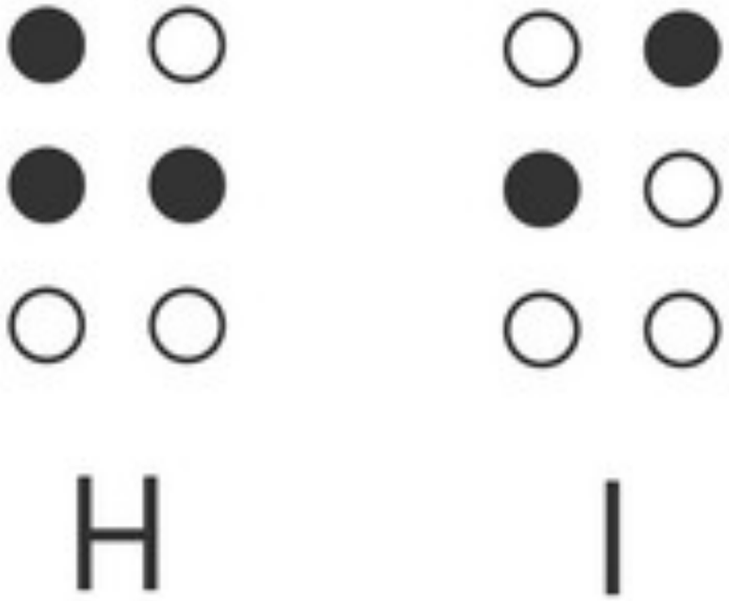
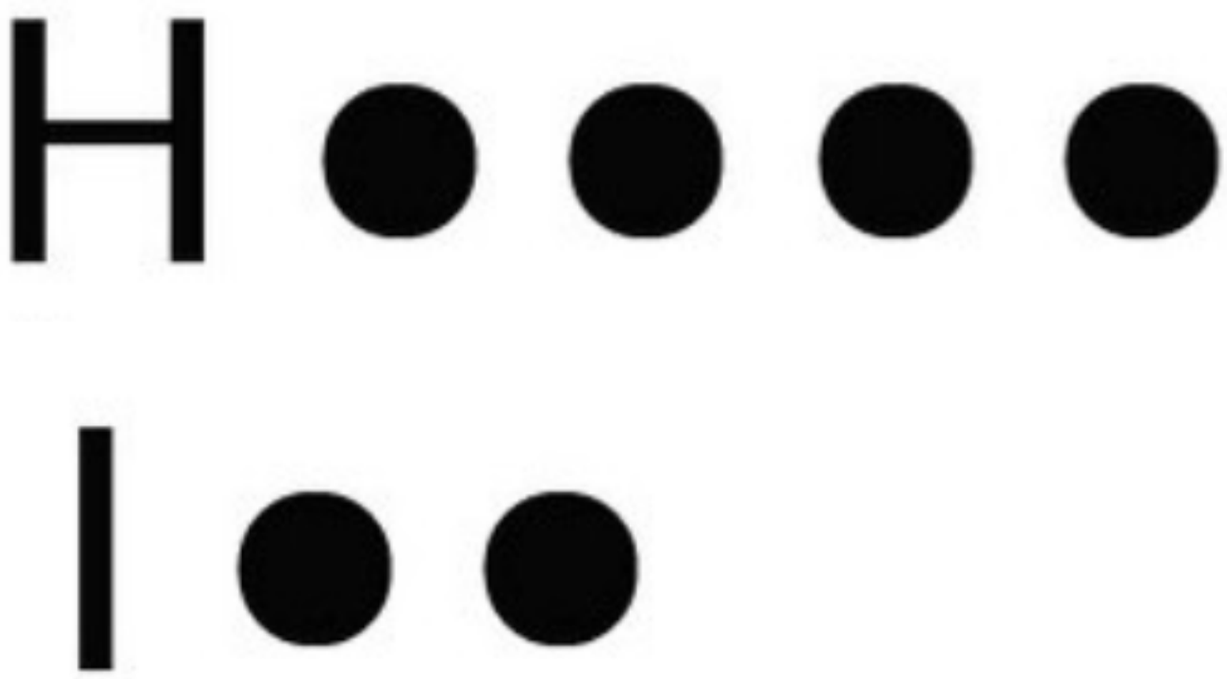
What does this binary code mean?

