实验报告 2
大杂烩

洪子翔

2024 年 9 月 16 日

目录

1 实验内容

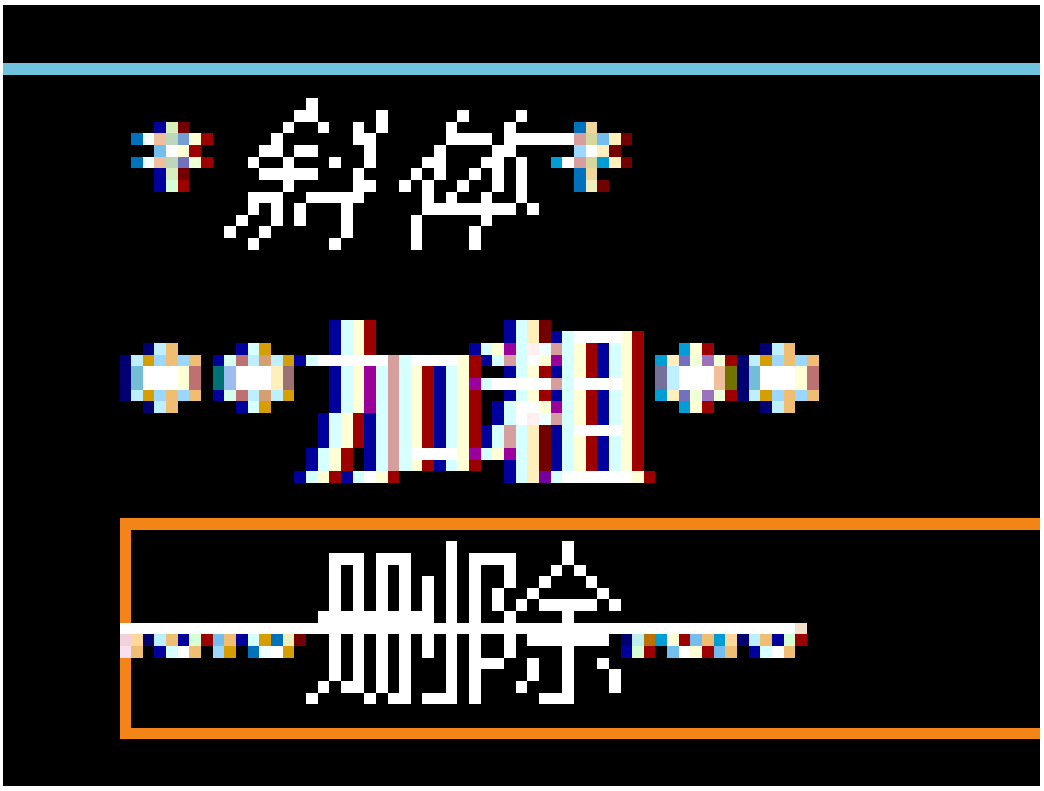
链接：https://github.com/SoraRosa0514/system-dev-tools/tree/main/4

