

# **Unit 12**

# Data Representation

Data noun

1. the quantities, characters, or symbols on which operations are performed by a computer, which may be stored and transmitted in the form of electrical signals and recorded on magnetic, optical, or mechanical recording media.

#### NAME \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#### CLASS \_\_\_\_\_\_\_\_\_\_\_\_\_

##### THE NEED TO REPRESENT DATA DIFFERENTLY?

There are 10 types of people in the world. Those who understand binary and those who don’t.

If that quote doesn’t make any sense now then hopefully by the end of this workbook it will. In primary school when learning about **place values** you may have put headings above numbers like the table below.

|  |  |  |
| --- | --- | --- |
| **H** | **T** | **U** |
|  |  | 5 |
|  | 1 | 2 |
| 2 | 5 | 4 |

Can you remember what do the headings mean? They stand for hundreds, tens and units. The smallest digit that a column can hold is 0 while the largest is 9. If 1 is added to a column then that column goes back to 0 and 1 is added to the next column. So when 1 is added to 9 this becomes 10. So we now have 1 ten and no units.

##### How data is stored in computers

Computers are made up from billions of switches. These switches can be ON or OFF. Everything in a computer from numbers to text, from sounds to pictures must be converted into a series of ON’s and OFF’s.

Computer scientists often think of these switches as 0s and 1s.

**0 = OFF**

**1 = ON**

Each 0 or 1 is known as a **binary** **digit** or **Bit**.

Computers would be pretty useless if the biggest number it could deal with was the number 1 unless it had a way of storing larger numbers from a series of 0s and 1s. This is **Binary**. It is a way of storing numbers as a series of 0s and 1s.

##### INTRODUCTION TO BINARY

Computers often group together 8 bits. This makes one byte.

Just like in denary (the number system we use) we often use column headings to remind us of the place value.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **128** | **64** | **32** | **16** | **8** | **4** | **2** | **1** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

This is the number 1. Each column heading is a power of 2.

##### HOW TO CONVERT FROM DENARY TO BINARY

You often need to convert a number from denary into binary. How to convert 37 into a binary number. What is the biggest number that goes into 37 without going over? Looking at the grid 32 is the highest number, so we put a 1 in that column.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **128** | **64** | **32** | **16** | **8** | **4** | **2** | **1** |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

Now we take 32 from 37.

This leaves 5. Now we repeat the process.

4 goes into 5 without going over. So we put a 1 in the 4 column.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **128** | **64** | **32** | **16** | **8** | **4** | **2** | **1** |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |

That leaves us with 1. So we can put a 1 in the 1 column and then this is our answer.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **128** | **64** | **32** | **16** | **8** | **4** | **2** | **1** |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |

So 37 in binary is 00100101 or just 100101.

##### CONVERTING BINARY NUMBERS INTO DENARY

This is much easier. It is just a matter of adding up the columns which contain a 1. Convert 01010110 into denary. Put the numbers under the place headings.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **128** | **64** | **32** | **16** | **8** | **4** | **2** | **1** |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |

So the answer is 64+16+4+2 = 86.

##### EXERCISE

Convert the following decimal numbers into an eight bit binary number

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Convert this number into Binary | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| 5 |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |
| 17 |  |  |  |  |  |  |  |  |
| 38 |  |  |  |  |  |  |  |  |
| 39 |  |  |  |  |  |  |  |  |
| 41 |  |  |  |  |  |  |  |  |
| 46 |  |  |  |  |  |  |  |  |
| 58 |  |  |  |  |  |  |  |  |
| 69 |  |  |  |  |  |  |  |  |
| 115 |  |  |  |  |  |  |  |  |
| 128 |  |  |  |  |  |  |  |  |
| 133 |  |  |  |  |  |  |  |  |
| 134 |  |  |  |  |  |  |  |  |
| 156 |  |  |  |  |  |  |  |  |
| 160 |  |  |  |  |  |  |  |  |
| 166 |  |  |  |  |  |  |  |  |
| 203 |  |  |  |  |  |  |  |  |
| 239 |  |  |  |  |  |  |  |  |
| 254 |  |  |  |  |  |  |  |  |

Convert the following binary numbers into decimal.

