# Learning Aims

- Explain how computers encode characters using 7-bit ASCII
- Identify the difference between ASCII and Unicode

# Character Set

ASCII and Unicode are the most widely used character sets.

The term Character set is used to describe the possible characters that can be represented in a computer system.

# Representation of text

- When you press a key on the keyboard a stream of 1's and 0's is input to the the processor
- The defined list of characters recognised by a computer's hardware and software is known as its **character set.**

**ASCII uses 7 bits**

- Used to represent lower-case and upper-case English characters and punctuation
- 65 to 90 upper-case
- 97 to 122 lower-case
- 48 to 57 0-9 numeric characters

However ASCII can not represent all languages

Computers represent characters in binary, these characters are represented by ASCII (American Standard Code for Information Interchange)

| D | B | C |
|---|---|---|
| 32 | 00100000 | space |
| 33 | 00100001 | ! |
| 34 | 00100010 | " |
| 35 | 00100011 | # |
| 36 | 00100100 | $ |
| 37 | 00100101 | % |
| 38 | 00100110 | & |
| 39 | 00100111 | ' |
| 40 | 00101000 | ( |
| 41 | 00101001 | ) |
| 42 | 00101010 | * |
| 43 | 00101011 | + |
| 44 | 00101100 | , |
| 45 | 00101101 | - |
| 46 | 00101110 | . |
| 47 | 00101111 | / |
| 48 | 00110000 | 0 |
| 49 | 00110001 | 1 |
| 50 | 00110010 | 2 |
| 51 | 00110011 | 3 |
| 52 | 00110100 | 4 |
| 53 | 00110101 | 5 |
| 54 | 00110110 | 6 |
| 55 | 00110111 | 7 |
| 56 | 00111000 | 8 |

| D | B | C |
|---|---|---|
| 57 | 00111001 | 9 |
| 58 | 00111010 | : |
| 59 | 00111011 | ; |
| 60 | 00111100 | < |
| 61 | 00111101 | = |
| 62 | 00111110 | > |
| 63 | 00111111 | ? |
| 64 | 01000000 | @ |
| 65 | 01000001 | A |
| 66 | 01000010 | B |
| 67 | 01000011 | C |
| 68 | 01000100 | D |
| 69 | 01000101 | E |
| 70 | 01000110 | F |
| 71 | 01000111 | G |
| 72 | 01001000 | H |
| 73 | 01001001 | I |
| 74 | 01001010 | J |
| 75 | 01001011 | K |
| 76 | 01001100 | L |
| 77 | 01001101 | M |
| 78 | 01001110 | N |
| 79 | 01001111 | O |
| 80 | 01010000 | P |
| 81 | 01010001 | Q |

| D | B | C |
|---|---|---|
| 82 | 01010010 | R |
| 83 | 01010011 | S |
| 84 | 01010100 | T |
| 85 | 01010101 | U |
| 86 | 01010110 | V |
| 87 | 01010111 | W |
| 88 | 01011000 | X |
| 89 | 01011001 | Y |
| 90 | 01011010 | Z |
| 91 | 01011011 | [ |
| 92 | 01011100 | \ |
| 93 | 01011101 | ] |
| 94 | 01011110 | ^ |
| 95 | 01011111 | _ |
| 96 | 01100000 | ` |
| 97 | 01100001 | a |
| 98 | 01100010 | b |
| 99 | 01100011 | c |
| 100 | 01100100 | d |
| 101 | 01100101 | e |
| 102 | 01100110 | f |
| 103 | 01100111 | g |
| 104 | 01101000 | h |
| 105 | 01101001 | i |
| 106 | 01101010 | j |

| D | B | C |
|---|---|---|
| 107 | 01101011 | k |
| 108 | 01101100 | l |
| 109 | 01101101 | m |
| 110 | 01101110 | n |
| 111 | 01101111 | o |
| 112 | 01110000 | p |
| 113 | 01110001 | q |
| 114 | 01110010 | r |
| 115 | 01110011 | s |
| 116 | 01110100 | t |
| 117 | 01110101 | u |
| 118 | 01110110 | v |
| 119 | 01110111 | w |
| 120 | 01111000 | x |
| 121 | 01111001 | y |
| 122 | 01111010 | z |
| 123 | 01111011 | { |
| 124 | 01111100 | \| |
| 125 | 01111101 | } |
| 126 | 01111110 | ~ |
| 127 | 01111111 | DEL |

KEY: D = denary
B = binary
C = character

# Unicode

- Unicode was developed to account for every language in the world.
- It uses 2 bytes that give us $2^{16}$ possibilities (65,536).
- An example use of this would allow a user from any country to select their language when setting up an operating system.
- The Unicode character set would account for every language

# Learning Objectives

Understand what pixels are and how images are represented in binary

Understand and be able to explain what resolution is and explain what colour depth is.

Understand and be able to explain why it is important to include metadata in bitmap images

# Keyword Definitions

| Keyword | Definition |
| --- | --- |
| Binary | is a sequence of 0's and 1's, it can read by a computer processor, decoded and executed |
| Bitmap Image | is made up of **pixels**, the colour of each pixel is defined by a unique sequence of **binary** |
| Image Resolution | Is the amount of **pixels per inch** in an image (width by height) |
| Colour Depth | is the amount of bits used to generate the colour in a single pixel |
| HEX | Base 16 number system used to represent HEX colour codes |
| Compression | Algorithms used to reduce the file size of an image (Lossy & Lossless Compression) |
| Metadata | Additional data stored with an image |

To store an image on a computer, the image is broken down into tiny elements called **pixels.**

A pixel represents just one colour.

