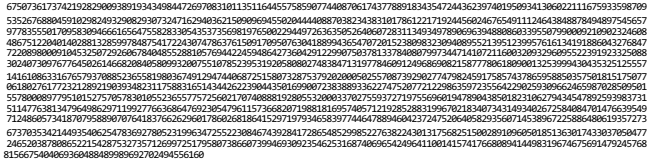
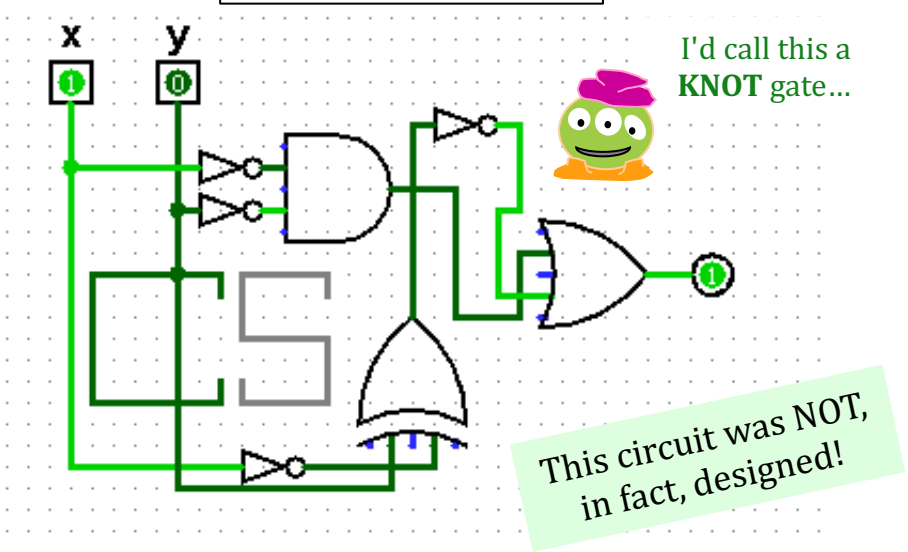
More ***bits*** of CS



Too many bits? <u>Compress!</u>

Below binary: ***physical circuits***

Circuit design, part 1



Hw #4 due Tue. **2/20**

pr0 *(reading)* A bug and a crash!

pr1 *(lab)* binary ~ decimal

pr2 conversion + compression

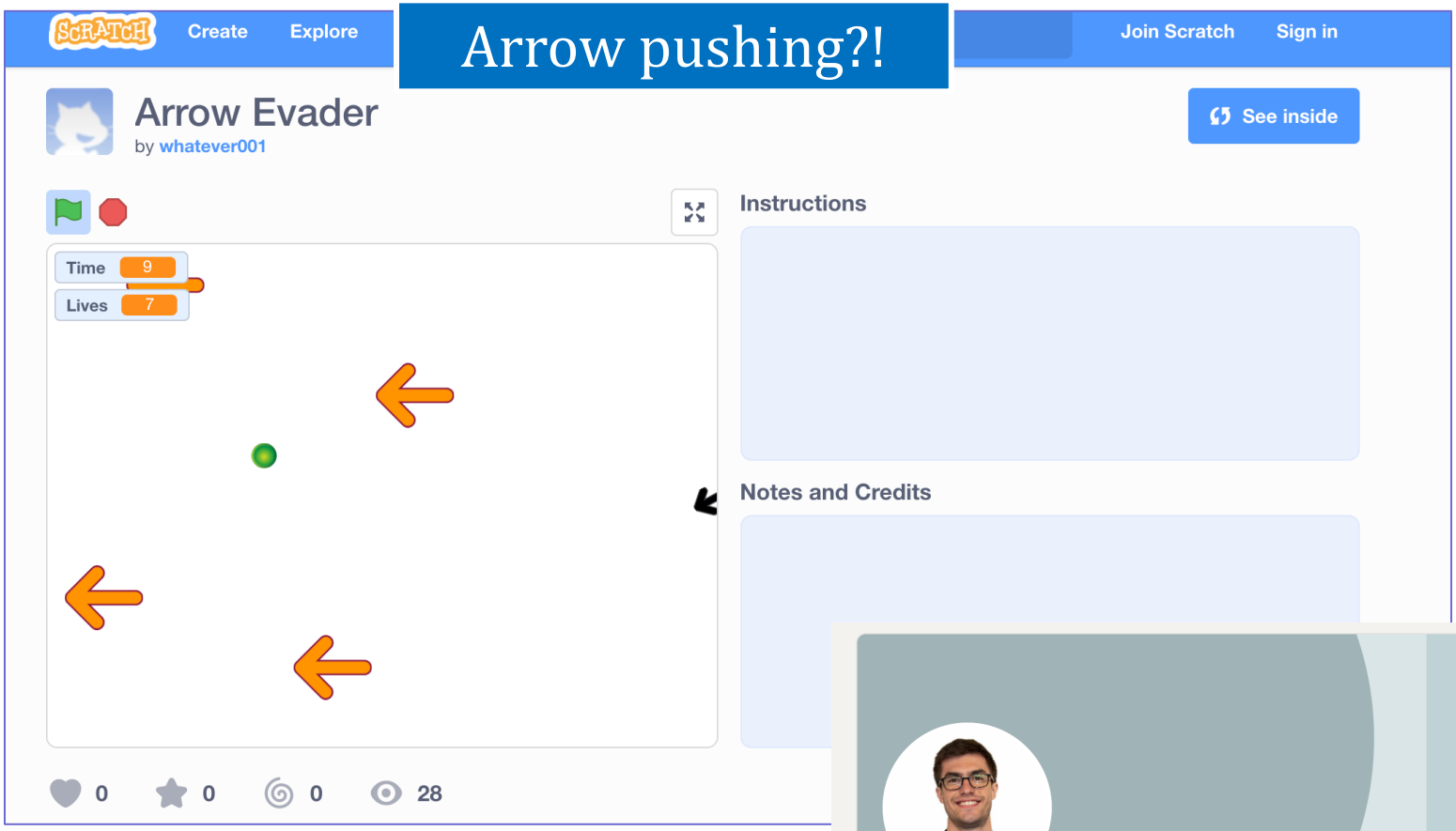
extra image processing...

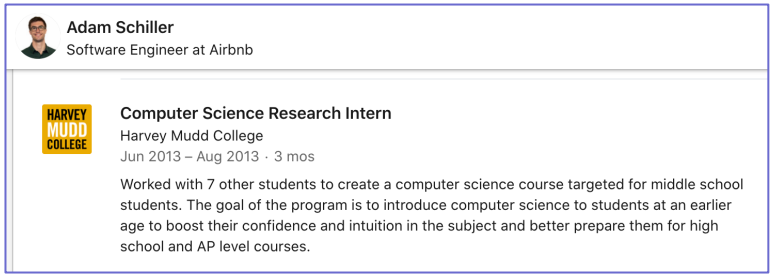
Lots of tutoring hrs - join in... !

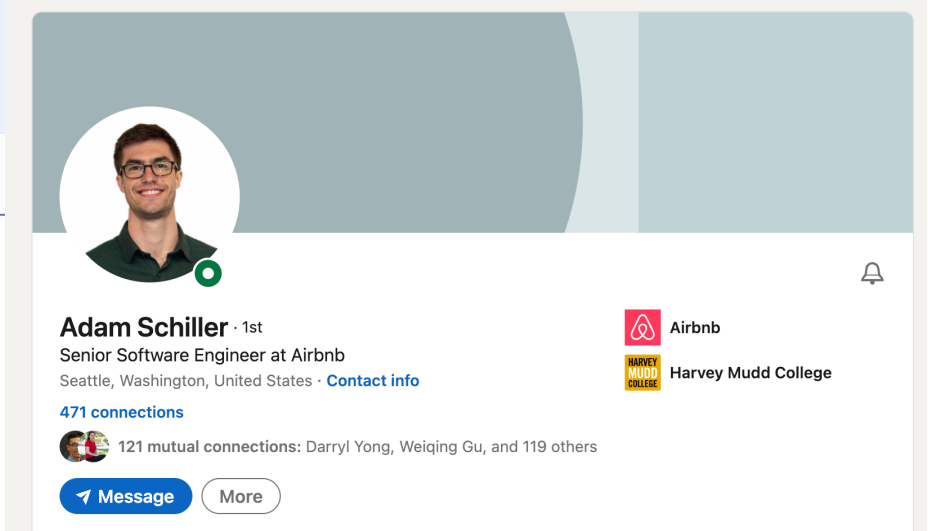
P/F vs T/F

Bio + Chem!?!!

https://scratch.mit.edu/projects/10866563/







Reasoning, *bit by bit*

<<	>>	&	|
left-shift	right-shift	and	or

and (both) **&** **|** or (either)

bitwise and

```
5:   101
6:   110
---------
&    100
```

5 & 6

4

bitwise and

```
11: 1011
 5: 0101
---------
&
```

11 & 5

bitwise or

```
5:   101
6:   110
---------
|    111
```

5 | 6

7

bitwise or

```
11: 1011
 5: 0101
---------
|
```

11 | 5

At its base ...

(case)

```
ntb( 42 )
ntb( 21 ) + '0'
ntb( 10 ) + '1'
ntb( 5 ) + '0'
ntb( 2 ) + '1'
ntb( 1 ) + '0'
ntb( 0 ) + '1'
''
```

in

42

out

'101010'

```
def numToBin( N      ):
    """ converts a decimal int to a binary string
    """
    if N==0:        return ''
    else:           return numToBin(N//2) + str(N%2)
```

What if you wanted base-3 output?! ***base-B output?***

At its base ...

(case)

in

42

```
ntb( 42 )
ntb( 21 ) + '0'
ntb( 10 ) + '1'
ntb( 5 ) + '0'
ntb( 2 ) + '1'
ntb( 1 ) + '0'
ntb( 0 ) + '1'
''
```

out

'101010'

```
def numToBin( N, b ):
    """ converts a decimal int to a binary string
    """
    if N==0:        return ''
    else:           return numToBin( N//~~2~~ b ) + str(N%~~2~~ b)
```

What if you wanted base-3 output?! ***base-B output?***

At its base ...

(case)

```
ntb( 42 )
ntb( 21 ) + '0'
ntb( 10 ) + '1'
ntb( 5 ) + '0'
ntb( 2 ) + '1'
ntb( 1 ) + '0'
ntb( 0 ) + '1'
''
```

in

42

out

'101010'

```
def numToBin( N      ):
    """ converts a decimal int to a binary string
    """
    if N==0:        return ''
    else:           return numToBin( N>>1 ) + str(N&1)
```

What if you wanted base-3 output?! ***base-B output?***

At its base ...

(case)

in

'101010'

```
btn('101010')
2*btn('10101') + 0
2*btn('1010') + 1
2*btn('101') + 0
2*btn('10') + 1
2*btn('1') + 0
2*btn('') + 1
0
```

2*1

2*2*2*1

2*2*2*2*2*1

out

42

```
def binToNum( S     ):
    """ converts a binary string to a decimal int
    """
    if S=='': return 0
    else:     return 2*binToNum(S[:-1]) + int(S[-1])
```

What if you wanted base-3 input?! ***base-B input?***

saves the need for another **if**

At its base ...

