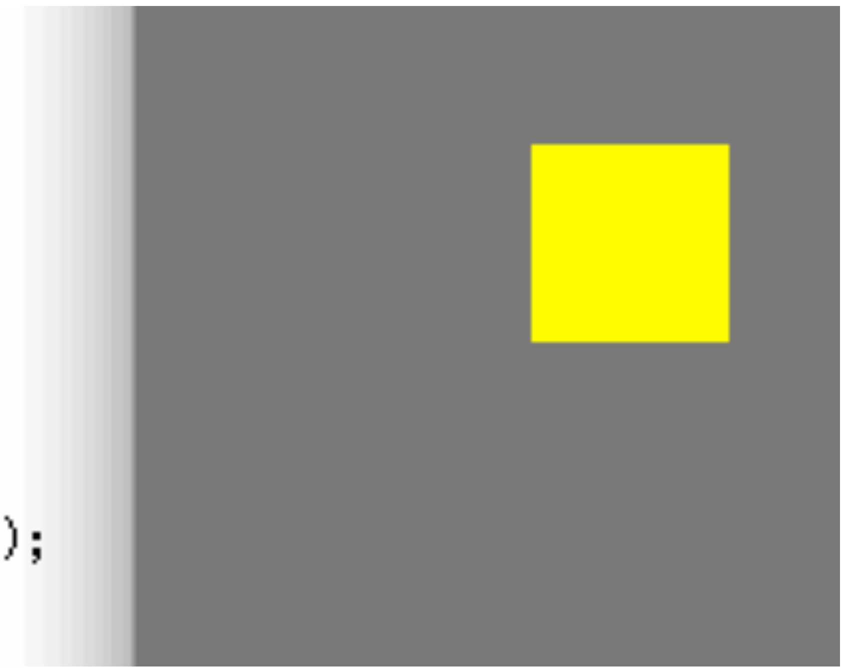
Using Functions

- Functions package up computation … when do we use them? All the time.
- Write some simple code to achieve a goal …

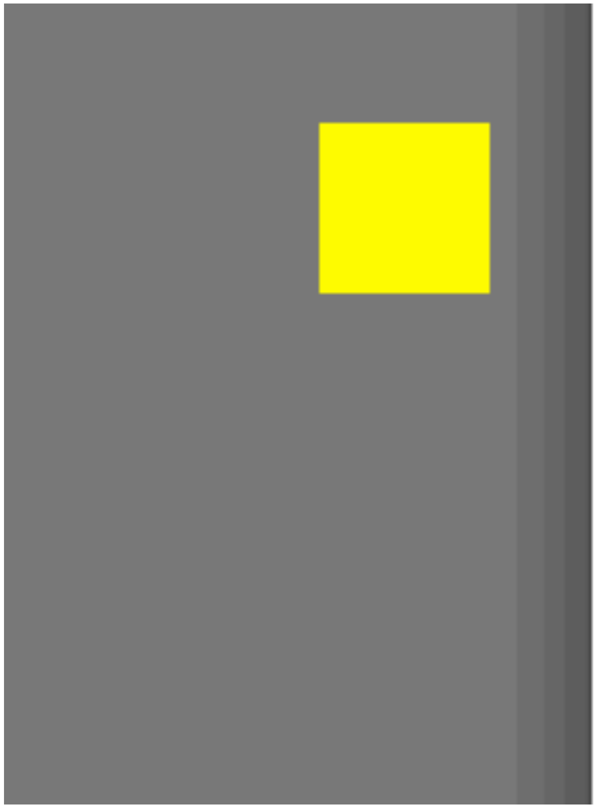
```
void setup( ) {
  size(300, 300);
  background(102);
  noStroke( );
  fill(255,255,0);
}

void draw( ) {
  rect(100, 100, 50, 50);
}
```



