

*Teaching Tip*  
**Decryption and Reverse Engineering - An Applied Tutorial  
for a Security Programming Course**

Gunjan Batra, Humayun Zafar, and Pamila Dembla

**Recommended Citation:** Batra, G., Zafar, H., & Dembla, P. (2025). Teaching Tip: Decryption and Reverse Engineering - An Applied Tutorial for a Security Programming Course. *Journal of Information Systems Education*, 36(2), 111-129. <https://doi.org/10.62273/STWE8998>

**Article Link:** <https://jise.org/Volume36/n2/JISE2025v36n2pp111-129.html>

Received:	September 12, 2024
First Decision:	November 4, 2024
Accepted:	January 6, 2025
Published:	June 15, 2025

Find archived papers, submission instructions, terms of use, and much more at the JISE website:  
<https://jise.org>

ISSN: 2574-3872 (Online) 1055-3096 (Print)

---

# **Teaching Tip**

## **Decryption and Reverse Engineering - An Applied Tutorial for a Security Programming Course**

**Gunjan Batra**

**Humayun Zafar**

**Pamila Dembla**

Coles College of Business

Kennesaw State University

Kennesaw, GA 30144, USA

[gbatra@kennesaw.edu](mailto:gbatra@kennesaw.edu), [hzafar@kennesaw.edu](mailto:hzafar@kennesaw.edu), [pdembla@kennesaw.edu](mailto:pdembla@kennesaw.edu)

### **ABSTRACT**

For professionals in the cyber security industry, knowledge of programming is a highly valuable skill. Therefore, it is crucial to teach programming courses to cybersecurity students in a way that aligns with industry needs. This paper presents a tutorial on encryption for a security programming course for undergraduate students specializing in cybersecurity. The challenge requires students to reverse engineer an encryption algorithm (using Python) and decrypt a text using the acquired knowledge. The tutorial also discusses the importance of programming skills for security students and why Python is an excellent choice for this purpose. The tutorial was conducted at a business school, receiving positive feedback from students and sparking increased interest in programming applications within cybersecurity.

**Keywords:** Tutorial, Security programming, Python, Encryption, Reverse engineering

### **1. INTRODUCTION**

Cybercrime poses a growing threat to global businesses, with the “Cost of a Data Breach 2022” report highlighting an average cost of \$4.35 million per breach (IBM, 2022). Concurrently, Cybersecurity Ventures forecasts that cybercrime costs will escalate to \$10.5 trillion annually by 2025, driven by the expansion of digital infrastructure and the increasing sophistication of cyberattacks (Morgan, 2021).

To address this challenge, comprehensive initiatives from academia, industry, and government aim to cultivate a skilled cybersecurity workforce. Notably, the National Security Agency’s (NSA) endorsement of 312 colleges as Centers of Academic Excellence highlights the increasing emphasis on advanced cybersecurity education, incorporating rigorous training in network protocol analysis, programming, and reverse engineering (Reeder & Paller, 2021).

Amid these developments, the intersection of programming skills and cybersecurity expertise has become pivotal. Understanding software development and security programming is crucial for professionals tasked with defending against and mitigating cyber threats. To address this need, our tutorial on “Cryptography” within a “Security Programming” course offers practical, hands-on training. Utilizing Python, students engage in reverse encryption exercises that enhance their capability to decrypt messages and apply security concepts effectively.

This tutorial, tested in the IS/ISA (Information Systems/Information Security and Assurance) program at a southeastern

U.S. business school, has demonstrated significant educational benefits. Feedback from a post-tutorial survey indicated that students not only appreciated the practical application of their programming skills but were also keen to explore further programming applications in cybersecurity.

This case study serves as an exemplar for integrating real-world cybersecurity challenges into academic curricula, ensuring that graduates are well-equipped to meet the demands of the cybersecurity sector.

The paper is organized as follows. Section 2 discusses related work, Section 3 explains the rationale for choosing this tutorial, Section 4 presents the tutorial, and Section 5 discusses evidence of student learning. Sections 6 and 7 cover the applications, benefits, and conclusions of our work. Appendix A presents detailed solution steps for students, including snapshots, and Appendix B presents the survey questions.

### **2. RELATED WORK**

In this section, we discuss the importance of security students knowing a programming language, the effectiveness of tutorial-based instruction for teaching programming, the reasons for choosing Python for our tutorial, prior work on using Python in security programming courses, and the NIST Cybersecurity Framework (NIST CSF 2.0, 2024).

## **2.1 Importance of Security Students Knowing a Programming Knowledge**

The objective of teaching programming languages to security students is to meet the skill, demand, and business needs in the industry. Most entry-level and intermediate-level roles in the information security field, such as cybersecurity manager, information security analyst, network engineer, malware analyst, threat intelligence expert, and network security architect, require and benefit from some level of programming ability. Security professionals sometimes need to manage and interact with professionals who are developers or programmers. Because many companies seek this skill, universities are offering new courses, certificates, and degree programs with programming components.

Most employers prefer graduates of security programs to have hands-on experience to reduce the need for (and expense of) on-the-job-training (Lewis & Crumpler, 2019). Employers and universities use cybersecurity assessment (cyber aptitude) tests that include programming testing (computers, programming, and security) (Reeder & Paller, 2021). During job interviews, employers may have a technical expert ask candidates to explain the vector used in a recent attack or to solve a few reverse engineering, network traffic protocol analysis, or Python programming problems.

Yang and Wen (2017) collected empirical data of U.S. business school (B-school) cybersecurity programs, developed a cybersecurity curriculum model, and identified programming as the second most important course in the curriculum. They recommended that cybersecurity professionals learn at least one object-oriented programming language. Programming knowledge gives them an edge over security professionals without those skills (Kuk et al., 2019).

## **2.2 Tutorial-Based Instruction for Teaching Security Programming**

For an IS/ISA graduate, the objective is not simply to learn to code but to learn how to use code to solve real-life business problems. The goal is to move away from lecturing and utilize a more application-based learning-based approach.

Multiple approaches are used to teach information security, such as textbooks, research, lectures, tutorials, and labs. The idea behind using a tutorial to add to the lectures on security programming is to provide a balanced understanding of the security problem from a business perspective and how to use a programming language to solve it. Students should not only practice commands and functions but also understand the contextual information such as when and why to use them in a real-world security problem. An extensive literature review to assess the methods to teach programming was performed.

Various approaches have been used by different programs: context-based (Kakucs et al., 2022; Wen & Katt, 2019), example-based (Gaber & Kirsh, 2021; Moumoutzis et al., 2018; Segal & Ahmad, 1993), lecture-based (Delialioğlu, 2012), game-based (Combéfis et al., 2016; Lindberg et al., 2019; Mathrani et al., 2016), problem-based (Bawamohiddin & Razali, 2017; Chutisowan et al., 2021; Delialioğlu, 2012; Nuutila et al., 2008; Omeh et al., 2022; Peng, 2010), project-based (Phuong, 2022; Sherman et al., 2019), case-based learning (Chutisowan et al., 2021; Moumoutzis et al., 2018), or a combination of these.

Teaching programming to security students requires a hands-on, application-based approach (Ksiezopolski et al.,

2022). This should be done using real-world examples and problems. Contextualized learning (Rivet & Krajcik, 2008), which often takes the form of real-world examples and problems, is a meaningful method for students to learn the applications of the code they learn in the programming class. Contextualization provides a powerful motivation for learning (Cooper & Cunningham, 2010; Perin, 2011). Security and programming knowledge should be contextualized together and embedded in a meaningful scenario that makes sense to students, enhances their understanding, and makes the concepts more relatable.

Psychology and education researchers have demonstrated that when knowledge is learned in a context similar to that in which the skills will actually be needed, applying the learning to the new context may be more likely (Dey, 2001; Perin, 2011). Learning knowledge content through real-world experience is important for students because “Once they can see the real-world relevance of what they’re learning, they become more interested and motivated” (Predmore, 2005, p. 23). In addition, studies in educational psychology have affirmed that example-based learning is quite effective, especially for students or novice problem solvers (Atkinson et al., 2000). Practical scenarios are great tools to simulate a workplace environment (Hamburg et al., 2008). Merrill’s (2002) First Principles of Instruction (FPI) for effective teaching and learning emphasize that meaningful learning occurs when:

- 1) Learning is problem-centered, i.e., learners are engaged in solving real-world problems
- 2) Existing knowledge is activated as a foundation for new knowledge
- 3) New knowledge is demonstrated to the learner
- 4) Learners apply new knowledge
- 5) Knowledge is integrated into the learner’s world

The tutorial presented in this paper is based on the problem-centered approach. The principles align with the tutorial’s approach, which requires the students to reverse engineer an encryption algorithm and decrypt a text. We chose a problem-based approach for the tutorial to give students exposure to practical and industry-relevant problems.

## **2.3 Why Are We Using Python?**

Python is one of the best programming languages for security experts (Bravo, 2023; CPOMagazine, 2020; NSA, CISA, & US, 2023). Python is easy and quick to learn, understand, implement, debug, and troubleshoot. Python is a general-purpose, server-side scripting language that can be implemented easily in security projects. The vast library of assets and a large community of developers make Python a powerful open-source language. A fundamental understanding of Python’s data structures can be applied to penetration testing and cybersecurity applications and can be very beneficial for those who want to advance in this area (Kuk et al., 2019). Considering the popularity, use, applicability, and demand for Python in academia and industry over the past few years, it is a great choice for teaching security programming.

Python is the number one programming language, according to the TIOBE Software Index, an indicator of the popularity of programming languages (TIOBE Index for April 2025, 2025). Python is also number one on the PYPL Popularity of Programming Language Index, which is created based on how often language tutorials are searched on Google (PYPL

Popularity of Programming Language, 2025). This index is indicative of which language might be valuable to study, or consider for use in a new software project.

