

## Teaching Case

# Using Python to Inspire Novice Cybersecurity Learners (K-12 and University Level)

Jennifer L. Breese  
jzb545@psu.edu  
The Pennsylvania State University

Brian Gardner  
bkg113@psu.edu  
The Pennsylvania State University

### Abstract

This teaching case will assist in cybersecurity education from K-12 upward including University level educational teaching efforts. There are both initial exercises for getting started and detailed directions to complete analysis for log files. We make the case for Python as a great introductory language for those with little or no programming experience, especially students from lower socio-economic technology-barren environments. Further, we make the connection to Python as a language to develop cybersecurity skills. This teaching case can bring a novice from a basic interest in cyber to a hands-on application understanding in a matter of hours. Further, this case can be combined to create a camp or course for hands-on cybersecurity instruction which is much needed to fill cyber industry workforce gaps.

**Keywords:** Python for cyber, cybersecurity education, cybersecurity programming development, cyber camps, cybersecurity curricula, cybersecurity skill development

### 1. WHY PYTHON

Python is considered to be one of the most popular and in-demand programming languages (Saabith et al., 2020). Python is a powerful, elegant programming language that is easy to both read and understand. It demonstrates most features common to many other languages and is useful for real-world applications. It is also free (Python, 2021). One of the most pressing issues is how to teach programming to beginners without overwhelming novice programmers, especially those from low socio-economic technology deprived environments (Ezeamuzie, 2023). Python is a programming language with a well-organized syntax and strong capabilities for solving any problem (Raj & Paliwal, 2021). The 'language' is similar to basic math reasoning. Raj and Paliwal (2021) reported that in most top institutions, Python is selected as the main

programming language for freshmen. Python has gained increasing popularity with cybersecurity experts due to its adherence to clear and simple syntax, availability of an extensive number of libraries that enables its employment in different applications and code readability (Odetokun et al., 2020). For example, utilizing Python's Cryptography library can enhance the study of the subject area by having students utilize the high-level recipes and low-level interfaces to common cryptographic algorithms such as symmetric ciphers, message digests, and key derivation functions (Bray, 2020).

### 2. USAGE and PURPOSE

This work considers the research findings by Phuong et al. (2023), who introduce the Project-Guided Learning (PGL) model, a wholistic approach to teach cybersecurity concepts. Their

hypothesis was that the learning experience, efficacy of the approach, and skills proficiency of the student would improve through combining student-centered active learning models. The findings were that student-centered active learning models like PGL have shown positive outcomes in the learning experience and engagement of cybersecurity students (Phuong et al. (2023). Sherman et al., (2019) reported findings on how project-based learning provided inspiration for cybersecurity students. Their research was based on a four-day camp and while our exercises are a small slice we contend they too can be added to a similar overall hands-on learning experience.

Research conducted by Henttonen and Rathod (2024) on the importance of programming in cybersecurity based on a 15-week Python course for targeted educational needs was also considered. Similarly to our student population, their students were comprised of diverse backgrounds. The study found positive impacts such as enhanced engagement implementing cybersecurity specific content into a programming course (Henttonen & Rathod, 2024). Eckroth (2018) released a study about teaching a 5-day 25-hour cybersecurity and Python programming camp for high school students including entering college freshman. While their pre and post surveys offered evidence of effectiveness they also stated some student disappointment with rushed curriculum as they covered so many foundational topics and exercises. The researcher's stated changes were made to address student feedback. We too suggest breaking down the module into chunks for better learning if teaching in an abbreviated timeframe rather than a 15-week long course.

### **3. TEACHING CASE DESCRIPTION**

This teaching case can be used in cybersecurity education from K-12 upward, including higher university-level educational teaching efforts. The first exercise introduces students to basic Python language concepts and increases the student's comfort level and understanding of the language overall through a structured programming exercise. Part 2 of the first exercise walks students through developing a basic number guessing game that also includes a primer on iterative development processes. Our exercise has been used in multiple cyber camps hosted by Penn State Schuylkill with K-12 students having little to no prior programming experience. The students follow a process adapted from the

software development life cycle (SDLC) to achieve the activity objectives.

The second exercise incorporates programming skills with cybersecurity education by providing detailed instructions on uploading sample log files and conducting a log file analysis. The second exercise contains "canned" log analysis files and additional learning materials developed for CYBER 262 Cyber-Defense Studio; a course required for second-year students pursuing Penn State's undergraduate cybersecurity degree. This fifteen-module course provides hands-on experience with a variety of cyber defense tools. The Python module is one of the many modules created to develop hands-on cybersecurity skills directly related to practical industry experience.

The combined activities in the two multi-part exercises can provide a quick start for novice programmers to understand both Python and its application to cybersecurity. Again, we make the case for Python as a great introductory language for those with little or no experience with programming. Furthermore, we make the connection to Python as a language for developing cybersecurity skills. These efforts can bring a novice from a basic interest in cyber to a hands-on application understanding within a few instructional hours.

#### **Number Guessing Game**

A simple programming activity that can be used to introduce Python to students with no prior programming experience is to develop a number guessing game. The game's basic objective from the player's perspective is to correctly guess a number within a specific range randomly generated by the program. The scope of the student activity can be tailored to fit the amount of time the facilitator can spend with the students. For example, walking through an exercise to implement a basic game that generates a random number, accepts multiple user guesses, compares user guesses with the random number, and outputs an appropriate message describing the outcome of the game can be accomplished within a two-hour session. Incorporating additional game capabilities such as multi-game/multi-player modes or covering advanced programming techniques such as exception handling and graphical user interfaces can be integrated into additional sessions depending on the format and schedule of the student programming event. Eckroth (2018) published a study of his findings from a five-day cybersecurity and Python summer camp he

facilitated in 2017 which included a similar activity.

The group programming activity yields the best results when it is facilitated with a group of students in person in a controlled lab environment with a basic Python installation on each machine. This activity has also been successfully moderated remotely multiple times over Zoom, although several risks must be taken into consideration. These include the variability of the equipment the students have available to support the activity, the disparity in internet service plans the students have access to (many students in our service area do not have access to speeds that align with the FCC definition of high-speed broadband), and of course the limitations imposed by the meeting software to easily pair students to collaborate on the solution. Students participating in remote programming events like these should receive communication from the event organizer prior to the start of the event that outlines the minimum technical configuration needed to support the activity to minimize any potential issues that can adversely affect the experience.

If a basic Python installation cannot be configured on an available piece of hardware, numerous free online Python programming environments can be used to facilitate the exercise. Some of them include [online-python.com](https://online-python.com), [programiz.com](https://programiz.com), and [replit.com](https://replit.com). Of course, a dedicated internet connection is required to access these platforms.

In addition to introducing the Python programming elements that will be used to develop the game solution, it is also worthwhile to cover the elements of the Software Development Life Cycle (SDLC), so they understand that programming has a process around it. Weaving in the elements of requirements gathering, solution design, and coding and testing the game capabilities is a good way not only to introduce the Python programming language but also exposes the students to a widely used methodology for solving business problems with programming solutions.

To ensure the activity is fun, engaging and educational, we followed this agenda to achieve the goal of getting a basic number guessing game working in a short two-hour session:

1. Collect the program requirements
2. Introduce the Python programming tools
3. Develop the first iteration of the game code
4. Create successive iterations of the original program
5. Capture/implement additional enhancements

### **1. Collect the program requirements**

The student's expectations about programming should be set immediately by describing program development as an iterative process. The first step in the activity is to engage them in a short brainstorming session where they offer ideas about what capabilities should be in the program. This is a great 'ice breaker' that affords the students an opportunity to share ideas about what capabilities the game should be able to perform. The list of requirements should be captured on a whiteboard or flipchart in a classroom setting so that you can review them later in the activity to see how closely your solution matches them. Short excerpts of the requirements can also be captured as comments inside of the program that serves as program documentation as you walk through the exercise.

### **2. Introduce the Python programming tools**

Students are introduced to the Python Integrated Development and Learning Environment (IDLE) program editor and Python interactive shell during this step. It is important to distinguish the purpose of each interface by explaining that the interactive shell can process Python statements individually to verify language syntax as well as display program output (including error/status messages) whereas the IDLE editor will store the entire program code that will be compiled and executed during the activity. Every high-level programming tutorial starts by showing the learner how to display the text string 'Hello World!' in the Python shell window using a simple *print* statement – e.g., `print("Hello World!")`.

### **3. Develop first iteration of game code**

The objective of this step is to build the first iteration of the program by demonstrating basic Python language elements needed to implement simple functions such as generating the random number to be guessed, accepting user inputs as guesses, and adding conditional logic to compare the guess against the random number chosen. The facilitator will methodically work through some of the requirements identified in step 1 and explain the corresponding Python code to implement it. Students will be asked to open the Python IDLE window to capture all the program code along with a demonstration on how to run the program.

