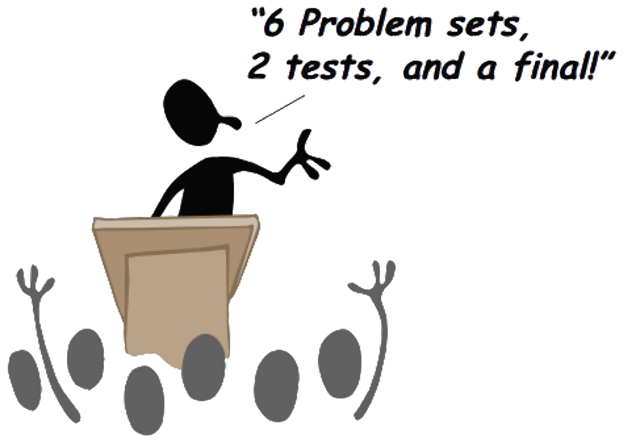
4-1-1: Information Please

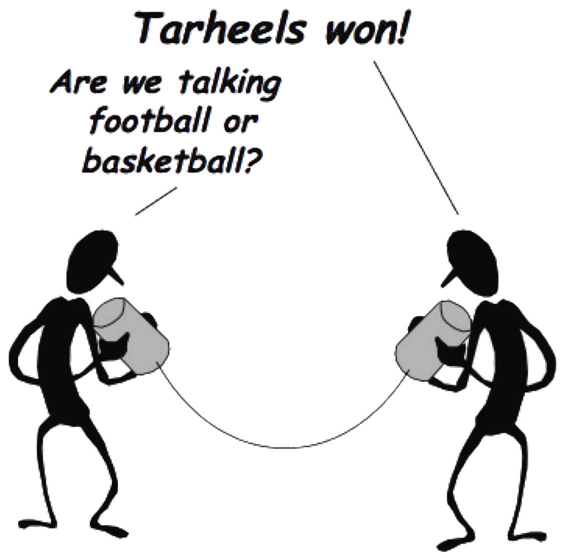
"2 bits, 4 bits, 6 bits a byte!"

- Representing Information as bits
- Number Representations
- Other bits

What is "Information"?

information, n. Knowledge communicated or received concerning a particular fact or circumstance.





A Computer Scientist's definition:

Information resolves uncertainty. Information is simply that which cannot be predicted. The less likely a message is, the more information it conveys.

Quantifying Information

(Claude Shannon, 1948)

Suppose you're faced with **N** equally probable choices, and I give you a fact that narrows it down to **M** choices. Then you've been given:

$$\log_2(N/M) \text{ bits of information}$$

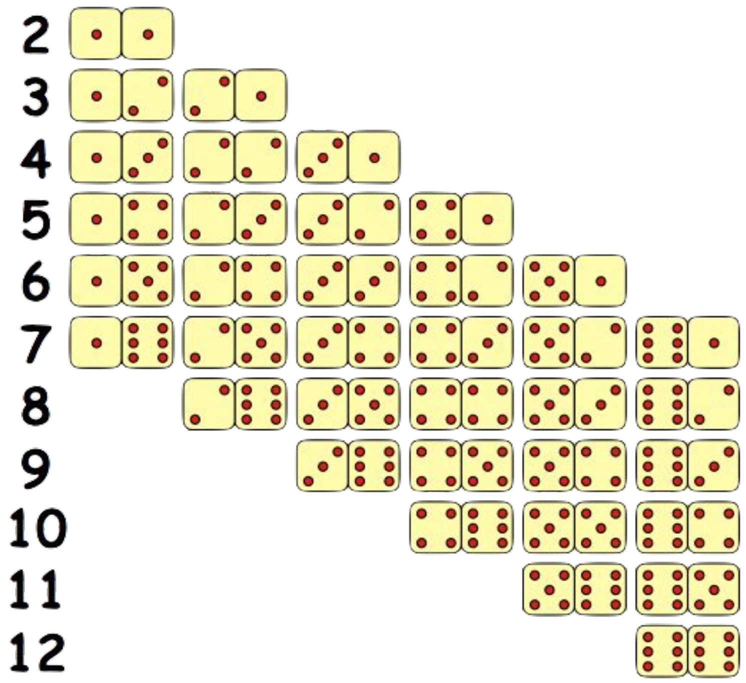
Information is measured in bits (binary digits) = number of 0/1's required to encode choice(s)

Examples:

