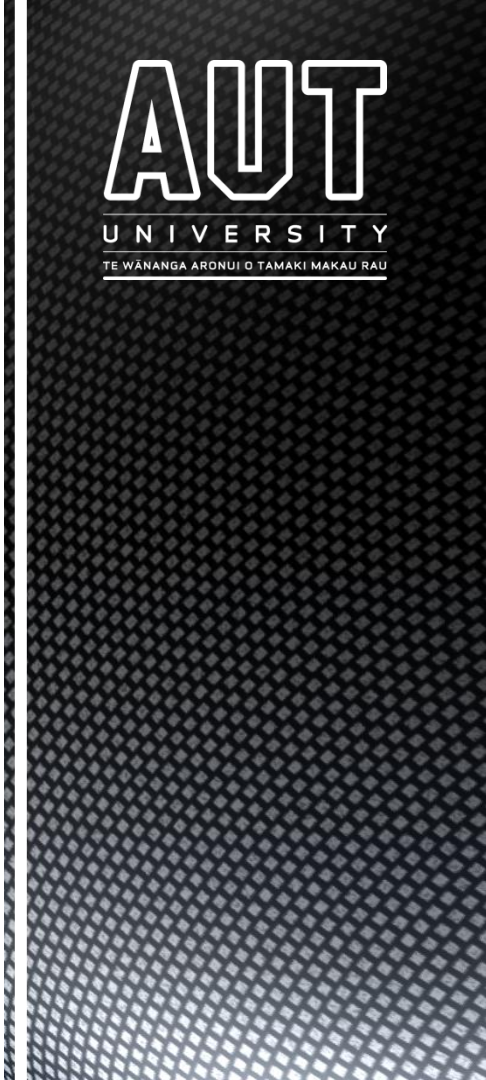
175003 Programming for Creativity

Lecture 02

Bits (& Bobs):
Digitised Media & Computer Architecture



Assignment 1

- Formative
- Deadline: Thursday, 20th of March, 8pm
- Description and Submission: AUTonline

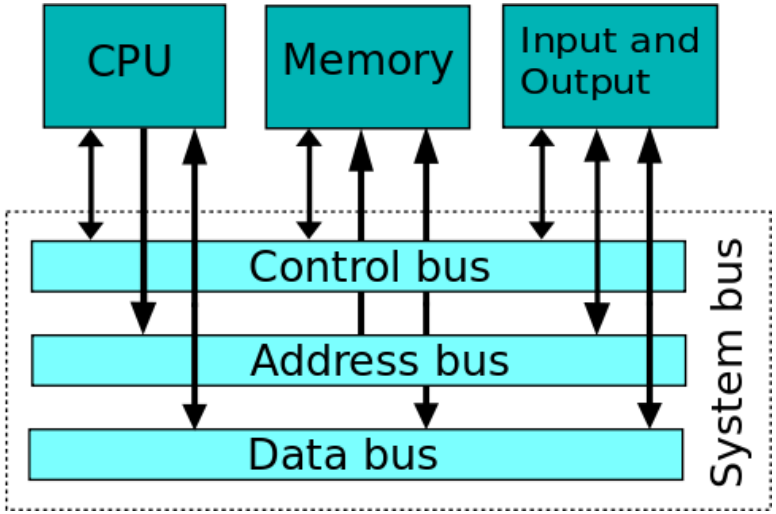
Reacp Lecture 1

- Computers are fast but stupid
- They need detailed instructions: Algorithms
- Algorithms must be
  - Complete
  - Precise