0	1	0	0	0	0	0	1

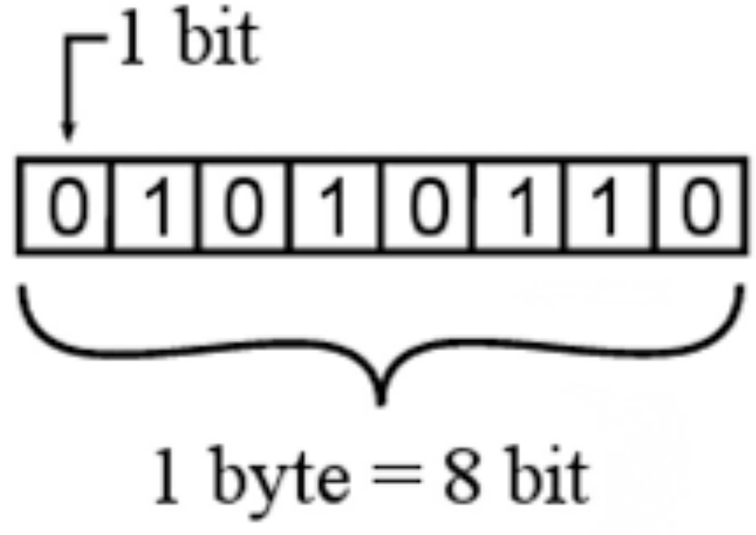


H I



01101000
01101001

Bytes



Single bits rarely handled

Smallest addressable ‘cell’ is a byte - 8 bits
e.g. 10011011

4GB of RAM is 4 billion bytes

Integers
typically 2 bytes 16 bits

What is the range for a 16 bit int?
for a 16 bit signed int?

$2^N - 1$ -2n-1 to 2n-1 – 1

0 to 65535 - 32768 to 32767

Integers

8 bit integers rare

Most high-level languages provide 2 or more formats

Short integer (16 bits)

Long integer (32 bits)

Up to you as the programmer to decide what bit-length representation scheme you need for your integers.

What would you use to represent a range from 0-124?

Bigger integers

16 bit integer ~ 64,000 (64k)

32 bit integer ~ 4,000,000,000 (4G)

