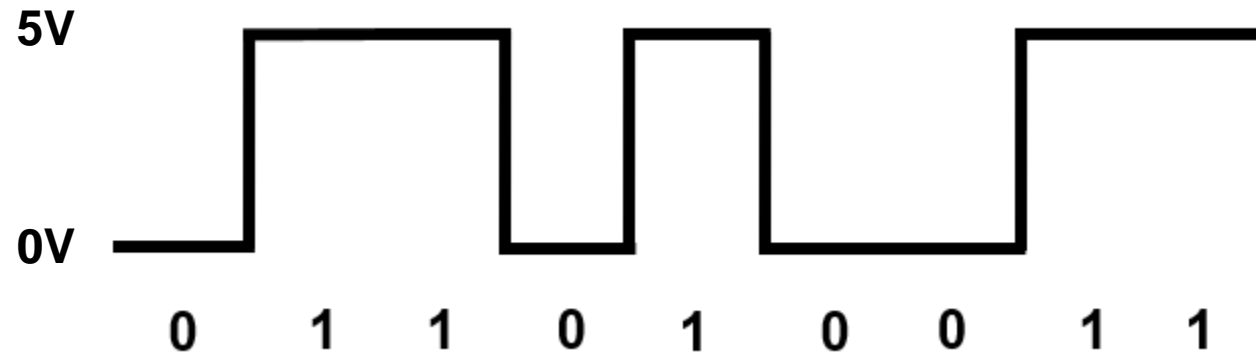


Data Representation

Binary

Computers are digital devices that use electricity to store and send data. This data is represented in binary because electricity can only have two states: **on** or **off**.

Here is an example of a digital signal:



Computers Using Binary

Appliances that you use everyday use 1s and 0s to represent on and off

In computers:

- On is represented by a 1
- Off is represented by a 0



Each one or zero is called a “bit”. The more bits used by the Computer, the more information can be represented.

Benefits of Binary

Simplicity: computers are made up of hundreds of thousands of transistors - in fact the most recent iPhones have over 3 billion transistors in them. Transistors are simple to make but can only be turned on or off to allow electricity to pass or not.

Storage: data can be stored in lots of different ways - magnetic north/south charges on a magnetic disk or reflective/non-reflective areas of a DVD-ROM. Programs used to be written by punching holes in card, and 1s and 0s were represented by hole/no hole.

Signal degradation: electricity can drop in voltage in circuits due to resistance. Binary is resistant to this as a small drop - a full 5v or a partial 3v voltage still represents a 1 in binary.