Multiple works compare Python with other object-oriented programming languages and discuss why it is better choice of language to learn for programming students (Bogdanichikov et al., 2013; Fangohr, 2004; Kuk et al., 2019). Further, the suitability of Python as a programming language for B-school IS curriculum has been discussed in detail (Ellis et al., 2019). Python gives B-school students academic and business-relevant programming experience.

From a cyber security perspective, Python programming skills will be of great value to students who want to build foundational understanding of technology with cybersecurity proficiency (Odetokun, 2020). Most importantly, Python is in high demand by employers. In the 2023 Robert Half Technology Salary Guide, Python is listed as one of the top five coding languages tech leaders are looking for (Johnson, 2022). Python has been widely adopted across various industries. It is being used at Google for mainframe foundation, Dropbox for cloud-based services, YouTube for integration of streaming videos into their Internet pages, and Instagram for its Django framework. Python is also used by NASA for Workflow Automation System (WAS) and scientific programming tasks, and NSA for cryptography and intelligence analysis. Python can perform a multitude of security functions, including writing scripts, automating information security processes, customizing tools, analyzing malware, scanning, penetration testing tasks, and automating security response operations. Most of these activities can be automated using Python scripts and easily coded in an English-like script of Python (Kuk et al., 2019). Python has numerous libraries for implementing a variety of functions such as Scikit, Nmap, Twisted, Scapy, BeautifulSoup, Cryptography, YARA, Pymetasploit3, Mechanize, Socket, and pygeoiip.

#### **2.4 Prior Work on Using Python for Security Programming Courses**

Python has been used for teaching information security. For instance, an approach to teach cybersecurity that has been implemented and successfully executed at Business and Technology University (BTU), Georgia was discussed by Chokhonelidze et al. (2020). They implemented a first-semester compulsory course "Introduction to Programming With Python Language," followed by a second-semester programming course, and multiple cybersecurity courses in subsequent semesters. Next, they mention that the approach has shown that students master cybersecurity subjects and are well prepared for the job market. Additionally, data shows students' interest in a security-oriented Python course and their readiness to participate in the class upon its introduction, as discussed by Odetokun (2020). They recommend creating effective learning environments (labs, tools) to enhance student engagement. A study conducted by Henttonen and Rathod (2024) highlights the importance of programming in cybersecurity based on a 15-week Python course for students seeking cybersecurity careers. The study found positive impacts, such as enhanced engagement, when cybersecurity-specific content was integrated into a programming course.

There are examples of positive experiences conducting camps teaching cybersecurity and Python programming to high school students, college freshman, students with no prior

programming experience. CyberPDX, a residential summer camp was conducted by Feng et al. (2017) to introduce cybersecurity to high school students. The camp focused on a range of societal issues impacted by cybersecurity through four learning threads, including Python programming. Initial results showed that the camp positively influenced participants by increasing their intention to pursue a career in cybersecurity. Another summer camp for high school students, including entering college freshman was conducted by Eckroth (2018), who shared his experiences teaching a 5-day 25-hour cybersecurity and Python programming. Although the curriculum was designed for students with no prior experiences in cybersecurity or programming, it covered a wide variety of topics including networking, encryption, password management, and penetration testing. A post-survey indicated that some students expressed disappointment with the rushed curriculum, as it attempted to cover many foundational topics and exercises within a limited time frame. The work by Breese and Gardner (2024) presents two teaching cases based on Python programming language for K-12 and university level cybersecurity learners. The first exercise introduces students to basic Python concepts and guides them through developing a simple number guessing game. The second exercise focuses on conducting log file analysis, integrating programming skills with cybersecurity education. These exercises have been implemented in both a cyber camp environment and in-class settings for 100- and 200-level courses, primarily to attract novice students to pursue cybersecurity as a major. They plan to publish student experiences as part of their future work.

In our work, our goal is to provide the learners with a practical, relevant to real world cybersecurity work and skill building exercise that demonstrates the applications of encryption in security. One of the biggest challenges is ensuring access to the right materials, tools, and techniques to practice the concepts and theories covered in course texts. We want to provide resources that make students self-sufficient to solve a given problem as many of them could be a novice in programming and cybersecurity. The tutorial with step-by-step screens will serve as an excellent resource to solve the given problem. Students are also provided sufficient time to complete it. Finally, we want to spark their interest in learning more applications of cybersecurity in programming.

#### **2.5 NIST Cybersecurity Framework (NIST CSF)**

The NIST Cybersecurity Framework (NIST CSF) is a comprehensive set of guidelines and best practices designed to help organizations manage and reduce cybersecurity risks. Developed by the National Institute of Standards and Technology (NIST), the framework provides a flexible, risk-based approach to cybersecurity, suitable for organizations of all sizes and sectors. The NIST CSF is adopted across various industries, making it highly relevant for students who will enter the cybersecurity workforce. The six core functions are as follows:

- 1) The GOVERN (GV) Function involves establishing, communicating, and monitoring an organization's cybersecurity risk management strategy and policies, aligning them with its mission and stakeholder expectations. This function integrates cybersecurity into the broader enterprise risk management (ERM) strategy and oversees roles, responsibilities, and the cybersecurity strategy.

- 2) The IDENTIFY (ID) Function involves understanding the organization's current cybersecurity risks, assets, and suppliers to prioritize efforts consistent with its risk management strategy and mission needs. This function identifies opportunities for improving policies, plans, processes, and practices that support cybersecurity risk management across all functions.
- 3) The PROTECT (PR) Function involves implementing safeguards to manage and secure the organization's assets, reducing the likelihood and impact of adverse cybersecurity events. Key areas include identity management, authentication, access control, awareness and training, data security, platform security (i.e., securing the hardware, software, and services of physical and virtual platforms), and technology infrastructure resilience.
- 4) The DETECT (DE) Function involves identifying and analyzing potential cybersecurity attacks and compromises. This function enables timely discovery of anomalies and indicators of compromise, supporting effective incident response and recovery efforts.
- 5) The RESPOND (RS) Function involves taking action on detected cybersecurity incidents, focusing on containing their effects. This function includes incident management, analysis, mitigation, reporting, and communication.
- 6) The RECOVER (RC) Function involves restoring assets and operations affected by a cybersecurity incident, aiming for timely recovery to minimize impact. This function also ensures effective communication during recovery efforts.

### **3. RATIONALE BEHIND SELECTION OF THIS TUTORIAL**

Understanding how encryption algorithms work and their weaknesses is very important in cybersecurity. The tutorial follows a step-by-step teaching and explanation approach to make it an engaging exercise for students. Based on a reverse engineering scenario, the tutorial can be self-learned by students who have some prerequisite knowledge and would like to gain hands-on experience in this area. There are multiple reasons due to which this tutorial was chosen for the course and as a subject in this paper.

- 1) Alignment with NIST CSF 2.0: This tutorial aims to leverage the principles of NIST Cyber Security Framework (NIST CSF 2.0, 2024) to enhance participants' understanding of cybersecurity threats and defense mechanisms. Focusing specifically on the PROTECT Function of the NIST CSF framework, it teaches students to implement safeguards and enhance encryption methods to secure data against unauthorized access. By reverse engineering, participants can identify weaknesses in encryption methods and learn how to strengthen them, contributing to the organization's overall protective measures.
- 2) Practical relevance of the tutorial topic: This tutorial equips students with knowledge and skills that are valued across multiple sectors. Students learn a topic on information security and gain understanding of how to manually reverse a script. Cryptographic algorithms play an important role in the security industry to protect

sensitive information. However, there are scenarios where reverse engineering a cryptographic algorithm is necessary, such as understanding its technical workings, assessing its security, or developing compatible software. Other applications of reverse engineering include searching for vulnerabilities in software, hardware, or systems, and analyzing malware. By understanding how a system works, security professionals can identify weaknesses that could allow attackers to gain access or cause damage.

- 3) Hands-on experience: The tutorial provides hands-on experience with the theoretical concepts learned in the course. The problem-based approach makes the exercise interesting and engaging. This approach expands the knowledge horizon of students and encourages them to use their skills to solve real business problems. In addition, this exercise gives students a taste of the types of questions asked during job interviews. Finally, the exercise adds to the students' practical skill which the student can talk about during a job interview. Our objective is to show how we can make security programming course in B-school a fun, challenging, and practical learning experience for students by including exercises such as the below tutorial.

### **4. TUTORIAL – DECRYPTION AND REVERSE ENGINEERING CHALLENGE**

In this section, we will describe the tutorial. The tutorial will be provided as part of the assignments or as a bonus exercise to the students in the "Cryptography" module of the security programming course. The instructor should ensure that the prerequisite course material is covered before providing the tutorial to the students. The course can be taught online/in-person and the material will be provided through the Learning Management System (e.g., D2L Brightspace) as part of the lecture content.

#### **4.1 Part of Tutorial for the Instructors**

Level: Undergraduate. Pre-requisites: To instruct the students in the Reverse Engineering Challenge, faculty members should ensure the students know the following concepts:

- 1) Basics of cryptography (encryption, decryption, hashing)
- 2) NIST CSF 2.0
- 3) Basic programming concepts of Python (variables, operators, decision structure, loops, functions, strings)
- 4) Basics of reverse scripting

The tutorial provides a hypothetical scenario to the students in which a job applicant interviewing for a position in cybersecurity is asked to reverse engineer an encryption algorithm written in Python. The goal is to demonstrate how to manually analyze the code of an encryption algorithm to decrypt a message and create its hash. The tutorial is self-explanatory and students can follow the step-by-step instructions provided. Each instruction is accompanied by visuals that illustrate the code, clarify what needs to be done, explain the reasoning behind it, and demonstrate how to implement it. In the tutorial exercise,

- 1) Students need to decrypt a string and find the hash of the string.



- 2) Students are provided with two files to start with: encryption code in “encryptor.py” and encrypted message in “message.txt.”
- 3) Students are given one week to complete the tutorial and are required to submit a snapshot of the hash as a proof of completion.
- 4) Student handouts include:
  - a. Learning objectives, prerequisite knowledge, scenario, challenge objectives, tools required.
  - b. Steps to solve the problem.
  - c. Reference snapshots of the solution (in Appendix A).