Package It … Make Position Vary

- To put the rectangle in different places, we "parameterize" the position, that is, use input to the function to place the rectangle



```
void setup( ) {
  size(300, 300);
  background(102);
  noStroke( );
  fill(255,255,0);
}

void draw( ) {
  rec(100, 100);
}

void rec(float x, float y) {
  rect(x, y, 50, 50);
}
```

Once Created, Use It Everywhere

- Now we quit thinking of drawing a rectangle, but now think of placing a 50x50 rectangle

```
void setup( ) {
  size(300, 300);
  background(102);
  noStroke( );
}

void draw( ) {
  fill(255,255,0);
  for (int i=0; i< 5; i++){
  rec(60*i, 50);
  }
  fill(255,0,0);
  rec(mouseX, mouseY);
}

void rec(float x, float y) {
  rect(x, y, 50, 50);
}
```

More On Parameters …

- Return to last lecture for two slides on the topic of parameters …

Parameters: Customize each function call to a specific situation – they are the input to the function

• *Parameters* are the names of the input values used inside of the procedure body

• *Arguments* are the values from outside to be used for each of the parameters

Arguments Become Parameters

- Notice that if the DEFINITION has *n* parameters, the CALL needs *n* arguments
- The parameters and arguments correspond

```
void draw( ) {
  fill(255);
  hexa(20, 40);
  hexa(50, 40);
  hexa(80, 40);
}
```

Inside of the function, the parameter, e.g. xbase, is declared and initialized to the corresponding argument, e.g. 80. Then, the definition uses it, e.g.

```
rect(80, 40+10, 20, 40)
```

```
void hexa(float xbase, float ybase) {
  rect(xbase, ybase+10, 20, 40);
  triangle(xbase, ybase+10, xbase+20, ybase+10, xbase+10, ybase);
  triangle(xbase, ybase+50, xbase+20, ybase+50, xbase+10, ybase+60);
}
```

Parameters

- Parameters are automatically declared (and initialized) on a call, and remain in existence as long as the function remains unfinished
- When the function ends, the parameters vanish, only to be recreated on the next call
- It is wise to choose parameter names, e.g. x-b-a-s-e that are meaningful to you
  - I chose xbase as the orientation point of the figure in the x direction
  - Notice that I used that name a lot, and the meaning to me remained the same

What Are Your Questions?

- We said (it was the 2nd day of class) that a function definition has 3 parts: name, params, body
  - Name is critical: it names the “concept” created by the function
  - Parameters are critical: they customize a function to many cases
  - Body is critical: it defines how the function works
- Function uses (calls) have 2 parts: name, args
  - Name is critical: says what concept you will use
  - Arguments are critical: says how this case handled