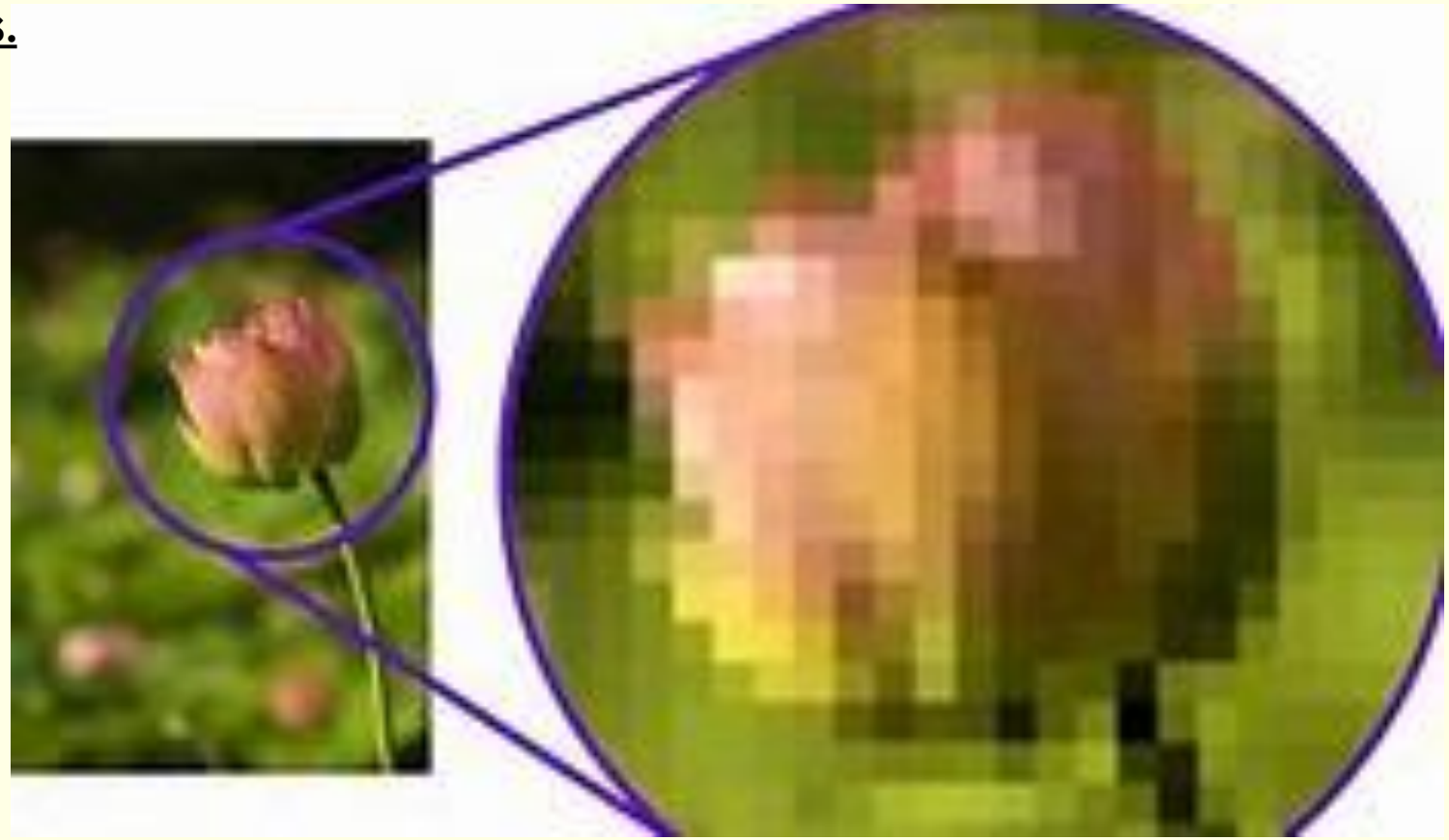


**Image Resolution** is the amount of pixels per inch in an image (width by height)

To generate a colour in a pixel can be between 1 to 24 bits (or more)

More bits = more colours = better image

**Colour depth** – is how many bits used to represent each pixel

For example: Image resolution of 1024 by 798 pixels has 1024 x 798 pixels (817,152 pixels).

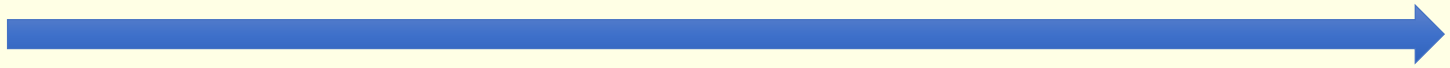
# Resolution

An image of size 100 x 60 when displayed in the same area as the one of 3390 x 2000 pixels, has far fewer pixels per inch and has a lower resolution
100 x 60 is pixelated but a smaller file size

▸ 100 × 60 pixels

▸ 3390 × 2000 pixels

File Size



Low High

# Encoding the pixel information

The colour of each pixel in a bitmal image is represented by a binary pattern

The colour depth of an image is the number of bits used to encode the colour of each pixel.

The greater number of bits per pixel = more colours

Example of 1bit colour depth (0 or 1)

Image Resolution: 8 x 8 = 64 pixels

Colour Depth: 1 bit

File size = 64 x 1 = 64 bits

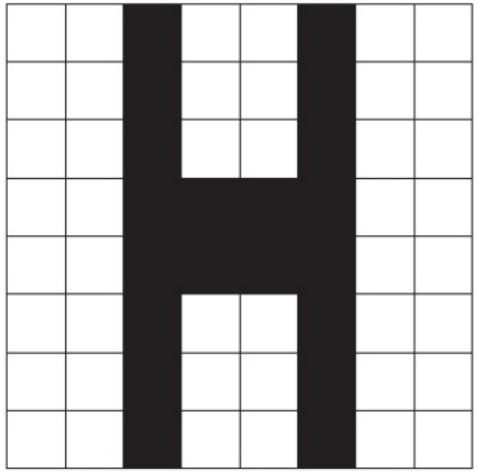
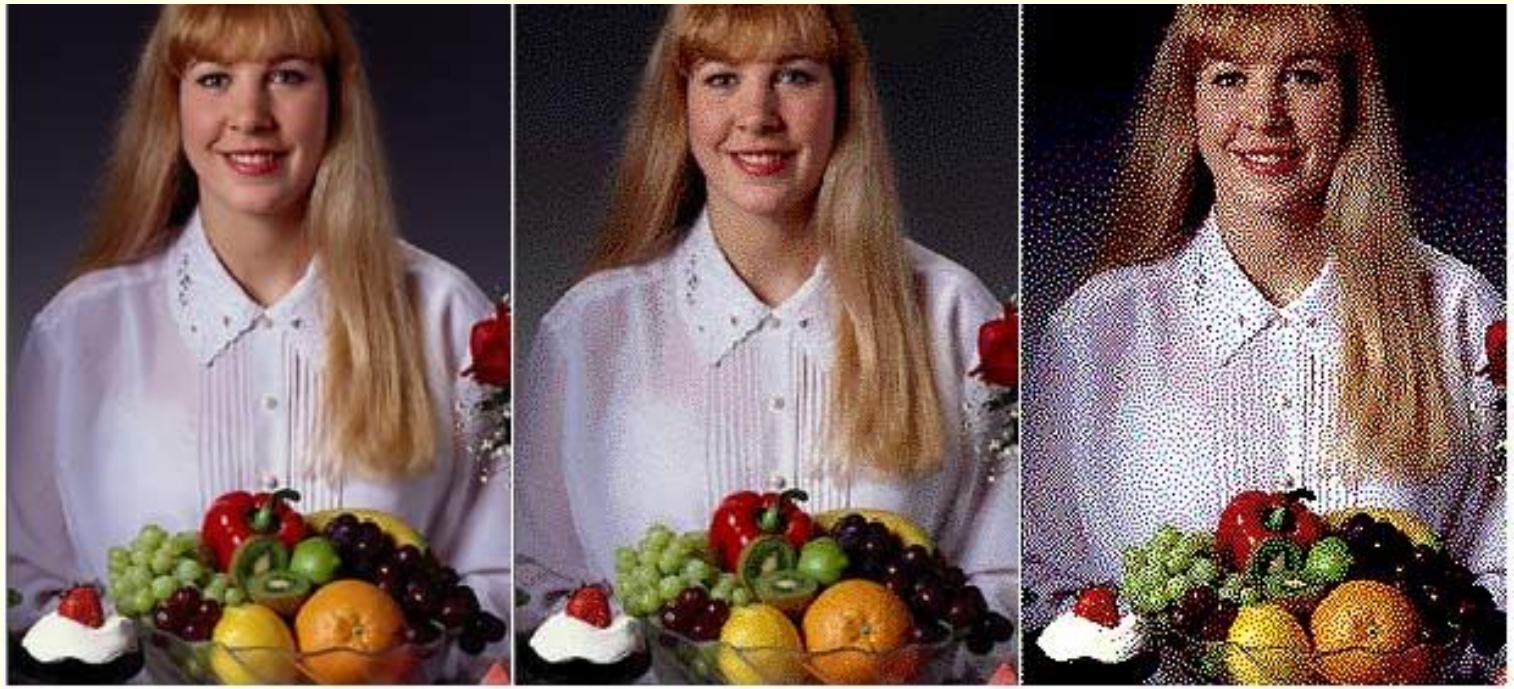


