Programming Techniques - Assignment 1 - Question 3 - Report

List of Contents

Conclusion, Appraisal and References

My Links

- Link to Social

What is the Problem?

We need to make a variable that will hold a particular string and also allow the user to input this own text / string.
Using the user's input and the initialised string; we need to "*encrypt*" them.
You could say:

`Plain Text` ⟶ `Cipher / Ceasar Text`

We also need to output the ASCII values and its binary equivalent of the encrypted text.

After the encryption, we are now going to perform a **decryption**!
Using both the user's input and the initialised character, we need to convert them back to plain text. Hence, again, you could say that:

`Plain Text` ⟶ `Cipher / Ceasar Text` ⟶ `Plain Text`

Example: Encryption and Decryption

`Hello World` —Encrypting→ `Khoor#Zruog` —Decrypting→ `Hello Wo`

Description

The way that I plan to do this program is **heavily inspired** by the show *Mr Robot*. The program that they used run like that:

1. User enters the text / string to encrypt / decrypt
2. User presses the `Encrypt` or `Decrypt`
3. The program will output the **either** *plain text* **or** *cipher text*

Here is the link to the YouTube video where they used this algorithm:
https://www.youtube.com/watch?v=i9CBKGLVCME

Solution

C Code

In C, I know that we can output the ASCII values using a `for` loop and using the `%c` format specifier. Lets go ahead and run this code:

```c
#include <stdio.h>

int main() {

    // create a for loop to display the ASCII characters
    // from 32 to 127
    // DECLARE i: INTEGER
    for(int i = 32; i < 128; i++) {
        // output the ASCII characters
        printf("'%c', ", i);
    }

    return 0;

}
```

ASCII Table

Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D	36	24	$	68	44	D	100	64	d
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f
7	7	Bell	BEL	CTRL-G	39	27	'	71	47	G	103	67	g
8	8	Backspace	BS	CTRL-H	40	28	(	72	48	H	104	68	h
9	9	Horizontal tab	HT	CTRL-I	41	29	)	73	49	I	105	69	i
10	0A	Line feed	LF	CTRL-J	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	VT	CTRL-K	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	FF	CTRL-L	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage feed	CR	CTRL-M	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	SO	CTRL-N	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	SI	CTRL-O	47	2F	/	79	4F	O	111	6F	o
16	10	Data line escape	DLE	CTRL-P	48	30	0	80	50	P	112	70	p
17	11	Device control 1	DC1	CTRL-Q	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	DC2	CTRL-R	50	32	2	82	52	R	114	72	r
19	13	Device control 3	DC3	CTRL-S	51	33	3	83	53	S	115	73	s
20	14	Device control 4	DC4	CTRL-T	52	34	4	84	54	T	116	74	t
21	15	Neg acknowledge	NAK	CTRL-U	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	SYN	CTRL-V	54	36	6	86	56	V	118	76	v
23	17	End of xmit block	ETB	CTRL-W	55	37	7	87	57	W	119	77	w
24	18	Cancel	CAN	CTRL-X	56	38	8	88	58	X	120	78	x
25	19	End of medium	EM	CTRL-Y	57	39	9	89	59	Y	121	79	y
26	1A	Substitute	SUB	CTRL-Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	ESC	CTRL-[	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	FS	CTRL-\	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	GS	CTRL-]	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	RS	CTRL-^	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	US	CTRL-_	63	3F	?	95	5F	_	127	7F	DEL

The above C code will output:

```
' ', '!', '"', '#', '$', '%', '&', '''', '(', ')', '*', '+', ',', '-', '.', '/', '0', '1', '2',
'3', '4', '5', '6', '7', '8', '9', ':', ';', '<', '=', '>', '?', '@', 'A', 'B', 'C', 'D', 'E',
'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X',
'Y', 'Z', '[', '\', ']', '^', '_', '`', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k',
'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '{', '|', '}', '~',
' ',
```

> **Note**
>
> From the above ↑ image (*ASCII Table*), we can see that from 0 to 31 are basically:
>
> - Null
> - End of line
> - Backspace
> - Tab
> Hence, that is why I started the variable `i` at `32`, so that we start with the character `<Space>` and end with `DEL`.