(case)

in

'101010'

```
btn('101010')
2*btn('10101') + 0
2*btn('1010') + 1
2*btn('101') + 0
2*btn('10') + 1
2*btn('1') + 0
2*btn('') + 1
0
```

2*1

2*2*2*1

2*2*2*2*2*1

out

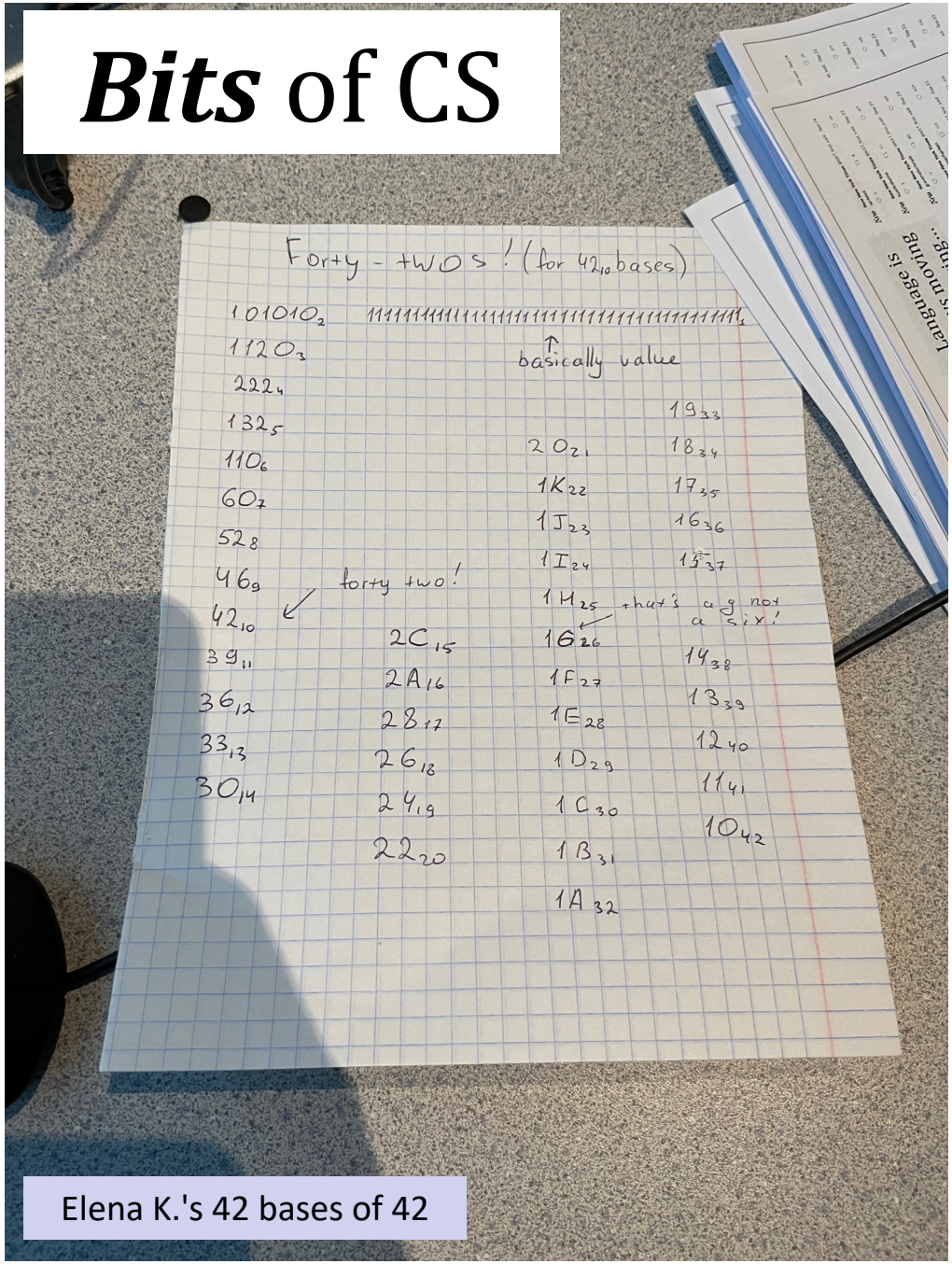
42

```
def binToNum( S     ):
    """ converts a binary string to a decimal int
    """
    if S==''': return 0
    else:      return 2*binToNum(S[:-1]) + int( S[-1] )
```

What if you wanted base-3 input?! ***base-B input?***

saves the need for another **if**

Bits of CS



Elena K.'s 42 bases of 42

Dec 25 == Oct 31

$54_{7.6} == 42_{10}$

Fractional bases are possible too!

https://brilliant.org/wiki/fractional-number-base/

Bits & Base-2

value	representation (64 32 16 8 4 2 1)
0	0
1	1
2	10
5	101
10	1010
15	1111
16	10000
21	10101
42	101010
127	1111111

What are 3 and 4 in base-2?

7?

8?

What does *shifting right* do? >> 1

What's **1000** ?

div by 2

What does *shifting left* do? << 1

What's **101000** ?

mult by 2

What's the max value **4** bits can hold?

What's the max value **7** bits can hold?

What's the max value **N** bits can hold?

Extra: How could bits represent *negative* numbers? *Fractions?*

Bits & Base-2

value	representation (64 32 16 8 4 2 1)
0	0
1	1
2	10
5	101
10	1010
15	1111
16	10000
21	10101
42	101010
127	1111111

What are 3 and 4 in base-2? **11** **100**

7? **111**

8? **1000**

What does *shifting right* do? >> 1

What's **1000** ? **8** *halves the value*

What does *shifting left* do? << 1

What's **101000** ? **40** *doubles the value*

15 What's the max value **4** bits can hold?

127 What's the max value **7** bits can hold?

What's the max value **N** bits can hold?

Extra: How could bits represent *negative* numbers? *Fractions?*

How *far* can we count…?

I can see some patterns here –
even *with one eye closed!*

			max count
with	**1 bit**	**1**	**1**
	2 bits	**11**	**3**
	3 bits	**111**	**7**
	4 bits	**1111**	**15**
	7 bits	**1111111**	**127**
	8 bits	**11111111**	**255**
	N bits	**1111...1111**	

8-bit signed values
-128 to 127

8-bit unsigned values
0 to 255

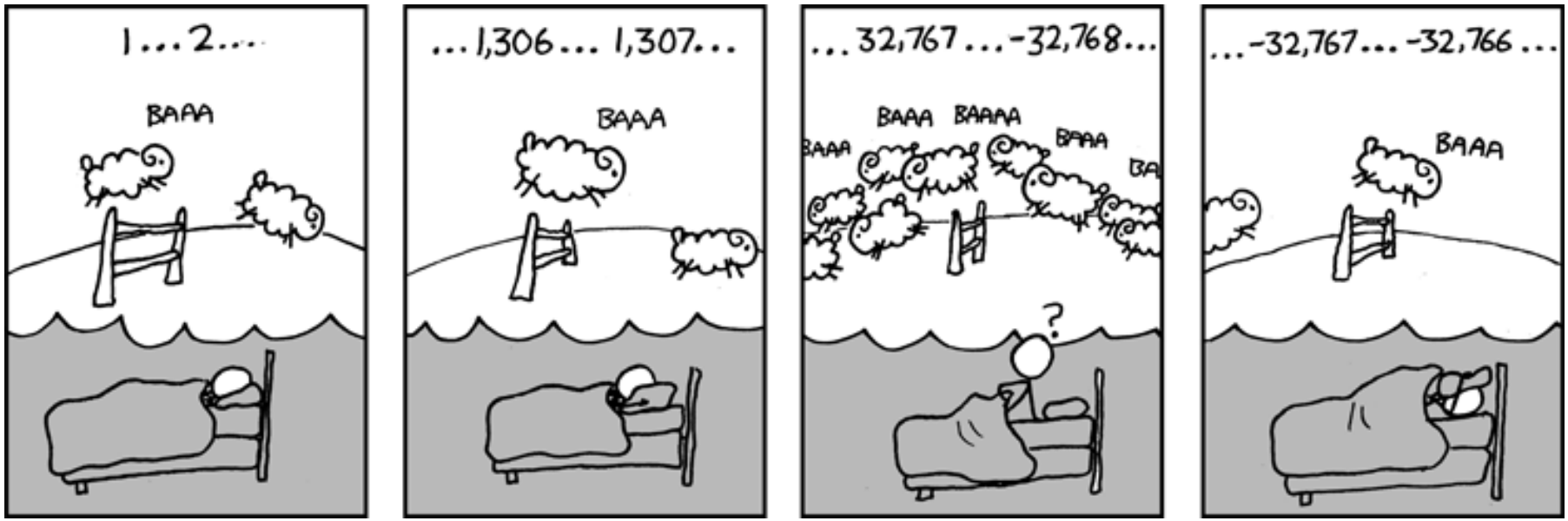
How could bits represent ***negative*** numbers?

15 bits

31 bits

Fractions?

Counting sheep, xkcd *style...*



How many bits?

How *far* can we count…?

I can see some patterns here –
even *with one eye closed!*

with **1 bit**	1	1
2 bits	11	3
3 bits	111	7
4 bits	1111	15
7 bits	1111111	127
8 bits	11111111	255
N bits	1111...1111	2^N-1 max count

8-bit signed values
-128 to 127

8-bit unsigned values
0 to 255

How could bits represent ***negative*** numbers?

15 bits ***32,767***

31 bits ***2,147,483,647***

Fractions?

Ariane 5

This week's reading: ***bits can be vital***

`IndexError` **`TypeError`** **`HumanError`**

version 4 version 5

a much less-critical error...

How *far* can we count…?

in 2015?

List of most viewed YouTube videos

From Wikipedia, the free encyclopedia

in 2022?

Also…

44 languages

Year 2038 problem

Read Edit View history Tools

Article Talk

From Wikipedia, the free encyclopedia

This article **needs additional citations for verification**. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed.

Find sources: "Year 2038 problem" – news · newspapers · books · scholar · JSTOR *(January 2024)* (*Learn how and when to remove this template message*)

The **year 2038 problem** (also known as **Y2038**,[1] **Y2K38**, **Y2K38 superbug** or the **Epochalypse**[2][3]) is a time computing problem that leaves some computer systems unable to represent times after 03:14:07 UTC on 19 January 2038.

The problem exists in systems which measure Unix time – the number of seconds elapsed since the *Unix epoch* (00:00:00 UTC on 1 January 1970) – and store it in a signed 32-bit integer. The data type is only capable of representing integers between $-(2^{31})$ and $2^{31} - 1$, meaning the latest time that can be properly encoded is $2^{31} - 1$ seconds after epoch (03:14:07 UTC on … to the following second (03:14:08) … 2^{31} seconds *before* epoch (20:45:52 UTC on … 32-bit integers will be

Binary	: 01111111 11111111 11111111 11111110
Decimal	: 2147483646
Date	: 2038-01-19 03:14:06 (UTC)
Date	: 2038-01-19 03:14:06 (UTC)

An animated visual of the bug in action. The overflow error will occur at 03:14:08 UTC on 19 January 2038.

How *far* can we count…?

in 2015?

in 2022?

List of most viewed YouTube videos

From Wikipedia, the free encyclopedia

This song?
The song that replaced it?
The song that replaced *it*?

Top videos

indicates a video that is not a music video.

Rank	Video name[A]	Uploader / artist	Views (as of September 29, 2015)	Upload date	Notes
1.					
2.	"Baby"[4]	Justin Bieber featuring Ludacris	1,216,729,955	February 19, 2010	[C]
3.	"Blank Space"[5]	Taylor Swift	1,173,509,710	November 10, 2014	[D]

The two "Starbucks songs"

How *far* can we count…?

in 2015!

in 2022?

List of most viewed YouTube videos

From Wikipedia, the free encyclopedia

This **list of most viewed YouTube videos** consists of the 30 most viewed videos of all time as derived from YouTube charts.[1] Videos that YouTube suspects have had their view counts manipulated[2] are not included in this list. View counts are based on the YouTube website; many of the videos are music videos that play through YouTube's partner site, Vevo, and YouTube view counts will lag those of Vevo by a few days.[1]

As of September 2015, nine music videos have received over 1 billion views, with the top video, "Gangnam Style", exceeding 2 billion views.

Psy's "Gangnam Style" is the most watched video on YouTube as of September 2015, with over 2.4 billion views.

Top videos

indicates a video that is not a music video.

recognize this number?!

Rank	Video name[A]		Views (as of September 29, 2015)	Upload date	Notes
1.	"Gangnam Style"[3]	Psy	2,421,271,749	July 15, 2012	[B]
2.	"Baby"[4]	Justin Bieber featuring Ludacris	1,216,729,955	February 19, 2010	[C]
3.	"Blank Space"[5]	Taylor Swift	1,173,509,710	November 10, 2014	[D]

31 bits

Bits bite back!

THE WALL STREET JOURNAL. ≡ ARTS & ENTERTAINMENT

9:19 am ET
Dec 3, 2014 MUSIC

Psy's 'Gangnam Style' Has Forced YouTube to 'Upgrade' Systems

Gangnam Style Video Overflows YouTube Counter

By Rick Regan (Published December 3rd, 2014)

On Monday, Psy's Gangnam Style video exceeded the limit of YouTube's view counter; this is what Google had to say

"We never thought a video would be watched in numbers greater than a 32-bit integer (=2,147,483,647 views)…"

The "sign bit" has flipped to one. Thus, the number has become *negative*… !