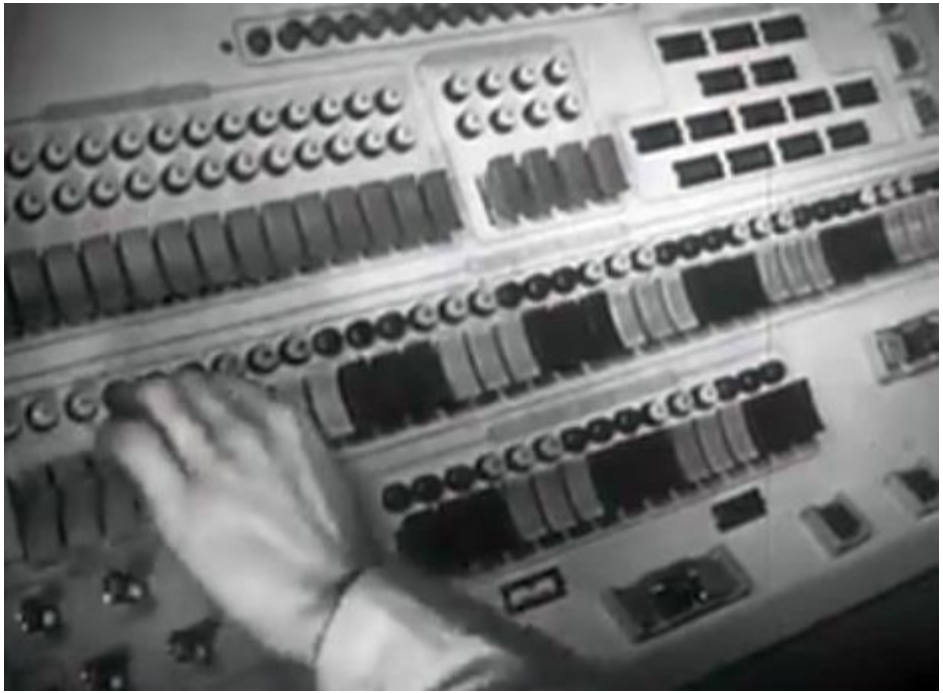
Overview

- This session will cover:
  - Basics of computer architecture
  - Binary number systems
  - Encodings
  - Digitised media
  - Drawing primitives

Computer Architecture



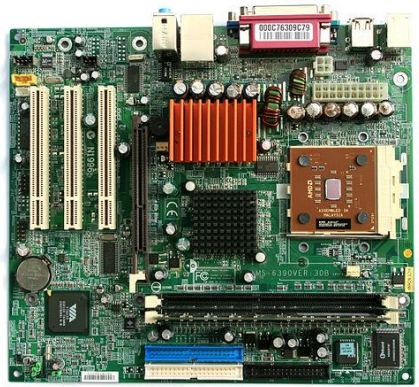
Nothing Much has Changed

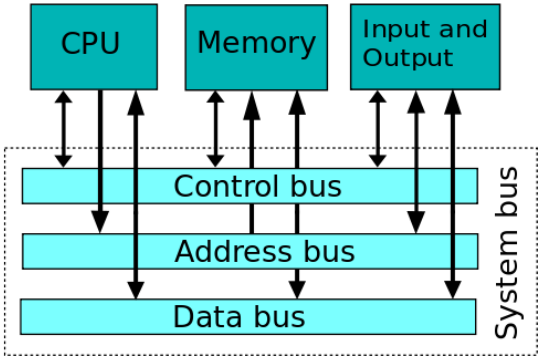


http://www.youtube.com/watch?v=KaFHrGjy7wo

Computer Architecture

- Main components in a typical computer system:
  - Processor or CPU (Central Processing Unit)
    - Maths, comparisons, etc.
  - Memory
    - Storing data (volatile, fast)
    - Each bit of data has its own address
  - Input/output devices
    - Keyboard, Screen
    - Harddrive, CD-ROM
    - Includes connections, e.g., USB
  - Buses
    - Data transfer and control
- "Von Neumann"-Architecture





What is a “Byte”?

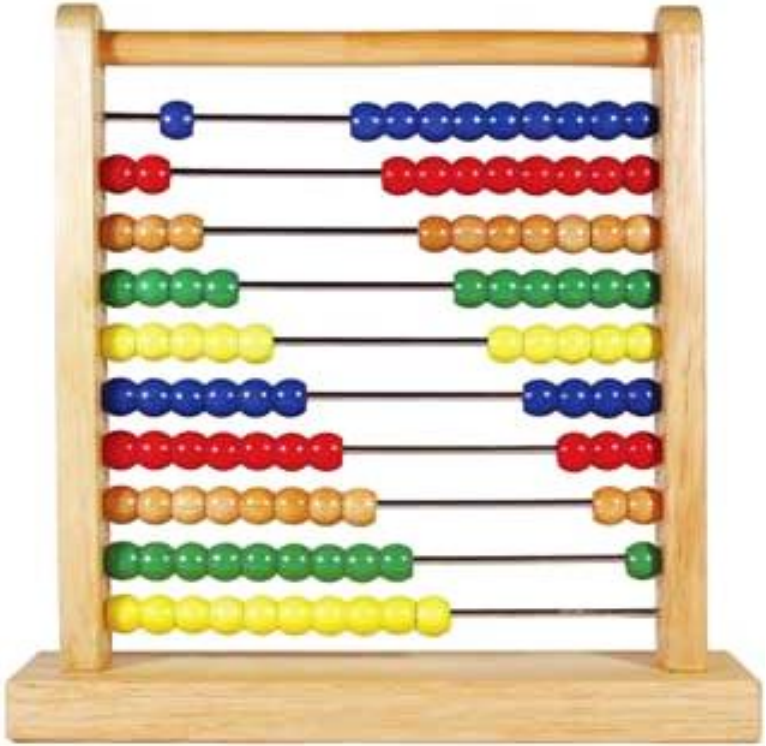
- What we are really talking about in this lecture is *abstraction*
- Abstraction is the simplification of reality
- In computer systems we see layered abstraction, each layer bridging the gap between Machine and Human a bit more
- To understand how our ideas/programs are turned into “electric” reality, we need to understand (a little bit) how a computer works

Binary Number Systems

- Everyone talks about bits and bytes, but without really understanding why binary numbers are important
- Example: “64 bit architecture”:
  What that really means is that there are 64 wires that connect the CPU to the memory
- Each wire carries an electrical signal that is nominally either 0 volts or >1.35 volts
  - Binary numbers are an abstraction of that voltage, we don’t care if it is actually +1.33 volts or +1.36 volts

Binary Number Systems

- We are all used to decimal numbering, so much so that we probably forget what the numbers actually mean
- If we sort that out, binary numbers are actually pretty straightforward!



