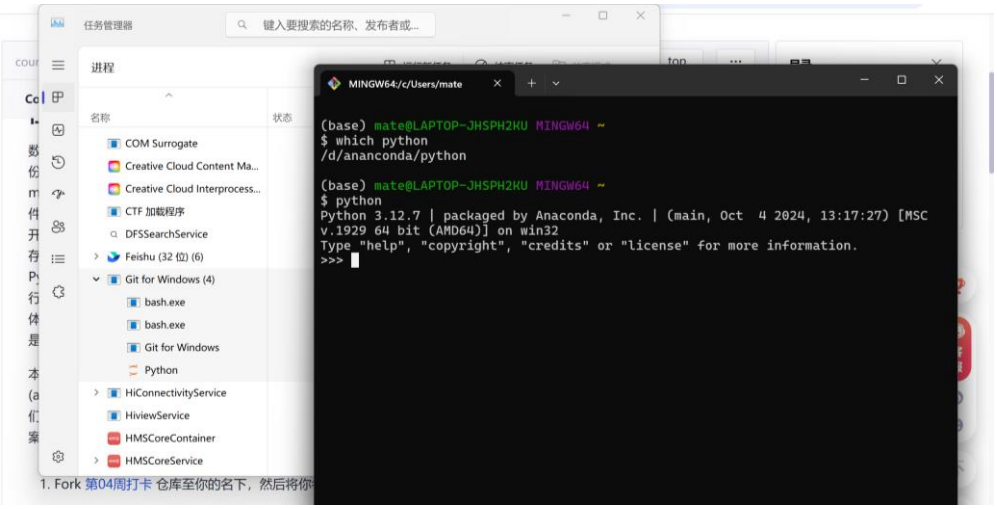
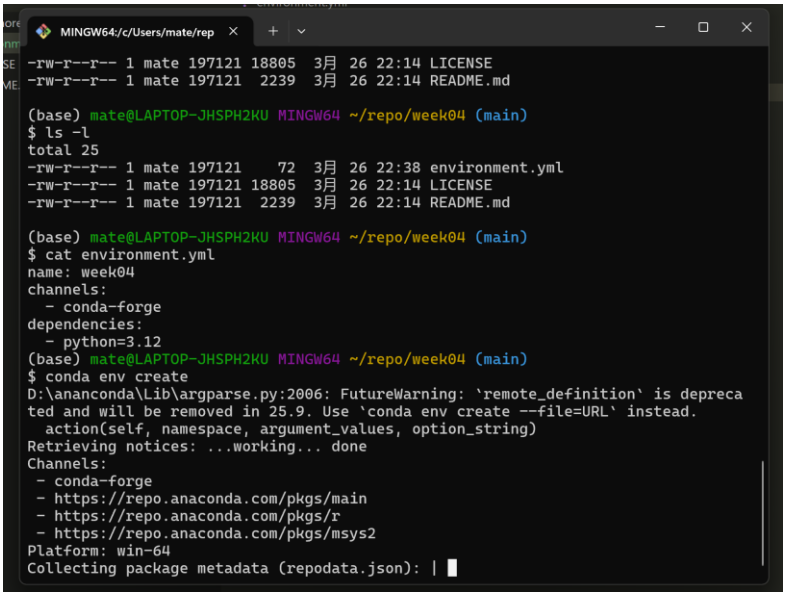
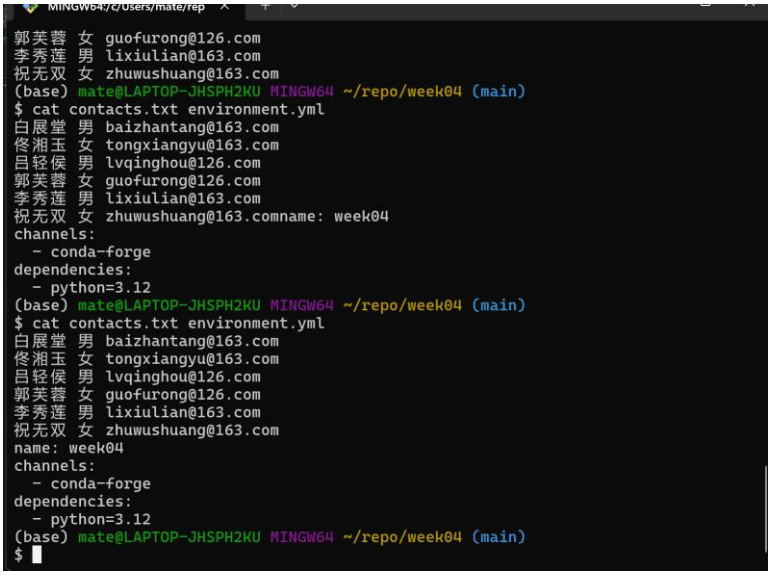
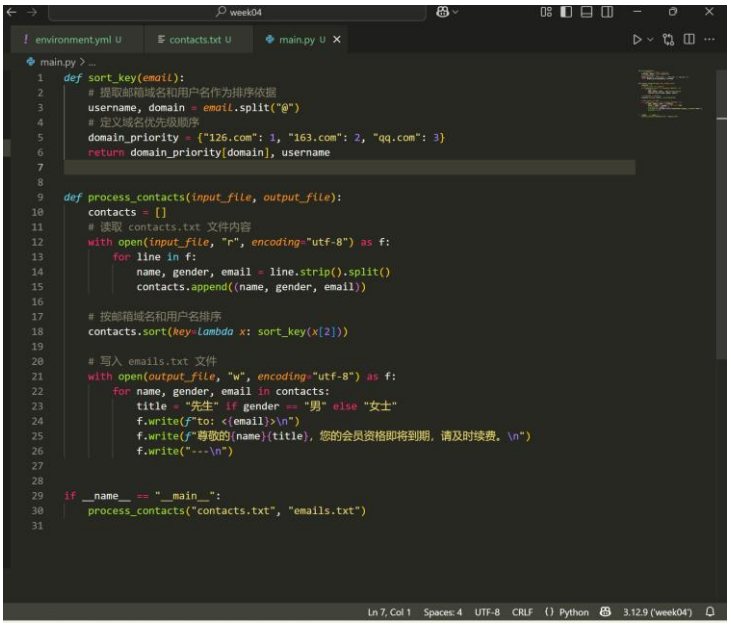
第四周学习报告


任务管理器
键入要搜索的名称、发布者或...
进程
名称
状态
COM Surrogate
Creative Cloud Content Ma...
Creative Cloud Interprocess...
CTF 加载程序
DFSSearchService
Feishu (32 位) (6)
Git for Windows (4)
bash.exe
bash.exe
Git for Windows
Python
HiConnectivityService
HiviewService
HMSCoreContainer
HMSCoreService
MINGW64:/c/Users/mate
(base) mate@LAPTOP-JHSPH2KU MINGW64 ~
$ which python
/d/ananconda/python
(base) mate@LAPTOP-JHSPH2KU MINGW64 ~
$ python
Python 3.12.7 | packaged by Anaconda, Inc. | (main, Oct 4 2024, 13:17:27) [MSC
v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
1. Fork 第04周打卡 仓库至你的名下，然后将你


MINGW64:/c/Users/mate/rep
-rw-r--r-- 1 mate 197121 18805 3月 26 22:14 LICENSE
-rw-r--r-- 1 mate 197121 2239 3月 26 22:14 README.md
(base) mate@LAPTOP-JHSPH2KU MINGW64 ~/repo/week04 (main)
$ ls -l
total 25
-rw-r--r-- 1 mate 197121 72 3月 26 22:38 environment.yml
-rw-r--r-- 1 mate 197121 18805 3月 26 22:14 LICENSE
-rw-r--r-- 1 mate 197121 2239 3月 26 22:14 README.md
(base) mate@LAPTOP-JHSPH2KU MINGW64 ~/repo/week04 (main)
$ cat environment.yml
name: week04
channels:
- conda-forge
dependencies:
- python=3.12
(base) mate@LAPTOP-JHSPH2KU MINGW64 ~/repo/week04 (main)
$ conda env create
D:\ananconda\Lib\argparse.py:2006: FutureWarning: `remote_definition` is depreca
ted and will be removed in 25.9. Use `conda env create --file=URL` instead.
