《Angr求解方法》实验报告

姓名：汤清云 学号：2013536 班级： 1075

**实验名称：**

**Angr求解方法。**

**实验要求：**

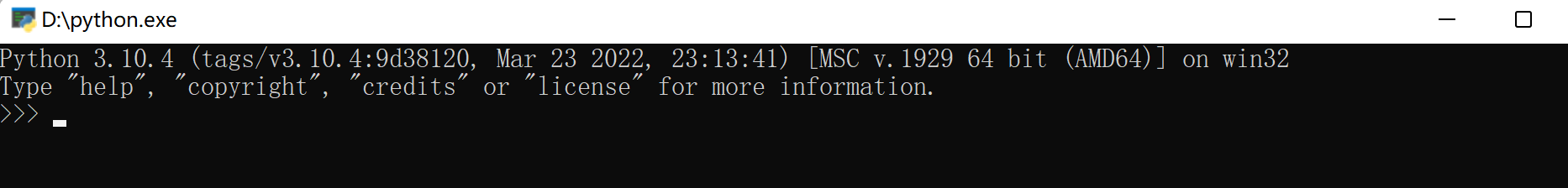
**学习使用Angr完成两种方式的对问题的求解。**

**就如何使用Angr和如何使用Angr进行实际问题求解做出思考。**

**实验过程：**

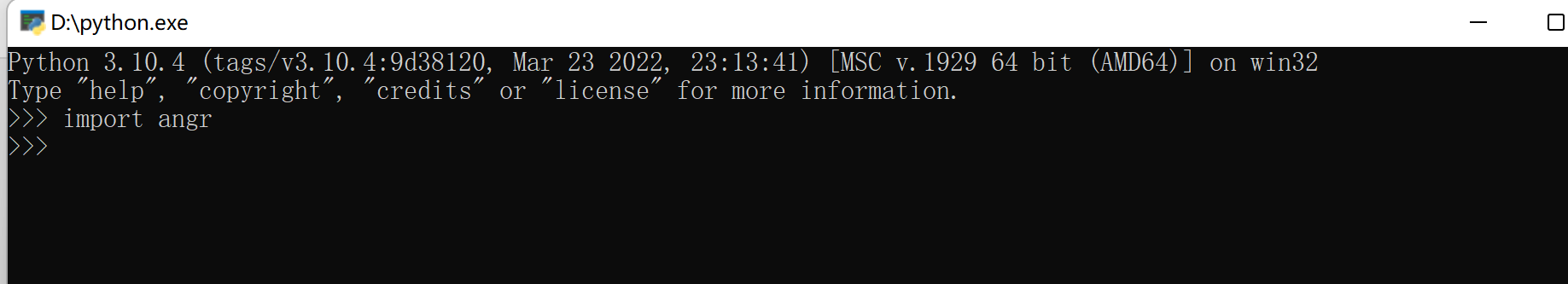
1. **安装Angr：**

**在windows系统下安装python3（此处我安装了python最新版），添加python环境变量。使用win+R输入python如图：**



**在控制台输入pip语句安装angr：**

**语句为：pip install angr**



1. **分析代码：**

**原代码片段为：**

|  |
| --- |
| #include <stdio.h>  char u=0;  int main(void)  {  int i, bits[2]={0,0};  for (i=0; i<8; i++) {  bits[(u&(1<<i))!=0]++;  }  if (bits[0]==bits[1]) {  printf("you win!");  }  else {  printf("you lose!");  }  return 0;  } |