The first step is to generate the random number that the player will guess. The program needs to import a Python library containing the code to implement the random number generation capability. A simple `import random` statement appearing at the beginning of the code accomplishes this. Next, the random number can

be generated using the `random.randint(x,y)` function, where 'x' is the lowest number in the range and 'y' is the highest number in the range. It is also important to explain to the students that the random number has to be stored in a memory location on their computer and assigned a user-defined variable name that will be accessed later in the program, e.g., `random_number = random.randint(1,10)`. To demonstrate that their random number function works, the students are asked to display the contents of the variable using a simple print statement like the one they used for the 'Hello World!' activity, e.g. `print(random_number)`. Notice that since we want to print the number that is stored in the `random_number` variable, there should not be any quotes around the variable name. The students will move this print statement to the end of the program after the rest of the program code is developed.

The next step demonstrates the Python input statement that accepts their guess from a prompt that is displayed in the Python shell. The input statement itself is a relatively easy concept to understand and includes the verbiage to be displayed in the text prompt that directs the player to input their guess. One concept that will need explanation is that all inputs captured in the Python shell are stored as character strings. Since the guesses are entered with the numeric keys on the keyboard, this may be a source of confusion for the students. Since the format of the user input and the random number both need to be numeric, conversion of the input from a string literal to an integer is handled simply by wrapping the input statement inside of Python's integer (`int`) function and saving the number to a variable, e.g., `guess = int(input("Please enter a number between 1 and 10"))`.

The final step to get this code snippet working is to explain the concept of conditional logic and how it can change the flow of the program execution. Students should be introduced to the `if` statement and how it can be coded to give the player feedback on whether they guessed the number correctly. The first pass at coding the `if` statement should include a condition that simply tests for inequality between the user's guess and the random number that was picked – e.g., `guess != random_number:`. An appropriate message acknowledging the incorrect guess should be displayed using a print statement right after the `if` statement. An `else` block should also be linked to the `if` statement that simply displays the random number selected by the `random.randint` function if the guess is incorrect.

A sample code snippet showing the code so far can be found in Appendix A under the heading 'Number Guessing Game Sample Code – Iteration 1'.

#### 4. Create successive iterations of the original program

When the students get the basic logic working this far, they will quickly realize that the logic only supports getting feedback on inputting one guess. The last major step in building the next iteration of the program is to add a loop that will execute the guessing logic a specified number of times without replicating the statements multiple times inside of the program. Students should be brought into the discussion again to decide how many guesses the game should allow based on the range of numbers the `random.randint` function is set to choose from. A `for` loop can be added to support validating multiple guesses along with removing the display of the secret number from the `print` statement in the `else` block so that it is not revealed after each wrong guess. For example, inserting the input statement and `if-else` logic inside of a loop that starts with the statement `for i in range(3):` is a simple way to execute this logic 3 times. It uses the variable `i` as a counter that keeps track of how many times the code inside of the loop has been executed until the value of `i` equals 3, i.e., the value in the `range()` function.

One of the remaining issues with the program at this point is that it will continue to ask for guesses even if the player guesses correctly before their last attempt. This can be resolved easily by adding a `sys.exit()` statement immediately after the message that shows the player has guessed the number correctly is displayed. Secondly, the print statement used in the first iteration to display the random number should be added to the end of the program. This final print statement should start in column 1 and should not be indented with the code inside of the loop so that it only executes once if the player does not guess the number within the allotted number of tries.

A sample code snippet showing the next iteration of the code can be found in Appendix A under the heading 'Number Guessing Game Sample Code – Iteration 2'.

#### 5. Capture/implement additional enhancements

The concepts students are exposed to in a two-hour student activity like this are typically covered in an introductory programming course that could ideally span the first half of an academic semester. Students should feel a sense

of accomplishment from getting this small program working. You can continue to encourage discussion around additional requirements for the program and implement them as time permits. One enhancement that does not require much modification from the concepts covered so far is to change the conditional logic to provide feedback on whether the player's guess was higher or lower than the random number. The single if statement that tests for the incorrect guess can be expanded to include two distinct checks for the guess being greater than or less than the random number along with a relevant message that is displayed in the Python interactive shell. Students can also lend their creativity to the output messages that appear as well to make the game more engaging for them. Also, the level of difficulty of the game can be changed by either adjusting the number of guesses the player gets and/or increasing the range of numbers the random number is selected from.

A final copy of a complete Python program that incorporates the concepts discussed in this paper including the advanced validation logic can be found in Appendix A under the heading 'Number Guessing Game Sample Code – Iteration 3'.

### Analyzing Log Files Using Python

This lab requires the student to create a Python script to analyze the two log files provided. To start this lab, students must have the Python shell installed on their native machines. There are four tasks to be completed in the log analysis:

1. Analyzing "read" events
2. Analyzing "read from keyboard" events
3. Analyzing "read from file" events
4. Challenge section for additional learning and bonus points.

Students are required to put the following items in a compressed ZIP archive file and submit to the Learning Management System (LMS):

- Python program(s)
- Outputs or results (in a txt file)

Students are directed to include first and last names in the file name of the ZIP archive. Further, the output from the program can be saved as a txt file after it has been run. Students can use the File tab at the top of the Python interactive shell window to direct the txt file to the desired save location.

Remarks:

1. Students should write a single Python program to print to console all the outputs; alternatively, they may write a separate Python program for each individual task.

### 1. Task 1: Analyzing the "read" events

#### A. Verify Python Installation

After you have successfully installed Python on your machine, create a Python script using IDLE. You can test if your script is working properly by testing a dummy code:

```
print("Hello World!")
```

Then, download the log files from Canvas and make sure that you place them in the same directory as your Python script. This will make your job easier as you would not have to specify the full path of your log files while reading them in your code.

#### B. Open log files and assign a variable.

Now open the files and read their contents to a variable like in the snippet given below.

```
logA = open("Log-A.strace", "r")
logB = open("Log-B.strace", "r")

logA_content = logA.readlines()
logB_content = logB.readlines()
```

Figure 1.1 – Opening and Reading log files

The "open" function has two parameters – File Location and Open mode. Here we have given the filename as the location as it is in the same directory as our script, and the open mode "r" is used to state that we are opening the file in read only mode.

Then we store the content of each file in 2 variables using the "readlines" function.

#### C. Parsing the Files

Now we will parse through the file line by line and find the "read" events.

First we will have to look into the files and find a unique expression that occurs only in the read statements. In this case the unique expression will be "read".

The code snippet below shows how to parse through the content line by line and search for this expression.

```
readExp = " read("
for line in logA_content:
    if readExp in line:
        print('Read from Log A: ' + line)
```

The code above finds the "read" events in the LogA file and prints them.

**1.1** (Question) – What is the total number of "read" events in both of the Logs? Hint: You can do this by incrementing a counter in the for loop.

## 2. Task 2: Analyzing the "read from keyboard" events

### A. Parsing for "read from keyboard" events

Now, you are supposed to find the "read from keyboard" events. We can find these events by looking for "tty" expression in all the "read" events. You can approach this in multiple possible ways. The code snippet below is an example which will help you find the "read from keyboard" events in the LogA file.

```
readExp = " read("
keyboardExp = "tty"
for line in logA_content:
    if readExp in line and keyboardExp in line:
        print("Read from Keyboard in Log A: " + line)
```

**Figure 2.1** – Example of read event showing "tty" expression

**2.1** (Question) – What is the total number of "read from keyboard" events in both of the Logs? Hint: You can do this by incrementing a counter in the for loop.

## 3. Task 3: Analyzing the "read from file" events

### A. Parsing for "read from file" events

In this task we will understand how to select something if we cannot put it into an exact expression.

There are only 3 possible input sources for a "read" event – **keyboard, files or pipe**. It is difficult to make an expression to search for filenames. But we can easily find the "read from keyboard" and "read from pipe" events. Then, the remaining events will be "read from file".

The code snippet below is an example which will help you do the same for LogA file.

```
readExp = " read("
keyboardExp = "tty"
pipeExp = "pipe"
for line in logA_content:
    if readExp in line and keyboardExp not in line and pipeExp not in line:
        print("Read from File in Log A: " + line)
```

**Figure 3.1** – Finding "read from file" events for LogA file.

**3.1** (Question) – What is the total number of "read from file" events in both of the Logs? Hint: You can do this by incrementing a counter in the for loop.

## 4. Task 4 Challenge

If you were able to complete all the above sections, this challenge will just elaborate on that and help you understand how to present your findings.

### A. Finding file Locations

In this particular task you are supposed to find the locations of the files that are present in the "read from file" events.

Every file location is stored between '< >' these braces. This makes it easier for us to look for the file locations in a particular line. The code snippet below is an example on how to do this.

```
readExp = " read("
keyboardExp = "tty"
pipeExp = "pipe"
for line in logA_content:
    if readExp in line and keyboardExp not in line and pipeExp not in line:
        filename_start = line.find('<')
        filename_end = line.find('>')
        filename = line[filename_start + 1:filename_end]
```

**Figure 4.1** – Parsing for filenames

**4.1** (Question) - Challenge Task 1 find the locations of the files that are present in the "read from file" events.

### BONUS - Counting Iterations

After you have found the file names (with full locations included with them), you will see that there are repetitions.

