金融计算机第四周作业

专题一 极简版计算机组成原理与操作系统

一、计算机组成原理和操作系统

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。

--硬件：

- CPU：即中央处理器（Central Processing Unit），是计算机系统的核心组件。主要功能为**指令执行、数据处理、协调控制。**
- 内存：内存是计算机中用于暂时存储数据和程序的组件，它对于计算机的运行速度和多任务处理能力起着关键作用。主要用于**数据存储与读取、多任务处理支持。**

--底层软件：

- 操作系统：操作系统（Operating System，简称 OS）是管理计算机硬件与软件资源的程序，是计算机系统的核心与基础。主要用于**管理 CPU 和内存**。

--处理过程：

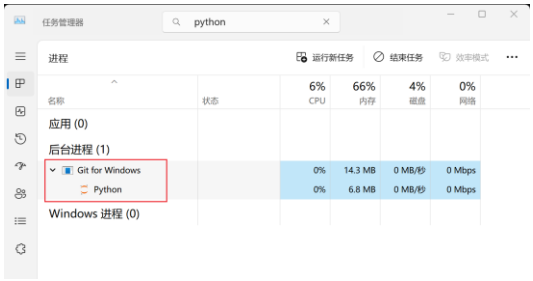
- 进程：进程是指正在运行的程序的实例，是操作系统进行资源分配和调度的基本单位。它包括程序计数器、寄存器、堆栈、程序代码和数据等，这些元素共同构成了进程执行的上下文环境。
  CPU 和内存在运行过程中，以进程为单位处理，而且相互隔离来完成相应计算任务。

e.g. 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。

二、python 解释器

Python 解释器也是一个**进程**，



按**流程**执行准备好的代码，根据代码要求，调用操作系统或其他软件（即依赖项），委托它们执行各种“读取—计算—写入”等工作。

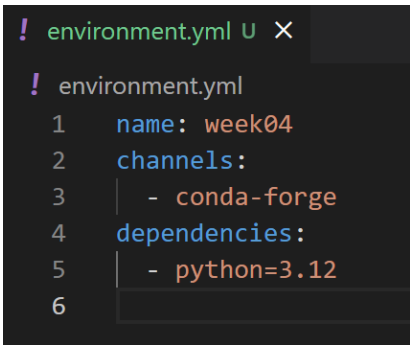
计算工作是在进程过程中完成的。

在编写程序的时候要注意调用的输入（参数）、输出(返回值)

Python 编程本质上是拼接操纵各种对象。

三、实践操作

1.新建一个 environment.yml 文件，然后创建 Conda 环境




```
(base)
86150@GFJ MINGW64 ~/repo/week04 (main)
$ ls -l
total 25
-rw-r--r-- 1 86150 197609    76  4月  6 21:23 environment.yml
-rw-r--r-- 1 86150 197609 18805  4月  1 09:39 LICENSE
-rw-r--r-- 1 86150 197609  2239  4月  1 09:39 README.md
(base)
done
#
# To activate this environment, use
#
#     $ conda activate week04
#
# To deactivate an active environment, use
#
#     $ conda deactivate
```

2. 新建一个 contacts.txt 文件，每行写一个联系人

示例：

```
(base) 86150@GFJ MINGW64 ~/repo/week04 (main)
$ ls -l
total 26
-rw-r--r-- 1 86150 197609   202  4月  6 21:41 contacts.txt
-rw-r--r-- 1 86150 197609    76  4月  6 21:23 environment.yml
-rw-r--r-- 1 86150 197609 18805  4月  1 09:39 LICENSE
-rw-r--r-- 1 86150 197609  2239  4月  1 09:39 README.md

(base) 86150@GFJ MINGW64 ~/repo/week04 (main)
$ cat contacts.txt
白展堂 男 baizhantang@163.com
佟湘玉 女 tongxiangyu@163.com
吕轻侯 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
```

p.s.：cat 命令是 contact（拼接）

```
(base) 86150@GFJ MINGW64 ~/repo/week04 (main)
$ cat contacts.txt environment.yml
白展堂 男 baizhantang@163.com
佟湘玉 女 tongxiangyu@163.com
吕轻侯 男 lvqinghou@126.com
郭芙蓉 女 guofurong@126.com
李秀莲 男 lixiulian@163.com
祝无双 女 zhuwushuang@163.comname: week04
channels:
  - conda-forge
dependencies:
  - python=3.12
```

