Data Representation in Hardware & Software

John Rodriguez

Today’s objectives

- how computers manipulate/interpret data
    - bytes & binary numbers
    - numbers, characters and images

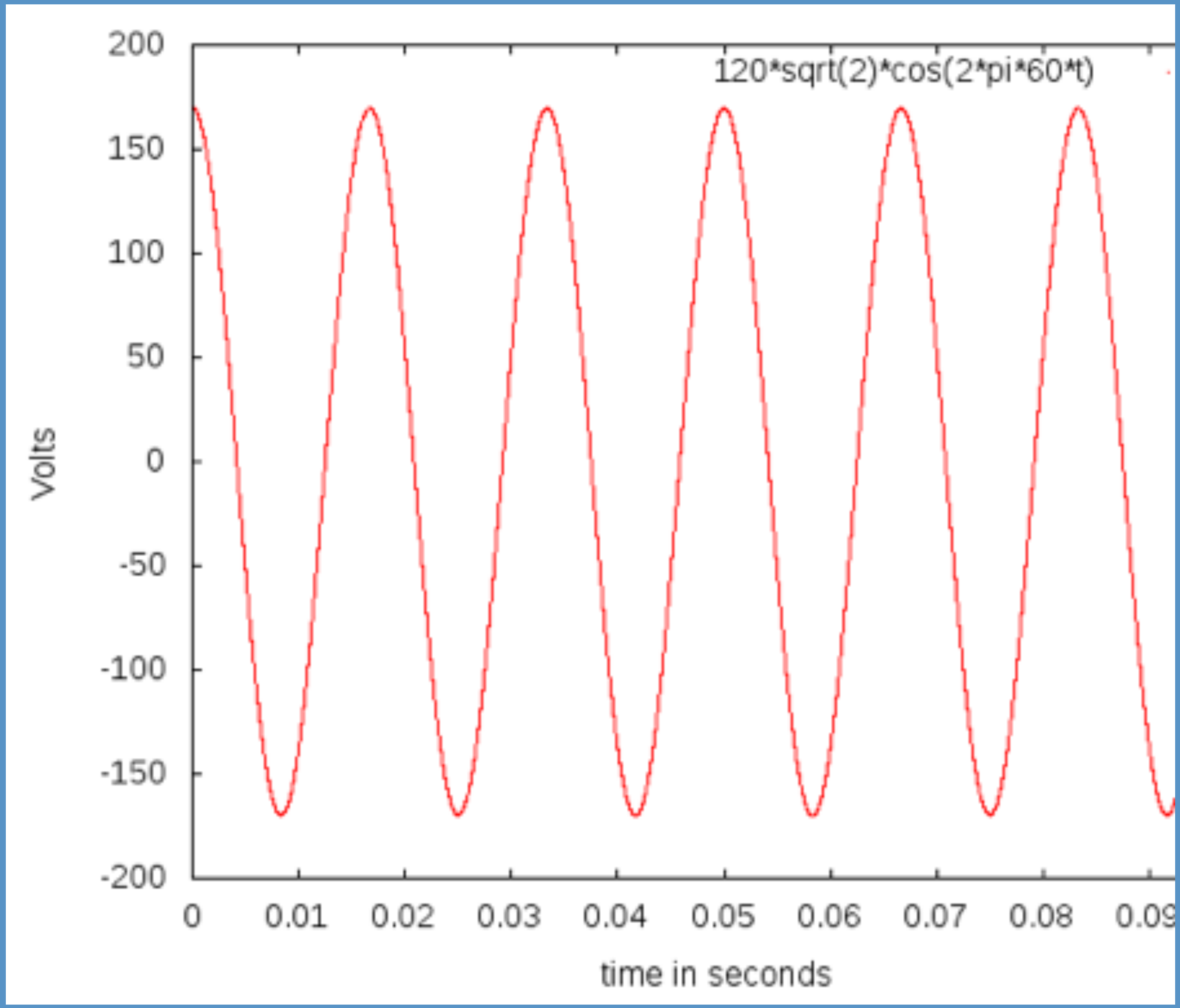
Electricity

- In the US, wall outlets provide 120V of alternating current (AC) at 60Hz
- Why not direct current (DC)?

https://www.youtube.com/watch?v=BcIDRet787k

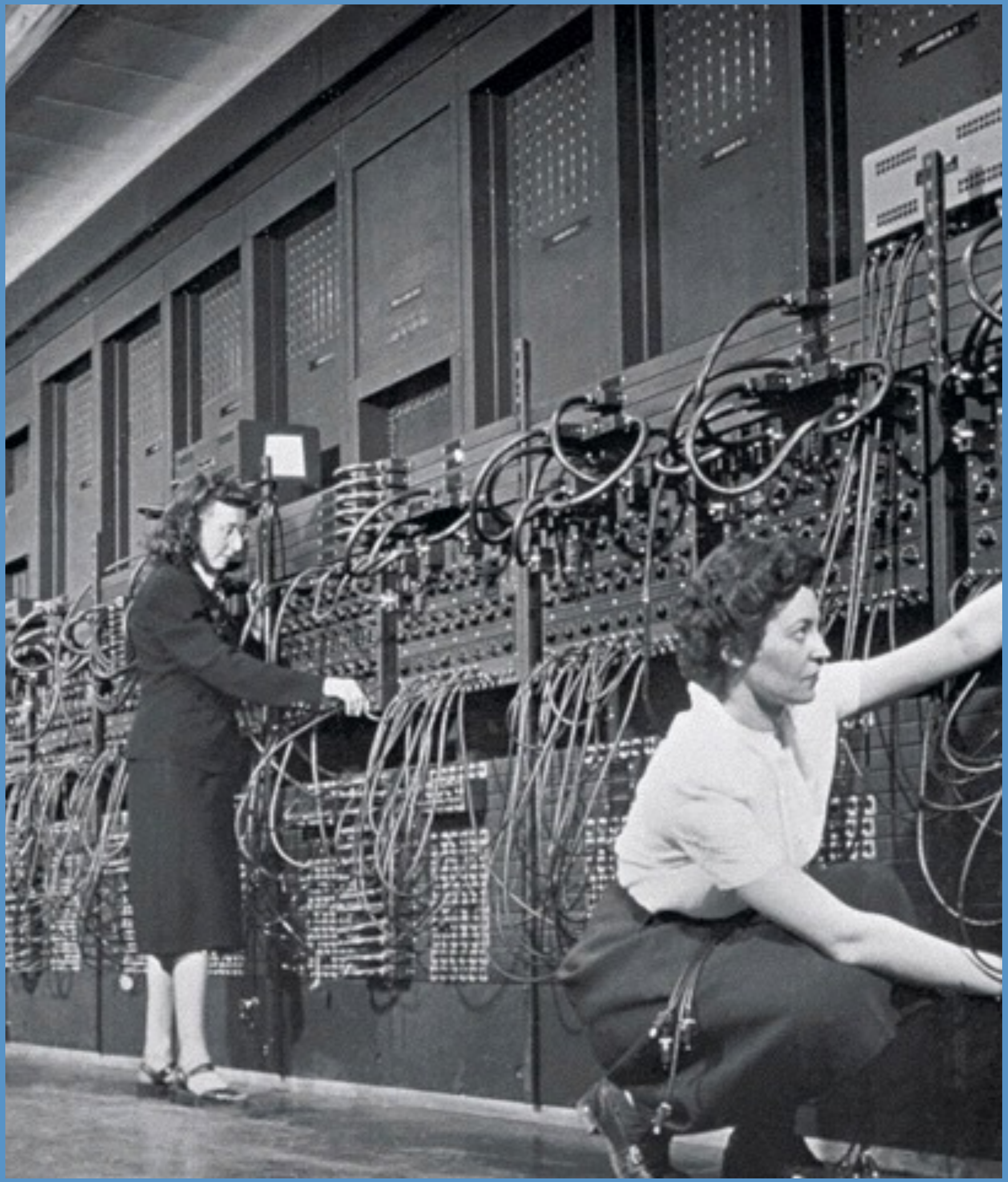
AC vs DC

- DC is built for short distances
- AC for long distances
- It’s hard to sample AC at exact intervals