- Outcome of a coin flip: $\log_2(2/1) = 1$ bit
- The roll of one die? $\log_2(6/1) = \sim 2.6$ bits
- Someone tells you that their 7-digit phone number is a palindrome?

$$\log_2(10^7/10^4) = \sim 9.966 \text{ bits}$$

Another Example: Sum of 2 dice



$i_2 = \log_2(36/1) = 5.170$ bits
$i_3 = \log_2(36/2) = 4.170$ bits
$i_4 = \log_2(36/3) = 3.585$ bits
$i_5 = \log_2(36/4) = 3.170$ bits
$i_6 = \log_2(36/5) = 2.848$ bits
$i_7 = \log_2(36/6) = 2.585$ bits
$i_8 = \log_2(36/5) = 2.848$ bits
$i_9 = \log_2(36/4) = 3.170$ bits
$i_{10} = \log_2(36/3) = 3.585$ bits
$i_{11} = \log_2(36/2) = 4.170$ bits
$i_{12} = \log_2(36/1) = 5.170$ bits

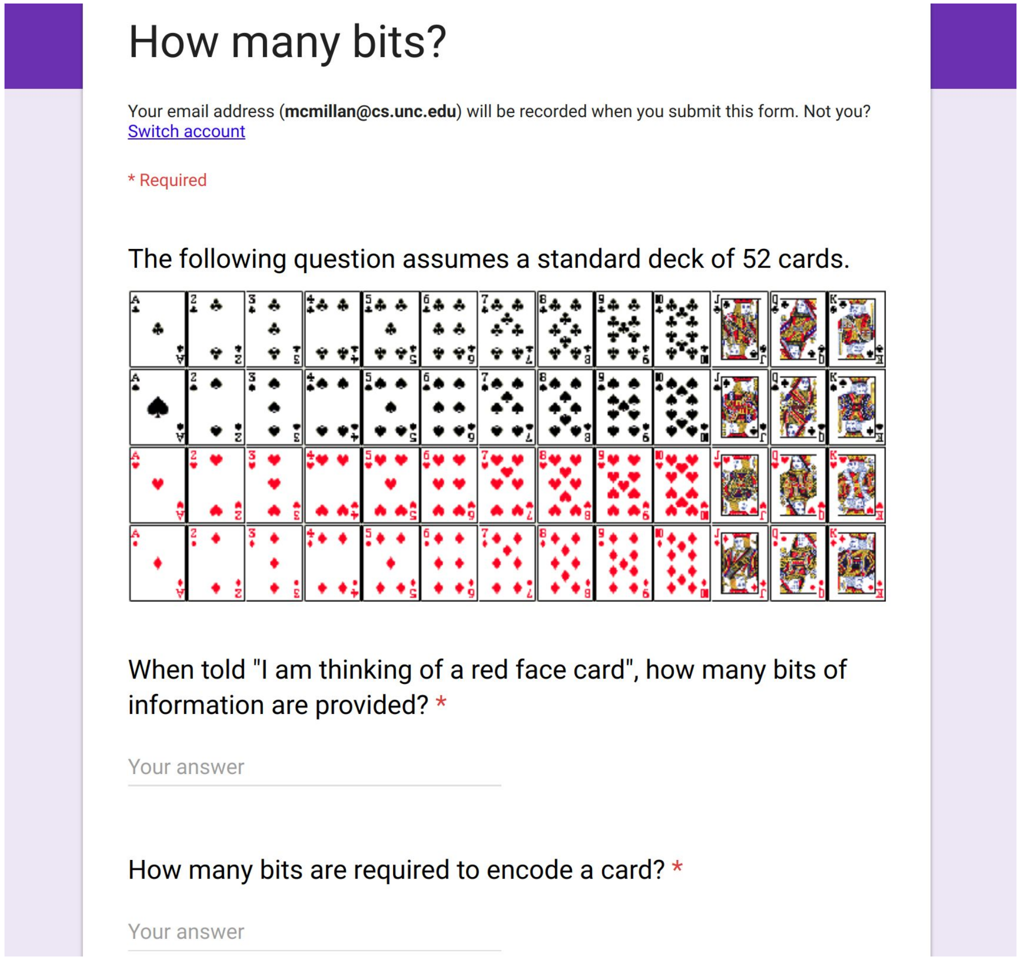
The average information provided by the sum of 2 dice is: **3.274**

$$i_{ave} = \sum_{i=2}^{12} \frac{M_i}{N} \log_2\left(\frac{N}{M_i}\right) = -\sum_i p_i \log_2(p_i)$$

The average information of a process is called its **Entropy**.

