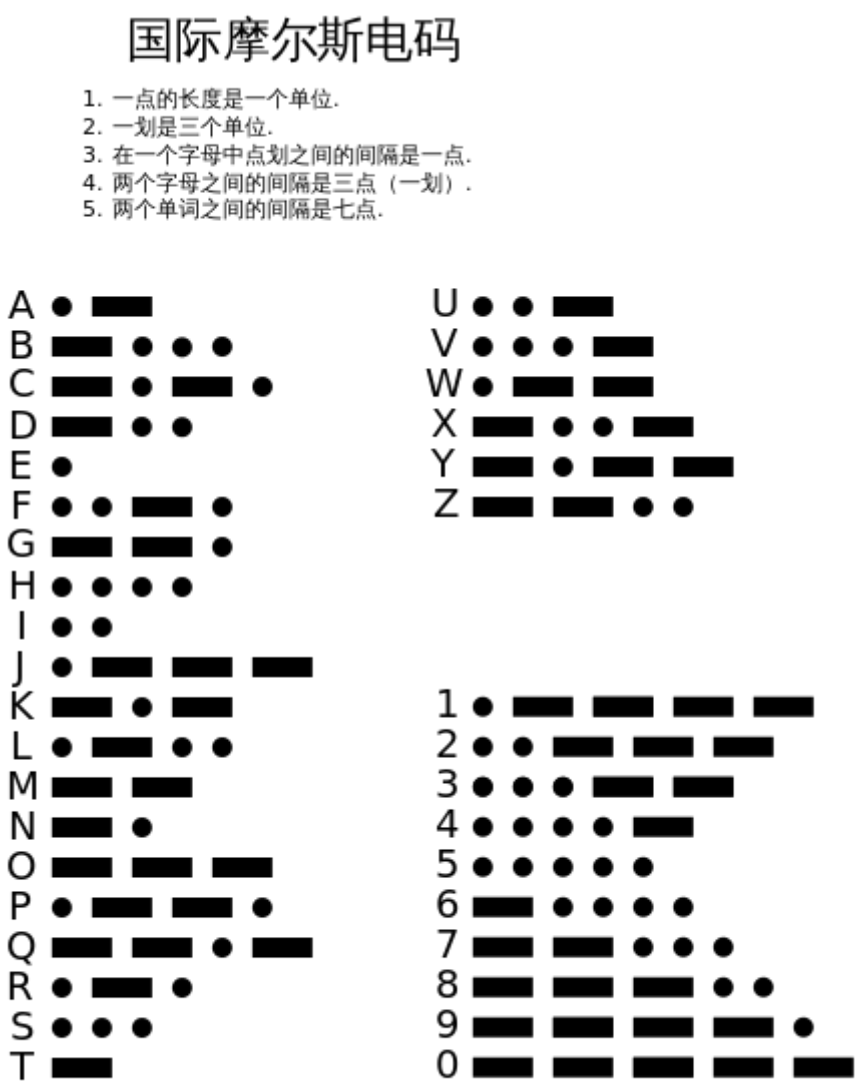
通信领域常用编码 ¶

电话拨号编码 ¶

1-9 分别使用 1-9 个脉冲，0 则表示使用 10 个脉冲。

Morse 编码 ¶

参见 摩尔斯编码 - 维基百科，对应表如下



特点 ¶

- 只有 `.` 和 `-`;
- 最多 6 位;
- 也可以使用 `01` 串表示。

工具 ¶

- 摩尔斯编码在线转换

题目 ¶

- JarvisOJ - Basec - 「-.- 字符串」

敲击码 ¶

敲击码（Tap code）是一种以非常简单的方式对文本信息进行编码的方法。因该编码对信息通过使用一系列的点击声音来编码而命名，敲击码是基于 5 ×5 方格波利比奥斯方阵来实现的，不同点是用 K 字母被整合到 C 中。

Tap Code	1	2	3	4	5
1	A	B	C/K	D	E
2	F	G	H	I	J
3	L	M	N	O	P
4	Q	R	S	T	U
5	V	W	X	Y	Z

源文本	F	O	X
位置	2,1	3,4	5,3
敲击码	.. .	... .	....