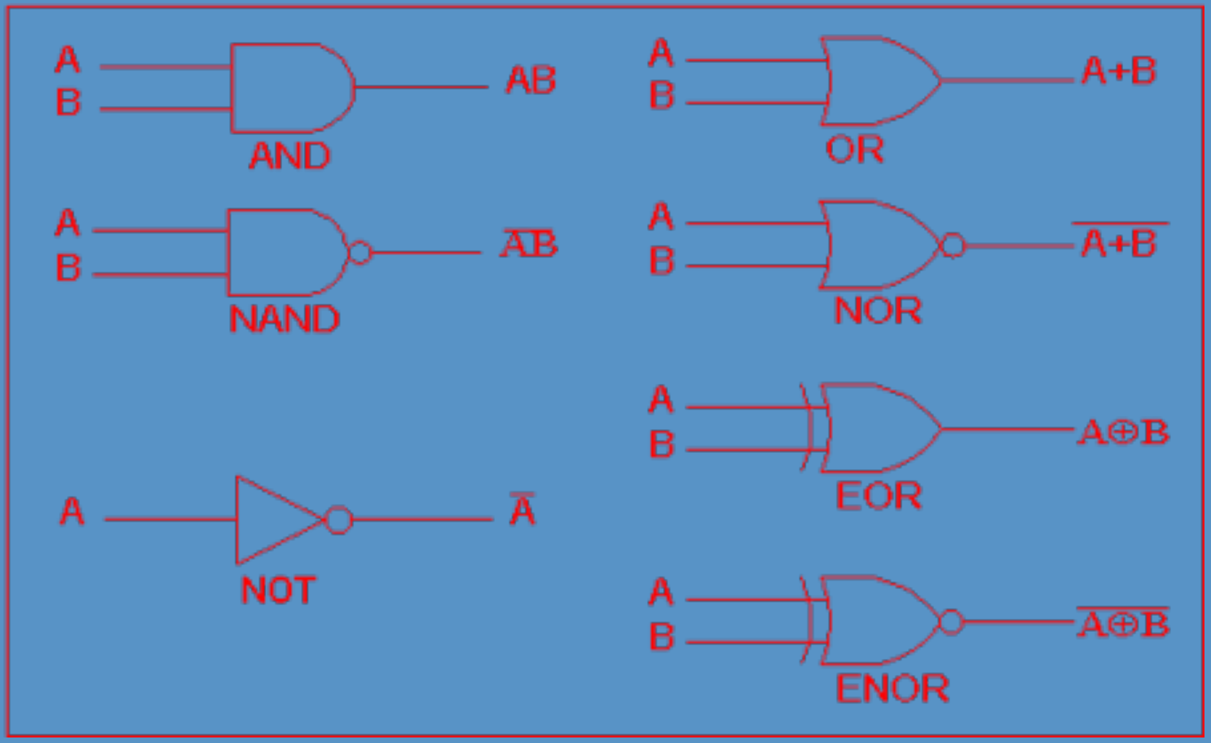
Storing Data

- Mark I in 1944 electromechanical switches
- ENIAC in 1946 electronic switches
- both designs used decimal representation of numbers

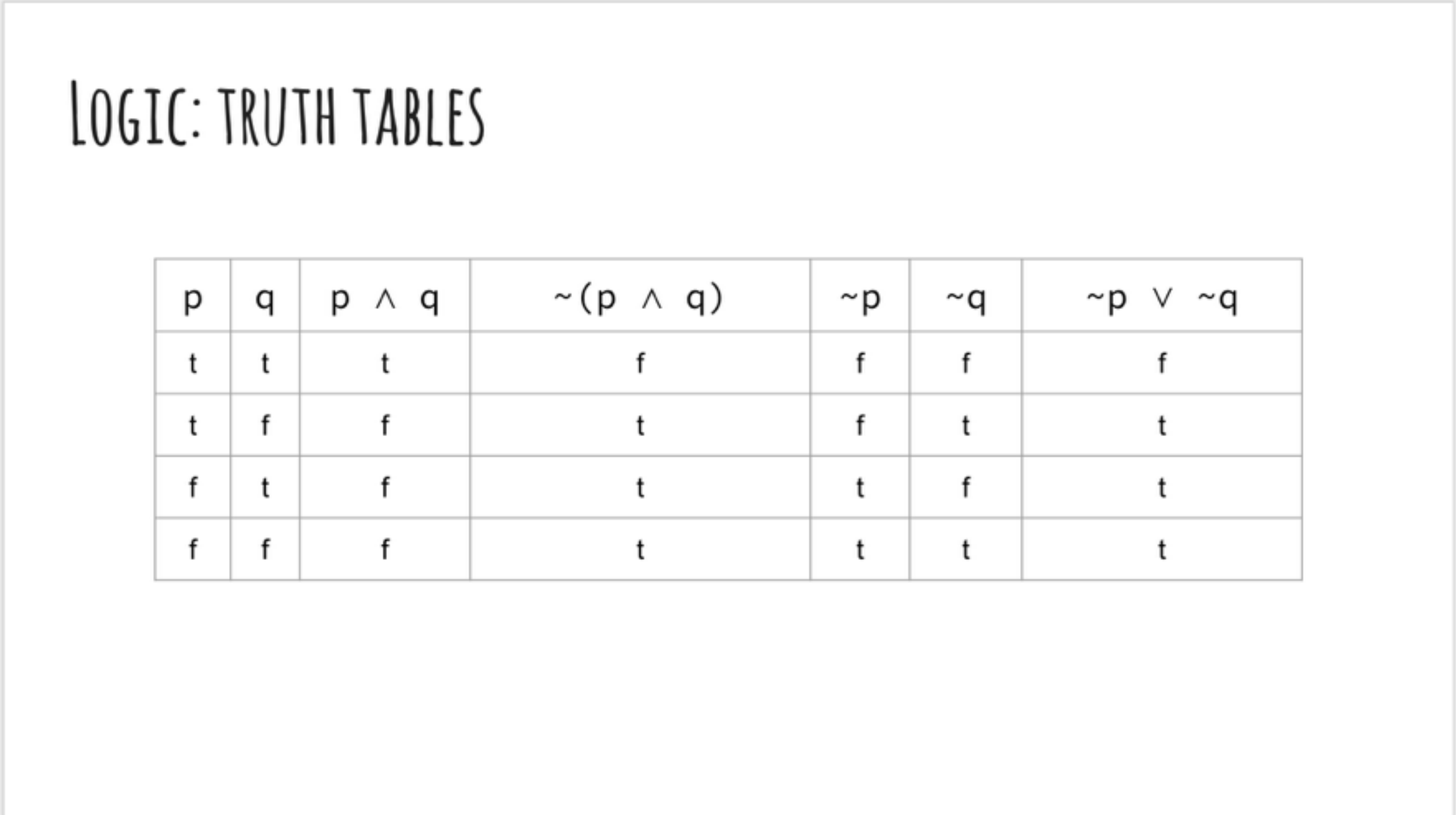


Logic Gates

- Shortly after, John von Neumann proposes binary numbers would simplify design
- on/off switches = 1’s/0’s in binary
- Binary arithmetic modeled after symbolic logic (George Boole, 1854).



Recap: Discrete Math



LOGIC: TRUTH TABLES

p	q	$p \wedge q$	$\sim(p \wedge q)$	$\sim p$	$\sim q$	$\sim p \vee \sim q$
t	t	t	f	f	f	f
t	f	f	t	f	t	t
f	t	f	t	t	f	t
f	f	f	t	t	t	t

Base 10

Decimal

Base 10

- digits = {0,1,2,3,4,5,6,7,8,9}
- from right to left, order of magnitude = x10

60159

Standard Notation

6 0 1 5 9

6	0	1	5	9
10000	1000	100	10	1

$$6 \times 10^{4} + 0 \times 10^{3} + 1 \times 10^{2} + 5 \times 10^{1} + 9 \times 10^{0}$$

Expanded Notation

$$6 \text{x} 10^4 + 0 \text{x} 10^3 + 1 \text{x} 10^2 + 5 \text{x} 10^1 + 9 \text{x} 10^0$$

$i=4$ $i=3$ $i=2$ $i=1$ $i=0$

$$6 \text{x} 10^4 + 0 \text{x} 10^3 + 1 \text{x} 10^2 + 5 \text{x} 10^1 + 9 \text{x} 10^0$$

$i=4$ $i=3$ $i=2$ $i=1$ $i=0$

MSD LSD

Base 2

Binary

Base 2

- bit = “binary digit”
- 8 bits = 1 “byte” or 1 “octet”
- digits = {0,1}
- from right to left, order of magnitude = x2

11001

1 1 0 0 1

1	1	0	0	1
16	8	4	2	1

1	1	0	0	1
2^4	2^3	2^2	2^1	2^0
MSB				LSB

$$1_{\times}2^{4} + 1_{\times}2^{3} + 0_{\times}2^{2} + 0_{\times}2^{1} + 1_{\times}2^{0}$$

Base 16

Hexadecimal

Base 16

- digits = {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}
- from right to left, order of magnitude = x16
- often prefixed with “0x”

