Programming Tips

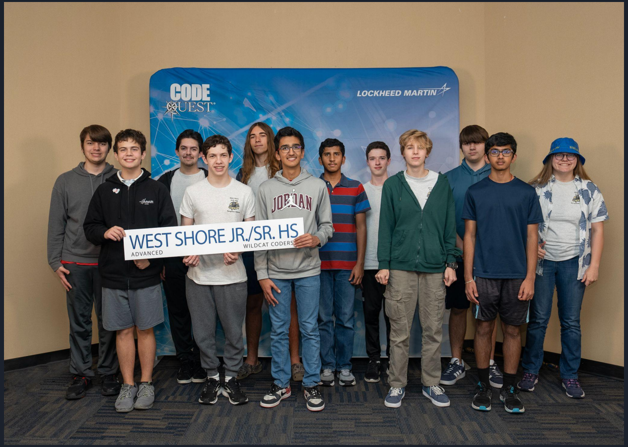
Level 0x0b

Quick Overview

- Events
- Hacker History
- Coding Tips

Upcoming Events

- Code Quest
  - Saturday, March 1
  - Teams are 2-3 people
- Pico CTF (online) March 7th - 17th
- Spring Break. March 17-21st
- Cyber Quest
  - March 29
  - Teams are 3-5 students
- Lockheed forms DUE!!!!
- Band Trip to Disney?



Code Quest

- 4 Teams (2-3 students per team), 5th team is maybe / waitlist
  - Team name
  - Novice or Advanced
- Light breakfast, and lunch are provided
- Each team ~~member~~ can bring 1 computer, 1 monitor
- No cell phones, no cameras, no family members
- Lockheed Requirements
  - Bring IDs / Passports
  - Non-US citizens are allowed (need additional information for registration)
  - Liability Waiver and Photo Release
- Brevard County Public Schools
  - Field Trip Permission Form for BCPS