**Figure 2.2.3** One-bit encoding forming the letter 'H'

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |

**Figure 2.2.4** Binary code for the black and white image shown in Figure 2.2.3

# Colour Depth




File Size

High

Low

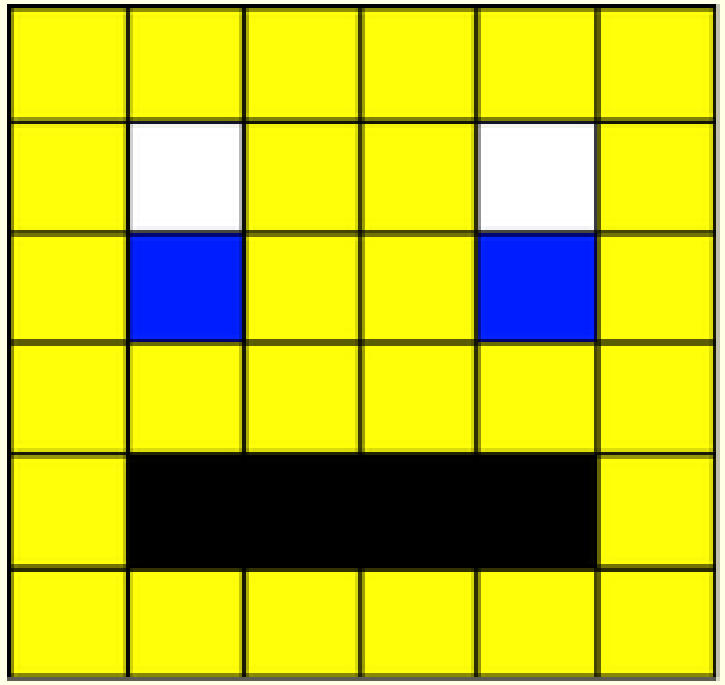
# Colour Depth 2

In an image that uses 4 colours, 2 bits are needed for each pixel.

The following example uses two bits to store the following colours:

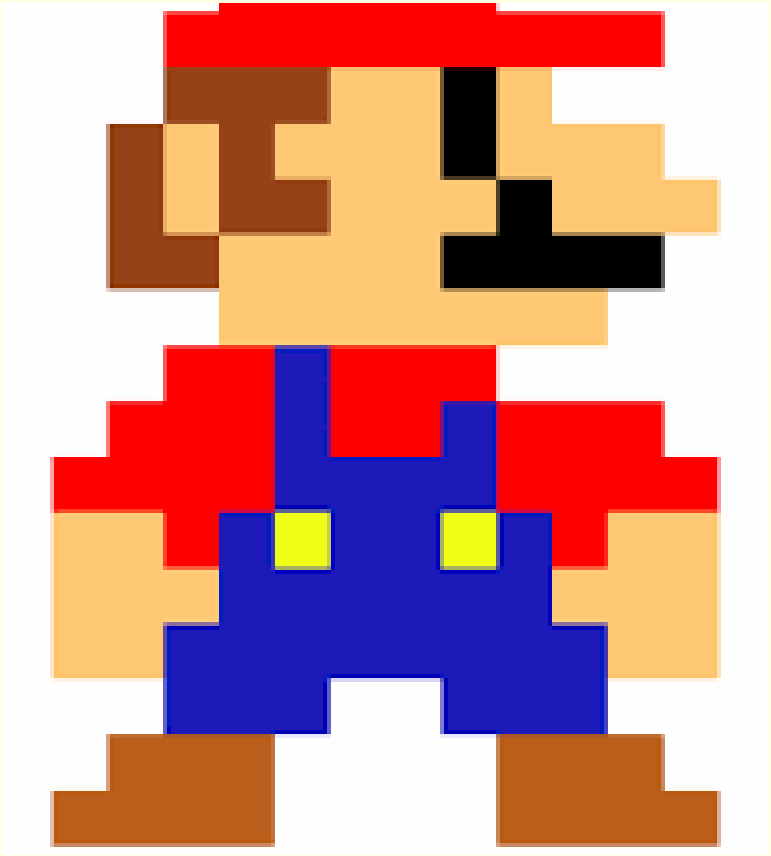
00 – White; 01 – Black; 10 – Yellow; 11 – Blue

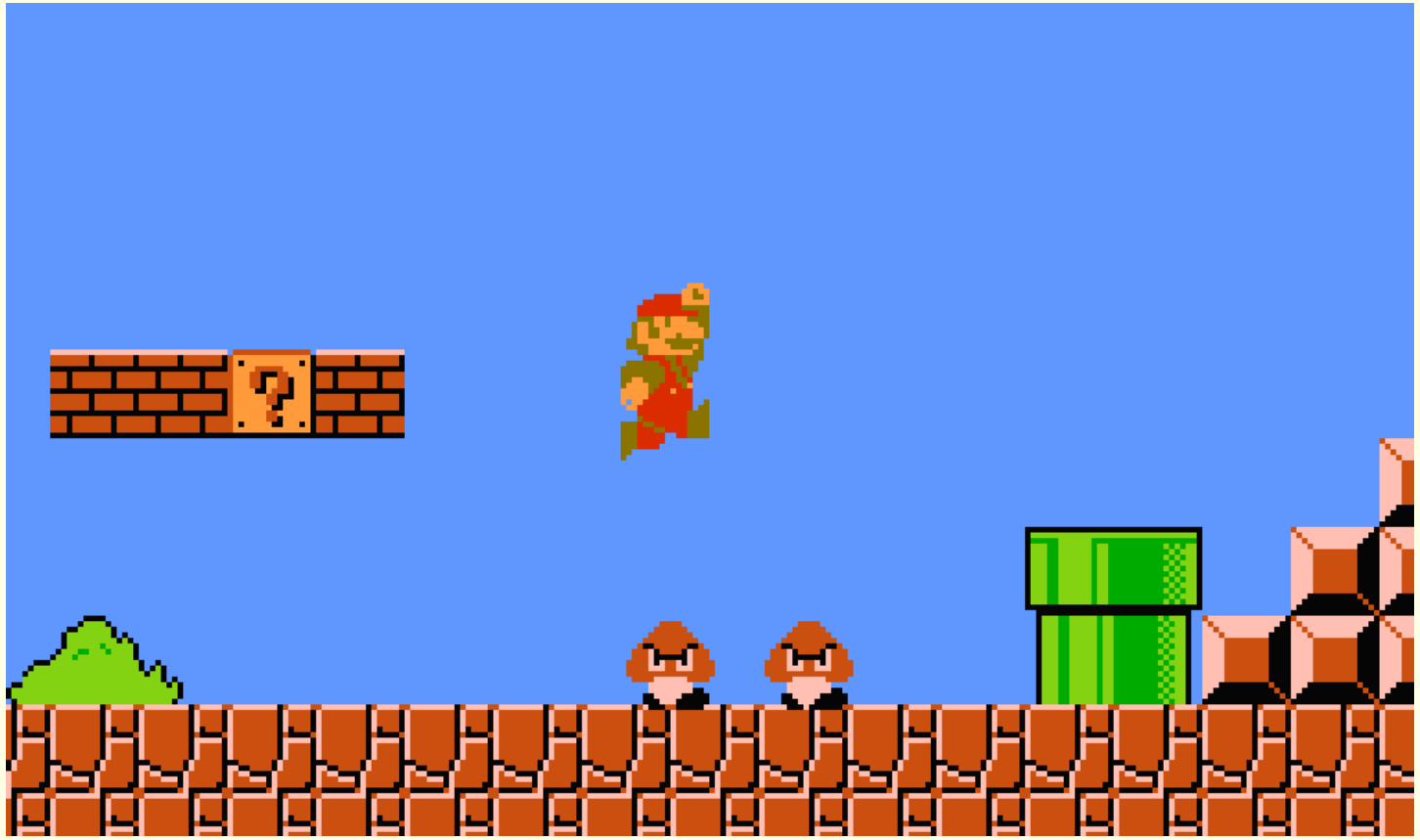
The image resolution is 8 x 8



| | | | | | |
|---|---|---|---|---|---|
| 10 | 10 | 10 | 10 | 10 | 10 |
| 10 | 00 | 10 | 10 | 00 | 10 |
| 10 | 11 | 10 | 10 | 11 | 10 |
| 10 | 10 | 10 | 10 | 10 | 10 |
| 10 | 01 | 01 | 01 | 01 | 10 |
| 10 | 10 | 10 | 10 | 10 | 10 |