Try it:

Go to: https://goo.gl/forms/blRxECRnFvWGBTlE3



How many bits?

Your email address (**mcmillan@cs.unc.edu**) will be recorded when you submit this form. Not you? Switch account

* Required

The following question assumes a standard deck of 52 cards.

When told "I am thinking of a red face card", how many bits of information are provided? *

Your answer

How many bits are required to encode a card? *

Your answer

SHOW ME THE BITS!

- Is there a concrete ENCODING that achieves its information content?
- Can the sum of two dice REALLY be represented using 3.274 bits?
- If so, how?
- The fact is, the average information content is a strict lower-bound on how small of a **representation** that we can achieve.
- In practice, it is difficult to reach this bound. But, we can come very close.

Variable-Length Encoding

- Of course we can use differing numbers of "bits" to represent each item of data
- This is particularly useful if all items are not equally likely
- Equally likely items lead to fixed length encodings:
  - Ex. Encode a "particular" roll of 5?
  - {(1,4),(2,3),(3,2),(4,1)} which are equally likely if we use fair dice

$$Entropy = \sum_{i=1}^{4} p(roll_i|roll = 5)log_2(p(roll_i|roll = 5)) = \sum_{i=1}^{4} \frac{1}{4} log_2(\frac{1}{4}) = 2$$

  - 00 = (1,4), 01 = (2,3), 10 = (3,2), 11 = (4,1)

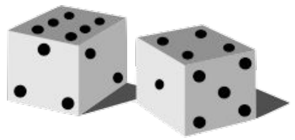
- Back to the original problem. Let's use this encoding:

2 = 10011	3 = 0101	4 = 011	5 = 001
6 = 111	7 = 101	8 = 110	9 = 000
10 = 1000	11 = 0100	12 = 10010	

Variable-Length Decoding

2 = 10011	3 = 0101	4 = 011	5 = 001
6 = 111	7 = 101	8 = 110	9 = 000
10 = 1000	11 = 0100	12 = 10010	

- Notice how unlikely rolls are encoded using more bits, whereas likely rolls use fewer bits



- Let's use our encoding to decode the following bit sequence

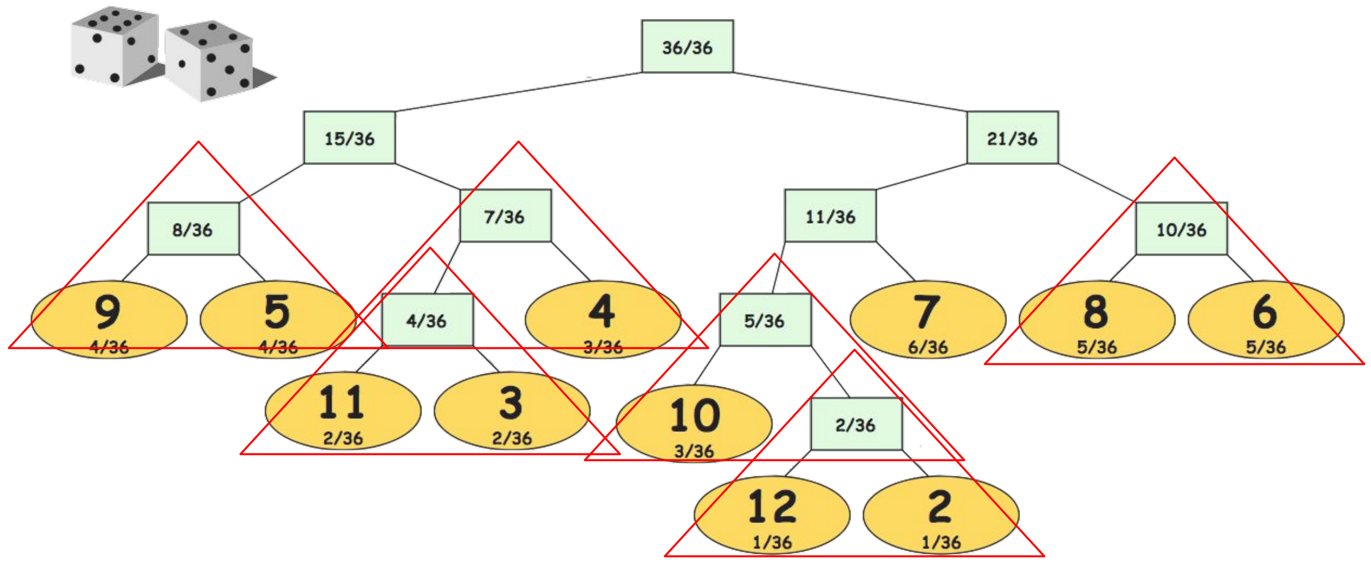
```
2    5  3   6  5  8  3
1001100101011110011100101
```

- Where did this code come from?

