



Programación

Tarea Semestral

Encriptando texto usando sustitución binaria

503203-0

Integrantes:

- Benjamín Junemann (2024436813)
- Pablo Nahuelquín (2024407686)
- Erick Raasch (2024443267)
- Terence O'Mahony (2024457225)

Carreras respectivamente:

- Ingeniería Civil Matemática
- Ingeniería Civil Matemática
- Ingeniería Civil Matemática
- Ingeniería Civil Matemática

Sección: 0

Índice

1. Introducción	2
2. Organización	3
2.1. Roles	3
2.2. Reuniones	3
3. Desarrollo	5
3.1. Análisis del Problema	5
3.2. Estrategias de solución	6
3.3. Implementación de la solución	7
3.3.1. Estructuras de datos	7
3.3.2. Validación de entradas	7
3.3.3. Conversión entre decimal y binario	7
3.3.4. Transformación de tablas	7
3.3.5. Función principal	8
3.4. Pruebas	9
3.4.1. Prueba 1.	9
3.4.2. Prueba 2.	9
3.4.3. Prueba 3.	10
3.4.4. Prueba 4.	11
4. Conclusión	12

1. Introducción

El presente trabajo tiene como objetivo desarrollar un algoritmo en el conocido lenguaje de programación Python, el cual sea capaz de codificar y decodificar mensajes utilizando el método de sustitución binaria descrito en el enunciado. Este método implica la transformación de cada símbolo de un mensaje en su representación binaria, seguida de la inversión de 0s por 1s y viceversa, para luego obtener un mensaje codificado. Para la decodificación, se sigue el proceso inverso.

El desarrollo de este proyecto no solo busca cumplir con los requisitos técnicos establecidos, sino también incentivar el trabajo en equipo, la planificación colaborativa y la aplicación de conceptos de programación en un problema de complejidad intermedia-alta. En este informe, se detallarán las etapas clave del proceso, como el análisis del problema, la estrategia de solución, la implementación de algoritmos y estructuras de datos, así como las pruebas realizadas para verificar el correcto funcionamiento de nuestro código.

Para aprovechar mejor nuestro tiempo, a cada integrante se le asignó una función para un trabajo óptimo y eficiente, estas serán detalladas a lo largo de este informe.

El objetivo principal es crear un algoritmo, usando Python, que implemente una solución al problema propuesto de forma rápida y efectiva. Adicionalmente, buscamos aprender a trabajar en equipo y utilizar el tiempo como aliado, dejando de lado el orgullo, el ego y las diferencias de cada uno, para así llegar a un código correcto, trabajando de manera óptima para un buen resultado final.

2. Organización

A continuación, se presentan las funciones que cumplieron cada uno de los integrantes, esto es, las responsabilidades que cada uno debió cumplir y un breve resumen de ellas. A su vez, también se presenta a modo de transparencia, la cantidad de veces que nos reunimos como grupo, cómo y con qué fin nos reunimos, los resultados obtenidos por dichas reuniones y los asistentes.

2.1. Roles

- **Análisis del problema:** Todos los integrantes del grupo participaron en este aspecto. Todos los integrantes fueron partícipes en el análisis del problema, dando distintas perspectivas sobre cómo abordar el problema, al igual que buscar relaciones y patrones con respecto a problemas vistos anteriormente en el curso.
- **Solución al problema:** Todos los integrantes del grupo participaron en este aspecto. Todos los miembros del grupo estuvieron presentes para plantear distintas sugerencias de ideas con respecto a la solución del problema, además de llegar a la solución más correcta utilizando aspectos vistos en el curso de programación.
- **Implementación en código:** Terence O'Mahony, Benjamín Junemann. Ambos integrantes se encargaron de desarrollar y finalizar el código. Crearon las funciones principales y reorganizaron el programa para que fuera claro y comprensible.
- **Realización del Informe:** Erick Raasch, Pablo Nahuelquín. La principal función que se les asignó y que llevaron a cabo fue la correcta organización y distribución del informe escrito, el cual fue desarrollado utilizando el sistema de composición de textos L^AT_EX. Se buscó mantener la prolijidad y cumplir con cada uno de los requisitos establecidos en la pauta, además de lograr una redacción coherente que permitiera traspasar adecuadamente todo el análisis del código al lenguaje escrito.

2.2. Reuniones

- **Reunión 1**

Fecha: 20/05.

Integrantes presentes: Todos los integrantes.

Medio usado: Presencial, ludoteca FCFM.