**C++中的<<为左移，1<<i即将i左移一位。&即按位与,for循环即为将u的每一位都取出来比较，如果为0，则bit[0]++,否则就bit[1]++。显然只有当u的二进制中1的个数与0的个数相同时才能胜利。**

**使用python求解，第一种方法代码为：**

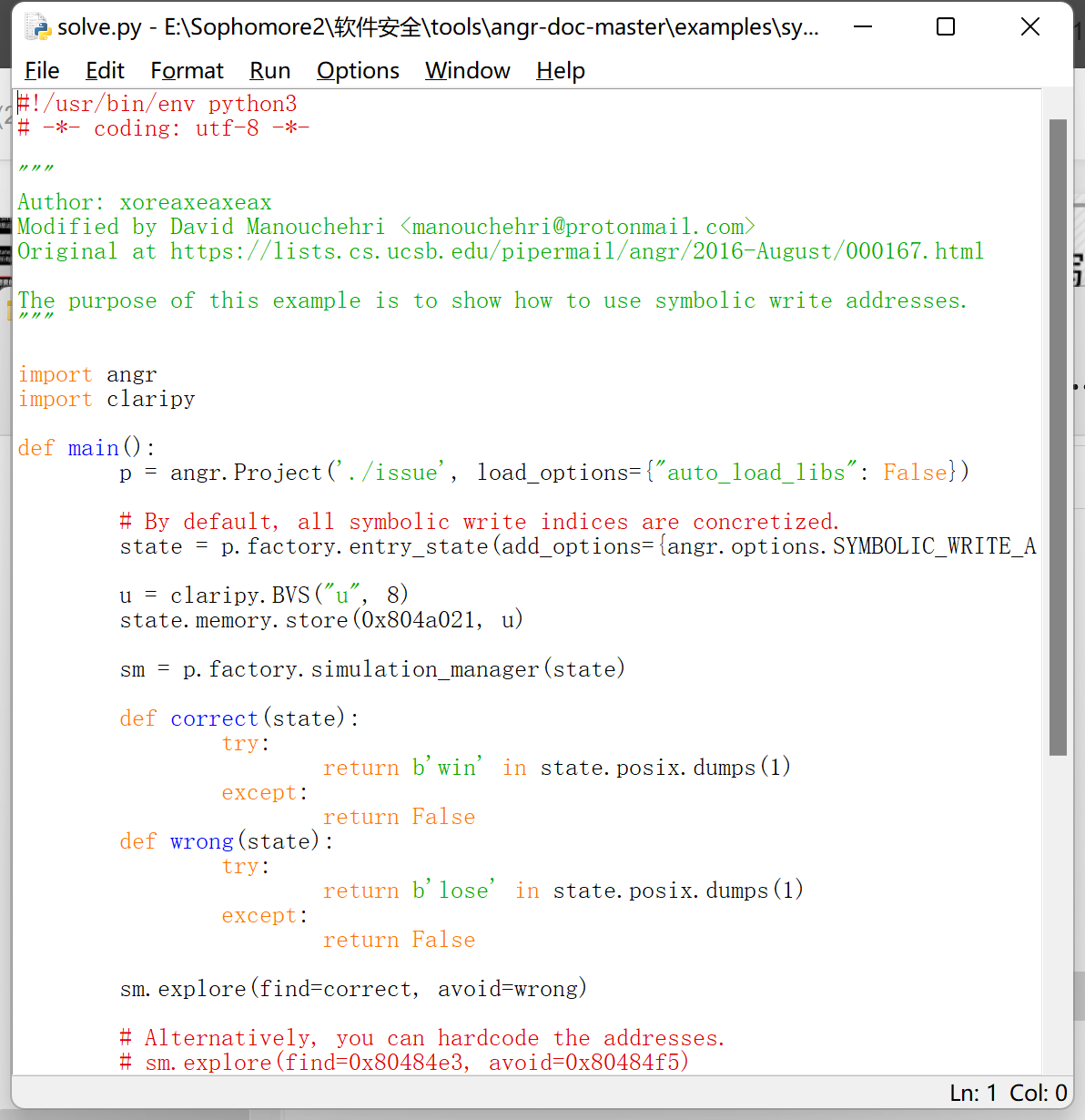
|  |
| --- |
| **import angr**  **import claripy**  **def main():**  **p = angr.Project('./issue', load\_options={"auto\_load\_libs": False}) #新建工程，导入二进制文件issue，选择“不自动加载依赖项”**  **state = p.factory.entry\_state(add\_options={angr.options.SYMBOLIC\_WRITE\_ADDRESSES})#初始化模拟程序状态的simstate对象state，用以记录程序运行时的动态数据，包括内存、寄存器和符号信息等。**  **u = claripy.BVS("u", 8)#创建符号变量u，以8为bitvector形式存在**  **state.memory.store(0x804a021, u)#将其存储到u的地址上。**  **sm = p.factory.simulation\_manager(state)#创建一个Simulation Manager对象，管理运行得到的状态对象**  **def correct(state):**  **try:**  **return b'win' in state.posix.dumps(1)#获得所有使得输出结果为win的标准输出**  **except:**  **return False**  **def wrong(state):**  **try:**  **return b'lose' in state.posix.dumps(1)**  **except:**  **return False**  **sm.explore(find=correct, avoid=wrong)#使用动态符号执行，使用explore函数进行状态搜寻，只找出结果为win的输出，避免结果为lose的输出。**  **# 也可以改成以下语句：**  **# sm.explore(find=0x80484e3, avoid=0x80484f5)**  **return sm.found[0].solver.eval\_upto(u, 256)#约束求解，得到state后通过solver求解器求解u的值。**  **def test():**  **good = set()**  **for u in range(256):**  **bits = [0, 0]**  **for i in range(8):**  **bits[u&(1<<i)!=0] += 1**  **if bits[0] == bits[1]:**  **good.add(u)#将能输出win的结果加入good集合中**  **res = main()**  **assert set(res) == good**  **if \_\_name\_\_ == '\_\_main\_\_':**  **print(repr(main()))** |