曼彻斯特编码 ¶

- 曼彻斯特编码 - 维基百科

格雷编码 ¶

- 格雷码 - 维基百科

计算机相关的编码 ¶

本节介绍一些计算机相关的编码。

字母表编码 ¶

- A-Z/a-z 对应 1-26 或者 0-25

ASCII 编码 ¶

十进制	二进制	符号	十进制	二进制	符号	十进制	二进制	符号	十进制	二进制	符号
0	0000 0000	NUL	32	0010 0000	[空格]	64	0100 0000	@	96	0110 0000	`
1	0000 0001	SOH	33	0010 0001	!	65	0100 0001	A	97	0110 0001	a
2	0000 0010	STX	34	0010 0010	"	66	0100 0010	B	98	0110 0010	b
3	0000 0011	ETX	35	0010 0011	#	67	0100 0011	C	99	0110 0011	c
4	0000 0100	EOT	36	0010 0100	$	68	0100 0100	D	100	0110 0100	d
5	0000 0101	ENQ	37	0010 0101	%	69	0100 0101	E	101	0110 0101	e
6	0000 0110	ACK	38	0010 0110	&	70	0100 0110	F	102	0110 0110	f
7	0000 0111	BEL	39	0010 0111	'	71	0100 0111	G	103	0110 0111	g
8	0000 1000	BS	40	0010 1000	(	72	0100 1000	H	104	0110 1000	h
9	0000 1001	HT	41	0010 1001	)	73	0100 1001	I	105	0110 1001	i
10	0000 1010	LF	42	0010 1010	*	74	0100 1010	J	106	0110 1010	j
11	0000 1011	VT	43	0010 1011	+	75	0100 1011	K	107	0110 1011	k
12	0000 1100	FF	44	0010 1100	,	76	0100 1100	L	108	0110 1100	l
13	0000 1101	CR	45	0010 1101	-	77	0100 1101	M	109	0110 1101	m
14	0000 1110	SO	46	0010 1110	.	78	0100 1110	N	110	0110 1110	n
15	0000 1111	SI	47	0010 1111	/	79	0100 1111	O	111	0110 1111	o
16	0001 0000	DLE	48	0011 0000	0	80	0101 0000	P	112	0111 0000	p
17	0001 0001	DC1	49	0011 0001	1	81	0101 0001	Q	113	0111 0001	q
18	0001 0010	DC2	50	0011 0010	2	82	0101 0010	R	114	0111 0010	r
19	0001 0011	DC3	51	0011 0011	3	83	0101 0011	S	115	0111 0011	s
20	0001 0100	DC4	52	0011 0100	4	84	0101 0100	T	116	0111 0100	t
21	0001 0101	NAK	53	0011 0101	5	85	0101 0101	U	117	0111 0101	u
22	0001 0110	SYN	54	0011 0110	6	86	0101 0110	V	118	0111 0110	v
23	0001 0111	ETB	55	0011 0111	7	87	0101 0111	W	119	0111 0111	w
24	0001 1000	CAN	56	0011 1000	8	88	0101 1000	X	120	0111 1000	x
25	0001 1001	EM	57	0011 1001	9	89	0101 1001	Y	121	0111 1001	y
26	0001 1010	SUB	58	0011 1010	:	90	0101 1010	Z	122	0111 1010	z
27	0001 1011	ESC	59	0011 1011	;	91	0101 1011	[	123	0111 1011	{
28	0001 1100	FS	60	0011 1100	<	92	0101 1100	\	124	0111 1100	|
29	0001 1101	GS	61	0011 1101	=	93	0101 1101	]	125	0111 1101	}
30	0001 1110	RS	62	0011 1110	>	94	0101 1110	^	126	0111 1110	~
31	0001 1111	US	63	0011 1111	?	95	0101 1111	_	127	0111 1111	DEL