Binary Number Systems

- Each “digit” in a decimal number means something
- Example: 264
  264 = 2 hundred + 6 tens + 4 ones

			...	1000	100	10	1
					2	6	4

264 = 2×100 and 6×10 and 4×1

- This is called a “base 10” number system,
  or “decimal”, derived from the Latin word for “ten”

Binary Number Systems

- Binary numbers work just in the same way, but we have to work in scales of *two*, not ten
- This is called “base 2”, or “binary” (derived from the Latin word “bi” for “two”) because our number multiplier is 2

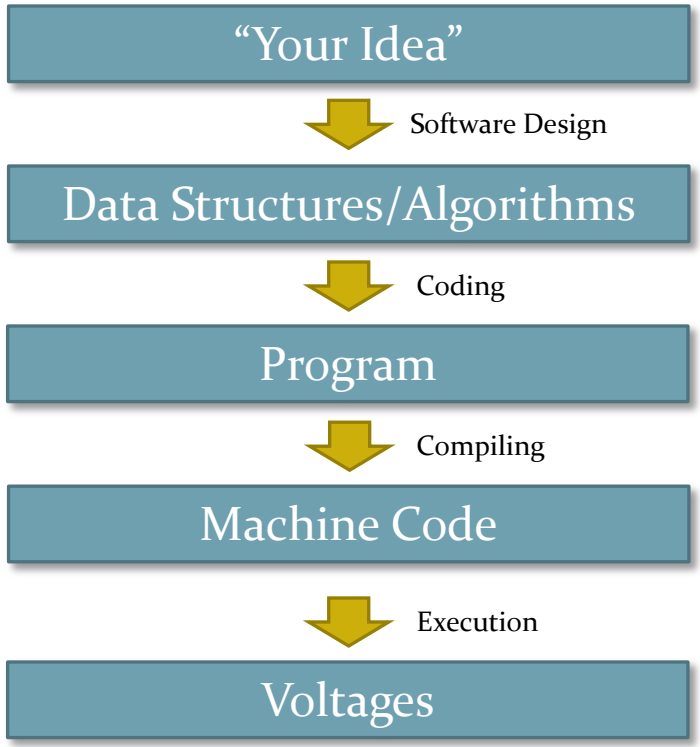
128	64	32	16	8	4	2	1
1	0	0	1	1	0	0	1

10011001 = 1×128 and 1×16 and 1×8 and 1×1

= 153 (in decimal)

So What?

- The program you write ultimately has to be turned into voltages that are carried along cables
- Thankfully, all this hard work has been done by other people
- But: Understanding how the analogue world becomes encoded digitally is very important



Let's Create a Digital "Me"

- Task:
  "Store" your age, height, and name as 0/1 in the computer's memory

Address	Memory								Decimal	ASCII
	MSB							LSB		
	128	64	32	16	8	4	2	1		
0										
1										
2										
3										
4										
5										
6										
7										

Age: _____

Height: _____

Name: _____

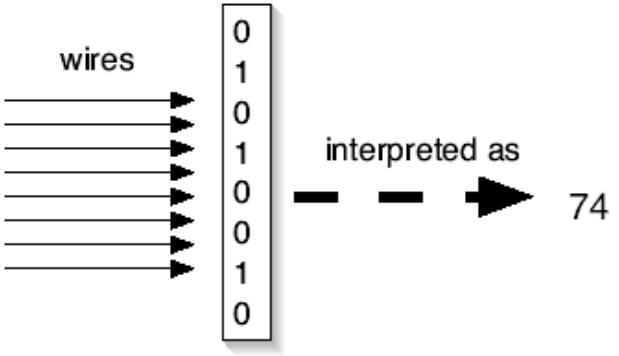
ENCODINGS

Regular ASCII Chart (character codes 0 - 127)

