Lecture 0 - Scratch

Binary

What is computer Science?

Programming is a big part of computer science, where you write code to express ideas and solve actual problems involving data, but computer science itself is the study of information.

What does it mean to solve a problem?

Problem solving is where you get some input (which represents the problem you want to solve) and then there will be an output (which represents your goal which is to solve the problem) and then there will be a secret recipe in the middle which will help you solve the problem.

What is Unary and Binary?

Unary is a type of system which would help us solve problems which include numbers (for ex. taking attendance at the beginning of class). In other words, unary is a very simple system of using a single symbol to solve some problems. Unary, in mathematical and technical terms is called "Base-1" meaning that the number that you are operating with has 1 digit in it.

Computers use a language other than Unary, this language is called Binary. Bi means 2, so computers have 2 digits which they would use to communicate with or solve problems within.

The technical term bit comes from the word binary digit. A bit is 0 & 1. So in simple words, us humans, in our everyday life use the Unary system to solve problems, meaning that we use the digits from 0 to 9. On the other hand computers use the Binary system to solve problems, meaning that they use the digits 0 or 1.

How do computers only speak in Binary?

The simplest way in which we could understand this by taking the light bulb as an example. So if humans were a computer, to represent the number 0 you would keep the light switch off & to represent the number 1 you would switch the light bulb on.

Now, this is relevant to computers because inside a computer there are thousands, even millions of tiny switches (we could metaphorically think of these switches as light bulbs). These switches are called transistors.

Just like a light bulb, these transistors could be switched on to represent 1 and switched off to represent 0.

Obviously, with this ability of computers they could count 0 to 1 but it turns out that they could count even higher with a little bit more electricity.

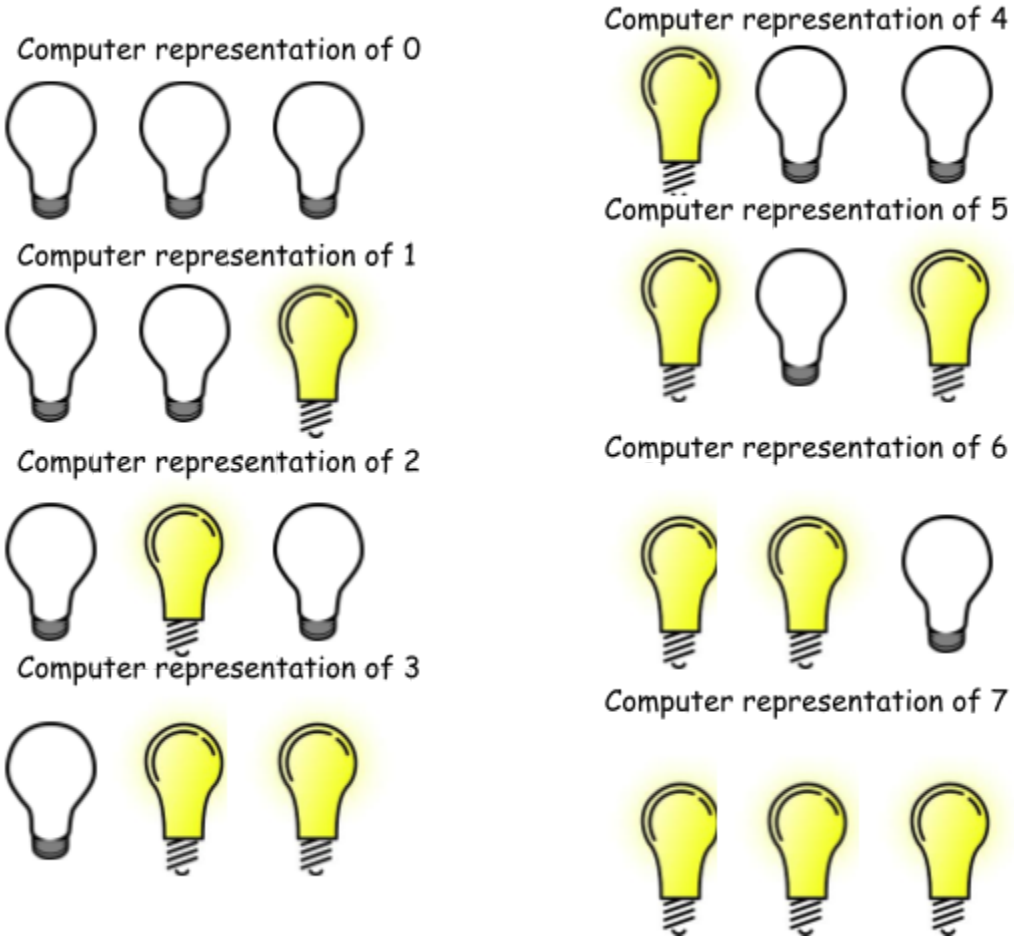
Based on the above concept, when we take a singular light bulb or a transistor and switch it on, it represents 1 and when we switch it off, it represents 0. So to represent more numbers we would need to use more light bulbs.