Huffman Coding

A simple **greedy** algorithm for approximating a minimum encoding

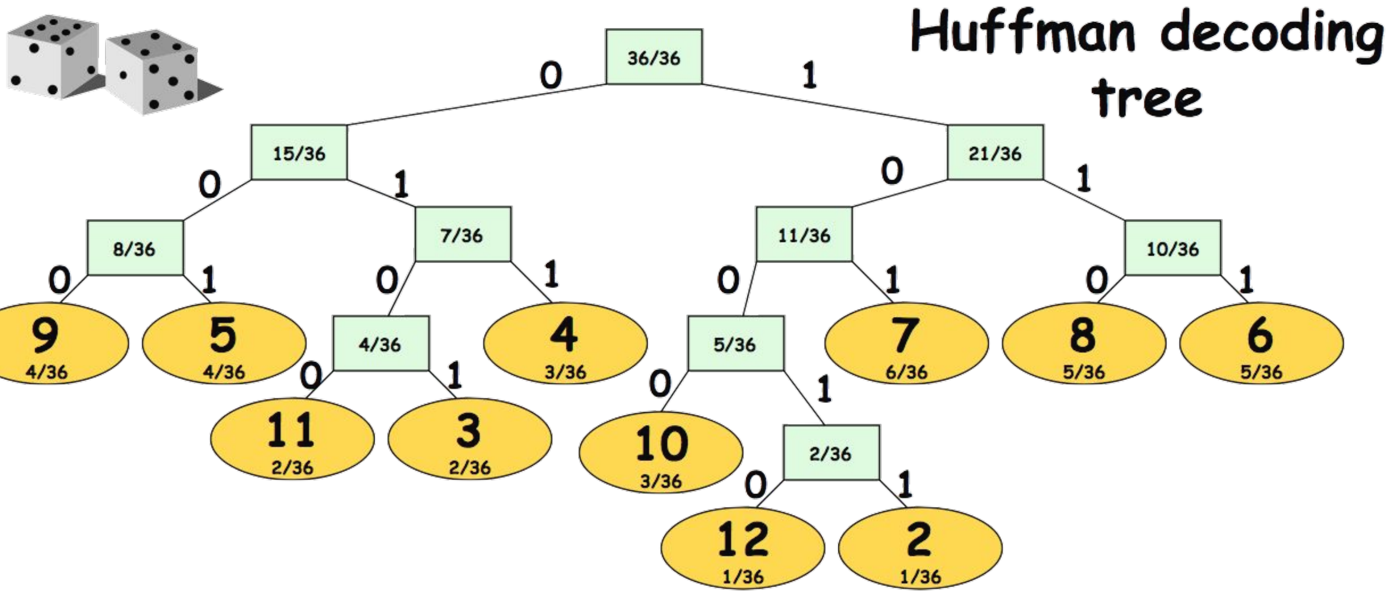
1. Find the 2 items with the smallest probabilities
2. Join them into a new *meta* item with probability of their sum
3. Remove the two items and insert the new meta item
4. Repeat from step 1 until there is only one item



Converting A Tree to an Encoding

Once the *tree* is constructed, label its edges consistently and follow the paths from the largest *meta* item to each of the real items to find the encoding.

2 = 10011	3 = 0101	4 = 011	5 = 001	6 = 111	7 = 101
8 = 110	9 = 000	10 = 1000	11 = 0100	12 = 10010	



Coding Efficiency

How does this code compare to the information content?

$$b_{ave} = \frac{1}{36}5 + \frac{2}{36}4 + \frac{3}{36}3 + \frac{4}{36}3 + \frac{5}{36}3 + \frac{6}{36}3 + \frac{5}{36}3 + \frac{4}{36}3 + \frac{3}{36}4 + \frac{2}{36}4 + \frac{1}{36}$$

$$b_{ave} = 3.306$$

Pretty close. Recall that the lower bound was 3.274 bits.

However, an efficient encoding (as defined by having an average code size close to the information content) is not always what we want!