1.1 MarkDown

MarkDown 是一种轻量级标记语言，使用时只需要键盘输入即可相当便捷流畅。
此处使用 VScode 作为编辑器。

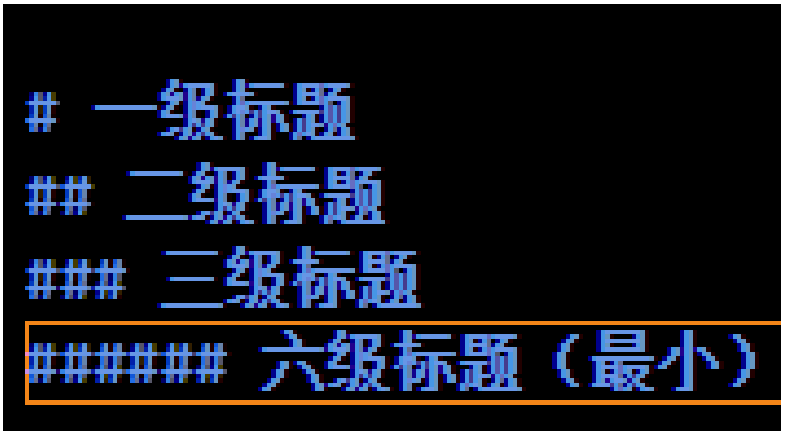
1.1.1 加粗、斜体，删除线

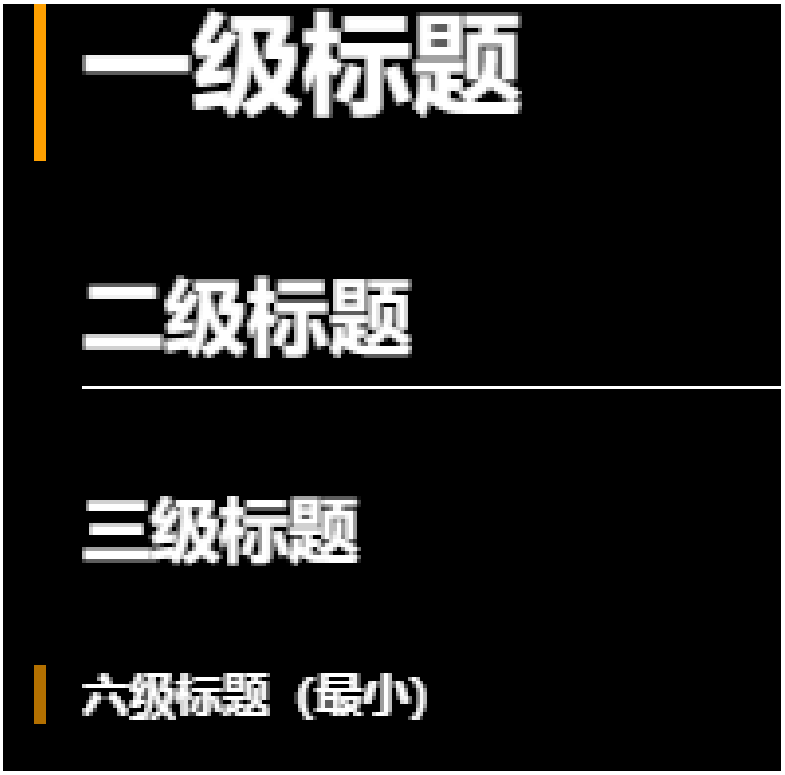
标记分别为`**<text>**` `*<text>*` `~~<text>~~`



