Representing Numbers

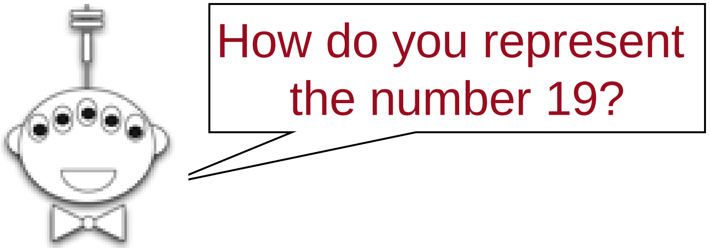
What is the number 4312?

10^3	10^2	10^1	10^0
4	3	1	2

What is this number in base 20?

20^2	20^1	20^0
1	3	2

Now we're using powers of 20



Olmec number representation
in base 20 (East Mexico 1200 BC-600 AD)

Olmec relief from http://www.meta-religion.com

Arbitrary Bases (base "*b*")

Which b?

When using base *b*, the digits permitted are:

What is 5 in…

base 2?

base 3?

base 4?

base 5?

base 6?

base 42?

What's the "algorithm" for counting in a general base *b*?

Arbitrary Bases (base “*b*”)

When using base *b*, the digits permitted are:

What is 5 in…

base 2?

base 3?

base 4?

base 5?

base 6?

base 42?

We write:

$$\mathbf{101_2 = 12_3 = 11_4 = 10_5 = 5_6 = 5_{10} = 5_{42}}$$

The subscript indicates the base

What’s the “algorithm” for counting in a general base *b*?

Is There Such a Thing as Base 1?

Unary!

$1^3 \quad 1^2 \quad 1^1 \quad 1^0$

Now we're using powers of 1 (Weird!)

Are we going to use 0 as our only digit?

Comparing Representations in Different Bases

Consider the number 10^9 in base 1, 2, 3, 10, and 20:

Base 1: 11111111111111111111111111111111111111111111…

At 10 “1’s” per inch, this will be…

Base 2: 111011100110101100101000000000

Base 3: 2120200200021010001

Base 10: 1000000000

Base 20: FCA0000

What’s the ratio between the lengths of a number in bases x and y?

Comparing Representations in Different Bases

Consider the number 10^9 in base 1, 2, 3, 10, and 20:

Base 1: 11111111111111111111111111111111111111111111…

At 10 "1's" per inch, this will be **1578 miles long!**

Base 2: 111011100110101100101000000000

Base 3: 2120200200021010001

Base 10: 1000000000

Base 20: FCA0000

What's the ratio between the lengths of a number in bases x and y?

Two "Special" Bases: 2 and 10

Base 10: Elamites in Iran use early form of base 10 system around 3500 B.C.

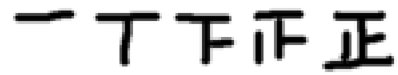
Base 2: References to base 2 appeared in the *I Ching.* (2800 B.C.)

Computers are "simple".
Base 2 is the simplest reasonable base.
Therefore, computers use base 2!

A Brief History of Bases

Unary: Used since at least 400 B.C.

Europe, New Zealand
North America



China, Japan, Korea

Base 60 (“Sexagesimal”): Sumerians in Mesopotamia (Iraq) around 300-400 B.C.

Base 20 (“Vigesimal”): Olmec and other Mesoamerican cultures - 3000 year period before Columbus arrives in the Americas

Base 8 (“Octal”): Yuki Tribe of Northern CA

Members of the Yuki Tribe c. 1858
(from wikipedia.org)

Converting Between Bases

The digits 0 and 1 are referred to as "bits" - that's short for "binary digits"

Convert 1101_2 to base 10

Convert 25_{10} to base 2

The "Power" of Shifting!

"Left Shifting"

10^3	10^2	10^1	10^0
		4	2
	4	2	0

2^3	2^2	2^1	2^0		
		1	1	=	3_{10}
	1	1	0	=	?

"Right Shifting"

10^3	10^2	10^1	10^0
	4	5	7
		4	5

2^3	2^2	2^1	2^0		
	1	0	1	=	5_{10}
		1	0	=	?