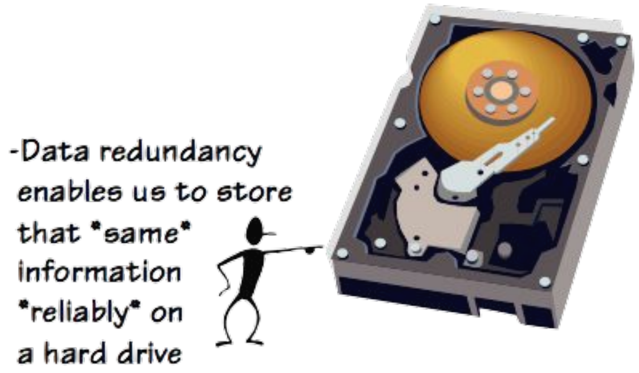
Sometimes a uniform code is easier to deal with.

Sometimes redundancy is a good thing.

Encoding Considerations

- Encoding schemes that attempt to match the information content of a data stream **remove redundancy**. They are **data compression** techniques.
- Make the information easier to manipulate (fixed-sized encodings)
- However, sometimes our goal in encoding information is **increase redundancy,** rather than remove it. Why?
- Adding redundancy can make data resilient to noise (**error detecting and correcting codes**)

-Data compression allows us to store our entire music and video collections in a pocketable device


-Data redundancy enables us to store that "same" information "reliably" on a hard drive

Information Encoding Standards

- "Encoding" is the process of assigning representations to information
- Choosing an appropriate and efficient encoding is an engineering challenge (and an art)
- Impacts design at many levels
  - Mechanism (devices, # of components used)
  - Efficiency (# bits used)
  - Reliability (tolerate noise)
  - Security (encryption)

Fixed-Size Codes

If all choices are equally likely (or we have no reason to expect otherwise), then a fixed-size code is often used. Such a code should use at least enough bits to represent the information content.

ex. Decimal digits 10 = {0,1,2,3,4,5,6,7,8,9}

4-bit BCD (binary coded decimal)

$\log_2(10/1) = 3.322 < 4$ bits

BCD		
0	-	0000
1	-	0001
2	-	0010
3	-	0011
4	-	0100
5	-	0101
6	-	0110
7	-	0111
8	-	1000
9	-	1001

ex. ~84 English characters = {A-Z (26), a-z (26), 0-9 (10), punctuation (8), math (9), financial (5)}

7-bit ASCII (American Standard Code for Information Interchange)

$\log_2(84/1) = 6.392 < 7$ bits

ASCII

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
000	NUL	SOH	STX	ETX	EOT	ACK	ENQ	BEL	BS	HT	LF	VT	FF	CR	SO	SI
001	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
010		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
011	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
100	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
101	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
110	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
111	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

- For letters upper and lower case differ in the 6th "shift" bit

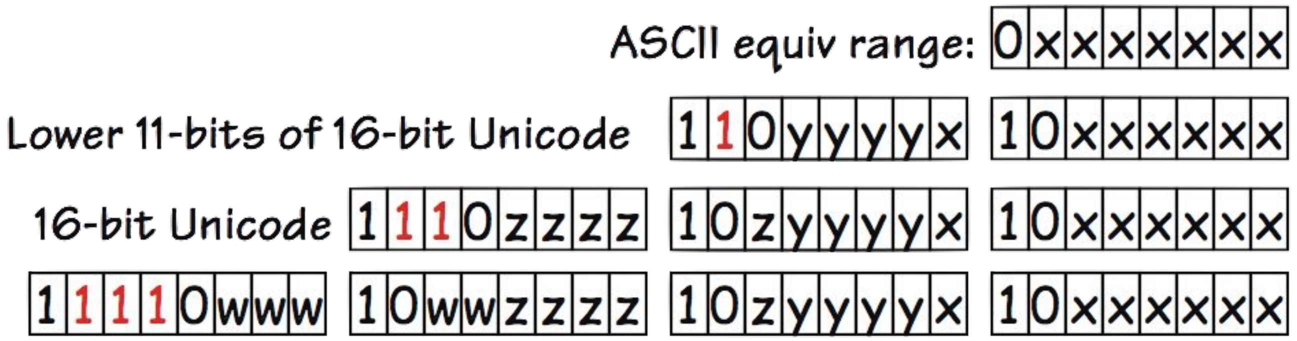
  10XXXXX is upper, and 11XXXXX is lower
- Special "control" characters set upper two bits to 00

  ex. cntl-g → bell, cntl-m → carriage return, cntl-[→ esc
- This is why bytes have 8-bits (ASCII + optional parity). Historically, there were computers built with 6-bit bytes, which required a special "shift" character to set case.