2^{32} = 4,294,967,296

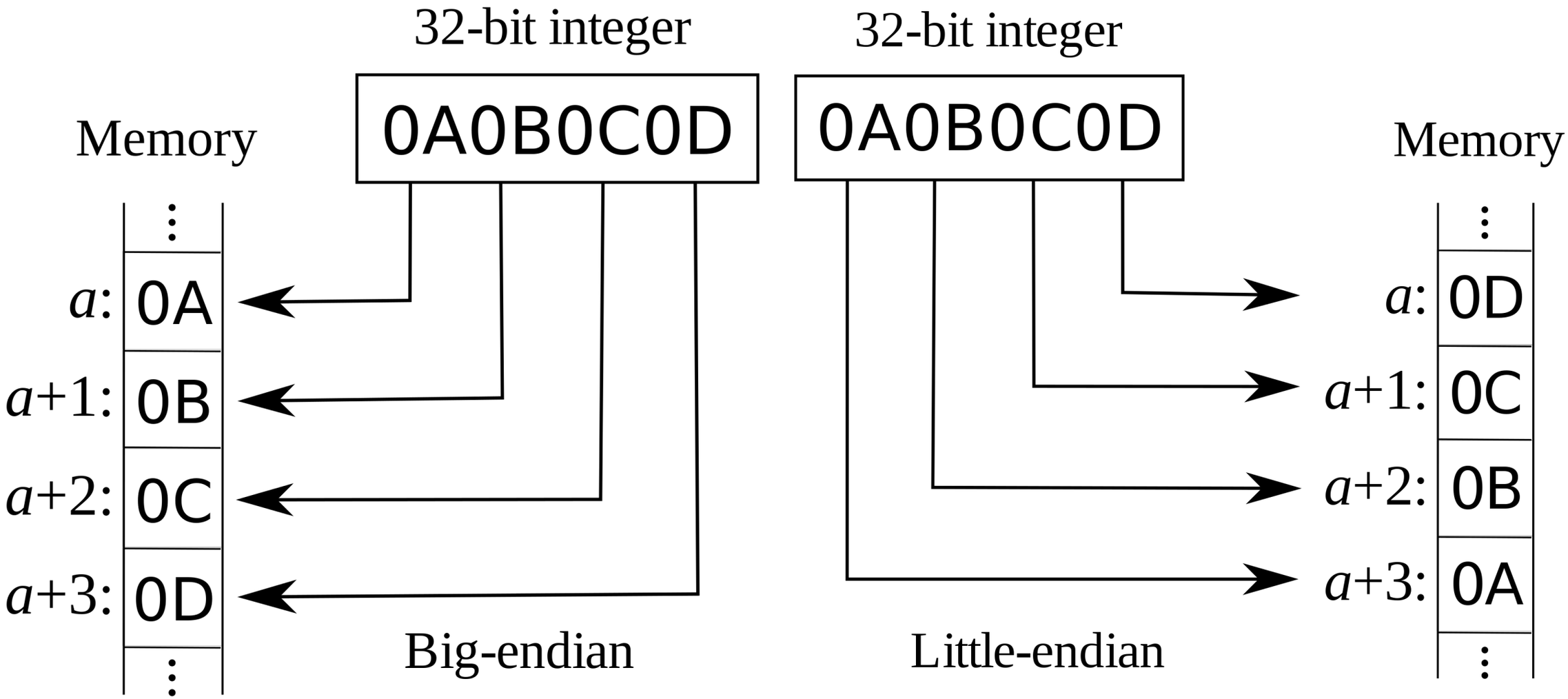
64 bit integer

approx. 1.6 x 10^{19} ,(19 decimal digits)

Endianness

Big Endian Most significant bytes in lower addresses

Little Endian Most significant bytes in higher addresses



Aside on Giga, Tera etc.

Name	Symbol	10^n	Decimal
yotta	Y	10^{24}	1,000,000,000,000,000,000,000,000
zetta	Z	10^{21}	1,000,000,000,000,000,000,000
exa	E	10^{18}	1,000,000,000,000,000,000
peta	P	10^{15}	1,000,000,000,000,000
tera	T	10^{12}	1,000,000,000,000
giga	G	10^{9}	1,000,000,000
mega	M	10^{6}	1,000,000
kilo	k	10^{3}	1,000
hecto	h	10^{2}	100
deca	da	10^{1}	10
		10^{0}	1

Caution: 1kB = 2^{10} = 1,024
1MB = 2^{20} = $1{,}024^2$

How many bits do you need?

How many bits to represent the day of the week?

How many candles would you need to represent 100 in binary on a birthday cake?

How many bits to represent the number of people in the world? a. 32 bits. b. 64 bits?

How many bits to represent the number of stars in the universe? (i.e. circa 100 billion)

a. 64 bits or 128 bits?

64 bits = 2^{64} = 18,446,744,073,709,551,616

128 bits = 2^{128} = 340,282,366,920,938,463,463,374,607,431,768,211,456

Floating point

Remember scientific notation?

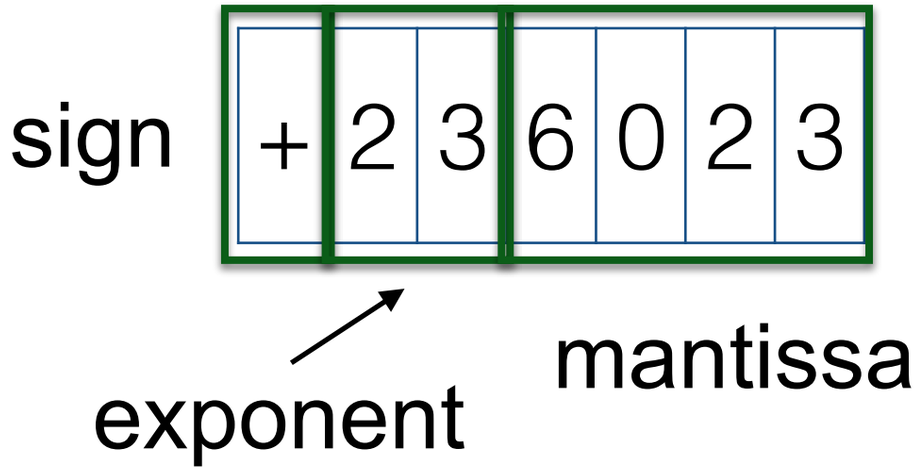
For numbers with a decimal point (or fraction)