|  |  |
| --- | --- |
| 00000011 |  |
| 00010110 |  |
| 00011011 |  |
| 00100010 |  |
| 00100111 |  |
| 00101011 |  |
| 00110000 |  |
| 00110010 |  |
| 00110100 |  |
| 00110101 |  |
| 01010110 |  |
| 01100000 |  |
| 01110000 |  |
| 01111010 |  |
| 10000110 |  |
| 10001101 |  |
| 10100011 |  |
| 10101001 |  |
| 11101101 |  |
| 11110101 |  |

##### BINARY ADDITION

You may be asked to add two binary numbers together. You will not be asked to add numbers bigger than eight bits. Binary addition is simpler than denary addition as there are fewer things to remember.

0 + 0 = 0

1 + 0 = 1

1 + 1 = 0 and carry 1

1 + 1 + 1 = 1 and carry 1

##### ADDING TWO BINARY NUMBERS

**+**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **128** | **64** | **32** | **16** | **8** | **4** | **2** | **1** |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
|  |  |  |  |  |  |  |  |

By following the simple rules we get the answer:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **128**  **+** | **64** | **32** | **16** | **8** | **4** | **2** | **1** |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1  1 | 0 | 1 | 0 | 0 | 1 |
| 1  1 | 1 | 0 | 1  1 | 0 | 0 | 1 | 1 |

You will never be asked to add up numbers that are more than 8 bits.

What happens if we need to carry a 1 after the 128 column?

1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **128**  **+** | **64** | **32** | **16** | **8** | **4** | **2** | **1** |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

The last carry digit has nowhere to go as the number needs a 9th bit. This is an **overflow** error.

##### EXERCISE

Add the following pairs of binary numbers. Answer as an 8 bit binary number.

|  |  |  |  |
| --- | --- | --- | --- |
| 00000111 | + | 00000011 |  |
| 00011000 | + | 00000101 |  |
| 00100000 | + | 00011111 |  |
| 01001110 | + | 00010110 |  |
| 11001000 | + | 00111000 |  |
| 10011000 | + | 10110011 |  |

##### HEXADECIMAL

Hexadecimal is another number system used in computer science. It is often used as a shorthand way for us (Humans) to write a binary number. Hexadecimal is a base 16 number system. So the column headings are based on powers of 16.

|  |  |
| --- | --- |
| **16** | **U** |
| 1 | 0 |

This gives us a problem as the number 16 has 2 digits and number systems need to have just a single digit in each column

|  |  |
| --- | --- |
| **16** | **U** |
| **0** | **10** |

So the number 10 above in base 16 is the same as 16 in denary, 1 sixteen and zero units. The problem arises when we have more than 9 units. The example above us impossible. If we were to write down 010 in base 16 we would confuse it with the previous example.

The solution is to use letters instead of numbers. So the values that you can get in a column in base

16 are as follows.

|  |  |
| --- | --- |
| **Denary** | **Hexadecimal** |
| 0-9 | 0-9 |
| 10 | A |
| 11 | B |
| 12 | C |
| 13 | D |
| 14 | E |
| 15 | F |

So if we want to display the number 12 as a hexadecimal number we would write C. We would write

**31** in hexadecimal as **1F** (1 x 16) + (15 x 1).

|  |  |
| --- | --- |
| **16** | **U** |
| **1** | **F** |

How is this used as a short cut for a binary number? Well look at the following binary number.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **128** | **64** | **32** | **16** | **8** | **4** | **2** | **1** |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |

We can split this into two 4 bit chunks (**nibbles**) like this.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **128** | **64** | **32** | **16** |  | **8** | **4** | **2** | **1** |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |

|  |  |  |  |
| --- | --- | --- | --- |
| **8** | **4** | **2** | **1** |
| 1 | 0 | 1 | 0 |

|  |  |  |  |
| --- | --- | --- | --- |
| **8** | **4** | **2** | **1** |
| 1 | 1 | 0 | 1 |

Looking at the right nibble we can see that it is the number 12 in denary. We can write this as the hexadecimal number C.

Now looking at the left nibble in the same way. We will deal with the column headings just like we did for the right nibble. So 1010 in binary is ten in denary. We would write this as A in hexadecimal.

So the binary number **10101100** can be written more simply as **AC** in hexadecimal.