1A3B

1 A 3 B

1	A	3	B
4096	256	16	1

1	A	3	B
16^3	16^2	16^1	16^0

$$1 \text{x} 16^3 + A \text{x} 16^2 + 3 \text{x} 16^1 + B \text{x} 16^0$$

$$1\,\mathrm{x}16^{3} + A\,\mathrm{x}16^{2} + 3\,\mathrm{x}16^{1} + B\,\mathrm{x}16^{0}$$

$$4096 + 266 + 48 + 11$$

$$= 4{,}421$$

Conversion

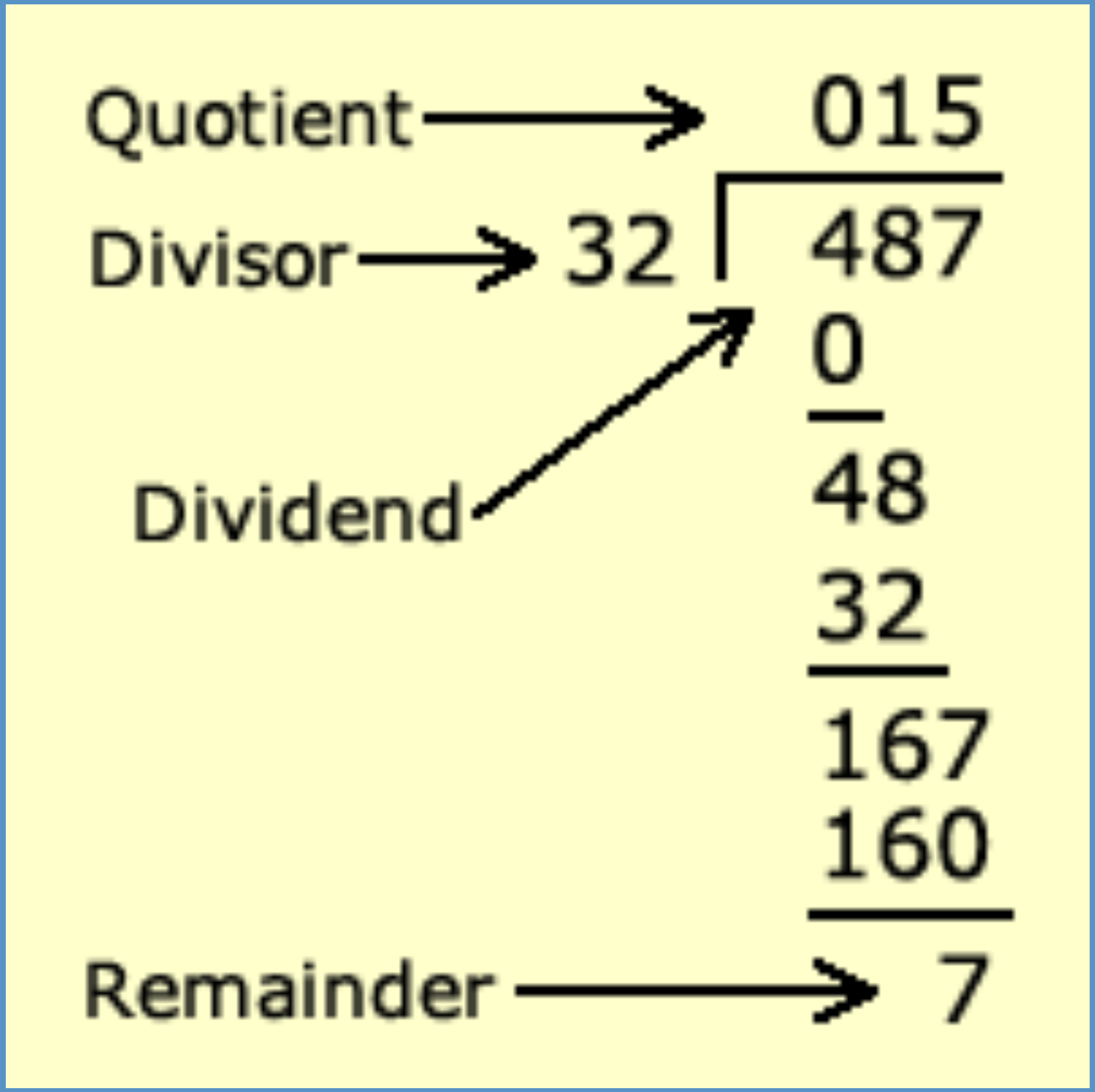
Decimal to Other Base

Algorithm:

```
while(quotient != 0)
  divide by base
  remainder -> $i^{th}$ digit
```

read digits in reverse

- easy to forget this step



Decimal to Binary

- 76 to base 2

76 / 2 = 38 R 0

38 / 2 = 19 R 0

19 / 2 = 9 R 1

9 / 2 = 4 R 1

4 / 2 = 2 R 0

2 / 2 = 1 R 0

1 / 2 = 0 R 1

Decimal to Binary

- 76 to base 2

76 / 2 = 38 R **0**

38 / 2 = 19 R **0**

19 / 2 = 9 R **1**

9 / 2 = 4 R **1**

4 / 2 = 2 R **0**

2 / 2 = 1 R **0**

1 / 2 = 0 R **1**

= 1001100

Problem

- Convert 235 from decimal to binary

235 / 2 = 117 R 1
117 / 2 = 58 R 1
58 / 2 = 29 R 0
29 / 2 = 14 R 1
14 / 2 = 7 R 0
7 / 2 = 3 R 1
3 / 2 = 1 R 1
1 / 2 = 0 R 1

=11101011

Other Base to Decimal

Algorithm:

```
sum = 0
iterate from LSD to MSD
  multiply $i^{th}$ digit by $base^i$
  add to sum
```

Example: 1A3B

Other Base to Decimal

Algorithm:

```
sum = 0
iterate from LSD to MSD
  multiply $i^{th}$ digit by $base^i$
  add to sum
```

Example: 1A3B

sum = 0

Other Base to Decimal

```
Algorithm:
sum = 0
iterate from LSD to MSD
  multiply $i^{th}$ digit by $base^i$
  add to sum
```

Example: 1A3**B** $(11*16^0) = 11*1 = 11$

sum = 0

Other Base to Decimal

```
Algorithm:
sum = 0
iterate from LSD to MSD
  multiply $i^{th}$ digit by $base^i$
  add to sum
```

Example: 1A**3**B (3*16^1) = 3*16 = 48

sum = 11

Other Base to Decimal

```
Algorithm:
sum = 0
iterate from LSD to MSD
  multiply $i^{th}$ digit by $base^i$
  add to sum
```

Example: 1**A**3B $(10*16^2) = 10*256 = 2560$

sum = 59

Other Base to Decimal

Algorithm:

```
sum = 0
iterate from LSD to MSD
  multiply $i^{th}$ digit by $base^i$
  add to sum
```

Example: **1**A3B $(1*16^3) = 1*4096 = 4096$

sum = 2619

Other Base to Decimal

Algorithm:

```
sum = 0
iterate from LSD to MSD
  multiply $i^{th}$ digit by $base^{i}$
  add to sum
```

Example: 1A3B

sum = 6715

Another way: Horner’s Rule

Algorithm:

```
result = 0
iterate from MSD to LSD
  result = digit + base x result
```

Example: 1A3B

Another way: Horner’s Rule

Algorithm:

```
result = 0
iterate from MSD to LSD
  result = digit + base x result
```

Example: 1A3B

result = 0

Another way: Horner’s Rule

Algorithm:

```
result = 0
iterate from MSD to LSD
  result = digit + base x result
```

Example: **1**A3B

result = **1** + 16 * 0

Another way: Horner’s Rule

Algorithm:

```
result = 0
iterate from MSD to LSD
  result = digit + base x result
```

Example: **1**A3B

result = 1

Another way: Horner’s Rule

```
Algorithm:
result = 0
iterate from MSD to LSD
  result = digit + base x result
```

Example: 1**A**3B

result = **10** + 16 * 1

Another way: Horner’s Rule

```
Algorithm:
result = 0
iterate from MSD to LSD
  result = digit + base x result
```

Example: 1**A**3B

result = 26

Another way: Horner’s Rule

Algorithm:

```
result = 0
iterate from MSD to LSD
  result = digit + base x result
```

Example: 1A**3**B

result = **3** + 16 x 26

Another way: Horner’s Rule

Algorithm:

```
result = 0
iterate from MSD to LSD
  result = digit + base x result
```

Example: 1A**3**B

result = 419

Another way: Horner’s Rule

Algorithm:

```
result = 0
iterate from MSD to LSD
  result = digit + base x result
```

Example: 1A3**B**

result = **11** + 16 * 419

Another way: Horner’s Rule

Algorithm:

```
result = 0
iterate from MSD to LSD
  result = digit + base x result
```

Example: 1A3**B**

result = 6715

Shortcut: Binary to Decimal

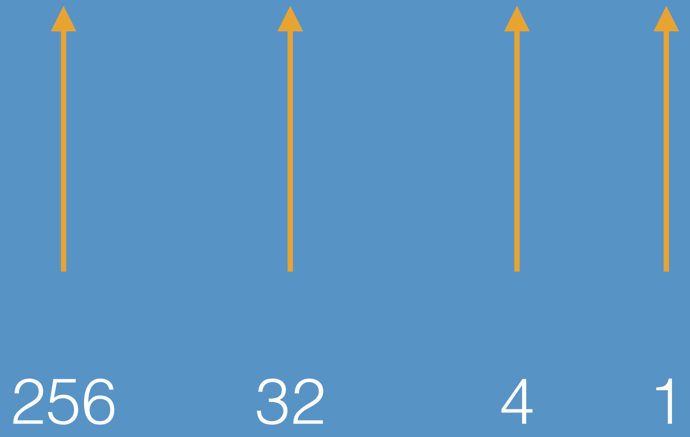
Example: 100100101

Shortcut: Binary to Decimal

Example: 1 0 0 1 0 0 1 0 1

Shortcut: Binary to Decimal

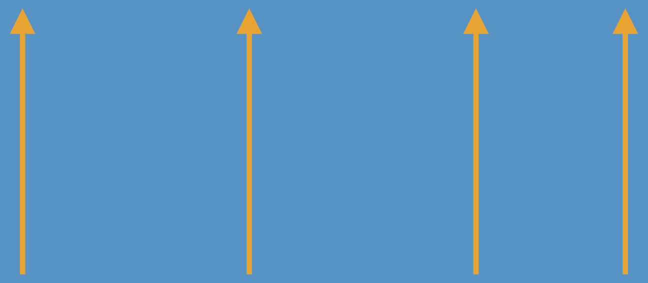
Example: 1 0 0 1 0 0 1 0 1



256 32 4 1

Shortcut: Binary to Decimal

Example: 1 0 0 1 0 0 1 0 1



256 + 32 + 4 + 1 = 293

Problems

- Convert 1001101 from binary to decimal

= 1+4+8+64

= 77

- Convert 0x04A3 from hexadecimal to decimal

= 3 + 10*(16) + 4*(256)