PSY - GANGNAM STYLE (강남스타일) M/V

officialpsy

Subscribe 7,600,830

-2142584554

Add to Share More

8,761,309 1,139,933

Hw4: *images are just bits, too*!

hw4pr3 (*extra*)

<u>old</u> pixel at 42,42 has

red = 1 (out of 255)
green = 36 (out of 255)
blue = 117 (out of 255)

how many <u>bits</u> are being used for each color channel?

Hw4: *images are just bits, too!*

hw4pr3 ***(extra)***

<u>old</u> pixel at 42,42 has

red = 1 (out of 255)
green = 36 (out of 255)
blue = 117 (out of 255)

<u>new</u> pixel at 42,42 has

any guesses as to what this transformation was?

how many <u>bits</u> are being used for each color channel?

Hw4: *images are just bits, too!*

hw4pr3 (*extra*)

rgb-invert

<u>old</u> pixel at 42,42 has

red = 1 (out of 255)
green = 36 (out of 255)
blue = 117 (out of 255)

<u>new</u> pixel at 42,42 has

red = 254 (out of 255)
green = 219 (out of 255)
blue = 138 (out of 255)

how many <u>bits</u> are being used for each color channel?

Hw4: *images are just bits, too!*

hw4pr3 (*extra*)

rgb-invert

<u>old</u> pixel at 42,42 has

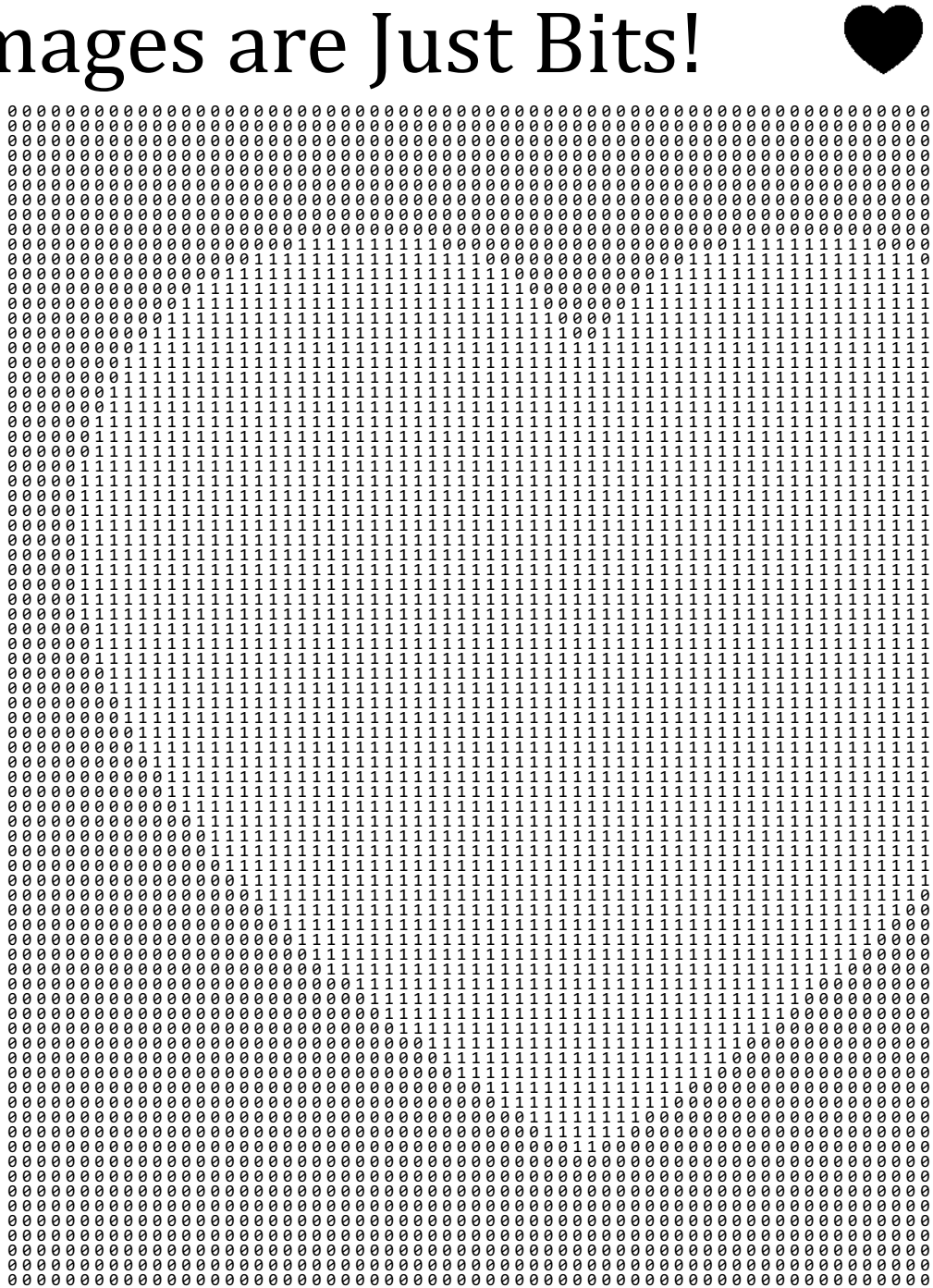
red = 1 (out of 255)
green = 36 (out of 255)
blue = 117 (out of 255)

<u>new</u> pixel at 42,42 has

red = 254 (out of 255)
green = 219 (out of 255)
blue = 138 (out of 255)

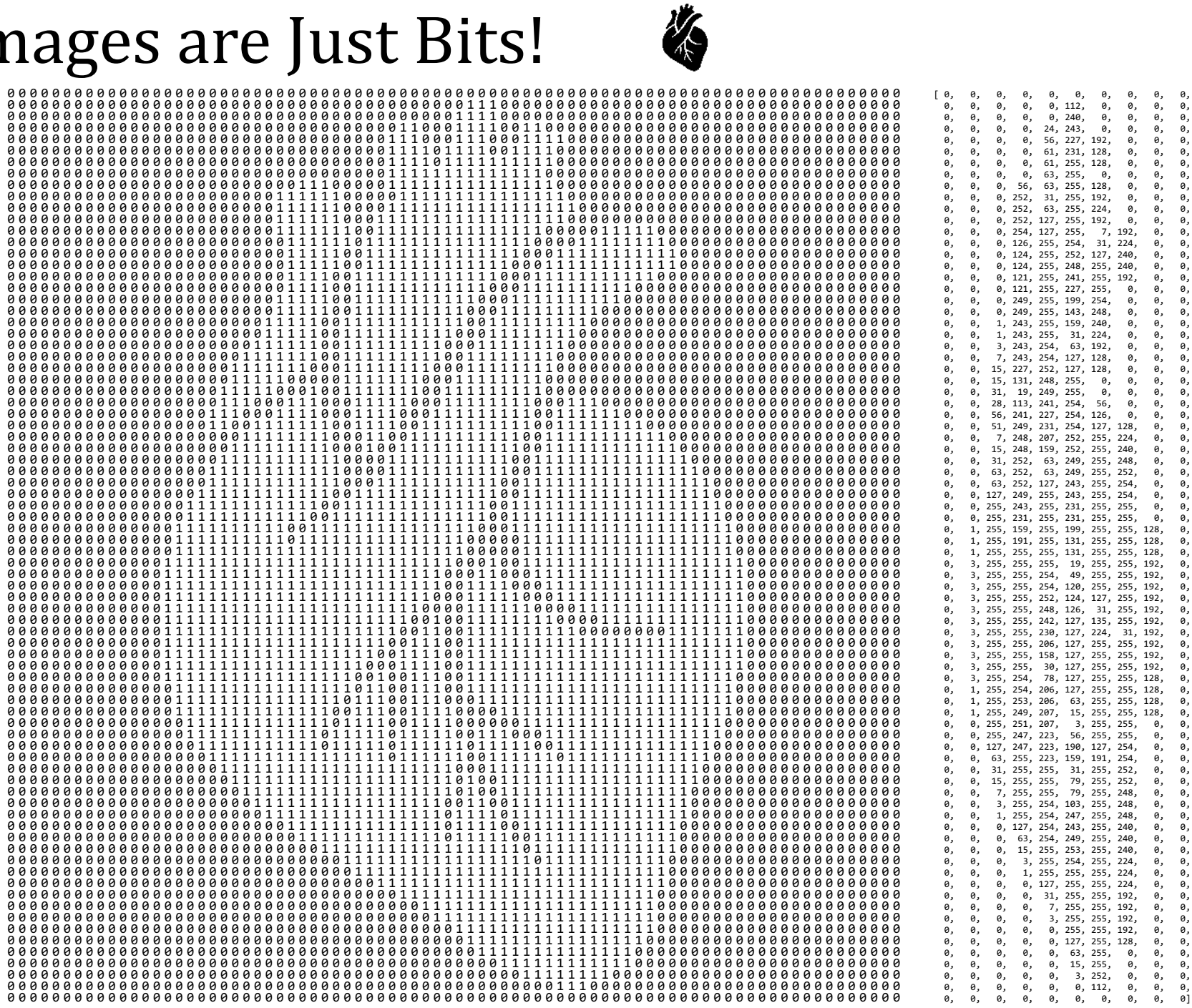
how many <u>bits</u> are being used for each color channel?

Images are Just Bits!

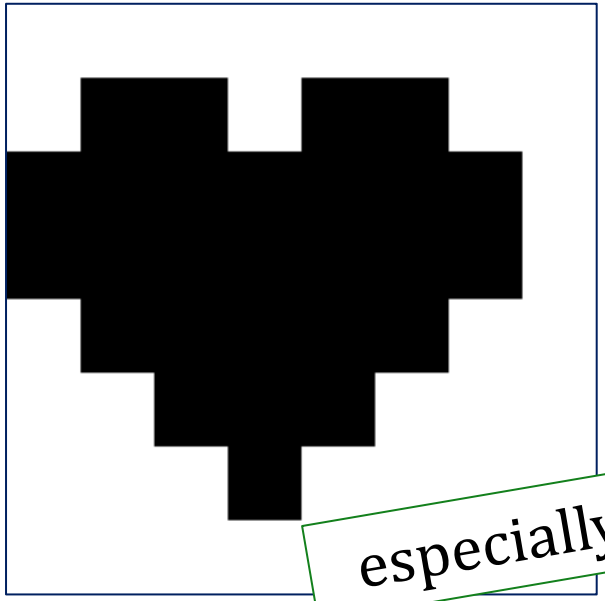


```
[ 0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,  15, 252,   0,   0,  63, 240,   0,   0,
  0,   0, 127, 255, 128,   1, 255, 254,   0,   0,
  0,   1, 255, 255, 224,   7, 255, 255, 128,   0,
  0,   7, 255, 255, 240,  15, 255, 255, 224,   0,
  0,  15, 255, 255, 248,  31, 255, 255, 240,   0,
  0,  31, 255, 255, 252,  63, 255, 255, 248,   0,
  0,  63, 255, 255, 254, 127, 255, 255, 252,   0,
  0, 127, 255, 255, 255, 255, 255, 255, 254,   0,
  0, 255, 255, 255, 255, 255, 255, 255, 255,   0,
  0, 255, 255, 255, 255, 255, 255, 255, 255,   0,
  1, 255, 255, 255, 255, 255, 255, 255, 255, 128,
  1, 255, 255, 255, 255, 255, 255, 255, 255, 128,
  3, 255, 255, 255, 255, 255, 255, 255, 255, 192,
  3, 255, 255, 255, 255, 255, 255, 255, 255, 192,
  3, 255, 255, 255, 255, 255, 255, 255, 255, 192,
  7, 255, 255, 255, 255, 255, 255, 255, 255, 224,
  7, 255, 255, 255, 255, 255, 255, 255, 255, 224,
  7, 255, 255, 255, 255, 255, 255, 255, 255, 224,
  7, 255, 255, 255, 255, 255, 255, 255, 255, 224,
  7, 255, 255, 255, 255, 255, 255, 255, 255, 224,
  7, 255, 255, 255, 255, 255, 255, 255, 255, 224,
  7, 255, 255, 255, 255, 255, 255, 255, 255, 224,
  7, 255, 255, 255, 255, 255, 255, 255, 255, 224,
  7, 255, 255, 255, 255, 255, 255, 255, 255, 224,
  7, 255, 255, 255, 255, 255, 255, 255, 255, 224,
  7, 255, 255, 255, 255, 255, 255, 255, 255, 224,
  3, 255, 255, 255, 255, 255, 255, 255, 255, 192,
  3, 255, 255, 255, 255, 255, 255, 255, 255, 192,
  3, 255, 255, 255, 255, 255, 255, 255, 255, 192,
  1, 255, 255, 255, 255, 255, 255, 255, 255, 128,
  1, 255, 255, 255, 255, 255, 255, 255, 255, 128,
  0, 255, 255, 255, 255, 255, 255, 255, 255,   0,
  0, 255, 255, 255, 255, 255, 255, 255, 255,   0,
  0, 127, 255, 255, 255, 255, 255, 255, 254,   0,
  0, 127, 255, 255, 255, 255, 255, 255, 254,   0,
  0,  63, 255, 255, 255, 255, 255, 255, 252,   0,
  0,  31, 255, 255, 255, 255, 255, 255, 248,   0,
  0,  31, 255, 255, 255, 255, 255, 255, 248,   0,
  0,  15, 255, 255, 255, 255, 255, 255, 240,   0,
  0,   7, 255, 255, 255, 255, 255, 255, 224,   0,
  0,   3, 255, 255, 255, 255, 255, 255, 192,   0,
  0,   3, 255, 255, 255, 255, 255, 255, 192,   0,
  0,   1, 255, 255, 255, 255, 255, 255, 128,   0,
  0,   0, 255, 255, 255, 255, 255, 255,   0,   0,
  0,   0, 127, 255, 255, 255, 255, 254,   0,   0,
  0,   0,  63, 255, 255, 255, 255, 252,   0,   0,
  0,   0,  31, 255, 255, 255, 255, 248,   0,   0,
  0,   0,  15, 255, 255, 255, 255, 240,   0,   0,
  0,   0,   7, 255, 255, 255, 255, 224,   0,   0,
  0,   0,   3, 255, 255, 255, 255, 192,   0,   0,
  0,   0,   0, 255, 255, 255, 255,   0,   0,   0,
  0,   0,   0, 127, 255, 255, 254,   0,   0,   0,
  0,   0,   0,  63, 255, 255, 252,   0,   0,   0,
  0,   0,   0,  31, 255, 255, 248,   0,   0,   0,
  0,   0,   0,   7, 255, 255, 224,   0,   0,   0,
  0,   0,   0,   3, 255, 255, 192,   0,   0,   0,
  0,   0,   0,   1, 255, 255, 128,   0,   0,   0,
  0,   0,   0,   0, 127, 254,   0,   0,   0,   0,
  0,   0,   0,   0,  63, 252,   0,   0,   0,   0,
  0,   0,   0,   0,  15, 240,   0,   0,   0,   0,
  0,   0,   0,   0,   7, 224,   0,   0,   0,   0,
  0,   0,   0,   0,   1, 128,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0]
```