**第二种方法代码为：**

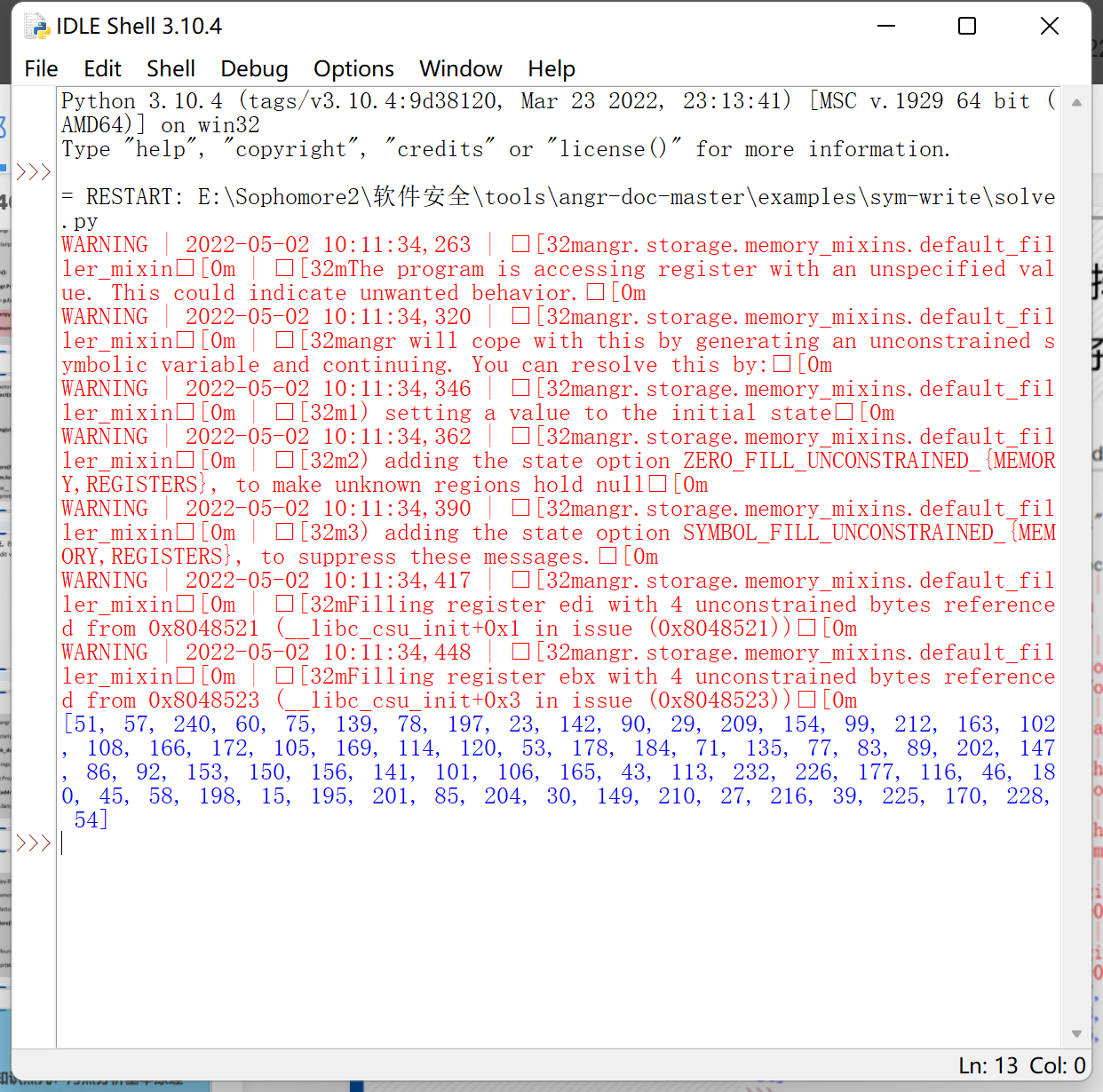
|  |
| --- |
| **import angr**  **import claripy**  **def hook\_demo(state):**  **state.regs.eax=0**  **p = angr.Project('./issue', load\_options={"auto\_load\_libs": False})**  **p.hook(addr=0x08048485,hook=hook\_demo,length=2)**  **# By default, all symbolic write indices are concretized.**  **state = p.factory.blank\_state(addr=0x0804846B,add\_options={"SYMBOLIC\_WRITE\_ADDRESSES"})**  **u = claripy.BVS("u", 8)**  **state.memory.store(0x804a021, u)**  **sm = p.factory.simulation\_manager(state)**  **sm.explore(find=0x080484DB)#此处指定分支语句条件，因为我们已经确定唯一路径**  **st=sm.found[0]**  **print(repr(st.solver.eval(u)))#eval（u）只输出一个结果** |