特点 ¶

我们一般使用的 ascii 编码的时候采用的都是可见字符，而且主要是如下字符

- 0-9, 48-57
- A-Z, 65-90
- a-z, 97-122

变形 ¶

二进制编码 ¶

将 ascii 码对应的数字换成二进制表示形式。

- 只有 0 和 1
- 不大于 8 位，一般 7 位也可以，因为可见字符到 127。
- 其实是另一种 ascii 编码。

十六进制编码 ¶

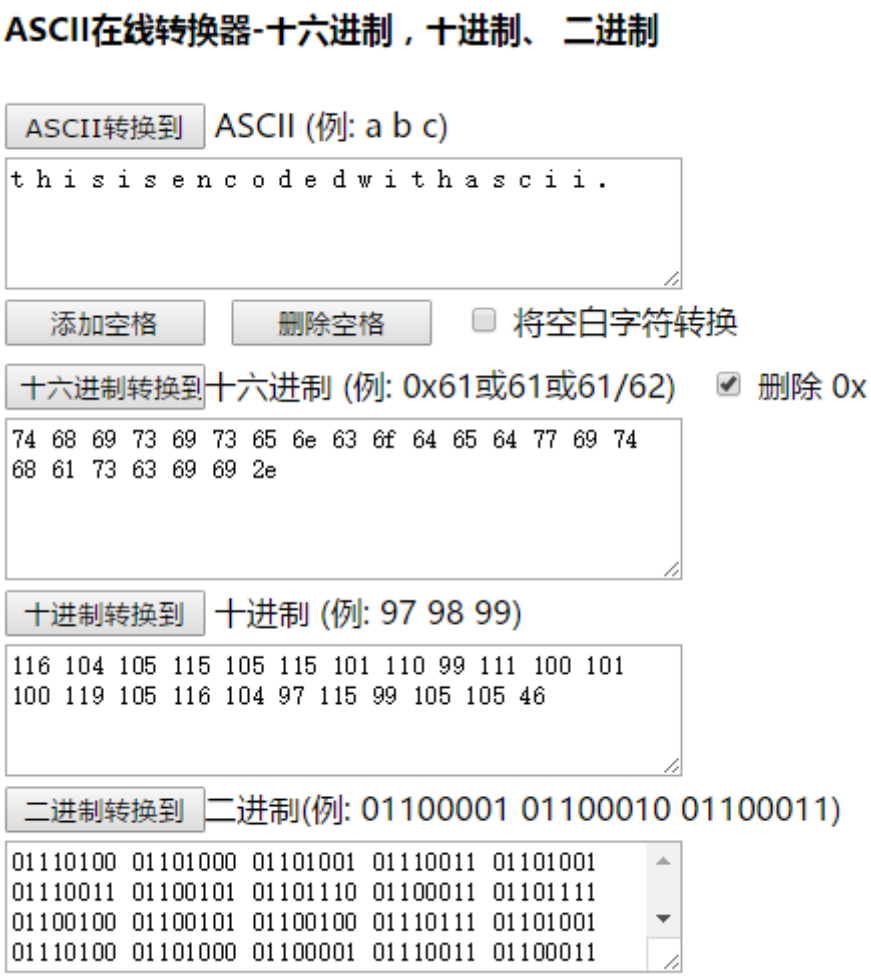
将 ascii 码对应的数字换成十六进制表示形式。

- A-Z→41-5 A
- a-z→61-7 A

工具 ¶

- jpk, ascii to number, number to ascii
- http://www.ab126.com/goju/1711.html

例子 ¶



2018 DEFCON Quals ghettohackers: Throwback ¶

题目描述如下

```
Anyo!e!howouldsacrificepo!icyforexecu!!onspeedthink!securityisacomm!ditytop!urin
toasy!tem!
```

第一直觉应该是我们去补全这些叹号对应的内容，从而得到 flag，但是补全后并不行，那么我们可以把源字符串按照 `!` 分割，然后字符串长度 1 对应字母 a，长度 2 对应字母 b，以此类推

```
ori =
'Anyo!e!howouldsacrificepo!icyforexecu!!onspeedthink!securityisacomm!ditytop!uri
ntoasy!tem!!'
sp = ori.split('!')
print repr(''.join(chr(97 + len(s) - 1) for s in sp))
```

进而可以得到，这里同时需要假设 0 个字符为空格。因为这正好使得原文可读。

```
dark logic
```

题目 ¶

- Jarvis-basic - 德军的密码

Base 编码 ¶

base xx 中的 xx 表示的是采用多少个字符进行编码，比如说 base64 就是采用以下 64 个字符编码，由于 2 的 6 次方等于 64，所以每 6 个比特为一个单元，对应某个可打印字符。3 个字节就有 24 个比特，对应于 4 个 Base64 单元，即 3 个字节需要用 4 个可打印字符来表示。它可用来作为电子邮件的传输编码。在 Base64 中的可打印字符包括字母 A-Z、a-z、数字 0-9，这样共有 62 个字符，此外两个可打印符号在不同的系统中而不同。

The Base64 Alphabet

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w	(pad)	=
15	P	32	g	49	x		
16	Q	33	h	50	y		

具体介绍参见 Base64 - 维基百科。

编码 man

文本	M								a								n							
ASCII编码	77								97								110							
二进制位	0	1	0	0	1	1	0	1	0	1	1	0	0	0	0	1	0	1	1	0	1	1	1	0
索引	19								22								5							
Base64编码	T								W								F							