To understand this concept better, let us take 3 light bulbs which are all switched off. We know that when a light bulb is turned off it represents 0.

So we would think that with these 3 light bulbs (when they are switched on) could represent 0, 1, 2, 3, but when we consider the combination of bulbs being on and off the computer might be able to represent more than 0, 1, 2, 3.

For example we could say that the computer stores the number 0 when all 3 light bulbs are off. When the first light bulb is on but the other two are off, this could be how the computer stores the number 1. Similarly, when the second light bulb is turned on and the other two are off, this could be how the computer stores the number 2.

Following the above example, with 3 light bulbs the computer can store the number 0, 1, 2, 3, 4, 5, 6 & 7.



What we have to understand from this example is that the way the computer stores these number values is by memorising the patterns in which these transistors or light bulbs turn on or off.

We cannot represent more than 0, 1, 2, 3, 4, 5, 6, 7 with 3 light bulbs because the pattern or the combinations in which all three of these lights could uniquely be arranged is only in 8 ways. Thus, we can only represent 8 numbers. The same method could be used for representing numbers with multiple light bulbs.

In the above example, the computer used base 2 (Binary) to represent numbers 0 through 7. Binary, just like the decimal system that we use, follows certain types of rules.

Let's recall back to how decimal place values work,

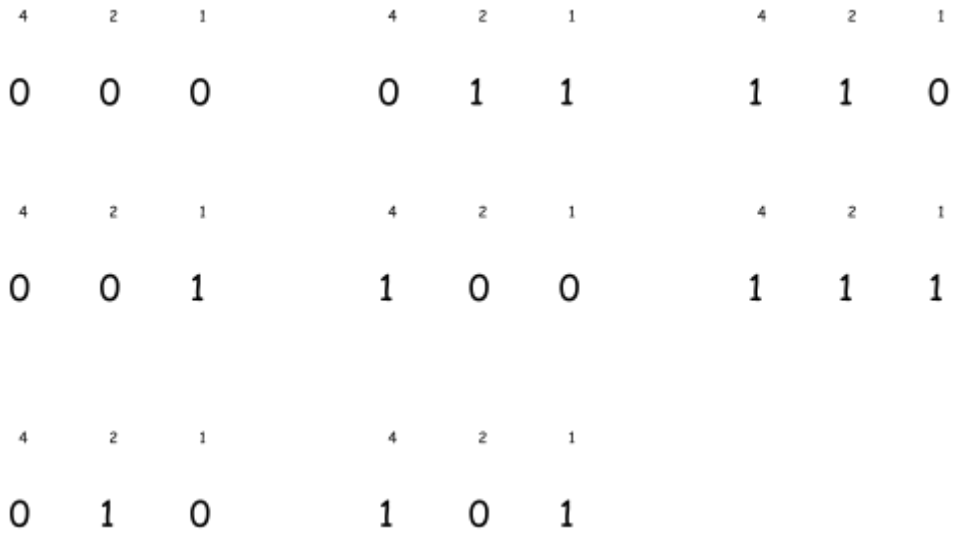
=> 123

When we look at the above number "123" our mind reads it as "one hundred and twenty three". This is because we know that the rightmost number (3) is in the 1's place, the middle digit (2) is in the 10's place and the leftmost digit (1) is in the 100's place. Now we just do quick mental maths, which would lead to "100 x 1 + 10 x 2 + 1 x 3" which is "100 + 20 + 3" gives us the number we know as "123".

We could also use base terminology to represent these places. For example the 1's place is 10^0, the 10's place is 10^1, the 100's place is 10^2 and so forth.

Now, if we take computers, they obviously only have access to electricity on or off (0 or 1). So they tend to use a different base. 2^0, 2^1, 2^2 and so forth, a.k.a 1's place, 2's place, 4's place and if we keep going 8's place,

16's place and so on. At the end of the day, the idea for both the decimal system and the binary system is fundamentally the same.



Finally, this is the computer representation of the above diagram of light bulbs into numbers.