Remember that **AC** means:

There are 10 sixteens and 12 units. This makes 160 + 12 = **172**.

Checking this against the binary 10101100 we have 128 + 32 + 8 + 4 = **172**

##### CONVERTING FROM DENARY TO HEXADECIMAL AND BACK AGAIN

To convert a number from denary to hexadecimal we first divide the number by 16.

So if we look at the denary number 75. If we divide 75 by 16 we get 4 remainder 11(16 \* 4 = 64 so 75 - 64 gives us a remainder of 11).

The number in the 16s column is 4 and we have 11 units. Remember we need to use letters for denary numbers above 9. So 11 is B in hex.

So 75 in denary is the same as 4B in hexadecimal.

To convert a hexadecimal number into denary we need to first convert any letters into numbers. Then multiply the first digit by 16 and add on the units.

So converting A7 into denary would be 10 x 16 + 7 = 167

##### EXERCISE

Convert these numbers from / into Hexadecimal.

|  |  |
| --- | --- |
| 10001100 |  |
| 11110000 |  |
| 11000011 |  |
| 01111111 |  |
| 00111100 |  |
| 27 |  |
| 89 |  |
| 2C |  |
| AF |  |
| D5 |  |

##### BINARY SHIFT

##### If we shift a decimal point in a decimal number once, to the right, it multiplies the number by 10. If we shift the decimal point two places we multiply by 100.

**For example:** 1.20 🡪 12.0 🡪 120.

Likewise, if we shift the decimal point once, to the left, we divide by 10. I we shift the decimal point two places we divide by 100.

**For example:** 730 🡪 73.0 🡪 7.30

Binary uses a base 2 number system and therefore a shift one place to the left or right will multiply or divide by 2.

**A left shift** 1 place looks like this. A zero has been added to the right column and each bit has moved one place to the left.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **128** | **64** | **32** | **16** | **8** | **4** | **2** | **1** | **Denary** |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 42 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | **0** | 84 |

**A right shift** 1 place looks like this. A 0 has been added to the left column and each bit has moved one place to the right. The 0 in the right column is removed.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **128** | **64** | **32** | **16** | **8** | **4** | **2** | **1** | **Denary** |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | **0** | 42 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 21 |

\*\* It is worth noting that a right shift on an odd number with be an approximation for the integer value because we lose the last digit in the table.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **128** | **64** | **32** | **16** | **8** | **4** | **2** | **1** | **Denary** |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 42 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 21 |

##### EXERCISE

Use Binary Shift on the following numbers to multiply / divide (Check your answers by converting to denary after).

|  |
| --- |
| Apply a **left** shift one place to 00001101 |
|  |
| Apply a **right** shift one place 11110000 |
|  |
| Apply a **left** shift two places 00100011 |
|  |
| Apply a **right** shift one places 00101000 |
|  |
| Apply a **right** shift one places 00110001 |
| Why would this be incorrect? |

##### STORING NON-NUMERICAL DATA

All non-numerical data such as text, sounds, images and instructions are first converted into a number and then stored as a binary sequence. The computer deals with data depending on the instructions given to it.

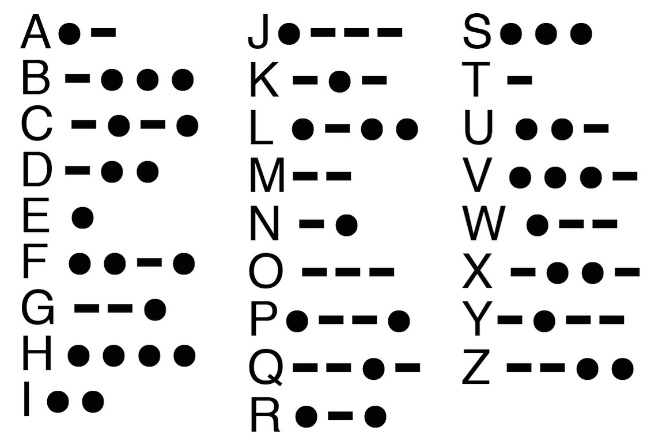
##### TEXT

**Character sets**

Just like numbers, text needs to be stored as a binary code too. Each letter has a particular binary code. It is important for transfer of information that the binary codes used for each character are known between Operating Systems, so that data can be accurately transferred between computers no matter what type of computer system is used. Character sets are nothing new.