action(self, namespace, argument_values, option_string)
Retrieving notices: ...working... done
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): |

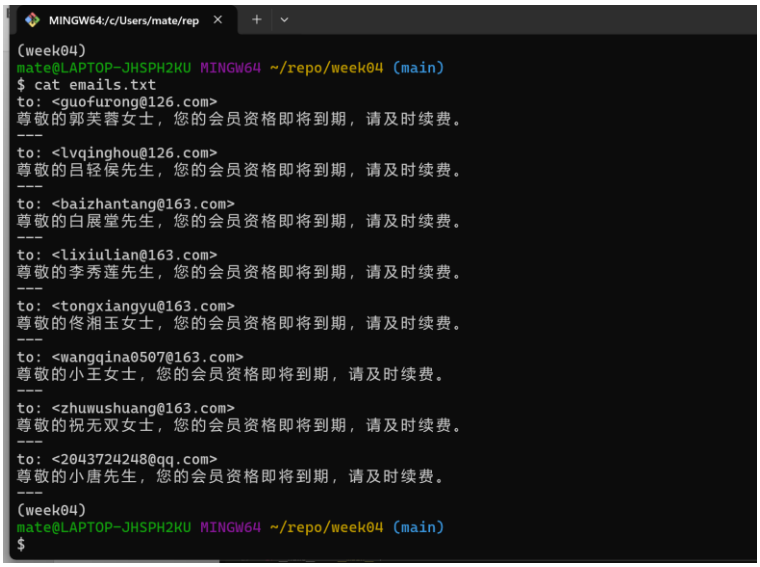

郭芙蓉 女 guofurong@126.com
李秀莲 男 lixiulian@163.com
祝无双 女 zhuwushuang@163.com
(base) mate@LAPTOP-JHSPH2KU MINGW64 ~/repo/week04 (main)
$ cat contacts.txt environment.yml
白展堂 男 baizhantang@163.com
佟湘玉 女 tongxiangyu@163.com
吕轻侯 男 lvqinghou@126.com
郭芙蓉 女 guofurong@126.com
李秀莲 男 lixiulian@163.com
祝无双 女 zhuwushuang@163.comname: week04
channels:
- conda-forge
dependencies:
- python=3.12
(base) mate@LAPTOP-JHSPH2KU 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.com
name: week04
channels:
- conda-forge
dependencies:
- python=3.12
(base) mate@LAPTOP-JHSPH2KU MINGW64 ~/repo/week04 (main)
$



```python
def sort_key(email):
    # 提取邮箱域名和用户名作为排序依据
    username, domain = email.split("@")
    # 定义域名优先级顺序
    domain_priority = {"126.com": 1, "163.com": 2, "qq.com": 3}
    return domain_priority[domain], username


def process_contacts(input_file, output_file):
    contacts = []
    # 读取 contacts.txt 文件内容
    with open(input_file, "r", encoding="utf-8") as f:
        for line in f:
            name, gender, email = line.strip().split()
            contacts.append((name, gender, email))

    # 按邮箱域名和用户名排序
    contacts.sort(key=lambda x: sort_key(x[2]))

    # 写入 emails.txt 文件
    with open(output_file, "w", encoding="utf-8") as f:
        for name, gender, email in contacts:
            title = "先生" if gender == "男" else "女士"
            f.write(f"to: <{email}>\n")
            f.write(f"尊敬的{name}{title}，您的会员资格即将到期，请及时续费。\n")
            f.write("---\n")


if __name__ == "__main__":
    process_contacts("contacts.txt", "emails.txt")
```

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: <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: <wangqina0507@163.com>
尊敬的小王女士，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。
---
to: <2043724248@qq.com>
尊敬的小唐先生，您的会员资格即将到期，请及时续费。
---

```



```
MINGW64:/c/Users/mate/rep
(week04)
mate@LAPTOP-JHSPH2KU 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: <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: <wangqina0507@163.com>
尊敬的小王女士，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。
---
to: <2043724248@qq.com>
尊敬的小唐先生，您的会员资格即将到期，请及时续费。
---
(week04)
mate@LAPTOP-JHSPH2KU MINGW64 ~/repo/week04 (main)
$
```

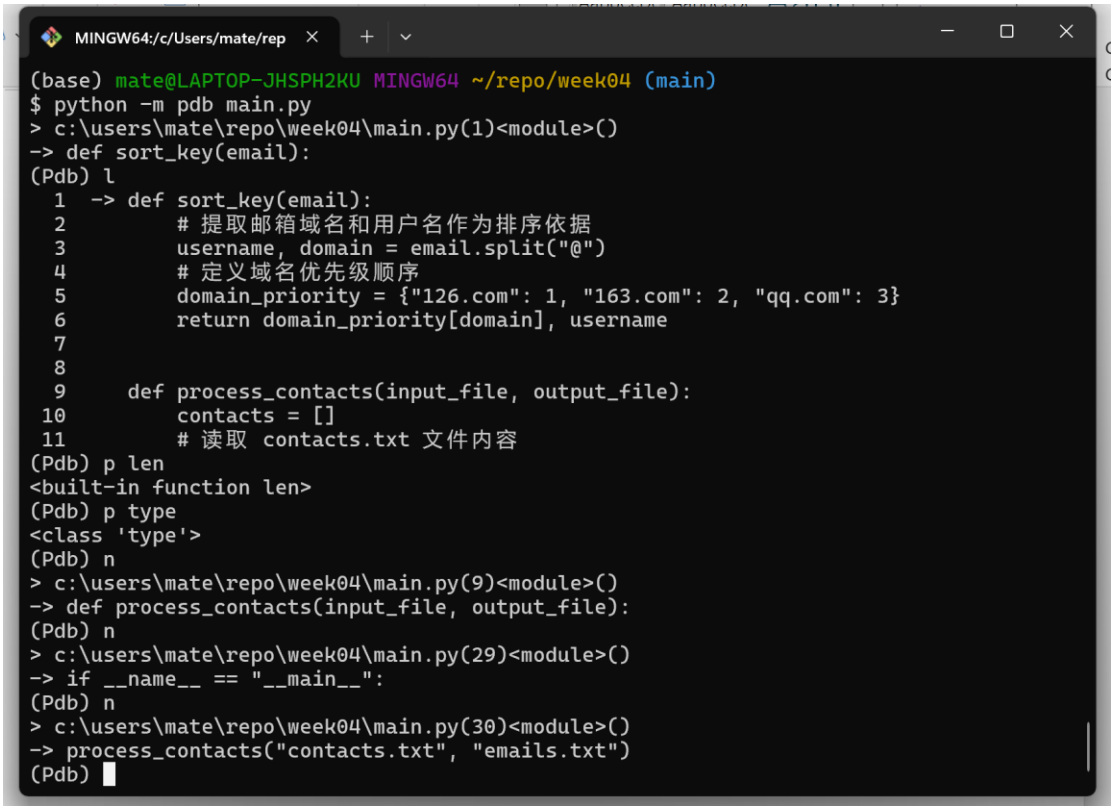
运行 `python -m pdb main.py` 命令 (作用是以调试模式 (debug mode) 启动 Python 解释器，准备执行 `main.py` 里的代码)

