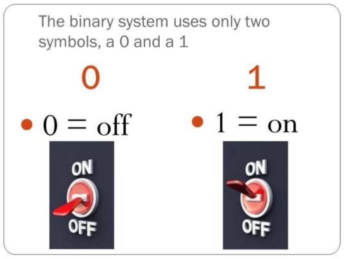
Binary – the programming language that every computer has to know!

1. At the **physical** level, computers are just a bunch of electrical circuits!

2. The only physical thing that is happening inside a computer is the **storage** and **organization** of large numbers of **on** and **off** signals. Crazy, right?

3. Each **on/off** signal is created by tiny switches that create areas of **low voltage** (OFF) or **high voltage** (ON) in a particular spot in the circuit.

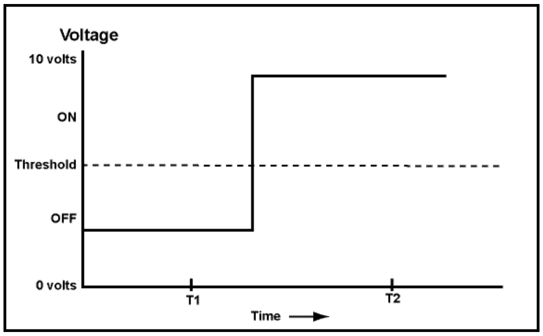
4. The “switches” are called: **transistors**.



The binary system uses only two symbols, a 0 and a 1

0 1

- 0 = off
- 1 = on



Transistor

The Binary Number System

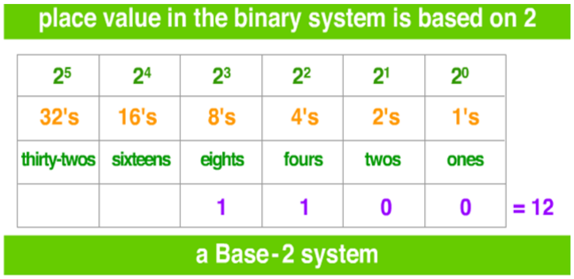
- The number system you are most familiar with (the **decimal** number system) has 10 digits 0,1,2,3,4,5,6,7,8,9. This is difficult for a computer represent with a bunch of **on/off** switches.

- The easiest way to represent and store numerical information using only two digits the **Binary** number system. Binary can represent any number we want using only 2 digits. “1” for **ON** and “0” for **OFF**! Perfect!

- With **binary** you can represent **any number** with just 0’s and 1’s (wow)!

- It’s like our familiar *base 10* system but the columns **don’t** go: 0’s, 10’s, 100’s, 1000’s. They go: 1’s 2’s 4’s 8’s 16’s…..

See if the following chart can help you:

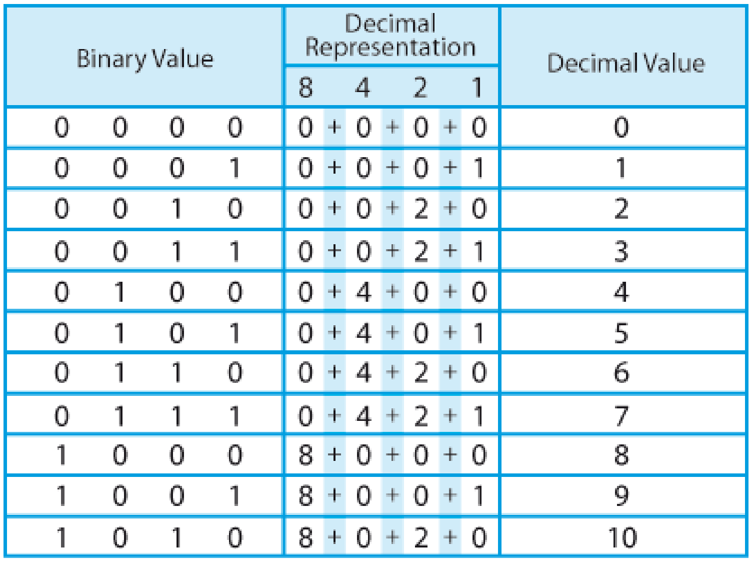


place value in the binary system is based on 2

2^5	2^4	2^3	2^2	2^1	2^0	
32's	16's	8's	4's	2's	1's	
thirty-twos	sixteens	eights	fours	twos	ones	
		1	1	0	0	= 12

a Base-2 system

Some more examples:

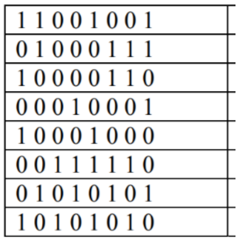


Binary Value	Decimal Representation				Decimal Value
	8	4	2	1	
0 0 0 0	0	+ 0	+ 0	+ 0	0
0 0 0 1	0	+ 0	+ 0	+ 1	1
0 0 1 0	0	+ 0	+ 2	+ 0	2
0 0 1 1	0	+ 0	+ 2	+ 1	3
0 1 0 0	0	+ 4	+ 0	+ 0	4
0 1 0 1	0	+ 4	+ 0	+ 1	5
0 1 1 0	0	+ 4	+ 2	+ 0	6
0 1 1 1	0	+ 4	+ 2	+ 1	7
1 0 0 0	8	+ 0	+ 0	+ 0	8
1 0 0 1	8	+ 0	+ 0	+ 1	9
1 0 1 0	8	+ 0	+ 2	+ 0	10

You Try:

Work with a partner (or on your own).

Find the **decimal number** represented by the **binary numbers below:**



1 1 0 0 1 0 0 1	
0 1 0 0 0 1 1 1	
1 0 0 0 0 1 1 0	
0 0 0 1 0 0 0 1	
1 0 0 0 1 0 0 0	
0 0 1 1 1 1 1 0	
0 1 0 1 0 1 0 1	
1 0 1 0 1 0 1 0	

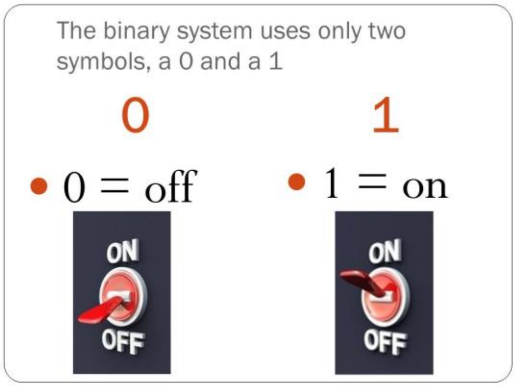
Answers on the next page.

Answers:

201
71
134
17
136
62
85
170

Memory: Bits and Bytes Storing 0’s and 1’s

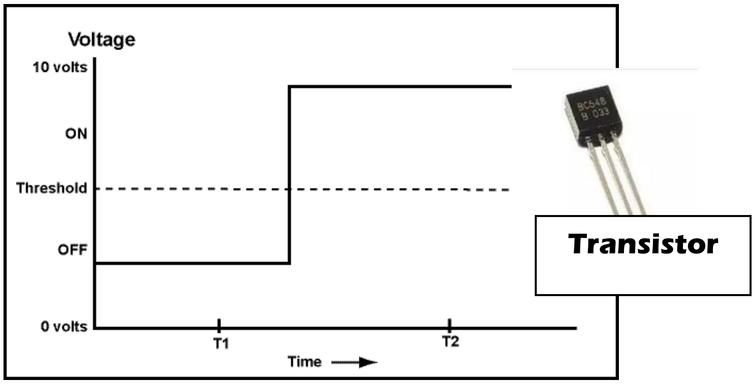
Again, at the physical level, computer memory consists of a large number of **on** and **off** signals that can be stored using areas of high and low voltages. **One** of these switches/signals is called: **one bit.**