1.1.2 标题

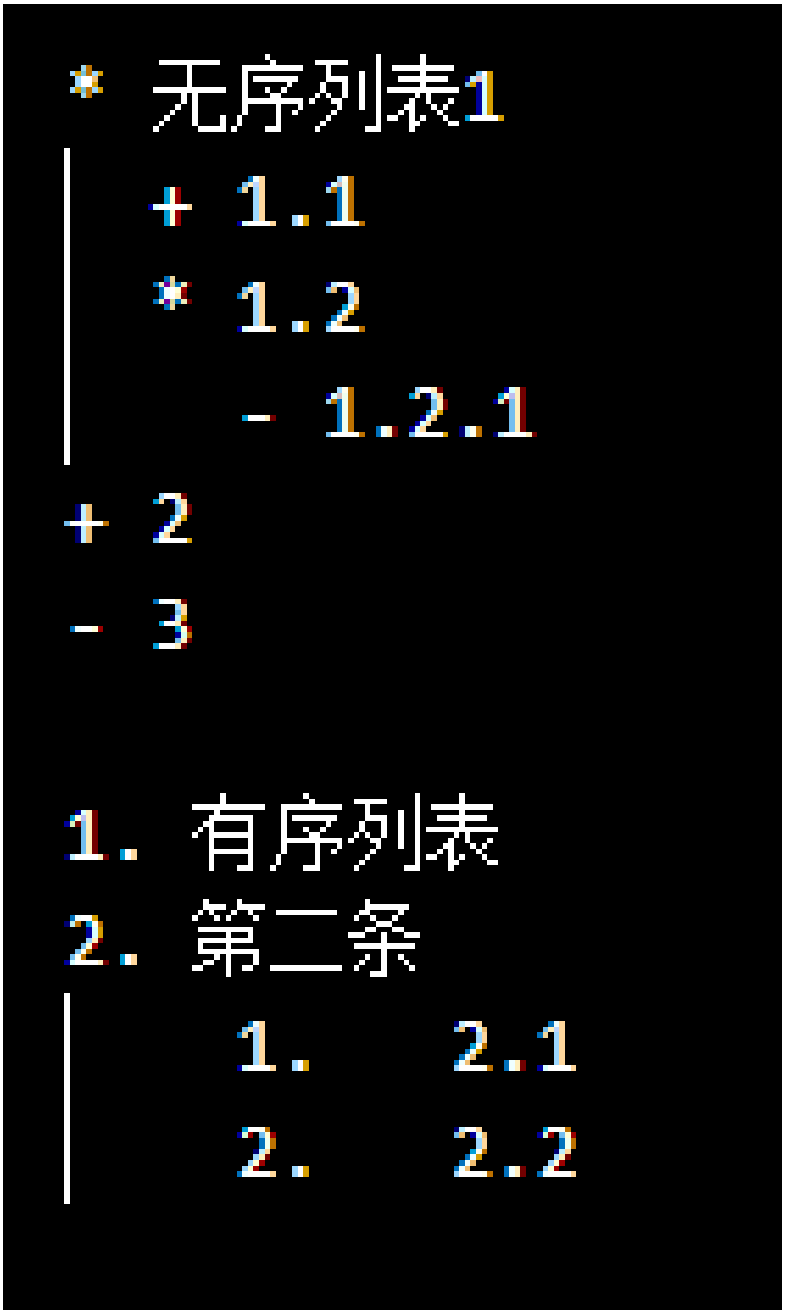
标题在文字前方用标记，的个数代表级别，级别越高字号越小，最高六级。



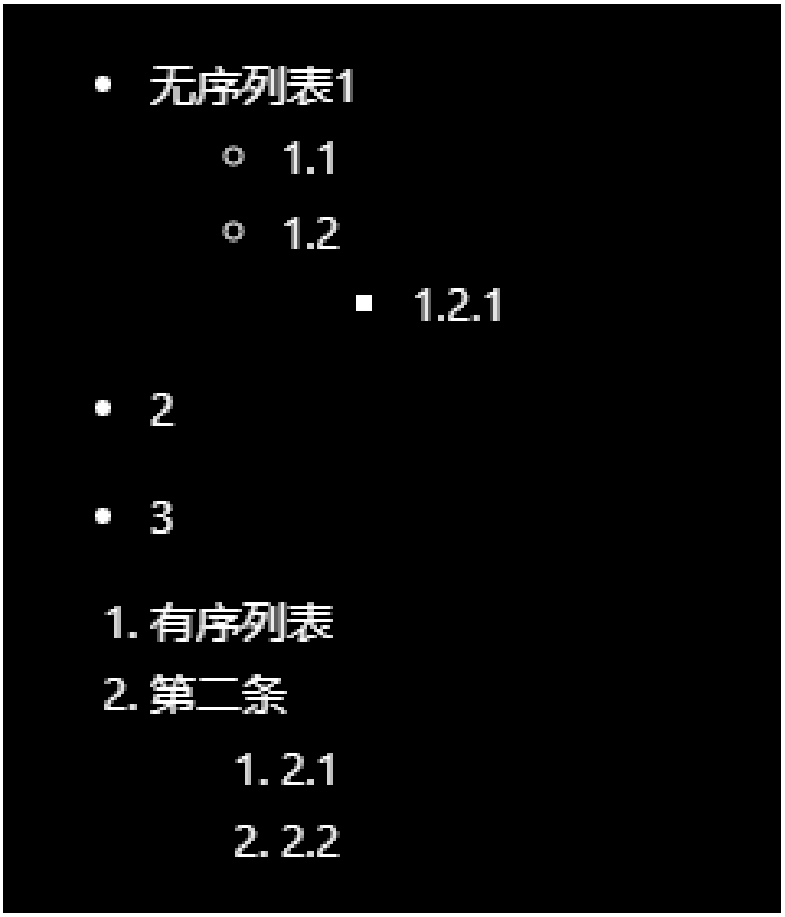


1.1.3 列表

开头用-标记为无序标题，数字加. 为有序标题，开头每有 2 个空格代表缩进一级。
可以在无序列表内设有序列表，反之亦然。

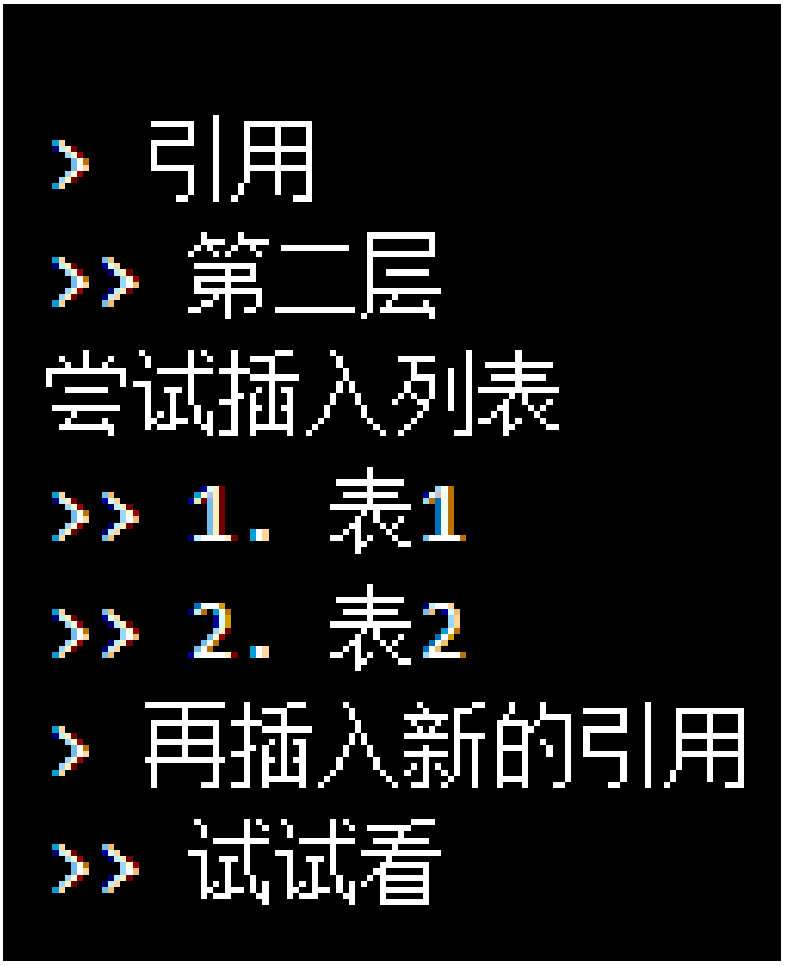


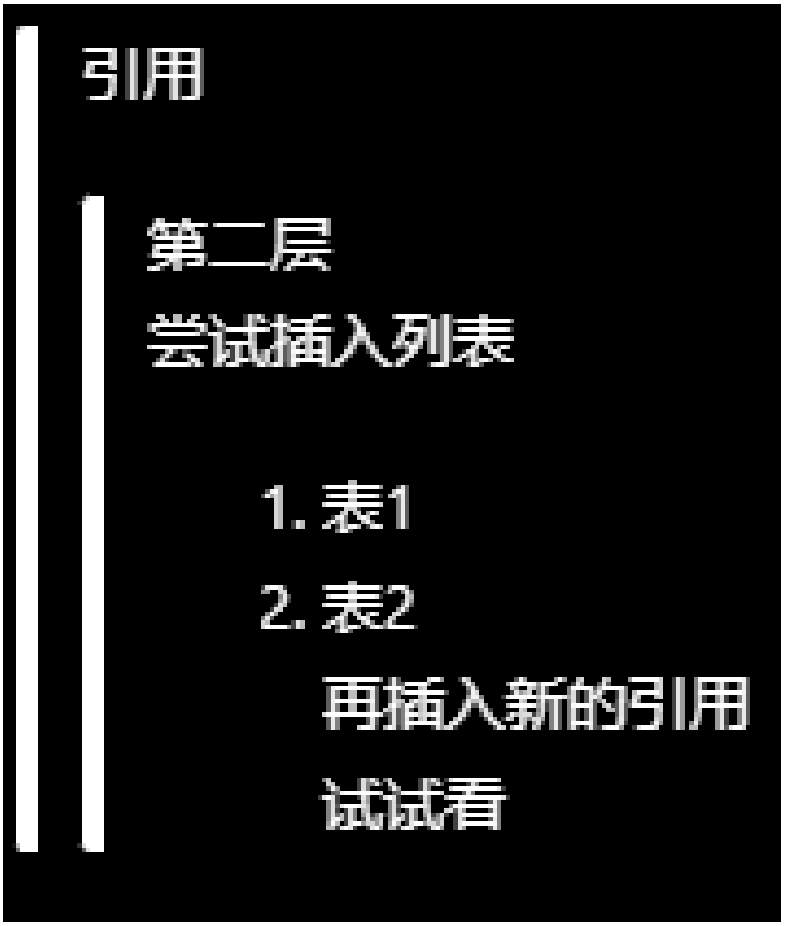
H



1.1.4 引用（区块）

开头用 > 标记代表引用，同样用 > 的个数代表层次，在引用内使用其他语法或反之都可行。