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 (参考文档)

```
(base) mate@LAPTOP-JHSPH2KU MINGW64 ~/repo/week04 (main)
$ python -m pdb main.py
> c:\users\mate\repo\week04\main.py(1)<module>()
-> def sort_key(email):
(Pdb) l
  1  -> def sort_key(email):
  2          # 提取邮箱域名和用户名作为排序依据
  3          username, domain = email.split("@")
  4          # 定义域名优先级顺序
  5          domain_priority = {"126.com": 1, "163.com": 2, "qq.com": 3}
  6          return domain_priority[domain], username
  7
  8
  9      def process_contacts(input_file, output_file):
 10          contacts = []
 11          # 读取 contacts.txt 文件内容
(Pdb) n
> c:\users\mate\repo\week04\main.py(9)<module>()
-> def process_contacts(input_file, output_file):
(Pdb) l
  4          # 定义域名优先级顺序
  5          domain_priority = {"126.com": 1, "163.com": 2, "qq.com": 3}
  6          return domain_priority[domain], username
```

```
MINGW64:/c/Users/mate/rep
 18          contacts.sort(key=lambda x: sort_key(x[2]))
 19
 20          # 写入 emails.txt 文件
 21          with open(output_file, "w", encoding="utf-8") as f:
 22              for name, gender, email in contacts:
 23                  title = "先生" if gender == "男" else "女士"
 24                  f.write(f"to: <{email}>\n")
 25                  f.write(f"尊敬的{name}{title}，您的会员资格即将到期，请及时续费
。\n")
 26                  f.write("---\n")
 27
 28
 29      if __name__ == "__main__":
 30          process_contacts("contacts.txt", "emails.txt")
(Pdb) s
> c:\users\mate\repo\week04\main.py(29)<module>()
-> if __name__ == "__main__":
(Pdb) ll
  1     def sort_key(email):
  2         # 提取邮箱域名和用户名作为排序依据
  3         username, domain = email.split("@")
  4         # 定义域名优先级顺序
  5         domain_priority = {"126.com": 1, "163.com": 2, "qq.com": 3}
  6         return domain_priority[domain], username
  7
  8
  9     def process_contacts(input_file, output_file):
 10         contacts = []
 11         # 读取 contacts.txt 文件内容
 12         with open(input_file, "r", encoding="utf-8") as f:
```

```
 28
 29  -> if __name__ == "__main__":
 30          process_contacts("contacts.txt", "emails.txt")
(Pdb) p sort_key
<function sort_key at 0x00000232494C39C0>
(Pdb) s
> c:\users\mate\repo\week04\main.py(30)<module>()
-> process_contacts("contacts.txt", "emails.txt")
(Pdb) l
 25                  f.write(f"尊敬的{name}{title}，您的会员资格即将到期，请及时续费
。\n")
 26                  f.write("---\n")
 27
 28
 29      if __name__ == "__main__":
 30  ->      process_contacts("contacts.txt", "emails.txt")
[EOF]
(Pdb) l .