> Lets start planning how I will make this program...

Plan

1. Have a greeting and selection screen

2. User can choose from:
    - Encryption
    - Decryption
    - Exit Program
3. Output the Encrypted / Decrypted text / string

`main` Function

Our `main` function will be our greetings and selection screen lets try to code it.

```
// import Scanner Object
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        // create Scanner Object
        Scanner user_input = new Scanner(System.in);

        // welcoming the user
        System.out.println("\nEncryption / Decryption Program\n");

        // ask the user for text to encrypt / decrypt
        System.out.print("Please Enter Text to Encrypt / Decrypt: ");
        // DECLARE user_text: STRING
        String user_text = user_input.nextLine();

        // display choices to user
        System.out.println("\nPress '1' To Encrypt");
        System.out.println("Press '2' To Decrypt");
        System.out.println("Press '3' To Exit\n");

        // ask the user if he wants to encrypt or decrypt
        System.out.print("Encrypt or Decrypt ( Please Input Number! ): ");
        // DECLARE choice: INTEGER
        int choice = user_input.nextInt();

    }

}
```

Adding Exception Handling

For example, when choosing whether to *encrypt* or *decrypt*; the program will prompt the user to enter an **Integer** number.
Hence, if they are going to input *Strings* or *Floats*... the program will explain the user why the program has not worked... instead of just displaying ("*yapping*") a lot of errors!

Python

In Python, to use Exception Handling, the syntax is like so:

```
try:
    statements
except ExceptionHandling:
    statements
finally:
    statements
```

Java

> This *code* snippet below ↓ could also be used as Explanation!

```
try {
    // actual code
    statements
}
catch (ExceptionHandling e) {
    // will run when it catches an Exception
    statements
}
// this block of code below will run no matter what
finally {
    statements
}
```

Updated `main` Function (with Exception Handling)

```
// import 'InputMismatchException' to handle exceptions
import java.util.InputMismatchException;
// import Scanner Object
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        // create Scanner Object
        Scanner user_input = new Scanner(System.in);

        // welcoming the user
        System.out.println("\nEncryption / Decryption Program\n");

        // exception handling
        try {
```

```
            // ask the user for text to encrypt / decrypt
            System.out.print("Please Enter Text to Encrypt / Decrypt: ");
            // DECLARE user_text: STRING
            String user_text = user_input.nextLine();

            // display choices to user
            System.out.println("\nPress '1' To Encrypt");
            System.out.println("Press '2' To Decrypt");
            System.out.println("Press '3' To Exit\n");

            // ask the user if he wants to encrypt or decrypt
            System.out.print("Encrypt or Decrypt ( Please Input Number! ): ");
            // DECLARE choice: INTEGER
            int choice = user_input.nextInt();

        }
        // this block of code will only execute when it finds an exception
        // like when user enters string / float values
        catch (InputMismatchException e) {

            System.out.print("\nPlease Enter '1' or '2' only\n\n");

        }
        // this block of code is always executed
        finally {

            // close the Scanner Object
            user_input.close();

        }

    }

}
```

`choose` Function

> Let me write the code first and then I will explain it to you.

```
// FUNCTION choose(DECLARE choice: INTEGER, DECLARE text: STRING, DECLARE key: INTEGER)
// passing by value ---> creates another "variable in memory"
// pass the user's "choice" and "number"
public static void choose(int choice, String text, int key) {

    // user selects to encrypt message
    if(choice == 1) {

        // call encryption function
        // pass the value of "choice" and "key" ( again passing-by-value )
        // the "key" is hard coded
        // we are going to take a look at the `encrypt` function later
```

```
        encrypt(text, key);

    }
    // user selects to decrypt message
    else if(choice == 2) {

        // call decryption function
        // pass the value of "choice" and "key" ( again passing-by-value )
        // again, the "key" is hard coded
        // we are going to take a look at the `decrypt` function later
        decrypt(text, key);

    }
    // if users enters the "choice" of '3'
    // user select to quit the program
    else if(choice == 3) {

        // terminate the program
        System.out.println("\nGoodbye!\n");
        // its Python equivalent is `exit()`
        System.exit(1);

    }
    // if the user "choice" is not equal to '1' or '2' or '3'
    // then the code below will run
    else {

        System.out.println("\nSomething Went Wrong... Exiting Program\n");
        // similar to `exit()` from Python
        System.exit(1);

    }

}
```

Explanation of `choose` Function

Let's get started!