Addition

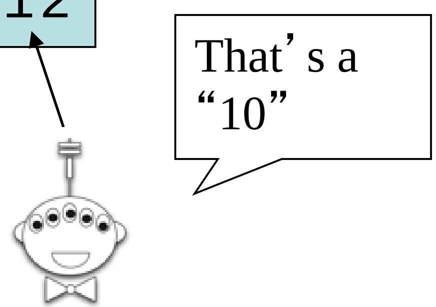
Base 10 Addition

10^2	10^1	10^0
	4	3
+	8	9

Addition

Base 10 Addition

	10^2	10^1	10^0
		4	3
+		8	9
			12



Addition

Base 10 Addition

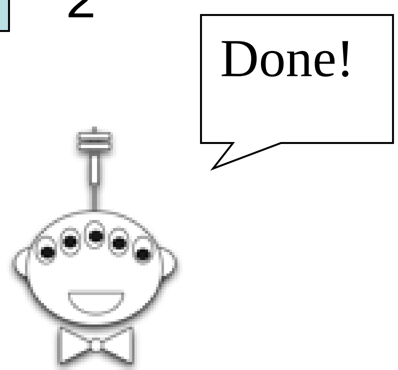
	10^2	10^1	10^0
		1	
		4	3
+		8	9
			2

Move the "1" to the ten's place

Addition

Base 10 Addition

10^2	10^1	10^0
	1	
	4	3
+	8	9
	13	2



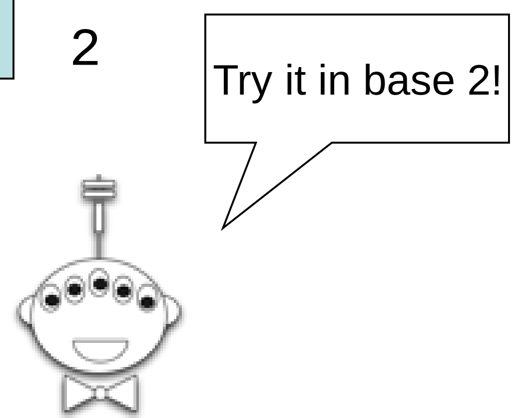
Addition

Base 10 Addition

	10^2	10^1	10^0
		1	
		4	3
+		8	9
		13	2

Base 2 Addition

	2^2	2^1	2^0
	1	0	1
+	0	1	1



Multiplication

Base 10 Multiplication

	10^2	10^1	10^0
	3	4	1
$\times$	1	0	2

Multiplication

Base 10 Multiplication

	10^2	10^1	10^0
	3	4	1
$\times$	1	0	2

		6	8	2
	0	0	0	
+ 3	4	1		

Multiplication

Base 10 Multiplication

	10^2	10^1	10^0
	3	4	1

$$
\begin{array}{rrrrrr}
 & & & 3 & 4 & 1 \\
 & & \times & 1 & 0 & 2 \\
\hline
 & & & 6 & 8 & 2 \\
 & & 0 & 0 & 0 & \\
+ & 3 & 4 & 1 & & \\
\hline
 & 3 & 4 & 7 & 8 & 2
\end{array}
$$

Multiplication

Base 10 Multiplication

$$
\begin{array}{cccccc}
 & & 10^2 & 10^1 & 10^0 \\
\hline
 & & 3 & 4 & 1 \\
 & \times & 1 & 0 & 2 \\
\hline
 & & 6 & 8 & 2 \\
 & 0 & 0 & 0 & \\
+\ 3 & 4 & 1 & & \\
\hline
3 & 4 & 7 & 8 & 2
\end{array}
$$

Base 2 Multiplication

$$
\begin{array}{cccc}
 & 2^2 & 2^1 & 2^0 \\
\hline
 & 1 & 1 & 1 \\
\times & 1 & 0 & 1 \\
\hline
\end{array}
$$

Multiplication with Russian Peasants

Compute 21×6:

21	6
10	12
5	24
2	48
1	96

(Translation: "Hello American Students!")

Multiplication with Russian Peasants

Compute `21 × 6`:

21	6
10	12
5	24
2	48
1	96

6+24+96 = $\boxed{126}$

(Translation: “Why does this work?”)

Multiplication

Base 10 Multiplication

$$
\begin{array}{cccccc}
 & & 10^2 & 10^1 & 10^0 \\
\hline
 & & 3 & 4 & 1 \\
 & \times & 1 & 0 & 2 \\
\hline
 & & 6 & 8 & 2 \\
 & 0 & 0 & 0 & \\
+\ 3 & 4 & 1 & & \\
\hline
3 & 4 & 7 & 8 & 2
\end{array}
$$

Base 2 Multiplication

$$
\begin{array}{cccc}
 & 2^2 & 2^1 & 2^0 \\
\hline
 & 1 & 1 & 1 \\
\times & 1 & 0 & 1 \\
\hline
\end{array}
$$

Multiplication with Russian Peasants

Compute 21×6:

21	6
10	12
5	24
2	48
1	96

(Translation: “Hello American Students!”)

