

**Algorithm Design and Analysis Project Report**

**Implementing a Simple Ceaser Cipher Encrypting Algorithm**

**Gresa Shala 192048697**

*\*University for Business and Technology, Prishtina 10000, Kosovo*

# **Introduction**

In today’s digital world, data protection is very important, even at the most basic level. Encryption is one way to protect information so that unauthorized people cannot access it. One of the oldest and simplest encryption methods is the Caesar Cipher. It works by shifting each letter in the text by a certain number of positions in the alphabet. For example, with a shift of 3, the letter 'A' becomes 'D', 'B' becomes 'E', and so on. This kind of encryption helps to hide the actual content of the message.

In this project, the Caesar Cipher is used to create a program that can secure the content of text files. The user can choose to encrypt or decrypt a file by giving a shift value. The program then reads the file, applies the shift, and saves the result to a new file. Both uppercase and lowercase letters are handled properly, while numbers, punctuation, and spaces remain unchanged

## **Motivation and relevance**

Even though the Caesar Cipher is not secure by modern standards, it is still a good way to understand the basic ideas behind encryption and algorithm design. It introduces key programming concepts like working with characters, using loops and conditionals, handling files, and applying mathematical operations such as modular arithmetic. Because of its simplicity, this method is commonly used in teaching and as a starting point for learning more advanced cryptographic algorithms. This project focuses on applying the Caesar Cipher to real text files, making it more useful than just a command-line demonstration.

## **Objectives**

The main goal of this project is to design and implement a working Caesar Cipher tool for text files. The objectives include:

* Creating a function that can both encrypt and decrypt text based on a given shift value.
* Handling file input and output, so users can easily apply encryption to any text file.
* Making sure the program works correctly with various types of input, including special characters and mixed-case text.
* Giving the user a simple menu to choose between encryption and decryption.

# **Literature Review**

The Caesar Cipher, also known as a shift cipher, is one of the earliest known and simplest classical encryption techniques. It is discussed in various cryptography and computer science textbooks as a foundational example of symmetric key encryption. One of the most commonly referenced sources is "Cryptography and Network Security: Principles and Practice" by William Stallings, which introduces the Caesar Cipher in the context of historical ciphers and basic cryptographic principles. The cipher is also described in "Introduction to Cryptography with Coding Theory" by Wade Trappe and Lawrence C. Washington, where it is used to illustrate the limitations of simple substitution techniques and the importance of keyspace size.

Academic discussion often highlights that the Caesar Cipher offers no real security in modern contexts due to its small keyspace (only 25 possible shifts). Nonetheless, it remains important for educational purposes and for introducing core concepts like modular arithmetic, frequency analysis, and brute-force attacks. Research papers on classical ciphers often use the Caesar Cipher to explain these basic ideas before moving on to stronger ciphers like the Vigenère Cipher or AES.

## **Gap identification**

Most academic texts and papers discuss the Caesar Cipher in theory or apply it to single-line input from users. They rarely focus on practical applications such as encrypting full text files or handling real-world character sets, such as preserving punctuation, digits, and file formatting. This project addresses that gap by providing a working implementation that supports file-based encryption and decryption, offering a more realistic scenario for applying the Caesar Cipher algorithm.

# **Problem Definition and Formulation**

Given a plaintext string and a shift value sss, return the encrypted version of the string by replacing each letter with the one sss positions ahead in the alphabet. For decryption, shift the characters sss positions backward. Handle uppercase and lowercase letters, and leave non-letter characters unchanged.

Input:

* A text file containing the plaintext (for encryption) or ciphertext (for decryption).
* An integer shift value.

Output:

* A new file containing the transformed text.

## **Assumptions and Constraints**

* Only alphabetic characters are shifted.
* File size is manageable (no large-scale data handling).
* Shift values can be positive or negative.

# **Algorithm Design**

The Caesar Cipher is a simple substitution algorithm that works by shifting each letter of the input text by a fixed number of positions in the alphabet. The shifting is circular—if the letter goes past 'Z' or 'z', it wraps around to the beginning of the alphabet.