 25                  f.write(f"尊敬的{name}{title}，您的会员资格即将到期，请及时续费
。\n")
 26                  f.write("---\n")
 27
 28
```



启动调试器

- **命令**：`python -m pdb main.py`

- **作用**：这个命令会以调试模式启动 Python 解释器，并准备执行`main.py`文件中的代码。`-m pdb`告诉 Python 使用 pdb 模块，这是一个内置的调试器。

在 pdb 提示符下使用的命令

在启动 pdb 后，你会看到一个`(pdb)`提示符，这时你可以输入以下命令来控制调试过程：

1. **`l`（list）**：
   - **作用**：显示当前行附近的代码。默认情况下，它会显示当前行及其周围的几行代码，帮助你查看上下文。
   - **用法**：`l` 或 `l 行号`（显示指定行号附近的代码）。

2. **`n`（next）**：
   - **作用**：执行当前行，并移动到下一行。如果当前行是一个函数调用，它不会进入函

数内部。

- **用法**：`n`。

3. **`p`（print）**：

- **作用**：打印一个表达式的值。这可以帮助你查看变量的当前值。

- **用法**：`p 表达式`（例如，`p my_variable`）。

4. **`s`（step）**：

- **作用**：执行当前行，并如果当前行是一个函数调用，它会进入该函数内部。

- **用法**：`s`。

5. **`pp`（pretty-print）**：

- **作用**：以更美观的格式打印一个表达式的值，特别是对于复杂的数据结构，如列表、字典等。

- **用法**：`pp 表达式`（例如，`pp my_list`）。

6. **`c`（continue）**：

- **作用**：继续执行程序，直到遇到下一个断点或程序结束。

- **用法**：`c`。

调试过程

调试过程通常包括以下步骤：

- 设置断点：你可以在代码中设置断点，这样当程序执行到该行时，pdb 会暂停执行，允许你检查当前状态。
- 单步执行：使用`n`和`s`命令逐步执行代码，观察程序的行为和变量的变化。
- 检查变量：使用`p`和`pp`命令查看变量的值，帮助理解程序的运行状态。
- 继续执行：使用`c`命令继续执行程序，直到下一个断点或程序结束。

```
(Pdb) c
The program finished and will be restarted
> c:\users\mate\repo\week04\main.py(1)<module>()
-> def sort_key(email):
(Pdb) c
The program finished and will be restarted
> c:\users\mate\repo\week04\main.py(1)<module>()
-> def sort_key(email):
(Pdb) q

(base) mate@LAPTOP-JHSPH2KU MINGW64 ~/repo/week04 (main)
$
```

environment.yml U | contacts.txt U | main.py U | *emails.txt U*

environment.yml

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

MINGW64:/c/Users/mate/rep

```