= 1,187

Fun with Binary

Operations

Addition

Subtraction

Bitwise Operations

Addition

$$\begin{array}{cccc} & 1 & 2 & 3 \\ + & & 5 & 4 \\ \cline{2-4} & 1 & 7 & 7 \end{array}$$

$$\begin{array}{cccc} & & {\scriptstyle 1} & \\ & 1 & 2 & 7 \\ + & & 5 & 4 \\ \cline{2-4} & 1 & 8 & 1 \end{array}$$

Addition

$$\begin{array}{rcccccc}
 & & 1 & 1 & 1 & 1 & \\
 & 1 & 0 & 0 & 1 & 0 & 1 \\
+ & & & 1 & 0 & 1 & 1 \\
\hline
 & 1 & 1 & 0 & 0 & 0 & 0
\end{array}
\qquad
\begin{array}{r}
 \\
37 \\
11 \\
48
\end{array}$$

Operations

Addition

Subtraction

Bitwise Operations

Subtraction

$$\begin{array}{r} 1\overset{6}{\cancel{7}}{}^{1}3 \\ -\ \ 54 \\ \hline 119 \end{array}$$

Signed Numbers

unsigned	1001	= 9
	0000001001	= 9
signed	0000001001	= 9
	1001	= a negative…

“sign bits”

Two’s complement

0100111 = 39

↑

+

invert aka ‘flip’ 1011000

add 1 1011001 = -39

↑

-

Subtraction

	..01101	13
+	..11011	-5
	..01000	8

Operations

Addition

Subtraction

Bitwise Operations

Bitwise Operations

AND

$$\begin{array}{r}1010\\ \underline{0110}\\ 0010\end{array}$$

OR

$$\begin{array}{r}1010\\ \underline{0110}\\ 1110\end{array}$$

XOR

$$\begin{array}{r}1010\\ \underline{0110}\\ 1100\end{array}$$

NOT

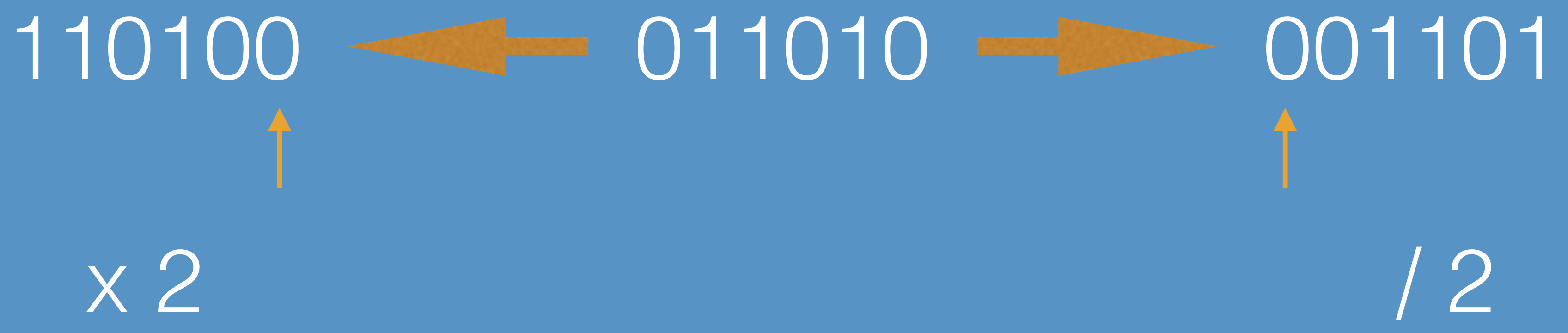
$$\begin{array}{r}\underline{0110}\\ 1001\end{array}$$

Bitwise Operations

```
// AND
System.out.println(10 & 6);  // 2

// OR
System.out.println(10 | 6);  // 14

// XOR
System.out.println(10 ^ 6);  // 12

// NOT
System.out.println(~6);  // -7
```

Bitwise Operations

Logical Shifts

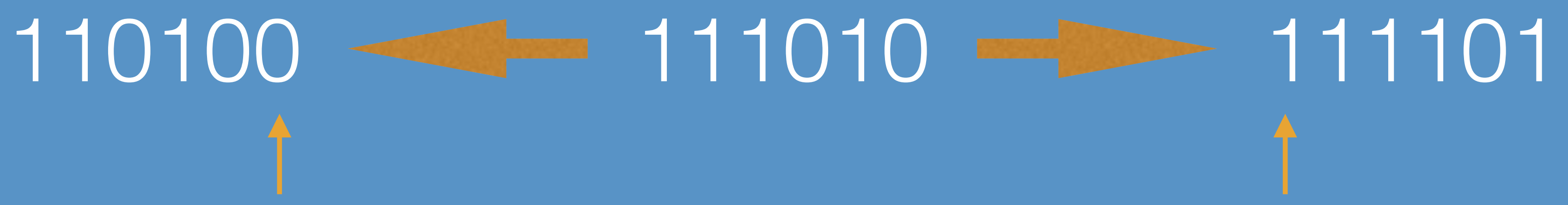


110100 ← 011010 → 001101

x 2

/ 2

Arithmetic Shifts



110100 ← 111010 → 111101

Bitwise Operations

```
byte i = -8;



// logical shifts
System.out.println(i >>> 1);  // 2147483644


// arithmetic shift
System.out.println(i << 1);   // -16
System.out.println(i >> 1);   // 4
```

Recap: Palindrome

Determine whether a string is a palindrome

- “racecar” -> true
- “racecare” -> false

```
private static boolean isPalindrome(String s) {
  int l = 0;
  int r = s.length() - 1;

  ...
}
```

```
private static boolean isPalindrome(String s) {
  int l = 0;
  int r = s.length() - 1;

  while(l < r) {
    ...
  }

  ...
}
```

```
private static boolean isPalindrome(String s) {
  int l = 0;
  int r = s.length() - 1;

  while(l < r) {
    if(s.charAt(l) != s.charAt(r)) {
      return false;
    }

    ...
  }

  ...
}
```

```
private static boolean isPalindrome(String s) {
  int l = 0;
  int r = s.length() - 1;

  while(l < r) {
    if(s.charAt(l) != s.charAt(r)) {
      return false;
    }

    l++;
    r--;
  }

  ...
}
```

```java
private static boolean isPalindrome(String s) {
  int l = 0;
  int r = s.length() - 1;

  while(l < r) {
    if(s.charAt(l) != s.charAt(r)) {
      return false;
    }

    l++;
    r--;
  }

  return true;
}
```

Bitstring Palindrome

- Determine whether a bitstring is a palindrome
- A bit trickier…

Java Types

Java Primitives

byte

short

int

long

char

boolean

float

double

Java Primitives

byte	8 bit (1 byte) signed
short	16 bit (2 bytes) signed
int	32 bit (4 bytes) signed
long	64 bit (8 bytes) signed
char	16 bit (2 bytes) Unicode
boolean	undefined, usually 1 byte
float	32 bit (4 bytes) IEEE fp
double	64 bit (8 bytes) IEEE fp

Java Primitives

byte	min: -128	max: 127
short	min: -32,768	max: 32,767
int	min: -2^{31}	max: $2^{31}-1$
long	min: -2^{63}	max: $2^{63}-1$
char	min: 0	max: 65,535
boolean	true, false	

Java Primitives

byte	0000 0000
short	0000 0000 0000 0000
int	0000 0000 0000 0000 0000 0000 0000 0000
long	0000 0000 0000 0000 0000 0000 0000 0000
	0000 0000 0000 0000 0000 0000 0000 0000
char	0000 0000 0000 0000
boolean	0000 0000

Java Primitives

byte	0000 0000
short	0000 0000 0000 0000
int	0000 0000 0000 0000 0000 0000 0000 0000
long	**0000 0000 0000 0000 0000 0000 0000 0000**
	0000 0000 0000 0000 0000 0000 0000 0000
char	0000 0000 0000 0000
boolean	0000 0000

5 as a…

byte	0000 0101
short	0000 0000 0000 0101
int	0000 0000 0000 0000 0000 0000 0000 0101
long	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0101