The algorithm:

* Reads each character from the text.
* If the character is a letter (A–Z or a–z), it is shifted by a fixed number using modular arithmetic.
* Non-letter characters (like spaces, digits, punctuation) are left unchanged.
* The result is added to the output string.

This process is the same for both encryption and decryption, except:

* **Encryption** shifts the character forward.
* **Decryption** shifts the character backward.

Pseudocode:

*function caesar\_cipher(text, shift, mode):*

*result = ""*

*for each character in text:*

*if character is a letter:*

*base = 'A' if character is uppercase else 'a'*

*if mode == "encrypt":*

*new\_char = (ord(character) - ord(base) + shift) % 26 + ord(base)*

*else if mode == "decrypt":*

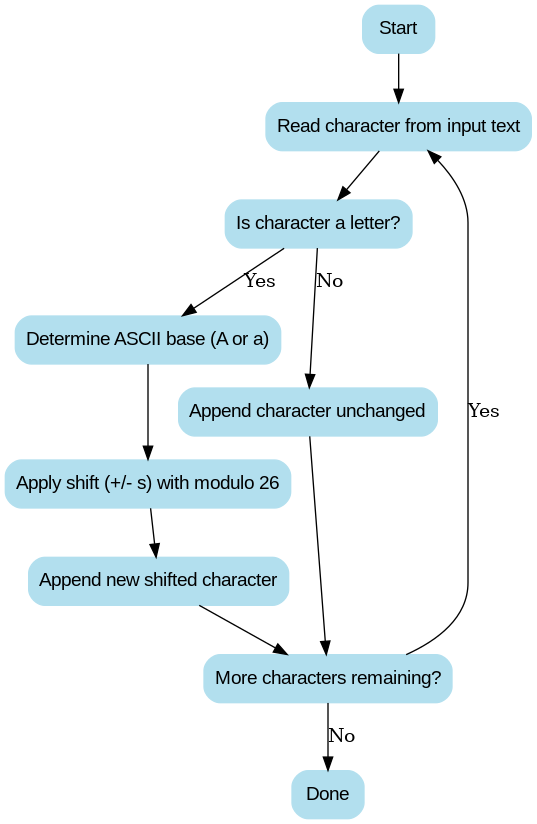
*new\_char = (ord(character) - ord(base) - shift) % 26 + ord(base)*

*result += chr(new\_char)*

*else:*

*result += character*

*return result*

**

*Figure 1 Flow chart*

## **Algorithm Choice and Justification**

I chose the Caesar Cipher due to its simplicity and educational value. It is easy to implement and understand, yet it demonstrates key algorithm concepts:

* Character-level manipulation
* Use of ASCII values and modular arithmetic
* Condition-based processing of inputs
* Linear time complexity O(n)

This algorithm is ideal for a beginner-friendly encryption tool and fits the scope of file-based text processing. It also helps illustrate the difference between secure and insecure encryption schemes.

## **Optimality Considerations**

The algorithm is optimal for its purpose: simple encryption with minimal computation. It runs in linear time with respect to input size, which is the best possible for this type of transformation.

For real-world secure applications, it is not suitable due to the limited keyspace (only 25 possible shifts) and ease of brute-force attacks or frequency analysis.

# **Complexity Analysis**

## **Time Complexity**

The time complexity of the Caesar Cipher algorithm is O(n), where n is the number of characters in the input text. This is because the algorithm processes each character in the input exactly once, regardless of whether it is a letter or not.

For each character, the algorithm performs:

* A check to see if it's an alphabetic character (char.isalpha()), which takes constant time O(1).
* If it is a letter, it computes a shifted character using basic arithmetic operations (subtraction, addition, modulo), all of which also take O(1) time.
* If it's not a letter, the character is added to the result as-is, which again takes constant time.

Since these constant-time operations are done once per character in the input, the overall runtime grows linearly with the input size.