101010101010
100010100010
101110101110
101010101010
100101010110
101010101010





8 bit colour depth gives 256 colours

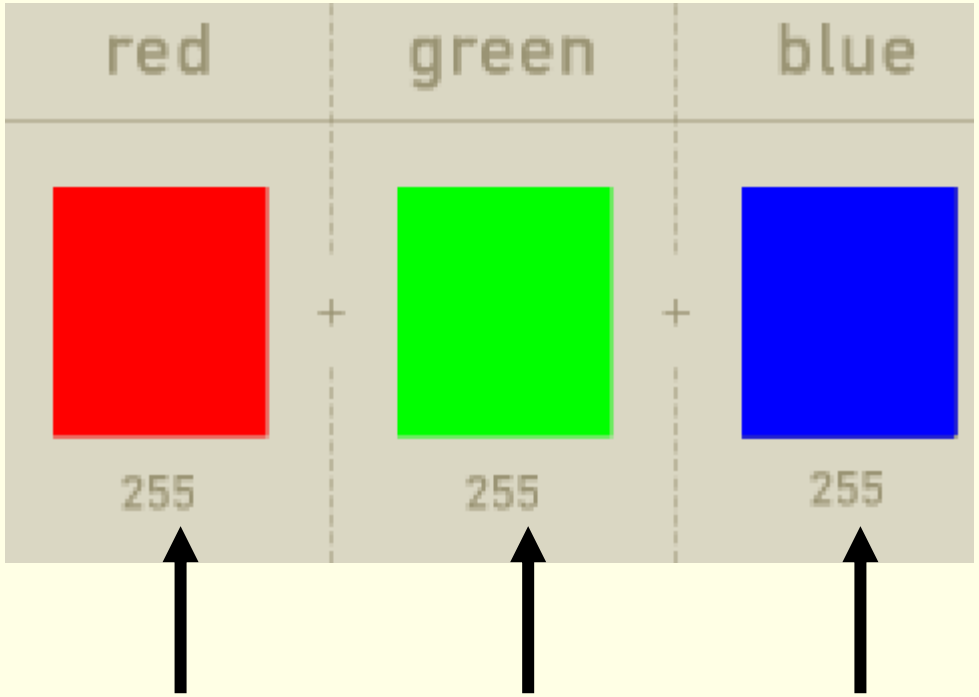
# True Colour 24 bit colour depth

By mixing the appropriate amount from each of the three colour channels you can get a variety of colours

| | R | G | B |
|---|---|---|---|
| | FF | FF | FF |
| | 00 | 00 | 00 |
| | 00 | 00 | FF |
| | 80 | 00 | 96 |
| | 96 | 00 | 80 |
| | 00 | FF | 00 |
| | FF | FF | 00 |
| | 80 | 80 | 00 |
| | FF | 00 | 00 |

What gets stored for each pixel is just a combination of each channel
E.g.

FFFFFF00 means the pixel is white
96008000 means the pixel is lilac




# 24 bit colour depth (True Colour)

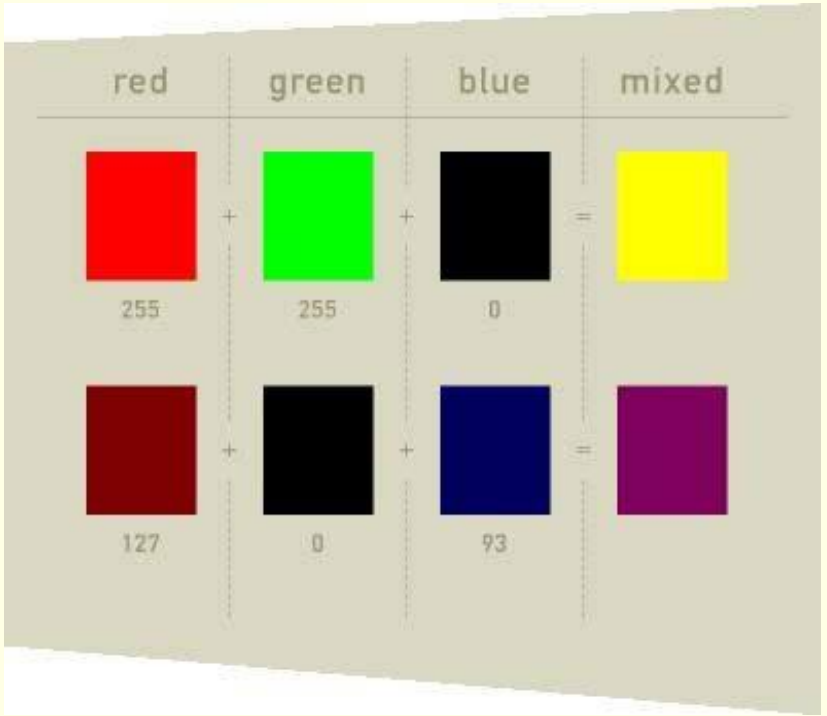
24-bit or “Truecolor “gives over 16.7 million colours

= 24 bits

111111111111111111111111

256 x 256 x 256 =

16,777,216 possible colours!



# Colour depth

| | |
|---|---|
| 1 bit allows 2 different values | 2 different colours (Black and White) |
| 2 bit allows 4 different values | 4 different colours |
| 3 bit allows 8 different values | 8 different colours |
| … | … |
| 8 bit allows 256 different values | 256 different colours |
| 24 bit allows 16,777,216 different values | 6,777,216 different colours (True Colour) |

The greater the colour depth:
- The more realistic colours
- The more data needs to be stored and the larger the file size on disk

# What effects the quality of an image?

- Resolution
(Number of Pixels per inch)

- Colour Depth
(Binary used to make the colour in each pixel)

***Balancing quality and image file size***

As the number of pixels and the colour depth increase, so too does the amount of data that has to be stored and manipulated.

If you were visiting a website with 4288 × 2848 24-bit images then, unless you had a very fast connection, you would quickly become frustrated as you waited for them all to download.

Often a compromise must be found between image quality and file size. As it happens, there's a limit to the number of colours the human eye can detect so increasing colour depth beyond 24 bits makes little sense.

In reality, images that are intended to be viewed on screen are usually compressed. You can read more about compression in Topic 2.3.