Other Java types

References: undefined, usually 4 to 8 bytes

Objects: 8 bytes + sum of all fields + padding (total size must be multiple of 8)

Arrays:

- container: 8 bytes + 4 bytes for length
  - if primitives: (length) * (primitive size)
  - if objects:
    - references: (length) * (reference size)
    - objects: (object size) * (# of non-null references)

Multi-arrays:

- a multidimensional array is a set of nested arrays, so every row of a 2D array is a separate object

Problems

Assuming 4-byte references, calculate the amount of memory needed to store:

- an object of type Person
- a 5-element array of Person
- a 3 x 5 multidimensional array of Person

```
class Person {
  String firstName;  // assume Strings are char[20]
  String lastName;
  int age;
  Address home;
}

class Address {
  double lat, lng;
}
```

Binary Prefixes

B	2^{0}	string
KB	$2^{10} \sim 1{,}000$	word document, icons
MB	$2^{20} \sim 1{,}000{,}000$	pictures, images
GB	$2^{30} \sim 1{,}000{,}000{,}000$	video
TB	2^{40}	hard drive limits
PB	2^{50}	a data center’s limits OR all photos on Facebook OR daily Google traffic

Rules of Exponents

$2 * 2 * 2 = 2^3$

$2^a * 2^b = 2^{a+b}$

$2^{a+b} / 2^a = 2^b$

$4 \text{ TB} = 4 * 2^{40} = 2^2 * 2^{40} \mathbf{= 2^{42}}$

Problems

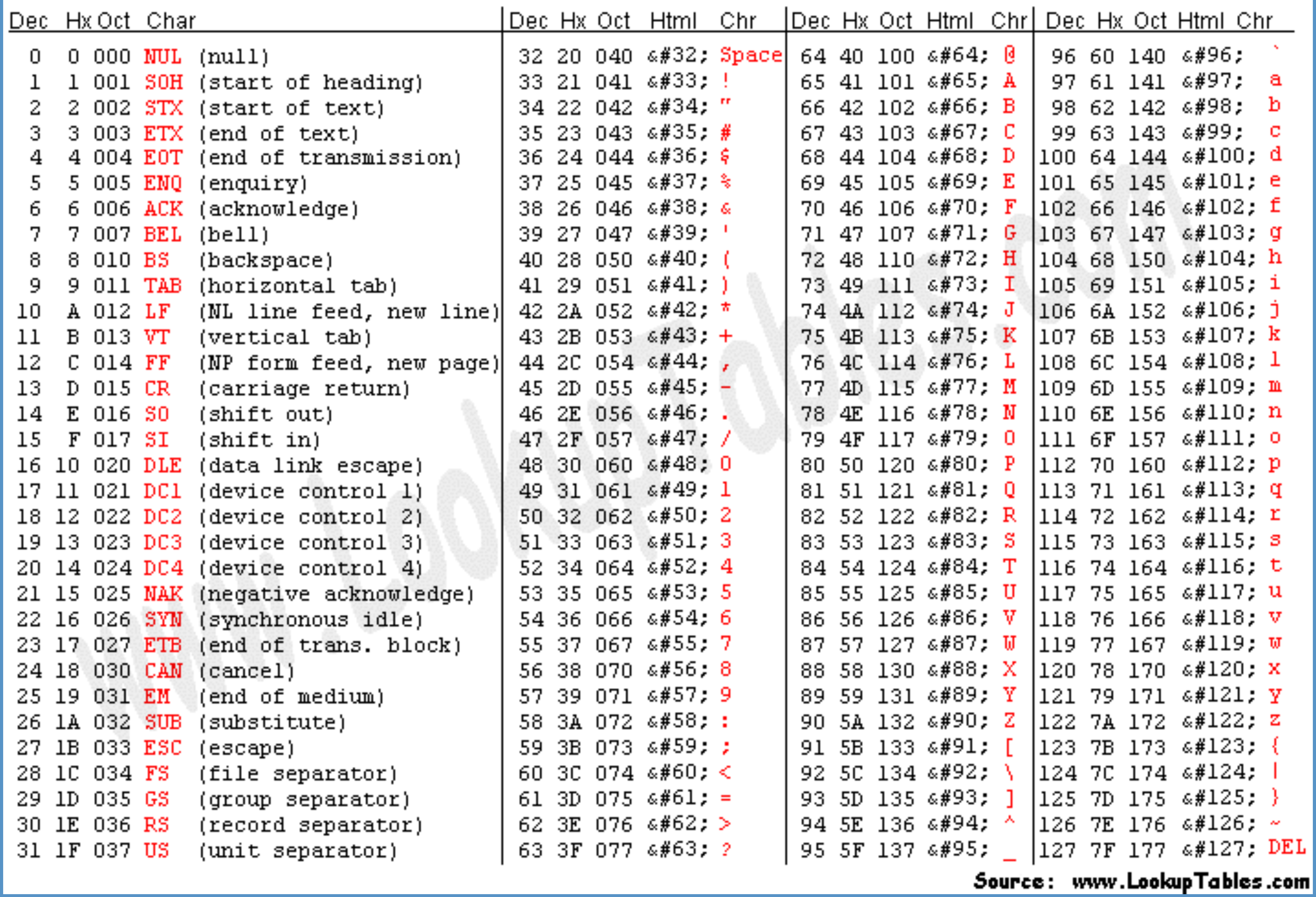
- How many 32-bit integers can be stored in 16 GB of RAM?
- How much memory to store all possible SSNs? Assume no restricted values
- How much memory to store US Yellow Pages? Assume each record in yellow pages has a first name, last name and phone number, each of which is 10 chars long. assume phone numbers are unique.
- How much memory to store a years worth of pictures? Assume 4 pictures are taken every day on average using a 1024 x 640 px camera using RGB values.

Text

Characters

```java
// up to now...
String x = "string";
x = new String("string");
x = new String(new char[]{'s','t','r','i','n','g'});


// defaults to machine setting
x = new String(new byte[]{});
x = new String(new byte[]{}, Charset.defaultCharset());


// specific
x = new String(new byte[]{}, Charset.forName("UTF-8"));
x = new String(new byte[]{}, "UTF-8");
```

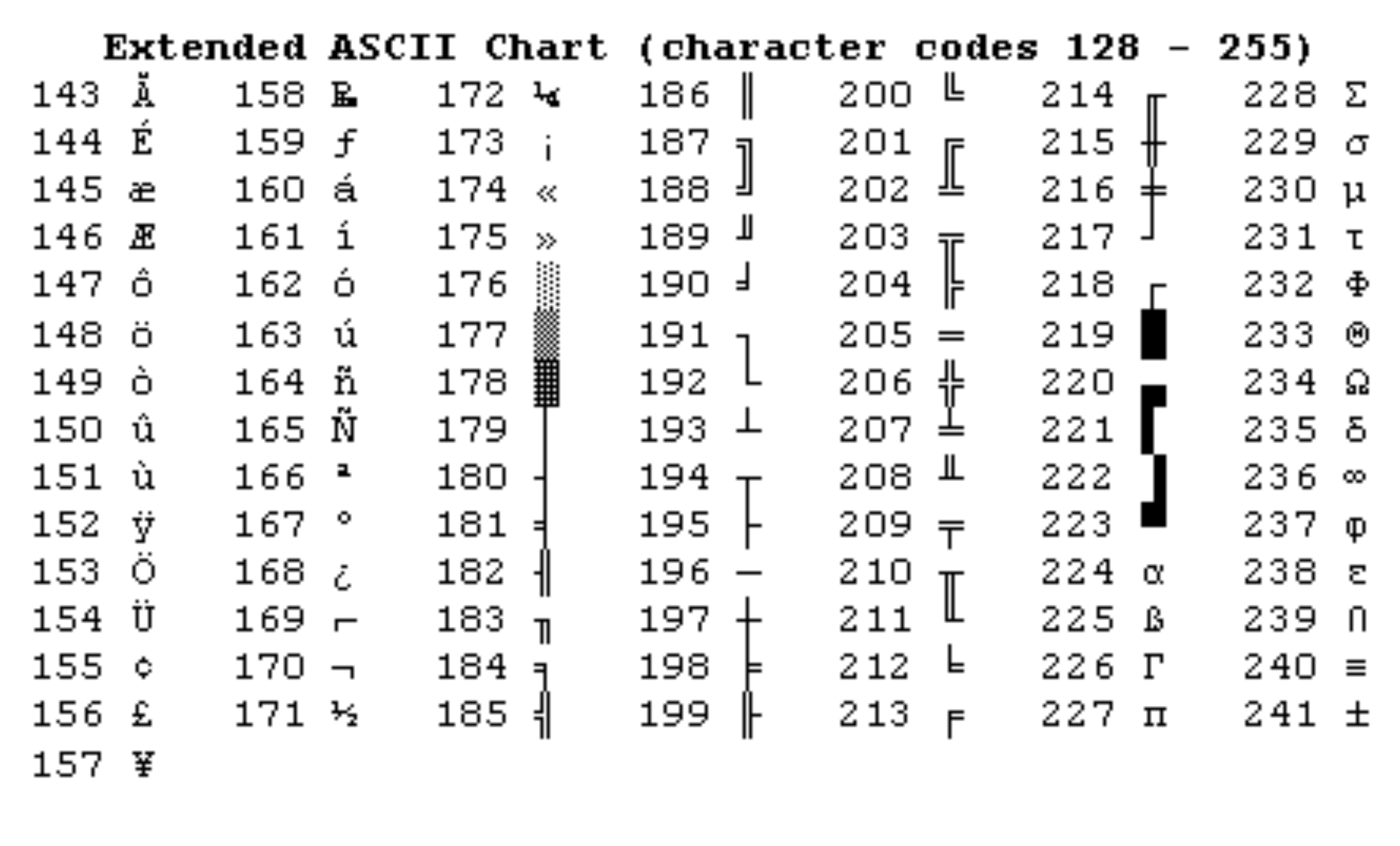
ASCII (7-bit)