1. **实验验证：**

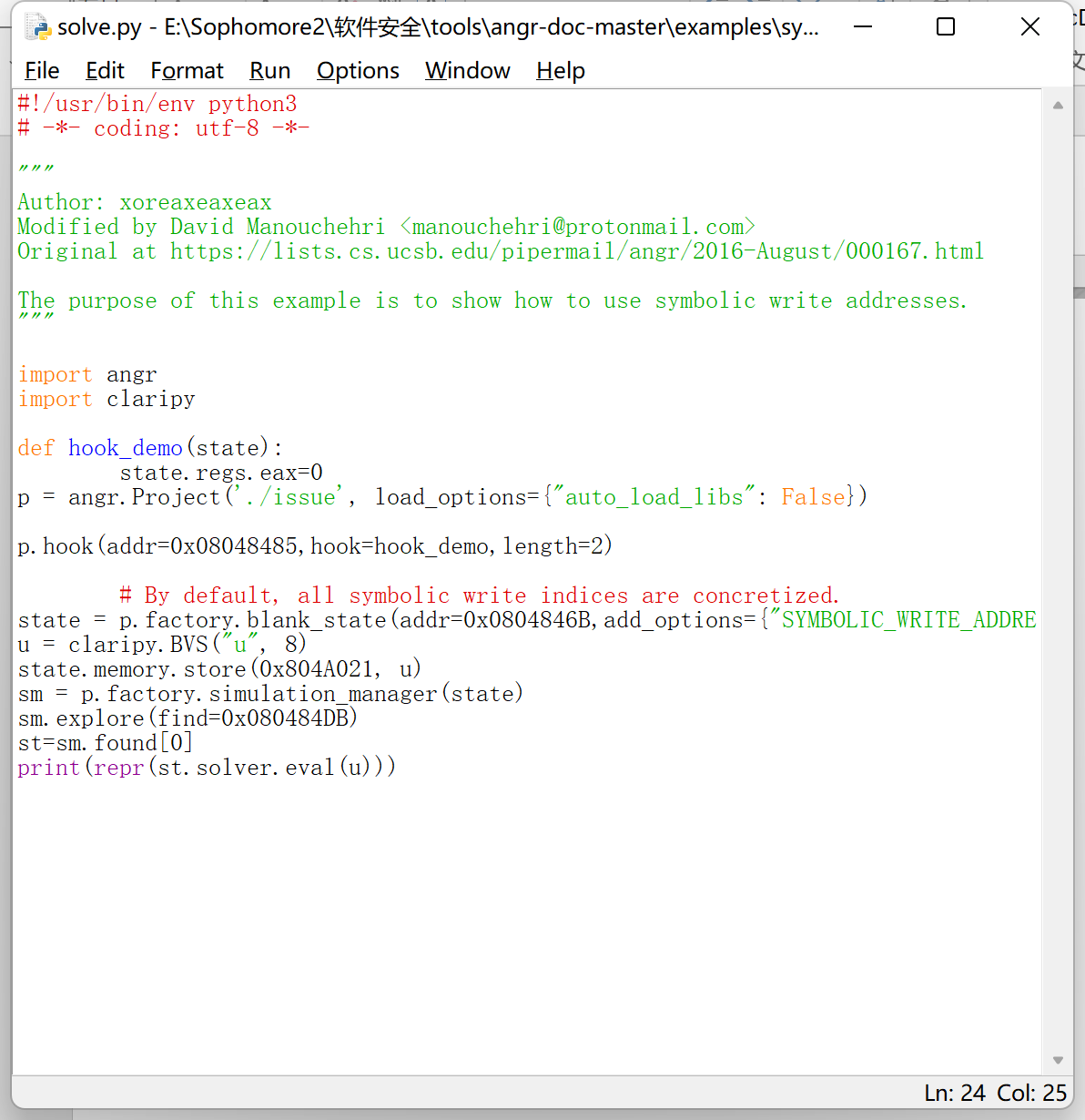
**第一种方法：右键点击solve.py文件，选择Edit with IDLE，可弹出以下对话框**