000		(nul)	016	▶	(dle)	032	sp	048	0	064	@	080	P	096	`	112	p
001	☺	(soh)	017	◀	(dc1)	033	!	049	1	065	A	081	Q	097	a	113	q
002	☻	(stx)	018	↕	(dc2)	034	"	050	2	066	B	082	R	098	b	114	r
003	♥	(etx)	019	‼	(dc3)	035	#	051	3	067	C	083	S	099	c	115	s
004	♦	(eot)	020	¶	(dc4)	036	$	052	4	068	D	084	T	100	d	116	t
005	♣	(enq)	021	§	(nak)	037	%	053	5	069	E	085	U	101	e	117	u
006	♠	(ack)	022	▬	(syn)	038	&	054	6	070	F	086	V	102	f	118	v
007	•	(bel)	023	↨	(etb)	039	'	055	7	071	G	087	W	103	g	119	w
008	◘	(bs)	024	↑	(can)	040	(	056	8	072	H	088	X	104	h	120	x
009	○	(tab)	025	↓	(em)	041	)	057	9	073	I	089	Y	105	i	121	y
010		(lf)	026	→	(eof)	042	*	058	:	074	J	090	Z	106	j	122	z
011	♂	(vt)	027	←	(esc)	043	+	059	;	075	K	091	[	107	k	123	{
012	♀	(np)	028	∟	(fs)	044	,	060	<	076	L	092	\	108	l	124	¦
013		(cr)	029	↔	(gs)	045	-	061	=	077	M	093	]	109	m	125	}
014	♫	(so)	030	▲	(rs)	046	.	062	>	078	N	094	^	110	n	126	~
015	☼	(si)	031	▼	(us)	047	/	063	?	079	O	095	_	111	o	127	⌂

Key Concept: Encodings

- We can interpret the 0's and 1's in computer memory any way we want
  - We can treat them as numbers.
  - We can encode information in those numbers
- Even the notion that the computer understands numbers is an interpretation
  - We encode the voltages on wires as 0's and 1's, eight of these defining a byte
  - Which we can, in turn, interpret as a decimal number



Layered Encodings

- One encoding, ASCII, defines an “A” as 65
  - If there’s a byte with a 65 in it, and we decide that it’s a character, it “turns” into an “A”!
  - (Note: not an “a”!)

Regular ASCII Chart (character codes 0 - 127)

000	(nul)	016 ▶	(dle)	032	sp	048	0	064	@	080	P	096	`	112	p
001 ☺	(soh)	017 ◀	(dc1)	033	!	049	1	065	A	081	Q	097	a	113	q
002 ☻	(stx)	018 ↕	(dc2)	034	"	050	2	066	B	082	R	098	b	114	r
003 ♥	(etx)	019 ‼	(dc3)	035	#	051	3	067	C	083	S	099	c	115	s
004 ♦	(eot)	020 ¶	(dc4)	036	$	052	4	068	D	084	T	100	d	116	t
005 ♣	(enq)	021 §	(nak)	037	%	053	5	069	E	085	U	101	e	117	u
006 ♠	(ack)	022 ▬	(syn)	038	&	054	6	070	F	086	V	102	f	118	v
007 •	(bel)	023 ↨	(etb)	039	'	055	7	071	G	087	W	103	g	119	w
008 ◘	(bs)	024 ↑	(can)	040	(	056	8	072	H	088	X	104	h	120	x
009 ○	(tab)	025 ↓	(em)	041	)	057	9	073	I	089	Y	105	i	121	y
010	(lf)	026 →	(eof)	042	*	058	:	074	J	090	Z	106	j	122	z
011 ♂	(vt)	027 ←	(esc)	043	+	059	;	075	K	091	[	107	k	123	{
012 ♀	(np)	028 ∟	(fs)	044	,	060	<	076	L	092	\	108	l	124	¦
013	(cr)	029 ↔	(gs)	045	-	061	=	077	M	093	]	109	m	125	}
014 ♫	(so)	030 ▲	(rs)	046	.	062	>	078	N	094	^	110	n	126	~
015 ☼	(si)	031 ▼	(us)	047	/	063	?	079	O	095	_	111	o	127	⌂

Layered Encodings

- We can string together lots of these numbers together to make usable text
  - "83, 116, 101, 102, 97, 110" is "Stefan"
  - "60, 97, 32, 104, 114, 101, 102, 61" is "<a href="

Regular ASCII Chart (character codes 0 - 127)

