

Get Cryptic

Work through these exercises from Cryptopals Set 1 (<http://cryptopals.com/sets/1>)

Get as far as you can.

Try to TDD the solution if able.

Optional but recommended

You should write the code to parse hexadecimal and base64 as well instead of using built in functions, it's more fun than it sounds.

Helpful information

Hex (*More info*)

In computing hexadecimal (also base 16, or hex) is a positional numeral system with a base of 16.

It uses sixteen distinct symbols the symbols 0–9 to represent values zero to nine, and A, B, C, D, E, F (or alternatively a, b, c, d, e, f) to represent values ten to fifteen.

Value	Hex
0-9	A-Z or a-z
A-F	10-15

Each hex digit represents exactly 4 bits of data.

387922	Decimal						
= 01011110101101010010	=	0101	1110	1011	0101	0010	Binary
	=	5	14	11	5	2	Decimal
	=	5	E	B	5	2	Hexidecimal
	=	5EB52	Hexidecimal				

Base64 (*More info*)

In computing base64 is a positional numeral system with a base of 64.

It uses sixty four distinct symbols the symbols A–Z to represent values 0-25, a-z to represent values 26-51, 0-9 to represent 52-61, + to represent 62 and / to represent 63.

The final '=' sequence indicates that the last group contained only one byte, and '=' indicates that it contained two bytes.

Value	Hex
0-25	A–Z
26-51	a-z
52-61	0-9
62	+
63	/
Padding	=

Each base64 digit represents exactly 6 bits of data.

Text	Man			
ASCII	M	a	n	
	77	97	110	
	01001101	01100001	01101110	
Bit pattern	0 1 0 0 1 1	0 1 0 1 1 0	0 0 0 1 0 1	1 0 1 1 1 0
	010011	010110	000101	101110
Index	19	22	5	46
Base64-encoded	T	W	F	u
	= TWFu			
Text	M			
ASCII	M			
	77	0	0	
	01001101	00000000	00000000	
Bit pattern	0 1 0 0 1 1	0 1 0 0 0 0	0 0 0 0 0 0	0 0 0 0 0 0
	010011	010000	000000	000000
Index	19	16	0	0
Base64-encoded	T	Q	=	=
	= TQ==			
Text	Ma			
ASCII	M	a		
	77	97	0	
	01001101	01100001	10000000	
Bit pattern	0 1 0 0 1 1	0 1 0 1 1 0	0 0 0 1 0 0	0 0 0 0 0 0
	010011	010110	000100	000000
Index	19	22	4	0
Base64-encoded	T	W	E	=
	= TWE=			

ASCII (*More info*)

See [More info] to learn about ASCII

Bitwise Operations

NOT (*More info*)

The **bitwise NOT**, or **complement**, is a unary operation that performs logical negation on each bit, forming the ones' complement of the given binary value. Bits that are 0 become 1, and those that are 1 become 0. For example:

NOT 0111 (decimal 7)
= 1000 (decimal 8)

AND (*More info*)

A bitwise AND takes two equal-length binary representations and performs the logical AND operation on each pair of the corresponding bits, by multiplying them.

Thus, if both bits in the compared position are 1, the bit in the resulting binary representation is 1 ($1 \times 1 = 1$); otherwise, the result is 0 ($1 \times 0 = 0$ and $0 \times 0 = 0$). For example:

0101 (decimal 5)
AND 0011 (decimal 3)
= 0001 (decimal 1)

OR (*More info*)

A bitwise OR takes two bit patterns of equal length and performs the logical inclusive OR operation on each pair of corresponding bits.

The result in each position is 0 if both bits are 0, while otherwise the result is 1. For example:

```
0101 (decimal 5)
OR 0011 (decimal 3)
= 0111 (decimal 7)
```

XOR ([More info](#))

A bitwise XOR takes two bit patterns of equal length and performs the logical exclusive OR operation on each pair of corresponding bits. The result in each position is 1 if only the first bit is 1 or only the second bit is 1, but will be 0 if both are 0 or both are 1.

In this we perform the comparison of two bits, being 1 if the two bits are different, and 0 if they are the same. For example:

```
0110 (decimal 6)
XOR 1010 (decimal 10)
= 1100 (decimal 12)
```

Arithmetic shift ([More info](#))

In an arithmetic shift, the bits that are shifted out of either end are discarded. In a left arithmetic shift, zeros are shifted in on the right; in a right arithmetic shift, the sign bit is shifted in on the left.

This example uses an 8-bit register:

```
00010111 (decimal +23) LEFT-SHIFT
= 00101110 (decimal +46)

10010111 (decimal -105) RIGHT-SHIFT
= 11001011 (decimal -53)

00010111 (decimal +23) LEFT-SHIFT-BY-TWO
= 01011100 (decimal +92)
```