实例：

```
(base) 86150@GFJ MINGW64 ~/repo/week04 (main)
$ cat contacts.txt
林悦悦 女 linyue1995@example.com
陈宇轩 男 chenyuxuan2000@hotmail.com
刘思瑶 女 liusiyao_2023@163.com
张皓阳 男 zhanghaoyang888@gmail.com
王诗涵 女 wangshihan0325@sina.com
赵俊辉 男 zhaojunhui123@outlook.com
周雅琴 女 zhouyaqin777@qq.com
```

3. 新建一个 main.py 文件，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：要求输出是先按邮箱域名排序，然后再按邮箱用户名排序

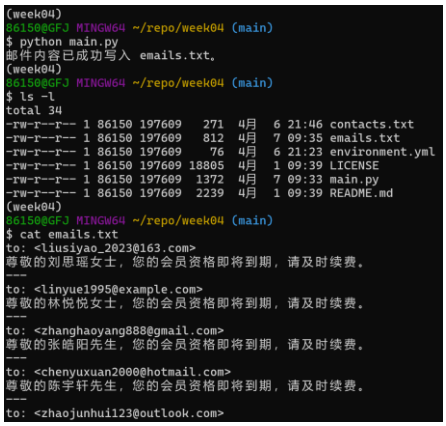
```python
def read_contacts(file_path):
    contacts = []
    try:
        with open(file_path, "r", encoding="utf-8") as file:
            for line in file:
                name, gender, email = line.strip().split()
                contacts.append((name, gender, email))
    except FileNotFoundError: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


def generate_emails(contacts):
    sorted_contacts = sorted(
        contacts, key=lambda x: (x[2].split("@")[1], x[2].split("@")[
    )
    email_text = ""
    for name, gender, email in sorted_contacts:
        title = "先生" if gender == "男" else "女士"
        email_text += f"to: <{email}>\n"
        email_text += f"尊敬的{name}{title}，您的会员资格即将到期，请及时
        email_text += "---\n"
    return email_text


def write_emails(file_path, email_text):
    try:
        with open(file_path, "w", encoding="utf-8") as file:
            file.write(email_text)
        print(f"邮件内容已成功写入 {file_path}。")
    except Exception as e:
        print(f"错误：写入文件 {file_path} 时出现问题：{e}。")


if __name__ == "__main__":
    contacts = read_contacts("contacts.txt")
    if contacts:
        email_text = generate_emails(contacts)
```

代码运行：



```
(week04)
86150@GFJ MINGW64 ~/repo/week04 (main)
$ python main.py
邮件内容已成功写入 emails.txt。
(week04)
86150@GFJ MINGW64 ~/repo/week04 (main)
$ ls -l
total 34
-rw-r--r-- 1 86150 197609   271  4月  6 21:46 contacts.txt
-rw-r--r-- 1 86150 197609   812  4月  7 09:35 emails.txt
-rw-r--r-- 1 86150 197609    76  4月  6 21:23 environment.yml
-rw-r--r-- 1 86150 197609 18805  4月  1 09:39 LICENSE
-rw-r--r-- 1 86150 197609  1372  4月  7 09:33 main.py
-rw-r--r-- 1 86150 197609  2239  4月  1 09:39 README.md
(week04)
86150@GFJ MINGW64 ~/repo/week04 (main)
$ cat emails.txt
to: <liusiyao_2023@163.com>
尊敬的刘思瑶女士，您的会员资格即将到期，请及时续费。
---
to: <linyue1995@example.com>
尊敬的林悦悦女士，您的会员资格即将到期，请及时续费。
---
to: <zhanghaoyang888@gmail.com>
尊敬的张皓阳先生，您的会员资格即将到期，请及时续费。
---
to: <chenyuxuan2000@hotmail.com>
尊敬的陈宇轩先生，您的会员资格即将到期，请及时续费。
---
to: <zhaojunhui123@outlook.com>
```

专题二 人类如何面对 AI 编程

大模型的确已经能够按照我们的要求生成文章、代码，而且无疑还在更加快速地变得更加强大。但人类的分工在无限深化，人类的定制化需求永无止境，永远需要我们人类自己来解决更多的创新性问题。是的，**创新**，因为不创新的常规问题都可以自动化了。所以我们不要以为“大模型都可以写代码了，我们就不用再学习编程了”，相反，我们不仅要学 (学已经变得很容易)，而且还要能创新，学习的压力反而更大了。真正创新的文章、代码是不一样的，是有灵魂的。尽管大模型能够生成，但在这里我写下的每一个字，我都不希望是由大模型生成的，我都希望由我自己来写。我所编写的每一行代码，我都更愿意由我自己来写，大模型顶多作为一个助手 (Copilot)，一个学习的帮手，我还是希望最终由我亲手掌控，因为我做的是做前所未有的创新工作。——这就是我所预见的人类和大模型 (AI) 相处的哲学关系。

AI is creating a generation of illterate programmers.