## 4.2 Part of Tutorial for the Students

### 4.2.1 Learning Objectives.

- 1) Python Programming Proficiency:
  - a. Apply basic Python programming concepts, including variables, operators, decision structures, loops, functions, and strings.
  - b. Enhance problem-solving skills using Python in a practical cybersecurity scenario.
- 2) Reverse Engineering Techniques:
  - a. Analyze and understand the structure and flow of an encryption algorithm.
  - b. Develop Practical Skills in reverse engineering a Python script.

### 4.2.2 Prerequisite Knowledge.

- 1) Basics of cryptography (encryption, decryption, hashing).
- 2) NIST CSF 2.0.
- 3) Basic programming concepts of Python (variables, operators, decision structure, loops, functions, strings).
- 4) Basics of reverse scripting.

**4.2.3 Scenario.** Assume you are a job applicant and while searching for a job, you find that the Acipher website’s career page is recruiting for a reverse engineering position. After you apply, an Acipher careers system bot sends you an email with a challenge to decrypt a message. The email contains an encrypted message (message.txt: 703e966d1ed86f08daf503d6678e99eb09d47f18), an encryptor script written in Python (encryptor.py) and Challenge Objectives.

### 4.2.4 Challenge Objectives.

- 1) Read the script and understand its flow.
- 2) Decrypt the encrypted message provided to you.
- 3) Find the hash of the message string and submit a snapshot of hash as a proof of completion of the tutorial.

### 4.2.5 Tools Required.

 Python3 and Theia IDE.

### 4.2.6 Solution Steps.

- 1) Part I) Understand the encryption of the message:
  - a. Simply execute the code to see what it does?
  - b. Attempt to understand the variables, operators and functions.
  - c. Give meaningful names to variables and functions based on their role in the program. For example,

the variable “value” that takes in user input can be named – “user input.”

- d. Focus on variables - value, ORD\_value, ORD\_key, lol, xor\_result, movebit, rotateLeft, chars.
- e. Focus on functions - GGGotate().

### 2) Explanation - (encryptor script in encryptor.py):

- a. The program performs encryption on a user-provided input string by manipulating its characters through bitwise operations and rotations. The program code starts by calculating XOR of first character’s ASCII value with the encryption key (the ASCII value of “a”) (Lines 22-24). Then, using a function, it applies left rotation to the result of XOR operation, ensures that the result stays within 8 bits and sets the least significant bit if the original value was above 127. The result is stored as an element in a list (Lines 1-6, 25). Now, for the remaining characters in the input string, it performs an XOR between the last element in the list and the ASCII value of the current character, and again rotates it left using the same function as before (Lines 26-33). After completing the process for all input characters, the values in the list are then converted to a hexadecimal format, resulting in the encrypted string (Lines 40-49).

### 3) Part II) Decryption of the message:

- a. Hexadecimal to decimal conversion: Create a list of left-rotated decimals by converting each two digits of the encrypted message to their decimal format. Each pair of hexadecimal digits in the string represents one byte (8 bits).
- b. Reverse the rotation: Create xor\_result by converting each two digits of the encrypted message to their binary bit format and rotate it right.
- c. Reverse XOR with previous values: Convert “xor\_result” and “encrypted\_key[k]” decimal numbers to their binary numbers. Next, compare which bits of “encrypted\_key[k]” differ from “input\_chars[k]” bits - get final decimal value of each of the character.
- d. Convert decimals numbers to characters: Convert the list of decimal numbers to characters using the ASCII value of the decimal number.
- e. Continue Steps 1-5 for all digit pairs, until the entire message is decrypted and original message is obtained.

### 4) Part III) Compute the flag (i.e., the hash of decrypted message):

- a. Hash the decrypted message using MD5 algorithm or Linux command
- b. Reference snapshots of the solution available in Appendix A.

## 5. STUDENT FEEDBACK

The tutorial and a short survey were provided to the students of a security programming course, where Python was the primary language taught. The students were enrolled in the IS/ISA program in the Department of Information Systems and Security at a business school within a large university in the southeastern United States. The tutorial and the subsequent survey were completed by 18 students. The tutorial was provided after the theoretical concepts had been covered in the course. The results of the survey are as follows:

- 1) 82% of respondents agreed that the exercise was reasonable and useful.
- 2) 72% of respondents agreed that the teaching method through the tutorial supported their learning process.
- 3) 83% of respondents expressed interest in learning more about the applications of programming in cybersecurity after completing the tutorial.

Regarding what they liked about the tutorial, some of the student comments included:

- “It helped me learn how to read and break down code better.”
- “I like how it was easy to follow and not too hard to learn.”
- “I liked it because the tutorial teaches in an easy way.”
- “I like there were steps provided along with pictures to help walk through the process.”
- “Taught me an area of knowledge that I had never learned up to this point in my college education.”

Regarding what they disliked about the tutorial, some of the student comments included:

- “At first glance it seems overwhelming.”
- “It was pretty difficult for me at times.”
- “There is not a lot of explanation and information; increase explanation.”

Overall, the students’ response to the tutorial has been positive. Most students appreciated the step-by-step walkthrough provided to complete the tutorial. However, some students found the tutorial overwhelming and difficult and wanted more explanation to be added. This reaction is understandable, given that many students were exposed to scripting for the first time in this course. Additionally, there was no prerequisite for prior knowledge of encryption concepts. In our view, the hands-on nature of this activity is one of its greatest benefits. The best part was the step-by-step instructions and snapshots that were available. The activity allows students to move beyond a purely theory-based approach and engage with practical, real-world applications. We hope that this activity will help students build confidence in scripting, making it easier for them to tackle similar challenges in future courses, such as the network security and penetration testing course.

## 6. DISCUSSION

### 6.1 Learnings From the Tutorial

The tutorial teaches students to make meaning out of the obfuscated code of a simple encryption algorithm. Some of the tasks/concepts that students do - understand the functions in the code, identify and rename the variables and functions, get

ASCII value of characters, use logical operators (XOR, AND, and OR), use loops and lists in a program, convert hexadecimal to decimal, and finally hash a given value.

In this tutorial example, students learn about reverse encryption using their knowledge of Python programming. Following the step-by-step process, they apply their programming skills to a real use-case that an information security manager may face.

### 6.2 Benefits of the Tutorial

This tutorial is relevant for educators who want to teach students programming in the context of information security (specifically reverse engineering within cryptography).

The tutorial process creates an engaging, active work environment. This tutorial is an example of how a topic in an information security programming course should be taught, emphasizing the importance of applying programming skills to solve real-world business problems. This approach enhances understanding of security concepts and improves student retention by linking programming concepts to business applications. Additionally, this experience boosts students’ confidence in applying learned concepts to various scenarios.

Instead of providing a problem and asking students to write code to solve it, we start by deconstructing preexisting Python scripts, encouraging critical thinking about understanding code they might encounter in real-world situations. Importantly, we take a context-driven approach: introducing a business problem first, then the appropriate methods to solve it.

### 6.3 Applications of the Learnings

Reverse engineering malware involves analyzing malware to understand its functionality and purpose, determine how to remove it from a system, or create defenses against it. The process of reverse engineering is used to investigate viruses and find practical solutions to counteract them. Reverse engineering malware is challenging due to obfuscation techniques, encryption, and frequent code changes by malware authors. Understanding the algorithm at a basic level gives students insight into reverse-engineering malware code. This knowledge prepares students to tackle such situations.

### 6.4 Adopting the Idea

The instructors who adopt the tutorial-based instruction idea should integrate it with a cybersecurity course module. Moreover, they should use real-world scenarios to make the tutorials relatable and applicable. They should define the purpose and objective of the tutorial. They should ensure the tutorial includes detailed, self-explanatory steps and code snippets. They should add comments in the code to explain key concepts. Additionally, they should provide the code files, datasets and log files for hands-on practice.

## 7. CONCLUSIONS

The objective of this paper was to present a sample tutorial for addressing security problems from a business perspective, which can be used for teaching programming languages in security programming courses. We provide a step-by-step process for instructing students on how to solve a Reverse Engineering challenge using Python.

The overall feedback about the tutorial has been positive and the fact that students want to learn more about the

Our next step is to expand the tutorial-based method of instruction to other modules in the course. Our goal is to introduce each topic in the security programming curriculum within the context of a different case study or real-world example (monitoring network traffic, analysis of security logs, etc.). We also plan to develop additional follow-up tutorials on the topic of encryption for other courses in our program, such as the network security course. We anticipate these topics will evolve as we continue teaching the course and as new topics gain importance and popularity. Throughout the course, we engage in important discussions about the applications of these concepts.

Atkinson, R. K., Derry, S. J., Renkl, A., & Wortham, D. (2000). Learning From Examples: Instructional Principles From the Worked Examples Research. *Review of Educational Research*, 70(2), 181-214. <https://doi.org/10.3102/00346543070002181>

Bawamohiddin, A. B., & Razali, R. (2017). Problem-Based Learning for Programming Education. *International Journal on Advanced Science, Engineering and Information Technology*, 7(6), 2035-2050. <https://doi.org/10.18517/ijaseit.7.6.2232>

Bogdanchikov, A., Zhaparov, M., & Suliye, R. (2013). Python to Learn Programming. *Journal of Physics: Conference Series*, 423(1), 012027. <https://doi.org/10.1088/1742-6596/423/1/012027>

Bravo, C. A. (2023, September 27). *5 of the Top Programming Languages for Cybersecurity*. WeLiveSecurity. <https://www.welivesecurity.com/en/secure-coding/5-top-programming-languages-cybersecurity/>

Breese, J. L., & Gardner, B. (2024). Using Python to Inspire Novice Cybersecurity Learners (K-12 and University Level). *Proceedings of the 2024 ISCAP Conference*, Baltimore, MD.