Images are Just Bits!



Hw4: *images are just bits, too*!



especially ***binary*** images

```
00000000
01101100
11111110
11111110
01111100
00111000
00010000
00000000
```

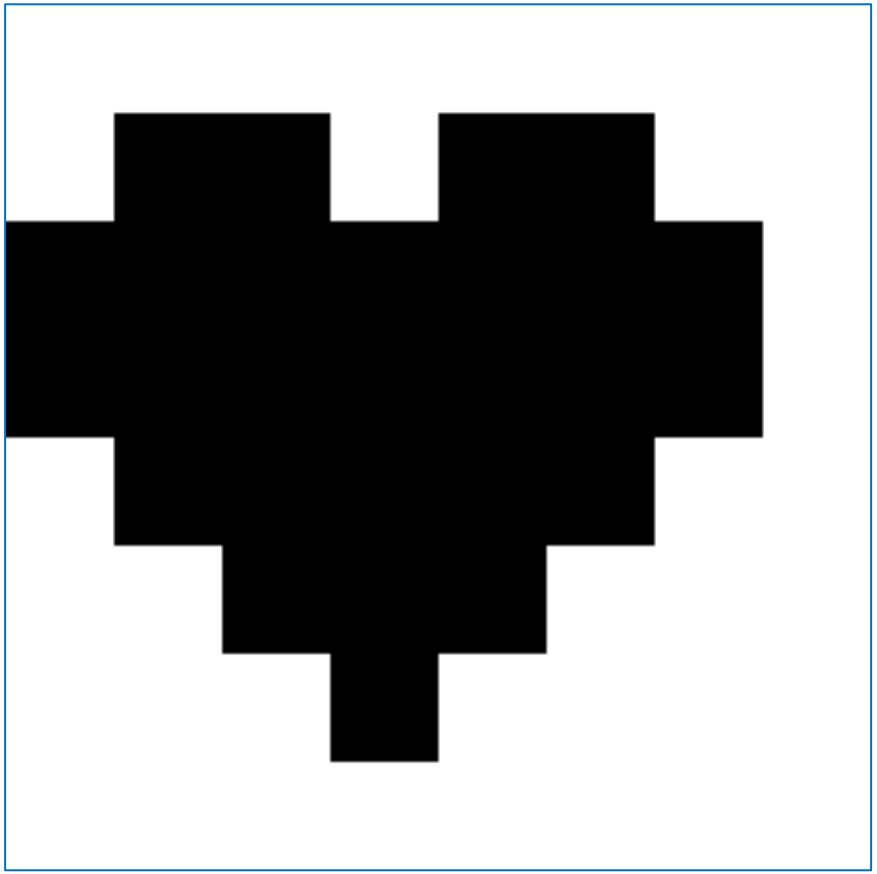
Binary Image

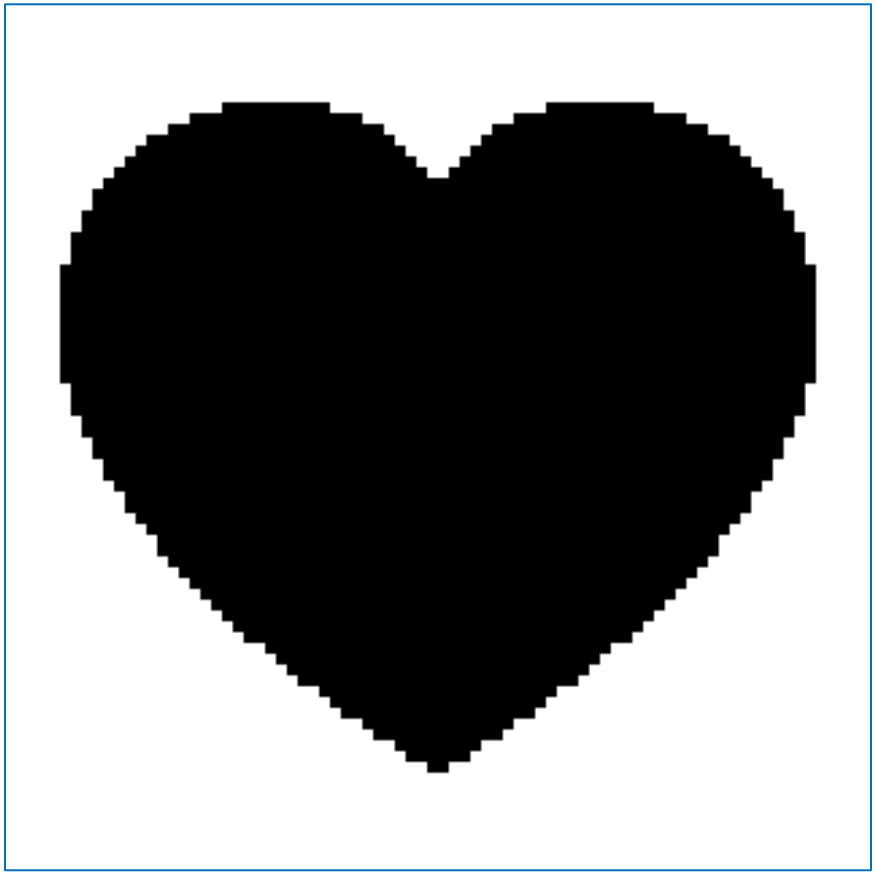
Encoding as raw bits

one big string of 64 characters

```
"0000000001101100111111101111111001111100001110000001000000000000"
```

What's the difference?





Lossy *compression vs.* ***Lossless***

Where in the World?

Think of three situations where ***Lossy*** *compression is better.*

Think of three situations where ***Lossless*** *compression is better.*

Does this also apply to numbers?

and...

Lossy *compression vs.* ***Lossless***

approximations	data!
uncertainty	science+math

Does this also apply to numbers?

56MB	56762440B
3.14	3.1415926535
$1000	$1342

Let's do both!

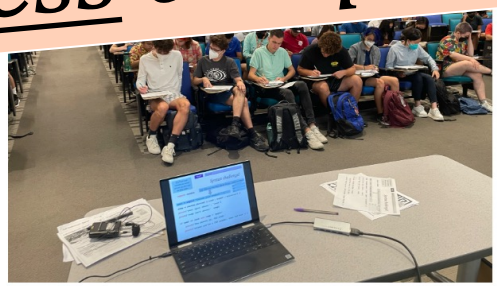
Lossy compression

3670

Bitmap - Uncompressed
57.7 MB

JPEG - 90% Quality
9.2 MB

Lossless compression



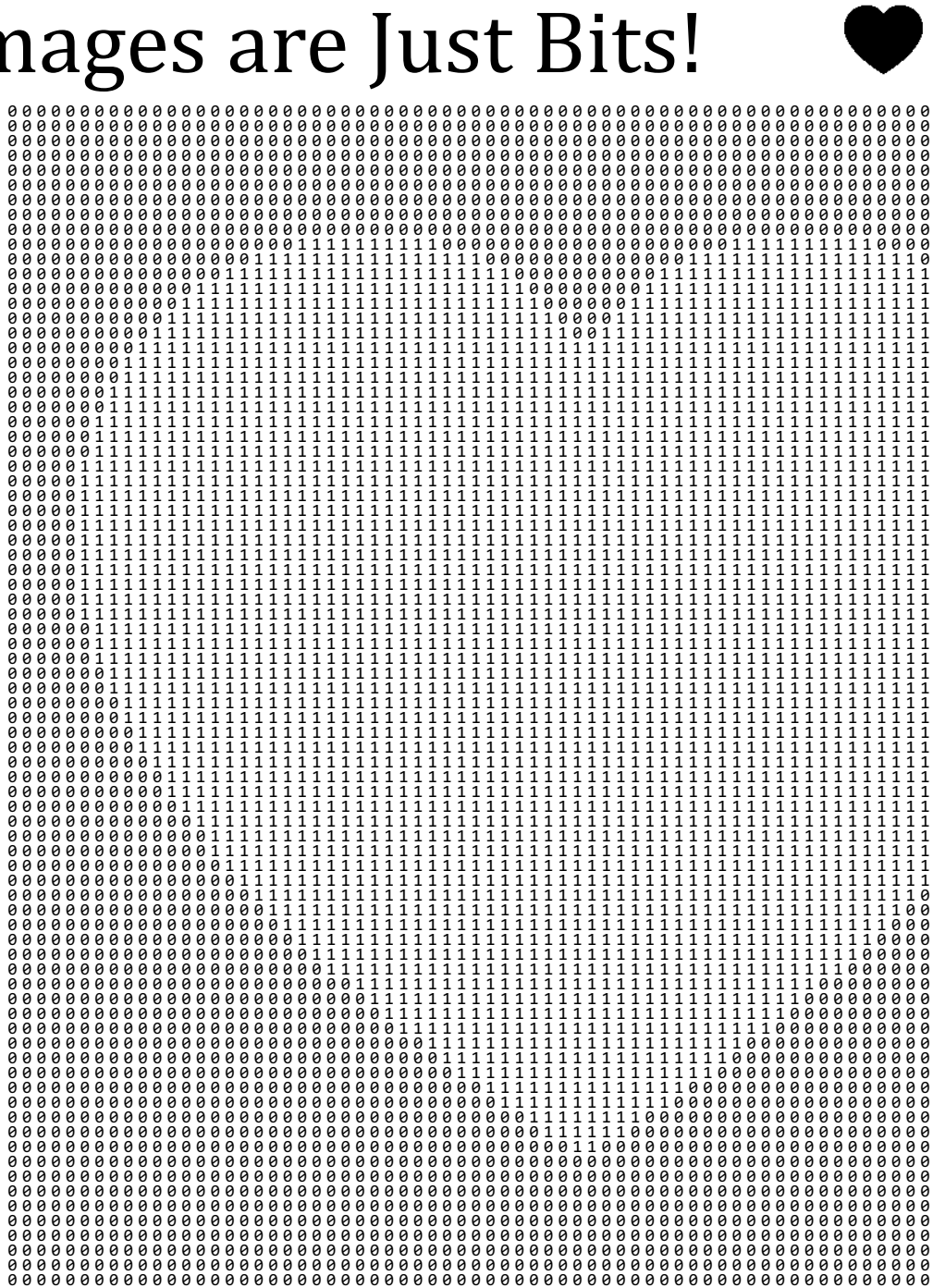
Bitmap – uncompressed
26.8 MB

PNG – compressed
14.5 MB

starting here ...