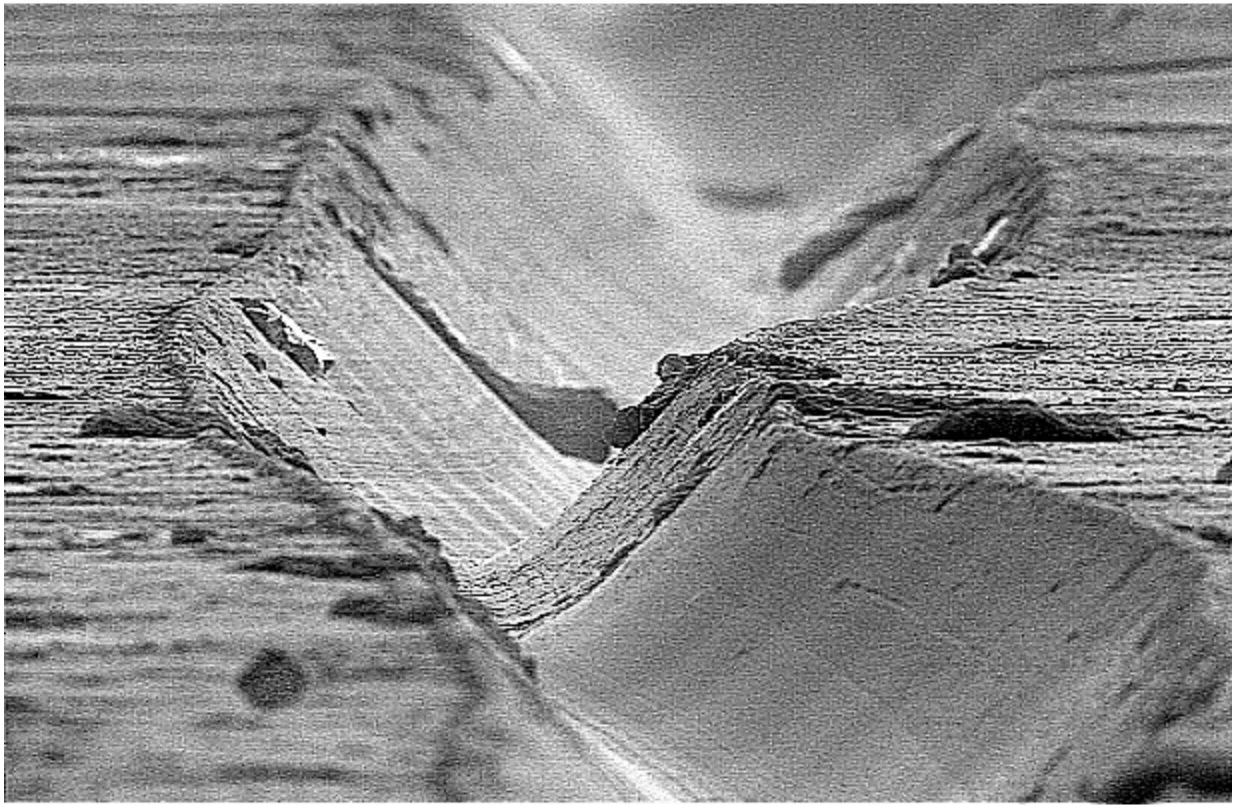
Bits are IT

Fundamental Principle of Information Representation

Lawrence Snyder
University of Washington, Seattle

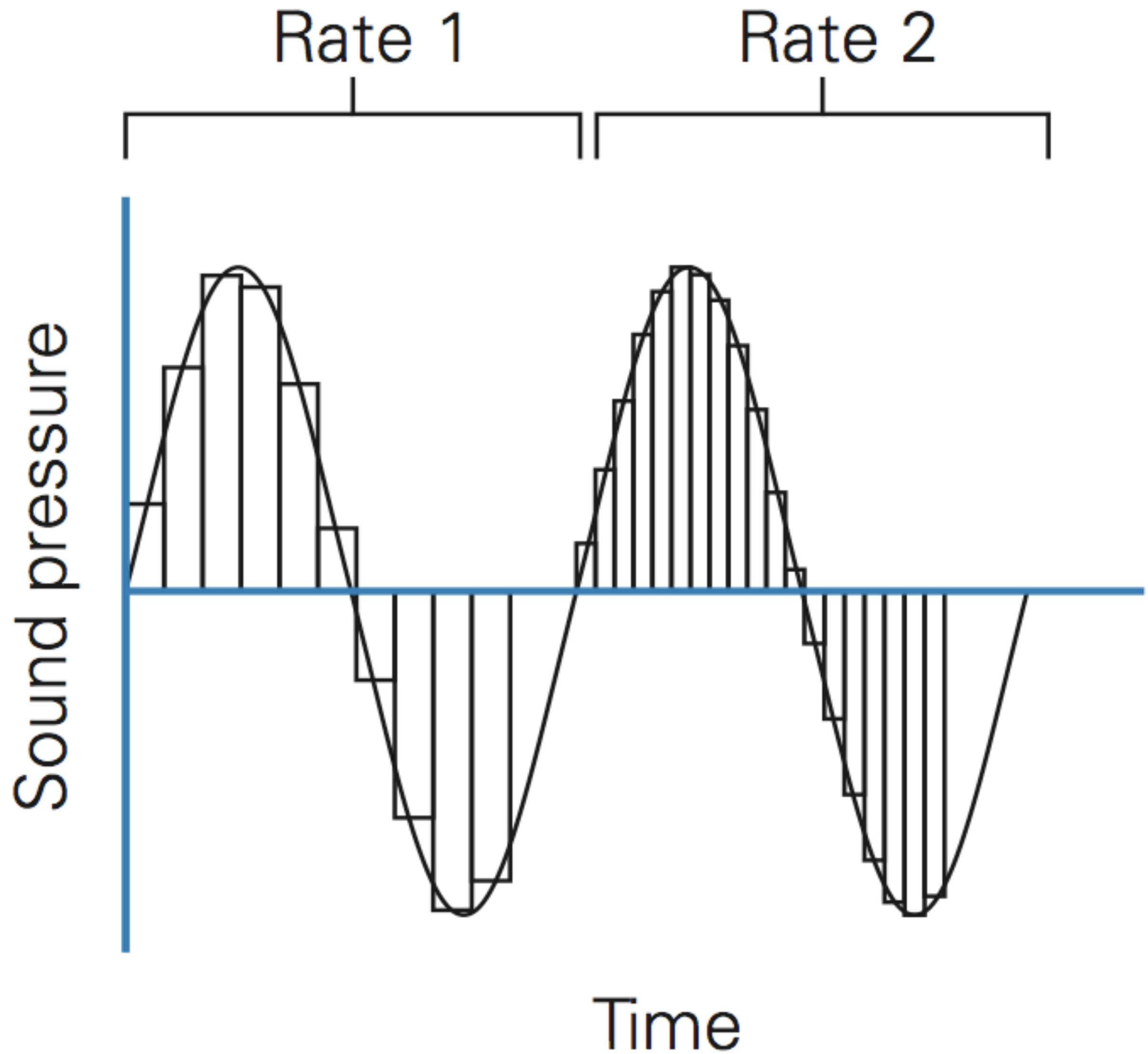