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

Source: www.LookupTables.com

Extended ASCII (8-bit)



Extended ASCII Chart (character codes 128 - 255)

143	Å	158	₧	172	¼	186	║	200	╚	214	╓	228	Σ
144	É	159	ƒ	173	¡	187	╗	201	╔	215	╫	229	σ
145	æ	160	á	174	«	188	╝	202	╩	216	╪	230	µ
146	Æ	161	í	175	»	189	╜	203	╦	217	┘	231	τ
147	ô	162	ó	176	░	190	╛	204	╠	218	┌	232	Φ
148	ö	163	ú	177	▒	191	┐	205	═	219	█	233	Θ
149	ò	164	ñ	178	▓	192	└	206	╬	220	▄	234	Ω
150	û	165	Ñ	179	│	193	┴	207	╧	221	▌	235	δ
151	ù	166	ª	180	┤	194	┬	208	╨	222	▐	236	∞
152	ÿ	167	º	181	╡	195	├	209	╤	223	▀	237	φ
153	Ö	168	¿	182	╢	196	─	210	╥	224	α	238	ε
154	Ü	169	⌐	183	╖	197	┼	211	╙	225	ß	239	∩
155	¢	170	¬	184	╕	198	╞	212	╘	226	Γ	240	≡
156	£	171	½	185	╣	199	╟	213	╒	227	π	241	±
157	¥												

But what about?

你好世界！

Добрый вечер!

السلام عليكم !

😂

Unicode

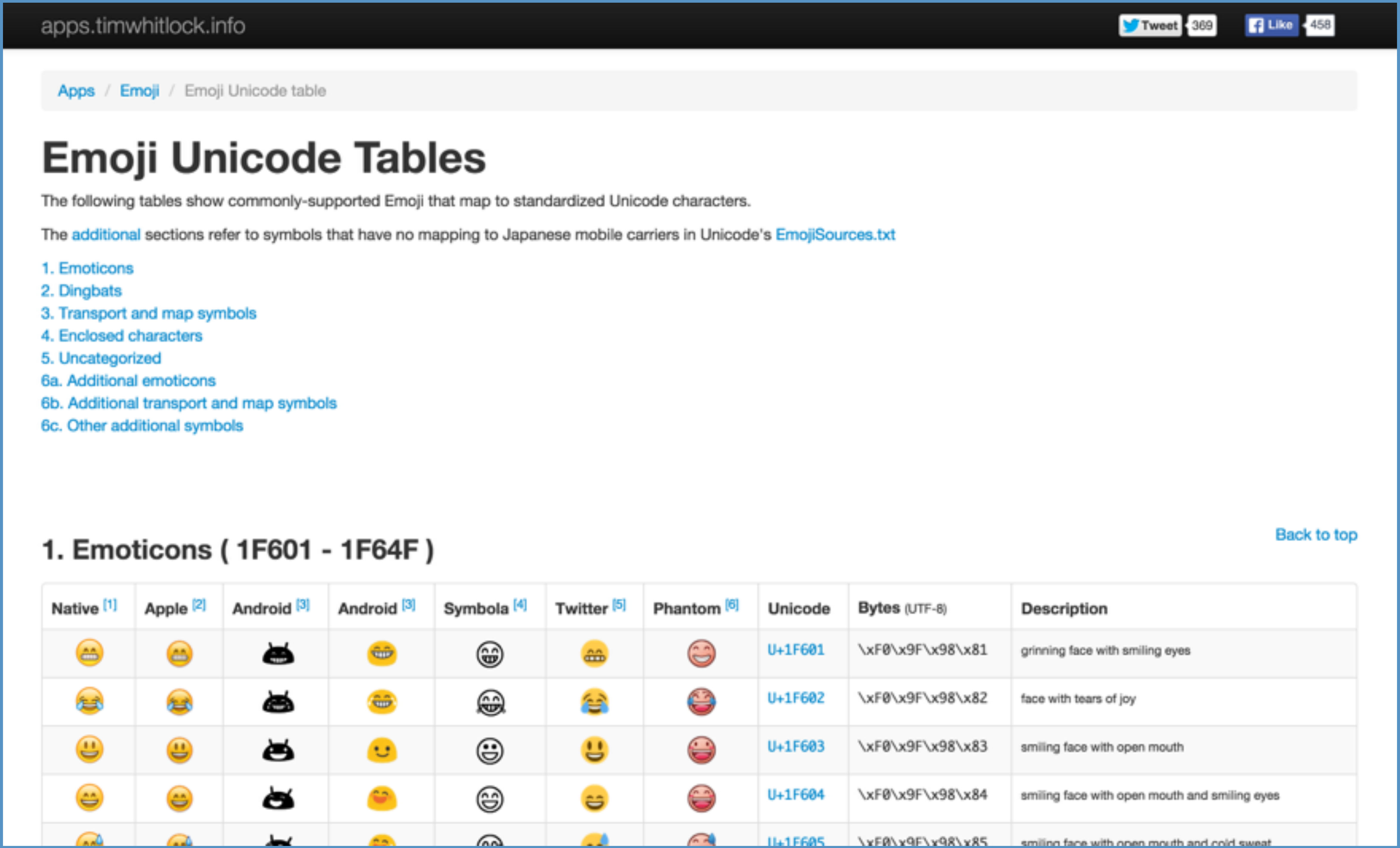
Unicode

- 3-byte character definition, includes other languages, etc.
- 17 planes, each with 65,536 (= 2^{16}) code points
- = 1,114,112 = $7F_{hex}$

UTF-8, UTF-16, UTF-32

- encodings capable of encoding Unicode characters, or code points. UTF-8 and UTF-16 are variable-length, while UTF-32 are fixed-length encodings. They use 8-bit, 16-bit, and 32-bit code units, respectively.

Unicode



apps.timwhitlock.info

Apps / Emoji / Emoji Unicode table

Emoji Unicode Tables

The following tables show commonly-supported Emoji that map to standardized Unicode characters.

The additional sections refer to symbols that have no mapping to Japanese mobile carriers in Unicode's EmojiSources.txt

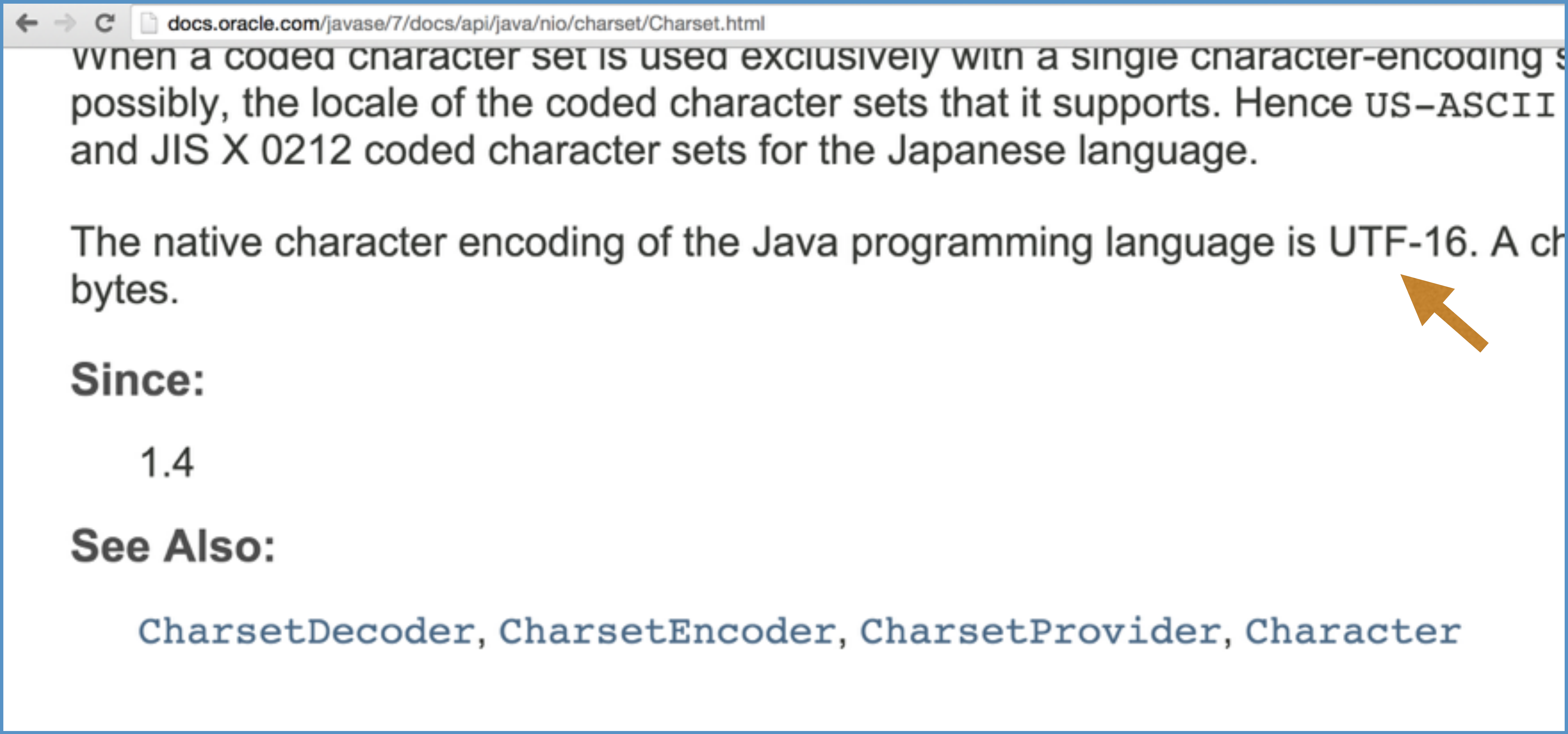
1. Emoticons
2. Dingbats
3. Transport and map symbols
4. Enclosed characters
5. Uncategorized
6a. Additional emoticons
6b. Additional transport and map symbols
6c. Other additional symbols