氛围编程 vibe coding：完全投入到氛

围中，拥抱技术的指数级发展，并忘记代码的存在

专题三 运用 pdb 检查程序的内部运行

由于专题一中实例的代码是 AI 编写，所以进行人工调试

流程包括循环、分支、函数的调用

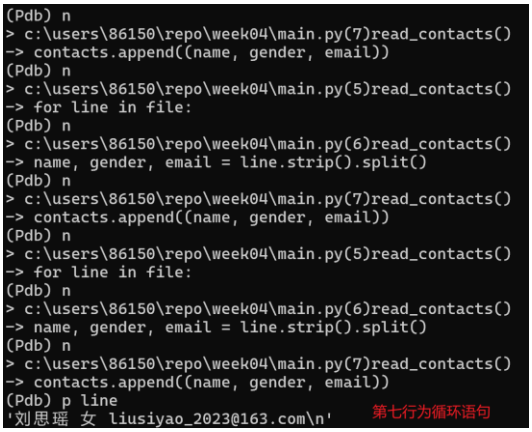
Pdb 常用命令：

1. l（list）显示代码

命令用于显示源代码的上下文。它能展示当前执行行附近的代码，让你知晓当前在代码中的位置，同时能看到周围代码的情况。

2. n（next）执行当前行

功能是**执行当前行，接着停在下一行**。当你想要逐行执行代码，并且不深入到函数内部时，这个命令就很实用。它会把当前行的代码执行完，然后在代码的下一行暂停，不管当前行是否调用了函数。



3. p（print）打印表达式

打印出指定表达式的值。在调试期间，若你想查看某个变量或者表达式的当前值，就可以使用此命令。

p.s.: n 命令推动程序逐行执行，而 p 命令用于查看变量的值

File_path：参数，只有在函数的内部才能访问到

4. s（step in）步入调用

该命令用于单步执行代码，与 n

（next）命令类似，都会执行当前行代码。但不同之处在于，当当前行代码调用了函数时，s 命令会进入被调用的函数内部，从函数的第一行开始继续单步调试。

```
(Pdb) s
> c:\users\86150\repo\week04\main.py(26)<module>()
-> def write_emails(file_path, email_text):
```

```
(Pdb) n
> c:\users\86150\repo\week04\main.py(36)<module>()
-> contacts = read_contacts("contacts.txt")
(Pdb) s
--Call--
> c:\users\86150\repo\week04\main.py(1)read_contacts()
-> 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                     name, gender, email = line.strip().split()
  7                     contacts.append((name, gender, email))
  8         except FileNotFoundError:
  9             print(f"错误：未找到文件 {file_path}。")
 10         return contacts
 11
(Pdb) n                                    调用函数
> c:\users\86150\repo\week04\main.py(2)read_contacts()
-> contacts = []
```

5. pp（pretty print）美观打印

主要功能是将指定表达式的结果以更美观、易读的格式打印出来。当要打印的数据结构（像列表、字典、嵌套结构等）比较复杂时，使用 pp 命令能够让输出的格式更清晰，便于查看和分析。

```
(Pdb) pp contacts
[('林悦悦', '女', 'linyue1995@example.com'),
 ('陈宇轩', '男', 'chenyuxuan2000@hotmail.com'),
 ('刘思瑶', '女', 'liusiyao_2023@163.com'),
 ('张皓阳', '男', 'zhanghaoyang888@gmail.com'),
 ('王诗涵', '女', 'wangshihan0325@sina.com'),
 ('赵俊辉', '男', 'zhaojunhui123@outlook.com'),
 ('周雅琴', '女', 'zhouyaqin777@qq.com')]
```