Not All Information Is Discrete

- Analogue information directly applies physical phenomena, e.g. vinyl records

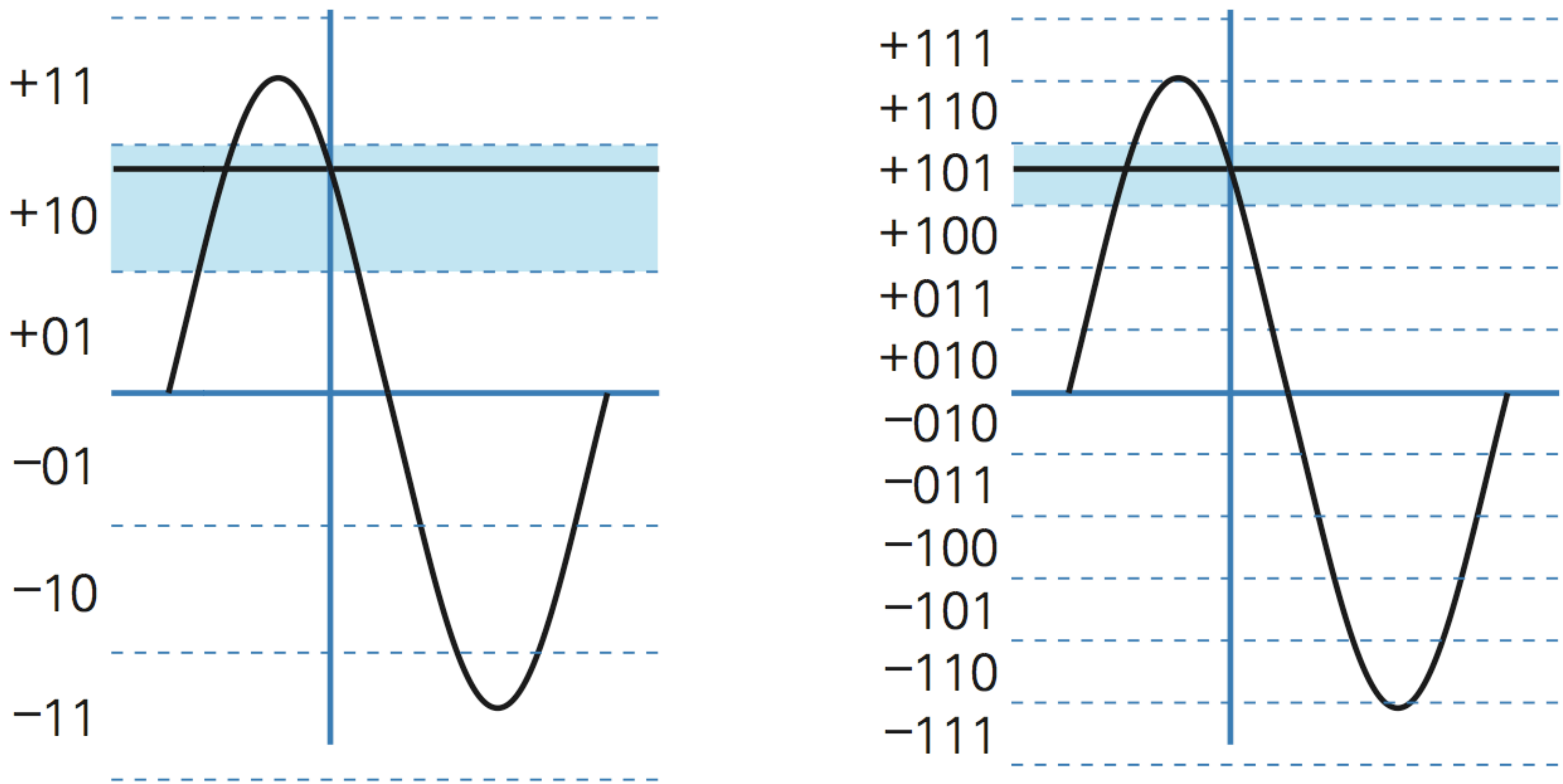