000	(nul)	016 ▶	(dle)	032	sp	048	0	064	@	080	P	096	`	112	p
001 ☺	(soh)	017 ◀	(dc1)	033	!	049	1	065	A	081	Q	097	a	113	q
002 ☻	(stx)	018 ↕	(dc2)	034	"	050	2	066	B	082	R	098	b	114	r
003 ♥	(etx)	019 ‼	(dc3)	035	#	051	3	067	C	083	S	099	c	115	s
004 ♦	(eot)	020 ¶	(dc4)	036	$	052	4	068	D	084	T	100	d	116	t
005 ♣	(enq)	021 §	(nak)	037	%	053	5	069	E	085	U	101	e	117	u
006 ♠	(ack)	022 ▬	(syn)	038	&	054	6	070	F	086	V	102	f	118	v
007 •	(bel)	023 ↨	(etb)	039	'	055	7	071	G	087	W	103	g	119	w
008 ◘	(bs)	024 ↑	(can)	040	(	056	8	072	H	088	X	104	h	120	x
009 ○	(tab)	025 ↓	(em)	041	)	057	9	073	I	089	Y	105	i	121	y
010	(lf)	026 →	(eof)	042	*	058	:	074	J	090	Z	106	j	122	z
011 ♂	(vt)	027 ←	(esc)	043	+	059	;	075	K	091	[	107	k	123	{
012 ♀	(np)	028 ∟	(fs)	044	,	060	<	076	L	092	\	108	l	124	¦
013	(cr)	029 ↔	(gs)	045	-	061	=	077	M	093	]	109	m	125	}
014 ♫	(so)	030 ▲	(rs)	046	.	062	>	078	N	094	^	110	n	126	~
015 ☼	(si)	031 ▼	(us)	047	/	063	?	079	O	095	_	111	o	127	⌂

Layered Encodings

- A number is just a number is just a number
- If you have to treat it as a letter, there's a piece of software that associates the number 65 with the graphical representation for "A"
- If you have to treat it as part of an HTML document, there's a piece of software that understands that "<a href=" is the beginning of a link

Multimedia is Unimedia

- But that same byte with a 65 in it might be interpreted as…
  - A very small piece of sound (e.g., 1/44100th of a second)
  - The amount of redness in a single dot in a larger picture
  - The amount of redness in a single dot in a larger picture which is a single frame in a full-length motion picture
  - The amount of light falling onto a sensor
  - etc.

" 'Multimedia' is the wrong word. We have created a Unimedia, really. Bits are bits."

-- Nicholas Negroponte, founder of the MIT Media Lab, quoted in Newsweek Magazine (1993)

The Illusionist

"Your computer successfully creates the illusion
that it contains photographs, letters, songs and movies.
All it *really* contains is *bits*, lots of them,
patterned in ways you can't see.
Your computer was designed to store just bits –
all the files and folders and different kinds of data
are illusions created by computer programmers....
We couldn't live without those more intuitive concepts,
but they are artifices. Underneath, it's all just bits."

-

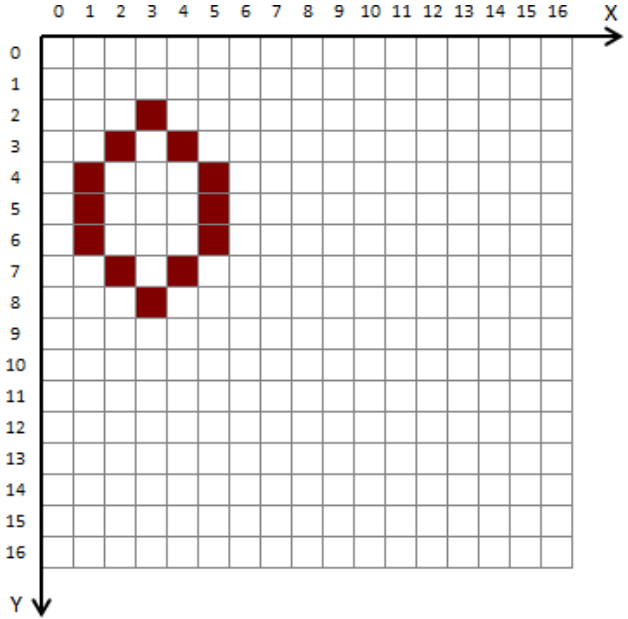
-- from "Blown to Bits" (2008), by Hal Abelson, Ken Ledeen and Harry Lewis

What Computers Understand

- Again: Computers are exceedingly stupid
  - The only data they understand is 0's and 1's
  - The only thing they can do with those 0's and 1's are simple:
    - Move this value here
    - Add, multiply, subtract, divide these values
    - Compare these values, and if one is less than the other, go follow this step rather than that one

