		`['to: <guofurong@126.com>\n尊敬的郭芙蓉女士，您的会员资格即将到期，请及时续费。\n---',`
		`'to: <lvqinghou@126.com>\n尊敬的吕轻侯先生，您的会员资格即将到期，请及时续费。\n---',`
		`'to: <baizhantang@163.com>\n尊敬的白展堂先生，您的会员资格即将到期，请及时续费。\n---',`
		`'to: <lixiulian@163.com>\n尊敬的李秀莲先生，您的会员资格即将到期，请及时续费。\n---',`
		`'to: <tongxiangyu@163.com>\n尊敬的佟湘玉女士，您的会员资格即将到期，请及时续费。\n---',`
		`'to: <zhuwushuang@163.com>\n尊敬的祝无双女士，您的会员资格即将到期，请及时续费。\n---']`
		输入：wat / emails　（wat 空格斜杠空格变量：仔细查看这个变量，也叫对象）

		得到 emails 的：value 值、tyle 类型、len 长度、public attribute 公开属性   黄线东西：Append 等：都是定义的方法 注意： 对象：都有类型，我们在编程的时候，了解这些对象的类型是很重要的 方法：就是这个对象，它能干些什么事 

Python 语法
保留字 (reserved key words)
语句 (statement) 和表达式 (expression)
缩进 (indent)
局部变量 (local variable)、全局变量 (global variable)、LEGB 规则
函数 (function) 的定义 (define) 和调用 (call)
字面值 (literal) (字符串 (str)、整数 (int)、列表 (list)、字典 (dict)、元组 (tuple))
运算符 (operator)
形参 (parameter)、实参 (argument)、返回值 (return value)
对象 (object)、类型 (type)、属性 (attribute)、方法 (method)