---
to: <2043724248@qq.com>
尊敬的小唐先生，您的会员资格即将到期，请及时续费。
---
(week04)
mate@LAPTOP-JHSPH2KU MINGW64 ~/repo/week04 (main)
$ ls -l
total 34
-rw-r--r-- 1 mate 197121   267  3月 26 23:02 contacts.txt
-rw-r--r-- 1 mate 197121   883  3月 27 21:16 emails.txt
-rw-r--r-- 1 mate 197121    91  3月 27 21:18 environment.yml
-rw-r--r-- 1 mate 197121 18805  3月 26 22:14 LICENSE
done
#
# To activate this environment, use
#
#     $ conda activate week04
#
# To deactivate an active environment, use
#
#     $ conda deactivate

(week04)
mate@LAPTOP-JHSPH2KU MINGW64 ~/repo/week04 (main)
$
```

- 在调试过程中，观察代码逐步运行的效果，学习理解以下 Python **基本概念** (建议观看下面的录播讲解)
  - 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) vs. 全局变量 (global variable)
  - 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)

Python 语法保留字（reserved key words）

- **定义**：Python 中预先定义好的具有特殊意义的单词，不能用作变量名、函数名等标识符。

- **示例**：`if`、`else`、`for`、`while`、`def`、`class`、`return`等。

- **作用**：用于构成程序的控制结构和语法规则。

语句（statement）和表达式（expression）（**语句包含表达式，表达式不包含语句**）

- **语句**：是 Python 代码中的一个独立单元，用于执行某种操作或命令，如赋值语句、控制流语句等。（**子语句**）

- **表达式**：由变量、运算符、字面值等组成的组合，用于计算并返回一个值，如`3 + 5`、`x * y`等。（**赋值语句=右边是表达式**）

- **区别**：语句主要关注执行操作，而表达式关注计算结果。

白色变量名；黄色字符串；绿色函数；红色保留字；蓝色定义函数保留字；橙色形参

缩进（indent）层级

- **定义**：Python 中使用空格或制表符来表示代码块的层次结构，通常用于循环、条件判断、函数定义等。

- **作用**：通过缩进将代码分组，体现代码的逻辑关系和执行顺序。

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

- **局部变量**：在函数内部定义的变量，只在该函数内部有效，函数执行完毕后变量会被

销毁。

- **全局变量**：在函数外部定义的变量，或在函数内部使用`global`关键字声明的变量，可以在整个程序中访问。

- **作用范围**：局部变量仅限于函数内部，全局变量则在整个程序范围内有效。

Python 解释器的作用域规则（**LEGB**）***变量查找顺序***

