Counting in Binary

The polynomial expansion of the decimal number 362 is $3 \times 10^2 + 6 \times 10^1 + 2 \times 10^0$. Binary numbers work in exactly the same way, but with powers of 2 instead of powers of 10. So the number $00100101 = 2^5 + 2^2 + 2^0 = 37$.

Data and Encodings

Imagine that we write a small amount of textual data to file a file called `output.txt` using the following Python code.

```
1 with open('output.txt', 'wt', encoding='ASCII') as fout:
2   print("De La Soul is Dead", file=fout, )
```

If we cat the file in unix we see

```
$ cat output.txt
De La Soul is Dead
```

We also know that the data must really just be a sequence of zeros and ones, so we can use the `xxd` command to see this underlying representation

```
$ xxd -b output.txt
0000000: 01000100 01100101 00100000 01001100 01100001 00100000  De La
0000006: 01010011 01101111 01110101 01101100 00100000 01101001  Soul i
000000c: 01110011 00100000 01000100 01100101 01100001 01100100  s Dead
0000012: 00001010                                               .
```

Each line tells you the start of the next byte, in hexadecimal, the next 6 bytes, and then the *textual* interpretation of the bytes. Bytes are just 8 bits of data. Notice that the first byte is 01000100, which is $2^6 + 2^2 = 68$. The first character is "D". This is not a coincidence. When the "D" was written to disk, it was written using the ASCII encoding. ASCII stands for American Standard Code for Information Interchange. It encodes the character "D" using the integer 68. ASCII encodes 128 characters, requiring 7 bits, which you can see in the cart below.

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

ASCII is so 1997

There's a sociology thesis to be written about ASCII—it encodes the English alphabet, numbers, some punctuation, and a few out-of-date control characters for your dot matrix printer and terminal like a LINE FEED (ASCII 10) and BELL (ASCII 7)—but nothing more. It is like saying the world is all of Iowa or any other state in the union. But it is also pervasive, so we need to deal with it intelligently.

Unicode

Unicode recognizes that the world doesn't start and end with the English alphabet. It associates an integer (called a code point) with a platonic character, like the symbol "@" (U+0040, 64 in decimal) or a glyph of an anchor (U+2693, 9875 in decimal). There are 1,114,112 code points overall, in the range from 0 to 10FFFF in hexadecimal.

Encodings

One represents unicode through *encodings*. For example, one could represent strings of characters as code points directly. For example, we could just use a standard 32-bit integer to store a code point and then strings would be lists or arrays of integers. This has several disadvantages:

- Because code points are grouped by locale, most of the bits in single encoded code point will be 0. This seems wasteful.
- Each character requires 21 bits to represent it. Because the basic unit of addressable memory on a computer is the byte, which is traditionally 8-bits, we are really working with 24 bits, so even going with a fixed-length, 3-byte encoding seems wasteful.
- Different processors order bytes differently, so a fixed-length encoding is not portable.

UTF-8

UTF-8 stands for Unicode Transformation Format. The 8 means that encodings are done in 8-bit blocks (called octets by UTF-8). UTF-8 is a variable-length encoding, meaning that depending on which unicode code point it's transforming, the corresponding encoding will either be 1,2,3 or 4 bytes long. Here is a schematic of how UTF-8 encodes data.

Bits in code pt	First code pt	Last code pt	# Bytes	Byte 1	Byte 2	Byte 3	Byte 4
7	U+0000	U+007F	1	0xxxxxxx			
11	U+0080	U+07FF	2	110xxxxx	10xxxxxx		
16	U+0800	U+FFFF	3	1110xxxx	10xxxxxx	10xxxxxx	
21	U+10000	U+1FFFFF	4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

Here are several advantages of UTF-8.

- UTF-8 is compatible with ASCII because it encodes all ASCII characters as ASCII values;
- UTF-8 can encode all the unicode code points; and
- UTF-8 is self-synchronizing—one can use the byte signatures to both determine the number of bytes and the order of the bytes.

Unicode in Python

The `str` type in Python is a sequence of unicode code points. Those points are stored compactly, under-the-hood, so space is not wasted. Given a character, one can grab it's code point using the `ord` built-in function.

```
>>> print(" ".join([str(ord(c)) for c in 'De La Soul is Dead']))
>>> 68 101 32 76 97 32 83 111 117 108 32 105 115 32 68 101 97 100
```

One can convert a unicode code point into a character using the built-in function `chr`.

```
>>> codepoints = [int(x) for x in "68 101 32 76 97 32 83 111 117 108 32 105 115 32 68 101 97 100".split()]
>>> print("".join([chr(cp) for cp in codepoints]))
>>> De La Soul is Dead
```

One can embed unicode code points in hexadecimal directly in string literals using the `u` preface, so

```
>>> '\u2603'
```

represents a picture of a snowman.

Encoding Unicode

Writing strings to files mean encoding them in some format. By default this is UTF-8, but Python supports writing in many different formats including ASCII. Use the `encode` method of Python to encode unicode data in a particular format.

```
>>> "De La Soul is Dead".encode("utf-8")
b'De La Soul is Dead'
```

Notice the `b` prefix means this is binary data—the interpreter is printing those characters to the screen by actually decoding them (it will do this for any ASCII character) but if we throw on a unicode star (U+2b50) we'll see some raw hex.

```
>>> "De La Soul is Dead\u2b50".encode("utf-8")
b'De La Soul is Dead\xe2\xad\x90'
```

Notice that in UTF-8, it takes three bytes to encode a star. These bytes in binary are

```
>>> " ".join(["{0:b}".format(x) for x in  "\u2b50".encode("utf-8")])
'11100010 10101101 10010000'
```

So if we wrote this string out to disk

```
with open('output2.txt', 'wt') as fout:
  print("De La Soul is Dead\u2b50", file=fout, end="" )
```

we see that the final three bytes are faithfully preserved and `xxd` does not know how to decode them, so it uses the period. Here you see firsthand that UTF-8 uses ASCII encodings for ASCII characters and is also variable length.

```
$ xxd -b output2.txt
0000000: 01000100 01100101 00100000 01001100 01100001 00100000  De La
0000006: 01010011 01101111 01110101 01101100 00100000 01101001  Soul i
000000c: 01110011 00100000 01000100 01100101 01100001 01100100  s Dead
0000012: 11100010 10101101 10010000                             ....
```