如果要编码的字节数不能被 3 整除，最后会多出 1 个或 2 个字节，那么可以使用下面的方法进行处理：先使用 0 值在末尾补足，使其能够被 3 整除，然后再进行 base64 的编码。在编码后的 base64 文本后加上一个或两个 `=` 号，代表补足的字节数。也就是说，当最后剩余一个八位字节（一个 byte）时，最后一个 6 位的 base64 字节块有四位是 0 值，最后附加上两个等号；如果最后剩余两个八位字节（2 个

byte）时，最后一个 6 位的 base 字节块有两位是 0 值，最后附加一个等号。参考下表：

文本（1 Byte）	A																								
二进制位	0	1	0	0	0	0	0	1																	
二进制位（补0）	0	1	0	0	0	0	0	1	**0**	**0**	**0**	**0**													
Base64编码	Q						Q																		
文本（2 Byte）	B								C																
二进制位	0	1	0	0	0	0	1	0	0	1	0	0	0	0	1	1			x	x	x	x	x	x	
二进制位（补0）	0	1	0	0	0	0	1	0	0	1	0	0	0	0	1	1	**0**	**0**	x	x	x	x	x	x	
Base64编码	Q						k						M												

由于解码时补位的 0 并不参与运算，可以在该处隐藏信息。

与 base64 类似，base32 使用 32 个可见字符进行编码，2 的 5 次方为 32，所以每 5 bit 为 1 个分组。5 字节为 40 bit，对应于 8 个 base32 分组，即 5 个字节用 8 个 base32 中字符来表示。但如果不足 5 个字节，则会先对第一个不足 5 bit 的分组用 0 补足了 5 bit ，对后面剩余分组全部使用 “=” 填充，直到补满 5 个字节。由此可知，base32 最多只有 6 个等号出现。例如：

文本 (1 Byte)	A																								
二进制位	0	1	0	0	0	0	0	1																	
二进制位 (补0)	0	1	0	0	0	0	0	1	0	0															
Base32编码	I					E			=				=				=				=				=

特点 ¶

- base64 结尾可能会有 `=` 号，但最多有 2 个
- base32 结尾可能会有 `=` 号，但最多有 6 个
- 根据 base 的不同，字符集会有所限制
- **有可能需要自己加等号**
- **= 也就是 3D**
- 更多内容请参见 base rfc

工具 ¶

- http://www1.tc711.com/tool/BASE64.htm
- python 库函数
- 读取隐写信息脚本

例子 ¶

题目描述参见 `ctf-challenge` 中 misc 分类的 base64-stego 目录中的 data.txt 文件。

使用脚本读取隐写信息。

```
import base64

def deStego(stegoFile):
    b64table =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
    with open(stegoFile,'r') as stegoText:
        message = ""
        for line in stegoText:
            try:
                text = line[line.index("=") - 1:-1]
```

```
                message += "".join([ bin( 0 if i == '=' else b64table.find(i))
[2:].zfill(6) for i in text])[2 if text.count('=') ==2 else 4:6]
            except:
                pass
    return "".join([chr(int(message[i:i+8],2)) for i in
range(0,len(message),8)])

print(deStego("text.txt"))
```

输出:

```
    flag{BASE64_i5_amaz1ng}
```

题目 ¶

霍夫曼编码 ¶

参见 霍夫曼编码。

XXencoding¶

XXencode 将输入文本以每三个字节为单位进行编码。如果最后剩下的资料少于三个字节，不够的部份用零补齐。这三个字节共有 24 个 Bit，以 6bit 为单位分为 4 个组，每个组以十进制来表示所出现的数值只会落在 0 到 63 之间。以所对应值的位置字符代替。

```
          1         2         3         4         5         6
012345678901234567890123456789012345678901234567890123456789012345678901234567890123
|         |         |         |         |         |         |
+-0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
```