Or for very large numbers

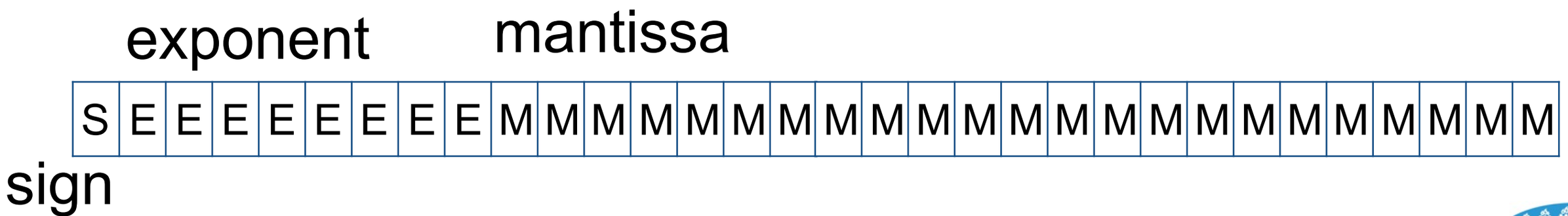
Big numbers

Speed of light 1.08×10^9 km/hr

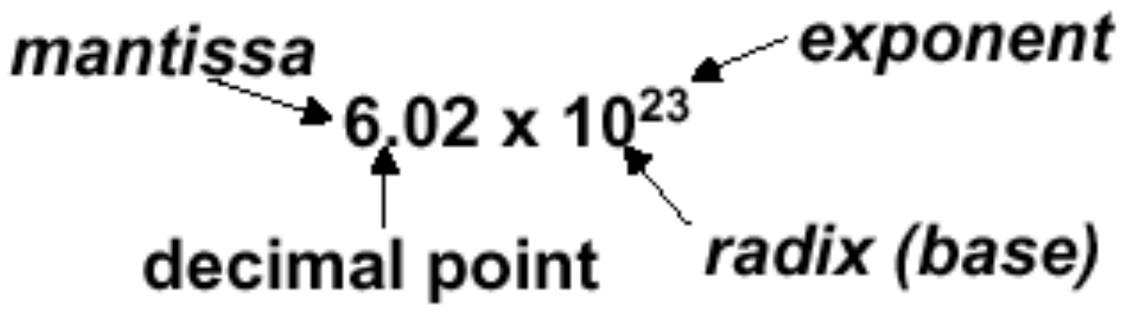
Avogadro's constant 6.023×10^{23}

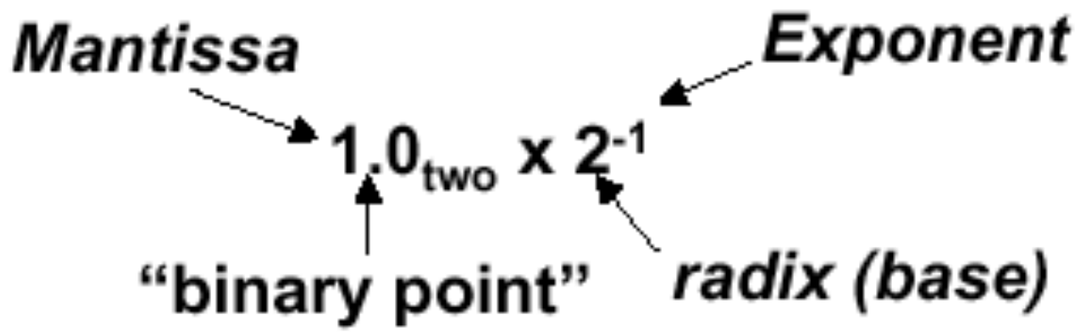


Same strategy works with binary



Scientific notation - binary





Floating point (reals)

We are skipping a lot of detail here.