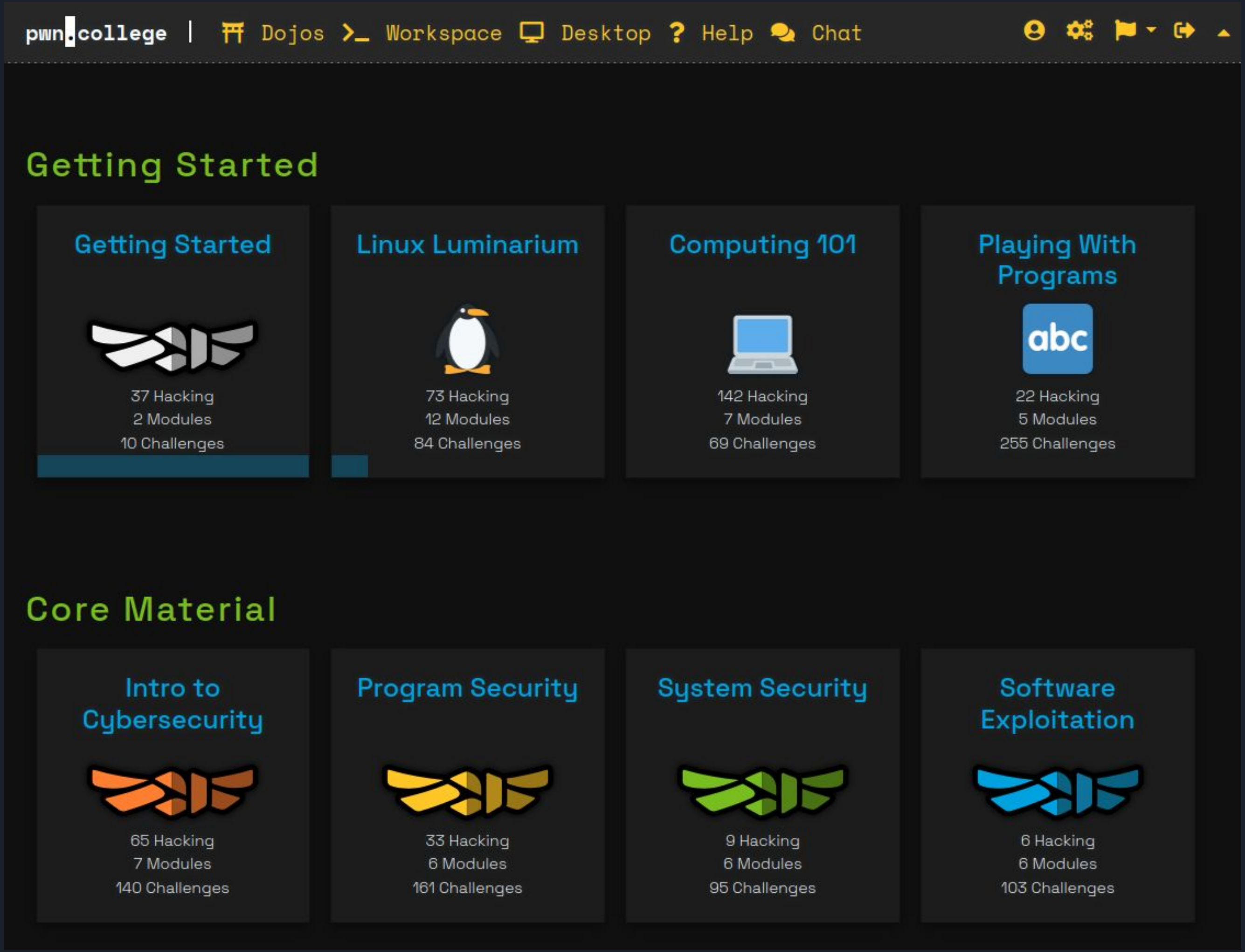
Yan Shoshitaishivili
aka zardus

CTF RADIOOO

- CTF Competitor with Shellphish for 11 years
- Helped run DEFCON CTF for 6 years (Order of the Overflow)
- 3rd Place DARPA AI Cyber Grand Challenge
- Co-host of CTF RadiOOO (youtube)
- Professor at Arizona State University
- Created pwn.college

pwn.college

Learn to Hack!