## **Space Complexity**

The space complexity is also O(n), where n is the length of the input text. This is because:

* A new string is created to store the result (encrypted or decrypted text), which requires space proportional to the input.
* The algorithm does not use any additional data structures like arrays or hash maps that scale with input size.
* File reading (read() method) loads the full content of the file into memory, which also contributes to the O(n) space usage.

Therefore, both time and space complexity are linear with respect to the number of characters in the text being processed.

## **Comparative Analysis**

The Caesar Cipher is a monoalphabetic substitution cipher, meaning each letter in the plaintext is replaced with another fixed letter from the alphabet based on a constant shift. This makes it extremely efficient but also extremely easy to break. Compared to more modern cryptographic algorithms, it has several major differences in both complexity and security:

### **Execution Time and Efficiency**

The Caesar Cipher has O(n) time complexity and runs in a single pass over the input text. Modern encryption algorithms like AES (Advanced Encryption Standard) and RSA are more complex. AES, for example, performs multiple rounds of substitution, permutation, and key mixing, resulting in higher computational cost-though it is still highly optimized and fast in practice. RSA, being an asymmetric algorithm, involves expensive operations like modular exponentiation and has much higher time complexity, often O(n¬≥) for naive implementations.

### **Space Requirements**

Caesar Cipher requires O(n) space, mainly to store the output. AES uses additional memory for round keys and internal state arrays. RSA may also require significant space for large integer representations, especially with key sizes of 1024 bits or more.

### **Keyspace and Security**

Caesar Cipher has a tiny keyspace: only 25 valid shifts (since a shift of 0 results in the original text). This makes brute-force attacks trivial-an attacker can try all shifts in seconds. In contrast, AES supports key sizes of 128, 192, and 256 bits, which means 2¬π¬≤‚Å∏ to 2¬≤‚Åµ‚Å∂ possible keys, making brute-force infeasible with current technology. RSA offers similar high-level security due to the difficulty of factoring large primes.

### **Use Cases**

Caesar Cipher is educational and used for teaching basic encryption and algorithm design.

AES is widely used in secure data transmission, including HTTPS, file encryption, and VPNs.

RSA is typically used for secure key exchange and digital signatures, not bulk data encryption.

### **Cryptanalysis Resistance**

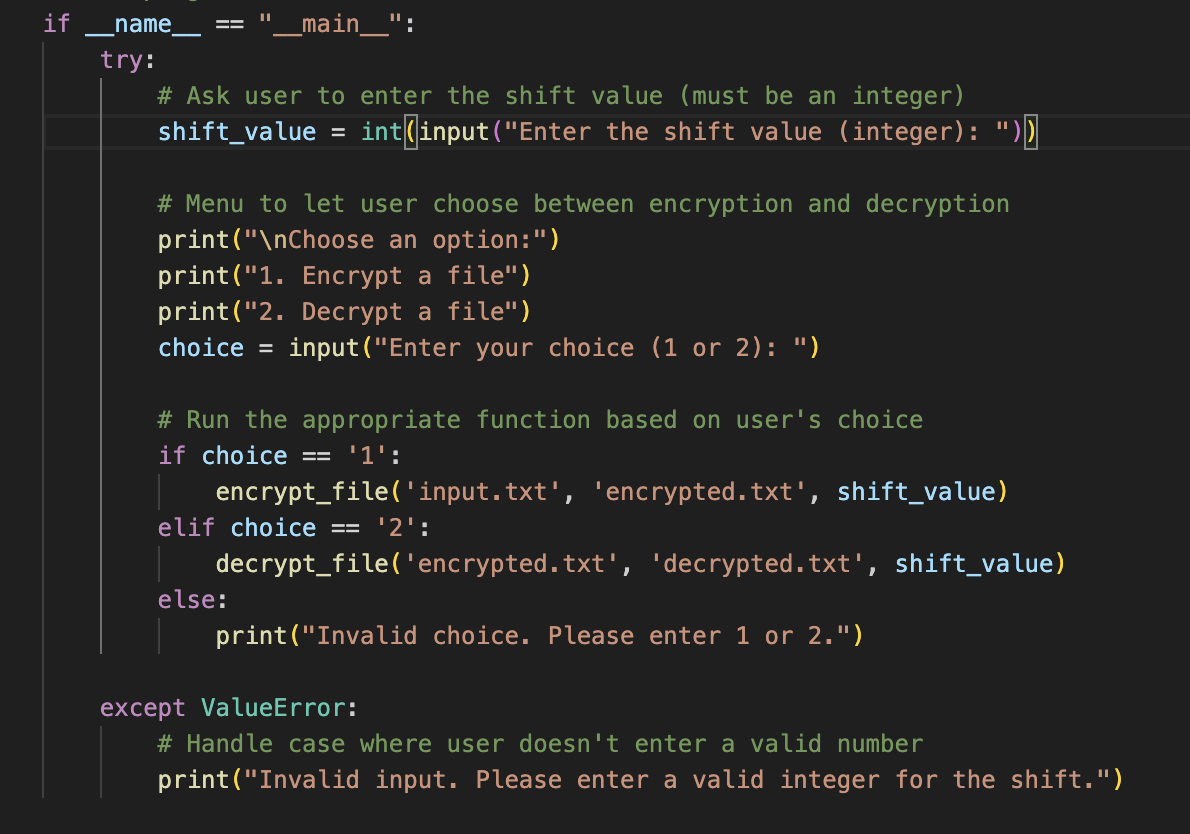
The Caesar Cipher is highly vulnerable to frequency analysis, since the letter relationships remain unchanged. More advanced ciphers like AES use complex transformations that completely obscure frequency patterns.