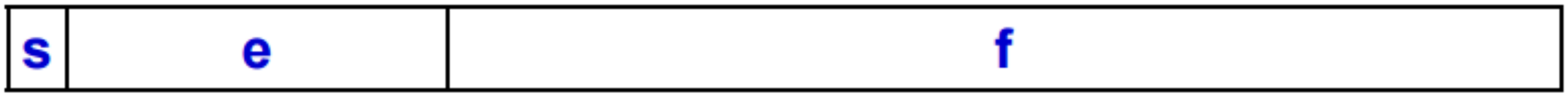
All you need to know:

Floating point numbers are needed for
- non-integer numbers
- very large numbers
- very small (fractional number)

Some decimal numbers cannot be represented exactly in binary

Mistakes converting between integers and reals
- 34.7% of all the worlds bugs

Floating point representation



IEEE 754 standard

Single precision (32-bit) and double precision (64-bit) formats

Sign: always 1 bit

Exponent: 8 [−126, 127] or 11 bits [−1022, 1023]

Significant/fraction: 23 or 52 bits

Characters and Text

ASCII

American Standard Code for Information Interchange
128 characters
One bit could be used for parity

‘H’	‘E’	‘L’	‘L’	‘O’
0x48	0x45	0x4C	0x4C	0x4F

Each character is a number

Important for sorting
Uppercase before lowercase

A string is an array (or list) of bytes

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

Python chr() and ord()

Python has two functions for converting between characters and ASCII codes.

```
chr(81)
Out[80]:
'Q'

ord('Q')
Out[81]:
81
```

Write an expression that will convert a lower case letter to upper :
- hint

```
diff = ord('a') - ord('A')

lower = 'c'
upper = X
```

Characters & Text

Unicode

Up to 4 bytes per character

128,000 characters in the standard

http://unicode-table.com

Recently added emojis

- http://www.unicode.org/emoji/charts/emoji-released.html

Options

UTF-8

- 1 byte for ASCII chars, up to 4 for others
- Economic so used a lot

UTF-16

- 2 or 4 bytes

UTF-32

- Fixed width

```
print ("\U0001f60d")
```

😍

UTF-8

UTF-32 is pretty wasteful

```
  P           y           t           h           o           n
0x50 00 00 00 79 00 00 00 74 00 00 00 68 00 00 00 6f 00 00 00 6e 00 00 00
   0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
```

UTF-8

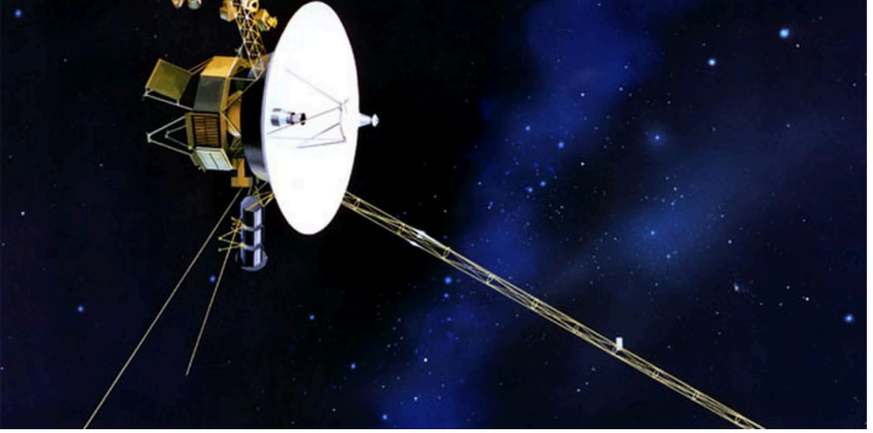
```
  P  y  t  h  o  n
0x50 79 74 68 6f 6e
   0  1  2  3  4  5
```

In Python

```
In [28]: MyString ='Python'

In [29]: MyStringU = MyString.encode('utf-8')

In [30]: MyStringU
Out[30]: b'Python'

In [31]: MyString
Out[31]: 'Python'

In [32]: MyStringU.hex()
Out[32]: '507974686f6e'
```

