

# Framing

(How to delineate different  
socket data transfers)

# Framing Example

- Consider python client/server scripts where a client can ask for repeated server downloads.
- e.g., the client sends a "GET" over the TCP connection.
- The server responds by sending a file/message back (In this example we will randomly choose between a number of message examples.
- Once the download has occurred, the client may send another "GET".

# Decoding "GET" at Server

- See `RecvBytes.py`
- See `get_messages_no_framing.py`

# Framing Methods

1. Ensure that fixed length messages are sent. The receiver can then keep doing `socket.recv` until the required number of bytes have been read. (See `get_messages_fixed_size.py`). This is not a very interesting case! (This is what our echo client/server does, i.e., we know the size of the download based on what we originally sent.)
2. Enforce a protocol rule that only one message is sent on a given TCP connection. Once that message is completed, the sender will close the socket. This will function as a signal to the client that the transfer is completed since `recv()` will return with zero bytes (See `get_messages_one_per_connection.py`)

# Framing

3. Use a special character to delimit the end of a message. The receiver keeps reading the socket until it sees the delimiter. Care must be taken that the message itself cannot include the special character! This can be done using base64 encoding. (See `get_messages_eom_char.py`)

4. Add a "message header" to the front of each message, indicating its length. The receiver can then read from the socket until the required number of bytes are read. Note that if the message size is encoded as an integer, we need to worry about byte order, etc. (See `file_download_protocol.py`)

# Base64 Encoding

Base 64 encoding maps binary data to a subset of the ASCII character set.

This is done by mapping 8-bit binary characters to 6-bits, represented by a known (ASCII) character subset, e.g., Base64 encoding, e.g.,

```
import base64
```

```
msg = "Hello World!"
```

```
msg_utf8 = msg.encode('utf-8')
```

```
msg_utf8_base64 = base64.b64encode(msg_utf8)
```

# ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

# Base64 Encoding

Binary	ASCII
000000	A
000001	B
000010	C
000011	D
000100	E
000101	F
000110	G
000111	H
001000	I
001001	J
001010	K
001011	L
001100	M
001101	N
001110	O
001111	P




Binary	ASCII
010000	Q
010001	R
010010	S
010011	T
010100	U
010101	V
010110	W
010111	X
011000	Y
011001	Z
011010	a
011011	b
011100	c
011101	d
011110	e
011111	f

Binary	ASCII
100000	g
100001	h
100010	i
100011	j
100100	k
100101	l
100110	m
100111	n
101000	o
101001	p
101010	q
101011	r
101100	s
101101	t
101110	u
101111	v

Binary	ASCII
110000	w
110001	x
110010	y
110011	z
110100	0
110101	1
110110	2
110111	3
111000	4
111001	5
111010	6
111011	7
111100	8
111101	9
111110	+
111111	/



# Base64 Encoding

	M	a	n	
ASCII	77	97	110	
Bit Pattern	01001101	01100001	01101110	
				
Index	19	22	5	46
Base64	T	W	F	u

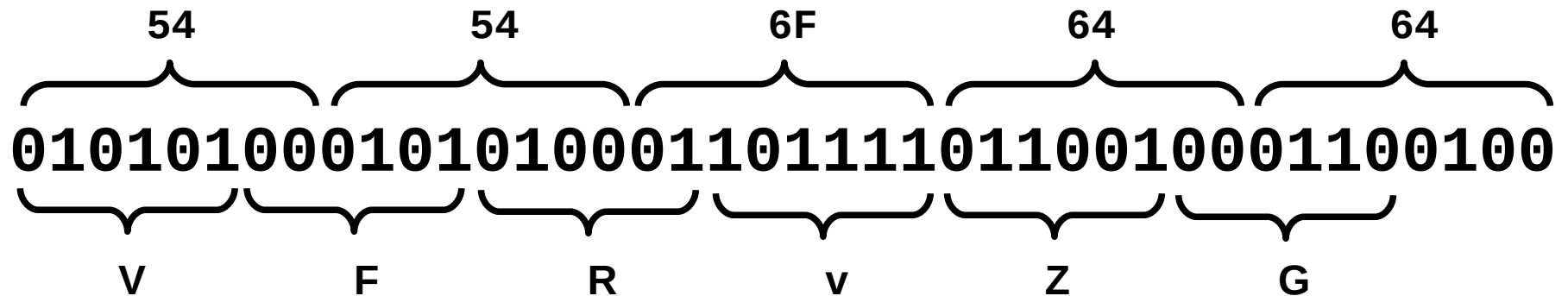
- Base64 encodes based on 24-bit (i.e., 3 x 8 bit) blocks as follows:
  - Lay out the data bytes to be sent in a contiguous stream of binary.
  - For each 24-bit chunk, divide it up into four 6-bit words.
  - Each 6-bit word is used as an index into the table of characters on the previous overhead. The 8-bit ASCII representation of that character is used as the base64 encoding.
- See <https://en.wikipedia.org/wiki/Base64> for examples.