Analog Signals Become Discrete

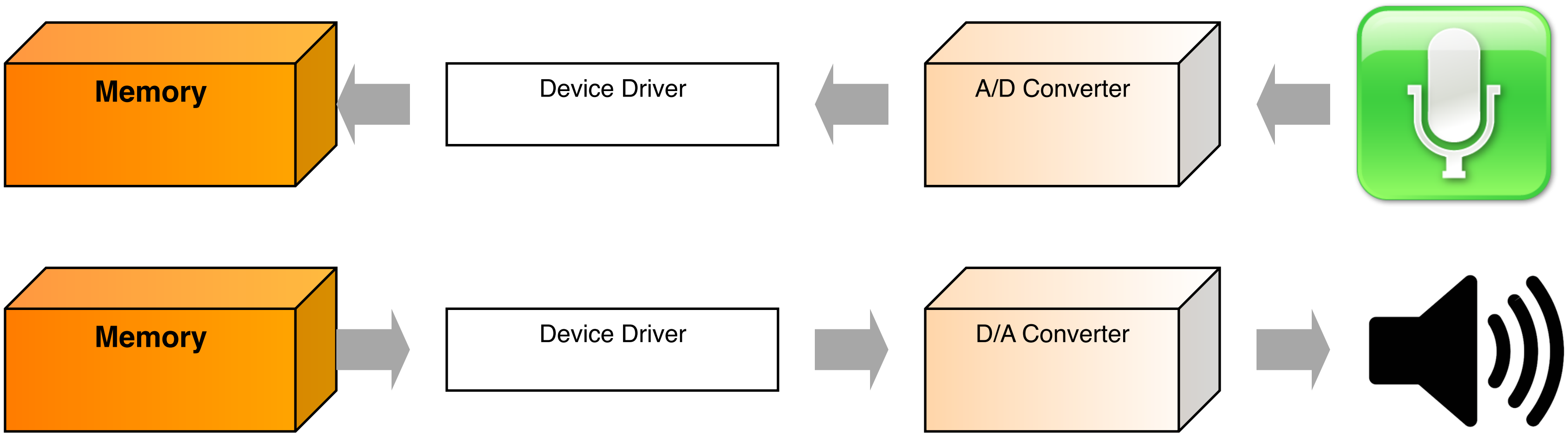
Sampling the wave …



Precision of the Sample



The World Is Analog – Go Between



Analog is needed for the "real world"