Higher image resolution + colour depth = Larger file size

# Metadata

## Bitmaps

```
00010001000011100000000111000111111110001110000011111000001000000001
11000000000000001110000000000000001000000000000000000000000100000
00000000001110000000000001111111000000011111111100000000011111110
00000000001111110000010000011111100001110000111111100111110011111
```

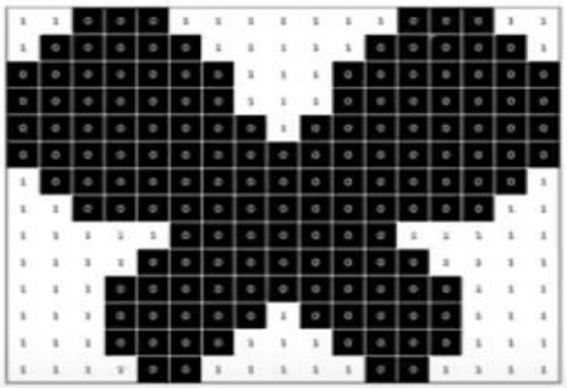
width: 17 pixels

height: 14 pixels

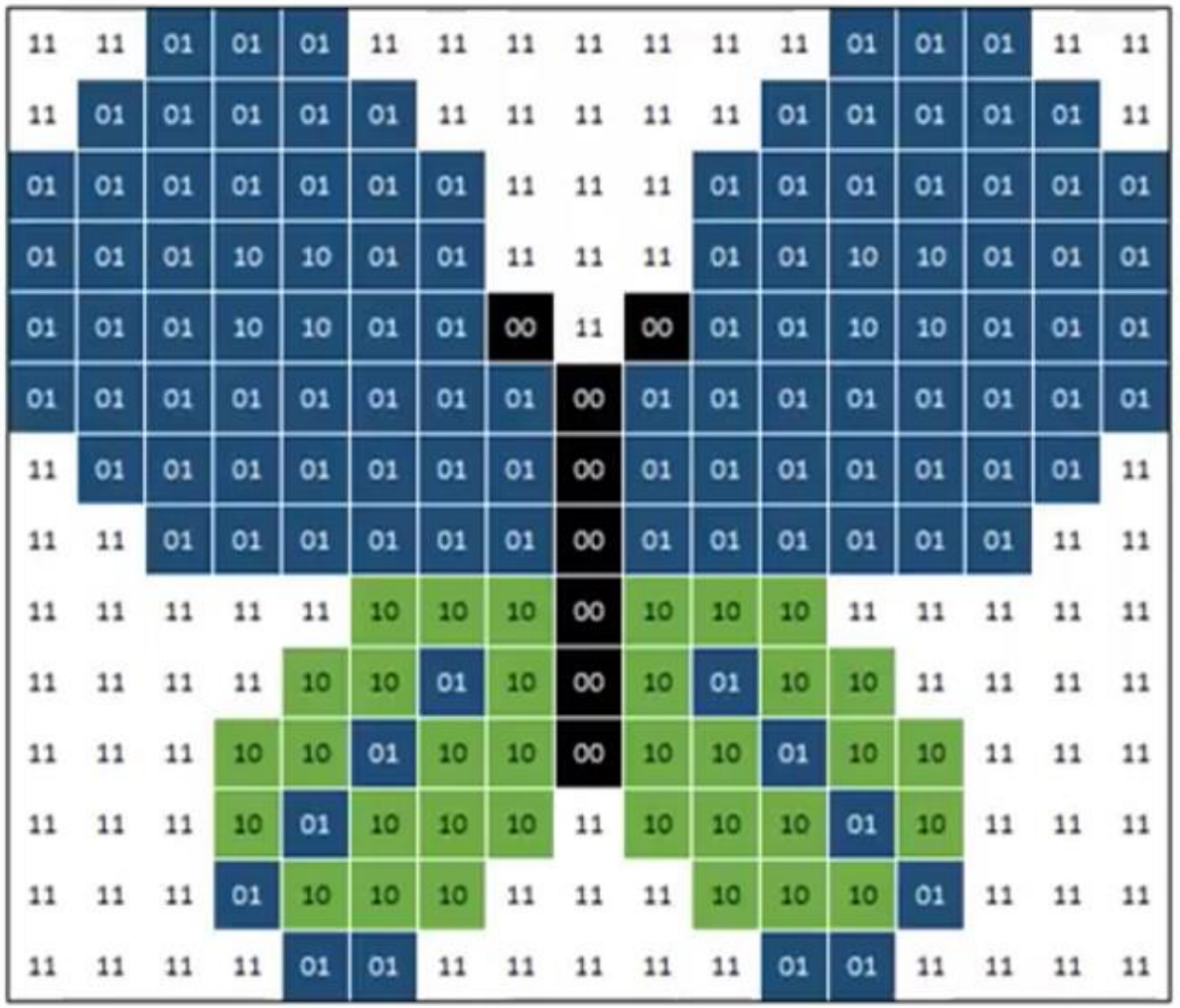
bits per pixel: 1

image data

metadata: additional data stored with the image such as width, height, colour depth of the image.



# Example




Metadata

Width 17 pixels

Height 14 pixels

Colour depth 2 bit

00

01

10

11

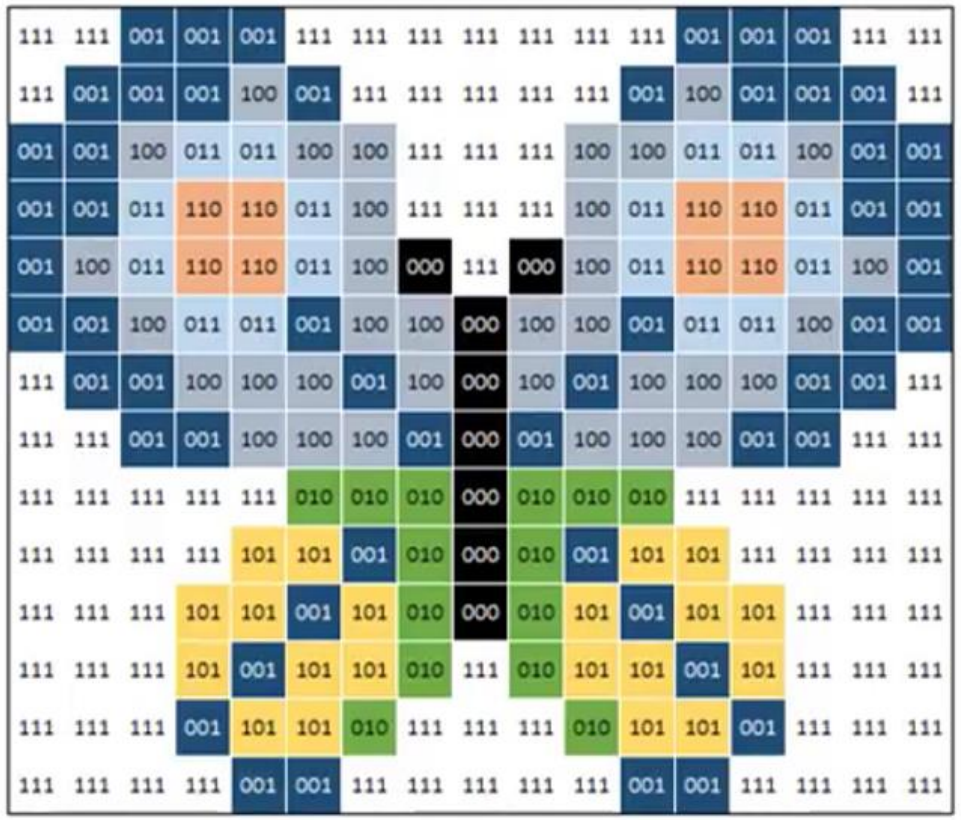
Raw file size 476 bits

+ 10% overheads for metadata

File size 524 bits

66 bytes

# Example




Metadata

| | | |
|---|---|---|
| Width | 17 | pixels |
| Height | 14 | pixels |
| Colour depth | 3 | bit |