> 引用
>> 第二层
尝试插入列表
>> 1. 表1
>> 2. 表2
> 再插入新的引用
>> 试试看



1.1.5 链接、段落与分割线

链接行内式：［链接文字](链接 "链接标题（可选）")
换行只需要在行末输入两个空格再回车，区分段落则需要段落上下至少有一个空白行
一行只包含三个以上的 * 或-即可，可以插入空格，两种符号不混用

[H]
````
测试代码`sudo rm -rf /*`
```

#include <iostream>
using namespace std;
int main(){
    cout << "Hello MD" << endl;
    return 0;
}
```
````

````
---

这是学校的 [官网](https://www.ouc.edu.cn/main.htm)

上面的内容是
一
个
段
落

* * **** *
````

````
> 这是学校的 官网
>
> 上面的内容是
> 一
> 个
> 段
> 落
````

1.1.6 代码

行内代码由 1 个反引号括起来，段落代码则在段落首行前与末行后加三个反引号。

测试代码 `sudo rm -rf /*`

```cpp
#include <iostream>
using namespace std;
int main(){
    cout << "Hello MD" << endl;
    return 0;
}
```

1.1.7 图片与链接图片

普通的图片插入格式与链接相同，在开头加上! 即可
添加图片链接，则将上述内容用中括号括起，之后加 (链接)