Objetivo: Dar inicio a la tarea definiendo los roles para cada integrante y realizar lluvia de ideas para la primera parte del proyecto.

Resultados: Además de la organización se logró una comprensión sobre cómo transformar un texto en código binario, lo cual sentó las bases para el desarrollo de la siguiente parte del proyecto.

■ Reunión 2

Fecha: 27/05.

Integrantes presentes: Todos los integrantes.

Medio usado: Presencial, departamento de Terence O'Mahony.

Objetivo: Analizar el problema, plantear una solución y empezar a implementarla en Python.

Resultados: Se logró un entendimiento claro e intuitivo del problema, permitiendo definir un enfoque eficaz para su solución. Se implementó con éxito la funcionalidad básica de conversión entre texto y binario, y viceversa.

■ Reunión 3

Fecha: 03/06.

Integrantes presentes: Todos los integrantes

Medio usado: Virtual, vía Discord.

Objetivo: Explorar distintas estrategias para abordar la siguiente etapa del problema, centrándonos especialmente en la tabla y su mecanismo de sustitución.

Resultados: El principal progreso fue la adopción de los diccionarios como herramienta clave para la implementación del algoritmo. Además, se empezó a reescribir y depurar algunas secciones del código inicial que habíamos desarrollado como boceto en las primeras dos reuniones.

■ Reunión 4

Fecha: 06/06.

Integrantes presentes: Todos los integrantes

Medio usado: Presencial, departamento de Terence O'Mahony.

Objetivo: Formalizar funciones relacionadas con el uso de listas para conseguir una versión funcional sobre el programa, misma que será verificada con una serie de pruebas que confirmen su efectividad.

Resultados: Esta reunión representó uno de los avances más significativos, ya que se finalizó una versión funcional del código. No obstante, también reconocimos la necesidad de mejorar la organización y presentación del programa para asegurar una mejor comprensión.

■ Reunión 5

Fecha: 07/06.

Integrantes presentes: Todos los integrantes

Medio usado: Presencial, departamento de Terence O'Mahony.

Objetivo: Mejorar y organizar el código, con el fin de ser lo más óptimo y comprensible posible, y dar inicio a la redacción del informe escrito.

Resultados: Se realizó un importante trabajo de reorganización del código y realización del informe. Esto resultó ser el paso final sobre la tarea semestral.

3. Desarrollo

3.1. Análisis del Problema

Comprender a fondo el problema es fundamental para diseñar una solución adecuada. En este caso, el objetivo es construir un programa en Python que permita al usuario codificar o decodificar un mensaje, según así lo quiera, utilizando una tabla de sustitución dinámica y una operación de inversión de bits sobre los valores numéricos asociados a los caracteres.

El conjunto de caracteres válidos está compuesto por 64 símbolos, los cuales son letras mayúsculas y minúsculas del alfabeto inglés, dígitos del 0 al 9, guion y espacio. Cada símbolo se asocia a una posición única dentro de una tabla base. Esta tabla puede rotarse para generar una nueva secuencia de sustitución, lo que establece un mecanismo de codificación adaptativo en el proceso de codificación. La posición inicial de la rotación depende de la instrucción dada por el usuario, si se trata de codificar, se rota desde el último carácter, y si se quiere decodificar, se rota desde el carácter que sigue a ese último.

Primeramente, el programa debe validar que el mensaje ingresado por el usuario sea válido. Esta validación contempla únicamente dos opciones: CODIFICAR y DECODIFICAR. Cualquier otro término deberá ser rechazado y el usuario deberá ingresar un nuevo mensaje. Luego, se valida el mensaje ingresado, el cual no debe superar los 1000 caracteres. Esta restricción garantiza que el programa no se vea afectado por entradas demasiado largas que afecten su correcto funcionamiento.

Una vez completadas estas validaciones, comienza el proceso de transformación del mensaje. Cada carácter se traduce a su posición numérica dentro de la tabla rotada, ese valor numérico se convierte a su forma binaria, y luego se invierten los bits. La inversión de bits implica cambiar cada 0 por un 1, y cada 1 por un 0, generando así una variación del patrón original. Esta transformación binaria es crucial, ya que representa el mecanismo central del cifrado.