Multiplication with Russian Peasants

Compute 21 × 6:

21	6
10	12
5	24
2	48
1	96

6+24+96 = 126

(Translation: "Why does this work?")

Multiplication with Russian Peasants

Compute 21×6:

21	6
10	12
5	24
2	48
1	96

6+24+96 = 126

10101	110
1010	1100
101	11000
10	110000
1	1100000

1111110

(Translation: “I love binary!”)

Multiplication with Russian Peasants

110		
× 10101		
110	10101	110
000	1010	1100
11000	101	11000
000	10	110000
1100000	1	1100000
1111110		**1111110**

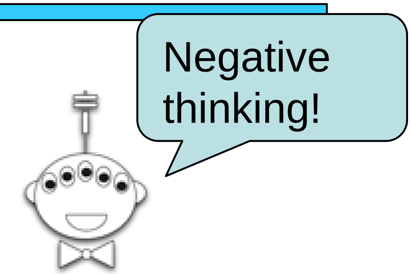
Try It!

Compute 33 x 7

Negative Numbers
(with the nifty "two's complement" method)

- Assume that we have only 8 bits to represent numbers
- If we try to increment `11111111` by 1, what happens?
- `00000011` represents 3_{10}. What property should the representation of -3_{10} have so that arithmetic with positive and negative numbers works nicely?

Exercise…

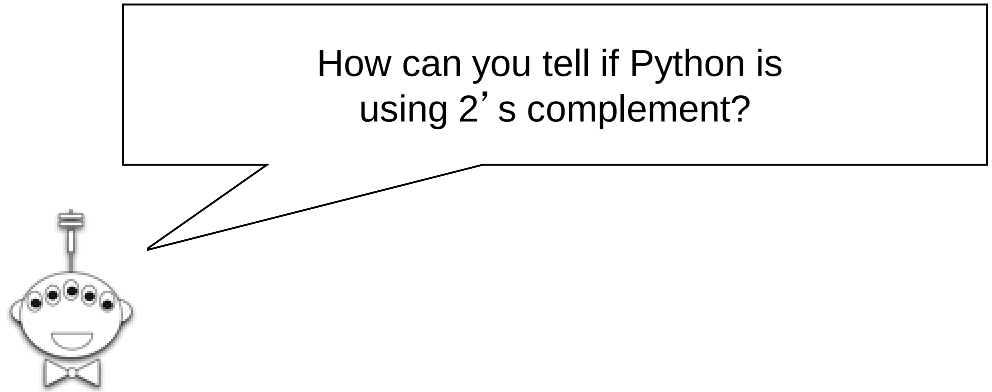


In two's complement (with 3 bits to keep things simpler)…

- What's the negative of 0?
- How is -1 represented?
- What's the largest positive number that can be represented?
- What's the smallest negative number that can be represented?
- Does addition work as expected?
- Is a double negative a positive?

Does Python Really Use This?

```
>>> x = 1
>>> ~x
```



What's up with this!?

```
>>> .1
0.10000000000000001
>>> .01*10 == .01/.1
False
```

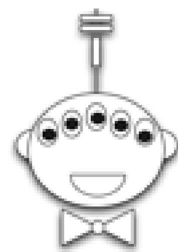
sinking with `float`s

2^{-1} 2^{-2} 2^{-3} 2^{-4}

0.0000	0.0000
0.0001	0.0625
0.0010	0.1250
0.0011	0.1875
0.0100	0.2500
0.0101	0.3125
...	...
0.1100	0.7500
0.1101	0.8125
0.1110	0.8750
0.1111	0.9375

4 bits

exact decimal equivalents



Imagine a computer that uses only 4 bits to represent decimals…

In reality, 23 bits or 53 bits will be used to represent the fractional part of a floating-point number

lots of gaps in here…

```
>>> X = 0.1
```

What’s up with this!?

```
>>> .1
0.10000000000000001

>>> .01*10 == .01/.1
False
```

http://docs.python.org/tutorial/floatingpoint.html

Explains why the actual value stored for .1 is about
0.1000000000000000055511151231257
and why it used to get displayed as above.

Beyond numbers…

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	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	Neg. 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 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	Substitution	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	□

ASCII Code

```
>>> chr(42)
‘*’

>>> ord(‘9’)
57
```

Data compression coming soon!