Getting Started

Getting Started

37 Hacking
2 Modules
10 Challenges

Linux Luminarium

73 Hacking
12 Modules
84 Challenges

Computing 101

142 Hacking
7 Modules
69 Challenges

Playing With Programs

abc

22 Hacking
5 Modules
255 Challenges

Core Material

Intro to Cybersecurity

65 Hacking
7 Modules
140 Challenges

Program Security

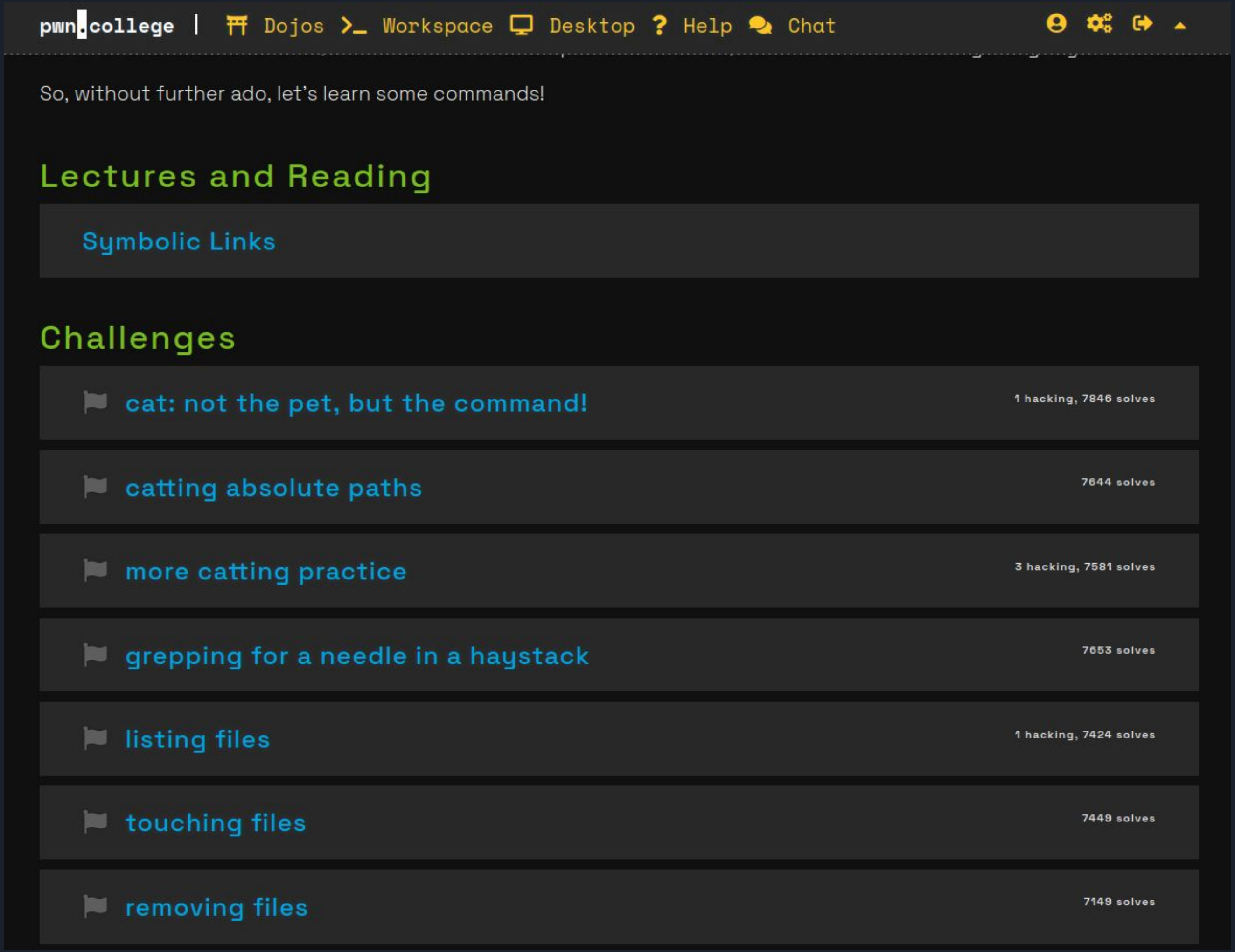
33 Hacking
6 Modules
161 Challenges

System Security

9 Hacking
6 Modules
95 Challenges

Software Exploitation

6 Hacking
6 Modules
103 Challenges



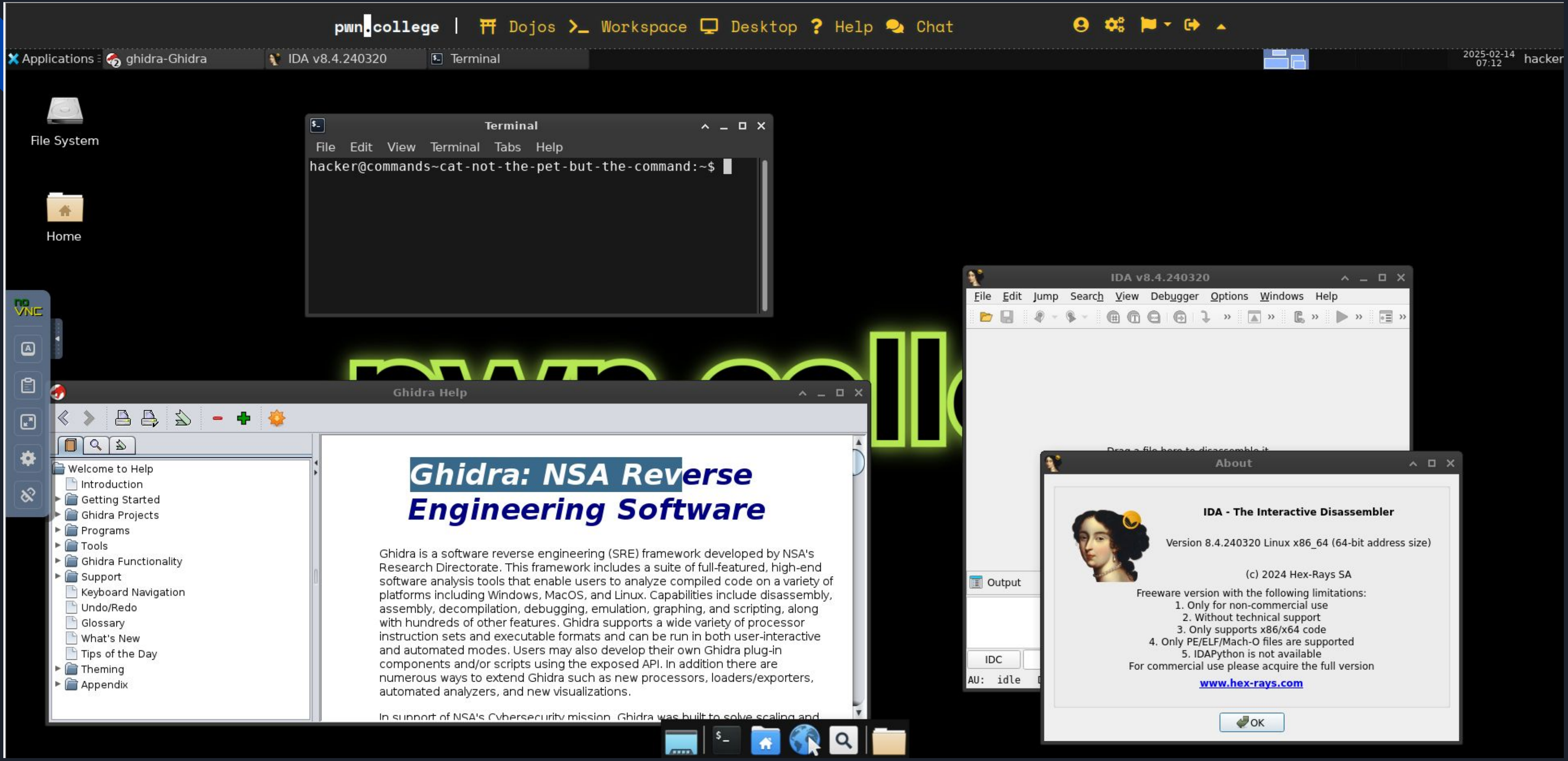
So, without further ado, let's learn some commands!

Lectures and Reading

Symbolic Links

Challenges

cat: not the pet, but the command!	1 hacking, 7846 solves
catting absolute paths	7644 solves
more catting practice	3 hacking, 7581 solves
grepping for a needle in a haystack	7653 solves
listing files	1 hacking, 7424 solves
touching files	7449 solves
removing files	7149 solves



```
hacker@commands~cat-not-the-pet-but-the-command:~$
```

Ghidra: NSA Reverse Engineering Software

Ghidra is a software reverse engineering (SRE) framework developed by NSA's Research Directorate. This framework includes a suite of full-featured, high-end software analysis tools that enable users to analyze compiled code on a variety of platforms including Windows, MacOS, and Linux. Capabilities include disassembly, assembly, decompilation, debugging, emulation, graphing, and scripting, along with hundreds of other features. Ghidra supports a wide variety of processor instruction sets and executable formats and can be run in both user-interactive and automated modes. Users may also develop their own Ghidra plug-in components and/or scripts using the exposed API. In addition there are numerous ways to extend Ghidra such as new processors, loaders/exporters, automated analyzers, and new visualizations.

In support of NSA's Cybersecurity mission, Ghidra was built to solve scaling and

IDA - The Interactive Disassembler

Version 8.4.240320 Linux x86_64 (64-bit address size)