En nuestro esquema de cifrado, tras procesar una palabra codificada, su último carácter siempre proviene del índice 63 de la tabla. Ese índice 63 no es arbitrario: en la tabla original se reservó para la letra que va justo antes de la que se utilizó como punto de rotación. Al asegurar que ese último símbolo siempre esté en la posición 63, se garantiza un marcador único y constante que permite identificar fácilmente el carácter base de rotación durante la decodificación, sin necesidad de transmitir información adicional. Por ello, basta con leer el último símbolo del mensaje cifrado para descubrir automáticamente cuál fue la letra de rotación; con esa información podemos reconstruir la tabla original y revertir todo el proceso para descifrar correctamente el texto.

En términos generales, el análisis del problema muestra que se trata de un sistema de cifrado que requiere aplicar validaciones claras, transformar datos numéricos y utilizar una tabla rotada para lograr un proceso de codificación y decodificación exitoso.

3.2. Estrategias de solución

A partir del análisis del problema previamente hecho, la solución se organiza en los siguientes pasos:

1. **Definición de estructuras:** Se cuenta con una lista que contiene 64 caracteres válidos y un diccionario que asigna cada carácter a un número. Estas estructuras están predefinidas en el programa y permiten consultar rápidamente las posiciones necesarias para codificar o decodificar según sea necesario.
2. **Ingreso y validación del modo de operación:** Se solicita al usuario que indique si desea CODIFICAR o DECODIFICAR. El programa no continúa hasta que se reciba una entrada válida.
3. **Ingreso y validación del mensaje:** Se solicita al usuario que ingrese una cadena de texto. Esta se verifica para asegurar que solo contenga caracteres permitidos y no supere el límite máximo de longitud. En caso de error, se vuelve a pedir que se ingrese un mensaje.
4. **Identificación del punto de rotación:** Según el modo seleccionado, se obtiene el último carácter del mensaje y se calcula su posición. En el caso de la decodificación, se toma el carácter siguiente como punto de inicio para la rotación de la tabla por lo discutido en Análisis del Problema.
5. **Rotación de estructuras:** A partir del punto de rotación determinado, se reorganizan tanto la lista de caracteres como el diccionario, generando nuevas versiones que se usan para el procedimiento del mensaje.
6. **Transformación binaria:**
 - Se recorre el mensaje carácter por carácter.
 - Se busca el índice correspondiente en la tabla rotada.
 - Se convierte el índice en binario.
 - Se invierten los bits.
 - Se convierte el binario invertido de nuevo a un número.
7. **Reconstrucción del mensaje:** Con los nuevos índices obtenidos tras la inversión, se accede a la tabla rotada para obtener los caracteres resultantes y se construye un nuevo mensaje transformado según la operación seleccionada.
8. **Entrega del resultado:** El programa muestra por pantalla el mensaje final, indicando si corresponde a una codificación o a una decodificación.

Cada uno de estos pasos se encuentra implementado en funciones dentro del programa, lo que permite mantener la organización y comprensión del programa.

3.3. Implementación de la solución

A continuación explicaremos cómo todas las ideas desarrolladas en Estrategias de solución fueron implementadas en Python.

3.3.1. Estructuras de datos

Para almacenar la información de las Tablas de codificación o decodificación se emplean dos estructuras principales:

- **Listas:** Se utiliza la lista `vec` la cual asocia a cada posición una letra de la tabla.
- **Diccionarios:** Usamos el diccionario `dic` el cual a cada letra de la tabla le asocia un número del 0 al 63.

3.3.2. Validación de entradas

Para el ingreso de los datos y verificar que estos sean correctos, usaremos las siguientes funciones:

- **validar1:** Es la función encargada de recibir la cadena de texto y verificar que esta cumpla los requisitos, es decir que su largo no sea mayor a 1000 y que solo incluya las letras consideradas en la tabla.
- **validar2:** Es la función encargada de recibir y verificar que el modo sea uno de los 2 modos permitidos (codificar o decodificar).

3.3.3. Conversión entre decimal y binario

Como vimos en Análisis del Problema el trabajo con números binarios será parte importante del programa, en particular con números binarios de hasta 6 bits. Las siguientes funciones son las encargadas de hacer las tareas requeridas por el algoritmo:

- **num2bin:** Convierte un número entero menor o igual a 63 en su representación binaria.
- **bin2num:** Convierte un binario de seis dígitos a su valor decimal.
- **not_bin:** Transforma un binario en su complemento, es decir invierte cada bit de la cadena ($0 \rightarrow 1$, $1 \rightarrow 0$).

3.3.4. Transformación de tablas

Una de las partes más importantes del programa son las rotaciones efectuadas a las tablas de correspondencia, las siguientes funciones se encargan de hacer estas rotaciones dado un valor de rotación previamente calculado y la representación de la tabla original.