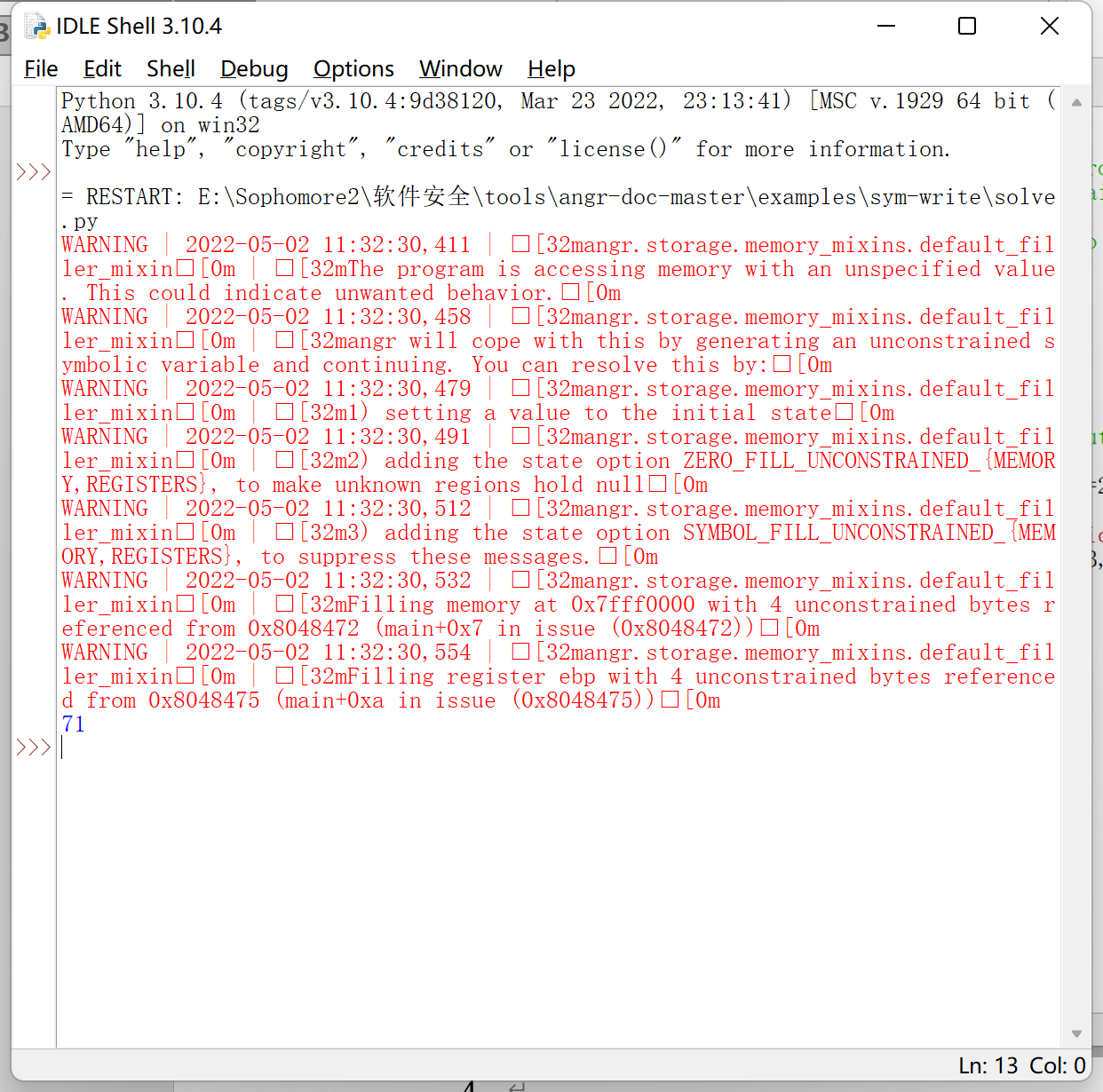
**选择Run-run module:**



**第二种方法：**



**运行结果为：**



**心得体会：**

**Angr是一个基于python的二进制漏洞分析框架，它将以前多种分析技术集成进来，既能够执行动态的符号执行分析，也能够进行多种动态分析。**

**如何使用Angr：可以从Angr的example中分析其运行代码逻辑，学习代码编写思路，参考官方网站所给出的指导手册进行Angr的编写使用。在CSDN网站上也有很多人已经总结了较为完善的基于python的Angr编写，可以进行参考。**

**如何使用Angr解决实际问题：可以在Angr之上开发以下工作：如angrop：rop链自动化生成器；Patcherex：二进制文件自动化patch引擎；Driller：用符号执行增强AFL的下一代fuzzer等。**