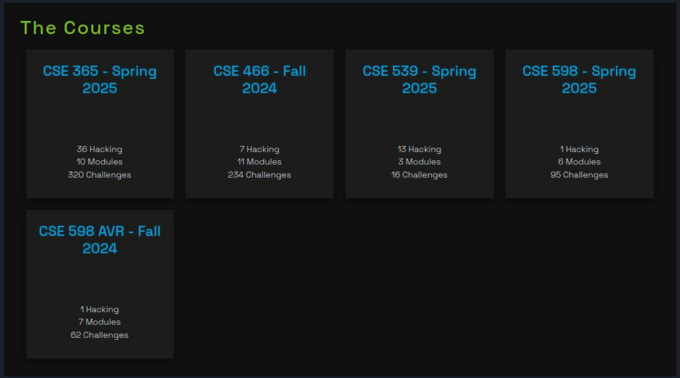
(c) 2024 Hex-Rays SA

Freeware version with the following limitations:
1. Only for non-commercial use
2. Without technical support
3. Only supports x86/x64 code
4. Only PE/ELF/Mach-O files are supported
5. IDAPython is not available
For commercial use please acquire the full version
www.hex-rays.com

Arizona State University Classes

- Many of the classes have lectures / presentations online (twitch / youtube)
- Homework assignments / labwork through pwn.college

- CSE 365 - Information Assurance (Intro to Cyber Security)
- CSE 466 - Computer Systems Security
- CSE 539 - Applied Cryptography
- CSE 598 - Distributed Software Development



The Courses

CSE 365 - Spring 2025
36 Hacking
10 Modules
320 Challenges

CSE 466 - Fall 2024
7 Hacking
11 Modules
234 Challenges

CSE 539 - Spring 2025
13 Hacking
3 Modules
16 Challenges

CSE 598 - Spring 2025
1 Hacking
6 Modules
95 Challenges

CSE 598 AVR - Fall 2024
1 Hacking
7 Modules
62 Challenges

Capturing standard error (stderr)

- If your program has lots of output, errors or warnings might be easily missed
-
  ```
  ./my_program 2>error_log.txt
  Normal output here
  ```
-
  ```
  cat error_log.txt
  Configuration error, using default security settings
  ```
- Compiler errors warnings use stderr
  - Easy to lose warnings when compiling a lot of files
  - Can miss errors when you are compiling many files in parallel
  - Wait till you see the pages that GCC spews when your C++ code goes wrong…

stdin vs stderr

- Different output streams you can write to from your program, both show up in shell
- stderr traditionally used for error messages
- stdout used for normal / nominal output

C / C++	Python3	Java

printf(“data for stdout\n”);
std::cout << “hello stdout\n”;
	print(“data for stdout”)
	
	import sys
	sys.stdout.write(“hi stdout\n”)
		System.out.println(“normal”);

fprintf(stderr, “its stderr\n”);
std::cerr << “more stderr\n”;
	import sys
	sys.stderr.write(“data for stderr\n”)
		System.err.println(“oh no errr”);

Configurable Debug Output

C / C++

```
#include <stdio.h>

#ifdef DEBUG
        #define debug(...) fprintf(stderr, __VA_ARGS__)
#else
        #define debug(...) if(0) fprintf(stderr, __VA_ARGS__)
#endif

// Compile with debug enabled
// gcc -D DEBUG test.c
```

Python

```
import sys

def debug(msg):
  if (True):
    sys.stderr.write(msg + “\n”)
```

```
$ cat test.c
#include <stdio.h>

#ifdef DEBUG
        #define debug(...) fprintf(stderr, __VA_ARGS__)
#else
        #define debug(...) if(0) fprintf(stderr, __VA_ARGS__)
#endif

int main(int argc, char** argv)
{

        printf("Hello World\n");
        debug("This is an error\n");
        return 0;
}


$ gcc -D DEBUG test.c
$ strings a.out | grep This
This is an error
$ gcc test.c
$ strings a.out | grep This
```

Reading input with Python

- Typically don’t want the prompt part of input

```
singleLine = input(“give me data:”)
```

- Instead, typically use

```
singleLine = sys.stdin.readline()
allText = sys.stdin.read()
# both typically have \n on the end, can remove with .strip()
singleLine = sys.stdin.readline().strip()
```