- **tdic:** Genera un nuevo diccionario con los índices desplazados por un valor dado.
- **tvec:** Rota la lista de símbolos para que comience en la posición indicada.

3.3.5. Función principal

La función `main` es la encargada de usar todas las funciones definidas anteriormente para ejecutar el programa, como sigue:

1. `vec` y `dic` son la lista y el diccionario respectivamente, encargados de almacenar la información de la tabla.
2. Posteriormente se leen y validan los datos del **modo** y de la **cadena de texto** usando las funciones `validar2` y `validar1` respectivamente, almacenando lo obtenido en las variables `modo` y `original`.
3. Se almacena la última letra de la cadena en `ultletra` y luego se calcula el número de desplazamiento usando la tabla según corresponda al modo, almacenamos esto en `numero`.
4. Rotamos la tabla usando lo obtenido en `numero`, es decir aplicamos las funciones `tdic` y `tvec` a `dic` y `vec` respectivamente, almacenando los resultados en `nuevo_dic` y `nuevo_vec`.
5. Calculamos de forma iterativa cada número asociado a las letras de la cadena usando `nuevo_dic` y posteriormente calculamos la representación binaria de cada número usando `num2bin` mostramos lo obtenido pantalla y lo almacenamos en la lista `binarios`.
6. De manera iterativa calculamos el complemento de los binarios obtenidos usando `not_bin`, luego transformamos estos binarios a decimal usando `bin2num` y con lo obtenido usamos `nuevo_vec` para obtener las letras asociadas según corresponda, almacenamos estas en `solucion`.
7. Mostramos lo obtenido en `solucion` por pantalla.

3.4. Pruebas

A continuación haremos pruebas a nuestro código, para verificar su funcionamiento en casos de interés.

3.4.1. Prueba 1.

Probaremos con el ejemplo propuesto en el documento de las instrucciones para realizar este informe:

```
{Desea CODIFICAR o DECODIFICAR?: CODIFICAR
Ingrese la cadena: This homework is very EASY

CODIFICAR
This homework is very EASY

59 (111011) 9 (001001) 10 (001010) 20 (010100) 39 (100111) 9 (001001)
16 (010000) 14 (001110) 6 (000110) 24 (011000) 16 (010000) 19 (010011)
12 (001100) 39 (100111) 10 (001010) 20 (010100) 39 (100111) 23 (010111)
6 (000110) 19 (010011) 26 (011010) 39 (100111) 44 (101100) 40 (101000)
58 (111010) 0 (000000)

4 (000100) 54 (110110) 53 (110101) 43 (101011) 24 (011000) 54 (110110)
47 (101111) 49 (110001) 57 (111001) 39 (100111) 47 (101111) 44 (101100)
51 (110011) 24 (011000) 53 (110101) 43 (101011) 24 (011000) 40 (101000)
57 (111001) 44 (101100) 37 (100101) 24 (011000) 19 (010011) 23 (010111)
5 (000101) 63 (111111)

CONDwOHJR HELwNDwARE9wrvdX
```

Notamos que hemos obtenido un resultado ligeramente distinto al esperado; sin embargo, creemos firmemente que esto se debe a un error en el documento y no en nuestro código.

3.4.2. Prueba 2.

Buscaremos probar nuestras funciones de validación; para esto, ingresaremos datos erróneos al programa, obteniendo lo siguiente:

```
{Desea CODIFICAR o DECODIFICAR?: Malinput
Ingrese un comando no conocido, intente denuevo: CODIFICAR2
Ingrese un comando no conocido, intente denuevo: CODIFICAR
Ingrese la cadena: ñññ
Error en la cadena ingresada, ingrese denuevo: !!!
Error en la cadena ingresada, ingrese denuevo: Soylaprueba2

CODIFICAR
Soylaprueba2

28 (011100) 50 (110010) 60 (111100) 47 (101111) 36 (100100) 51 (110011)
53 (110101) 56 (111000) 40 (101000) 37 (100101) 36 (100100) 0 (000000)

35 (100011) 13 (001101) 3 (000011) 16 (010000) 27 (011011) 12 (001100)
10 (001010) 7 (000111) 23 (010111) 26 (011010) 27 (011011) 63 (111111)

ZD5GRCA9NQR1
```

De esta prueba concluimos que nuestras funciones de validación funcionan correctamente.