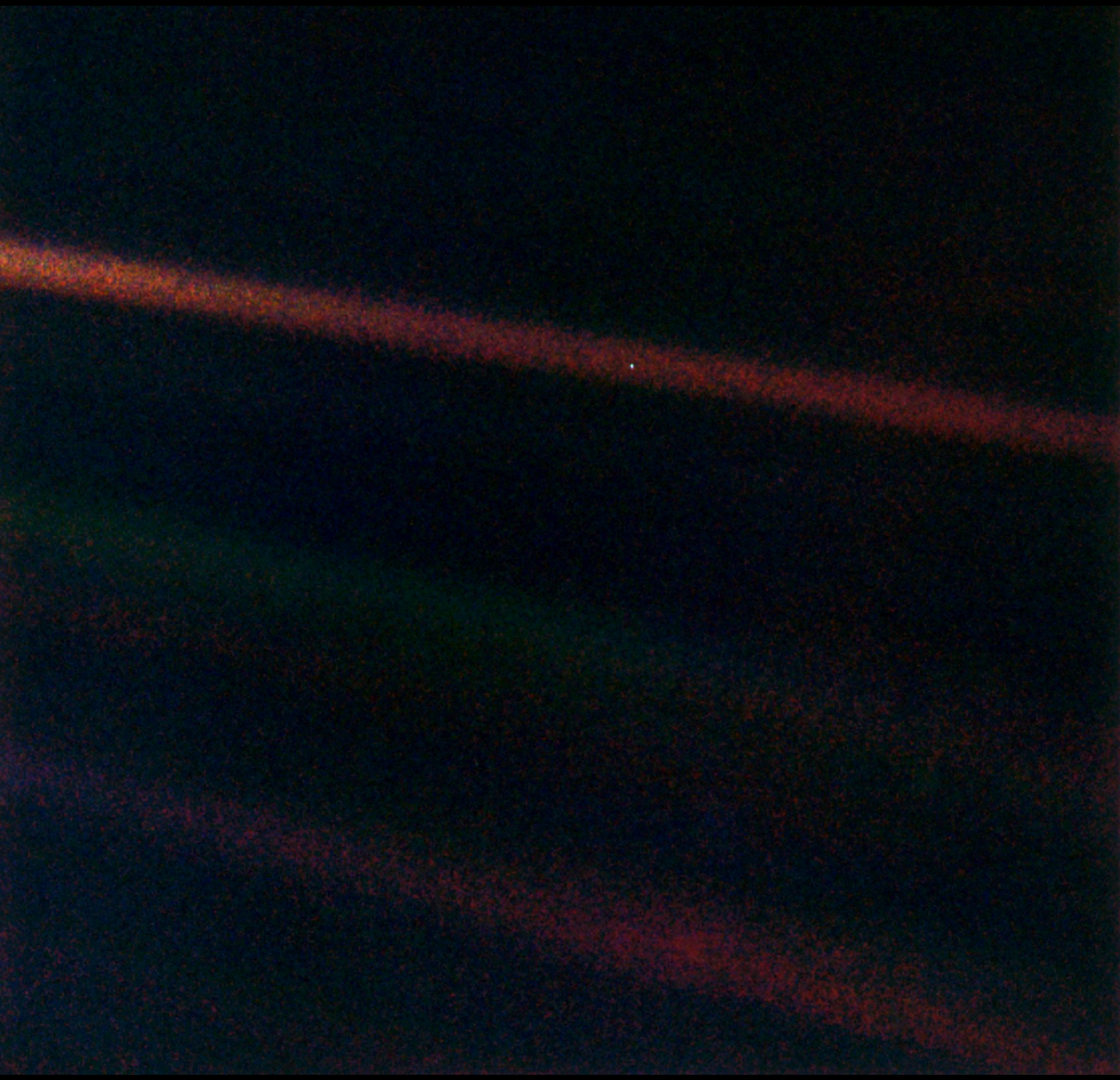
Alien Alert System

Devise an efficient code to communicate:
1) aliens contacted, 2) no aliens signed,
3) unknown object sighted, 4) all systems OK



Alien Alert System

00 Aliens contacted, RUN!
01 No aliens sighted
11 Unknown object sighted
10 All systems OK



Basic Data Representations

Learning Outcomes

You should be able to:

explain the size implications of representing integers using 1,2,4 or more bytes

describe how alphanumeric characters can be represented using 1 byte (ASCII)

explain how Unicode works (UTF-8 and UTF-32)

describe how numbers are represented in floating point in binary