The binary system uses only two symbols, a 0 and a 1

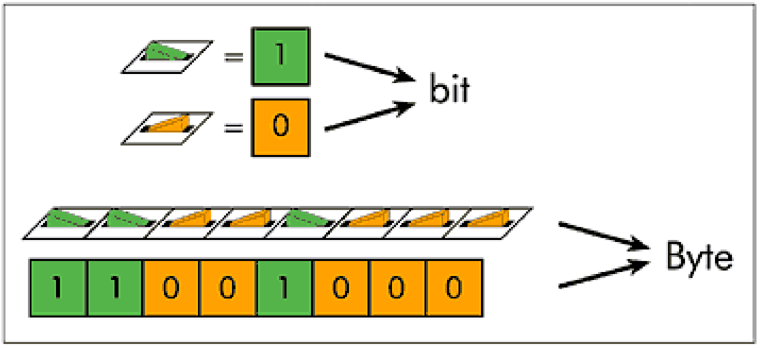
0

1

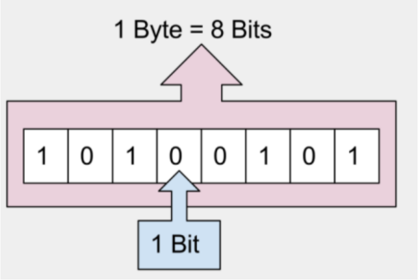
- 0 = off
- 1 = on

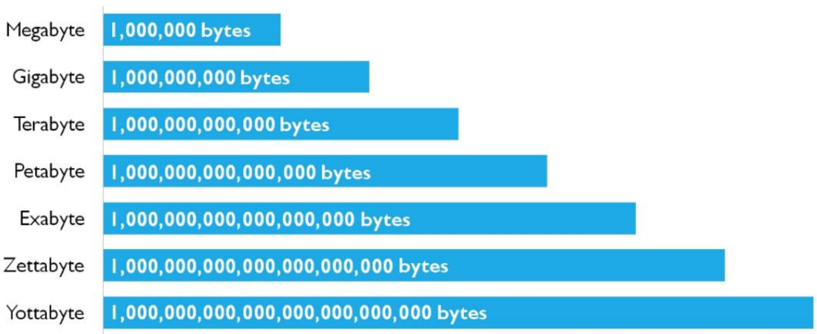


A bit is the name of **1 on/off** signal.

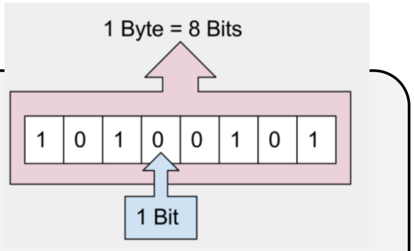


A **Byte** is 8 Bits. (very important!) (bits are usually grouped into **BYTES**)





A **Gigabyte** is **1 billion bytes** (that's **8 billion** of these **on/off switches**)

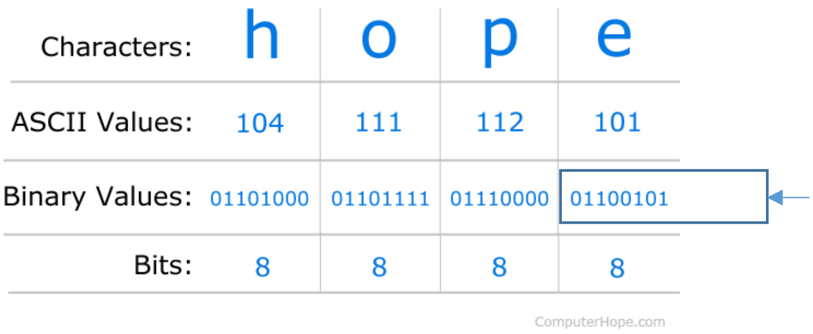


To store information, computers **always** group bits into **BYTES**.

In a computer, **bits** are usually grouped in packs of 8 bits or "switches" (called **one byte**). **Using Binary** One **Byte** can represent 256 different decimal numbers (0-255).

This means a single **Byte** we can have 256 different unique patterns/combinations of **on/off** signals to represent 256 different numbers.... **or anything** else we want.

Examples of different **Bytes** used to represent **letters**:



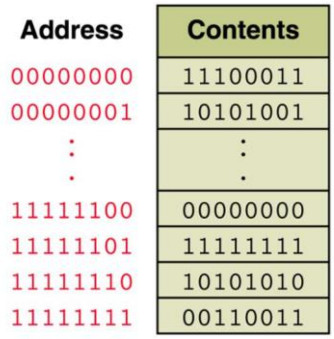
Characters:	h	o	p	e
ASCII Values:	104	111	112	101
Binary Values:	01101000	01101111	01110000	01100101
Bits:	8	8	8	8

ComputerHope.com

1 **Byte** (8 bits)

Memory Addresses:

It is very important to also note that each **Byte** has an actual **physical location** in a computer's memory and can be given a **unique address**. We can think of all of our computer's **memory as just one giant group** of **bytes** that we can read and write.