The `if` Statement

Again, as I have been saying, in the `main` function we have our "*greeting*" to the user. Which looks something like this:

```
Press '1' To Encrypt
Press '2' To Decrypt
Press '3' To Exit

Encrypt or Decrypt ( Please Input Number! ):
```

Hence, the user will input **either** `1` or `2` or `3`. This user's input will be stored on the `choice` variable. We are then going to pass this value (*passing-by-value*) into our `choose` function like so:

```
public static void choose(int choice, String text, int key) {

    statements

}
```

> I have only showed you the Code Snippet here!

User Choice

Now what will happen if the user's choice was:

- `1`
  - ⇒ It will call the function `encrypt` so that the user's text can be encrypted.
- `2`
  - ⇒ It will call the function `decrypt` so that the user's text can be decrypted.
- `3`
  - ⇒ will exit the program.
- Users enters something that is not part of the program
  - ⇒ then the program will output this message "*Something Went Wrong... Exiting Program*"

`System.exit(1)`

What does `System.exit(1)` means?

This is similar to doing `exit()` in Python.

But then you might ask, why the hell there is a `1` inside the `()`?
In Java, the program can terminate "*successfully*" / correctly or "*unsuccessfully*" / un-correctly.

Hence, we can say that the if you used:

- A value of `0` ⇒ will terminate the program correctly
- A value of `1` ⇒ error has occurred, terminate program!

> Note

We also need to pass in the **key** and **text** so that our functions `encrypt` and `decrypt` (*will be covered below* ↓) will also get the values when we are going to be using them!

> ⚠ Warning
>
> Check the Python Code in the References section.

Key

> What is the Key? What is it used for?
>
> The "*key*" is simply an integer value that will allow us to **encrypt** / **decrypt** the string / text.
> Let me give you an example so you can understand it better.
>
> You have the character `A` and the value of you **key** is equal to `1`.
> Then when you are encrypting the original character `A` will become `B` (`A` → `B`).
> Lets refer to the ASCII Table above; the *decimal* value of `A` (*beware, its case-sensitive*) is `65`.
> Now, we know that the value of our *key* is `1`. Hence, when I am going to encrypt, I am **adding** that value (*key*) to `65`
> $\Rightarrow 65 + 1 = 66 \Rightarrow$ `66` is the character `B` will be the **encrypted** text.
> For decrypting, the process is the same but now in **reversed** (*i.e instead of adding, we subtract*).

`encrypt` Function

> This function will be responsible for **encrypting** the user's text / string.
> Again, let me show you the code first then we start the explanation process.

```java
// FUNCTION encrypt(DECLARE text: STRING, DECLARE key: INTEGER)
public static void encrypt(String text, int key) {

    System.out.println();

    // output the word "Encrypting" in a asthetic manner
    String welcome = "Encypting";

    // iterate through the word
    for (int i = 0; i < welcome.length(); i++) {

        // output each character / letter in the string
        System.out.print(welcome.charAt(i));
```

```java
        // this is similar to passing the argument
        // `flush = True` in Python's `print` function
        System.out.flush();

        try {
            // sleep the program for 200ms
            // this is equivalent to Python's `time.sleep(0.2)`
            Thread.sleep(200);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    System.out.println("\n");

    // create an array of characters
    // will be used to convert the "string" `text` into an array of characters
    char[] characters = text.toCharArray();

    // create another array to hold the encrypted text
    // this array will be converted to string later
    // takes the size / length of the array `characters`
    char[] encrypted_array = new char[characters.length];

    // populate the array `encrypted_array`
    // with the value from array `characters` with the offset applied
    for (int i = 0; i < characters.length; i++) {
        // adding value with offset APPLIED to new array
        encrypted_array[i] = (char) (characters[i] + key);
    }

    // convert "contents" of array `encrypted_array` to string
    // `new String()` creates another instance of a class string
    String encrypted_text = new String(encrypted_array);

    // output the ecrypted string to user
    System.out.println("Encrypted Text: " + encrypted_text);

    System.out.println("\nASCII Values of '" + text + "' BEFORE Encryption:");

    // outputs the ASCII values of text / string provided by the user
    // these values will be BEFORE the offset has been applied
    for (int i : characters) {
        System.out.printf("%d ", i);
    }

    System.out.println("\n\nASCII Values of '" + text + "' AFTER Encryption:");

    // outputs the ASCII values of text / string provided by the user
    // these values will be AFTER the offset has been applied
    for (int i : characters) {
        // applying offset
        i += key;
        System.out.printf("%d ", i);
    }
```

```
    System.out.println("\n");

}
```

