The encryption method is inspired by the **Caesar Cipher** code that works as follows:

* The Caesar Cipher is a monoalphabetic rotation cipher used by Gaius Julius Caesar. Caesar rotated each letter of the plaintext forward three times to encrypt, so that A became D, B became E, etc.

Our **encryption** method works for characters with the ASCII code between 33 and 126. It uses 4 variables:

**ascii \_characters**

* list of characters with the ASCII code between 33 and 126

**pad**

* represents the number of rotations we are going to use for encryption
* it can have values between 1 and 9
* its value is random

**fb**

* comes from *forward-backward*
* represents the direction of the current rotation (**forward - 1 or backward - 2**, based on the parity)
* at first, its value is 1 or 2 (random)
* its value is incremented for every character in the message to be encrypted when we iterate through it

**initial\_fb**

* keeps the first value of **fb** (useful for decryption)

The **encryption** method that we implemented shifts the letters in the message to be encrypted as follows:

1. **pad** value is generated
2. **fb** value is generated
3. **initial\_fb** equals fb
4. the message to be encrypted is iterated
   1. index of the current character is searched in the **ascii\_characters** string
   2. index of the new character is calculated, based on the **fb** parity
   3. **pad** is added to the index found (**fb** = 1), or subtracted from it (**fb** = 2)
   4. if the new index value is **> 0** we **% len(ascii\_characters)** to be sure that we iterate correctly through the string
   5. if the new index is **< 0**, we do not **% len(ascii\_characters)** anymore
5. we add **pad** value to the encrypted message
6. we add **initial\_fb** value to the encrypted message
7. we return the encrypted message

The **decryption** method works similarly, but the directions of the rotations are inverted now, compared to the encryption method (**forward - 2, backward - 1**).

The encrypted message will always have 2 characters more than the original message, the last one is **fb** and the second last is **pad** (useful for decryption)

**Example:**

ascii\_characters = "!"#$%&'()\*+,-./0123456789:;<=>?@ABCDEFGHIJKL**M**NOPQ**R**STUVWXYZ[\]^\_`abcdefghijklmnopqrstuvwxyz{|}~"

message = "M!@uV1@t@?!"

Encrypted message: **RzEp[,EoE:&51**

Decrypted message: **M!@uV1@t@?!**

pad = 5

fb = 1 ⇒ first rotation goes forward

initial\_fb = 1

1. **R** is the fifth character forward in the ascii\_characters after **M**
2. **z** is the fifth character backward in the ascii\_characters after **!**
3. **E** is the fifth character forward in the ascii\_characters after **@**
4. **p** is the fifth character backward in the ascii\_characters after **u**
5. **[** is the fifth character forward in the ascii\_characters after **V**
6. **,** is the fifth character backward in the ascii\_characters after **1**
7. **E** is the fifth character forward in the ascii\_characters after **@**
8. **o** is the fifth character backward in the ascii\_characters after **t**
9. **E** is the fifth character forward in the ascii\_characters after **@**
10. **:** is the fifth character backward in the ascii\_characters after **?**
11. **&** is the fifth character forward in the ascii\_characters after **!**
12. **pad** is added to the encrypted message ⇒ **5**
13. **initial\_fb** is added to the encrypted message ⇒ **1**