Address	Contents
00000000	11100011
00000001	10101001
: .	: .
11111100	00000000
11111101	11111111
11111110	10101010
11111111	00110011

Each **Byte** can be thought to have an actual physical **location!**

More Than Just Numbers? ASCII

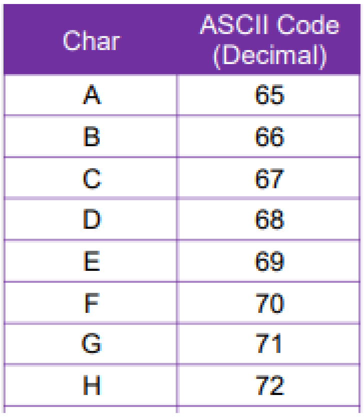
Great! So now we understand that computers can store 0's and 1's and combinations of those 0's and 1's can represent any number we want.

The main way we communicate with a computer is through a keyboard so we need a way for turn a key stroke into a number and then 1's and 0's so it can be stored.

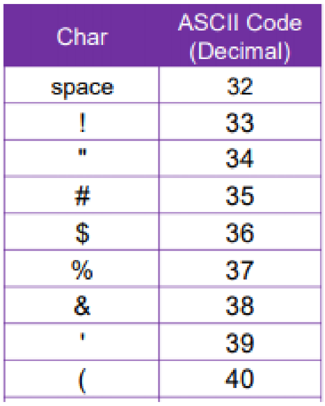
This is done through: **ASCII** Code:

The ASCII code (Pronounced ask-ee) is a code for representing English characters as numbers, with each character assigned a number from 0 to 256. For example, the ASCII code for uppercase A is 65.

Esc F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 Prt Sc Sys Rq Delete Scr Lk Insert Num Lk Pause Break
¬ ` | ! 1 " 2 £ 3 $ 4 % 5 ^ 6 & 7 * 8 (9) 0 _ - + = Home
Q W E R T Y U I O P { [}] PgUp
Caps Lock A S D F G H J K L : ; @ ' ~ # PgDn
| \ Z X C V B N M < , > . ? / End
Ctrl Fn Alt Alt Gr Ctrl



Char	ASCII Code (Decimal)
A	65
B	66
C	67
D	68
E	69
F	70
G	71
H	72



Char	ASCII Code (Decimal)
space	32
!	33
"	34
#	35
$	36
%	37
&	38
'	39
(	40

Full ASCII Table here:

https://www.101computing.net/wp/wp-content/uploads/ASCII-Table.pdf

Challenge#1

Use the function **`bin()`** and **`int()`** functions in python to create a quiz game that asks the user to convert binary numbers to decimal and vice versa. The game should use **byte** sized binary numbers only (eight digits). The max decimal numbers should be involved in the game should be 256.

Make the game as excited as possible. You should include:

- An engaging interface with the user.
- A points system
- An end to the game and option to restart.

Make sure you have a classmate play the game!

Challenge#2

chr() and ord() functions:

Using Python you can get the ASCII values of a character using the **`ord()`** function and character the values of an ASCII using **`chr().`**

For instance:

`ord("A")` returns 65
`chr(65)` returns "A".

Using the functions above and the bin() and int() functions, create a secret message game where the user is instructed to **decode a series of 0's and 1's to gain access to the secret message.**

Assume the user **has** a **Decimal to ASCII table** but **not** a **binary to decimal** table.
Their job (and the intention of the exercise) is to practice converting binary to decimal.

Your game should have a bank of several short messages that can be selected randomly and presented as challenges to the user each time they play. The game should tell the user if they got the answer correct and ask them if they want to play again.

Example:

```
01001101 01110010 00101110 00100000 01010111 01100001 01101100 01111010 01101100
00100000 01101001 01110011 00100000 01100011 01101111 01101111 01101100 00001101
00001010

Answer:

Mr. Walzl is cool
```