So, in this task you are required to keep track of the number of occurrences of every file and present them in form of a table.

You can do this by creating 2 separate lists – **filenames[]** and **filename\_count[]**. The filenames[i] will have filename\_count[i] occurrences.

A simple algorithm that you can use to accomplish this task is given below:

For every line

- find the filename

- *check if it already exists in the filenames[] lists*
  - *if it does, just increment the filename\_count[] at that index by 1*
  - *if it doesn't, add that filename to filenames[] and set the filename\_count[] at that index to be 1*

You will have to use functions like:

- **append()** – *to append to the lists you have created*
- **index()** – *to search for a filename in filenames[] list and get its index which you can use to increment the relevant counter in the filename\_count[] list.*

**Save the output of your algorithm as a txt file for your lab archive submission. Your output should be similar to the one below.**

Log file 1

```
Filename: /lib/x86_64-linux-gnu/libc-2.23.so, Count: 4
Filename: /etc/nsswitch.conf, Count: 4
Filename: /lib/x86_64-linux-gnu/libnss_compat-2.23.so, Count: 2
Filename: /lib/x86_64-linux-gnu/libnsl-2.23.so, Count: 2
Filename: /lib/x86_64-linux-gnu/libnss_nis-2.23.so, Count: 2
Filename: /lib/x86_64-linux-gnu/libnss_files-2.23.so, Count: 2
Filename: /lib/x86_64-linux-gnu/libselinux.so.1, Count: 1
Filename: /lib/x86_64-linux-gnu/libpcre.so.3.13.2, Count: 1
Filename: /lib/x86_64-linux-gnu/libdl-2.23.so, Count: 1
Filename: /lib/x86_64-linux-gnu/libpthread-2.23.so, Count: 1
Filename: /proc/filesystems, Count: 2
Filename: /etc/locale.alias, Count: 2
Filename: /proc/sys/kernel/ngroups_max, Count: 2
```

Log file 2

```
Filename: /lib/x86_64-linux-gnu/libc-2.23.so, Count: 9
Filename: /home/user/test/in.txt, Count: 2
Filename: /lib/x86_64-linux-gnu/libselinux.so.1, Count: 3
Filename: /lib/x86_64-linux-gnu/libpcre.so.3.13.2, Count: 3
Filename: /lib/x86_64-linux-gnu/libdl-2.23.so, Count: 3
Filename: /lib/x86_64-linux-gnu/libpthread-2.23.so, Count: 3
Filename: /proc/filesystems, Count: 6
```

**BONUS** (Question) - In this BONUS Task you are required to keep track of the number of occurrences of every file and present them in form of a table. The format of your output should be similar to the above example.

#### 4. ADDITIONAL APPLICATIONS/FUTURE RESEARCH

As we mentioned these exercises can and should be combined with additional facilitator-led hands-on projects to create a cohesive learning environment on Python specifically and cybersecurity overall. We intend to use these exercises to publish student perception indicating their learning experiences and actual outcomes in

future work. Further, the differences in mode of instruction, particularly online synchronous versus face-to-face, will be explored.

#### 5. CONCLUSIONS

These exercises were completed in a cyber camp environment and in-class for 100- and 200-level learning. These exercises both prepare novice learners and those moving into concepts in cyber education. We are excited by this material and the broad usage. Further, while these exercises can be used to provide foundational material in a K-12 camp, they can also be used in a 100-level General Education course to draw interest in the major and continue to meet higher level program requirements in 200-level courses and beyond. Further, these exercises will be uploaded to CLARK, the Cybersecurity Labs and Resource Knowledgebase, which is the largest platform that provides free cybersecurity curriculum. This site states, "It is home to high-value, high-impact cyber curriculum created by top educators and reviewed for relevance and quality."

#### 6. ACKNOWLEDGEMENTS

Some of the teaching case instructions, exercises, and files included in this work were developed by David Hozza and J. Andrew Landmesser from Penn State University.

#### 7. REFERENCES

- Bray, S. W. (2020). CHAPTER 1 Introduction to Cryptography and Python in *Implementing Cryptography Using Python*, John Wiley & Sons (p. 8) <https://doi.org/10.1002/9781119612216.ch1>
- Eckroth, J. (2018). Teaching cybersecurity and python programming in a 5-day summer camp. *Journal of Computing Sciences in Colleges*, 33(6), 29-39.
- Ezeamuzie, N. O. (2023). Project-first approach to programming in K-12: Tracking the development of novice programmers in technology-deprived environments. *Education and Information Technologies*, 28(1), 407-437. <https://doi.org/10.1007/s10639-022-11180-8>
- Henttonen, K., & Rathod, P. (2024, May). Importance of Programming in

- Cybersecurity: Preliminary Findings from a Pilot Study Tailoring a Python Course for Targeted Educational Needs. In *2024 IEEE Global Engineering Education Conference (EDUCON)* (pp. 1-6). IEEE. <https://doi.org/10.1109/EDUCON60312.2024.10578580>
- Odetokun, I., & Clarksville, T. N. (2020). AN ASSESSMENT OF THE EFFECTIVENESS OF PYTHON IN CYBERSECURITY.
- Phuong, C., Saied, N., & Yang, L. (2023, October). A Hands-on Education Framework for Cybersecurity. In *2023 IEEE Frontiers in Education Conference (FIE)* (pp. 1-5). IEEE. <https://doi.org/10.1109/FIE58773.2023.10343268>
- Python, W. (2021). Python. *Python releases for windows*, 24.
- Raj, S., & Paliwal, M. (2021). Why Python is Most Famous. *International Journal of Innovative Research in Computer Science & Technology*, 9(6), 234-238. <https://doi.org/10.55524/ijrcst.2021.9.6.52>
- Saabith, A. S., Vinothraj, T., & Fareez, M. (2020). Popular python libraries and their application domains. *International Journal of Advance Engineering and Research Development*, 7(11).
- Sherman, A. T., Peterson, P. A., Golaszewski, E., LaFemina, E., Goldschen, E., Khan, M., & Suess, J. (2019). Project-based learning inspires cybersecurity students: A scholarship-for-service research study. *IEEE Security & Privacy*, 17(3), 82-88. <https://doi.org/10.1109/MSEC.2019.2900595>



## Appendix A – Number Guessing Game Sample Code

### Number Guessing Game Sample Code – Iteration 1

The following is an example of how the code looks after implementing the first round of Python features described in step 3.

```
#import random library to support random number generation
import random

#generate a random number
random_number = random.randint(1,10)
print(random_number)

#prompt the user to input their guess
guess = int(input("Please enter a number between 1 and 10"))

#compare user's guess against random number
if guess != random_number:
    print("Sorry, your guess is incorrect. The number is", random_number)
else:
    print("Congratulations! You won!")
```

### Number Guessing Game Sample Code – Iteration 2

The following is an example of how the code looks after implementing the second round of Python features that support multiple guesses as described in step 4.

```
#import random library to support random number generation
import random

#import sys library to support program exit function
import sys

#generate a random number
random_number = random.randint(1,10)

#logic to compare user guesses against random number
for i in range(3):
    #prompt the user to input their guess
    guess = int(input("Please enter a number between 1 and 10"))
    if guess != random_number:
        print("Sorry, your guess is incorrect")
    else:
        print("Congratulations! You won!")
        sys.exit()

#Display the random number if it is not guessed within the allotted number of turns
print("The number is", random_number)
```

### Number Guessing Game Sample Code – Iteration 3

The following is an example of code that includes the enhanced decision logic that provides more informative prompts to the player about their guess relative to the random number.

```
#import random library to support random number generation
import random

#import sys library to support program exit function
import sys
```

```
#generate a random number
random_number = random.randint(1,10)

#logic to compare user guesses against random number
for i in range(3):
    #prompt the user to input their guess
    guess = int(input("Please enter a number between 1 and 10"))
    if guess < random_number:
        print("Your guess is too low!")
    elif guess > random_number:
        print("Your guess is too high!")
    else:
        print("Congratulations! You won!")
        sys.exit()

#Display the random number if it is not guessed within the allotted number of turns
print("The number is", random_number)
```

## **Appendix B – Cyber 262 Lab: Analyzing Log Files using Python**

### **Tasks**

- 1. Analyzing “read” events**
- 2. Analyzing “read from keyboard” events**
- 3. Analyzing “read from file” events**
- 4. Challenge**

### **Introduction**

This lab requires you to create a Python script to analyze the 2 given Log Files. We are using Python for this lab because it is a relatively easy language that can handle large amounts of data. To start with this lab, you must have Python shell installed in your native machines. You can follow the guidelines provided in the class and on Canvas to install the basic Python interactive shell.