Morse code has a character set to describe the codes used for letters and numbers. As each telegraph operator knew Morse code it made it possible to transmit text in a binary form (dots and dashes).

When computers were developed, they needed a way to store text that could be encoded. The problem with Morse code is that each character has a different length. This would cause issues for computers as they wouldn’t know when one letter stops and another one starter.

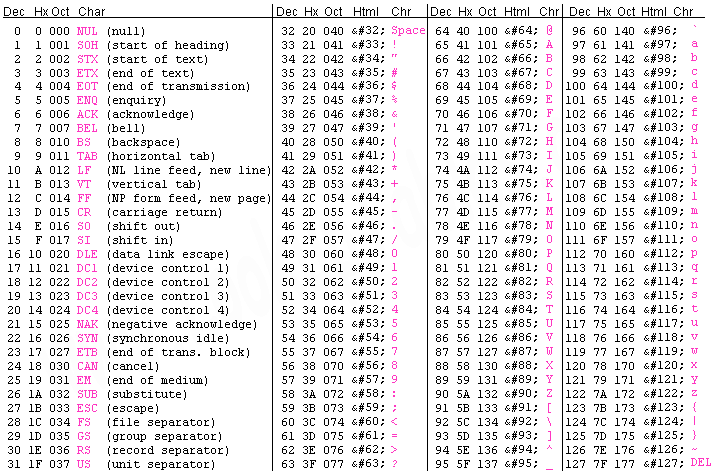


##### ASCII

One of the first character sets created was ASCII. This stands for **American Standard Code for Information Interchange**. Below you can see the ASCII table to show how each **character** is represented by a number. The ASCII **character set**, for example, uses the numbers 0 through 127 to represent all English **characters** as well as special control **characters**.

How many bits are used to store ASCII code?

|  |
| --- |
|  |
|  |



What is Extended ASCII?

|  |
| --- |
|  |
|  |
|  |

In the 2000’s it became clear that ASCII did not have enough characters for all the symbols that needed to be used. Some languages such as Japanese and Mandarin have character sets that are far bigger than the Latin character sets used by the west.

**UNICODE**

Unicode was developed to store letters as a **16 bit** code. As more bits are used then more characters could be encoded. Unicode was designed to be large enough so that would it encompass all the characters in all the major languages of the world. Almost all modern computer systems now use Unicode to store text. The characters in ASCII are assigned the same binary number as those in Unicode. For example, 1000000 (64) would be @ in both ASCII and Unicode.

Remember that the letter is not actually stored, but a binary code to represent that character. When the computer reads that data, a computer program has to figure out which character to display on the screen for each binary code.

There is also a problem when storing numbers. The binary code for the digits 0-9 is not the binary numbers 0-9. The ASCII value for the character 5 is 53 in binary. In some computer programs data may be stored as text or as numbers. If a calculation needs to be done on numbers which have been stored as text characters, then the data will need to be converted into the right data type first.

Use the table on the previous page to work out the following characters from their binary value

|  |  |
| --- | --- |
| Binary value | Character |
| 01000001 |  |
| 01100001 |  |
| 01100100 |  |
| 00110010 |  |

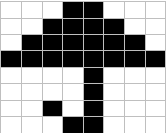
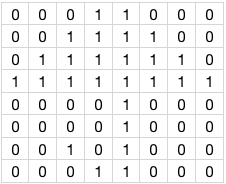
##### IMAGES

##### Pixels

Images are broken down into tiny coloured squares called **picture elements** or **pixels** for short. Each pixel represents just **one** colour. Different images can store different numbers of colours. A simple black and white image only needs to store two colours, black and white, and so an image could store the colours as a simple binary sequence.

##### Example

The following binary sequence could be used to store this simple picture of an umbrella



Shade in the image using the data below.