In summary, while the Caesar Cipher is extremely simple and fast, it is not suitable for any real-world security application. However, it is a valuable learning tool for introducing fundamental algorithm concepts and string manipulation techniques.

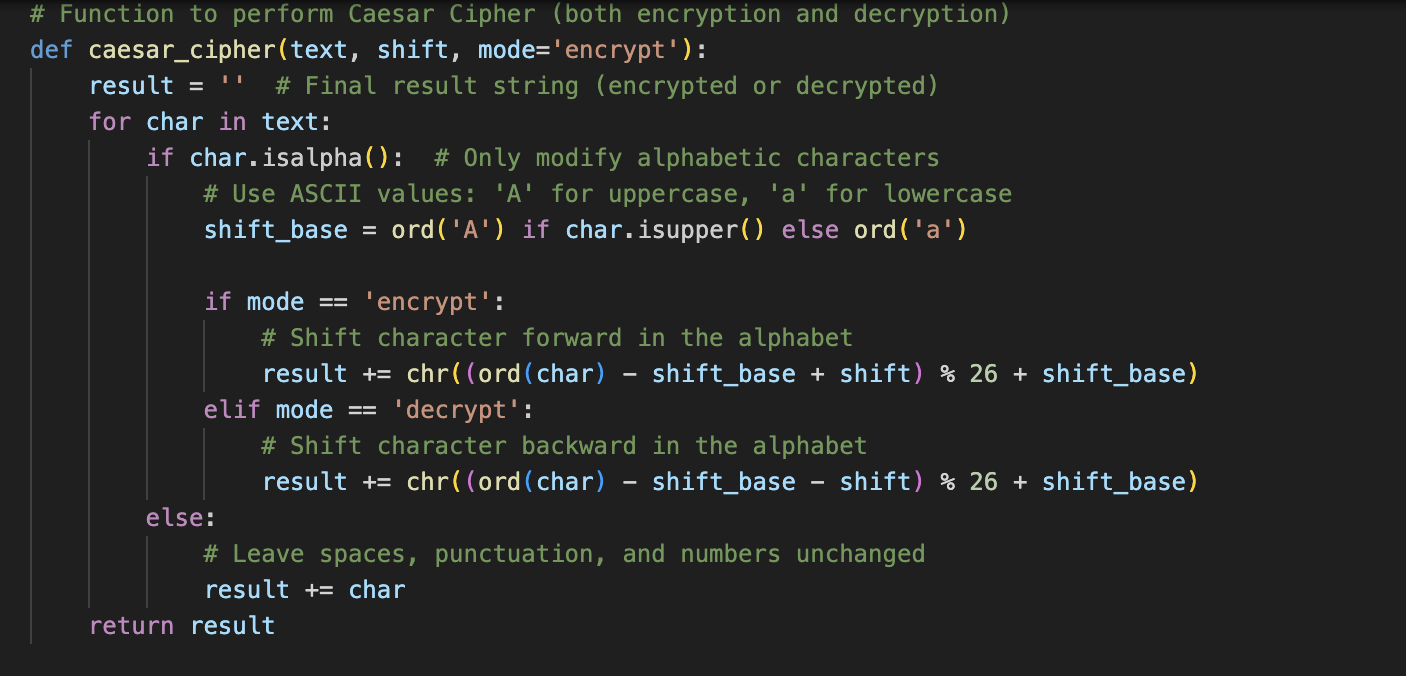
# **Implementation**

This implementation uses Python, a high-level language that is well-suited for text manipulation due to its straightforward syntax and powerful built-in functions. Because it only relies on the standard library, no additional installations are required beyond Python itself. The code can be run on any operating system that supports Python 3.

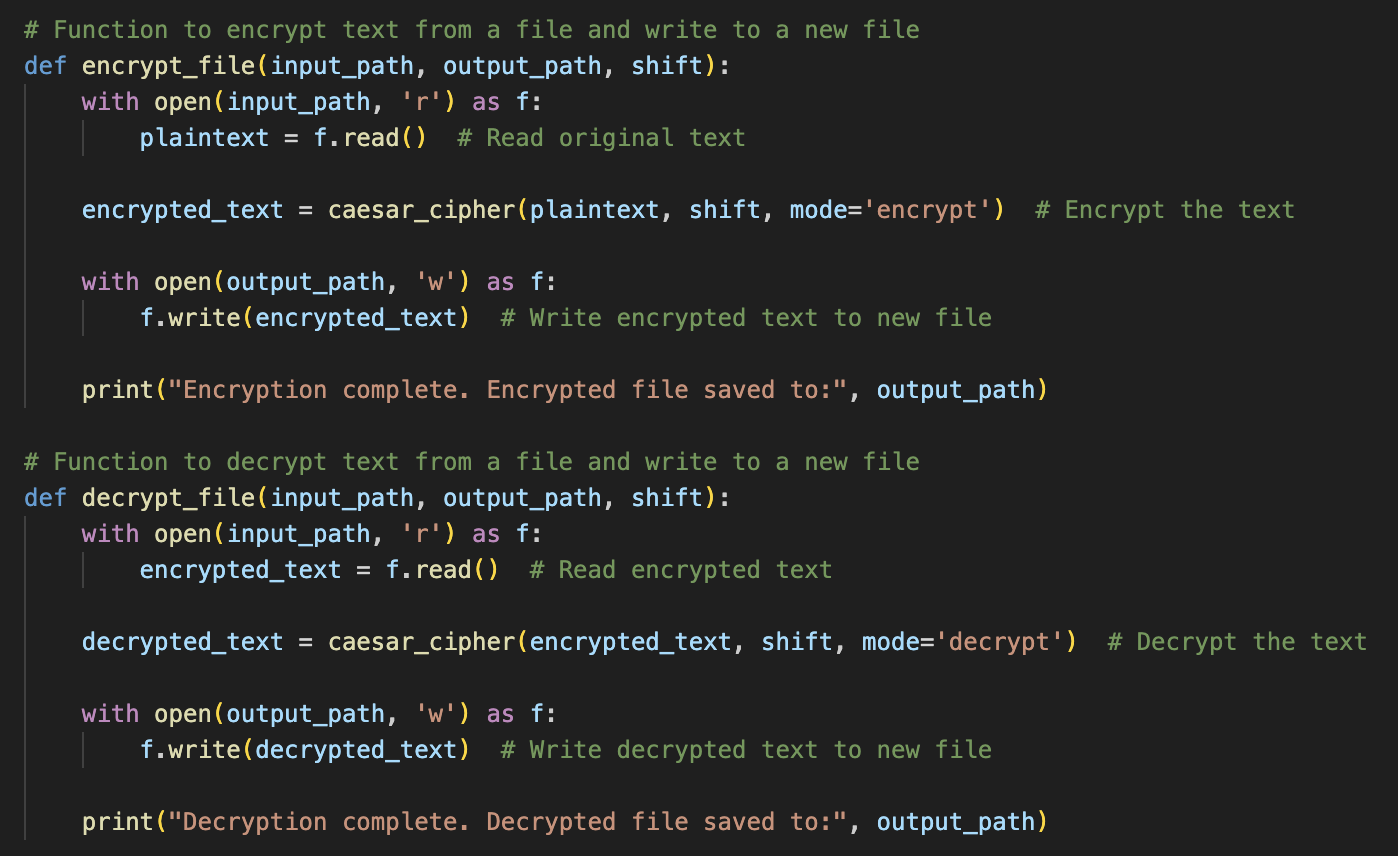
The program begins by prompting the user for a shift value and an action—either to encrypt or decrypt a file. It relies on a function called caesar\_cipher to perform the actual letter shifts, ensuring that only alphabetic characters are transformed. The flow is divided into three main parts: reading user input, calling the appropriate encryption or decryption function on a file, and then writing the results back to a new file. Here, an integer shift value is requested from the user. If the input cannot be converted to an integer, the program displays an error message without crashing. A menu then appears, letting you choose whether to run the encryption or decryption process. Depending on your selection, the program calls either encrypt\_file or decrypt\_file.