### **Deliverables for Lab**

#### **What to submit:**

- Your Python program(s)
- The outputs or results (in a txt file)

**Please put all these items in a folder and compress the folder into a single ZIP archive file. Include your first and last names in the file name of the ZIP archive. Your output from the program can be saved as a txt file after you run it, refer to the File tab at the top of Python.**

#### **Remarks:**

- You are encouraged to write a single Python program to print to console all the outputs; alternatively, you may write a separate Python program for each individual task.

### **Learning Objectives**

The Learning Objectives for this Lab are outlined in the Canvas Lab Assignment page.

## Task 1: Analyzing the “read” events

### A. Verify Python Installation

After you have successfully installed Python on your machine, create a Python script using IDLE. You can test if your script is working properly by testing a dummy code:

- `print(“Hello World!”)`

Then, download the Log files from Canvas and make sure that you place them in the same directory as your Python script. This will make your job easier as you would not have to specify the full path of your Log files while reading them in your code.

### B. Open log files and assign a variable.

Now open the files and read their contents to a variable like in the snippet given below.

```
logA = open("Log-A.strace", "r")
logB = open("Log-B.strace", "r")

logA_content = logA.readlines()
logB_content = logB.readlines()
```

Figure 1.1 – Opening and Reading log files

The “open” function has two parameters – File Location and Open mode. Here we have given the filename as the location as it is in the same directory as our script, and the open mode “r” is used to state that we are opening the file in **read only mode**.

Then we store the content of each file in 2 variables using the “**readlines**” function.

### C. Parsing the Files

Now we will parse through the file line by line and find the “read” events.

First, we will have to look into the files and find a unique expression that occurs only in the read statements. In this case the unique expression will be “**read()**”.

The code snippet below shows how to parse through the content line by line and search for this expression.

```
readExp = " read("
for line in logA_content:
    if readExp in line:
        print('Read from Log A: ' + line)
```

The code above finds the “read” events in the LogA file and prints them.

1.1 (Question) – What is the total number of “read” events in both of the Logs? Hint: You can do this by incrementing a counter in the for loop.

### Task 2: Analyzing the “read from keyboard” events

#### A. Parsing for “read from keyboard” events

Now, you are supposed to find the “read from keyboard” events. We can find these events by looking for “tty” expression in all the “read” events. You can approach this in multiple possible ways. The code snippet below is an example which will help you find the “read from keyboard” events in the LogA file.

```
readExp = " read("
keyboardExp = "tty"
for line in logA_content:
    if readExp in line and keyboardExp in line:
        print("Read from Keyboard in Log A: " + line)
```

Figure 2.1 – Example of read event showing “tty” expression

2.1 (Question) – What is the total number of “read from keyboard” events in both of the Logs? Hint: You can do this by incrementing a counter in the for loop.

### Task 3: Analyzing the “read from file” events

#### A. Parsing for “read from file” events

In this task we will understand how to select something if we cannot put it into an exact expression. There are only 3 possible input sources for a “read” event – **keyboard, files or pipe**. It is difficult to make an expression to search for filenames. But we can easily find the “read from keyboard” and “read from pipe” events. Then, the remaining events will be “read from file”.

The code snippet below is an example which will help you do the same for LogA file.

```
readExp = " read("
keyboardExp = "tty"
pipeExp = "pipe"
for line in logA_content:
    if readExp in line and keyboardExp not in line and pipeExp not in line:
        print("Read from File in Log A: " + line)
```

Figure 3.1 – Finding “read from file” events for LogA file.

3.1 (Question) – What is the total number of “read from file” events in both of the Logs? Hint: You can do this by incrementing a counter in the for loop.

### Task 4: Challenge

If you were able to complete all the above sections, this challenge will just elaborate on that and help you understand how to present your findings.

#### A. Finding file Locations

In this particular Task you are supposed to find the locations of the files that are present in the “read from file” events.

Every file location is stored between ‘<>’ these braces. This makes it easier for us to look for the file locations in a particular line. The code snippet below is an example on how to do this.

```
readExp = " read("
keyboardExp = "tty"
pipeExp = "pipe"
for line in logA_content:
    if readExp in line and keyboardExp not in line and pipeExp not in line:
        filename_start = line.find('<')
        filename_end = line.find('>')
        filename = line[filename_start + 1:filename_end]
```

Figure 4.1 – Parsing for filenames

4.1 (Question) - Challenge Task 1 find the locations of the files that are present in the “read from file” events.

#### BONUS - Counting Iterations

After you have found the file names (with full locations included with them), you will see that there are repetitions.

So, in this Task you are required to keep track of the number of occurrences of every file and present them in form of a table.

You can do this by creating 2 separate lists – **filenames[]** and **filename\_count[]**. The filenames[i] will have filename\_count[i] occurrences.

A simple algorithm that you can use to accomplish this task is given below:

*For every line*

- *find the filename*
- *check if it already exists in the filenames[] lists*
  - *if it does, just increment the filename\_count[] at that index by 1*
  - *if it doesn't, add that filename to filenames[] and set the filename\_count[] at that index to be 1*

You will have to use functions like:

- **append()** – to append to the lists you have created

- **index()** – to search for a filename in filenames[] list and get its index which you can use to increment the relevant counter in the filename\_count[] list.

**Save the output of your algorithm as a txt file for your lab archive submission.  
Your output should be similar to the one below.**

Log file 1

```
Filename: /lib/x86_64-linux-gnu/libc-2.23.so, Count: 4
Filename: /etc/nsswitch.conf, Count: 4
Filename: /lib/x86_64-linux-gnu/libnss_compat-2.23.so, Count: 2
Filename: /lib/x86_64-linux-gnu/libnsl-2.23.so, Count: 2
Filename: /lib/x86_64-linux-gnu/libnss_nis-2.23.so, Count: 2
Filename: /lib/x86_64-linux-gnu/libnss_files-2.23.so, Count: 2
Filename: /lib/x86_64-linux-gnu/libselinux.so.1, Count: 1
Filename: /lib/x86_64-linux-gnu/libpcre.so.3.13.2, Count: 1
Filename: /lib/x86_64-linux-gnu/libdl-2.23.so, Count: 1
Filename: /lib/x86_64-linux-gnu/libpthread-2.23.so, Count: 1
Filename: /proc/filesystems, Count: 2
Filename: /etc/locale.alias, Count: 2
Filename: /proc/sys/kernel/ngroups_max, Count: 2
```

Log file 2

```
Filename: /lib/x86_64-linux-gnu/libc-2.23.so, Count: 9
Filename: /home/user/test/in.txt, Count: 2
Filename: /lib/x86_64-linux-gnu/libselinux.so.1, Count: 3
Filename: /lib/x86_64-linux-gnu/libpcre.so.3.13.2, Count: 3
Filename: /lib/x86_64-linux-gnu/libdl-2.23.so, Count: 3
Filename: /lib/x86_64-linux-gnu/libpthread-2.23.so, Count: 3
Filename: /proc/filesystems, Count: 6
```

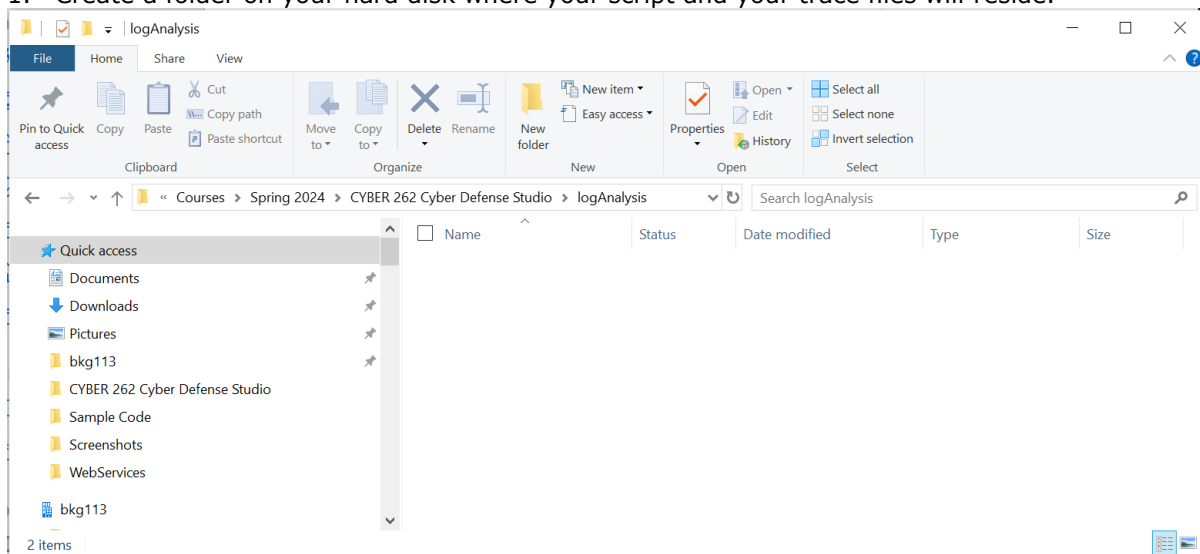
**BONUS (Question)** - In this BONUS Task you are required to keep track of the number of occurrences of every file and present them in form of a table. The format of your output should be similar to the above example.