Explanation of `encrypt` Function

> Let's break it down!

Passing Values into the Function

If you just take a look back at the `choose` function above ↑; you can see that we have passed the value of `choice`, `text` and `key`.
Now we are going to pass `text` and the `key` into the `encrypt` function.

> Obviously, we need to pass the user's text / string and they *hard-coded* key... else how are you actually going to be encrypting / decrypting.

Displaying "Encrypting" in *Slow Motion*

I wanted to implement / translate something like this in Java (*see below* ↓):

```
# import time module
import time

# DECLARE text: STRING
text = "\nHello World\n"

for i in text:
    # output each character in the variable `text`
    print(i, end="", flush=True)
    # sleep the program for 0.2 seconds
    time.sleep(0.2)
```

To be able to perform this action of **sleep** in Java, we need to use the `Thread.sleep()` function and pass the time (*in miliseconds*) as an *argument*.
Hence, using some documentation from GeeksforGeek, I made this code below ↓:

```
// output the word "Encrypting" in a asthetic manner
String welcome = "Encypting";

// iterate through the word
for (int i = 0; i < welcome.length(); i++) {

    // output each character / letter in the string
    System.out.print(welcome.charAt(i));
    // this is similar to passing the argument
    // `flush = True` in Python's `print` function
```

```
    System.out.flush();

    try {
        // sleep the program for 200ms
        // this is equivalent to Python's `time.sleep(0.2)`
        Thread.sleep(200);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

> I think that the comments in the code block are self-explanatory

> **Tip**
>
> In Java, you don't need to import anything to use `Thread.sleep()` function.

Encrypting the Text

> Again, go down in the References section and look at my own Python Code.
> I used this same logic / principle to **encrypt** the text / string.

STEPS:

1. Convert the user's input into an array of characters (`characters` array) by creating an array and using `.toCharArray()` function
2. Create another empty array (`encrypted_array`) in the with the size / length of the initial array (*i.e the array of characters in step 1*)
3. Populate the empty array (`encrypted_array`)
   - using a `for` loop, iterate through the array `characters` while also applying the `key` ...

     > **Note**
     > Even though `characters[i]` is of type `char`
     > When we type `characters[i] + key`; its of type `int`
     >
     > > This is because as soon as you add some integer calculation with strings in Java, it automatically uses its ASCII values
     > > Hence, we need to convert it back to `char` by type casting

   - add all the updated characters in the array `encrypted_array`
4. Convert the (*contents of the*) array `encrypted_array` into string by creating a new instance of a class String with `String()` function
5. Output the encrypted text to the user
6. In addition, also use 2 other `for` loops to output the ASCII values of the text / string before and after the key has been applied

Output of Encryption Function

In this case the user entered the text / string "Hello World"

```
Encypting

Encrypted Text: Khoor#Zruog

ASCII Values of 'Hello World' BEFORE Encryption:
72 101 108 108 111 32 87 111 114 108 100

ASCII Values of 'Hello World' AFTER Encryption:
75 104 111 111 114 35 90 114 117 111 103
```

`decrypt` Function

The decrypt function is literally the same thing as the `encrypt` function but just the reversed.
Instead of adding the `key`, we are removing / subtracting the `key`

```java
// FUNCTION decrypt(DECLARE text: STRING, DECLARE key: INTEGER)
public static void decrypt(String text, int key) {

    System.out.println();

    // output the word "decrypting" in a asthetic manner
    String welcome = "Decrypting";