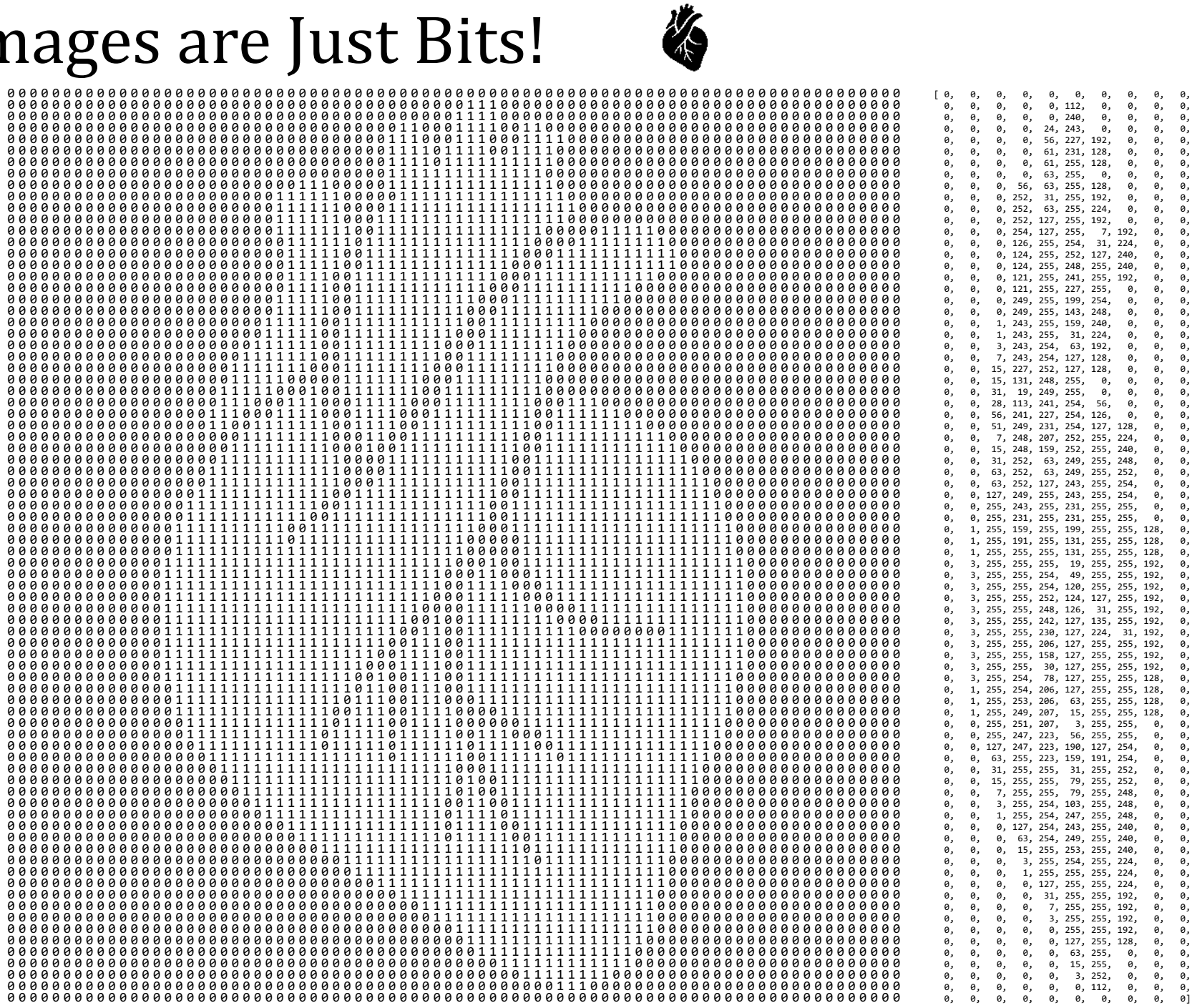
... with *BINARY* images!

Images are Just Bits!

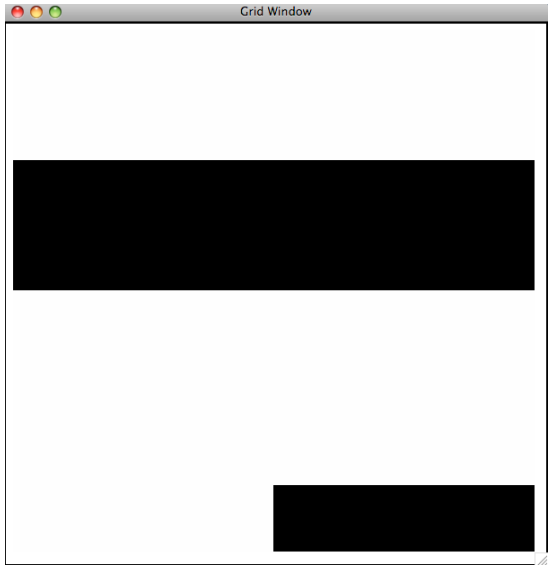


```
[ 0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,  15, 252,   0,   0,  63, 240,   0,   0,
  0,   0, 127, 255, 128,   1, 255, 254,   0,   0,
  0,   1, 255, 255, 224,   7, 255, 255, 128,   0,
  0,   7, 255, 255, 240,  15, 255, 255, 224,   0,
  0,  15, 255, 255, 248,  31, 255, 255, 240,   0,
  0,  31, 255, 255, 252,  63, 255, 255, 248,   0,
  0,  63, 255, 255, 254, 127, 255, 255, 252,   0,
  0, 127, 255, 255, 255, 255, 255, 255, 254,   0,
  0, 255, 255, 255, 255, 255, 255, 255, 255,   0,
  0, 255, 255, 255, 255, 255, 255, 255, 255,   0,
  1, 255, 255, 255, 255, 255, 255, 255, 255, 128,
  1, 255, 255, 255, 255, 255, 255, 255, 255, 128,
  3, 255, 255, 255, 255, 255, 255, 255, 255, 192,
  3, 255, 255, 255, 255, 255, 255, 255, 255, 192,
  3, 255, 255, 255, 255, 255, 255, 255, 255, 192,
  7, 255, 255, 255, 255, 255, 255, 255, 255, 224,
  7, 255, 255, 255, 255, 255, 255, 255, 255, 224,
  7, 255, 255, 255, 255, 255, 255, 255, 255, 224,
  7, 255, 255, 255, 255, 255, 255, 255, 255, 224,
  7, 255, 255, 255, 255, 255, 255, 255, 255, 224,
  7, 255, 255, 255, 255, 255, 255, 255, 255, 224,
  7, 255, 255, 255, 255, 255, 255, 255, 255, 224,
  7, 255, 255, 255, 255, 255, 255, 255, 255, 224,
  7, 255, 255, 255, 255, 255, 255, 255, 255, 224,
  7, 255, 255, 255, 255, 255, 255, 255, 255, 224,
  7, 255, 255, 255, 255, 255, 255, 255, 255, 224,
  3, 255, 255, 255, 255, 255, 255, 255, 255, 192,
  3, 255, 255, 255, 255, 255, 255, 255, 255, 192,
  3, 255, 255, 255, 255, 255, 255, 255, 255, 192,
  1, 255, 255, 255, 255, 255, 255, 255, 255, 128,
  1, 255, 255, 255, 255, 255, 255, 255, 255, 128,
  0, 255, 255, 255, 255, 255, 255, 255, 255,   0,
  0, 255, 255, 255, 255, 255, 255, 255, 255,   0,
  0, 127, 255, 255, 255, 255, 255, 255, 254,   0,
  0, 127, 255, 255, 255, 255, 255, 255, 254,   0,
  0,  63, 255, 255, 255, 255, 255, 255, 252,   0,
  0,  31, 255, 255, 255, 255, 255, 255, 248,   0,
  0,  31, 255, 255, 255, 255, 255, 255, 248,   0,
  0,  15, 255, 255, 255, 255, 255, 255, 240,   0,
  0,   7, 255, 255, 255, 255, 255, 255, 224,   0,
  0,   3, 255, 255, 255, 255, 255, 255, 192,   0,
  0,   3, 255, 255, 255, 255, 255, 255, 192,   0,
  0,   1, 255, 255, 255, 255, 255, 255, 128,   0,
  0,   0, 255, 255, 255, 255, 255, 255,   0,   0,
  0,   0, 127, 255, 255, 255, 255, 254,   0,   0,
  0,   0,  63, 255, 255, 255, 255, 252,   0,   0,
  0,   0,  31, 255, 255, 255, 255, 248,   0,   0,
  0,   0,  15, 255, 255, 255, 255, 240,   0,   0,
  0,   0,   7, 255, 255, 255, 255, 224,   0,   0,
  0,   0,   3, 255, 255, 255, 255, 192,   0,   0,
  0,   0,   0, 255, 255, 255, 255,   0,   0,   0,
  0,   0,   0, 127, 255, 255, 254,   0,   0,   0,
  0,   0,   0,  63, 255, 255, 252,   0,   0,   0,
  0,   0,   0,  31, 255, 255, 248,   0,   0,   0,
  0,   0,   0,   7, 255, 255, 224,   0,   0,   0,
  0,   0,   0,   3, 255, 255, 192,   0,   0,   0,
  0,   0,   0,   1, 255, 255, 128,   0,   0,   0,
  0,   0,   0,   0, 127, 254,   0,   0,   0,   0,
  0,   0,   0,   0,  63, 252,   0,   0,   0,   0,
  0,   0,   0,   0,  15, 240,   0,   0,   0,   0,
  0,   0,   0,   0,   7, 224,   0,   0,   0,   0,
  0,   0,   0,   0,   1, 128,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0]
```

Images are Just Bits!



Hw4: ***lossless*** binary image compression



Binary Image

16 0s

16 1s

28 0

00000000
00000000
11111111
11111111
00000000
00000000
00000000
00001111

4 1s

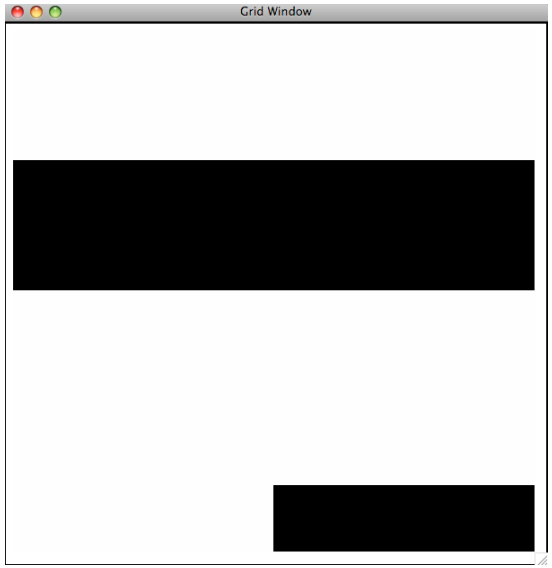
same-data streaks

Encoding as raw bits

one big string of 64 characters

a very compressible image...

Hw4: *lossless* binary image compression




```
00000000
00000000
11111111
11111111
00000000
00000000
00000000
00001111
```

same-data streaks

Binary Image

Encoding as raw bits

one big string of 64 characters

If our images tend to have <u>long streaks of unchanging data</u>, how might we represent it more efficiently, ***but still in binary***?

compress
uncompress

64 bits

"0000000000000000111111111111111100000000000000000000000000001111"

16 **0**'s 16 **1**'s 28 **0**'s 4 **1**'s

Hw4: lossless image compression

64 bits

"0000000000000000111111111111111100000000000000000000000000001111"

0 is the first digit

There are 16 of them.