Here, as we could see
=> "off off off" represents 0 because "4 x 0 + 2 x 0 + 1 x 0" is 0
=> "off off on" represents 1 because "4 x 0 + 2 x 0 + 1 x 1" is 1
=> "off on off" represents 2 because "4 x 0 + 2 x 1 + 1 x 0" is 2, and so on

Now if the computer wanted to represent 8, it would just move to the 8's place (another transistor) just like how we move to the thousands place when we want to represent a number in the thousand's column.

Essentially at the end of the day the binary system used by computers and the decimal system used by humans are similar but instead of base 10 for decimal system, we use base 2 for binary system and only the numbers 0 and 1 are used.

Now, singular "Bits" are not that useful as they cannot be used to store memory. This is why "Bytes" is a more useful measurement unit to use when talking about data or memory. Generally, there are 8 "Bits" in a Byte.

For example, when a picture or a video is downloaded, the memory that they take up is measured in a larger number of Bytes such as Kilobytes, Megabytes or even Gigabytes. This just means that there are lots of combinations of 8 Bits in the computer's hard drive.

**Fun Fact: A Computer can count up to 256 with 8 Bits or 1 Byte **

Representation

How do Computers represent letters?

Obviously, we use computers for more than just mathematics. They are used to send messages or texts or emails and so on. At the end of the day, every device has millions of transistors that they use in units of Bytes (8 Bits) to represent numbers using various combinations and patterns of 0's and 1's.

"So then how might a computer represent the alphabets?"

We could possibly assign a number to every letter. That would lead to (for simplicity purposes) A represented by 0, B 1, C 2 and so on.

Now how would we represent 0 or 1 or 2?

How might we be able to differentiate between the numbers representing the letters and the numbers representing numbers themselves?

Before we think about the solution to that, computers, phones etc actually represent the letter "A" by the number 65 (the numbers 0 to 64 have been used to represent special and control characters). It turns out that the letter "B" is going to be 66, "C" being 67 and so forth.

[Example:

Representation of "A" by computers etc - 01000001

(1 being in the 1's place and another 1 being in the 64's place)]

In the above example, we can visually see how the computer represents the letter A.

So now how do we represent all of the other letters?

It turns out that Americans, years ago, came up with ASCII (American Standard Code for Information Interchange). ASCII is an acronym that describes what is being explained above. Representing letters (A, B, C …) with binary numbers starting from 65.

ASCII value	Character	Control character	ASCII value	Character	ASCII value	Character	ASCII value	Character
000	(null)	NUL	032	(space)	064	@	096	
001	☺	SOH	033	!	065	A	097	a
002	☻	STX	034	"	066	B	098	b
003	♥	ETX	035	#	067	C	099	c
004	♦	EOT	036	$	068	D	100	d
005	♣	ENQ	037	%	069	E	101	e
006	♠	ACK	038	&	070	F	102	f
007	(beep)	BEL	039	'	071	G	103	g
008	◘	BS	040	(	072	H	104	h
009	(tab)	HT	041	)	073	I	105	i
010	(line feed)	LF	042	*	074	J	106	j
011	(home)	VT	043	+	075	K	107	k
012	(form feed)	FT	044	,	076	L	108	l
013	(carriage return)	CR	045	-	077	M	109	m
014	♫	SO	046	.	078	N	110	n
015	☼	SI	047	/	079	O	111	o
016	►	DLE	048	0	080	P	112	p
017	◄	DC1	049	1	081	Q	113	q
018	↕	DC2	050	2	082	R	114	r
019	‼	DC3	051	3	083	S	115	s
020	¶	DC4	052	4	084	T	116	t
021	§	NAK	053	5	085	U	117	u
022	▬	SYN	054	6	086	V	118	v
023	↨	ETB	055	7	087	W	119	w
024	↑	CAN	056	8	088	X	120	x
025	↓	EM	057	9	089	Y	121	y
026	→	SUB	058	:	090	Z	122	z
027	←	ESC	059	;	091	[	123	{
028	(cursor right)	FS	060	<	092	\	124	|
029	(cursor left)	GS	061	=	093	]	125	}
030	(cursor up)	RS	062	>	094	^	126	~
031	(cursor down)	US	063	?	095	_	127	⌂

Now whenever you receive a text or a message you could see under the hood what patterns on 0's and 1's were sent.

[Example:

Text Message you received in Binary Code -

01001000 01001001 00100001

72 73 33]

In the above example we could see that the text message received is "HI!". 72 representing "H", 73 representing "I" and 33 representing "!" (sometimes you would need a cheat sheet like the one above to know the binary code for certain characters).

Now back to the original question, if 65 represents A, then how would you represent 65 the number or 66 the number?

When the chart above is clearly observed, we could see that there are ***other*** numbers that represent numbers 0 through 9.