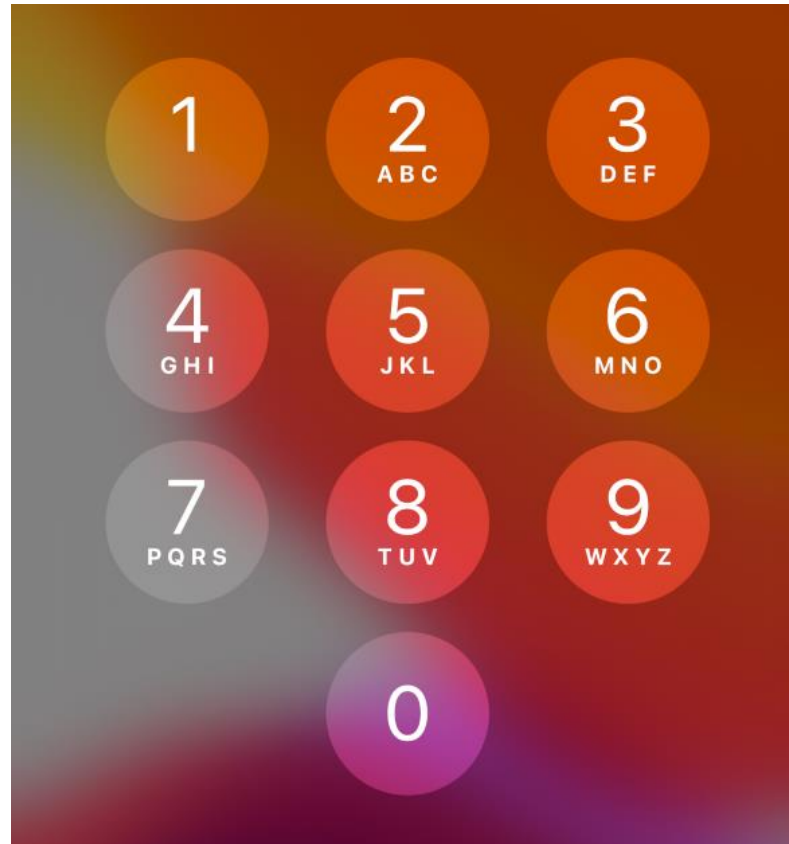
Representing Data in Binary

Watch [this video](#) to understand how data is represented in binary.



Positive Integers

We use the denary system in everyday life to count. Denary means that we use ten different symbols.




Positive Integers

The position of these symbols also tells us what value that symbol represents:

1000s	100s	10s	1s
1	4	8	7
1 x 1000	4 x 100	8 x 10	7 x 1
1000 + 400 + 80 + 7 = 1487			

Positive Integers

Each position towards the left represents a value that is ten times bigger than the one before.




1000s	100s	10s	1s
1	4	8	7
1 x 1000	4 x 100	8 x 10	7 x 1
1000 + 400 + 80 + 7 = 1487			

Representing Positive Integers

We can count in the same way in binary, but we only have two symbols that we can use to represent data: 0 and 1.

The position of these symbols still tells us what value it represents, but in binary the values increase by a factor of two as we move left.



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

Representing Positive Integers

Using the binary number system, we can represent positive integers.

For example, representing the number 215 in binary:

128s	64s	32s	16s	8s	4s	2s	1s
1	1	0	1	0	1	1	1
1x128	1x64	0x32	1x16	0x8	1x4	2x1	1x1
128 + 64 + 16 + 4 + 2 + 1 = 215							

215 (denary) = 11010111 (binary)

Converting From Denary to Binary

You need to be able to **convert a denary number to binary**.

Step 1: Write the column headings	13 <table><tr><td>8s</td><td>4s</td><td>2s</td><td>1s</td></tr></table>	8s	4s	2s	1s				
8s	4s	2s	1s						
Step 2: Put a one in the largest column that fits inside the denary number	13 <table><tr><td>8s</td><td>4s</td><td>2s</td><td>1s</td></tr><tr><td>1</td><td></td><td></td><td></td></tr></table>	8s	4s	2s	1s	1			
8s	4s	2s	1s						
1									
Step 3: Subtract that column value from your denary number	$13 - 8 = 5$								

Converting From Denary to Binary

Step 4: Repeat Steps 2 and 3 until your denary number equals 0

~~13~~ 5

8s	4s	2s	1s
1	1		

$$5 - 4 = 1$$

~~13~~ 5 1

8s	4s	2s	1s
1	1	0	1

$$1 - 1 = 0$$

Step 5: Double check your answer by adding up the columns with a 1 in them

$$8 + 4 + 1 = 13 \checkmark$$

Converting From Binary to Denary

You also need to be able to **convert a binary number to denary**.

Step 1: Write out your binary number	<table><tr><td>1</td><td>0</td><td>1</td><td>0</td></tr></table>	1	0	1	0				
1	0	1	0						
Step 2: Write the column headings used in the binary system	<table><tr><td>8s</td><td>4s</td><td>2s</td><td>1s</td></tr><tr><td>1</td><td>0</td><td>1</td><td>0</td></tr></table>	8s	4s	2s	1s	1	0	1	0
8s	4s	2s	1s						
1	0	1	0						
Step 3: Add up the columns with a 1 in the number	$8 \times 1 + 4 \times 0 + 2 \times 1 + 1 \times 0$ $8 + 2 = 10$								

Binary number system

128s	64s	32s	16s	8s	4s	2s	1s
------	-----	-----	-----	----	----	----	----

Each 1 or 0 is a 'bit'. The more bits used, the more numbers that can be represented. For example:

- Using **4 bits** we can represent the numbers 0 to 15
- Using **8 bits** we can represent the numbers 0 to 255

The range of positive integers that can be represented using n bits is 0 to $2^n - 1$

Representing Text

The set of characters that can be represented by a computer is known as the **character set**, and is made up of letters, numbers and symbols.