Python 解释器在查找变量时遵循 LEGB 规则，即**局部（Local）→封闭（Enclosing）→全局（Global）→内置（Built-in）**的作用域顺序：

- **局部作用域**（Local）**：在函数或方法内部定义的变量，只在该函数或方法内部有效。

- **封闭作用域**（Enclosing）**：对于嵌套函数，外层函数的作用域就是内层函数的封闭作用域。

- **全局作用域**（Global）**：在模块文件中定义的变量，属于全局作用域，在整个模块中都有效。

- **内置作用域**（Built-in）**：Python 内置的函数、异常、类型等，如`len()`、`str`等。

Python 解释器的工作原理

Python 解释器（如 CPython）的工作原理大致如下：

1. **解析（Parsing）**：将源代码解析成抽象语法树（AST）。

2. **编译（Compiling）**：将 AST 编译成字节码（Bytecode），字节码是 Python 虚拟机（Python Virtual Machine）能够理解的低级表示。

3. **执行（Executing）**：Python 虚拟机逐条解释执行字节码，通过一系列操作完成代码的功能。

在执行过程中，解释器会根据 LEGB 规则查找变量，确保代码的正确运行。

函数（function）的定义（define）和调用（call）

- **定义**：使用`**def**`关键字创建一个函数，指定函数名、参数列表和函数体。

- **调用**：通过函数名并传递相应的实参来执行函数中的代码。

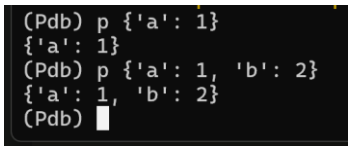
字面值（literal）（字符串（str）、整数（int）、列表（list）、字典（dict）、元组（tuple））

- **字符串**：用单引号`' '`或双引号`" "`括起来的字符序列，如`"hello"`。

- **整数**：表示整数数值，如`42`。

- **列表**：用方括号`[ ]`括起来的有序元素集合，元素之间用逗号分隔，如`[1, 2, 3]`。

- **字典**：用花括号`{ }`括起来的键值对集合，键和值之间用冒号分隔，如`{"name": "Alice", "age": 25}`。



- **元组**：用圆括号`()`括起来的有序元素集合，元素不可修改，如`(1, 2, 3)`。

运算符（operator）

- **定义**：用于执行特定的计算或操作的符号，如算术运算符`+`、`-`、`*`、`/`，比较运算符`==`、`>`、`<`等。

- **作用**：对一个或多个操作数进行运算，返回结果。

形参*x*（parameter）、**实参具体**（argument）、**返回值**（***return value***）

- **形参**：在函数定义时指定的参数，用于接收调用时传递的值。

- **实参**：在函数调用时实际传递给函数的值。

- **返回值**：函数执行完毕后返回给调用者的结果，通过`return`语句指定。

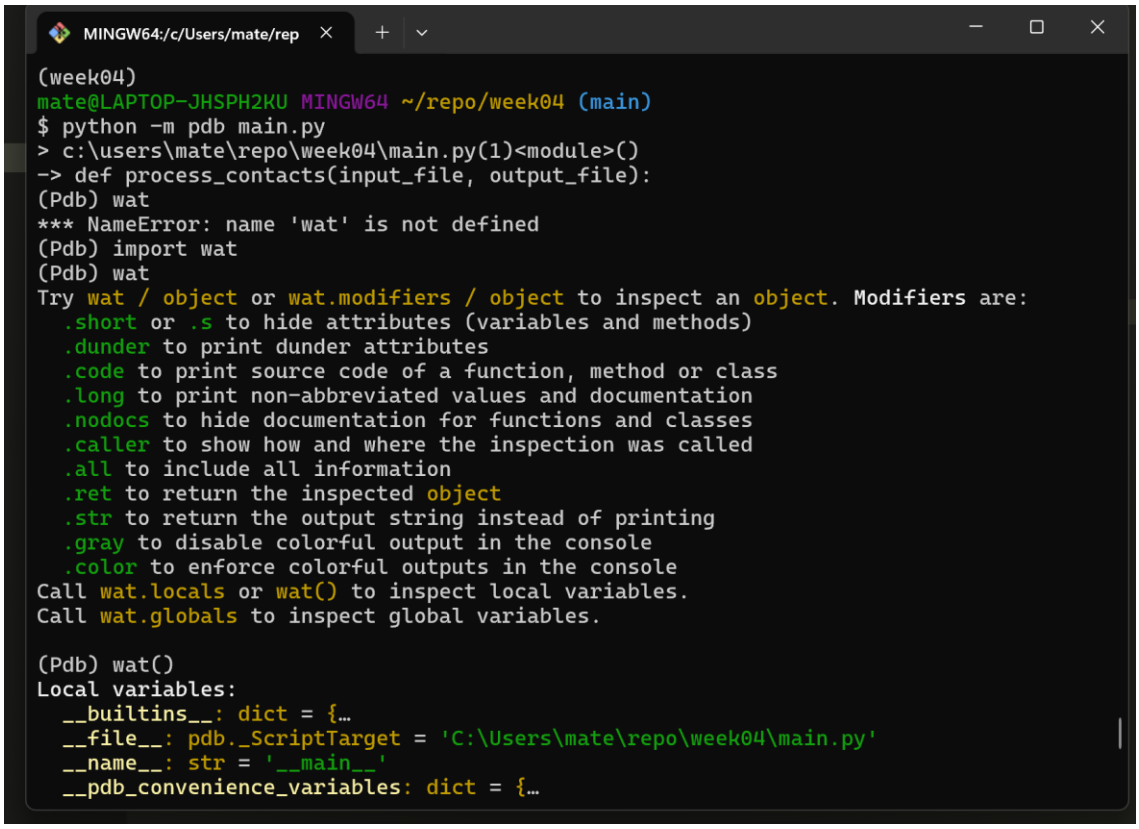
对象（object）、类型（type）、属性（attribute）、方法（method）

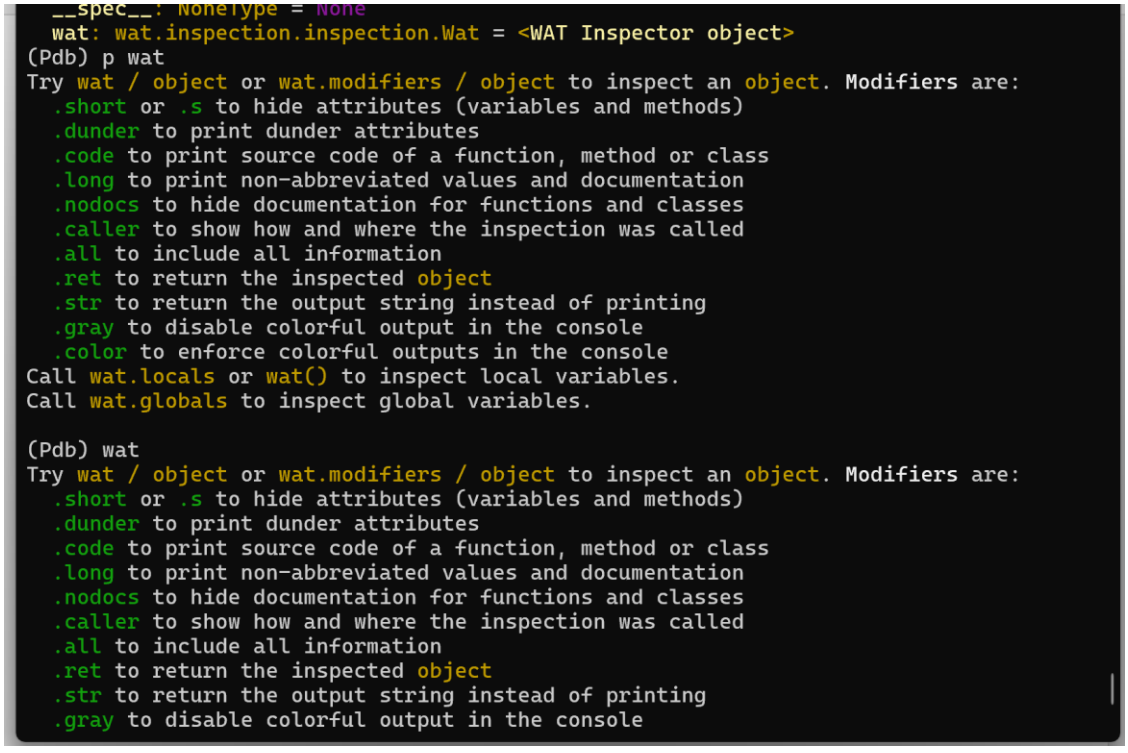
- **对象**：Python 中一切都是对象，包括数字、字符串、列表等，每个对象都有唯一的标识、类型和值。

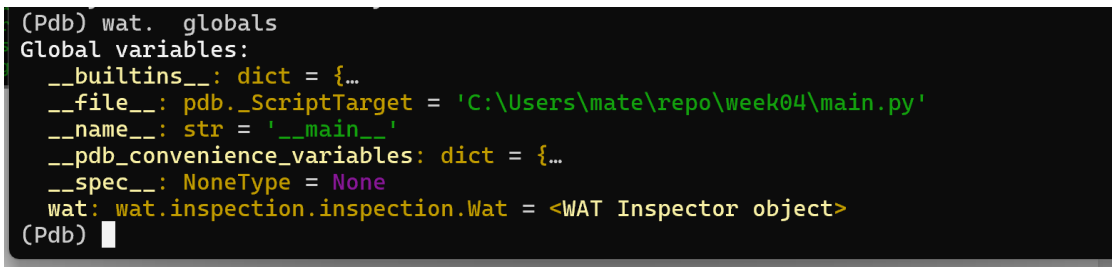