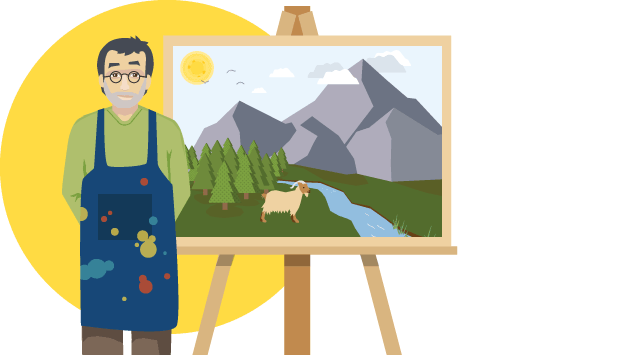
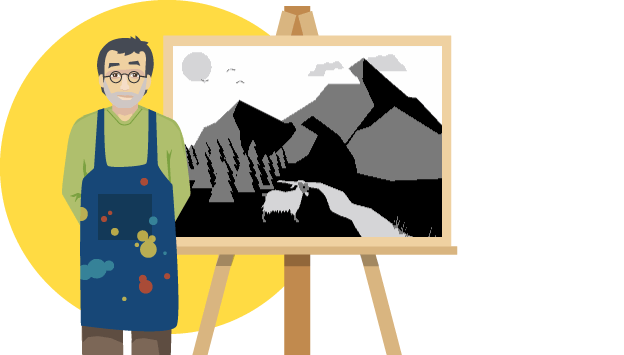
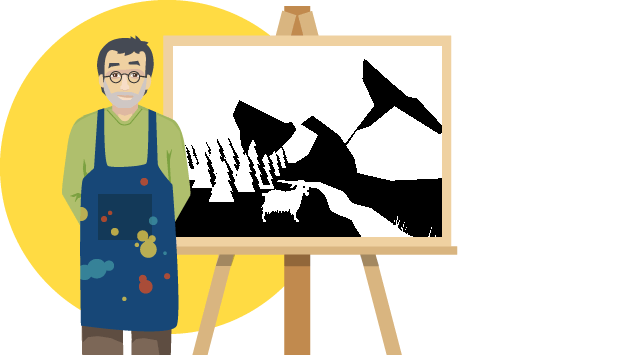
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |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

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

DATA IMAGE

##### COLOUR DEPTH

In the example on the previous page a 1 represents black and a 0 represents white. If we want more colours then we need to use more data to represent more colours. If we use more bits to represent each colour then we can have more colours available to use. This is known as an image’s **colour depth**.



1 bit (2 colours) 2 bit (4 colours) 4 bit (16 colours)

To display photographic images on the screen, **3 bytes** (24 bits) are used to represent the colour of each pixel. This is split up into one byte for each of the **primary colours** of light, red, green and blue.

This is because computer displays uses these primary colours of light to display all the different colours we see on the screen. By mixing the three colours with different intensities the display is able to show over **16 million** different colours. The amount of red, green and blue is usually stored as an 8 bit value, giving 256 levels of brightness for each colour (zero representing no colour and 255 representing the maximum brightness of that colour). So to represent bright yellow the values 255, 255, 0 can be used to represent the brightness of red, green and blue.

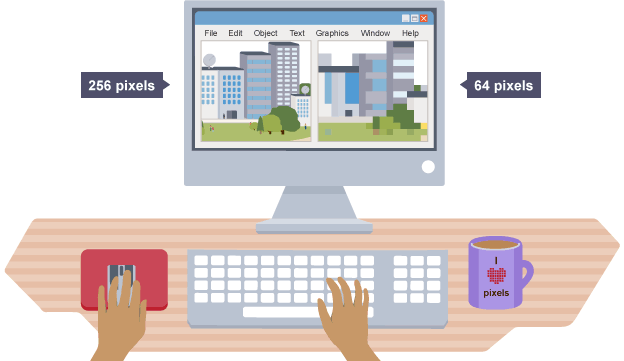
##### Complete the table

|  |  |
| --- | --- |
| **No of Bits per pixel** | **No of possible colours** |
| 1 |  |
| 2 |  |
| 4 |  |
| 8 |  |
| 24 |  |

##### IMAGE RESOLUTION

Resolution is a measure of pixel density, usually measured in dots per inch (dpi). **Images** on **websites** usually have a resolution of **72 dpi**. This means that a 1-inch square contains a grid of pixels that is 72 pixels wide by 72 pixels high. 72 x 72 = 5184 pixels per square inch.

High quality **printed images** in books and magazines have a **higher** **resolution** than computer screens. Magazines often use either 300 dpi or even 600 dpi.