## Appendix C – Log Analysis Lab Setup Guide (Windows)

The following guide describes the preliminary steps needed to setup a workstation to process a basic set of log files using a Python script running on a Windows computer. This activity is used as the lead-in for a detailed lab analysis exercise completed in CYBER 262 Cyber-Defense Studio, a course normally taken by Penn State’s second-year cybersecurity degree pre-majors. Students will require access to a working Python configuration either through a local installation on their personal machine, a lab machine setup by faculty or your local IT department or running inside of a virtual machine. These steps should be completed to verify you have a working Python environment before you proceed with the rest of the log analysis steps. This exercise supplies the log files via our Canvas learning management system, however, you may place a shared copy of the files in any secure location that is convenient for the students to access.

### Downloading the files

1. Create a folder on your hard disk where your script and your trace files will reside.

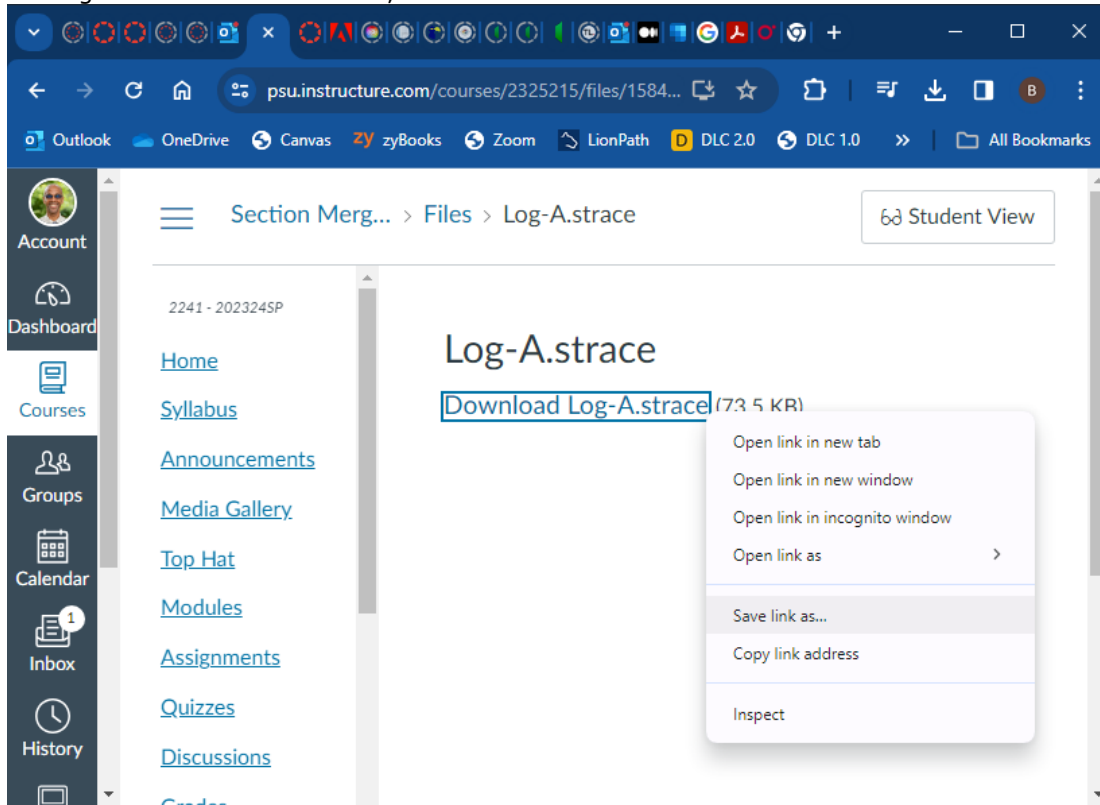


2. Go to the Lesson 10 module on the Modules page, then click on the **Log-A.strace** link.

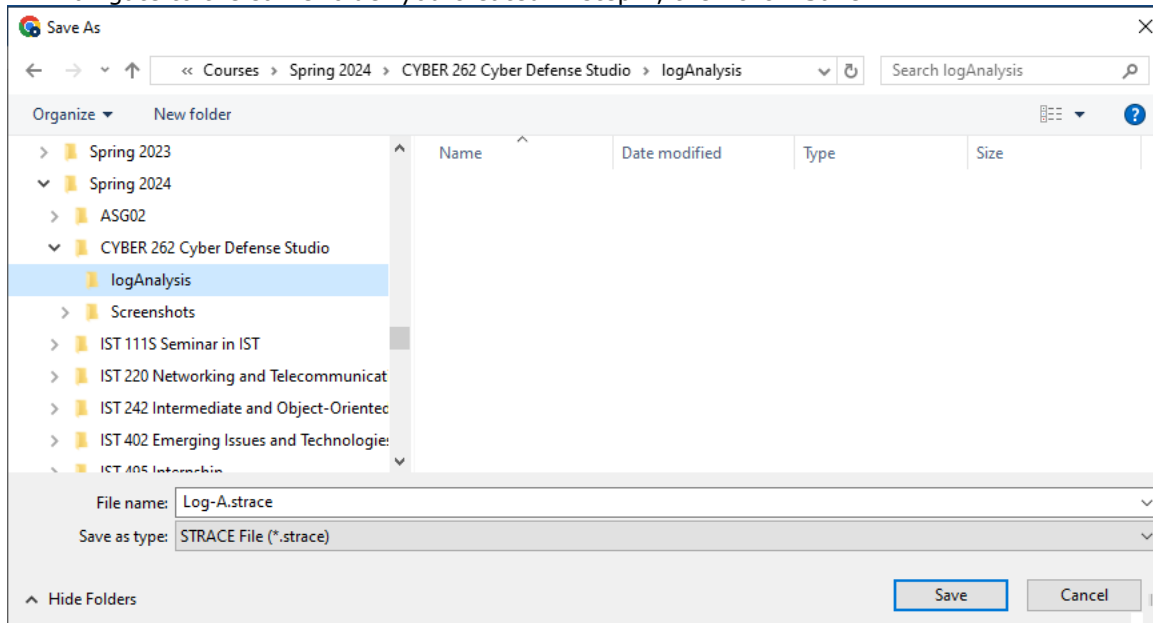




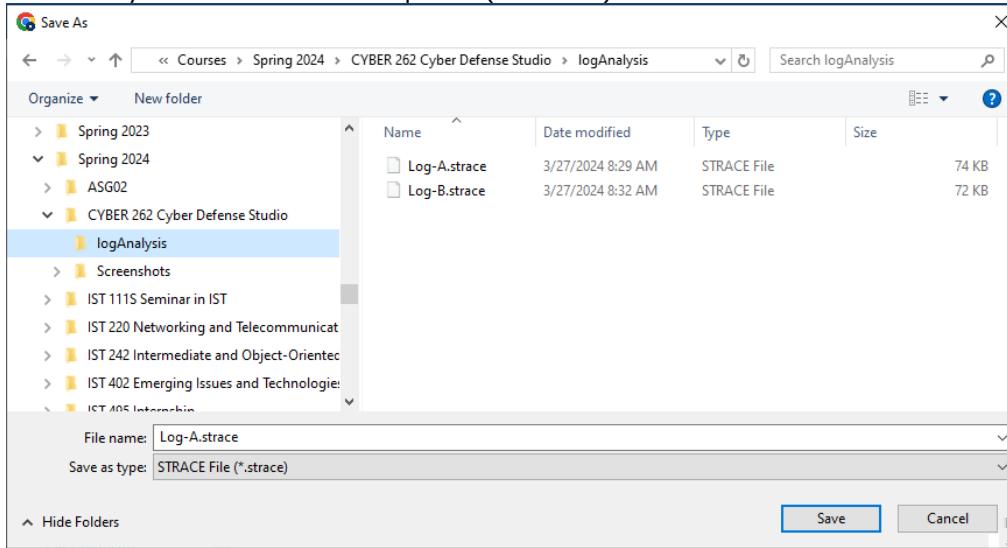
3. Right-click on the file name, then left-click on 'Save link as...'.