- But: Done fast enough, those simple things can be amazing

Drawing Primitives

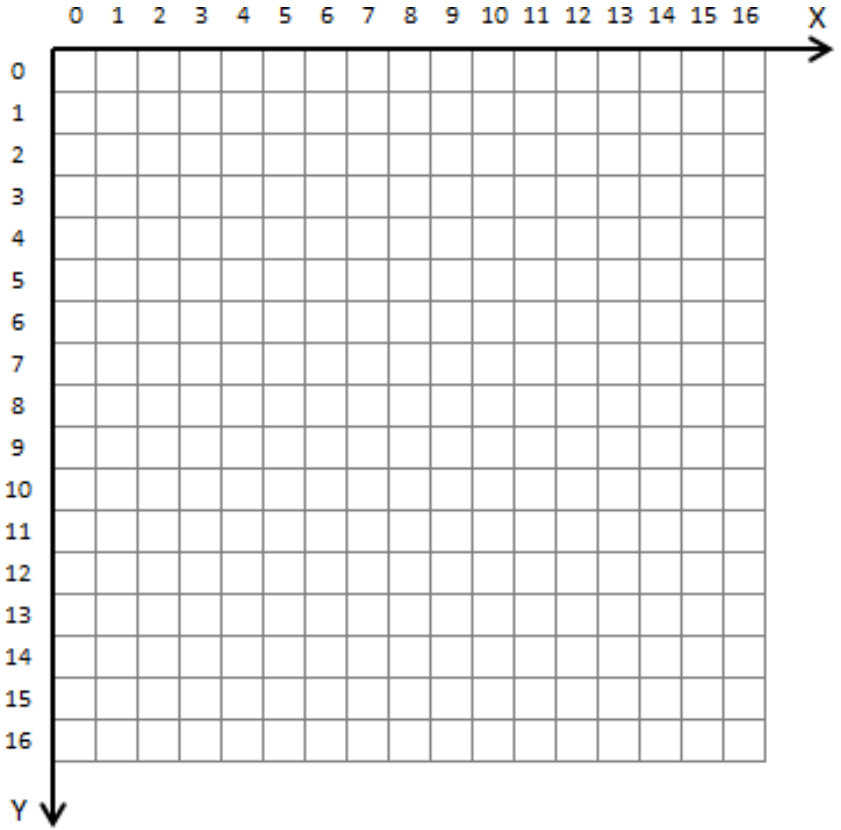


Implications for Programming

- Whilst most of the levels of abstraction and encoding are dealt with outside of our programs, we still need to understand how certain objects are defined (on screen) and manipulated
- We touched on this last week whilst writing code, but we can now formalise our knowledge of the “screen” as a digitised entity

Screen Coordinates

- The screen is like a piece of graph paper
- Each cell is a pixel ("picture element")
- The origin (0, 0) is at the top left
- X-axis: + is to the right,
  - is to the left
  Y-axis: + is down,
  - is up



Canvas Size

- By default, Processing only gives you a small amount of drawing space (the “canvas”)
- The size() command tells Processing how much drawing space you want to use

```
size(width_in_pixels , height_in_pixels);
```

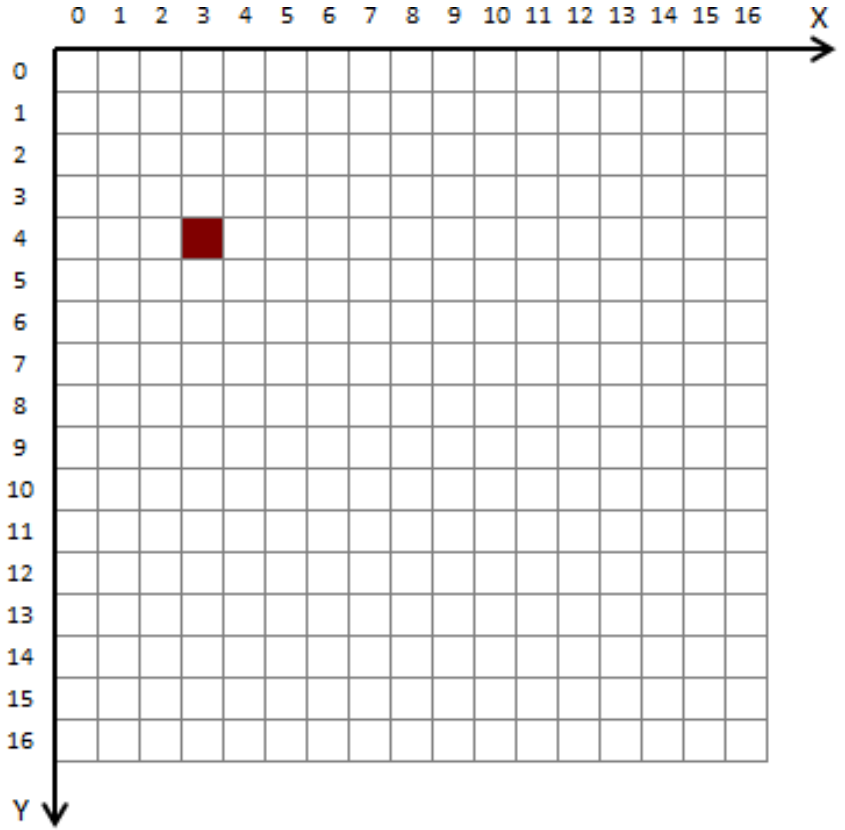
Primitive Shapes

- A “primitive” is a building block
- Primitives can be combined in different ways to produce complex elements
- The main primitives in Processing are:
  - Point
  - Line
  - Rectangle
  - Ellipse