3.4.3. Prueba 3.

Con el fin de demostrar la consistencia de nuestro programa, codificaremos un mensaje y a continuación lo decodificaremos para verificar que recuperamos exactamente el texto original:

Prueba 3.1

```
¿Desea CODIFICAR o DECODIFICAR?: CODIFICAR
```

```
Ingrese la cadena: Soylaprueba3
```

```
CODIFICAR
```

```
Soylaprueba3
```

```
27 (011011) 49 (110001) 59 (111011) 46 (101110) 35 (100011) 50 (110010)  
52 (110100) 55 (110111) 39 (100111) 36 (100100) 35 (100011) 0 (000000)
```

```
36 (100100) 14 (001110) 4 (000100) 17 (010001) 28 (011100) 13 (001101)  
11 (001011) 8 (001000) 24 (011000) 27 (011011) 28 (011100) 63 (111111)
```

```
bF7ITEC PST2
```

De acá obtenemos la cadena codificada **bF7ITEC PST2** la cual usaremos como entrada de la siguiente prueba, para verificar que obtenemos la cadena ingresada originalmente.

Prueba 3.2

```
¿Desea CODIFICAR o DECODIFICAR?: DECODIFICAR
```

```
Ingrese la cadena: bF7ITEC PST2
```

```
DECODIFICAR
```

```
bF7ITEC PST2
```

```
36 (100100) 14 (001110) 4 (000100) 17 (010001) 28 (011100) 13 (001101)  
11 (001011) 8 (001000) 24 (011000) 27 (011011) 28 (011100) 63 (111111)
```

```
27 (011011) 49 (110001) 59 (111011) 46 (101110) 35 (100011) 50 (110010)  
52 (110100) 55 (110111) 39 (100111) 36 (100100) 35 (100011) 0 (000000)
```

```
Soylaprueba3
```

Dado que obtuvimos la cadena ingresada originalmente, hemos mostrado que nuestro programa es consistente.

3.4.4. Prueba 4.

Para esta última prueba, usaremos una entrada con un texto más extenso extraído de una canción.

```
{Desea CODIFICAR o DECODIFICAR?: CODIFICAR
Ingrese la cadena: Blackbird singing in the dead of night Take these broken wings and learn to fly All your life You were only waiting for this moment
to arise

CODIFICAR
Blackbird singing in the dead of night Take these broken wings and learn to fly All your life You were only waiting for this moment to arise

35 (100011) 7 (000111) 60 (111100) 62 (111110) 6 (000110) 61 (111101) 4 (000100) 13 (001101) 63 (111111)
33 (100001) 14 (001110) 4 (000100) 9 (001001) 2 (000010) 4 (000100) 9 (001001) 2 (000010) 33 (100001)
4 (000100) 9 (001001) 33 (100001) 15 (001111) 3 (000011) 0 (000000) 33 (100001) 63 (111111) 0 (000000)
60 (111100) 63 (111111) 33 (100001) 10 (001010) 1 (000001) 33 (100001) 9 (001001) 4 (000100) 2 (000010)
3 (000011) 15 (001111) 33 (100001) 53 (110101) 60 (111100) 6 (000110) 0 (000000) 33 (100001) 15 (001111)
3 (000011) 0 (000000) 14 (001110) 0 (000000) 33 (100001) 63 (111101) 13 (001101) 10 (001010) 6 (000110)
0 (000000) 9 (001001) 33 (100001) 18 (010010) 4 (000100) 9 (001001) 2 (000010) 14 (001110) 33 (100001)
60 (111100) 9 (001001) 63 (111111) 33 (100001) 7 (000111) 0 (000000) 60 (111100) 13 (001101) 9 (001001)
33 (100001) 15 (001111) 10 (001010) 33 (100001) 1 (000001) 7 (000111) 20 (010100) 33 (100001) 34 (100010)
7 (000111) 7 (000111) 33 (100001) 20 (010100) 10 (001010) 16 (010000) 13 (001101) 33 (100001) 7 (000111)
4 (000100) 1 (000001) 0 (000000) 33 (100001) 58 (110101) 10 (001010) 16 (010000) 33 (100001) 18 (010010)
0 (000000) 13 (001101) 0 (000000) 33 (100001) 10 (001010) 9 (001001) 7 (000111) 20 (010100) 33 (100001)
18 (010010) 60 (111100) 4 (000100) 15 (001111) 4 (000100) 9 (001001) 2 (000010) 33 (100001) 1 (000001)
10 (001010) 13 (001101) 33 (100001) 15 (001111) 3 (000011) 4 (000100) 14 (001110) 33 (100001) 8 (001000)
10 (001010) 8 (001000) 0 (000000) 9 (001001) 15 (001111) 33 (100001) 15 (001111) 10 (001010) 33 (100001)
60 (111100) 13 (001101) 4 (000100) 14 (001110) 0 (000000)

28 (011100) 56 (111000) 3 (000011) 1 (000001) 57 (111001) 2 (000010) 59 (111011) 50 (110010) 0 (000000)
30 (011110) 49 (110001) 59 (110111) 54 (110110) 61 (111101) 59 (110111) 54 (110110) 61 (111101) 30 (011110)
59 (111011) 54 (110110) 30 (011100) 48 (110000) 60 (111100) 63 (111111) 30 (011100) 0 (000000) 63 (111111)
3 (000000) 0 (000000) 30 (011110) 53 (110101) 62 (111110) 30 (011100) 54 (110110) 59 (111011) 61 (111101)
60 (111100) 48 (110000) 30 (011110) 10 (001010) 3 (000011) 57 (111001) 63 (111111) 30 (011110) 48 (110000)
60 (111100) 63 (111111) 49 (110001) 63 (111111) 30 (011110) 2 (000010) 50 (110010) 53 (110101) 57 (111001)
63 (111111) 54 (110110) 30 (011110) 45 (101101) 59 (110111) 54 (110110) 61 (111011) 49 (110001) 30 (011110)
3 (000011) 54 (110110) 0 (000000) 30 (011110) 56 (111000) 63 (111111) 3 (000011) 50 (110010) 54 (110110)
30 (011110) 48 (110000) 53 (110101) 30 (011100) 62 (111110) 56 (110000) 43 (101011) 30 (011110) 29 (011101)
56 (111000) 56 (111000) 30 (011110) 43 (101011) 53 (110101) 47 (101111) 50 (110010) 30 (011110) 56 (111000)
59 (111011) 62 (111110) 63 (111111) 30 (011110) 5 (000101) 53 (110101) 47 (101111) 30 (011110) 45 (101101)
63 (111111) 50 (110010) 63 (111111) 30 (011110) 53 (110101) 54 (110110) 56 (111000) 43 (101011) 30 (011110)
45 (101101) 3 (000011) 59 (110111) 48 (110000) 59 (110111) 54 (110110) 61 (111101) 30 (011110) 62 (111110)
53 (110101) 50 (110010) 30 (011110) 48 (110000) 60 (111100) 59 (110111) 49 (110001) 30 (011110) 55 (110111)
53 (110101) 55 (110111) 63 (111111) 54 (110110) 48 (110000) 30 (011110) 48 (110000) 53 (110101) 30 (011110)
3 (000011) 50 (110010) 59 (110111) 49 (110001) 63 (111111)
```