Back to top

1. Emoticons (1F601 - 1F64F)

Native [1]	Apple [2]	Android [3]	Android [3]	Symbola [4]	Twitter [5]	Phantom [6]	Unicode	Bytes (UTF-8)	Description
😁	😁	😁	😁	😁	😁	😁	U+1F601	\xF0\x9F\x98\x81	grinning face with smiling eyes
😂	😂	😂	😂	😂	😂	😂	U+1F602	\xF0\x9F\x98\x82	face with tears of joy
😃	😃	😃	😃	😃	😃	😃	U+1F603	\xF0\x9F\x98\x83	smiling face with open mouth
😄	😄	😄	😄	😄	😄	😄	U+1F604	\xF0\x9F\x98\x84	smiling face with open mouth and smiling eyes
😅	😅	😅	😅	😅	😅	😅	U+1F605	\xF0\x9F\x98\x85	smiling face with open mouth and cold sweat

Java



docs.oracle.com/javase/7/docs/api/java/nio/charset/Charset.html

When a coded character set is used exclusively with a single character-encoding s
possibly, the locale of the coded character sets that it supports. Hence `US-ASCII`
and JIS X 0212 coded character sets for the Japanese language.

The native character encoding of the Java programming language is UTF-16. A ch
bytes.

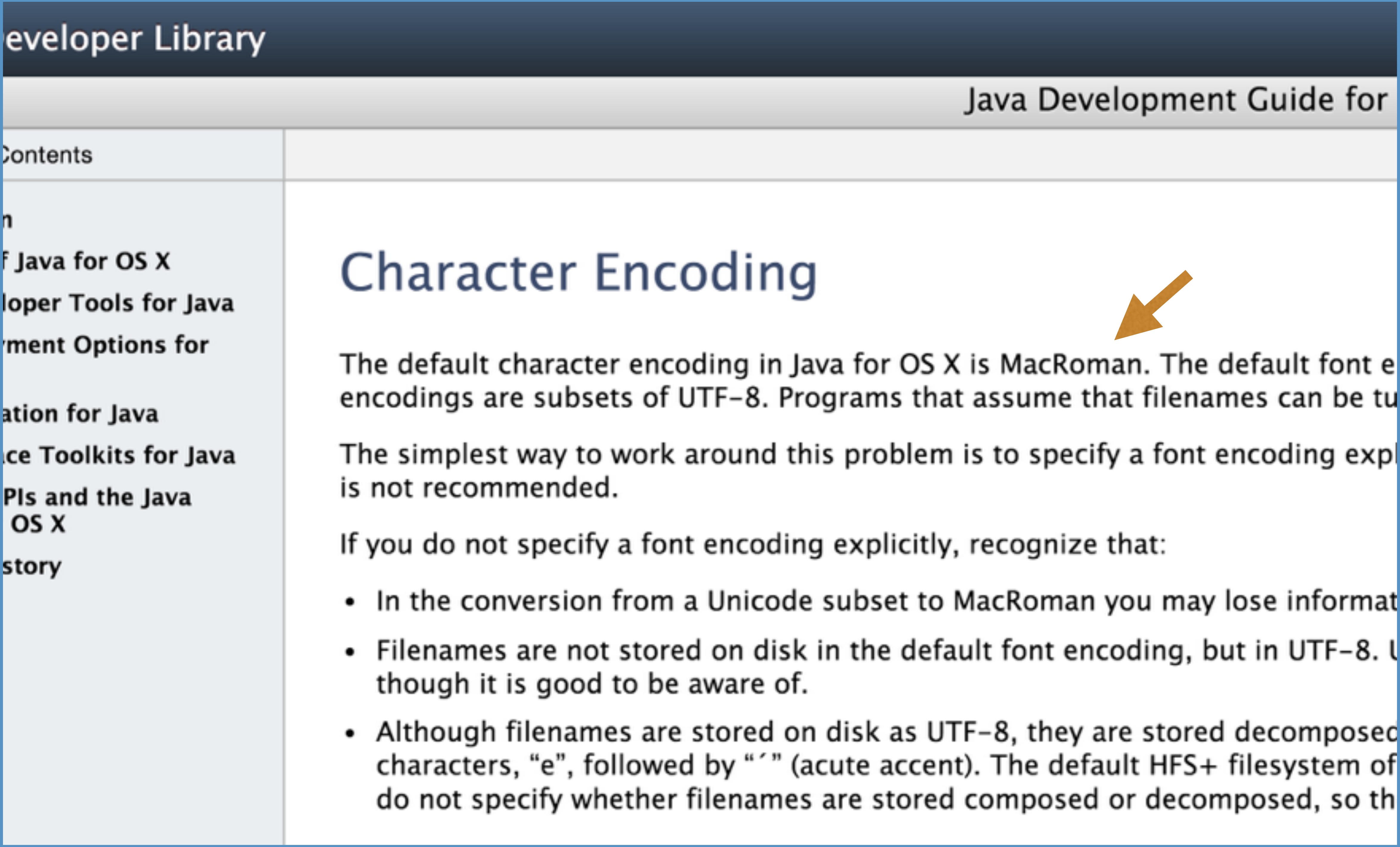
Since:

1.4

See Also:

`CharsetDecoder`, `CharsetEncoder`, `CharsetProvider`, `Character`

?!



eveloper Library

Java Development Guide for

:ontents

n
f Java for OS X
loper Tools for Java
ment Options for
ation for Java
ce Toolkits for Java
PIs and the Java
OS X
story

Character Encoding

The default character encoding in Java for OS X is MacRoman. The default font e
encodings are subsets of UTF-8. Programs that assume that filenames can be tu

The simplest way to work around this problem is to specify a font encoding expl
is not recommended.