# Base64 Encoding

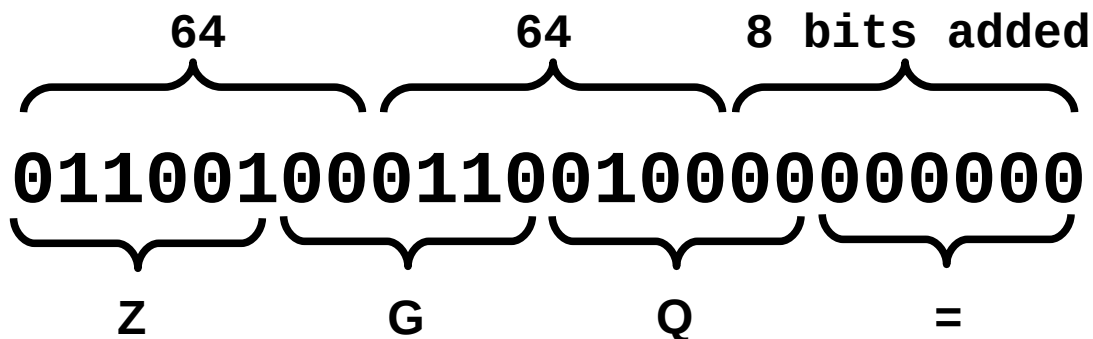
- Special padding rules are used when the last bits in the byte stream do not end on a 24-bit boundary (i.e., the message is not a multiple of 3 bytes)
- If the number of input bytes is not divisible by three, i.e., when there are only one or two bytes left from the last 3-bytes (24-bit) block, then:
  - Extra bytes with value zero are appended so that there are three bytes. Then do the base64 conversion.
  - If there is only one significant input byte, only the first two base64 digits are picked (12 bits).
- The '==' sequence means that the last group contained only one byte (i.e., two 6-bit zero words), and '=' that it contained two bytes (i.e., one 6-bit zero word), i.e., "==" means we appended 16 zero-bits and "=" means we appended 8 zero-bits.

# Base64 Encoding

- Encode the ASCII string "TTodd" (hex: 54 54 6F 64 64)



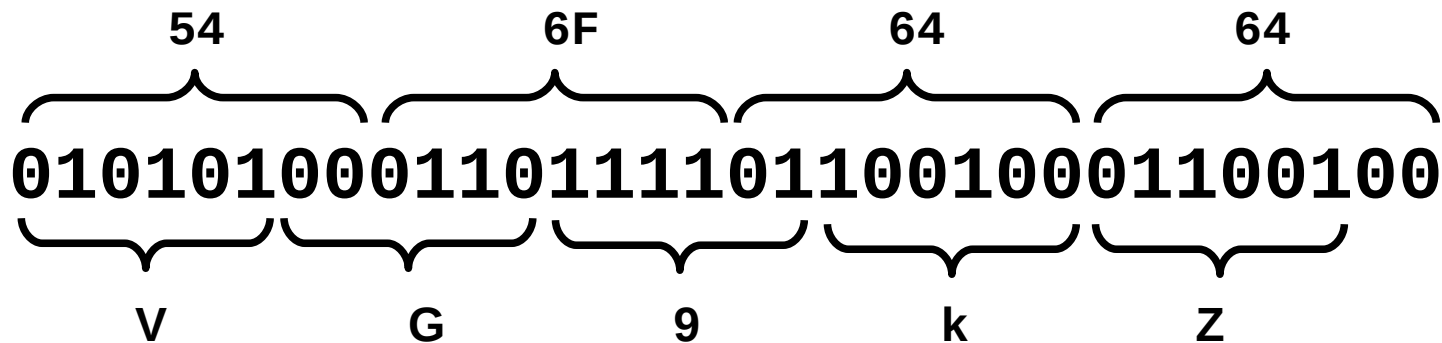
- The last block was only 16 bits. Add 8 zero bits to make it 24:



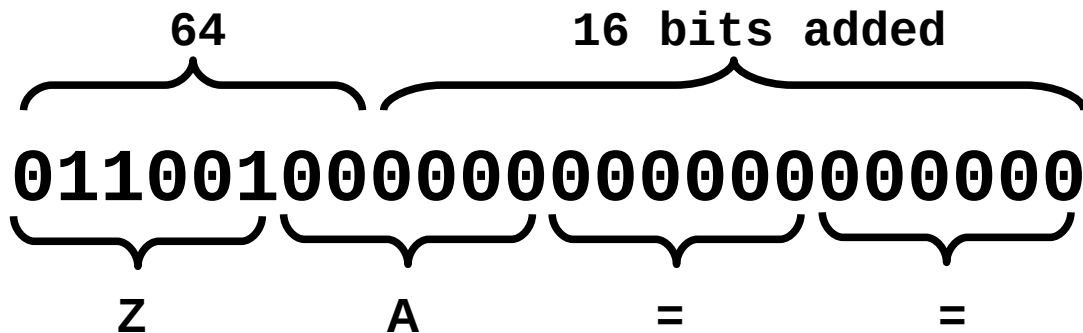
- Base64 encoding: **VFRvZGQ=**

# Base64 Encoding

- If instead, I encode "Todd" (hex: 54 6F 64 64)



- The last block was only 8 bits. Add 16 zero bits to make it 24:



- Base64 encoding: **VG9kZA==**

base64examples.py  
base64\_image\_example.py  
image\_email.txt