1 is the next digit

Again, there are 16 of them.

0 is the next digit

There are 28

1 is the final digit

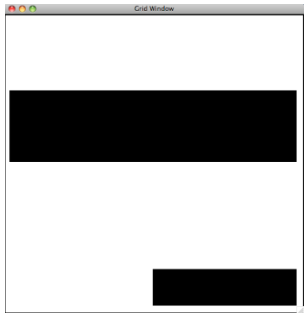
There are 4

0100001100000111001100

0100001100000111001100

22 bits !!!

00000000
00000000
11111111
11111111
00000000
00000000
00000000
00001111



One possible algorithm:

bit #repeats

Any problems with this?

Hw4: lossless image compression

0 is the first digit

and there are 1,098,188 of them.

0100001100000111001100

It's ambiguous! this could just be a ***huge number*** of 0 pixels!

0100001100000111001100

our algorithm:

bit #repeats

could be misinterpreted!

00000000
00000000
00000000
00000000
00000000
00000000
00000000
000000...

1,098,188 zeros!

huge empty image

fixed-width compression

64 bits

"0000000000000000111111111111111100000000000000000000000000001111"

0 is the first digit

There are 16 of them.

1 is the next digit

Again, there are 16 of them.

28 zeros

and so on...

4 ones

0001000010010000000111001000010 0

7 bits: # of repeats

7 bits: # of repeats

8-bit data block

8-bit data block

8-bit data block

8-bit data block

00000000
00000000
11111111
11111111
00000000
00000000
00000000
00001111

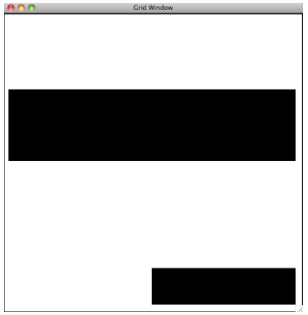
00010000100100000001110010000100

32 bits !!!

We need ***fixed-width*** blocks:

bit	#repeats
1 bit fill	7 bits for the # of repeats

← **8-bits in each "block"** →



fixed-width compression

64 bits

"0000000000000000111111111111111100000000000000000000000000001111"

0 is the first digit

There are 16 of them.

1 is the next digit

Again, there are 16 of them.

28 zeros

and so on...

4 ones

0001000010010000000111001000010 0

7 bits: # of repeats

7 bits: # of repeats

8-bit data block

8-bit data block

8-bit data block

8-bit data block

00000000
00000000
11111111
11111111
00000000
00000000
00000000
00001111

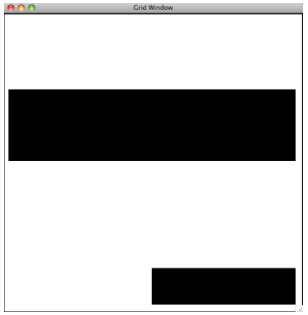
00010000100100000001110010000100

32 bits !!!

We need ***fixed-width*** blocks:

bit	#repeats
1 bit fill	7 bits for the # of repeats

← **8-bits in each "block"** →



WIKIPEDIA
The Free Encyclopedia

Article Talk

Run-length encoding

From Wikipedia, the free encyclopedia

If you use ***7 bits*** to hold the # of consecutive repeats, what is the largest number of bits that *one block can represent*?

00010000

1 bit: the initial pixel

7 bits: # of repeats

8-bit total data block

7 bits?

B bits?

*What if you need a **larger** # of repeats?*

hw4 pr2

```
def compress( I ):
    """ returns the RLE of the
        input binary image, I """
```

a binary image, **I**

"000000000000000000000000000000000000000000111111111111111111111111111111"

42 zeros

31 ones

"001010101001111 1"

42, in binary

31, in binary

repeated bit

# of repeats

the "compressed" output returned by **compress(I)**

Names: _____

```
def compress( I ):
    """ returns the RLE of the
        input binary image, I """
```

a 64-bit binary image, **I**

IQuiz

"000000000000111111111111111111110000000000000000000011111111111"

12 zeros | 20 ones | 21 zeros | 11 ones

compress(I)

1 bit | 7 bits | 1 bit | 7 bits | 1 bit | 7 bits | 1 bit | 7 bits

the "compressed" output, returned by compress(I)

Then, discuss

(1) What helper function would be useful for this **compress** process?

(2) What's an image **I** whose compressed output ***gets larger, not smaller***?

(2a) What are the BEST-compressible / WORST-compressible 64-bit images?

(2b) How could you ***improve*** the algorithm so that it always compresses?!!

hw4 pr2

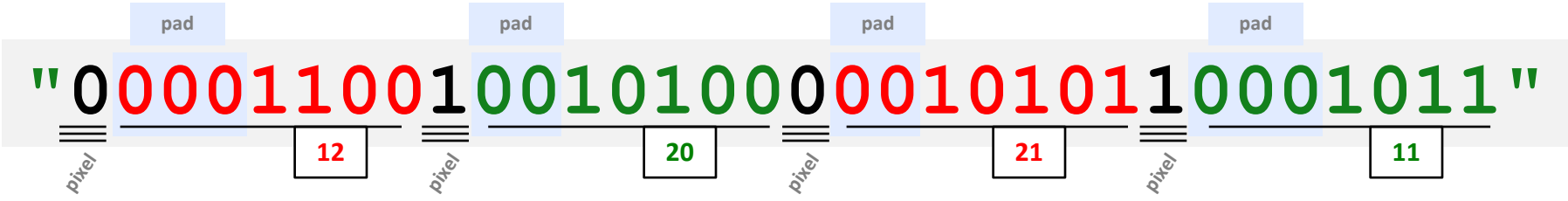
```
def compress( I ):
    """ returns the RLE of the
        input binary image, I """
```

Helper function?

a binary image, **IQuiz**

"000000000000111111111111111111110000000000000000000011111111111"

12 zeros | 20 ones | 21 zeros | 11 ones



the "compressed" output, returned by compress(I)

hw4 pr2

```
def compress( I ):
    """ returns the RLE of the
        input binary image, I """
```

a binary image, **IQuiz**

"000000000000111111111

de

a

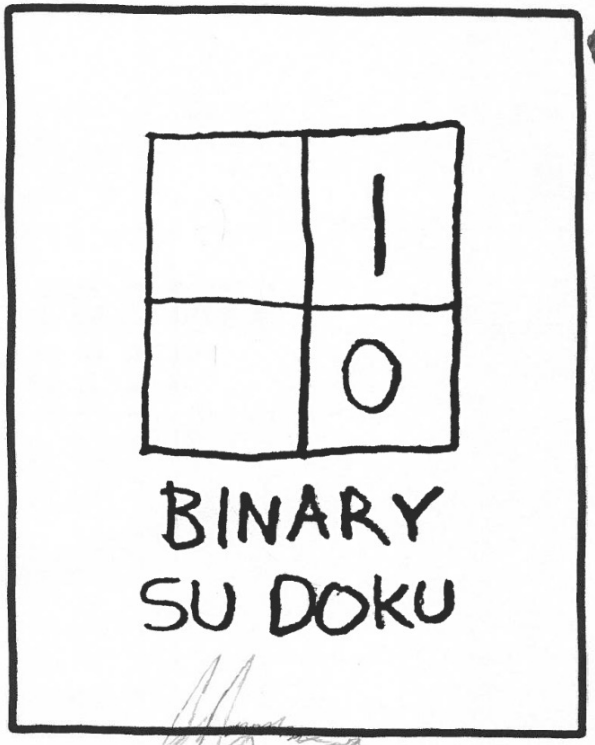
"

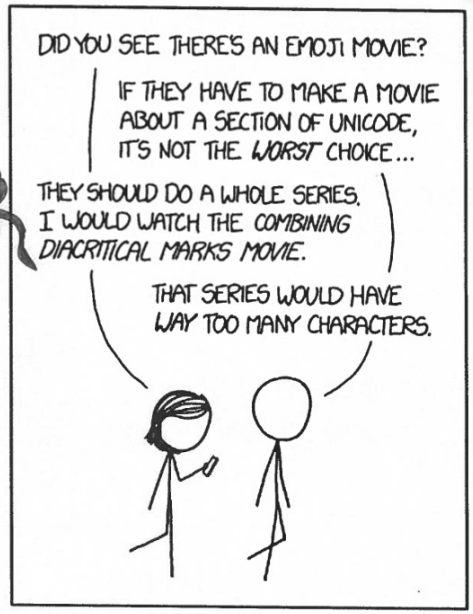
"

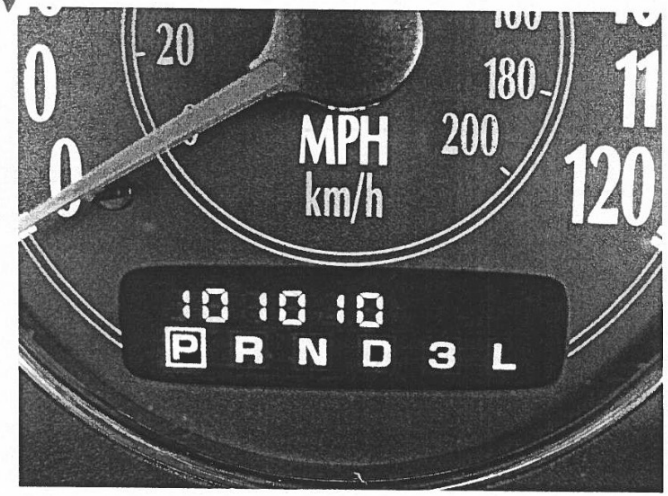
"

the "compressed" output, returned by compress(1)

The best thing about a Boolean is that even if you are wrong, you are only off by a bit.



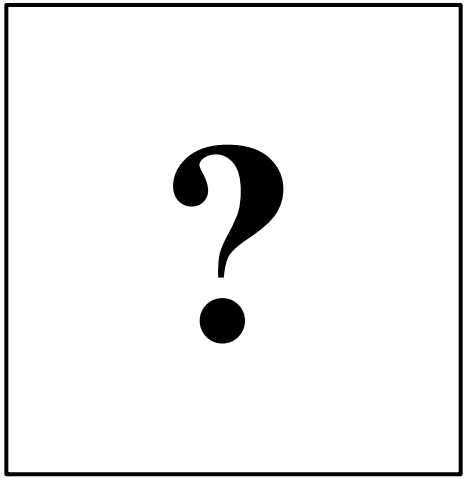




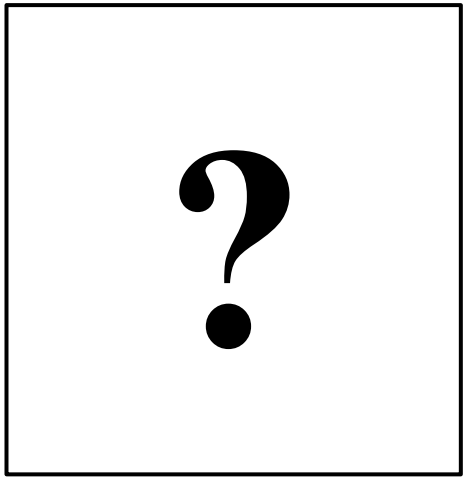
shortest compressed representation

longest compressed representation

What are the **BEST** and the **WORST** compression results you can get for an 8x8 image input (64 bits)?



BEST



WORST

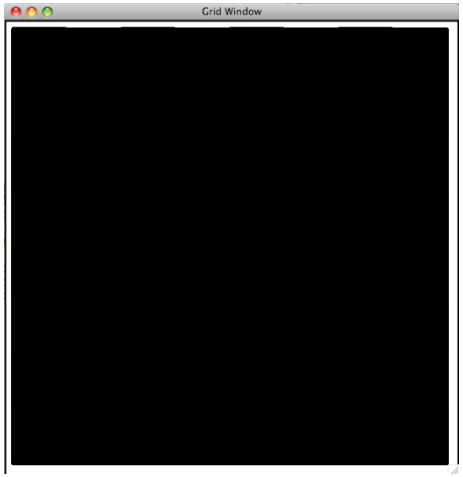
How could we improve this compression algorithm so that ***all images*** compress to smaller than the originals? That is, how can we make compression always work ?