    // iterate through the word
    for (int i = 0; i < welcome.length(); i++) {
        System.out.print(welcome.charAt(i));
        // this is similar to passing the argument
        // `flush = True` in Python's `print` function
        System.out.flush();

        try {
            // sleep the program for 200ms
            Thread.sleep(200);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    System.out.println("\n");

    // create an array of characters
    // will be used to convert the "string" `text` into an array of characters
    char[] characters = text.toCharArray();

    // create another array to hold the decrypted text
    // we are going to be converting this array to string later
    // this array uses the same length / size as the array `characters`
```

```java
    char[] decrypted_array = new char[characters.length];

    // populate the array `decrypted_array`
    // with the value from array `characters` with the offset removed
    for (int i = 0; i < characters.length; i++) {
        // adding value with offset REMOVED to new array
        decrypted_array[i] = (char) (characters[i] - key);
    }

    // convert "contents" of array `encrypted_array` to string
    // `new String()` creates another instance of a class string
    String decrypted_text = new String(decrypted_array);

    // output the ecrypted string to user
    System.out.println("Decrypted Text: " + decrypted_text);

    System.out.println("\nASCII Values of '" + text + "' BEFORE Decryption:");

    // outputs the ASCII values of text / string provided by the user
    // these values will be the values that has NOT been decrypted yet
    for (int i : characters) {
        System.out.printf("%d ", i);
    }

    System.out.println("\n\nASCII Values of '" + text + "' AFTER Decryption:");

    // outputs the ASCII values of text / string provided by the user
    // these values will be the values that has been decrypted
    // back to original / original ASCII values
    for (int i : characters) {
        // removing offset
        i -= key;
        System.out.printf("%d ", i);
    }

    System.out.println("\n");

}
```

> **Info**
>
> Here instead of doing `encrypted_array[i] = (char) (characters[i] + key`;
> We are doing `decrypted_array[i] = (char) (characters[i] - key)`!

Output of Decryption Function

```
Decrypting

Decrypted Text: Hello World

ASCII Values of 'Khoor#Zruog' BEFORE Decryption:
75 104 111 111 114 35 90 114 117 111 103

ASCII Values of 'Khoor#Zruog' AFTER Decryption:
72 101 108 108 111 32 87 111 114 108 100
```

Whole Java Program Code

```java
// import 'InputMismatchException' to handle exceptions
import java.util.InputMismatchException;
// import Scanner Object
import java.util.Scanner;

public class Question3 {

    // as I said, Java / C are similar in terms of memory management
    // I am going to write the functions first, so that it can "find" in
    // in the main program / function
    // even though this is NOT required

    // FUNCTION encrypt(DECLARE text: STRING, DECLARE key: INTEGER)
    public static void encrypt(String text, int key) {

        System.out.println();

        // output the word "Encrypting" in a asthetic manner
        String welcome = "Encypting";

        // iterate through the word
        for(int i = 0; i < welcome.length(); i++) {

            // output each character / letter in the string
            System.out.print(welcome.charAt(i));
            // this is similar to passing the argument
            // `flush = True` in Python's `print` function
            System.out.flush();

            try {
                // sleep the program for 200ms
                // this is equivalent to Python's `time.sleep(0.2)`
                Thread.sleep(200);
            }
            catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        System.out.println("\n");
```

```java
        // create an array of characters
        // will be used to convert the "string" `text` into an array of characters
        char[] characters = text.toCharArray();

        // create another array to hold the encrypted text
        // this array will be converted to string later
        // takes the size / length of the array `characters`
        char[] encrypted_array = new char[characters.length];

        // populate the array `encrypted_array`
        // with the value from array `characters` with the offset applied
        for(int i = 0; i < characters.length; i++) {
            // adding value with offset APPLIED to new array
            encrypted_array[i] = (char) (characters[i] + key);
        }

        // convert "contents" of array `encrypted_array` to string
        // `new String()` creates another instance of a class string
        String encrypted_text = new String(encrypted_array);

        // output the ecrypted string to user
        System.out.println("Encrypted Text: " + encrypted_text);

        System.out.println("\nASCII Values of '" + text + "' BEFORE Encryption:");