6WhfxgZQe8PZUbZUb8ZU80ad8edhe8Tc8UZbaO8ohXd80adPd8gQTXdu8LZUbP8hUe8WdhQU80T8cWJ87WlW8JTNQ8WZcd8jTN8LdQd8TUWJ8LhZ0ZUb8cTQ80aZP8VTvdU080T8hQZPd

De esta prueba concluimos que nuestro código funciona con ejemplos un poco más complicados en términos del largo de la cadena.

4. Conclusión

El cumplimiento de esta tarea grupal nos permitió abordar la **codificación** y **decodificación** de textos mediante nuestro código implementado en el lenguaje de programación "Python".

El desarrollo de este trabajo fue un proceso bastante interesante, pues se abordó un problema nunca antes visto ni conocido por la gran mayoría de participantes, pero mediante nuestras reuniones, discusiones y entendimiento, pudimos crear un sistema de **codificación** y **decodificación** efectivo, en donde se puede apreciar la importancia de un análisis de forma individual y grupal, antes de directamente intentar solucionar un problema dado, en este caso, la tarea semestral.

Nuestro informe demuestra que supimos utilizar el tiempo como aliado, como plantea la introducción, por nuestra buena gestión de tiempo y efectividad en las reuniones, ya que, por razonamiento y diversas pruebas, se aprecia la efectividad del código, solucionando el problema en sí.

Más allá de la solución del problema, el proyecto representó una valiosa oportunidad para fortalecer habilidades de trabajo colaborativo, distribución de tareas y gestión del tiempo, los cuales son elementos esenciales en el desarrollo de proyectos de carácter grupal, al igual que para el desarrollo de las personas que somos y seremos. En esencia, el trabajo permitió cumplir los objetivos planteados, proporcionando una solución funcional y un aprendizaje significativo en términos de lo que es la programación y coordinación de equipo.