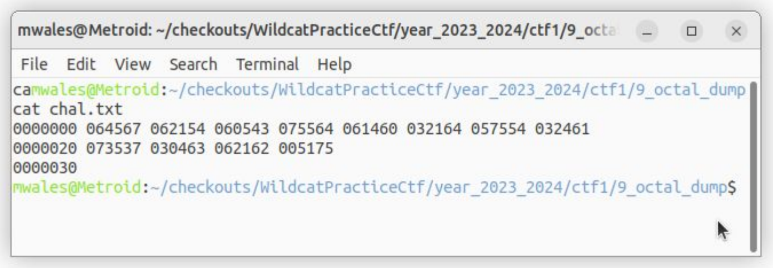
- To read in an integer:

```
myvar = int(sys.stdin.readline())
```

Python Split

```
>>> print(bunchOfText)
This is a bunch
of text on multiple
lines
>>> bunchOfText.split("\n")
['This is a bunch', 'of text on multiple', 'lines']
>>> bunchOfText.split()
['This', 'is', 'a', 'bunch', 'of', 'text', 'on', 'multiple', 'lines']
```

```
#!/usr/bin/env python3

import sys

allLines = sys.stdin.read().strip()

octalData = []
for singleLine in allLines.split("\n"):
        singleLineParts = singleLine.split(" ")
        singleLineParts.pop(0)

        for eachItem in singleLineParts:
                octalData.append(eachItem)

print("OD: {}".format(octalData))


# When I run this
# cat chal.txt | ./solve.py
# OD: ['064567', '062154', '060543', '075564', '061460', '032164',
'057554', '032461', '073537', '030463', '062162', '005175']
```

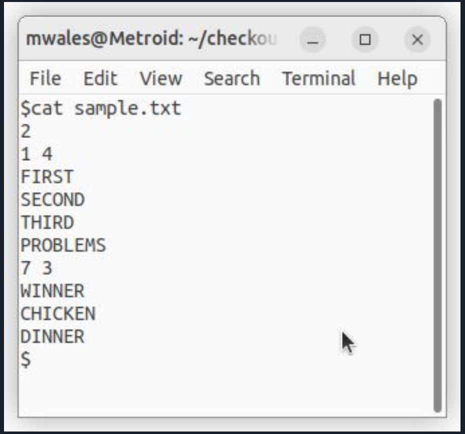

mwales@Metroid: ~/checkouts/WildcatPracticeCtf/year_2023_2024/ctf1/9_octa
File
Edit
View
Search
Terminal
Help

```
camwales@Metroid:~/checkouts/WildcatPracticeCtf/year_2023_2024/ctf1/9_octal_dump
cat chal.txt
0000000 064567 062154 060543 075564 061460 032164 057554 032461
0000020 073537 030463 062162 005175
0000030
mwales@Metroid:~/checkouts/WildcatPracticeCtf/year_2023_2024/ctf1/9_octal_dump$
```

Template / Boilerplate Code

```
#!/usr/bin/env python3

import sys

def debug(msg):
    if True:
        sys.stderr.write(f"{msg}\n")

def addPts(pt0: tuple[int, int], pt1: tuple[int, int]) -> tuple[int,int]:
    return ( pt0[0] + pt1[0], pt0[1] + pt1[1] )

def round_half_to_zero(number, decimals=0):
    factor = 10 ** decimals
    if number > 0:
        return math.floor(number * factor + .5) / factor
    else:
        return math.ceil(number * factor - .5) / factor

def solveTestCase():
    print("do stuff")

def main():
    numTestCases = int(sys.stdin.readline())

    for i in range(numTestCases):
        solveTestCase()

if __name__ == "__main__":
    main()
```

```
#!/usr/bin/env python3

import sys

def processTestCase():
        print("Do TC")
        firstLine = sys.stdin.readline()

        firstLineParts = firstLine.split(" ")
        firstNum = int(firstLineParts[0])
        secondNum = int(firstLineParts[1])

        print("1st = {} and 2nd = {}".format(firstNum, secondNum))

        wordList = []
        for i in range(secondNum):
                wordList.append(sys.stdin.readline().strip())

        print(wordList)



numTestCases = int(sys.stdin.readline())

print("Number of test cases = {}".format(numTestCases))

for i in range(numTestCases):
        processTestCase()
```



```
$cat sample.txt
2
1 4
FIRST
SECOND
THIRD
PROBLEMS
7 3
WINNER
CHICKEN
DINNER
$
```

Attributions

-
-