4. Navigate to the same folder you created in step 1, then click 'Save'.

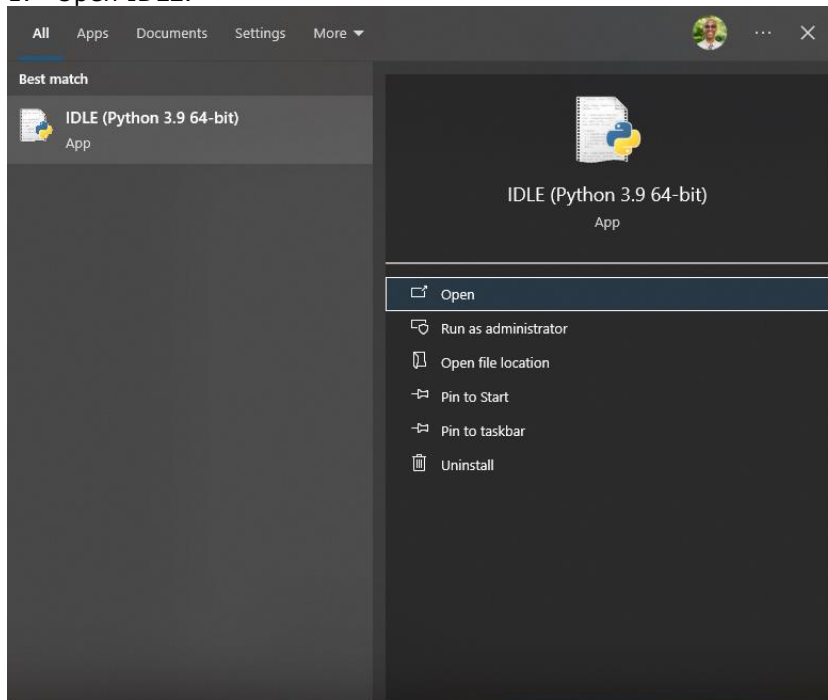


- Return to the Module 10 section of the Modules page in Canvas and repeat this process for the second file. When you have downloaded both files, verify they exist by navigating to the folder where they were saved in File Explorer (Windows).

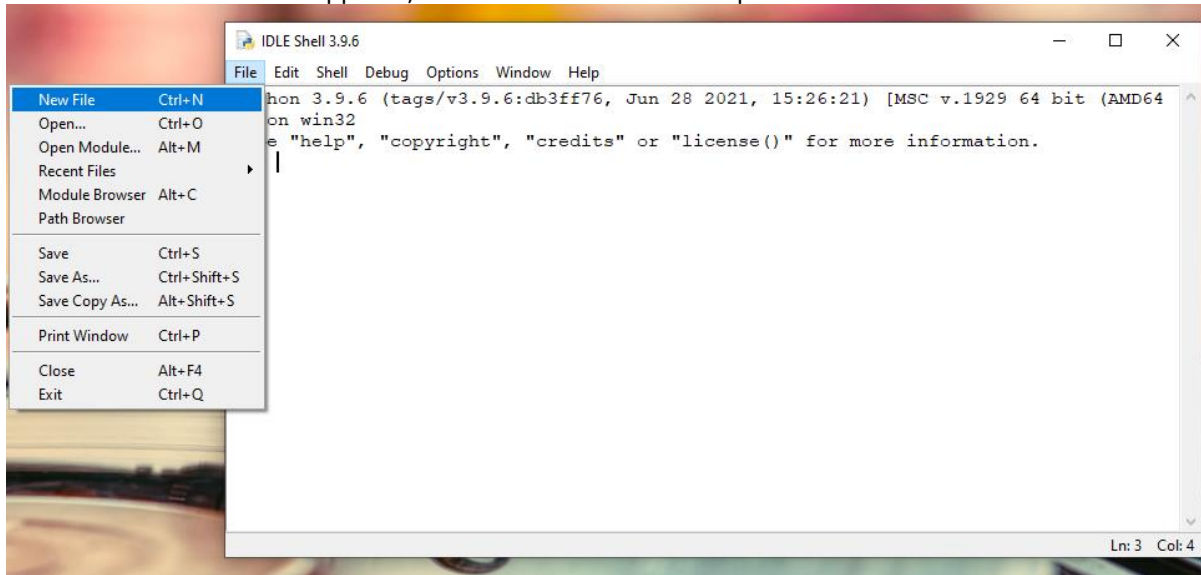


## Python programming setup

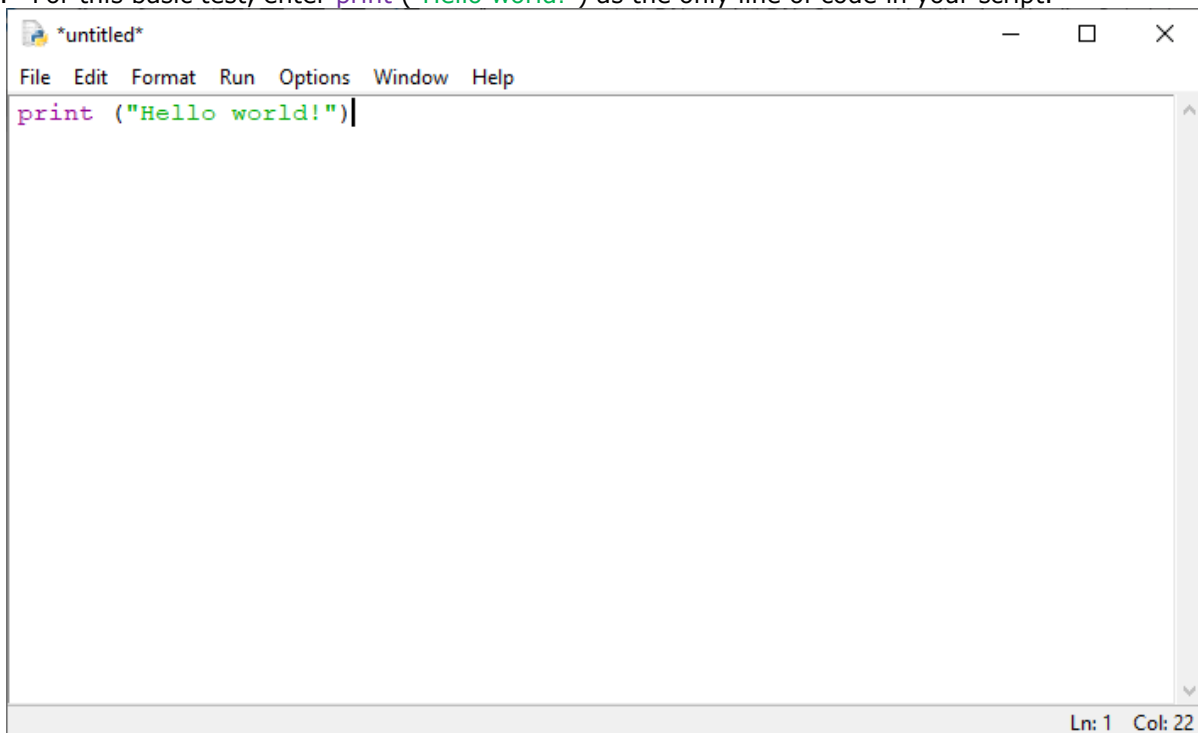
- Open IDLE.



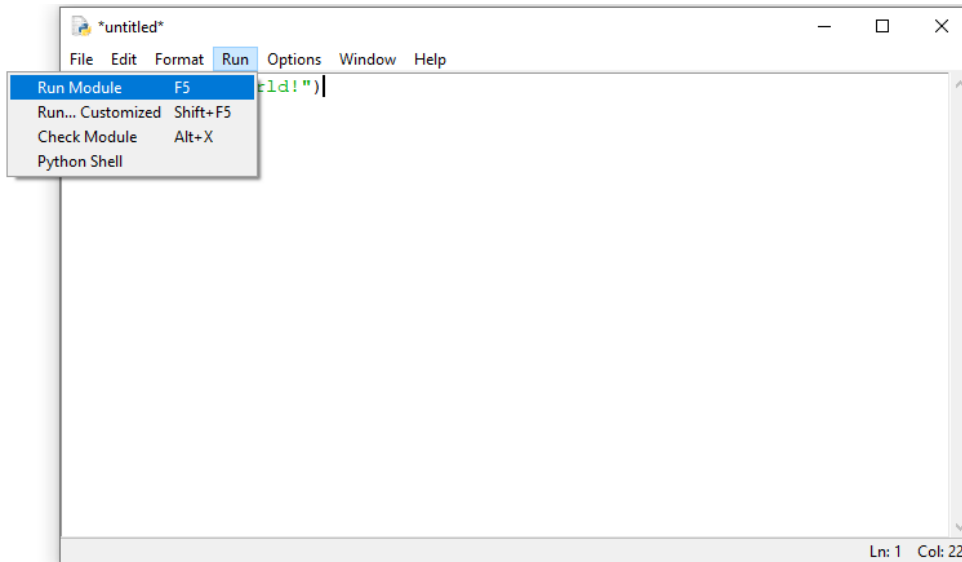
2. When the IDLE Shell appears, click `File → New File` to open the IDLE editor



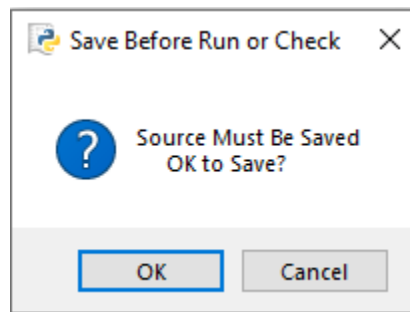
3. For this basic test, enter `print ("Hello world!")` as the only line of code in your script.