Points

- Use the **`point()`** command to draw a single pixel
- **`point()`** takes two integers as input values:
  - The x and y coordinates of the pixel
- **`point()`** draws a point at the indicated location in the current drawing colour
- Use a semicolon to end the command

```
point(3, 4);
```



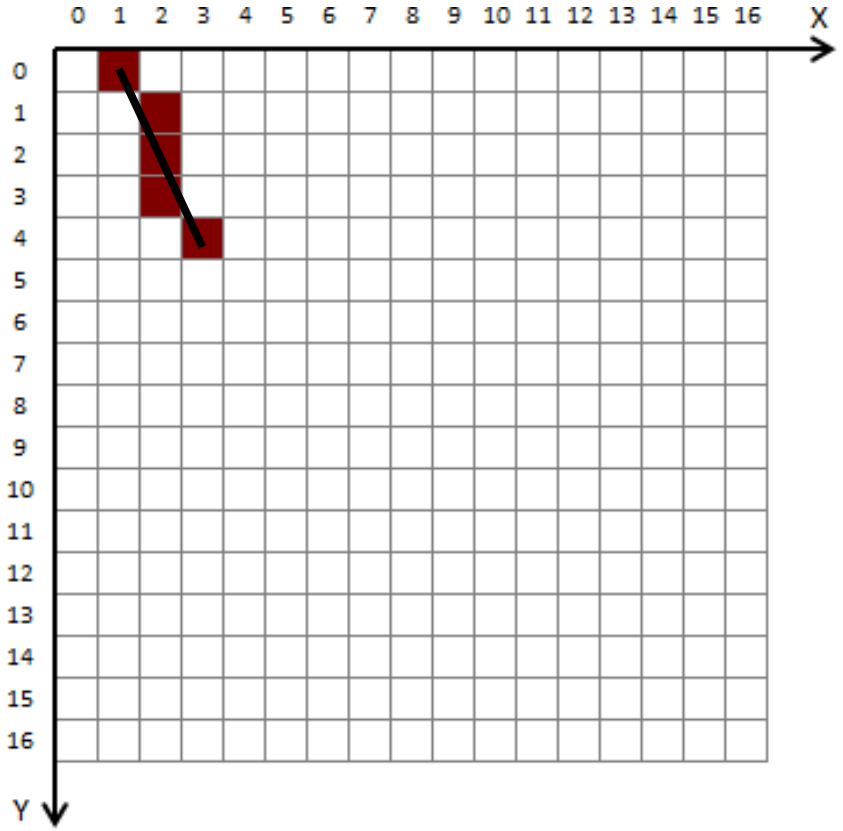
Drawing Lines

- To draw a line, use the **`line()`** command
- Tell **`line()`** where the line should begin and end
- Format:

```
line(start_x, start_y,
     end_x,   end_y);
```

- Example

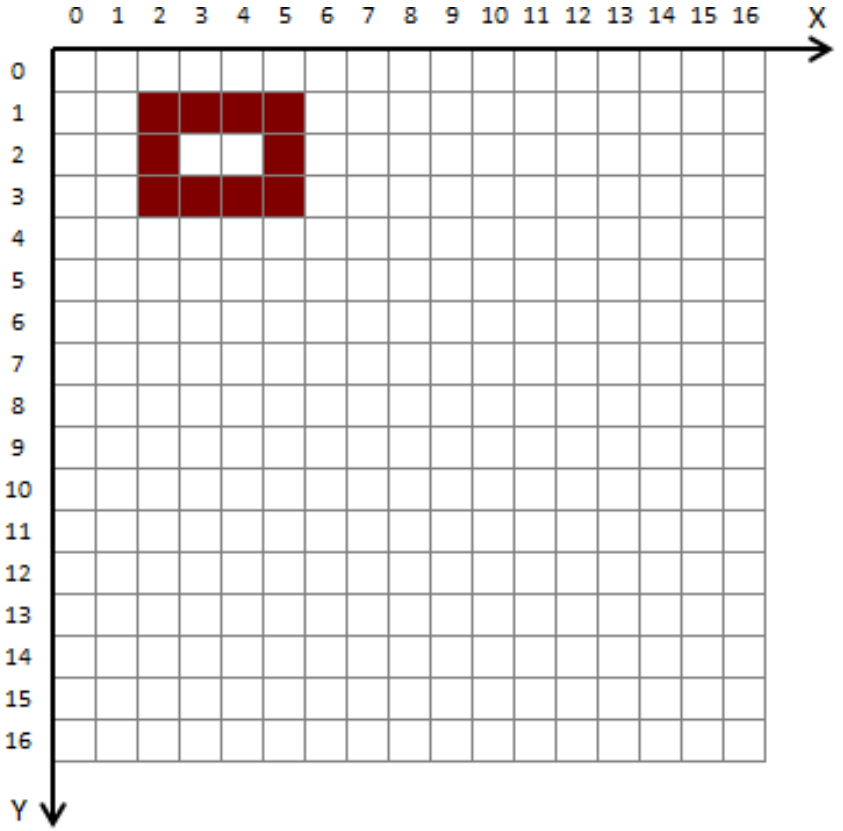
```
line(1, 0, 3, 4);
```



Rectangles

- To draw a rectangle, you need to know:
  - The coordinates of its top-left corner
  - Its width (in pixels)
  - Its height (in pixels)
- Example