- **类型**：对象所属的类别，决定了对象支持的操作和属性，如`int`、`str`、`list`等。

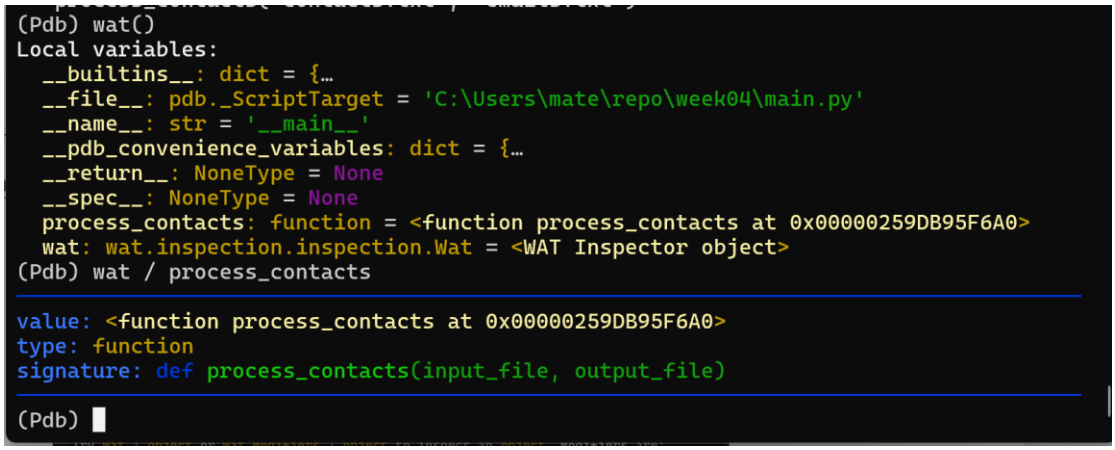
- **属性**：对象的特征或状态，通过点`.`操作符访问，如字符串的`upper`方法、列表的`length`属性。

- **方法**：对象可以执行的行为或操作，是与对象相关联的函数，通过点`.`操作符调用，如列表的`append()`方法、字符串的`split()`方法。


MINGW64:/c/Users/mate/rep
(week04)
mate@LAPTOP-JHSPH2KU MINGW64 ~/repo/week04 (main)
$ python -m pdb main.py
> c:\users\mate\repo\week04\main.py(1)<module>()
-> def process_contacts(input_file, output_file):
(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
.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.
(Pdb) wat()
Local variables:
__builtins__: dict = {…
__file__: pdb._ScriptTarget = 'C:\Users\mate\repo\week04\main.py'
__name__: str = '__main__'
__pdb_convenience_variables: dict = {…


__spec__: NoneType = None
wat: wat.inspection.inspection.Wat = <WAT Inspector object>
(Pdb) p 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.
(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


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


(Pdb) wat()
Local variables:
__builtins__: dict = {…
__file__: pdb._ScriptTarget = 'C:\Users\mate\repo\week04\main.py'
__name__: str = '__main__'
__pdb_convenience_variables: dict = {…
__return__: NoneType = None
__spec__: NoneType = None
process_contacts: function = <function process_contacts at 0x00000259DB95F6A0>
wat: wat.inspection.inspection.Wat = <WAT Inspector object>
(Pdb) wat / process_contacts
value: <function process_contacts at 0x00000259DB95F6A0>
type: function
signature: def process_contacts(input_file, output_file)
(Pdb)