        // outputs the ASCII values of text / string provided by the user
        // these values will be BEFORE the offset has been applied
        for(int i : characters) {
            System.out.printf("%d ", i);
        }

        System.out.println("\n\nASCII Values of '" + text + "' AFTER Encryption:");

        // outputs the ASCII values of text / string provided by the user
        // these values will be AFTER the offset has been applied
        for(int i : characters) {
            // applying offset
            i += key;
            System.out.printf("%d ", i);
        }

        System.out.println("\n");

    }

    // FUNCTION decrypt(DECLARE text: STRING, DECLARE key: INTEGER)
    public static void decrypt(String text, int key) {

        System.out.println();

        // output the word "decrypting" in a asthetic manner
        String welcome = "Decrypting";

        // iterate through the word
        for(int i = 0; i < welcome.length(); i++) {
```

```
            System.out.print(welcome.charAt(i));
            // this is similar to passing the argument
            // `flush = True` in Python's `print` function
            System.out.flush();

            try {
                // sleep the program for 200ms
                Thread.sleep(200);
            }
            catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        System.out.println("\n");

        // create an array of characters
        // will be used to convert the "string" `text` into an array of characters
        char[] characters = text.toCharArray();

        // create another array to hold the decrypted text
        // we are going to be converting this array to string later
        // this array uses the same length / size as the array `characters`
        char[] decrypted_array = new char[characters.length];

        // populate the array `decrypted_array`
        // with the value from array `characters` with the offset removed
        for(int i = 0; i < characters.length; i++) {
            // adding value with offset REMOVED to new array
            decrypted_array[i] = (char) (characters[i] - key);
        }

        // convert "contents" of array `encrypted_array` to string
        // `new String()` creates another instance of a class string
        String decrypted_text = new String(decrypted_array);

        // output the ecrypted string to user
        System.out.println("Decrypted Text: " + decrypted_text);

        System.out.println("\nASCII Values of '" + text + "' BEFORE Decryption:");

        // outputs the ASCII values of text / string provided by the user
        // these values will be the values that has NOT been decrypted yet
        for(int i : characters) {
            System.out.printf("%d ", i);
        }

        System.out.println("\n\nASCII Values of '" + text + "' AFTER Decryption:");

        // outputs the ASCII values of text / string provided by the user
        // these values will be the values that has been decrypted
        // back to original / original ASCII values
        for(int i : characters) {
            // removing offset
            i -= key;
            System.out.printf("%d ", i);
```

```
        }

        System.out.println("\n");

    }

    // FUNCTION choose(DECLARE choice: INTEGER, DECLARE text: STRING, DECLARE key: INTEGER)
    public static void choose(int choice, String text, int key) {

        // user selects to encrypt message
        if(choice == 1) {

            // call encryption function
            encrypt(text, key);

        }
        // user selects to decrypt message
        else if(choice == 2) {

            // call decryption function
            decrypt(text, key);

        }
        // user select to quit the program
        else if(choice == 3) {

            // terminate the program
            System.out.println("\nGoodbye!\n");
            // its Python equivalent is `exit()`
            System.exit(1);

        }
        else {

            System.out.println("\nSomething Went Wrong... Exiting Program\n");
            System.exit(1);

        }

    }

    public static void main(String[] args) {

        // create Scanner Object
        Scanner user_input = new Scanner(System.in);

        // welcoming the user
        System.out.println("\nEncryption / Decryption Program\n");

        // exception handling
        try {

            // ask the user for text to encrypt / decrypt
            System.out.print("Please Enter Text to Encrypt / Decrypt: ");
            // DECLARE user_text: STRING
            String user_text = user_input.nextLine();
```

```java
        // display choices to user
        System.out.println("\nPress '1' To Encrypt");
        System.out.println("Press '2' To Decrypt");
        System.out.println("Press '3' To Exit\n");

        // ask the user if he wants to encrypt or decrypt
        System.out.print("Encrypt or Decrypt ( Please Input Number! ): ");
        // DECLARE choice: INTEGER
        int choice = user_input.nextInt();

        // key will be hardcoded
        // DECLARE key: INTEGER
        int key = 3;