A standardised code that is used by most systems is the **Extended American Standard Code for Information Interchange** (Extended ASCII).

Using a standardised code means that text can be exchanged across platforms easily.

ASCII

Every key on your keyboard is a **printable character** that has an ASCII code, meaning that it can be represented by a binary number.

Extended ASCII uses a unique **8-bit code** to represent each character, giving a possible 256 different characters. Each character uses 8 bits of memory.

There are some Extended ASCII codes worth remembering:

- The uppercase alphabet starts at code 65
- The lowercase alphabet starts at code 97
- Special control characters are codes 0 to 31 (non-printable)

Extended ASCII Codes

0	<NUL>	32	<SPC>	64	@	96	`	128	À	160	†	192	ˆ	224	†
1	<SOH>	33	!	65	A	97	a	129	Á	161	°	193	ı	225	˙
2	<STX>	34	"	66	B	98	b	130	Â	162	¢	194	ı	226	ı
3	<ETX>	35	#	67	C	99	c	131	Ã	163	£	195	√	227	ı
4	<EOT>	36	\$	68	D	100	d	132	Ä	164	§	196	f	228	‰
5	<ENQ>	37	%	69	E	101	e	133	Å	165	•	197	≈	229	Å
6	<ACK>	38	&	70	F	102	f	134	Ö	166	¶	198	Δ	230	Ê
7	<BEL>	39	'	71	G	103	g	135	á	167	ß	199	«	231	Á
8	<BS>	40	(72	H	104	h	136	à	168	®	200	»	232	È
9	<TAB>	41)	73	I	105	i	137	â	169	©	201	...	233	É
10	<LF>	42	*	74	J	106	j	138	ä	170	™	202		234	ı
11	<VT>	43	+	75	K	107	k	139	å	171	ˆ	203	À	235	ı
12	<FF>	44	,	76	L	108	l	140	â	172	"	204	Ã	236	ı
13	<CR>	45	-	77	M	109	m	141	ç	173	≠	205	Ö	237	ı
14	<SO>	46	.	78	N	110	n	142	é	174	Æ	206	Œ	238	Ó
15	<SI>	47	/	79	O	111	o	143	è	175	Ø	207	œ	239	Ô
16	<DLE>	48	0	80	P	112	p	144	ê	176	∞	208	-	240	■
17	<DC1>	49	1	81	Q	113	q	145	ë	177	±	209	—	241	Ò
18	<DC2>	50	2	82	R	114	r	146	í	178	≤	210	"	242	Ú
19	<DC3>	51	3	83	S	115	s	147	ì	179	≥	211	"	243	Û
20	<DC4>	52	4	84	T	116	t	148	ï	180	¥	212	'	244	Ü
21	<NAK>	53	5	85	U	117	u	149	î	181	μ	213	'	245	ı
22	<SYN>	54	6	86	V	118	v	150	ñ	182	ð	214	÷	246	ˆ
23	<ETB>	55	7	87	W	119	w	151	ó	183	Σ	215	◊	247	˜
24	<CAN>	56	8	88	X	120	x	152	ò	184	Π	216	̄	248	˘
25	<EH>	57	9	89	Y	121	y	153	ô	185	π	217	̂	249	˙
26	<SUB>	58	:	90	Z	122	z	154	õ	186	ƒ	218	/	250	˚
27	<ESC>	59	;	91	[123	{	155	ö	187	ª	219	€	251	˛
28	<FS>	60	<	92	\	124		156	ú	188	º	220	<	252	˜
29	<GS>	61	=	93]	125	}	157	û	189	Ω	221	>	253	˘
30	<RS>	62	>	94	^	126	~	158	ü	190	æ	222	fi	254	˙
31	<US>	63	?	95	_	127		159	ü	191	ø	223	fi	255	˚

ASCII Control Characters

Control characters are characters that do not print anything on screen when you press the key on the keyboard. These control characters include **delete**, **backspace** and **escape**.