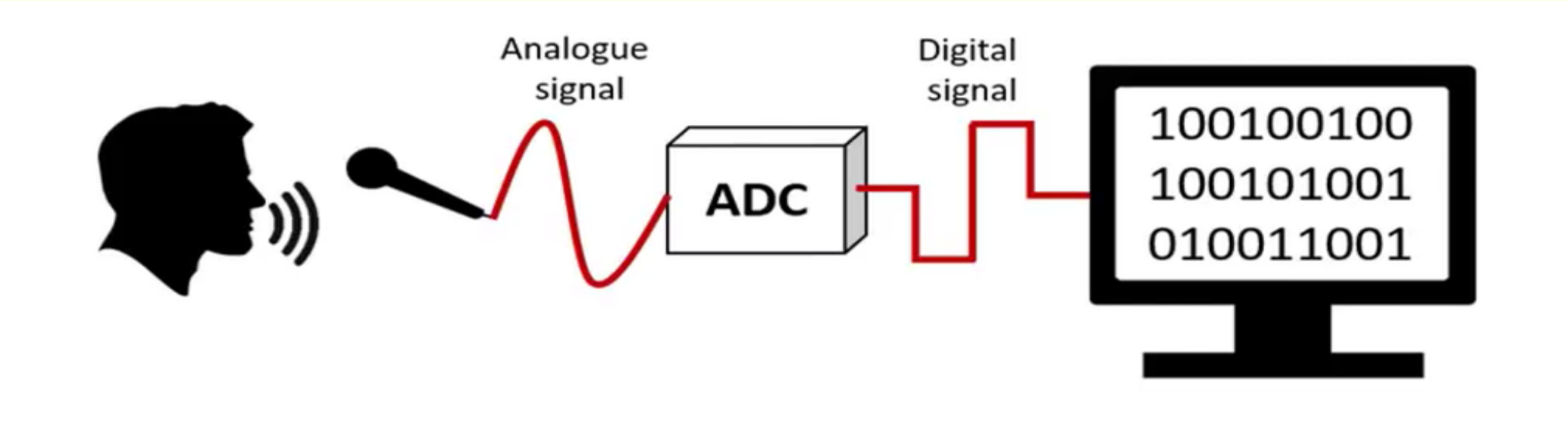
000
001
010
011
100
101
110
111

| | | |
|---|---|---|
| Raw file size | 714 | bits |
| + 10% overheads for metadata | | |
| File size | 785 | bits |
| | 99 | bytes |

# Learning Aims

- Understand how sound is stored into binary values
- Understand the factors that affect how sound is stored and how this affects the memory needed for storage.
- Understand and be able to explain why the factors affect memory storage and how this can be overcome through file compression.

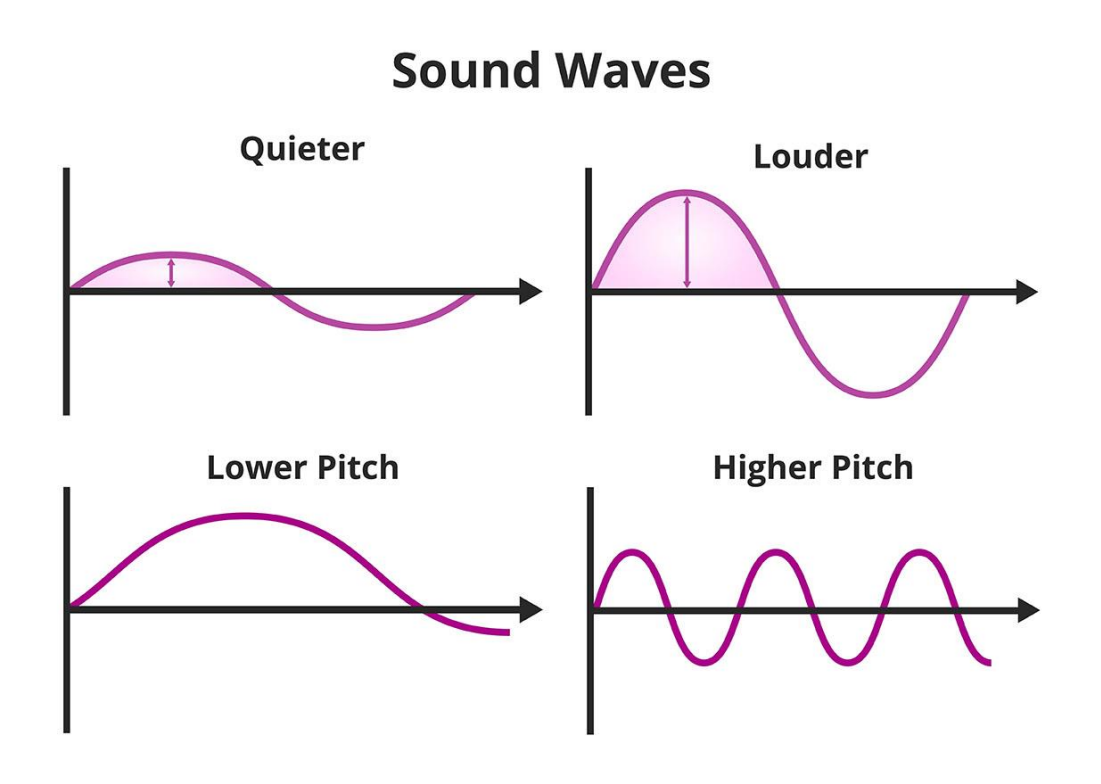
# How sound is sampled and stored in digital form




Digital sound is broken down into thousands of samples per second – each of these samples is then stored as binary data.