Digital is best for "information world"

- Can be modified, enhanced, remixed, etc
- Shared, stored permanently, reproduced, …

Review What We Know About Bits

- Facts about physical representation:
  - Information is represented by the presence or absence of a physical phenomenon (PandA)
    - Hole punched in a card; no hole [Hollerith]
    - Dog barks in the night; no barking in the night [Holmes]
    - Wire is electrically charged; wire is neutral
    - ETC
- Abstract all these cases with 0 and 1; it unifies them so we don't have to consider the details

Bits Work For Arithmetic

- Binary is sufficient for number representation (place/value) and arithmetic
  - The number base is 2, instead of 10
  - Binary addition is just like addition in any other base except it has fewer cases … better for circuits
  - All arithmetic and standard calculations have binary equivalents
- We conclude: bits “work” for quantities

Bytes – 8 bits in a row

- Bytes illustrate that bits can be grouped in sequence to generate unique patterns
  - 2 bits in sequence, $2^2 = 4$ patterns: 00, 01, 10, 11
  - 4 bits in sequence, $2^4 = 8$ patterns: 0000, 0001, …
  - 8 bits in sequence, $2^8=256$ patterns: 0000 0000, …
- ASCII groups 8 bits in sequence
  - They seem to be assigned intelligently, but they're just patterns

ASCII	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	NU	SH	SX	EX	ET	EQ	AK	BL	BS	HT	LF	YT	FF	CR	S0	SI
0001	DL	D1	D2	D3	D4	NK	SY	EΣ	CN	EM	SB	EC	FS	GS	RS	US
0010		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0011	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0100	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0101	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
0110	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0111	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DT
1000	80	81	82	83	IN	NL	SS	ES	HS	HJ	YS	PD	PV	RI	S2	S3
1001	DC	P1	PZ	SE	CC	MM	SP	EP	Q8	QQ	QA	CS	ST	OS	PM	AP
1010	AO	¡	¢	£	♀	¥	¦	§	¨	©	♂	«	¬	-	®	¯
1011	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
1100	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
1101	Đ	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	β
1110	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
1111	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	Þ	ÿ

Representing Anything

- Compare binary arithmetic to ASCII
  - Binary encodes the positions to make using the information (numbers) easy, like for addition
  - ASCII assigns some pattern to each letter
- Given any finite set of things – colors, computer addresses, English words, etc.
  - We might figure out a smart way to represent them as bits – colors can give light intensity of RGB
  - We can just assign patterns, and manipulate them by pattern matching – red can be 0000 0001, dark red 0000 0010, etc.

Bits Have No Inherent Meaning

- What does this represent:

  0000 0000 1111 0001 0000 1000 0010 0000?
- You don't know until you how it was encoded
  - As a binary number: 15,796,256
  - As a color, RGB(241,8,32)
  - As a computer instruction: Add 1, 7, 17
  - As ASCII: $^{n}{}_{u}$ $^{b}{}_{s}$ ñ <space>
  - IP Address: 0.241.8.32
  - Hexadecimal number: 00 F1 08 20
  - … → to infinity and beyond

A Bias-free Universal Medium

- This is the principle:

Bias-free Universal Medium Principle:
Bits can represent all discrete information;
bits have no inherent meaning

Summary

- Analog information must be made discrete (digitized) before it can be processed by computers … this is done by A/D converter
- The reverse process lets us hear it: D/A
- Bits are sufficient to encode all discrete information
- Bits have no inherent meaning, so they can be used for anything