Unicode

- ASCII is biased towards western languages. English in particular.
- There are, in fact, many more than 256 characters in common use:

  â, ö, ß, ñ, è, ¥, £, 揗, 敇, 횔, 力, א, ג, ж, š
- Unicode is a worldwide standard that supports all languages, special characters, classic, extinct, and arcane.
- Several encoding variants 16-bit (UTF-8)
- Variable length (as determined by first byte)

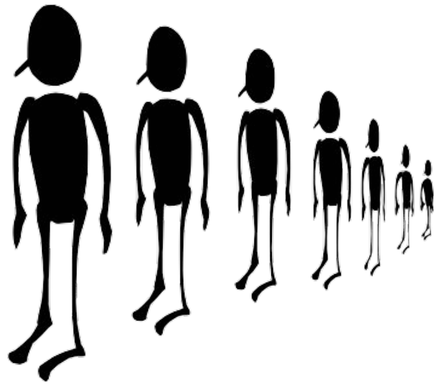

ASCII equiv range: 0xxxxxxx
Lower 11-bits of 16-bit Unicode 110yyyyx 10xxxxxx
16-bit Unicode 1110zzzz 10zyyyyx 10xxxxxx
11110www 10wwzzzz 10zyyyyx 10xxxxxx

Encoding Positive Integers

It is straightforward to encode positive integers as a sequence of bits. Each bit is assigned a weight. Ordered from right to left, these weights are increasing powers of 2. The value of an n-bit number encoded in this fashion is given by the following formula:

$$v = \sum_{i=0}^{n-1} 2^i b_i$$

2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	0	0	0	1	1	1	1	1	1	0	0	0	1	0



$$\begin{array}{lrr} & 2^1 = & 2 \\ + & 2^5 = & 32 \\ + & 2^6 = & 64 \\ + & 2^7 = & 128 \\ + & 2^8 = & 256 \\ + & 2^9 = & 512 \\ + & 2^{10} = & \underline{1024} \\ & & 2018 \end{array}$$

Favorite Bits

- You are going to have to get accustomed to working in binary. Specifically for Comp 411, but it will be helpful throughout your career as a computer scientist.
- Here are some helpful guides:
  1. Memorize the first 10 powers of 2

$2^0 = 1$	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$
$2^5 = 32$	$2^6 = 64$	$2^7 = 128$	$2^8 = 256$	$2^9 = 512$

  2. Memorize the prefixes for powers of 2 that are multiples of 10

2^{10} = Kilo (1024)	2^{40} = Tera (1024^4)
2^{20} = Mega (1024*1024)	2^{50} = Peta (1024^5)
2^{30} = Giga (1024*1024*1024)	2^{60} = Exa (1024^6)

Tricks with Bits

- The first thing that you'll do a lot of is cluster groups of contiguous bits.
- Using the binary powers that are multiples of 10 we can do the most basic clustering.

1. When you convert a binary number to decimal, first break it down from the right into clusters of 10 bits.
2. Then compute the value of the leftmost remaining bits (1)
3. Find the appropriate prefix (GIGA)
4. Often this is sufficient (might need to round up)

$$01\boxed{0001100000}\boxed{0001100000}\boxed{0000101000}$$

A "Giga" something or other

Other Helpful Clusterings

Oftentimes we will find it convenient to cluster groups of bits together for a more compact written representation. Clustering by 3 bits is called **Octal**, and it is often indicated with a leading zero, **0**. Octal is not that common today.

$$v = \sum_{i=0}^{n-1} 8^i d_i$$

$$\begin{array}{cccccccccccc} 2^{11} & 2^{10} & 2^9 & 2^8 & 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ \hline 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \end{array} = 2000_{10}$$

$$\underbrace{0\,1\,1}_{3}\;\underbrace{1\,1\,1}_{7}\;\underbrace{0\,1\,0}_{2}\;\underbrace{0\,0\,0}_{0}$$