# Sound Wave Amplitude

**Amplitude** = How loud or soft a sound is (depends on wave height).



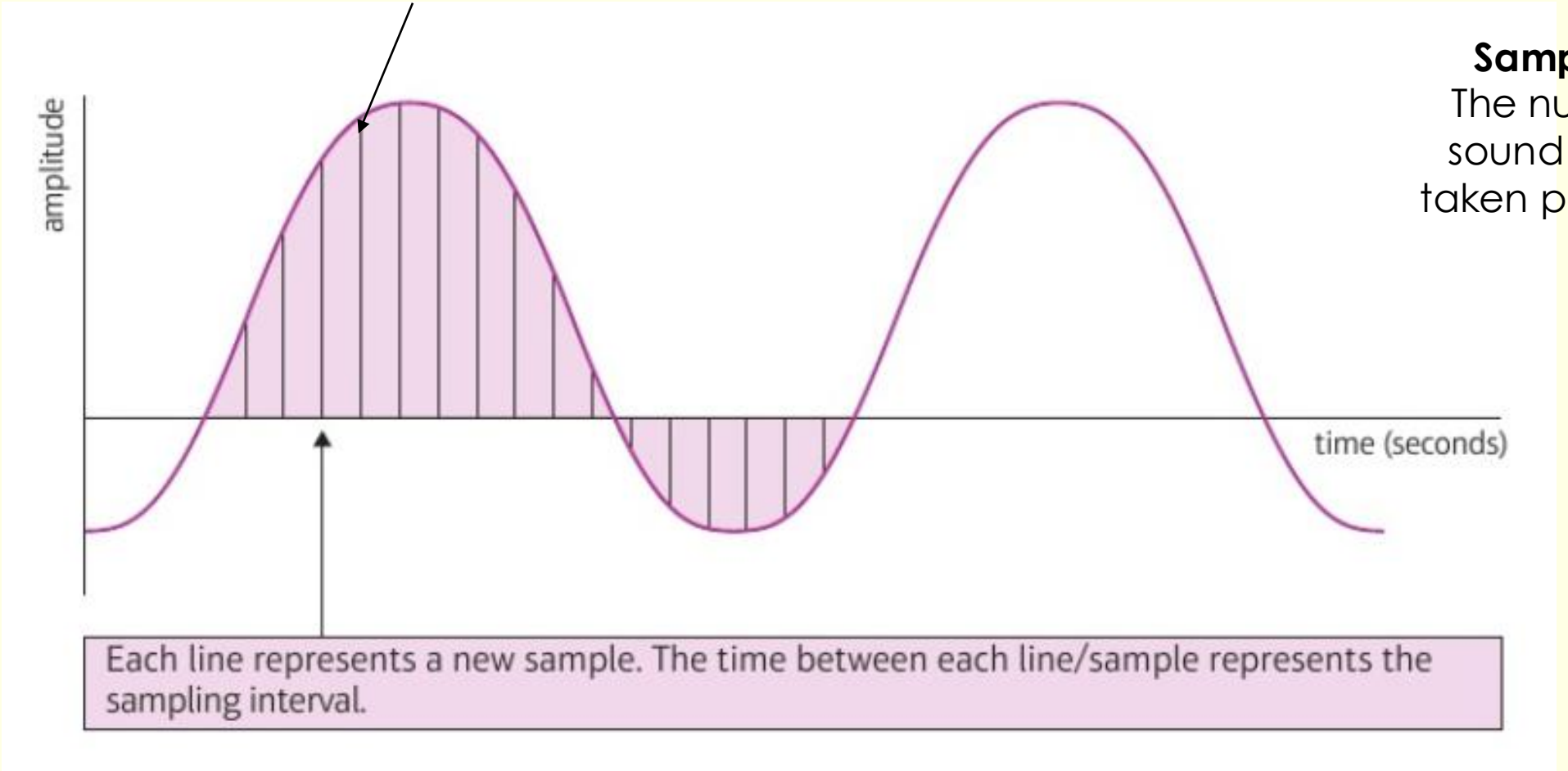

**Pitch** = How high or low a sound is (depends on frequency).

# How a sound wave is sampled digitally

**Amplitude** is stored as binary at each sampling point

**Sampling interval**
time between sampling points

**Sampling points**



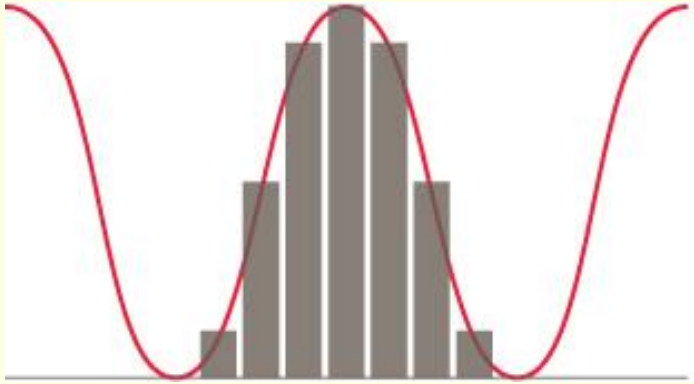

**Sample rate**
The number of sound samples taken per second

# Digital Audio Key Words

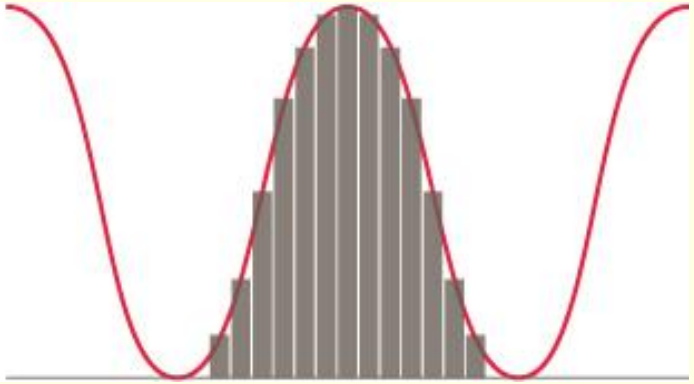
| **Sample Rate** | **Bit Depth** | **Sample Size** | **Bit Rate** |
| --- | --- | --- | --- |
| Sample rate<br>The number of sound samples taken per second<br>Measured in hertz | Number of bits used to encode each sample | The number of bits in a sound file | The number of bits processed per second of audio |

# Sample Rate

**Sample rate is measured in hertz.**



Low Sample Rate



High Sample Rate

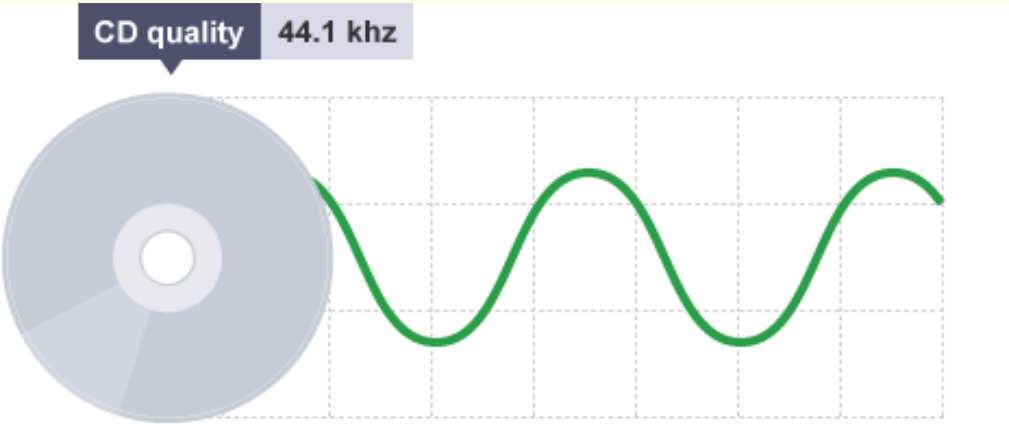
This refers to the number of samples taken per second

The higher the sample frequency the **more accurate** it represents the true sound wave

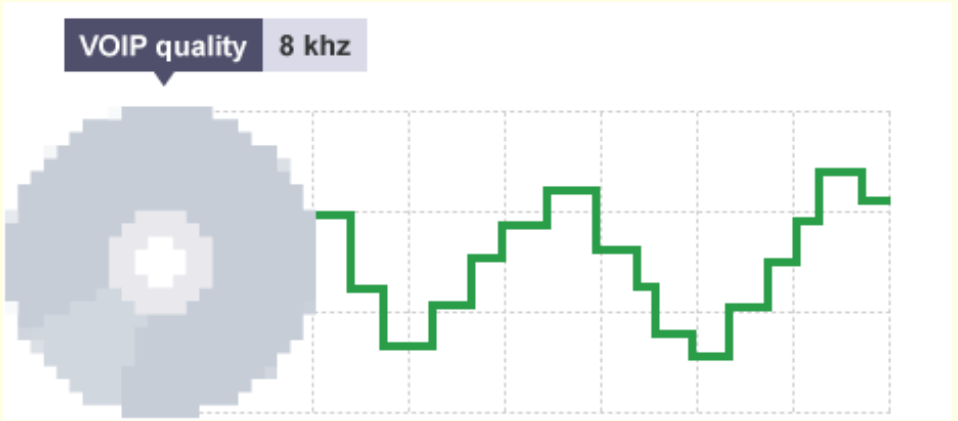
ALTHOUGH it increases the size of the file!

# Sample Rate Example

Common audio sample rate for music is 44,100 samples **per second (CDs) – the unit this is measured in is hertz –** This is 44,100 hertz or 44.1 kilohertz (kHz)




**Telephone networks and VOIP** services use a sample rate as low as 8 kHz



# Bit Depth

- Bit depth is the number of bits used to encode each sound sample.
- Using a higher bit depth allows much smaller changes in the volume difference to be recorded.
- If the bit depth is too low, the re3cording is not accurate and a lot of differences in the sound are lost.
- Using a bit depth of 8 bits (256) changes of sound can be measured compared with 24bits 16.7 million measured volume levels.