Chokhonelidze, G., Basilaia, G., Kantaria, M., & Dgebuadze, M. (2020). Teaching the Cybersecurity Courses at the University in Georgia. *International Journal of Innovative Science and Research Technology*, 5(4), 648-651. <https://www.ijisrt.com/teaching-the-cybersecurity-courses-at-the-university-in-georgia>

Chutisowan, K., Trinantararat, P., Ratnarangsank, K., Jundang, N., & Suwatcharakulthorn, J. (2021). Combination of Problem-Based Learning and Case-Based Learning in SQL Programming for Data Analysis. *The 6th International STEM Education Conference (iSTEM-Ed)*, Pattaya, Thailand. <https://doi.org/10.1109/iSTEM-Ed52129.2021.9625105>

Combéfi, S., Beresnevičius, G., & Dagienė, V. (2016). Learning Programming Through Games and Contests: Overview, Characterisation and Discussion. *Olympiads in Informatics*, 10(1), 39-60. <https://doi.org/https://doi.org/10.15388/oi.2016.03>

Cooper, S., & Cunningham, S. (2010). Teaching Computer Science in Context. *ACM Inroads*, 1(1), 5-8. <https://doi.org/10.1145/1721933.1721934>

Ksiezopolski, B., Mazur, K., Miskiewicz, M., & Rusinek, D. (2022). Teaching a Hands-on CTF-Based Web Application Security Course. *Electronics*, 11(21), 3517. <https://doi.org/10.3390/electronics11213517>



- Kuk, K., Milic, P., Spalević, P., & Gocic, M. (2019). Algorithm Design in Python for Cybersecurity. *The 28th International Electrotechnical and Computer Science Conference*, Portorož, Slovenia.
- Lewis, J. A., & Crumpler, W. (2019). *The Cybersecurity Workforce Gap*. <https://www.csis.org/analysis/cybersecurity-workforce-gap>
- Lindberg, R. S., Laine, T. H., & Haaranen, L. (2019). Gamifying Programming Education in K-12: A Review of Programming Curricula in Seven Countries and Programming Games. *British Journal of Educational Technology*, 50(4), 1979-1995. <https://doi.org/10.1111/bjet.12685>
- Mathrani, A., Christian, S., & Ponder-Sutton, A. (2016). PlayIT: Game Based Learning Approach for Teaching Programming Concepts. *Journal of Educational Technology & Society*, 19(2), 5-17. <https://www.jstor.org/stable/jeductechsoci.19.2.5>
- Merrill, M. D. (2002). First Principles of Instruction. *Educational Technology Research and Development*, 50, 43-59. <https://doi.org/10.1007/BF02505024>
- Morgan, S. (2021). *Cybersecurity Jobs Report: 3.5 Million Unfilled Positions In 2025*. Cyber Crime Magazine. <https://cybersecurityventures.com/jobs/>
- Moumoutzis, N., Boukeas, G., Vassilakis, V., Pappas, N., Xanthaki, C., Maragkoudakis, I., Deligiannakis, A., & Christodoulakis, S. (2018). Design, Implementation and Evaluation of a Computer Science Teacher Training Programme for Learning and Teaching of Python Inside and Outside School: Establishing and Supporting Code Clubs to Learn Computer Programming by Self-contained Examples. *Proceedings of the 11th Interactive Mobile Communication Technologies and Learning (IMCL) Conference*, Thessaloniki, Greece. [https://doi.org/10.1007/978-3-319-75175-7\\_56](https://doi.org/10.1007/978-3-319-75175-7_56)
- NIST CSF 2.0. (2024). <https://doi.org/10.6028/NIST.CSWP.29>
- NSA, CISA, & US. (2023). *The Case for Memory Safe Roadmaps* <https://www.nsa.gov/Press-Room/Press-Releases-Statements/Press-Release-View/article/3608324/us-and-international-partners-issue-recommendations-to-secure-software-products/>
- Nuutila, E., Törmä, S., & Malmi, L. (2008). Learning Programming With the PBL Method — Experiences on PBL Cases and Tutoring. In J. Bennedsen, M. E. Caspersen, & M. Kölling (Eds.), *Reflections on the Teaching of Programming: Methods and Implementations* (Vol. 4821, pp. 47-67). [https://doi.org/10.1007/978-3-540-77934-6\\_5](https://doi.org/10.1007/978-3-540-77934-6_5)
- Odetokun, I. (2020). An Assessment of the Effectiveness of Python in Cybersecurity. *Proceedings of the International Conference on Computational Science and Computational Intelligence*. [https://www.researchgate.net/profile/Itunuoluwa-Odetokun/publication/366276195\\_AN\\_ASSESSMENT\\_OF\\_THE\\_EFFECTIVENESS\\_OF\\_PYTHON\\_IN\\_CYBERS-ECURITY/links/63a1f13e9835ef25903595eb/AN-ASSESSMENT-OF-THE-EFFECTIVENESS-OF-PYTHON-IN-CYBERSECURITY.pdf](https://www.researchgate.net/profile/Itunuoluwa-Odetokun/publication/366276195_AN_ASSESSMENT_OF_THE_EFFECTIVENESS_OF_PYTHON_IN_CYBERS-ECURITY/links/63a1f13e9835ef25903595eb/AN-ASSESSMENT-OF-THE-EFFECTIVENESS-OF-PYTHON-IN-CYBERSECURITY.pdf)
- Omeh, C. B., Olelewe, C. J., & Nwangwu, E. C. (2022). Impact of Teaching Computer Programming Using Innovative Pedagogy Embedded With Live Online Lectures and Related Tools: A Randomized Control Trial. *Computer Applications in Engineering Education*, 30(5), 1390-1405. <https://doi.org/10.1002/cae.22527>
- Peng, W. (2010). Practice and Experience in the Application of Problem-Based Learning in Computer Programming Course. *International Conference on Educational and Information Technology*, Chongqing, China.
- Perin, D. (2011). Facilitating Student Learning Through Contextualization: A Review of Evidence. *Community College Review*, 39(3), 268-295. <https://doi.org/10.1177/0091552111416227>
- Phuong, C. (2022). *Teaching Cybersecurity: A Project-Based Learning and Guided Inquiry Collaborative Learning Approach* [Doctoral dissertation, University of Tennessee at Chattanooga]. <https://scholar.utc.edu/theses/769/>
- Predmore, S. R. (2005). Putting It Into Context. *Techniques: Connecting Education and Careers*, 80(1), 22-25. <https://eric.ed.gov/?id=EJ718606>
- PYPL Popularity of Programming Language. (2025). <https://pypl.github.io/PYPL.html>
- Reeder, F., & Paller, A. (2021). *What Works in Finding Elite Cybersecurity Talent: Promising Practices for Chief Information Officers*. CIO Institute. [https://www.cio.org/assets/pdf/CIO\\_EliteCyberTalent%20NEW\\_16pp.pdf](https://www.cio.org/assets/pdf/CIO_EliteCyberTalent%20NEW_16pp.pdf)
- Rivet, A. E., & Krajcik, J. S. (2008). Contextualizing Instruction: Leveraging Students' Prior Knowledge and Experiences to Foster Understanding of Middle School Science. *Journal of Research in Science Teaching: The Official Journal of the National Association for Research in Science Teaching*, 45(1), 79-100. <https://doi.org/10.1002/tea.20203>
- Segal, J., & Ahmad, K. (1993). The Role of Examples in the Teaching of Programming Languages. *Journal of Educational Computing Research*, 9(1), 115-129. <https://doi.org/10.2190/X63F-X1QX-V4KL-BJEX>
- Sherman, A. T., Peterson, P. A. H., Golaszewski, E., LaFemina, E., Goldschen, E., Khan, M., Mundy, L., Rather, M., Solis, B., Tete, W., Valdez, E., Weber, B., Doyle, D., O'Brien, C., Oliva, L., Roundy, J., & 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>
- TIOBE Index for April 2025. (2025). TIOBE. <https://www.tiobe.com/tiobe-index/>
- Wen, S.-F., & Katt, B. (2019). Toward a Context-Based Approach for Software Security Learning. *Journal of Applied Security Research*, 14(3), 288-307. <https://doi.org/10.1080/19361610.2019.1585704>
- Yang, S. C., & Wen, B. (2017). Toward a Cybersecurity Curriculum Model for Undergraduate Business Schools: A Survey of AACSB-Accredited Institutions in the United States. *Journal of Education for Business*, 92(1), 1-8. <https://doi.org/10.1080/08832323.2016.1261790>

#### AUTHOR BIOGRAPHIES

**Gunjan Batra** is an assistant professor of information systems



and security in the Coles College of Business at Kennesaw State University, GA. She teaches information security related courses such as Security Script Programming, Management, IT Audit, Management of Information Security. She received her Ph.D. in Management from Rutgers

University, NJ, specializing in Information Security. Prior to her doctoral degree, she worked as a consultant at KPMG in their IT Advisory practice, helping clients with Information security related issues. Gunjan is also a Certified Information Systems Auditor (CISA). Her research interests include Information Systems Security, Access Control, Information Systems Audit, and Data Analytics. She has published her research in several journals and presented at conferences in these areas.

**Humayun Zafar** is a professor of information security and



assurance, and the Director of Fintech at Kennesaw State University. He completed his doctorate in Business Administration with an emphasis in IT from the University of Texas at San Antonio. His research and teaching interests include cybersecurity and fintech. His work has appeared in journals

and conferences such as *Communications of the AIS*, *Information Systems Frontiers*, and *HICSS*.

**Pamila Dembla** is an associate professor of information



systems and security in the Coles College of Business at Kennesaw State University, GA. She teaches database related courses. She is an interdisciplinary scholar, and her research interests include working on cross cultural issues related to global IT projects, specifically with respect to gender. To that effect, her research

extends across many cultures including the United States, India, Russia, Ukraine, and Mexico. She has published numerous articles in journals, book chapters, and presented at various regional, national, and international conferences. She has mentored a number of students, interns, and chaired doctoral students during her tenure.