6. c（continue）继续执行

命令的作用是让程序继续执行，直到遇到下一个断点或者程序结束。也就是说，当你在调试过程中，觉得当前的单步调试阶段已经完成，想让程序快速运行到下一个需要关注的位置时，就可以使用 c 命令。

```
(Pdb) c
邮件内容已成功写入 emails.txt。
The program finished and will be restarted
> c:\users\86150\repo\week04\main.py(1)<module>()
-> def read_contacts(file_path):
```

专题四 python 基础概念与对象检视

1. 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

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

Wat-inspector：深度监视 python 对象

2. python 基本概念

a．python 语法保留字

```
>>> name="GUO"
>>> print(name)
GUO
>>> def='GUO'   保留字
  File "<stdin>", line 1
    def='GUO'
        ^
SyntaxError: invalid syntax
```

在 Python 里，保留字是被编程语言预留的特殊单词，每个保留字都有特定的用途，在定义变量、函数或者类时不能使用这些保留字作为名称。

有以下这些保留字：

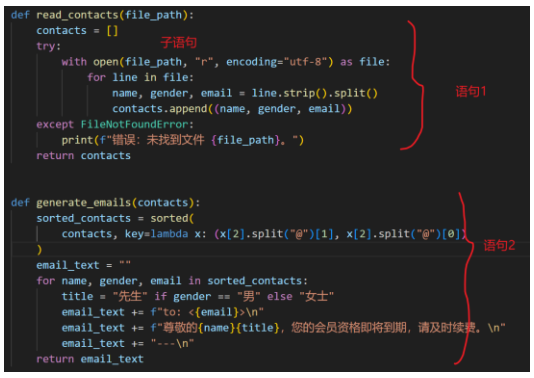
```
plaintext
```

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

b. 语句（statement）和表达式（expression）

- 语句：

语句是编程语言的基本组成单元，它是向计算机发出的一个指令，用于完成特定操作。

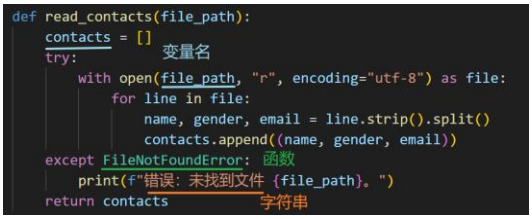


主语句中可以嵌套子语句

- 表达式:

构成语句的元素，比语句小。

表达式是由值、变量、运算符和函数调用等组合而成的代码片段，它会计算出一个结果。表达式可以是简单的，也可以是复杂的。



c. 缩进（indent）

缩进代表层级，通过缩进界定子语句的边界。

在 Python 中不像其他一些编程语言（如 Java、C++ 等）使用大括号{}来界定代码块，而是使用缩进来明确代码块的范围。像 if、else、elif、for、while、def、class 这类语句，后面的代码块都需要通过缩进来表示。

d. 局部变量（local variable）vs. 全局变量（global variable）

```
(week04)
86150@GFJ MINGW64 ~/repo/week04 (main)
$ python -m pdb main.py
> c:\users\86150\repo\week04\main.py(1)<module>()
-> def read_contacts(file_path):
(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
  .gray to disable colorful output in the console
  .color to enforce colorful outputs in the console
Call wat.locals or wat() to inspect local variables.
Call wat.globals to inspect global variables.
```

- 局部变量

在 def 里面，在只有调用函数运行后才会显示的是局部变量

局部变量是在特定代码块内定义的变量，这些代码块可以是函数、类方法或者代码块结构（如循环、条件语句）。

```
(Pdb) wat()
Local variables:
  __builtins__: dict = {…
  __file__: pdb._ScriptTarget = 'C:\Users\86150\repo\week04\main.py'
  __name__: str = '__main__'
  __pdb_convenience_variables: dict = {…
  __spec__: NoneType = None
  wat: wat.inspection.inspection.Wat = <WAT Inspector object>
```

```
(Pdb) s
--Call--
> c:\users\86150\repo\week04\main.py(1)read_contacts()
-> def read_contacts(file_path):
(Pdb) wat()
Local variables:
  file_path: str = 'contacts.txt'
```

在调用函数运行结束后，该语句下的变量将不会被显示

- 全局变量

总是能访问到的是全局变量，在顶层运行的语句。

全局变量是定义在所有函数、类和代码块外部的变量。它可以在整个程序的任何地方被访问和使用。

```
(Pdb) wat.globals
Global variables:
  __builtins__: dict = {…
  __file__: pdb._ScriptTarget = 'C:\Users\86150\repo\week04\main.py'
  __name__: str = '__main__'
  __pdb_convenience_variables: dict = {…
  __spec__: NoneType = None
  generate_emails: function = <function generate_emails at 0x00000284BD29F740>
  read_contacts: function = <function read_contacts at 0x00000284BD29F6A0>
  wat: wat.inspection.inspection.Wat = <WAT Inspector object>
  write_emails: function = <function write_emails at 0x00000284BD29FD80>
```