The ***caesar\_cipher*** function is at the heart of this program. It takes a string of text, along with a shift value and a mode (either “encrypt” or “decrypt”). It loops through each character in the text and applies the shift only to alphabetic characters. Uppercase letters use A as a reference point, while lowercase letters use a. Non-alphabetic characters, such as digits or punctuation, remain unchanged. By subtracting or adding the shift and then using the modulo operation, letters wrap around the alphabet if the shift moves beyond z or before a. This ensures that characters stay within the correct range of uppercase or lowercase letters.



To encrypt or decrypt a file, the program reads the entire contents of the file as a single string, passes it to caesar\_cipher with the chosen shift value and mode, then writes the transformed text to a new file. This approach helps keep the logic clean by separating the code that manages file reading and writing from the logic that actually shifts the letters. These two functions illustrate how the Caesar Cipher is integrated into a real-world file-based workflow. They simply read text from one file, process it, and write the output to a second file. The program also prints a confirmation message so you know the operation completed successfully.

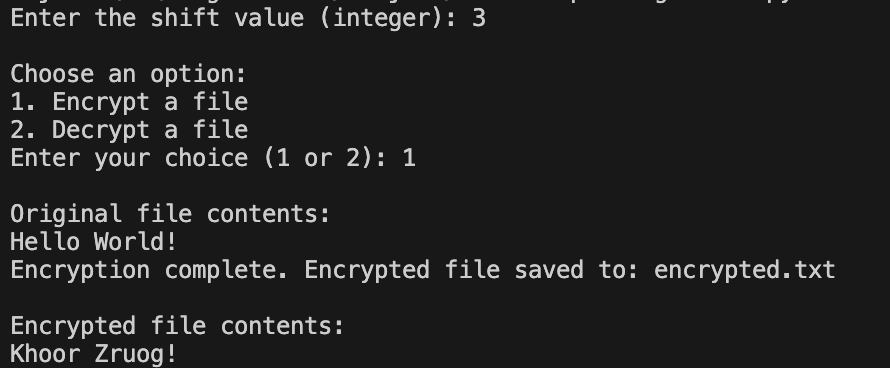


## **Test Cases and Validation**

Validation of the Caesar Cipher algorithm involved running multiple tests on a variety of inputs to ensure both correctness.