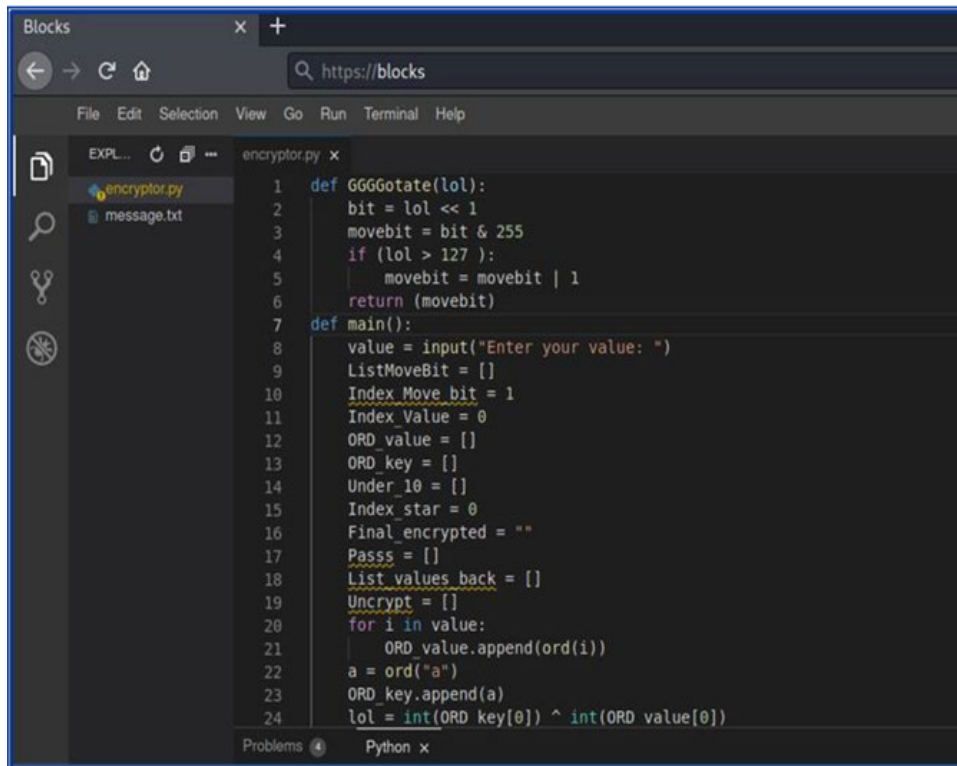
## APPENDICES

### Appendix A. Solution Steps

Below are the solution steps:

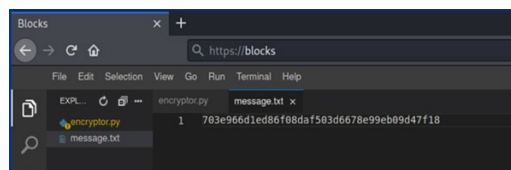
#### 1. Initial encounter

The challenge presents you with an interactive IDE with two files in the editor explorer: some Python code in `encryptor.py`. (**Figure A1**) and an encrypted message in `message.txt`: “703e966d1ed86f08daf503d6678e99eb09d47f18” (**Figure A2**). Copy the code to your Python IDE of choice or use the presented editor.



```
1 def GGGotate(lol):
2     bit = lol << 1
3     movebit = bit & 255
4     if (lol > 127):
5         movebit = movebit | 1
6     return (movebit)
7
8 def main():
9     value = input("Enter your value: ")
10    ListMoveBit = []
11    Index Move bit = 1
12    Index_Value = 0
13    ORD_value = []
14    ORD_key = []
15    Under_10 = []
16    Index_star = 0
17    Final_encrypted = ""
18    Passs = []
19    List_values_back = []
20    Uncrypt = []
21    for i in value:
22        ORD_value.append(ord(i))
23        a = ord("a")
24        ORD_key.append(a)
25        lol = int(ORD_key[0]) ^ int(ORD_value[0])
```

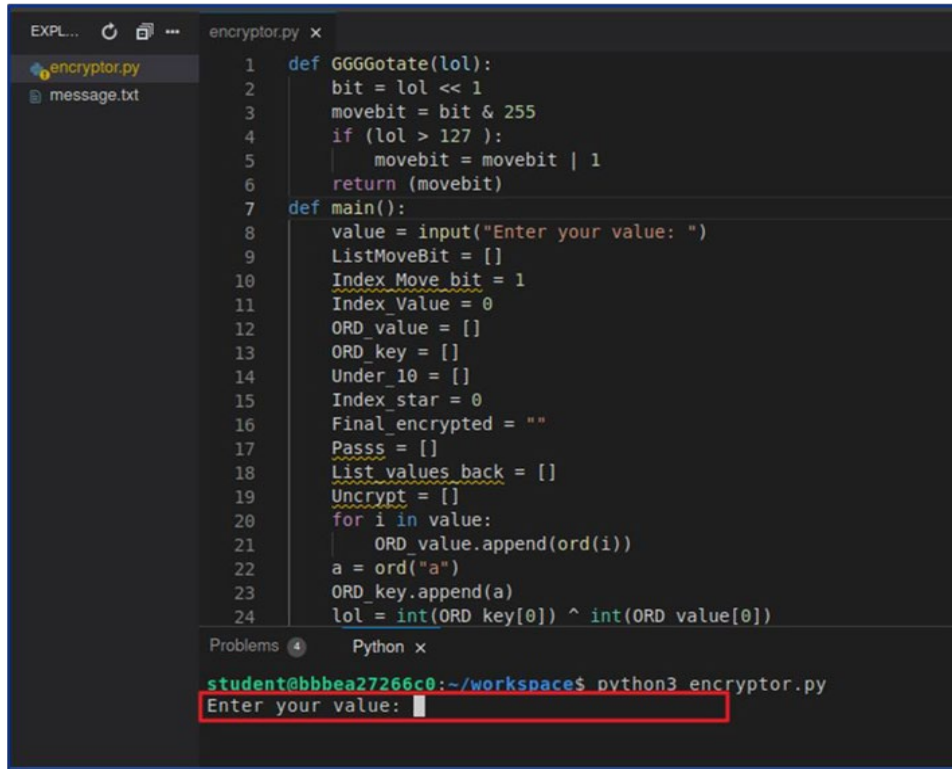
Figure A1. `encryptor.py` in the Interactive Editor



```
1 703e966d1ed86f08daf503d6678e99eb09d47f18
```

Figure A2. `message.txt` in the Interactive Editor

Executing the code reveals that the program is an interactive script that waits for the user to input a value (Figure A3).



```

1 def GGGGotate(lol):
2     bit = lol << 1
3     movebit = bit & 255
4     if (lol > 127):
5         movebit = movebit | 1
6     return (movebit)
7 def main():
8     value = input("Enter your value: ")
9     ListMoveBit = []
10    Index Move bit = 1
11    Index_Value = 0
12    ORD_value = []
13    ORD_key = []
14    Under_10 = []
15    Index_star = 0
16    Final_encrypted = ""
17    Passs = []
18    List values back = []
19    Uncrypt = []
20    for i in value:
21        ORD_value.append(ord(i))
22        a = ord("a")
23        ORD_key.append(a)
24        lol = int(ORD_key[0]) ^ int(ORD_value[0])

```

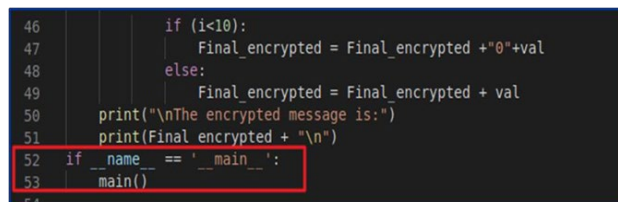
student@bbbea27266c0:~/workspace\$ python3 encryptor.py  
Enter your value:

Figure A3. Execution of encryptor.py

First, by looking at the script, you can see that some functions and variables are obfuscated. Start with understanding their role and changing their names to human-readable names.

## 2. Understanding the code

On lines 1–51, the code contains functions called GGGGotate() and main(). On lines 52–53, an “if” condition verifies that the script runs as the main program, then executes the function called main() (Figure A4). More information about the “if” `__name__ == “__main__”` condition can be found in the link: <https://stackoverflow.com/questions/419163/what-does-if-name-main-do/>



```

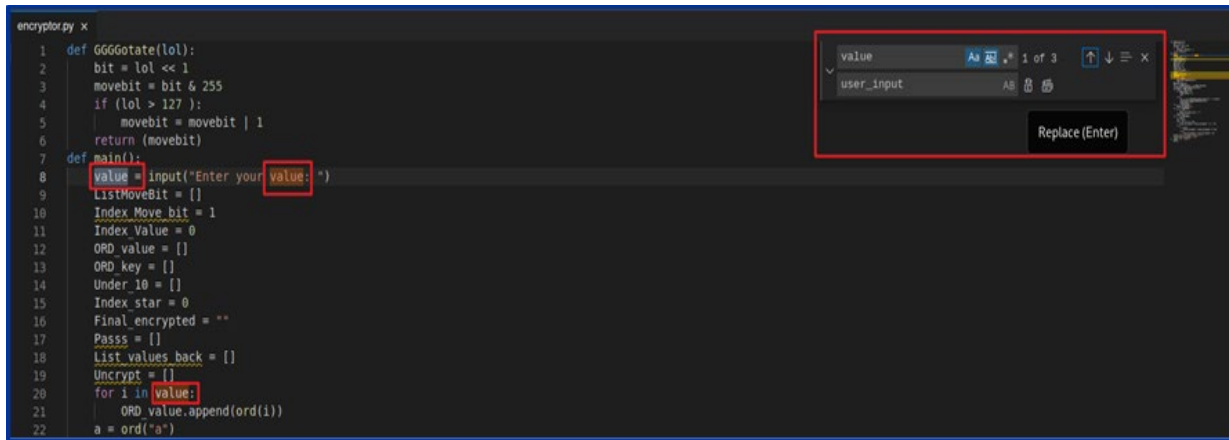
46     if (i<10):
47         Final_encrypted = Final_encrypted + "0"+val
48     else:
49         Final_encrypted = Final_encrypted + val
50     print("\nThe encrypted message is:")
51     print(Final_encrypted + "\n")
52 if __name__ == '__main__':
53     main()