- LEGB 规则

在 Python 中，当你引用一个变量时，Python 解释器会按照特定的顺序来查找这个变量，这个顺序就是 LEGB 规则。LEGB 是 **Local**（局部作用域）、**Enclosing**（闭包作用域，也称为嵌套作用域）、**Global**（全局作用域）、**Built - in**（内置作用域）的首字母缩写。

LEGB 查找规则

当 Python 解释器遇到一个变量引用时，会按照以下顺序查找该变量：

1. Local：首先在当前函数或方法的局部作用域中查找变量。

2. Enclosing：如果在局部作用域中没有找到变量，解释器会在闭包作用域（即外层函数的作用域）中查找。

3. Global：如果在闭包作用域中也没有找到，解释器会在全局作用域中查找。

4. Built - in：如果在全局作用域中仍

然没有找到，解释器会在内置作用域中查找。

如果在所有这些作用域中都没有找到变量，Python 会抛出 Name Error 异常。

e. 函数（function）的定义和调用（call）

调用：

函数调用就是在代码里使用函数名，并且按照函数定义的要求传入相应参数，以此来执行函数体中的代码。

```
(Pdb) p print
<built-in function print>
(Pdb) p print('aaa')
aaa
None
```
调用

f. 字面值（literal）

在 Python 里，字面值（Literal）指的是在代码中直接表示数据值的符号。简单来说，就是你直接写在代码里的具体的数据。

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

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

g. 运算符（operator）

在 Python 中，运算符是用于执行各种操作的符号，它们可以对一个或多个操作数进行运算，从而得到一个结果。

```python
def generate_emails(contacts):
    sorted_contacts = sorted(
        contacts, key=lambda x: (x[2].split("@")[1], x[2].split("@")[0])
    )
    email_text = ""
    for name, gender, email in sorted_contacts:
        title = "先生" if gender == "男" else "女士"
        email_text += f"to: <{email}>\n"
        email_text += f"尊敬的{name}{title}，您的会员资格即将到期，请及时续费。\n"
        email_text += "---\n"
    return email_text
```

.:名称访问运算符

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

()：调用运算符

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

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

- 形参：(抽象)

在 Python 里，形参即形式参数，它是在定义函数时声明的参数。形参的作用是在函数内部代表调用函数时传入的实际数据。

在**定义函数**时，你可以指定函数需要接收的参数，这些参数就是形参

```python
def generate_emails(contacts):
```

- 实参：（具体）

在 Python 函数调用的过程中，实参（实际参数）是调用函数时传递给函数的具体数据。实参为函数的形参（形式参数）提供具体的值，使得函数能够依据这些值进行相应的操作。

当**调用函数**时，需要传入与形参相对应的实际值，也就是实参。

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

- 返回值：

在 Python 中，返回值是函数执行完毕后传递给调用者的结果。函数可以通过 return 语句将一个值或多个值返回给调用它的地方。

函数的主要目的之一是完成特定的任务并产生结果。返回值使得函数能够将计算结果传递给调用者，调用者可以根据这个结果进行后续的操作。这增强了代码的复用性和模块化程度。

```
return email_text
```

Return 连接表达式的值即为返回值，若语句中无 return，则返回 None。

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

- 对象：

Python 所管理的内存都是对象。

对象是类的实例。类就像是一个模板或者蓝图，定义了对象的属性（数据）和方法（操作数据的函数）。当根据类创建一个具体的实例时，就得到了一个对象。

- 类型：

在 Python 里，类型指的是数据的种类，它定义了数据的特征以及可以对其进行的操作。Python 是一种动态类型语言，在定义变量时无需声明其类型，解释器会在运行时自动确定变量的类型。

- 属性：

在 Python 中，属性是与对象或类相关联的数据项，它用于描述对象或类的特征。属性可以分为实例属性和类属性。

- 方法：

在 Python 里，方法是与对象或类相关联的函数，它定义了对象可以执行的操作。方法可以分为实例方

法、类方法和静态方法