        // initialised string
        String intialised_string = "I L O V E Y O U A N D I M I S S E D Y O U A L O T";

        /*
         * As my program is a bit like the decryption scene from Mr Robot
         * I need to have a variable that will hold the encrypted version of `initialised_s
         */
        String encrypted_inialised_string = "L#O#R#Y#H#\\#R#X#D#Q#G#L#P#L#V#V#H#G#\\#R#X#D#

        // call function `choose`
        choose(choice, user_text, key);

        // outputs the character '-', 80 times
        System.out.println("-".repeat(80));

        // encrypt the initialised string
        encrypt(intialised_string, key);

        // outputs the character '-', 80 times
        System.out.println("-".repeat(80));

        // decrypt the initialised string
        decrypt(encrypted_inialised_string, key);

    }
    // this block of code will only execute when it finds an exception
    // like when user enters string / float values
    catch (InputMismatchException e) {

        System.out.print("\nPlease Enter '1' or '2' only\n\n");

    }
    // this block of code is always executed
    finally {

        // close the Scanner Object
        user_input.close();

    }

}
```

```
}
```

Output of Whole Java Program

Encrypting `Hello World`

```
Encryption / Decryption Program

Please Enter Text to Encrypt / Decrypt: Hello World

Press '1' To Encrypt
Press '2' To Decrypt
Press '3' To Exit

Encrypt or Decrypt ( Please Input Number! ): 1

Encypting

Encrypted Text: Khoor#Zruog

ASCII Values of 'Hello World' BEFORE Encryption:
72 101 108 108 111 32 87 111 114 108 100

ASCII Values of 'Hello World' AFTER Encryption:
75 104 111 111 114 35 90 114 117 111 103
```

Decrypting `Khoor#Zruog`

```
Encryption / Decryption Program

Please Enter Text to Encrypt / Decrypt: Khoor#Zruog

Press '1' To Encrypt
Press '2' To Decrypt
Press '3' To Exit

Encrypt or Decrypt ( Please Input Number! ): 2

Decrypting

Decrypted Text: Hello World

ASCII Values of 'Khoor#Zruog' BEFORE Decryption:
75 104 111 111 114 35 90 114 117 111 103

ASCII Values of 'Khoor#Zruog' AFTER Decryption:
72 101 108 108 111 32 87 111 114 108 100
```

Conclusion

Hence, we can say that we have successfully implemented the question into Java Code.

Appraisal

Learnt how to convert strings into array of characters using the `.toCharArray()` function and also how to populate arrays and convert them to strings using `String()` methods to create a new instance of the class string.

References

- Inspired by Mr Robot decryption scene
- Initial Code taken from Alex Lee: https://www.youtube.com/watch?v=8wlE6DgOWBs

  > Don't worry my code is completely different that him!
  > Was just to use to understand `.toCharArray()`!

- My Python Code (*using the logic from below* ↓)

```
# DECLARE ARRAY a: INTEGER
# initialise array a with integer values
a = [1, 2, 3, 4, 5]
# DECLARE ARRAY b: INTEGER
b = []
# DECLARE ARRAY c: INTEGER
c = []

# add values of array `a` to `b`
# DECLARE i: INTEGER
for i in a:
    # adds the values of array `a` to `b`
    b.append(i)

# DECLARE j: INTEGER
for j in a:
    c.append(j + 1)

# output the values of array
print(f"Array `a` = {a}")
print(f"Array `b` = {b}")
print(f"Array `c` = {c}")
```

Outputs

```
Array `a` = [1, 2, 3, 4, 5]
Array `b` = [1, 2, 3, 4, 5]
Array `c` = [2, 3, 4, 5, 6]
```

- GeeksForGeeks for learning Exception Handling in Java: https://www.geeksforgeeks.org/exceptions-in-java/
- GeeksForGeeks for learning how to sleep program: https://www.geeksforgeeks.org/thread-sleep-method-in-java-with-examples/

Socials

- **Instagram**: https://www.instagram.com/s.sunhaloo/
- **YouTube**: https://www.youtube.com/channel/UCMkQZsuW6eHMhdUObLPSpwg
- **GitHub**: https://www.github.com/Sunhaloo

S.Sunhaloo
Thank You!