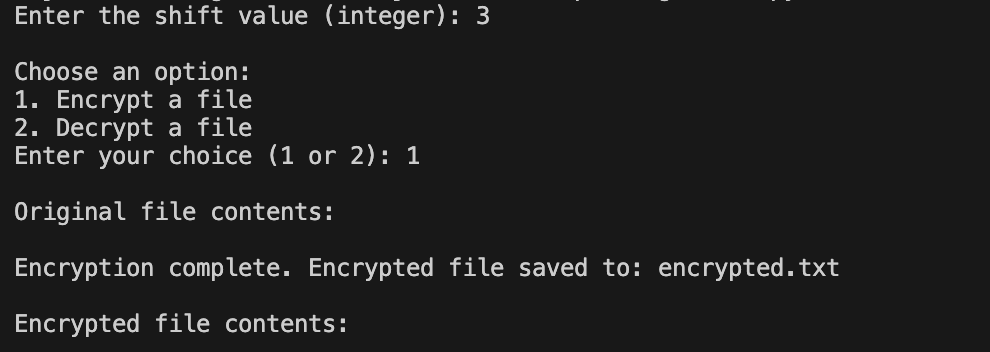
### **Typical text strings**

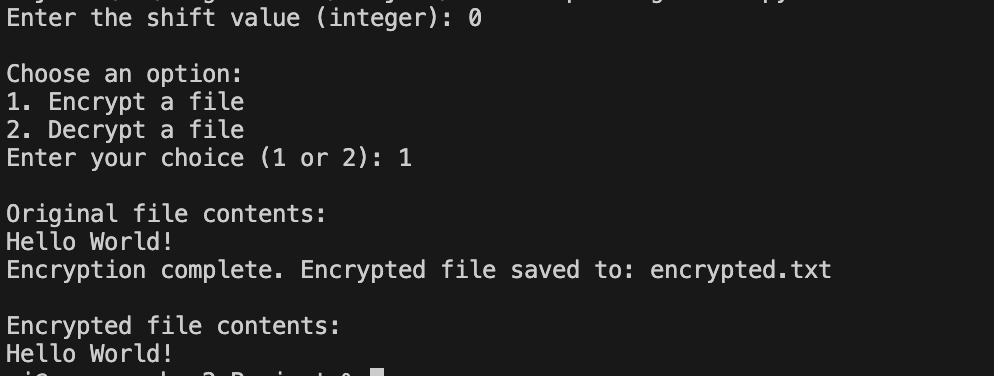
Validation of the Caesar Cipher algorithm began with straightforward tests on typical text strings to verify its basic correctness. For instance, encrypting “Hello, World!” with a shift of 3 gave “Khoor, Zruog!”, which decrypted back to “Hello, World!” when run in reverse. This confirmed that the shifting logic handled normal alphabetic characters correctly and that punctuation remained untouched.



### **Boundary conditions and edge cases**

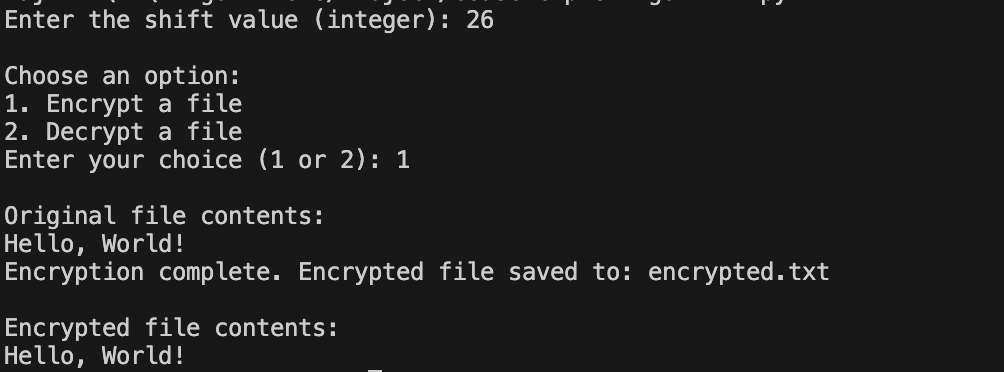
Next, boundary conditions and edge cases were tested. An empty file check ensured that the implementation gracefully handled a situation with no data, producing an empty output without errors. A shift value of zero also behaved as expected, leaving the text entirely unchanged. This validated that the program could handle edge cases without failing or corrupting data.