?

shortest compressed representation

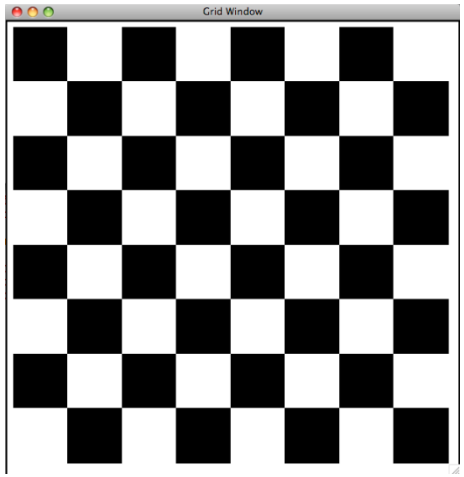
longest compressed representation

What are the **BEST** and the **WORST** compression results you can get for an 8x8 image input (64 bits)?



BEST

only 8 bits total!



WORST

aargh! 512 bits!

Anyone see why this is NOT QUITE the ***worst-compressable*** image?

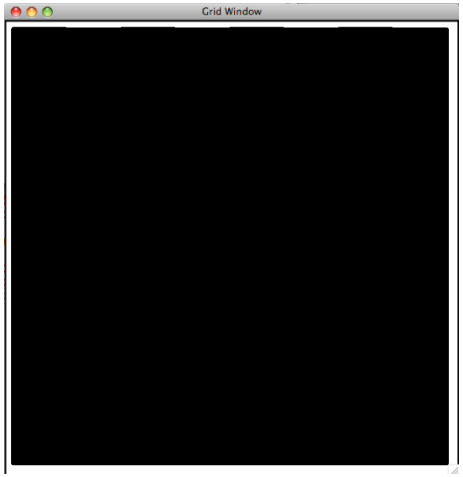
How could we improve this compression algorithm so that ***all images*** compress to smaller than the originals? That is, how can we make compression always <u>work</u> ?

?

shortest compressed representation

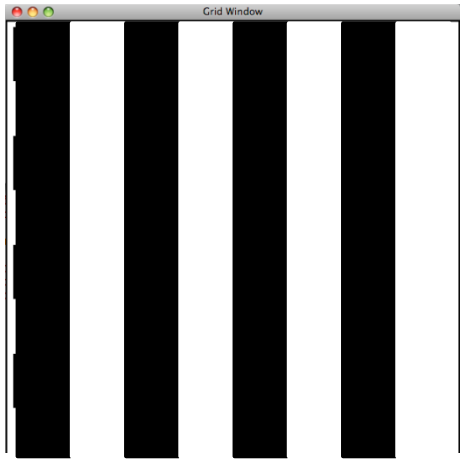
longest compressed representation

What are the BEST and the WORST compression results you can get for an 8x8 image input (64 bits)?



BEST

only 8 bits total!



WORST

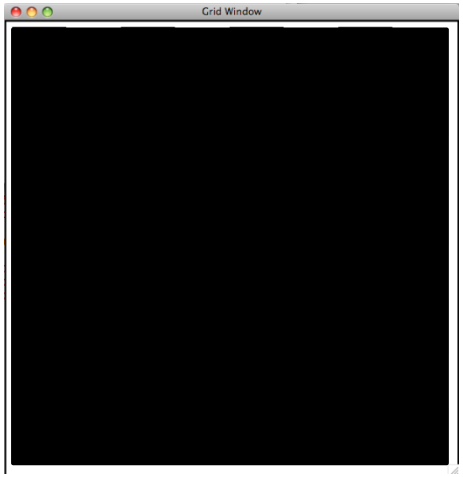
aargh! 512 bits!

How could we improve this compression algorithm so that ***all images*** compress to smaller than the originals? That is, how can we make compression always <u>work</u> ?

shortest compressed representation

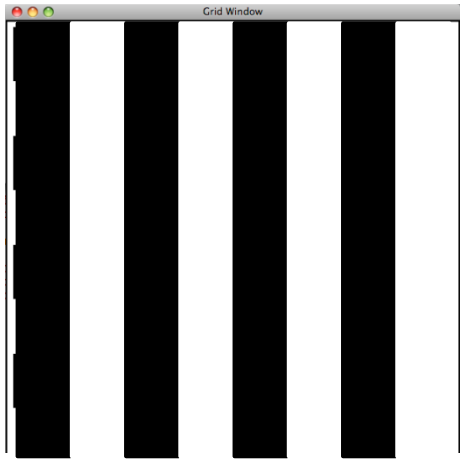
longest compressed representation

What are the BEST and the WORST compression results you can get for an 8x8 image input (64 bits)?



BEST

only 8 bits total!



WORST

aargh! 512 bits!

Impossible! *Provably!*

How could we improve this compression algorithm so that ***all images*** compress to smaller than the originals? That is, how can we make compression always work ?

!

Helper fun.!

`frontNum(S)` returns the # of times the first element of the input **S** appears consecutively ***at the start*** of S:

```
frontNum('1111010')
4
frontNum('00110010')
2
```

S

1 3

S

frontNum('01')

or 2 base cases:

```
len(S)== 0:
len(S)== 1:
```

or 3 tests :

```
S== '' or
S== '1' or
S== '0'
```

```
def frontNum(S):

    if len(S) <= 1:
        return len(S)

    elif S[0]==S[1] :
        return 1+ frontNum(S[1:])

    else:
        return 1
```

1 base case:

If the first two bits **DO** match....

If the first two bits ***DON'T*** match....

Grutors!!!

f Week	Timespan (pacific)	Grutor (name+email)
Thursdays	**No Thursday tutoring from 5:30-8:00pm (This is the cs5 grading meeting, at least til 7.)**	
	8:00-10:00pm (McGregor lab)	Katie Baakkonen (kabaakkonen@g.hmc.edu)
	8:00-10:00pm (McGregor lab)	Nico Villalba (nvillalba@hmc.edu)
Date/Day of Week	**Timespan (pacific)**	Grutor (name+email)
Fridays	1:00-3:00pm (OUTSIDE McGregor lab)	Norah Pack (npack@hmc.edu)
	1:00-3:00pm (OUTSIDE McGregor lab)	Isaac Chung (ischung@hmc.edu)
	1:00-3:00pm (OUTSIDE McGregor lab)	Michelle Lum (milum@hmc.edu)
	3:00-5:00pm (McGregor lab)	Isabelle Kemp (ikemp@hmc.edu)
	3:00-5:00pm (McGregor lab)	Laney Goldman (lgoldman@hmc.edu)
	3:00-5:00pm (McGregor lab)	David Aguirre (daguirre@hmc.edu)
Saturdays	1:00-3:00pm (McGregor lab)	Henry Trinh (htrinh@hmc.edu)
	1:00-3:00pm (McGregor lab)	Lydia Yang (lydyang@hmc.edu)
	3:00-5:00pm (McGregor lab)	Waverly Wang (wwang@g.hmc.edu)
	3:00-5:00pm (McGregor lab)	Richard Chang (richang@g.hmc.edu)
	7:00-9:00pm (McGregor lab)	Jessica Lin (jelin@hmc.edu)
	7:00-9:00pm (McGregor lab)	Audrey Cole (apcole@hmc.edu)

It's <u>all</u> bits!

images, text, sounds, data, …

even the string `'Hi *'` is just a sequence of bits…

`'Hi *'`

4 ASCII characters
8 bits each

4*8 == 72 bits total

`01001000 01101001 00100000 00101010`

`ord('H') == 72` `ord('i') == 105` `ord(' ') == 32` `ord('*') == 42`

All computation boils down to manipulating bits!

All computation is simply ***functions of bits***

binary inputs **S** and **T** → output, **S+T**

S	T	S+T
00	00	000
00	01	001
00	10	010
00	11	011
01	00	001
01	01	010
01	10	011
01	11	100
10	00	010
10	01	011
10	10	100
10	11	101
11	00	011
11	01	100
11	10	101
11	11	110

This week: you'll build this in Python.

bitwise addition function

addB

Next week: you'll design this with *wires*!

```
if S[-1] == '0' and T[-1] == '0':
    return _____________________ + '0'
```

binary inputs **S** and **T** → output, **S+T**

S	T	S+T
00	00	000
00	01	001
00	10	010
00	11	011
01	00	001
01	01	010
01	10	011
01	11	100
10	00	010
10	01	011
10	10	100
10	11	101
11	00	011
11	01	100
11	10	101
11	11	110

This week: you'll build this in Python.

bitwise addition function

addB

```
if S[-1] == '1' and T[-1] == '1' :
    return _____________________________ + '0'
```

binary inputs **S** and **T**

S	T	output, S+T
00	00	000
00	01	001
00	10	010
00	11	011
01	00	001
01	01	010
01	10	011
01	11	100
10	00	010
10	01	011
10	10	100
10	11	101
11	00	011
11	01	100
11	10	101
11	11	110

This week: you'll build this in Python.

bitwise addition function

addB

Adding strings?

is **circuit** *addition!*

is **syntactic** *addition!*

syntactic ~ *"meaning-free"*

Multiplying by machine:

is **circuit** *multiplying!*

is **syntactic** *multiplying!*

Doing anything by machine…

is **circuit** *interaction!*

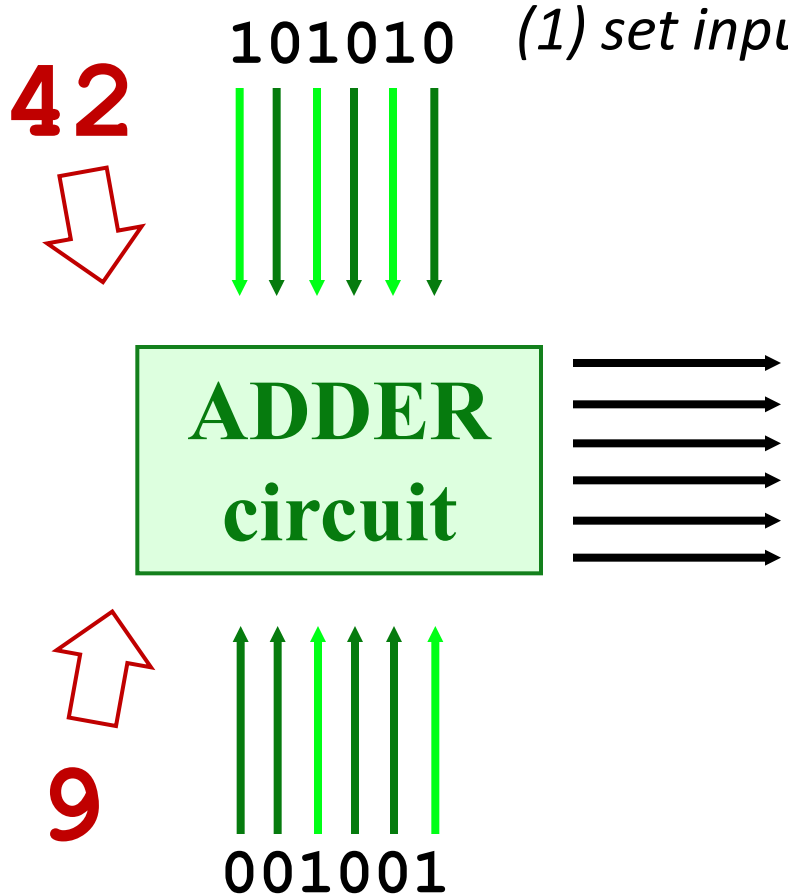
is **syntactic** *interaction!*

means it can be done purely via ***surface syntax****, which means it can be done* ***without thinking****…*

In a computer, each bit is represented as a <u>voltage</u> (**1** is +5v and **0** is 0v)

Computation is simply the **deliberate combination** of those voltages!