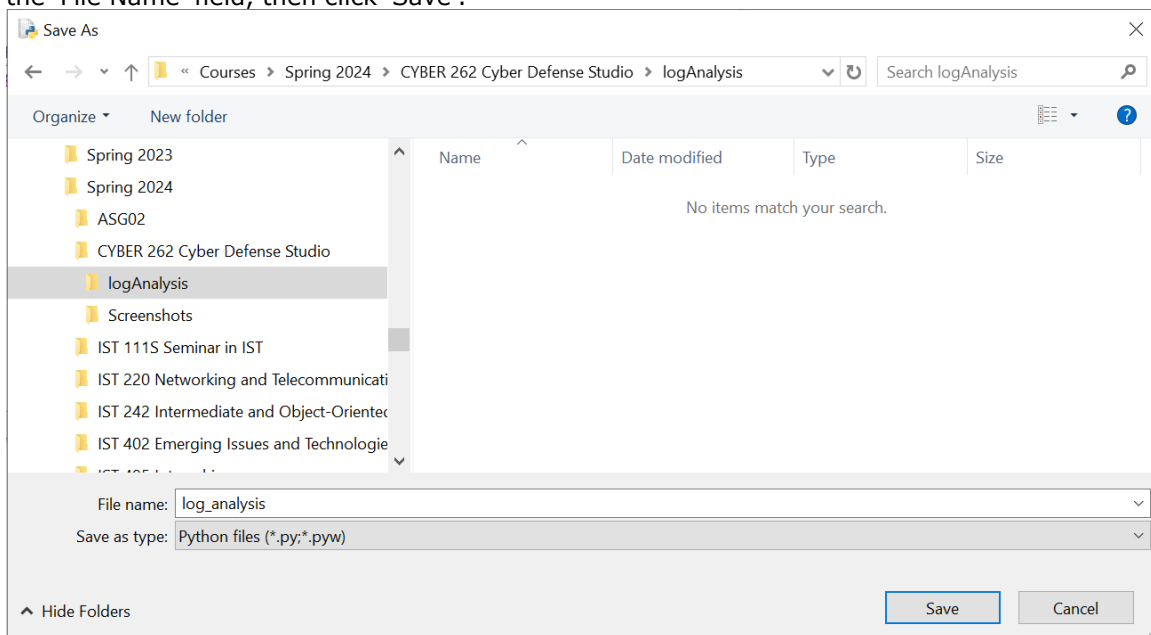
4. Click 'Run → Run module...'



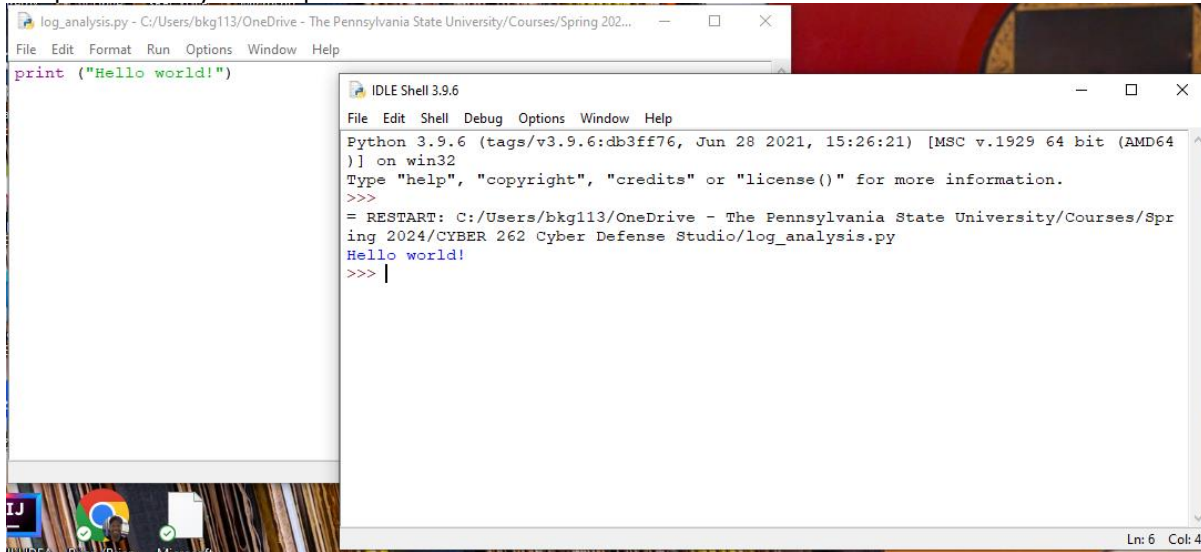
5. Click 'OK' to proceed.



6. Navigate to the same folder where your trace files are located, enter the file name of your script in the 'File Name' field, then click 'Save'.



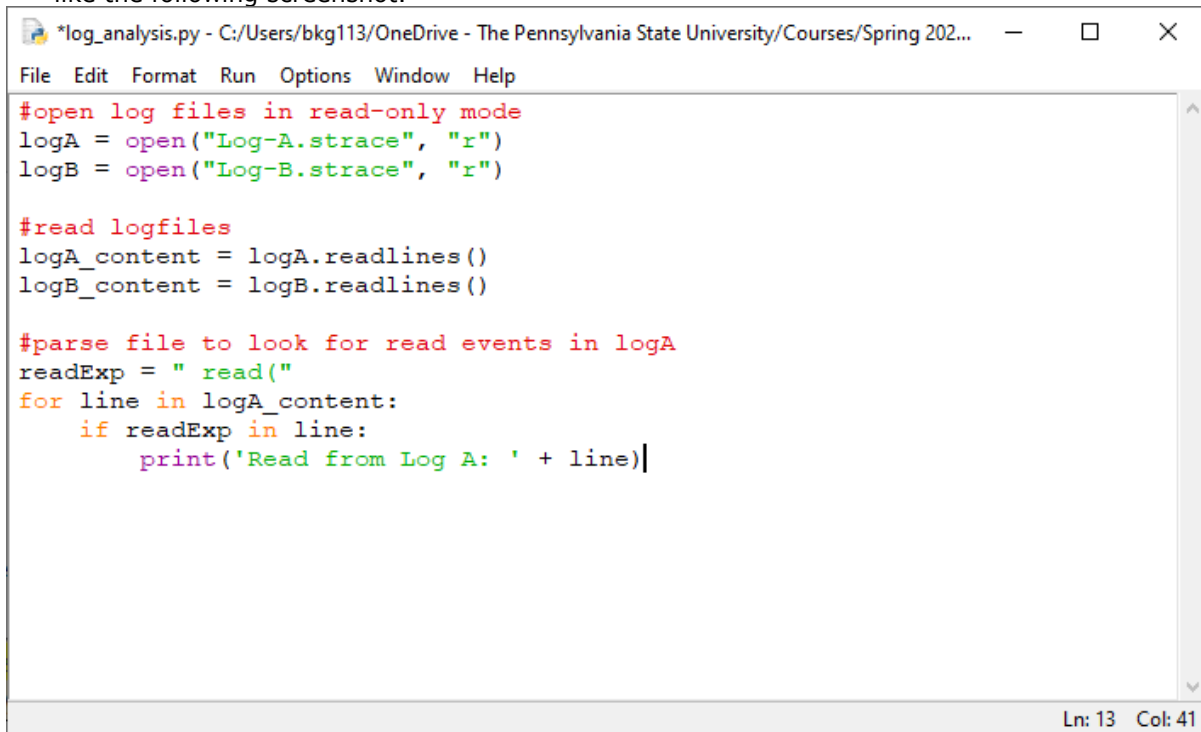
7. If the string 'Hello world!' appears in the IDLE Shell, then your script is starting to take shape. Also note that the file path in the IDLE shell just above the output should reflect the file name and the path where your script and trace files are stored.



```
log_analysis.py - C:/Users/bkg113/OneDrive - The Pennsylvania State University/Courses/Spring 2024...
File Edit Format Run Options Window Help
print ("Hello world!")

IDLE Shell 3.9.6
File Edit Shell Debug Options Window Help
Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/bkg113/OneDrive - The Pennsylvania State University/Courses/Spring 2024/CYBER 262 Cyber Defense Studio/log_analysis.py
Hello world!
>>> |
```

8. Switch back to the IDLE editor and enter the first part of the code in the assignment that should look for 'read' events in LogA by parsing each record for the string ' read('. Your code should look like the following screenshot.