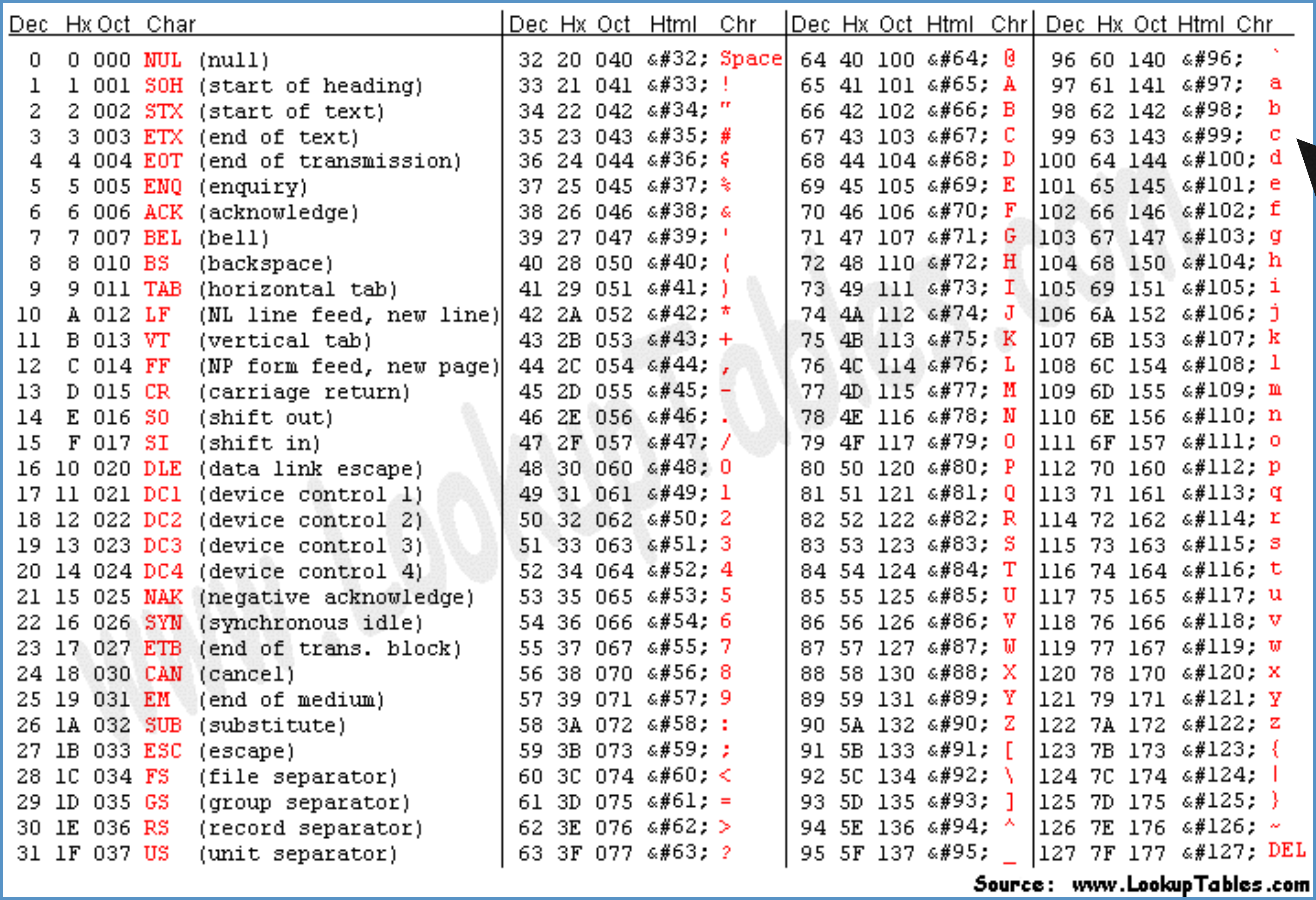
If you do not specify a font encoding explicitly, recognize that:

- In the conversion from a Unicode subset to MacRoman you may lose informat
- Filenames are not stored on disk in the default font encoding, but in UTF-8. U
though it is good to be aware of.
- Although filenames are stored on disk as UTF-8, they are stored decomposed
characters, "e", followed by "´" (acute accent). The default HFS+ filesystem of
do not specify whether filenames are stored composed or decomposed, so th

Java

```
System.out.println((int)'c');  // 99

System.out.println((int)'界'); // 30028
```

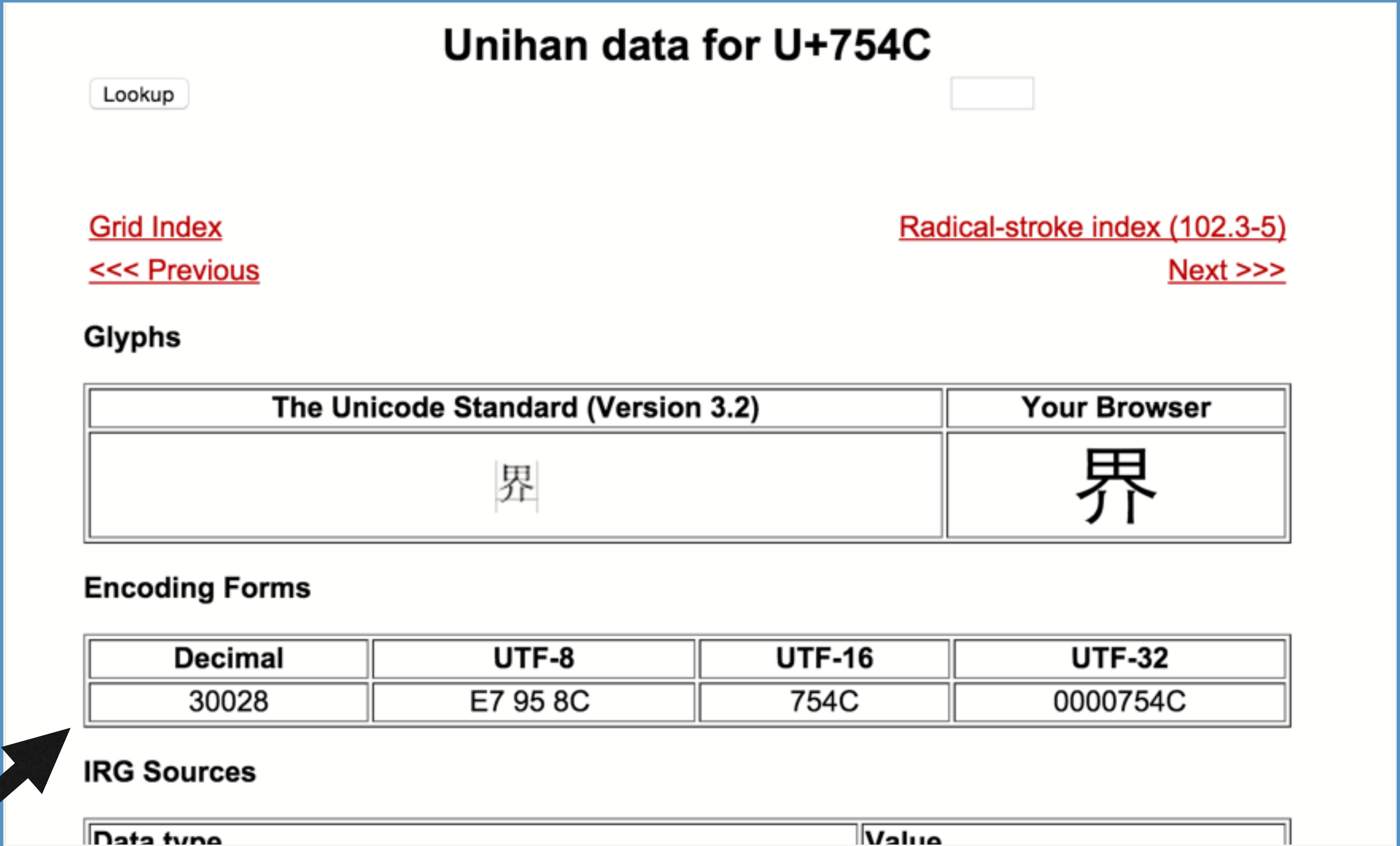
ASCII



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

Source: www.LookupTables.com

Unicode



Unihan data for U+754C

Grid Index

Radical-stroke index (102.3-5)

<<< Previous

Next >>>

Glyphs

The Unicode Standard (Version 3.2)	Your Browser
界	界

Encoding Forms

Decimal	UTF-8	UTF-16	UTF-32
30028	E7 95 8C	754C	0000754C

IRG Sources

Data type	Value

Problem Solving

- visualizing the problem is very crucial
  - before tackling a problem, draw it out
- solve the general problem first; then consider edge cases

- Identify your assumptions
- Confirm your understanding
- Isolate your trials
- Iterate accordingly

References

- http://engineering.mit.edu/ask/what%E2%80%99s-difference-between-ac-and-dc
- http://nookkin.com/articles/computer-science/why-computers-use-binary.ndoc
- http://electronics.stackexchange.com/questions/5949/is-it-possible-for-a-computer-to-use-ac-power
- http://www.csudh.edu/oliver/smt310-handouts/computer/computer.htm
- http://www.ee.surrey.ac.uk/Projects/CAL/digital-logic/gatesfunc/index.html
- https://docs.oracle.com/javase/tutorial/java/nutsandbolts/op3.html
- http://docs.oracle.com/javase/7/docs/api/java/nio/charset/Charset.html
- http://www.joelonsoftware.com/articles/Unicode.html
- http://kunststube.net/encoding/

Homework

- Convert 134 and 562 to binary, add them, convert the sum back to decimal. Confirm your final answer is 696 (= 134 + 562) **[1]**
- Do the same for 51 and -8. Confirm your final answer is 43 (= 51 - 8) **[2]**
- What's 52 in octal (base 8)? **[1]**
- Read about 'od' or 'hexdump' and how they read/display bytes. Use either one and play around with the options to view the human-readable information inside 1) any downloaded Facebook photo, 2) any Java source file, and 3) any Java class file **[1]**
- Using your preferred text editor, save this text "Привет мир" using Cyrillic (ISO 8859-5). Open the file using Chrome. Play with different encodings and observe the difference. **[1]**
- Given an integer, write code to print out its bitstring. **[2]**
- Given an integer, write code to count the number of 1s in its bitstring. **[2]**
- Given an integer, write code to determine whether its bitstring is a palindrome. **[3]**
- Implement Lempel–Ziv–Welch compression (link 1, link 2) **[5]**

Exit Ticket

- What’s the largest number that can fit in 8 bits?
- Explain why the max value of a byte in Java is 127 and not the number you calculated.