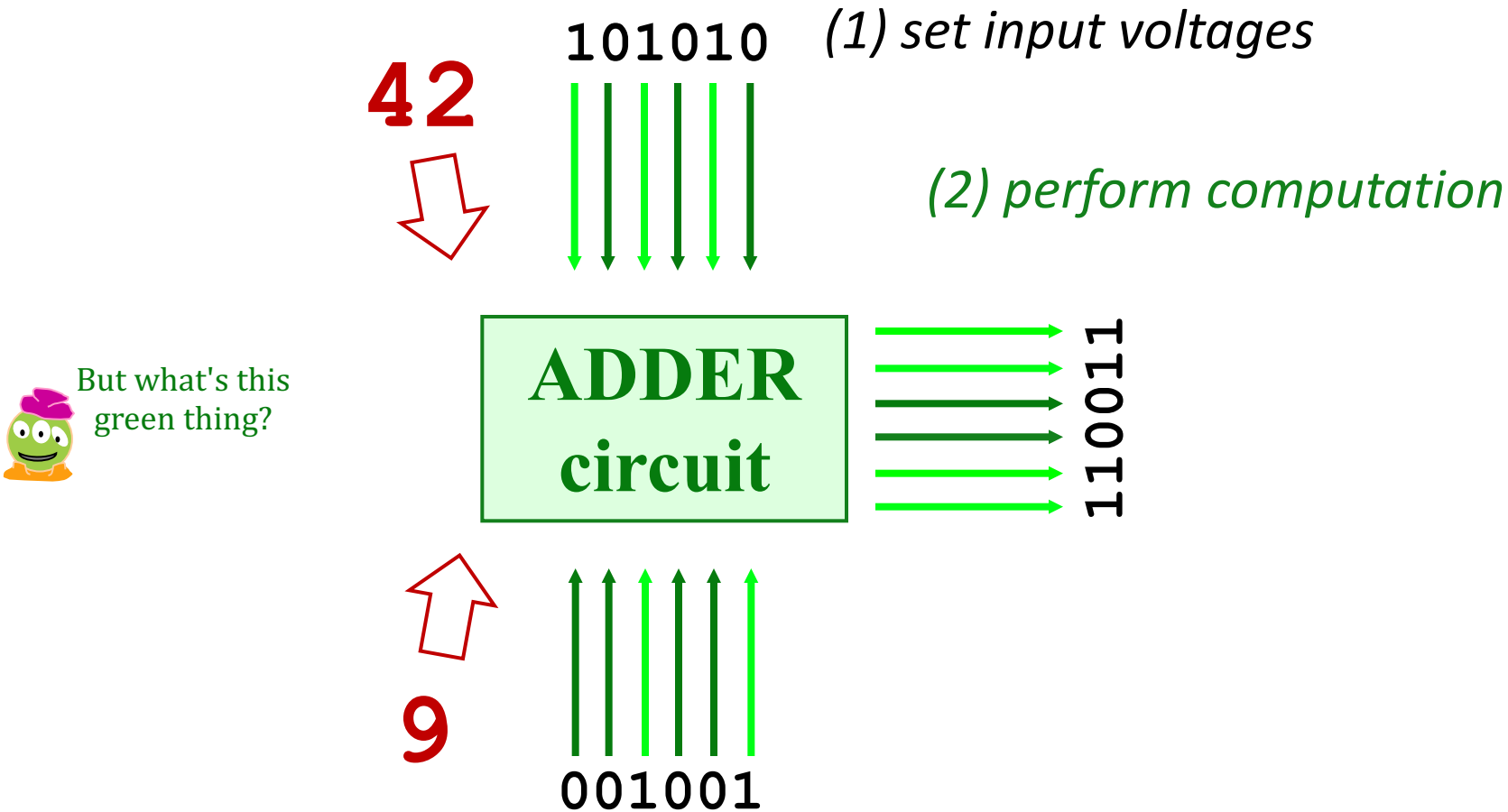
(1) set input voltages



But what's this green thing?

In a computer, each bit is represented as a <u>voltage</u> (**1** is +5v and **0** is 0v)

Computation is simply the **deliberate combination** of those voltages!

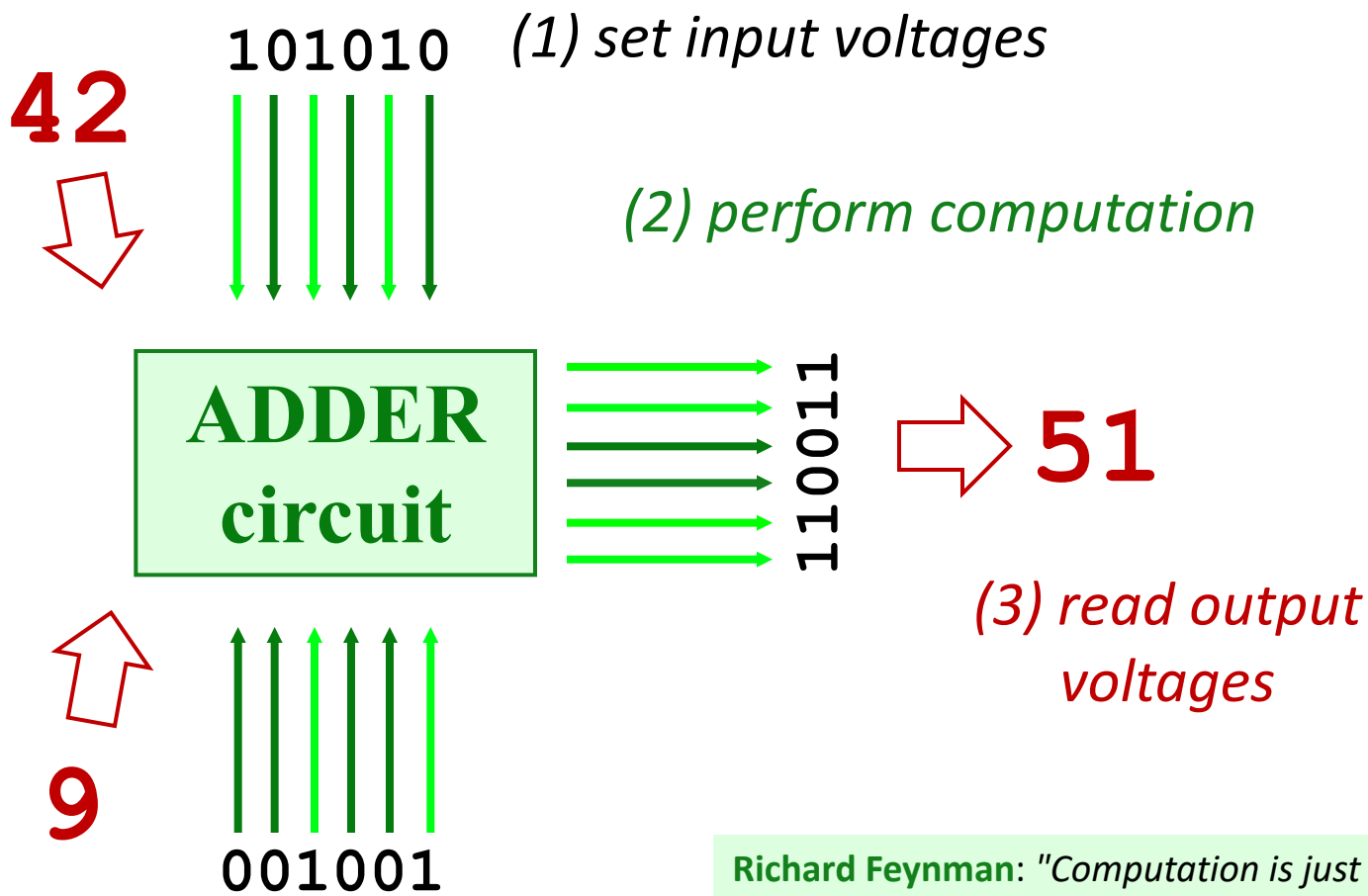


But what's this green thing?

In a computer, each bit is represented as a <u>voltage</u> (**1** is +5v and **0** is 0v)

Computation is simply the **deliberate combination** of those voltages!

But what's this green thing?



Richard Feynman: *"Computation is just a physics experiment that always works!"*

Our building blocks: ***logic gates***

AND outputs 1 only if **ALL** inputs are 1

AND

OR outputs 1 if **ANY** input is 1

OR

NOT reverses its input

NOT

These circuits are ***physical*** functions of bits…

… and *all* mathematical functions can be built from them!

Our building blocks: ***logic gates***

AND outputs 1 only if **ALL** inputs are 1

AND

OR outputs 1 if **ANY** input is 1

OR

NOT reverses its input

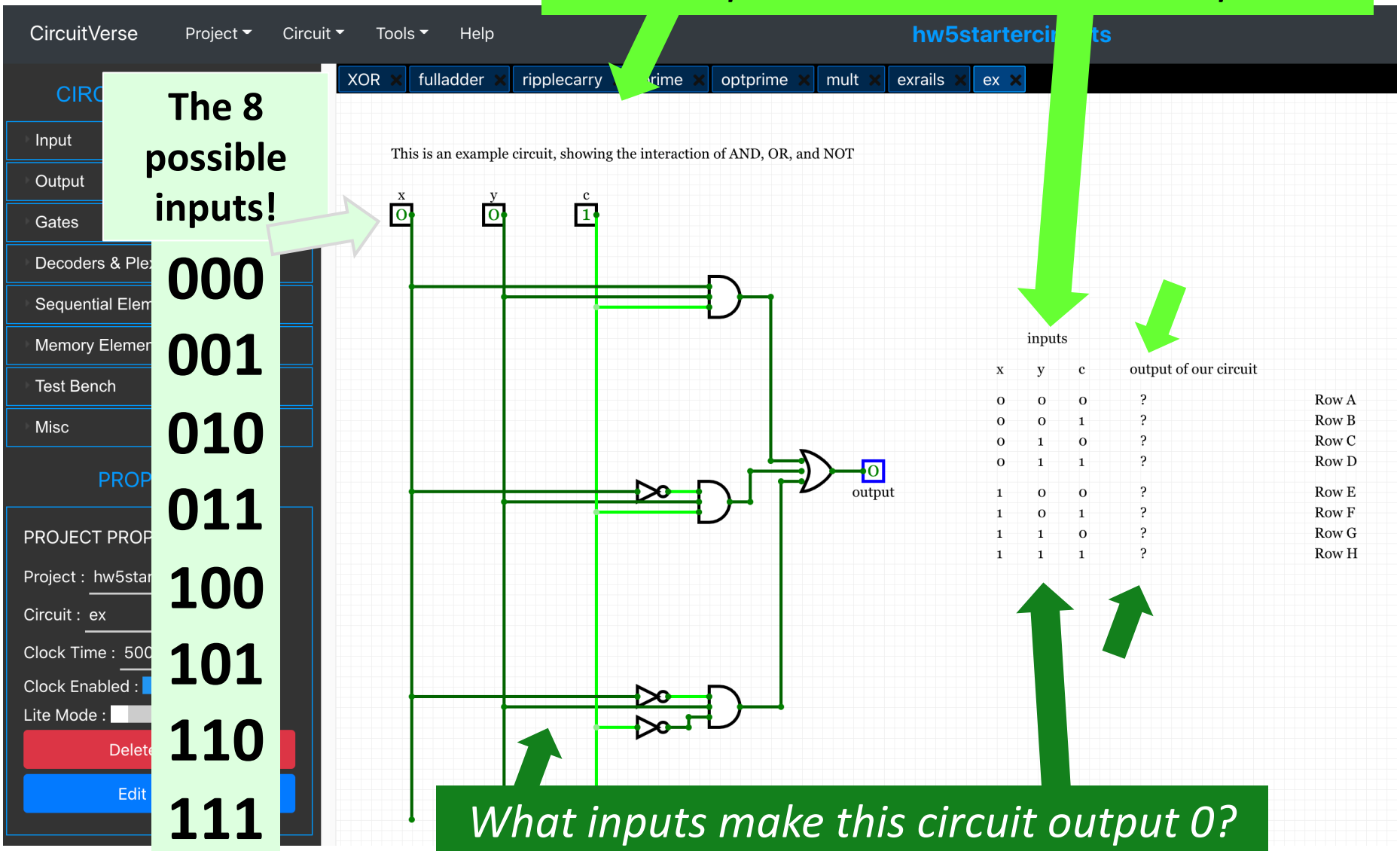
NOT

ALL circuits are **ANY** tions of bits…

… and *all* mathematical functions can be built from them!

From gates to ***circuits***...

What inputs make this circuit output 1?



The 8 possible inputs!

000
001
010
011
100
101
110
111

inputs

x	y	c	output of our circuit	
0	0	0	?	Row A
0	0	1	?	Row B
0	1	0	?	Row C
0	1	1	?	Row D
1	0	0	?	Row E
1	0	1	?	Row F
1	1	0	?	Row G
1	1	1	?	Row H

What inputs make this circuit output 0?

from circuit design…

next 2 weeks

…to a full computer!

Have an outst ingly

gl ious weekend!

Why ?!

AND

OR

NOT