Fill in the table below to show the difference between high and low resolution images. Think about **image** **quality** and **file size**

|  |  |
| --- | --- |
| **High Resolution** | **Low Resolution** |
|  |  |

##### METADATA

Image files usually also contain **metadata**. Metadata means **'data** **about** **data'** and provides information about the image and instructions for how to display it correctly.

When you take a photo on a smartphone, what other information is recorded?

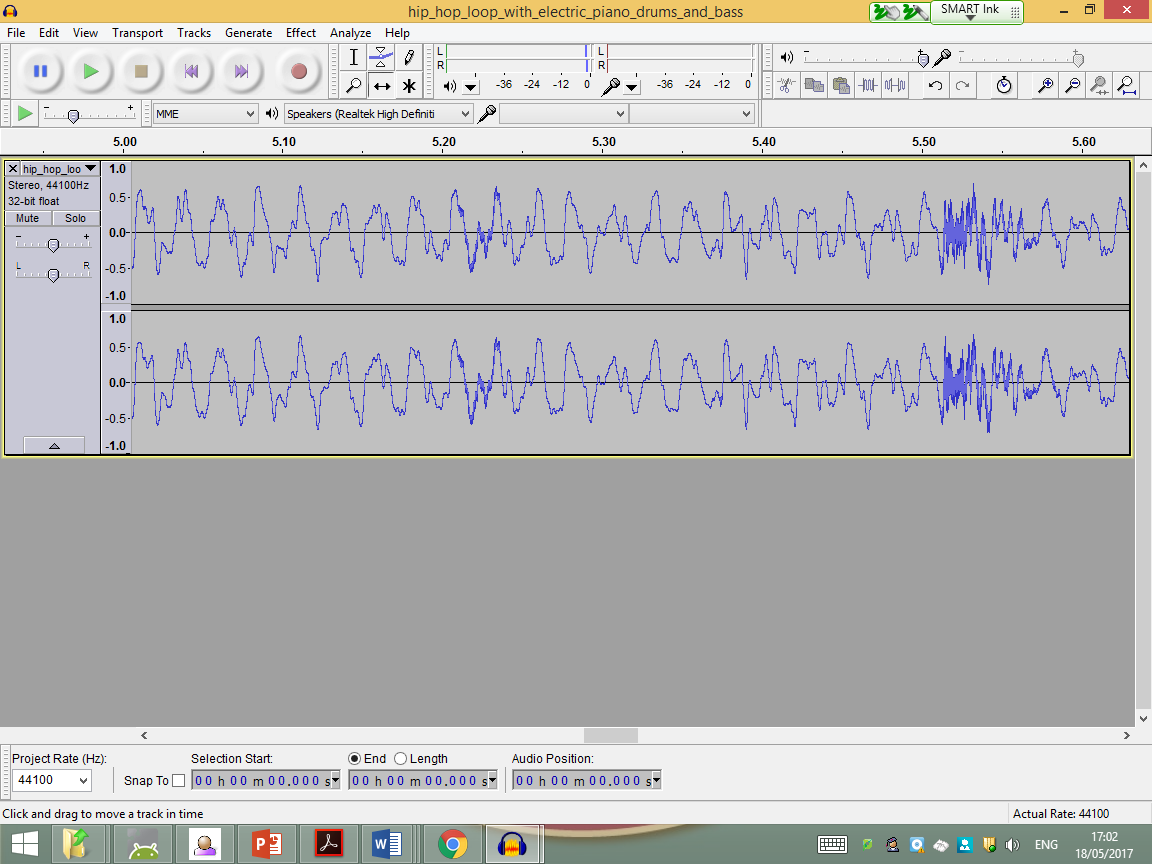
.

Metadata stored when taking a photo?