```

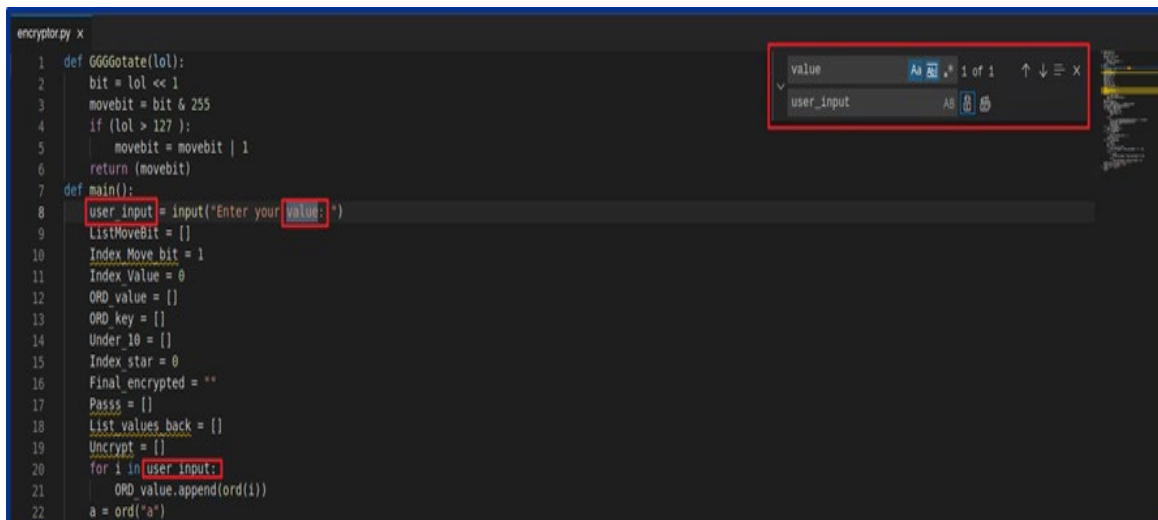
Figure A4. Calling the main() Function

The main() function starts on line 7. On line 8, the variable “value” receives an input from the user. Rename the variable for the whole script to a name that matches the variable role.

You can rename a variable for a whole script by using Ctrl+F to search for the variable name. Then, click on “aa” and “ab” to filter only match case sensitive and whole words. Next, click on the small arrow to open the “Replace” pane and insert a new variable name, for example, “user\_input.” Finally, use the “Find” pane arrows to navigate between the three matches and replace only the variable name using “Replace” (Figure A5 and Figure A6).



**Figure A5. Before Renaming the “value” Variable to “user input”**



**Figure A6. After Renaming the “value” Variable to “user input”**

Lines 9-19 define other variables with different types and values whose role is unknown for the moment.

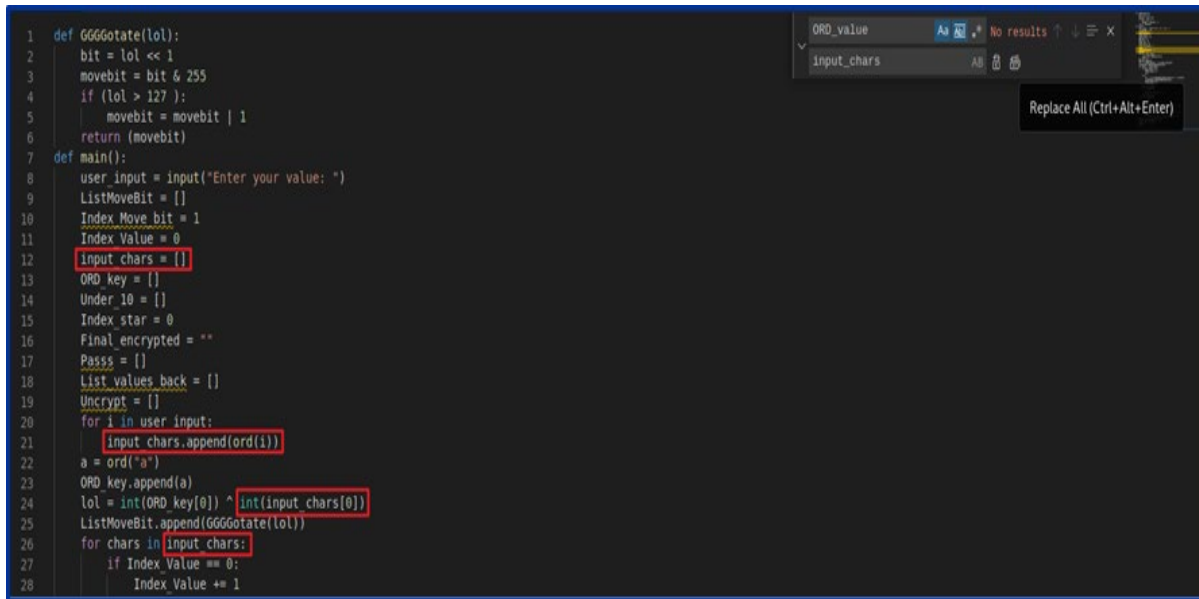
! Not all variables are used during this script execution.

On lines 20–21, the script iterates through a string stored inside “user\_input” and adds each character in ASCII decimal format to a list called “ORD\_value.”

● ASCII stands for “American Standard Code for Information Interchange”, and it is a character encoding standard for electronic communication. It follows a table containing numbers, letters, control characters, and other symbols. Each character is assigned a unique 7-bit code.

Rename the “ORD\_value” variable for the whole script to a name that matches its role in storing a list of user input characters. For example, “input\_chars” (Figure A7).





```

1 def GGGotate(lol):
2     bit = lol << 1
3     movebit = bit & 255
4     if (lol > 127):
5         movebit = movebit | 1
6     return (movebit)
7 def main():
8     user_input = input("Enter your value: ")
9     ListMoveBit = []
10    Index_Move_bit = 1
11    Index_Value = 0
12    input_chars = []
13    ORD_key = []
14    Under_10 = []
15    Index_star = 0
16    Final_encrypted = ""
17    Passs = []
18    List_values_back = []
19    Uncrypt = []
20    for i in user_input:
21        input_chars.append(ord(i))
22    a = ord("a")
23    ORD_key.append(a)
24    lol = int(ORD_key[0]) ^ int(input_chars[0])
25    ListMoveBit.append(GGGotate(lol))
26    for chars in input_chars:
27        if Index_Value == 0:
28            Index_Value += 1

```

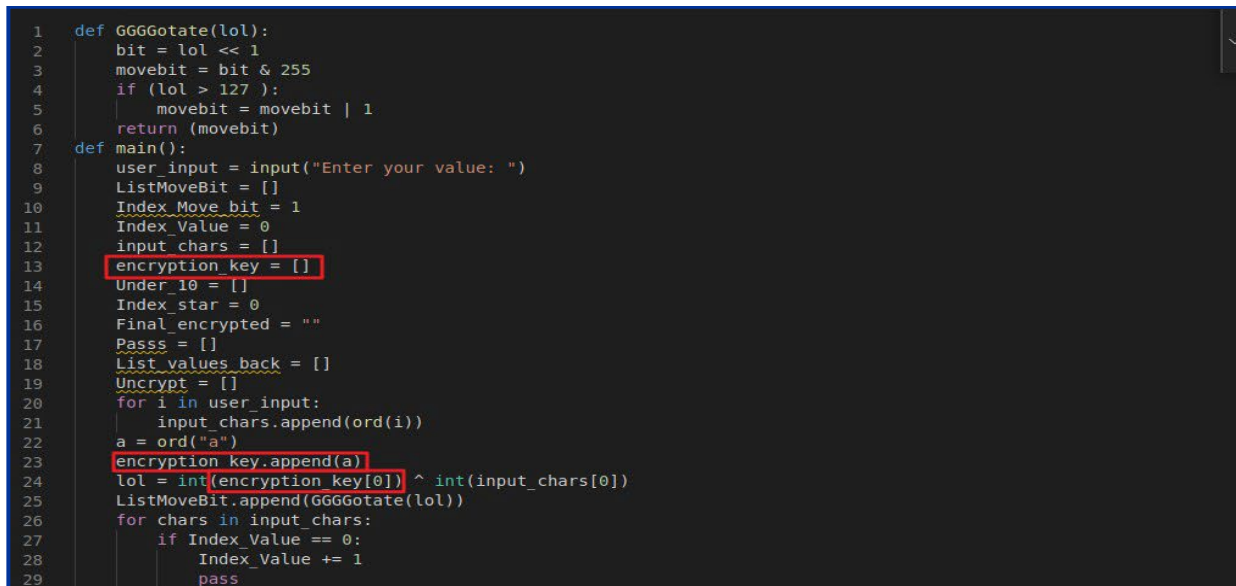
Figure A7. “ORD\_value” Variable Renamed to “input\_chars”

On lines 22–23, the script adds the decimal ASCII value of “a,” which equals 97, to a variable called “ORD\_key.” Next, on line 24, the script uses “^” operator between the first value inside of the “ORD\_key” list, which is 97 (0110 0001), and the first character of the user input in ASCII decimal format. The result is stored into a variable called “lol.”

The “^” manipulation converts a decimal value to a binary value then performs an “XOR” operation between the binary values.

! XOR is a logical operation that is true (!) only if its arguments differ.

The script hints that “ORD\_key” stores the encryption key because of the variable name and the “XOR” operation between “ORD\_key” to the user input’s first ASCII decimal character. Also, by inspecting “lol” usage, you can infer that its role is to store “XOR” operation results. Therefore, you should rename “ORD\_key” to “encryption\_key” (Figure A8 ) and “lol” to “xor\_result” (Figure A9).