03720

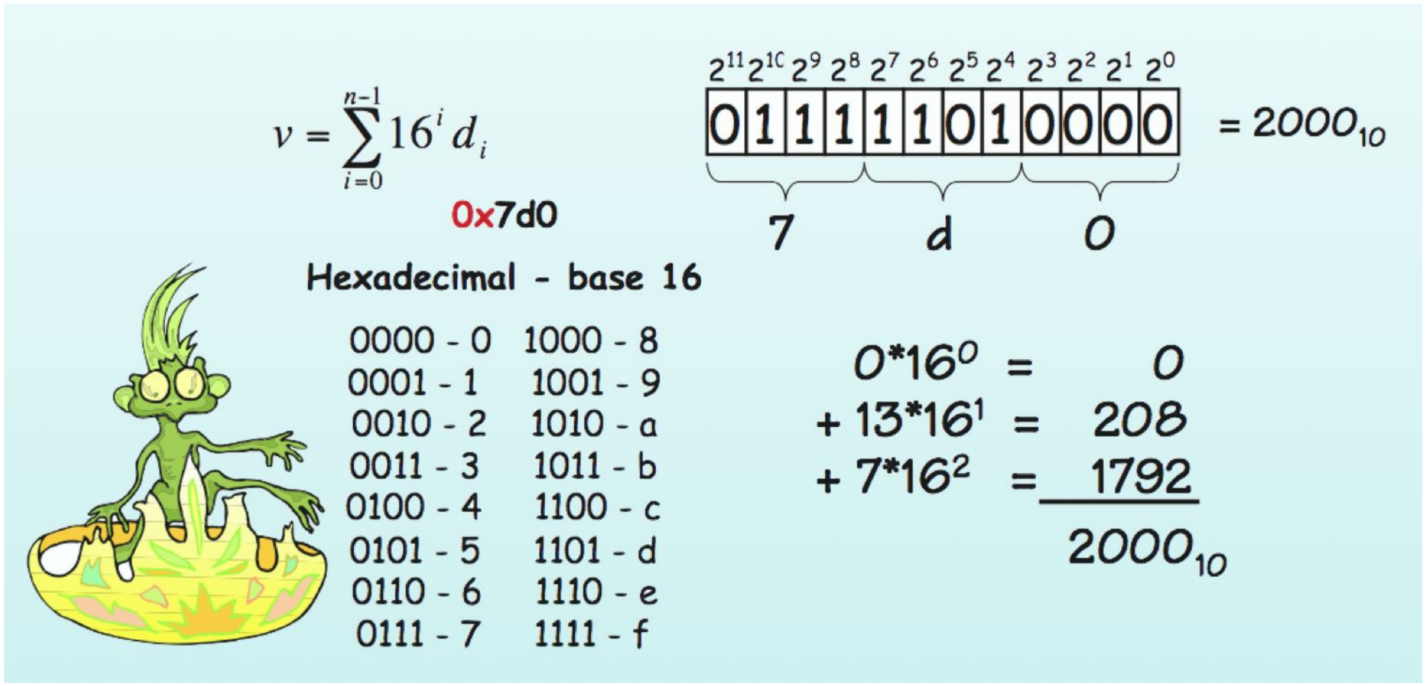
Octal - base 8

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

$$\begin{aligned} 0*8^0 &= 0 \\ +\ 2*8^1 &= 16 \\ +\ 7*8^2 &= 448 \\ +\ 3*8^3 &= \underline{1536} \\ & \quad 2000_{10} \end{aligned}$$

One more Clustering

Clusters of 4 bits are used most frequently. This representation is called hexadecimal. The **hexadecimal** digits include 0-9, and A-F, and each digit position represents a power of 16. Commonly indicated with a leading "**0x**".



$$v = \sum_{i=0}^{n-1} 16^i d_i$$

0x7d0

$$\begin{array}{cccccccccccc} 2^{11} & 2^{10} & 2^9 & 2^8 & 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ \hline 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \end{array} = 2000_{10}$$