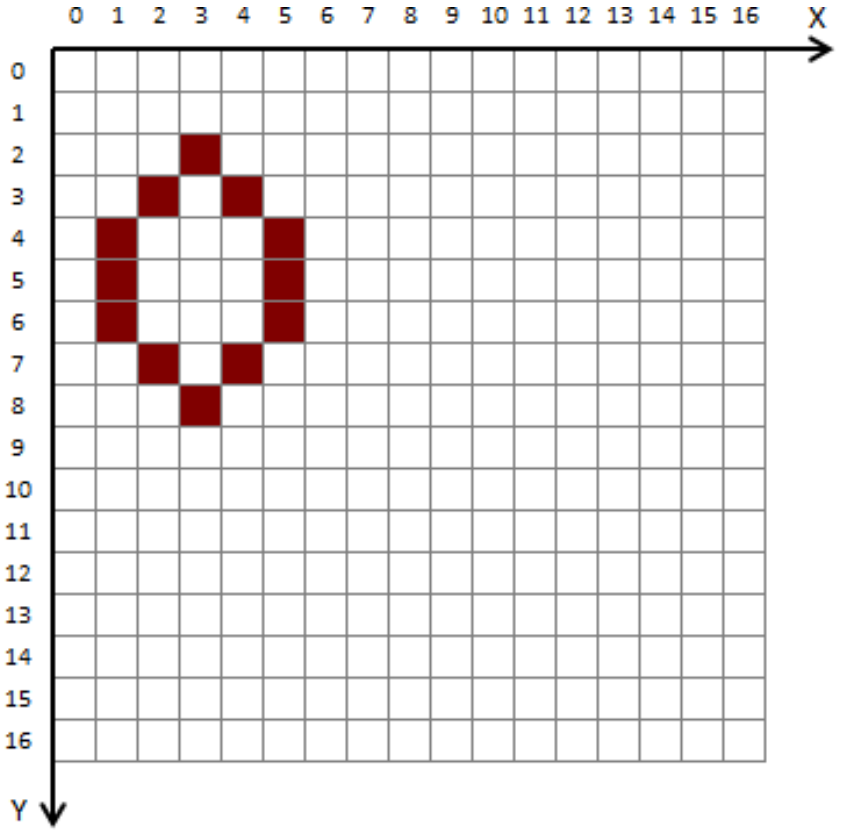
```
rect(2, 1, 4, 3);
```



Ellipses

- Ellipses are drawn in the centre of a bounding box (a rectangle that encloses all of the ellipse's points)
- To draw an ellipse, you need to know:
  - Its centre coordinates
  - Its width in pixels
  - Its height in pixels
- Example

```
ellipse(3, 5, 5, 7);
```



Colours

- With the `stroke(…)` command,
  you can change the outline drawing colour.
- With the `fill(…)` command,
  you can change the fill colour.
- With `noFill()` / `noStroke()`,
  you can turn these features off
- The colour setting is valid from that point on
  until the program terminates,
  or you set a new colour

Rectangle Modes

- You can change what the parameters actually mean
- **`rectMode(mode);`**
  - **`CORNER`** (default)
    - x_left, y_top, width, height
  - **`CORNERS`**
    - x_left, y_left, x_right, y_bottom
  - **`CENTER`**
    - x_centre, y_centre, width, height
  - **`RADIUS`**
    - x_centre, y_centre, width/2, height/2

Ellipse Modes

- You can change what the parameters actually mean
- **`ellipseMode(`*`mode`*`);`**
  - **`CORNER`**
    - x_left, y_top, width, height
  - **`CORNERS`**
    - x_left, y_left, x_right, y_bottom
  - **`CENTER`** (default)
    - x_centre, y_centre, width, height
  - **`RADIUS`**
    - x_centre, y_centre, width/2, height/2

Processing Reference

- "What was the syntax of that command again…?"
- The reference section of the Processing website is your friend!
- http://processing.org/reference/
- It tells you all of the available functions that Processing provides
- Shortcut: Ctrl+Shift+F

Time to Program

You won't learn to program
just by listening to me talking about concepts....

... so let's do some programming!