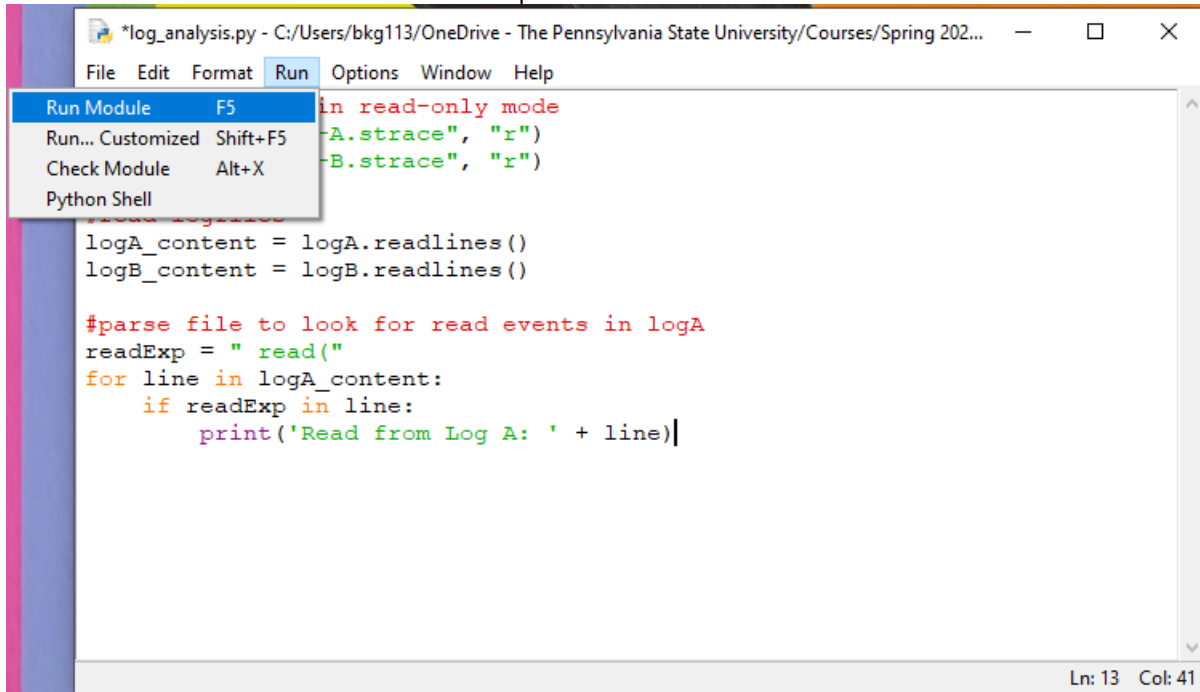


```
*log_analysis.py - C:/Users/bkg113/OneDrive - The Pennsylvania State University/Courses/Spring 2024...
File Edit Format Run Options Window Help
#open log files in read-only mode
logA = open("Log-A.strace", "r")
logB = open("Log-B.strace", "r")

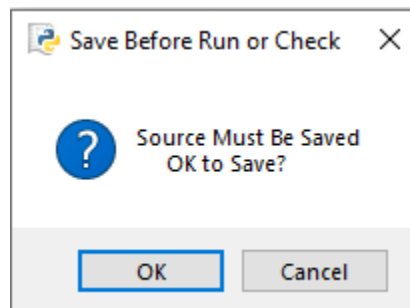
#read logfiles
logA_content = logA.readlines()
logB_content = logB.readlines()

#parse file to look for read events in logA
readExp = " read("
for line in logA_content:
    if readExp in line:
        print('Read from Log A: ' + line)|
```

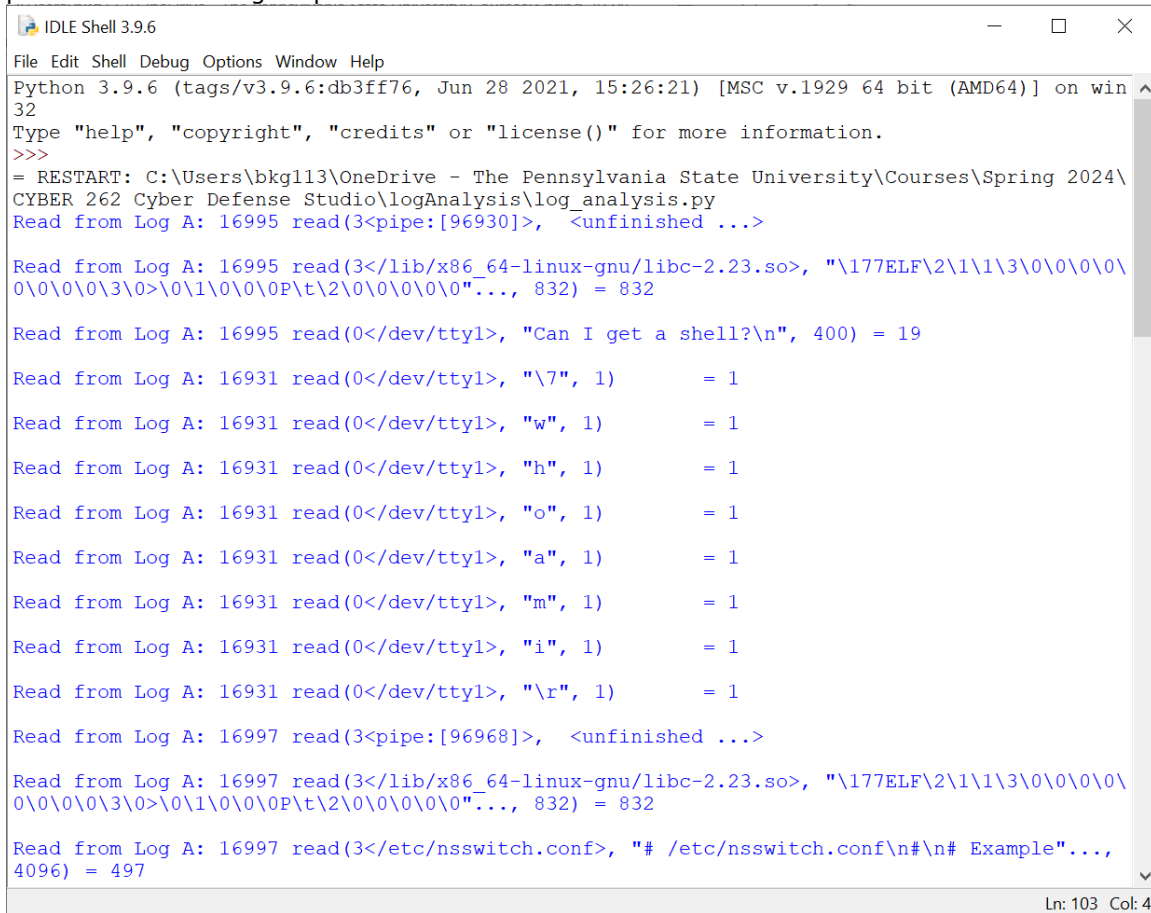
9. Click 'Run → Run Module' to run the script.



10. Click 'OK' to proceed.



11. Your program should find the LogA trace file in the same folder where your script is running and produce the following output in the IDLE Shell.



```
Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\bkg113\OneDrive - The Pennsylvania State University\Courses\Spring 2024\CYBER 262 Cyber Defense Studio\logAnalysis\log_analysis.py
Read from Log A: 16995 read(3<pipe:[96930]>, <unfinished ...>

Read from Log A: 16995 read(3</lib/x86_64-linux-gnu/libc-2.23.so>, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\t\2\0\0\0\0"... , 832) = 832

Read from Log A: 16995 read(0</dev/tty1>, "Can I get a shell?\n", 400) = 19

Read from Log A: 16931 read(0</dev/tty1>, "\7", 1) = 1
Read from Log A: 16931 read(0</dev/tty1>, "w", 1) = 1
Read from Log A: 16931 read(0</dev/tty1>, "h", 1) = 1
Read from Log A: 16931 read(0</dev/tty1>, "o", 1) = 1
Read from Log A: 16931 read(0</dev/tty1>, "a", 1) = 1
Read from Log A: 16931 read(0</dev/tty1>, "m", 1) = 1
Read from Log A: 16931 read(0</dev/tty1>, "i", 1) = 1
Read from Log A: 16931 read(0</dev/tty1>, "\r", 1) = 1

Read from Log A: 16997 read(3<pipe:[96968]>, <unfinished ...>

Read from Log A: 16997 read(3</lib/x86_64-linux-gnu/libc-2.23.so>, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\t\2\0\0\0\0"... , 832) = 832

Read from Log A: 16997 read(3</etc/nsswitch.conf>, "# /etc/nsswitch.conf\n#\n# Example"... , 4096) = 497
```

At this point, you can proceed with the remainder of the lab knowing that your Python program can access the data sources provided in the assignment.