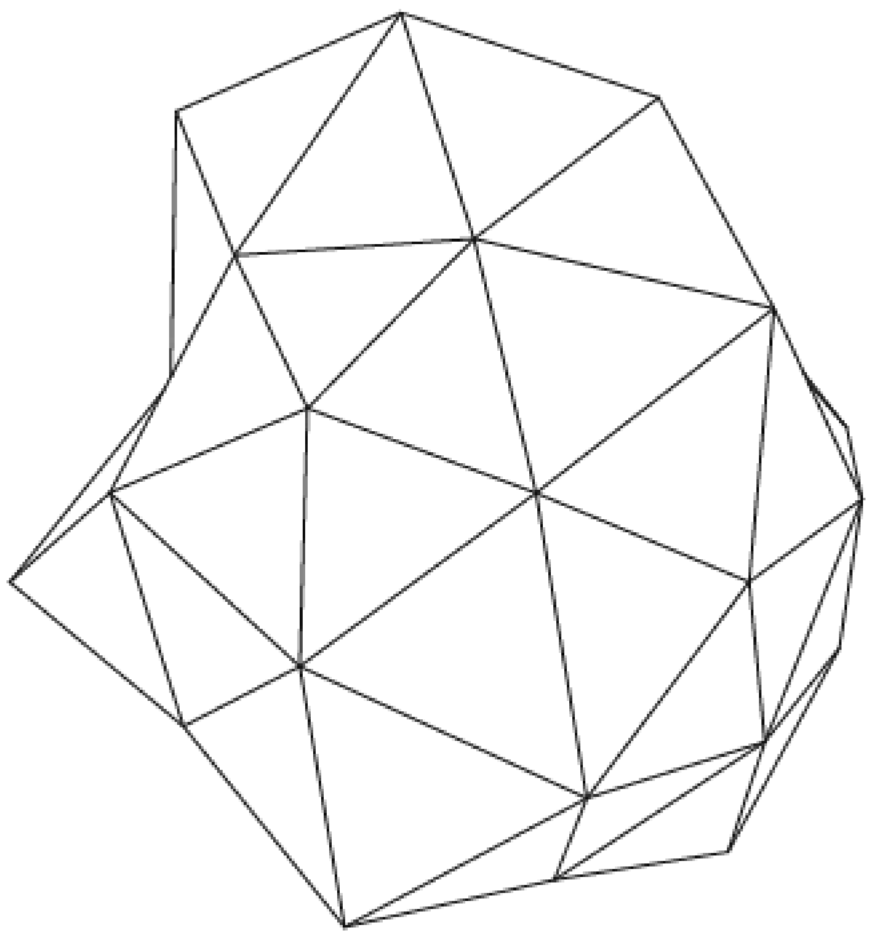
/web-dev-ak/tricodex



Tri-CodeX Documentation

Adarsh Singh

Table of Contents

Introduction to Tri-CodeX

Hey Visitors!

TricodeX is a custom character encoding algorithm that converts text into a structured three-digit numeric code. Each letter, number, space, and common symbol is mapped to a unique three-digit code, ensuring a consistent and reversible transformation.

The primary purpose of TricodeX is to help users understand the basics of encoding and decoding while also serving as a fun tool for message encryption.

Encoding and Decoding Process

Encoding (Text → Code)

1. Take a string of text as input.
2. For each character in the text:
   - Find its corresponding three-digit code from the encoding table.
   - Append the code to the encrypted message, separating each code with a . (dot).
3. Return the encrypted message as a string of numeric codes.

Example:

```
Input:  "Hii Adarsh!"
Output: "081.090.090.000.011.040.010.180.190.080.280"
```

Decoding (Code → Text)

1. Take an encrypted numeric string as input.
2. Split the string by the . separator to extract individual codes.
3. Convert each three-digit code back into its corresponding character using the decoding table.
4. Reconstruct and return the original text.

Example:

```
Input:  "081.090.090.000.011.040.010.180.190.080.280"
Output: "Hii Adarsh!"
```

Logic Behind Character Codes

Why we need three digits?

Each character (a-z,A-Z) can be represented by a two-digit number:

- a = 01
- b = 02
- z = 26

However, we need to differentiate between lowercase and uppercase letters. That's where the third digit comes in. In our system:

- 0 represents lowercase letters
- 1 represents uppercase letters

So now, we have:

- a = 010, A = 011
- b = 020, B = 021
- z = 260, Z = 261

How Do We Separate Characters?

Now that each character has a unique numeric representation, we need a way to distinguish characters within an encrypted message.

This is where the . (dot) separator is used.

Example:

```
//        'H' 'i' 'i' ' ' 'A' 'd' 'a' 'r' 's' 'h' '!'
Input:  "081.090.090.000.011.040.010.180.190.080.280"
Output: "Hii Adarsh!"
```

The dot ensures that each encoded character remains separate and can be correctly decrypted back into text.