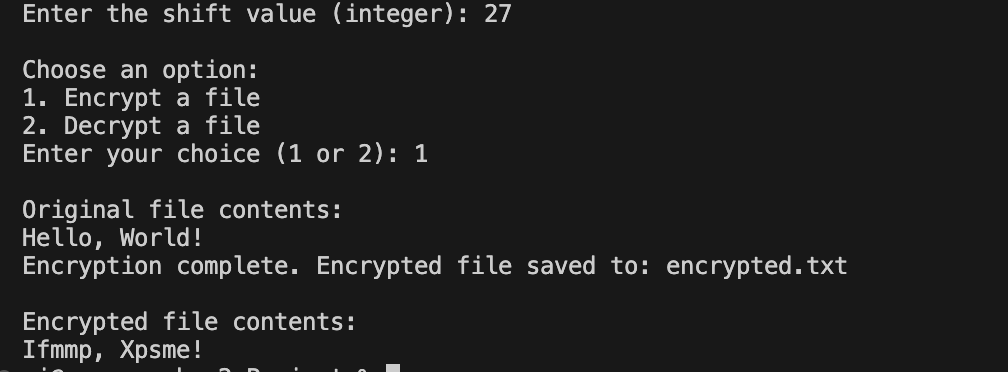


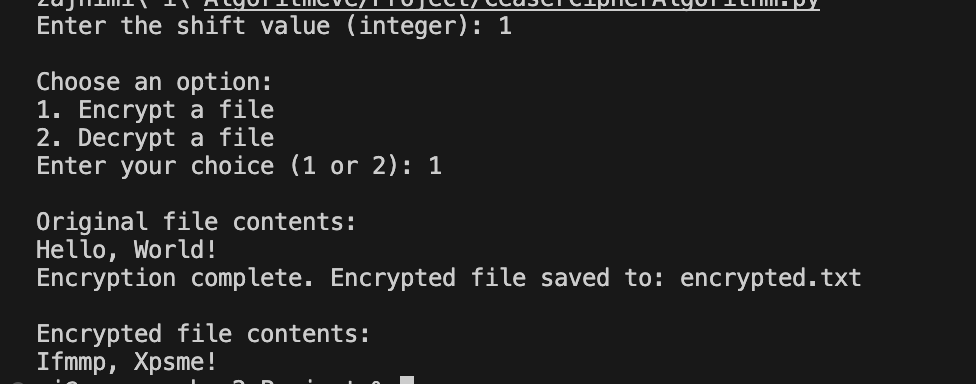


### **Large shifts**

Finally, larger shift values were used to confirm correct wrapping behavior. Shifting “Hello, World!” by 26 letters yielded the same string, while shifting by 27 letters acted like a shift of 1 and produced “Ifmmp, Xpsme!”. Special characters and numbers remained unaltered, demonstrating the code’s selective approach to shifting only alphabetic characters. Altogether, these tests showed that the Caesar Cipher implementation performed reliably across normal usage, boundary conditions, and special character scenarios.







# **Experimental Results**

# **Conclusion**

# **References**

 Stallings, W. (2016). *Cryptography and Network Security: Principles and Practice* (7th ed.). Pearson.

 Trappe, W., & Washington, L. C. (2006). *Introduction to Cryptography with Coding Theory* (2nd ed.). Pearson.

 Singh, S. (1999). *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography*. Anchor Books.