# Balancing quality and sound file size

- Sample rate and bit depth determine how accurately a digital representation matches the original analogue sound

- But the higher the sample rate and bit depth, the larger the file size

# Sample Size

**BIT DEPTH X SAMPLE RATE X NUMBER OF SECONDS**

# Bit rate

- Bit rate is simply a measure of how much data is processed for each second of sound. Bit rate is calculated by:

**bit rate = Frequency/sample rate × bit depth × channels**

- As with sample rate, the higher the bit rate, the better quality of the recorded sound.

# Learning Aims

01

**Understand** the need for compression

02

**Know** the difference between lossy and lossless.

03

**Analyse** when lossy and lossless would be used

# What is compression?

- Compression algorithms **reduce the size of the file**
- Data compression is the process of encoding data so that is needs fewer bits/bytes to represent it.
- Compression is useful because it helps reduce the consumption of expensive resources, such as **hard disk space** or transmission **bandwidth over the internet**.

# Lossless compression

- Lossless compression can compress data files **without** losing any of the information.

**Advantage:**
Lossless compression schemes are reversible so that the original data can be reconstructed.

**Disadvantage**
Reduces file sizes by around 50% compared with lossy that will reduce the file by 90%

# Lossy compression

- Lossy compression compresses data files but does lose some of the information.
- It removes data that is usually not detectable by humans, either sound or image.

**ADVANTAGE:**

- Can produce smaller data files.

**DISADVANTAGE:**

- Not good if 100% accuracy required e.g. text files

Common lossy format include JPEG and MP3

# Lossless compression algorithms

Lossless compression algorithms typically reduce file sizes by around 50 per cent. This is achieved by identifying and removing redundant data. In a page of text, for example, many words are used more than once. By creating a lookup table of recurring words, these can be removed from the text and replaced by tokens pointing to their position in the table. When the file is decompressed, the words can be put back into the text.

# Lossy compression algorithm

Lossy compression is particularly suited for reducing the file size of images.

Digital images often comprise 4000 pixels or more, each with a 24-bit colour depth. However, many of the minute differences in colour are wasted on us humans. Our eyesight is not capable of distinguishing them.

A lossy compression algorithm analyses all of the data in the image and when it finds areas with minute differences it gives them the same colour values. It can then rewrite the file using fewer bits.

Similarly, much of the data in an audio file encodes tones and frequencies that our ears can't hear, and small differences in volume and frequency that we cannot distinguish. MP3 files use a lossy algorithm to remove this superfluous data.

**Figure 2.3.2** Digitally compressing an image makes little difference to our eyes but gives huge savings in file size

# Learning Aims

- Use the units bit, byte, kibibyte, mebibyte gibibyte, tebibyte in calculating file size
- Construct an expression to convert units
- Construct an expression to calculate the file size of an image (width x height x colour depth) and – given the file size and the values of any two of the variables – to calculate the value of the remaining one
- Construct an expression to calculate the file size of a sound (sample rate x bit depth x time)

# Powers of 2

A binary is base 2 and uses binary multiples that are 2 to the power

- 1 **kibi**byte is 1,024 bytes = $2^{10}$ bytes  2 to power of 10
- Count in powers of 2:

  1, 2, 4, 8, 16, 32, 64, 128, 256, 512, **1024**

| Unit | Abbreviation | Bytes | Equivalent to |
|---|---|---|---|
| bit | | | 1 bit |
| nibble | | | 4 bits |
| byte | | $2^0$ bytes | 8 bits or 2 nibbles |
| kibibyte | KiB | $2^{10}$ bytes | 1024 bytes |
| mebibyte | MiB | $2^{20}$ bytes | 1024 kibibytes |
| gibibyte | GiB | $2^{30}$ bytes | 1024 mebibytes |
| tebibyte | TiB | $2^{40}$ bytes | 1024 gibibytes |

# In the exam

- In the exam you **must** use **binary bytes** for calculating **file sizes** and **storage**
- You will not have access to a calculator and so questions will only ask you to create an expression.
- You should also be able to rank units of measurement in size order and convert from a larger unit to a smaller one and vice-versa.

# Worked Example

A file uses 28 KiB of storage.

a) Construct an expression to calculate the number of bytes in the file.

$= \ file\ size\ in\ KiB \times 1024$

$= \ 28 \times 1024$ bytes

# Worked Example

Construct an expression to calculate how many bytes there are in 3 MiB.

$$= file\ size\ in\ MiB \times 1024 \times 1024$$

$$= 3 \times 1024 \times 1024 \quad \text{bytes}$$

# Worked Example

Construct an expression to calculate how many hexadecimal digits (nibbles) are needed to represent a binary bit pattern with a word length of 32 bits.

$$= \frac{number\ of\ bits}{4} = \frac{32}{4} \text{ hex digits}$$

$$= 32 / 4$$

# Worked Example

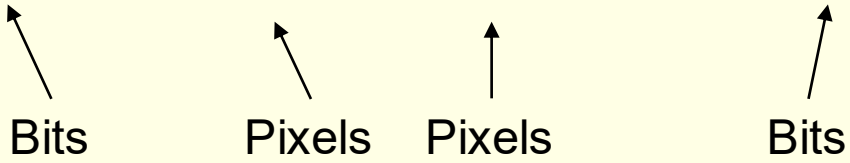
Construct an expression to show 23,354,273 MiB in TiB.

$$= \frac{file\ size\ in\ MiB}{1024\ x\ 1024} = \frac{23{,}354{,}273}{1024\ x\ 1024}\ \text{TiB}$$

# Calculate the file size of an image

The file size of an image is calculated by:

File size = width × height × colour depth




$$\frac{\text{(file size in bits)}}{\text{((bits in a byte) × (bytes in a kibibyte) etc.)}}$$

# Worked Example

An image is 500 pixels high and 400 pixels wide, with a colour depth of 16.

- Construct an expression to calculate the file size of the image in bits.

$= 500 \times 400 \times 16$

# Worked Example

Construct an expression to calculate the file size in kibibytes of:

i. An 8-bit colour image with a width of 640 pixels and a height of 480 pixels.

$$\frac{(640 \times 480 \times 8)}{(8 \times 1024)}$$

Converting to kibibytes

# Mix it up

- Now, the formula all together is:

$$\frac{\text{(width} \times \text{height} \times \text{colour depth)}}{\text{(file size in units)}}$$

- If you know three of the four values, you can calculate the missing one
- This is just rearranging the formula the same way you do in Maths and Science.

# Worked Example

An image has a file size of 454 kibibytes, its height is 800 pixels and its width is 600 pixels.

- Construct an expression to calculate the colour depth of the image.

$$\frac{(800 \times 600 \times colour)}{(1024 \times 8)}$$

colour depth = file size / width × height

Rearranged:

$$\frac{(454 \times 8 \times 1024)}{(800 \times 600)}$$

# Calculating file size of an audio file

- Just like image representation, we can calculate the size of an audio file based on its characteristics.

$$\text{file size} = \text{sample rate} \times \text{bit depth} \times \text{time}$$

- Just like image representation, we can also calculate any one unknown, if we know the other three

- Remember to use the correct units.

# Worked Example

Construct an expression to show the size of a sound file, in KiB, with a sample rate of 44.1kHz, a bit depth of 16, and a duration of 90 seconds.

file size = sample rate × bit depth × time

$$\text{size} = \frac{(44100 \times 16 \times 90)}{(8 \times 1024)}$$

# Worked Example

The sample rate of a file is 10 kHz. Its file size is 3 MiB and its bit depth is 16 bits.

Construct an expression to calculate how long the sound will play.

Time = file size / (sample rate × bit depth)

$$\text{Time} = \frac{(3 \times 8 \times 1024 \times 1024)}{10 \times 16}$$