具体信息参见维基百科

特点 ¶

- 只有数字，大小写字母
- + 号，- 号。

工具 ¶

- http://web.chacuo.net/charsetxxencode

题目 ¶

URL 编码 ¶

参见 URL 编码 - 维基百科。

特点 ¶

- 大量的百分号

工具 ¶

题目 ¶

Unicode 编码 ¶

参见 Unicode - 维基百科。

注意，它有四种表现形式。

例子 ¶

源文本： `The`

&#x [Hex]: `The`

&# [Decimal]: `The`

\U [Hex]: `\U0054\U0068\U0065`

\U+ [Hex]: `\U+0054\U+0068\U+0065`

工具 ¶

题目 ¶

HTML 实体编码 ¶

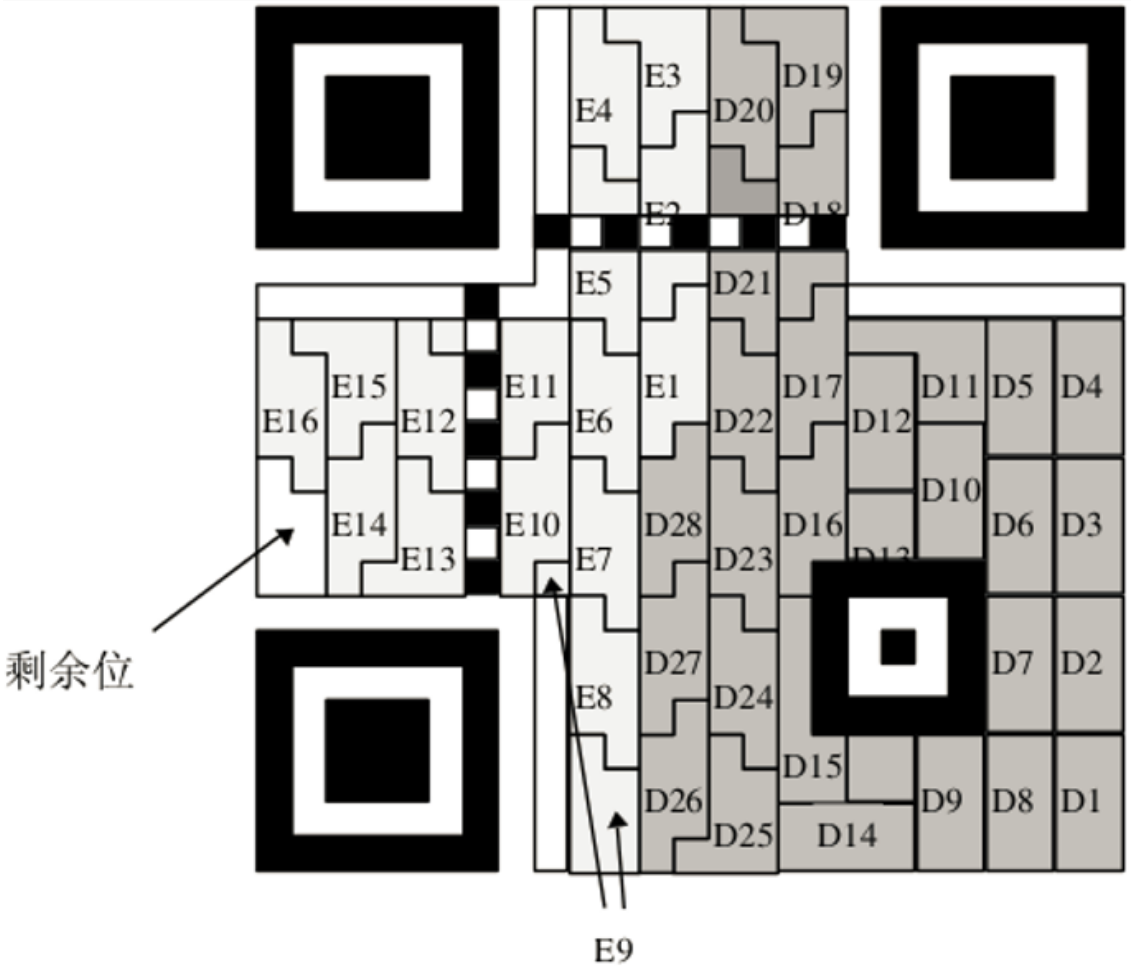
现实世界中常用的编码 ¶

条形码 ¶

- 宽度不等的多个黑条和空白，按照一定的编码规则排列，用以表达一组信息的图形标识符
- 国际标准
- EAN-13 商品标准，13 位数字
- Code-39：39 字符
- Code-128：128 字符
- 条形码在线识别

二维码 ¶

- 用某种特定几何图形按一定规律在平面分步的黑白相间的图形记录数据符号信息
- 堆叠式 / 行排式二维码：Code 16 k、Code 49、PDF417
- 矩阵式二维码：QR CODE



Version Info

Alignment

Format Info

Timing

Finder

Quiet Zone

Data and Error Correction Codewords