Encoding & Decoding Functions

Encoding function

Version 1: Using Loops (Iterative Approach)

```
function encrypt(msgDecrypted, charToCode){
  let msgEncrypted = [];

  for (let char of msgDecrypted){
    if (charToCode[char]){
      msgEncrypted.push(charToCode[char])
    }
  }

  return msgEncrypted.join('.');
}
```

Version 2: Using String Methods (Efficient Approach)

```
function encrypt(text, charToCode) {
  return text.split("").map(char => charToCode[char] ||
"").join(".");
}
```

Decoding function

Version 1: Using Loops (Iterative Approach)

```
function decrypt(msgEncrypted, codeToChar){
  let msgDecrypted = "";
  let codeOfChar = msgEncrypted.split(".")

  for (let code of codeOfChar){
    if (codeToChar[code]){
      msgDecrypted += codeToChar[code];
    }
  }

  return msgDecrypted;
}
```

Version 2: Using String Methods (Efficient Approach)

```
function decrypt(encryptedText, codeToChar) {
  return encryptedText.split(".").map(code => codeToChar[code] ||
"").join("");
}
```

These functions efficiently convert text into its encoded form and back using a mapping system.

Character Mapping

Character Mapping (JSON)

Below is the JavaScript object mapping of characters to their respective three-digit codes:

Example:

```
{
    " ": "000",
    "a": "010", "A": "011", "b": "020", "B": "021",
    "c": "030", "C": "031", "d": "040", "D": "041",
    "e": "050", "E": "051", "f": "060", "F": "061",
    "g": "070", "G": "071", "h": "080", "H": "081",
    "i": "090", "I": "091", "j": "100", "J": "101",
    "k": "110", "K": "111", "l": "120", "L": "121",
    "m": "130", "M": "131", "n": "140", "N": "141",
    "o": "150", "O": "151", "p": "160", "P": "161",
    "q": "170", "Q": "171", "r": "180", "R": "181",
    "s": "190", "S": "191", "t": "200", "T": "201",
    "u": "210", "U": "211", "v": "220", "V": "221",
    "w": "230", "W": "231", "x": "240", "X": "241",
    "y": "250", "Y": "251", "z": "260", "Z": "261",

    "0": "270", "1": "271", "2": "272", "3": "273", "4": "274",
    "5": "275", "6": "276", "7": "277", "8": "278", "9": "279",

    "!": "280", "@": "281", "#": "282", "$": "283", "%": "284",
    "^": "285", "&": "286", "*": "287", "(": "288", ")": "289",
    "-": "290", "_": "291", "+": "292", "=": "293", "[": "294",
    "]": "295", "{": "296", "}": "297", "|": "298", ";": "299",
    ":": "300", "'": "301", "\"": "302", ",": "303", ".": "304",
    "<": "305", ">": "306", "/": "307", "?": "308", "\\": "309"
}
```

Character Mapping Table

Below is the complete mapping of characters to their respective three-digit codes:

Character	Code	Character	Code	Character	Code	Character	Code	Character	Code
" "	000								
a	000	A	011	b	020	B	021		
c	030	C	031	d	040	D	041		
e	050	E	051	f	060	F	061		
g	070	G	071	h	080	H	081		
i	090	I	091	j	100	J	101		
k	110	K	111	l	120	L	121		
m	130	M	131	n	140	N	141		
o	150	O	151	p	160	P	161		
q	170	Q	171	r	180	R	181		
s	190	S	191	t	200	T	201		
u	210	U	211	v	220	V	221		
w	230	W	231	x	240	X	241		
y	250	Y	251	z	260	Z	261		
0	270	1	271	2	272	3	273	4	274
5	275	6	276	7	277	8	278	9	279
!	280	@	281	#	282	$	283	%	284
^	285	&	286	*	287	(	288	)	289
-	290	_	291	+	292	=	293	[	294
]	295	{	296	}	297	|	298	;	299
:	300	'	301	"	302	,	303	.	304
<	305	>	306	/	307	?	308	\	309

About Me

Hey Visitors!

I’m Adarsh, a BCA graduate, and here’s the story of how I created this project.
During my 5th semester in college, I studied the RSA algorithm in my syllabus. That got me curious about the world of cryptography!

One day, after my exams, I was just lying on my bed, thinking about making something fun. That’s when the idea of TricodeX popped into my mind, and I started working on it. The first version of the code was ready in just an hour! After some improvements and optimizations, I decided to turn it into a web tool so I could share it with you all.

I know this code isn’t super efficient or ready for complex projects, but hey—who cares? It’s just a fun project that I built while learning.