1.1.8 表格

输出表格只需要模仿表格的样式来，大致就是正确的

语法	描述
Header	Title
Paragraph	Text

此处横线分隔的上面部分是表格标题，下面是内容。

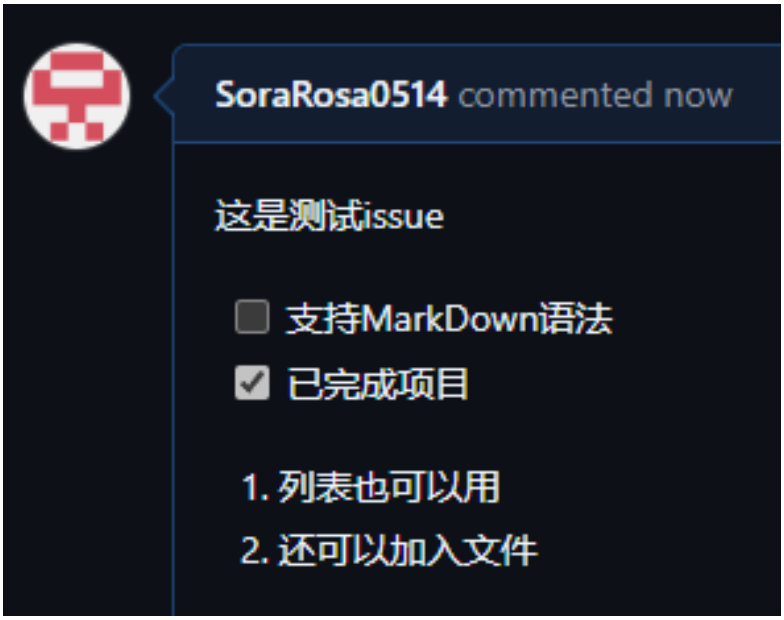
如果要处理对齐，在横线左右对应位置加:，居中两边都加。

1.2 GitHub

1.2.1 issue

issue 可以记录待办事项和任务，也可以向开发者反馈错误
在 github 的仓库页面选择 issue->new issue，输入标题和内容即可


Add a title
Test issue 1
Add a description
Write
Preview
这是测试issue
- [ ] 支持MarkDown语法
- [x] 已完成项目
1. 列表也可以用
2. 还可以加入文件



完成之后，将 issue 调成 close 就会归档。

1.2.2 fork 与 pull request

fork 可以将某个项目复刻到自己的仓库，在本地修改后，使用 pull request 就可以将更改提交给原开发者。他们可以与提交者交流反馈，也可以接受提交。
许多开源项目都通过这种方式接受社区贡献。

1.3 调试与性能分析

1.3.1 日志配置

此处以 python 为语言。可以使用 python 自带的 logging 模块生成日志。
使用`import logging`导入模块，之后配置，示例如下：

```python
# 设置日志配置
logging.basicConfig(
    level=logging.DEBUG, # 日志的级别，有DEBUG INFO WARNING ERROR CRITICAL等级别
    format='%(asctime)s - %(levelname)s - %(message)s',  # 日志格式，此处输出时间、级别与日志消息
    handlers=[ # 配置输出到何处
        logging.FileHandler('debug.log'),  # 输出到文件
        logging.StreamHandler()  # 输出到控制台
    ]
)
```