048	0
049	1
050	2
051	3
052	4
053	5
054	6
055	7
056	8
057	9

As we can see above, if you hit "5" on your computer, the computer does not store 5, it would store 53.

How do Computers represent other Characters?

In the above ASCII chart, one can clearly notice that the letters used are of the English language. This is a problem as the ASCII chart does not include characters from other languages and we might want to send messages in a language other than English, this could include accented or Asian characters/alphabets.

If we recall back, ASCII uses a byte (7 bits to be specific) to represent English letters. Meaning that ASCII could represent up to 256 total characters (256 based on the possibilities of pattrents formed using 8 bits). This is obviously not enough characters to represent all of the letters in every language in this world.

Therefore, to represent all of these various characters including Asian characters, we would just have to add another byte to create more patterns. So instead of using 1 byte, we could use 2 or 3 or even 4 bytes to represent all of these characters.

The solution then to ASCII, is Unicode which is also a mapping of numbers and letters just like ASCII but in many different languages. People from various countries, companies and cultures came together to agree on the specific Binary representation of each character of each and all languages of humans.

"Now where do emojis come in?"

If we start using 32 bits, which is about 4 bytes, we could represent as many as 4 billion patterns which is way more than we need to represent

all human languages in the world. Now, we could utilise these patterns to represent more playful things such as emojis.

[Example:

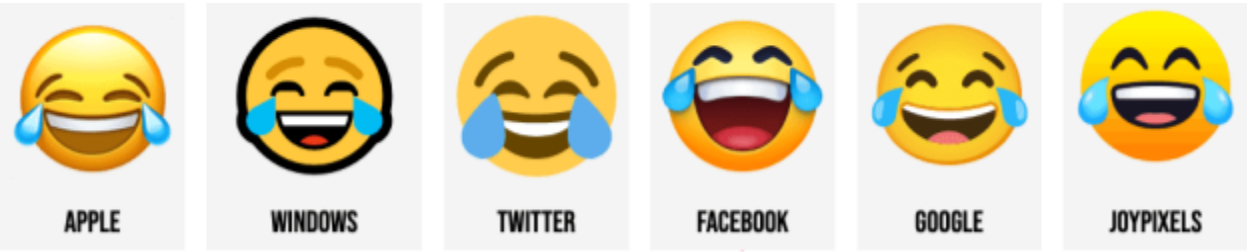
Emoji you received in Binary Code -

11110000100111111001100010000010

😂]

The above is the Binary code that the people of Unicode decided to represent the popular emoji "Tears of Joy".

Now sometimes we notice that the same emojis look different on IOS or Android or Windows or other softwares. Even though we look at these emojis as images they're really characters and IOS, Android and other softwares just have different fonts just like how there are different fonts we use in the English language. So that same pattern of 0's and 1's might just slightly be different.



Tears of Joy emoji representation in different softwares.

Every year, the people at Unicode add more emojis and other characters.

How do Computers represent Emoji customisation?

As discussed above, emojis are commonly used text messages. Different emojis would be trending at different times.

Although a lot of the emojis are the default Yellow colour, some emojis have the option to pick specific skin tones. If it's supported by the software and Unicode we could actually interact with the emoji and customise it which modifies the display.

Now, if a specific emoji has 5 different skin tones, it doesn't make sense to have five different identical patterns of 0's and 1's except for the change in skin tones because it's just repeated code which takes up more bits than actually required.

"Now how then, did companies like Apple and Google implement support for emoji skin tones without wasting data?"

The way Unicode solves this problem is the first byte or bytes (patterns of 0's and 1's) that you would receive in text just represents the structure of the emoji, the default Yellow version, then if the text is immediately followed by a certain reusable pattern of bits which are standardised to represent each of the skin tones, the phone, computer, Mac will change the default colour into the apt human skin tone.

This does not take up that much data or information because we are only using twice as many bits and not five times as many bits.

Again, there are some emojis these days, which are combinations of 2 or more emojis.

[Example:

One Emoji made with combinations of 2 emojis

Example 1: 

Example 2: ]

These above emojis, are examples of a single emoji which is made of 3 or 2 other emojis, the bike; the person; the mountains.

Now, the way computers represent this emoji (example 1) is that they have a specific pattern of 0's and 1's for "the bike" emoji, another pattern for "the person" and finally a pattern for "the mountains". At the end, to assemble this emoji together the computer brings these different patterns of 0's and 1's together.

Obviously, if the computer wants to represent the other emoji (example 2), it would just reuse the same patterns of 0's and 1's for "the bike" and "the person" instead of creating a new pattern every time.