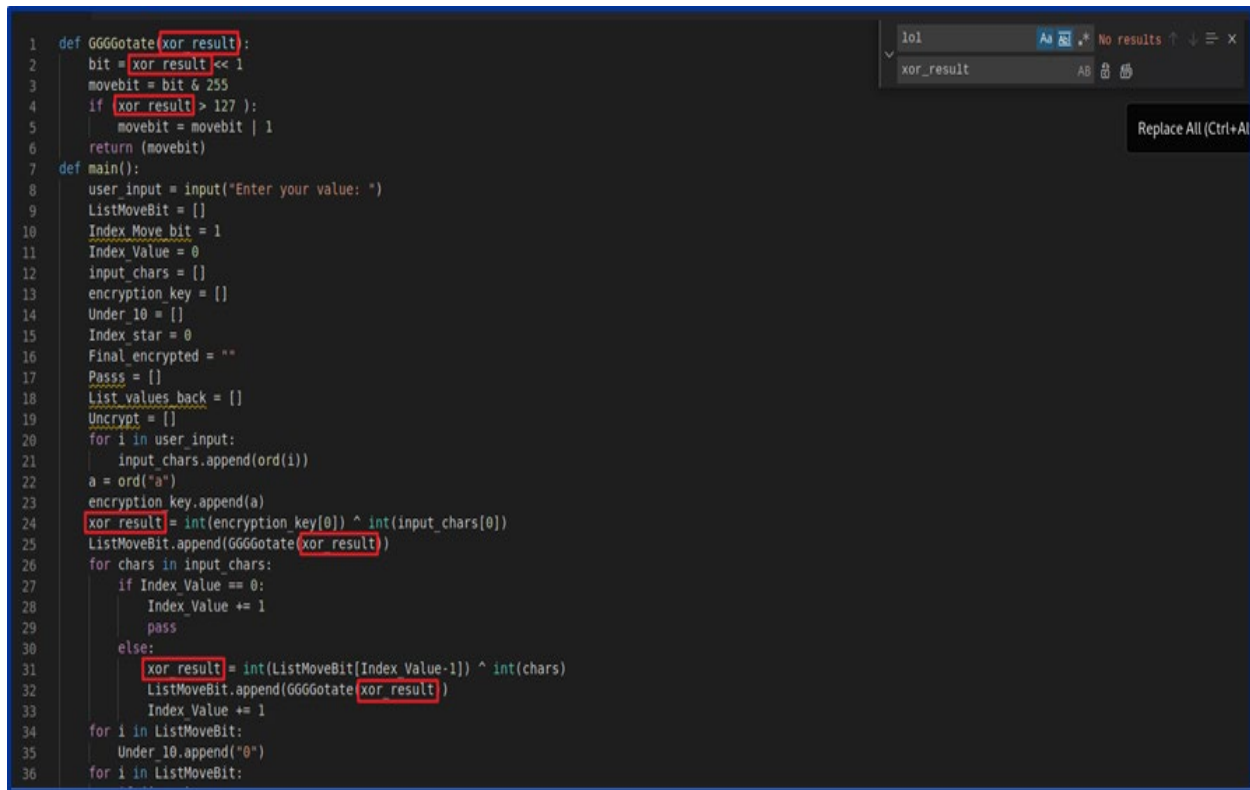


```

1 def GGGotate(lol):
2     bit = lol << 1
3     movebit = bit & 255
4     if (lol > 127):
5         movebit = movebit | 1
6     return (movebit)
7 def main():
8     user_input = input("Enter your value: ")
9     ListMoveBit = []
10    Index_Move_bit = 1
11    Index_Value = 0
12    input_chars = []
13    encryption_key = []
14    Under_10 = []
15    Index_star = 0
16    Final_encrypted = ""
17    Passs = []
18    List_values_back = []
19    Uncrypt = []
20    for i in user_input:
21        input_chars.append(ord(i))
22    a = ord("a")
23    encryption_key.append(a)
24    lol = int(encryption_key[0]) ^ int(input_chars[0])
25    ListMoveBit.append(GGGotate(lol))
26    for chars in input_chars:
27        if Index_Value == 0:
28            Index_Value += 1
29    pass

```

Figure A8. “ORD\_key” Variable Renamed to “encryption\_key”



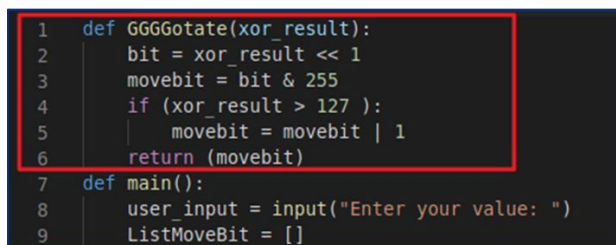
```

1 def GGGotate(xor_result):
2     bit = xor_result << 1
3     movebit = bit & 255
4     if (xor_result > 127):
5         movebit = movebit | 1
6     return (movebit)
7 def main():
8     user_input = input("Enter your value: ")
9     ListMoveBit = []
10    Index_Move_bit = 1
11    Index_Value = 0
12    input_chars = []
13    encryption_key = []
14    Under_10 = []
15    Index_star = 0
16    Final_encrypted = ""
17    Passes = []
18    List_values_back = []
19    Uncrypt = []
20    for i in user_input:
21        input_chars.append(ord(i))
22        a = ord("a")
23        encryption_key.append(a)
24        xor_result = int(encryption_key[0]) ^ int(input_chars[0])
25        ListMoveBit.append(GGGotate(xor_result))
26    for chars in input_chars:
27        if Index_Value == 0:
28            Index_Value += 1
29            pass
30        else:
31            xor_result = int(ListMoveBit[Index_Value-1]) ^ int(chars)
32            ListMoveBit.append(GGGotate(xor_result))
33            Index_Value += 1
34    for i in ListMoveBit:
35        Under_10.append("0")
36    for i in ListMoveBit:

```

Figure A9. “lol” Variable Renamed to “xor\_result”

Line 25 calls to the GGGotate() function while providing “lol” as a variable input, then the function’s output is appended to the “ListMoveBit” list. The GGGotate() function is defined in lines 1–6 (Figure A10).



```

1 def GGGotate(xor_result):
2     bit = xor_result << 1
3     movebit = bit & 255
4     if (xor_result > 127):
5         movebit = movebit | 1
6     return (movebit)
7 def main():
8     user_input = input("Enter your value: ")
9     ListMoveBit = []

```

Figure A10. GGGotate() Function

The “<<” operator in line 2 rotates left several times the binary bits of the variable in hand. In this case, the bits rotate one bit left and add 0 on the right side of the “xor\_result” binary code, the binary’s decimal output after the “<<” operation is stored in “bit” variable. Line 3 uses “&” between 255 decimal (1111 1111) and “bit” variable (which is “xor\_result” in decimal format). The result is stored in a variable called “movebit.”

The “&” manipulation converts a decimal value to a binary value, then performs “AND” operation between the binary values.

! AND is a logical operation that is true (!) when one of the arguments are true (!).

A condition in line 4 checks if “xor\_result” decimal is higher than 127 (0111 1111), then in line 5 it performs “|” manipulation between 1 decimal (0000 0001) and “movebit” (which stores the output of “bit” and 255). The “|” manipulation converts a decimal value to a binary value and performs an “OR” operation between the binary values.

! OR is a logical operation that is true (!) when one of the arguments are true (!).

On line 6, the function ends and returns the value of “movebit.” Upon understanding the role of the GGGotate function, rename the function and its variables accordingly. Start with “bit,” which stores the left rotated “xor\_result” variable value in integer format and can be renamed to “xor\_rotated\_left.” The function returns the “movebit” variable after a few binary manipulations; therefore, it can be renamed to “bits\_to\_return.” After understanding the GGGotate function’s main role is to rotate left, you can rename it to “rotateLeft.” (Figure A11).

```
1 def rotateLeft(xor_result):
2     xor_rotated_left = xor_result << 1
3     bits_to_return = xor_rotated_left & 255
4     if (xor_result > 127):
5         bits_to_return = bits_to_return | 1
6     return (bits_to_return)
7
8 def main():
9     user_input = input("Enter your value: ")
10    ListMoveBit = []
11    Index_Move_bit = 1
12    Index_Value = 0
13    input_chars = []
14    encryption_key = []
15    Under_10 = []
16    Index_star = 0
17    Final_encrypted = ""
18    Passs = []
19    List_values_back = []
20    Uncrypt = []
21    for i in user_input:
22        input_chars.append(ord(i))
23    a = ord("a")
24    encryption_key.append(a)
25    xor_result = int(encryption_key[0]) ^ int(input_chars[0])
26    ListMoveBit.append(rotateLeft(xor_result))
27    for chars in input_chars:
28        if Index_Value == 0:
29            Index_Value += 1
30            pass
31        else:
32            xor_result = int(ListMoveBit[Index_Value-1]) ^ int(chars)
33            ListMoveBit.append(rotateLeft(xor_result))
34            Index_Value += 1
35    for i in ListMoveBit:
```

Figure A11. Renaming GGGotate Function and Its Variables

On line 25, the output of the “rotateLeft” function is appended to a list called “ListMoveBit,” which you can infer its role is to store a list of “rotateLeft” outputs. You can rename it to “rotated\_left\_list” (Figure A12).

```

7 def main():
8     user_input = input("Enter your value: ")
9     rotated_left_list = []
10    Index_Move_bit = 1
11    Index_Value = 0
12    input_chars = []
13    encryption_key = []
14    Under_10 = []
15    Index_star = 0
16    Final_encrypted = ""
17    Passes = []
18    List_values_back = []
19    Uncrypt = []
20    for i in user_input:
21        input_chars.append(ord(i))
22        a = ord("a")
23        encryption_key.append(a)
24        xor_result = int(encryption_key[0]) ^ int(input_chars[0])
25        rotated_left_list.append(rotateLeft(xor_result))
26        for chars in input_chars:
27            if Index_Value == 0:
28                Index_Value += 1
29                pass
30            else:
31                xor_result = int(rotated_left_list[Index_Value-1]) ^ int(chars)
32                rotated_left_list.append(rotateLeft(xor_result))
33                Index_Value += 1
34        for i in rotated_left_list:
35            Under_10.append("0")
36        for i in rotated_left_list:
37            if (i < 9):
38                Under_10[Index_star] = "1"