Most of these characters are the first 32 codes, using ASCII values 0-31. The exception is delete, which is ASCII code 127.

Representing Images

Computers use two different ways of storing images:

- as a grid of pixels (bitmapped graphics)
- as a series of instructions describing shapes and lines (vector graphics)

You need to be able to describe and compare both ways.

Bitmapped Images

A bit mapped graphic is a grid of pixels, where the computer stores the colour of each individual pixel.

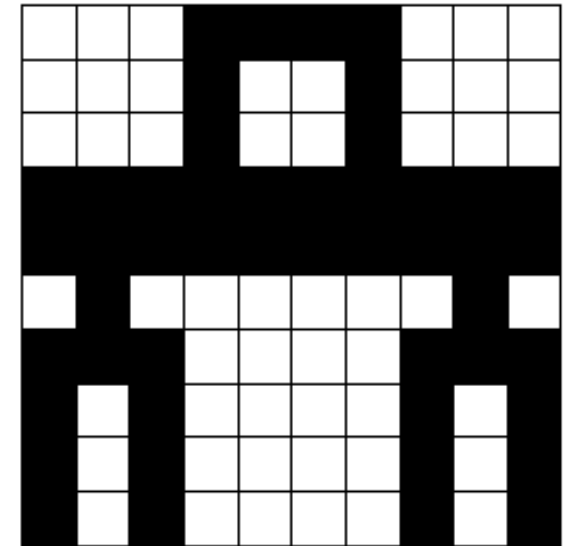
There are two definitions you need to know for bitmapped images:

- resolution
- colour depth

Resolution

Resolution is the number of pixels that make up the image. The resolution is calculated by multiplying the number of pixels horizontally by the number of pixels vertically.

In this image there are 10 pixels horizontally and 10 pixels vertically, so the resolution is 100 pixels.



Resolution

The resolution impacts the quality of the image - the more pixels, the better quality image, however the bigger the file size.



Colour Depth

Colour depth is the number of bits allocated to each pixel to represent colour.

For example, a 2-bit colour depth would allow four different values: 00, 01, 10, 11.

Binary Code	Colour
00	White
01	Light Grey
10	Dark Grey
11	Black

Colour Depth

The greater the colour depth, the more colours are available.

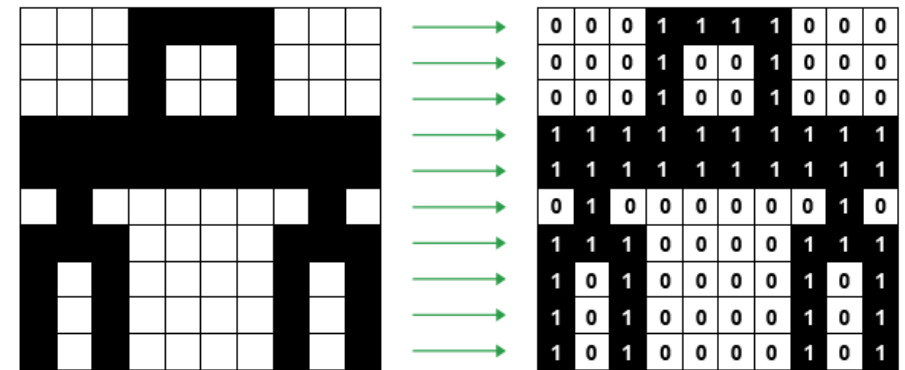
Colour Depth	Available Colours
1-bit	2
2-bit	4
3-bit	8
4-bit	16
8-bit	256