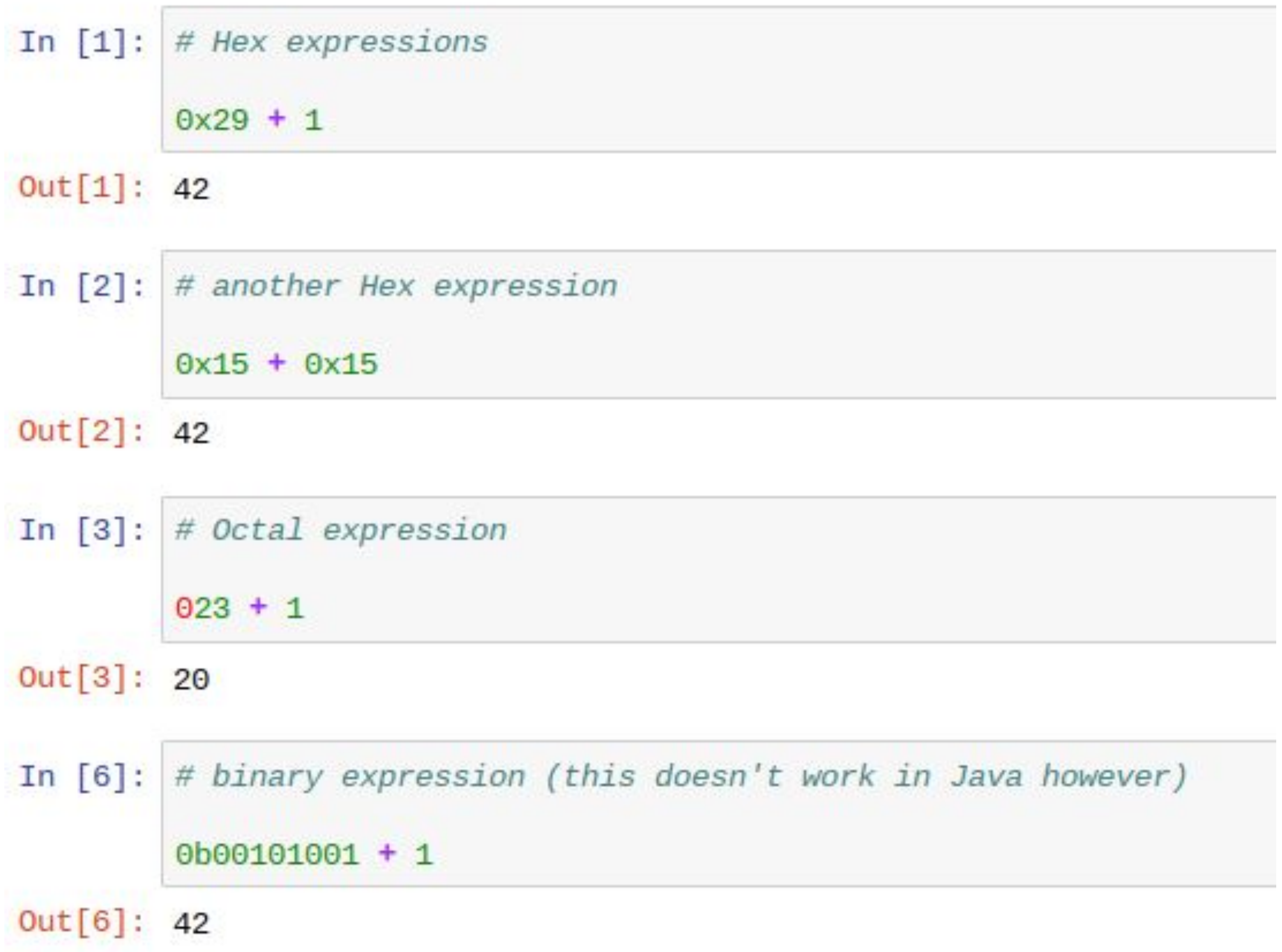
$$\underbrace{0111}_{7}\ \underbrace{1101}_{d}\ \underbrace{0000}_{0}$$

Hexadecimal - base 16

0000 - 0	1000 - 8
0001 - 1	1001 - 9
0010 - 2	1010 - a
0011 - 3	1011 - b
0100 - 4	1100 - c
0101 - 5	1101 - d
0110 - 6	1110 - e
0111 - 7	1111 - f

$$\begin{array}{rcr} 0 \cdot 16^0 & = & 0 \\ + 13 \cdot 16^1 & = & 208 \\ + 7 \cdot 16^2 & = & 1792 \\ \hline & & 2000_{10} \end{array}$$

They've always been there...



```
In [1]: # Hex expressions

        0x29 + 1

Out[1]: 42

In [2]: # another Hex expression

        0x15 + 0x15

Out[2]: 42

In [3]: # Octal expression

        023 + 1

Out[3]: 20

In [6]: # binary expression (this doesn't work in Java however)

        0b00101001 + 1

Out[6]: 42
```

Summary and Next Time

- Information is all about bits, Entropy
- Using bits to encode things
  - Efficient variable-length encodings
  - Redundancy
- Next: more about encoding numbers
  - Signed Numbers (there is a choice)
  - Non-integers (Fractions and Fixed-point)
  - Floating point numbers
- Encoding other things...