```

Figure A12. Renamed “ListMoveBit” to “rotated\_left\_list”

Lines 26–33 run in for loop iterations through “input\_chars,” which is a list that stores the input characters previously provided on lines 20–21 in ASCII decimal format. Each item is stored in the “chars” variable. Because each item is singular, rename “chars” to “char” (Figure A13).

```

18    List_values_back = []
19    Uncrypt = []
20    for i in user_input:
21        input_chars.append(ord(i))
22        a = ord("a")
23        encryption_key.append(a)
24        xor_result = int(encryption_key[0]) ^ int(input_chars[0])
25        rotated_left_list.append(rotateLeft(xor_result))
26        for char in input_chars:
27            if Index_Value == 0:
28                Index_Value += 1
29                pass
30            else:
31                xor_result = int(rotated_left_list[Index_Value-1]) ^ int(char)
32                rotated_left_list.append(rotateLeft(xor_result))
33                Index_Value += 1
34        for i in rotated_left_list:

```

Figure A13. Renamed “chars” to char

Because the “Index\_Value” initial value provided on line 11 is 0, the “if” condition on lines 27–29 is true only on the first iteration, which increases “Index\_Value” by 1 and skips lines 30–33 until the next iteration.

On lines 30–33, “xor\_result” stores a new output of an “XOR” operation between the previous item on “rotated\_left\_list” that “Index\_Value” points to, and the current iteration character is ASCII decimal format from “input\_chars” list. Then, the “rotateLeft” function rotates the “xor\_result” left and it is appended to “rotated\_left\_list.” Finally, in line 33, “Index\_Value” increases by 1. Lines 34–39 modify a few unused variables and can be skipped.

The for loop in lines 40–49 run through “rotated\_left\_list” and set “i” as the variable name for each item in the list (Figure A14). On lines 41–42, the variable “val” receives the hexadecimal format of “i.” Lines 43–49 have two “if” conditions that, together, append “val” value to the final encrypted message. The conditions checks if the “i” integer value is lower than 16. If true, add the prefix “0” to “val” for the encrypted message (Figure A14).

In lines 50–51, the script prints the final encrypted message.

```
34 for i in rotated_left_list:
35     Under_10.append("0")
36 for i in ListMoveBit:
37     if (i < 9):
38         Under_10[Index_star] = "1"
39         Index_star += 1
40 for i in ListMoveBit:
41     x = hex(i)
42     val = x[2:]
43     if(i > 9) and (i < 16):
44         Final_encrypted = Final_encrypted + "0" + val
45     else:
46         if (i<10):
47             Final_encrypted = Final_encrypted + "0"+val
48         else:
49             Final_encrypted = Final_encrypted + val
50 print("\nThe encrypted message is:")
51 print(Final_encrypted + "\n")
52 if __name__ == '__main__':
53     main()
```

**Figure A14. Final Encrypted Message**

### 3. Decrypting the message

To decrypt a message, you need to reverse the steps every two characters take before being printed to the terminal. The first two characters of the message are “70.” On lines 41–42, “val” stores “70,” which is the hexadecimal format of the “i” variable. Retrieve the decimal of 0x70 by executing the command “echo \$(16#70),” or with the online tool in Figure A15, available at <https://www.rapidtables.com/convert/number/binary-to-decimal.html>.

Hexadecimal to Decimal converter

From: Hexadecimal To: Decimal

Enter hex number: 70

Decimal number: 112

Binary number: 1110000

**Figure A15. Converting 0x70 to Decimal and Binary**

Converting each two digits of the encrypted message to their decimal format, forms a list of left-rotated decimals: 112, 62, 150, 109, 30, 216, 111, 8, 218, 245, 3, 214, 103, 142, 153, 235, 9, 212, 127.



! In addition to the manual decryption, you can also create an automation script that decrypts the message.

Next, lines 30–33 are skipped on the first iteration. Therefore, skip to line 25, which append “xor\_result” to “rotated\_left\_list” upon left rotation. To find “xor\_result,” rotate the binary bits right. If the 0x70 binary number is 0111 0000, then rotating the bits right turns to 0011 1000, which is equal to 0x38 or 56 in decimal.

Finally, obtain the first input character by reversing the “XOR” operation output stored inside “xor\_result.” Convert “xor\_result” and “encrypted\_key[0]” decimal numbers to their binary numbers, then compare which bits of “encrypted\_key[0]” differ from “input\_chars[0]” bits. Every “0” in “xor\_result” means the bit is identical, and every “1” means the bit is different.

For example, “xor\_result” is “56” (0011 1000) and “encryption\_key[0]” is “97” (0110 0001). The first digit of “xor\_result” is “0,” then the first digit of “encryption\_key[0],” which is “0,” will stay “0.” The second digit of “xor\_result” is “0,” and for “encryption\_key[0]” is “1.” Because “xor\_result” has “0,” then the bit “1” of “encryption\_key[0]” stays. Because “1” in the third digit of “xor\_result” indicates that the original parameter bit was different from “encryption\_key[0],” the “encryption\_key[0]” third binary number flips to “0.” By continuing to reverse the “XOR” operation, you reveal the final decimal value of the first character to be “89” (0101 1001). According to the ASCII table, the decimal number “89” is “Y.”

From the second digits’ pair until the last pair, the flow changes a bit since “Index\_Value” is not equal to “0” anymore. The decimal “62” (00111110) turns to “31” (0001 1111) upon right rotation on line 32 by reversing the “rotateLeft” function. Next, on line 31, reverse the “XOR” operation of the previous decimal value “112” (0111 0000) and the current iteration decimal char while considering that “xor\_result” is decimal “31” (0001 1111). Calculating the binary differences again yields that the second “char” decimal equals “111” (0110 1111), which is “o” according to the ASCII table. Proceed to the next digits pair until full message decryption. The final decrypted message is “You broke the blocks.”

#### 4. Submitting the flag (or the hash of decrypted message)

To capture the flag (or the hash of the decrypted message) & complete the challenge, you need to hash the decrypted message using MD5 via an online MD5 hashing site, or by running the following command in a Linux machine:

```
> echo -n You broke the Blocks | md5sum
```

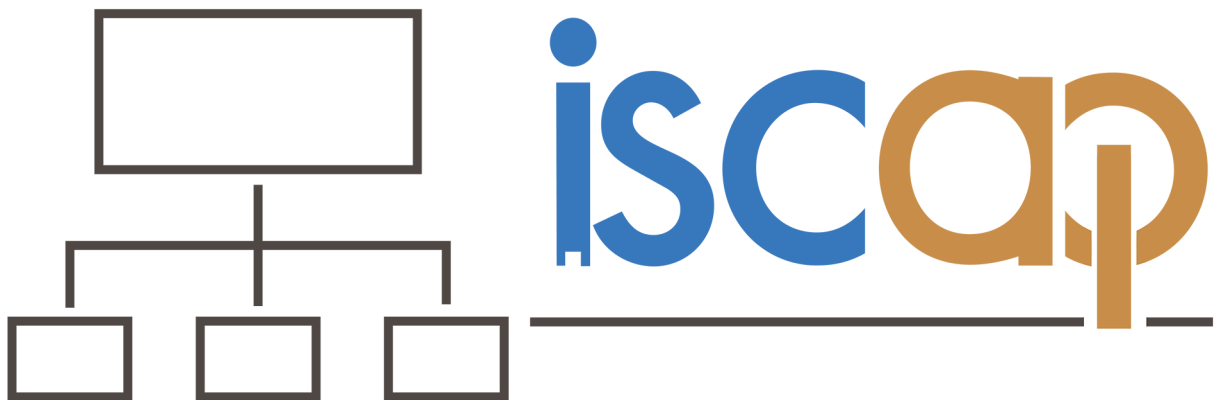
The final flag is:

```
🚩 b1a39cdaaa63c9fbf43be186fc5c21ca
```

**Appendix B. Survey Questions**

1. What is your major?
  - a) Accounting
  - b) Economics
  - c) Entrepreneurship
  - d) Finance
  - e) Hospitality Management
  - f) Information Security & Assurance
  - g) Information Systems
  - h) Management
  - i) Marketing
  - j) Professional Sales
2. The exercise was reasonable and useful.
  - a) Strongly disagree
  - b) Somewhat disagree
  - c) Neither agree nor disagree
  - d) Somewhat agree
  - e) Strongly agree
3. The teaching method through tutorial supported my learning process.
  - a) Strongly disagree
  - b) Somewhat disagree
  - c) Neither agree nor disagree
  - d) Somewhat agree
  - e) Strongly agree
4. Would you like to learn more about the applications of programming in cybersecurity after learning this tutorial?
  - a) Strongly disagree
  - b) Somewhat disagree
  - c) Neither agree nor disagree
  - d) Somewhat agree
  - e) Strongly agree
5. What do you like about this tutorial?
6. What do you dislike about this tutorial?

## INFORMATION SYSTEMS & COMPUTING ACADEMIC PROFESSIONALS



### STATEMENT OF PEER REVIEW INTEGRITY

All papers published in the *Journal of Information Systems Education* have undergone rigorous peer review. This includes an initial editor screening and double-blind refereeing by three or more expert referees.

Copyright ©2025 by the Information Systems & Computing Academic Professionals, Inc. (ISCAP). Permission to make digital or hard copies of all or part of this journal for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial use. All copies must bear this notice and full citation. Permission from the Editor is required to post to servers, redistribute to lists, or utilize in a for-profit or commercial use. Permission requests should be sent to the Editor-in-Chief, *Journal of Information Systems Education*, [editor@jise.org](mailto:editor@jise.org).

ISSN: 2574-3872 (Online) 1055-3096 (Print)