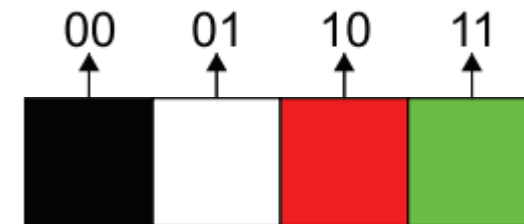
Bitmapped Images

When you know the resolution and colour depth, you can create a grid to show the binary value of each pixel and see how the image is stored in memory.

Here is how an image with a 1-bit colour depth would be mapped.



If the image has a higher colour depth, each pixel would be represented by multiple bits.



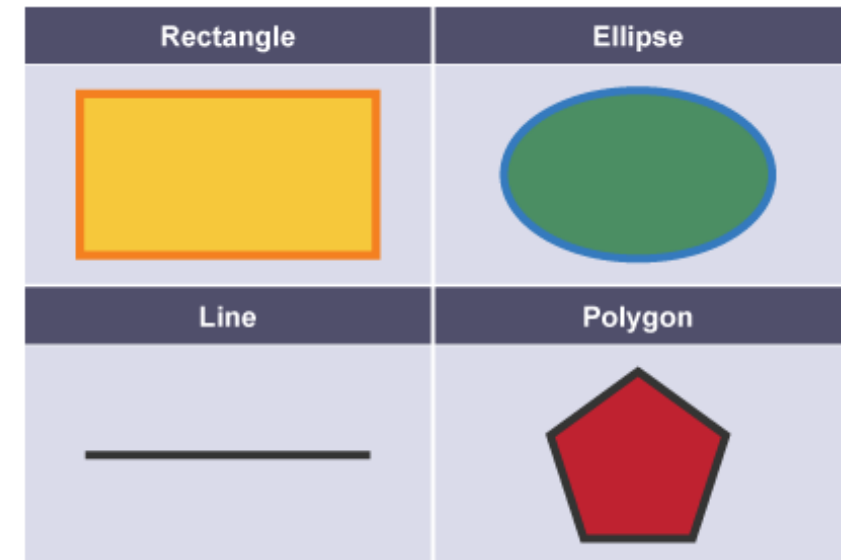
Vector Images

Instead of storing information about every single pixel, vector graphics store a list of shapes and attributes. When the image is opened, the computer will generate the shapes by looking at their attributes.

Vector Objects

There are many types of object (shapes) in vector graphics, but at National 5 you should know the following four:

- Polygons (irregular shapes with multiple points/sides)
- Lines
- Ellipses
- Rectangles



Vector Attributes

Attributes help to describe the shape. There are many attributes that can be used to describe objects, but at National 5 level you should know the following attributes:

Line colour: defines the colour of the outer line of an object.

Fill colour: defines the colour inside an object.

X and Y coordinates: the start position is the top left-hand corner of the drawing area (0,0). The X value defines how far to the right the object should be placed and the Y value defines how far down the object should be placed.

Vector vs Bitmap

	Advantages	Disadvantages
Bitmapped Graphics	<ul style="list-style-type: none">• Can represent highly detailed images e.g. photos• Can edit images at pixel level e.g. removing red-eye from photos• There are lots of standard file formats• Can create a wide range of irregular shapes and patterns	<ul style="list-style-type: none">• Large file sizes – even blank pixels need to be saved• Resizing results in pixilation• You cannot change objects once they have been drawn
Vector Graphics	<ul style="list-style-type: none">• Small file size – you only store data for the objects created• Resizing does not result in pixilation• Objects can be changed after they have been drawn	<ul style="list-style-type: none">• Cannot represent highly detailed images e.g. photos• Cannot edit image at pixel level• Cannot create a wide range of irregular shapes and patterns