##### SOUND

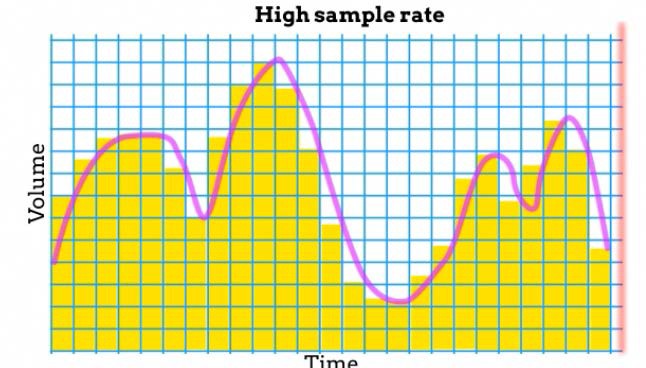
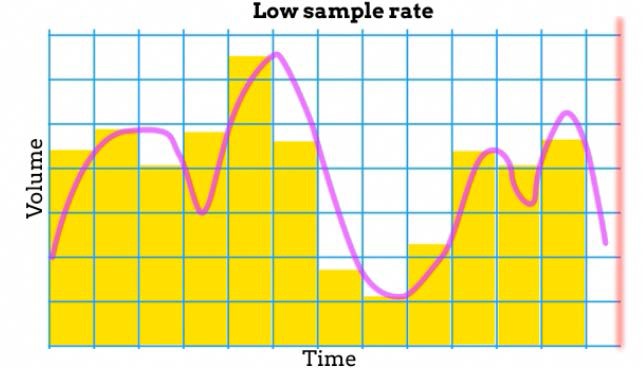
##### Analogue Sound

Sounds are **analogue** signals. This means that the sound wave has a range of values. Look at the sound wave below. The height of the wave (**amplitude**) is the **volume** of the sound. The distance between two peaks is known as the **frequency** and corresponds to the **pitch** of the sound.



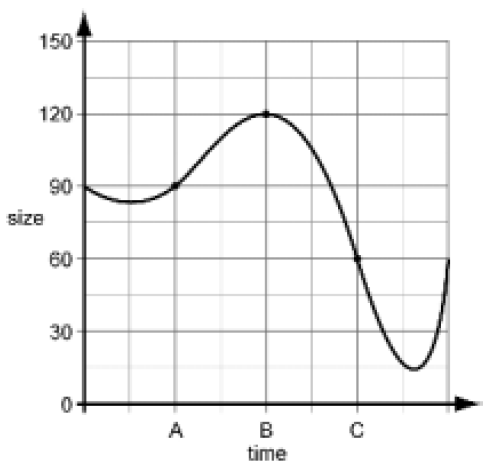
##### Digital Conversion

The sound wave needs to be converted to into numbers that can be stored in binary. To convert an analogue sound into a digital sound, the computer samples the height of the sound wave a number of times a second.



The number of samples in one second is known as the **sample** **rate**. This is measured in **Hertz** (samples per second) or more commonly **Kilohertz** (1000 Hz). Sampling is represented by the yellow bars. Notice that this does not follow the pink analogue line perfectly. The **more** times a sound wave is sampled in one second the closer to the original sound.

By increasing the number of bits used to store each sample, the amount of detail contained in each sample increases. Using more bits enables the sound to be more accurately represented.

An artist is recording sound using a computer. The graph below represents the pressure wave of the sound being recorded.

At point A on the graph, the height of the sound wave is 90. This is stored digitally using the binary value of 0101 1010 (or 5A in Hex). Complete the table below to show how points B and C are stored:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Point A | Point B | Point C |
| Size | 90 |  |  |
| Binary value | 01011010 |  |  |
| Hex Value | 5A |  |  |

Explain how changing the sample rate and bit depth of an MP3 track affect the quality of playback?

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

##### COMPRESSION

On Facebook more than 200 million images are uploaded every day. On Snapchat, almost 100,000 are uploaded each minute. It is estimated that 1.8 billion images are uploaded to social media sites each day. Millions of audio files and video files are also shared over the internet each day too. Therefore, it is in everyone’s interests to make these files as **small** as possible but try and keep the quality so they are still **usable**. This is why file compression is important.

The benefits of file compression are…

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

Compression algorithms are used to ensure that the file size is as small as possible but with no great loss to the quality of the file. If the compressed file can be restored to its original without any loss of data, it is called **lossless compression**. However, if the file is compressed by removing some data so that the original cannot be restored, it is known as **lossy** **compression**.

Describe how each compression technique works below

|  |  |
| --- | --- |
| **Lossless Compression** | **Lossy Compression** |
|  |  |