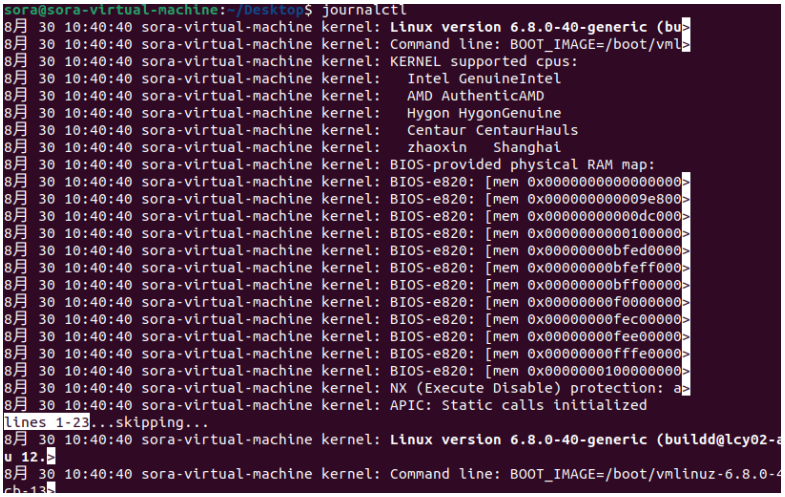
1.3.2 日志记录与异常处理

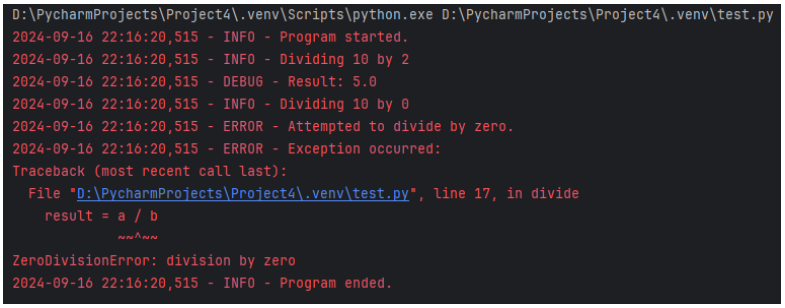
日志输出使用`logging.info()` `logging.debug()` `logging.error()`等代码要捕获异常信息，需要写 try-catch 语句，类似于 C++

```python
# 以除法函数为例:
def divide(a, b):
    logging.info(f"Dividing {a} by {b}")
    try:
        result = a / b
        logging.debug(f"Result: {result}")
        return result
    except ZeroDivisionError as e:
        logging.error("Attempted to divide by zero.")
        logging.exception("Exception occurred:")  # 输出异常信息
        return None

# 主程序
if __name__ == "__main__":
    logging.info("Program started.")
    divide( a: 10,  b: 2)  # 正常情况
    divide( a: 10,  b: 0)  # 错误情况
    logging.info("Program ended.")
```

输出如下:





1.3.3 第三方日志

许多操作系统自带日志，例如 linux
使用`journalctl`显示 system log

1.3.4 调试器

同样以 python 举例，使用标准配置器 pdb

```
1
2
3
4
```

```python
import pdb
def add(a, b):
    return a + b
```

```
5                   def main():
6                       x = 5
7                       y = 10; pdb.set_trace()  # 在此处设置断点
8                       result = add(x, y)
9                       print(f"Result: {result}")
10
11                  if __name__ == "__main__":
12                      main()
```

运行到有`pdb.set_trace()`处，就可以输入命令来调试了
常用命令如下：

- `h` 或 `help`：显示帮助信息。
- `n` 或 `next`：执行下一行代码。
- `c` 或 `continue`：继续执行，直到下一个断点。
- `s` 或 `step`：进入函数内部。
- `q` 或 `quit`：退出调试器。
- `p` 或 `print`：打印变量的值，例如 `p variable_name`。
- `l` 或 `list`：查看当前代码行的上下文。
- `b` 或 `break`：设置断点，例如 `b 12` 在第 12 行设置断点。
- `cl` 或 `clear`：清除断点，例如 `cl 1` 清除第 1 个断点。

```
(Pdb) l
  3         return a + b
  4
  5     def main():
  6         x = 5
  7         y = 10; pdb.set_trace()  # 在此处设置断点
  8  ->     result = add(x, y)
  9         print(f"Result: {result}")
 10
 11     if __name__ == "__main__":
 12         main()
[EOF]
(Pdb) c
Result: 15
```

1.3.5 静态分析

在程序执行之前使用程序自动分析代码，可以提前发现问题，使代码更规范

此处使用 pyflakes，分析下面的代码错误：

```
import time

def foo():
    return 42

for foo in range(5):
    print(foo)
bar = 1
bar *= 0.2
time.sleep(60)
print(baz)
```

```
PS C:\Users\ASUS> pyflakes D:\PycharmProjects\Project4\.venv\test3.py
D:\PycharmProjects\Project4\.venv\test3.py:6:5: redefinition of unused 'foo' from line 3
D:\PycharmProjects\Project4\.venv\test3.py:11:7: undefined name 'baz'
```

程序的错误之处已经被标注

1.3.6 程序计时

使用 python 自带的计时模块，输出 real,user 和 system（后两项仅限 linux）时间

```python
import time

def example_function():
    total = 0
    for i in range(1, 1000000):
        total += i
    return total

start_time = time.time()
result = example_function()
end_time = time.time()

real_time = end_time - start_time

print(f"Result: {result}")
print(f"Real time: {real_time:.6f} seconds")
```

```
D:\PycharmProjects\Project4\.v
Result: 499999500000
Real time: 0.024156 seconds

进程已结束，退出代码为 0
```

一般来说，程序实际用时 =user+system，real 包含其他进程运行等干扰。

1.4 杂项

1.4.1 键位映射

以 Windows 为例，使用 PowerToys-> 键盘管理器-> 重新映射键

2 心得

通过这节课的学习，我学到了许多新的东西，相信他们会在之后的学习和工作中发挥大用处。