Binary
Base-2 number system. (The **radix** of binary is 2)
Computers are limited in their ability to represent things, so on/off, high/low, 0/1 is much more straightforward.
There are only 0s and 1s in Binary, but using these two symbols you can create all the same numbers as the decimal system can.
*The farthest right digit is always the least significant (least weight on number)

Binary -> Decimal

0=0
1=1
10=2
11=3
100=4
101=5
110=6
111=7
1000=8
1001=9
1010=10

Exponentiating the number 2 to a power is how you calculate binary with decimal
Can divide by 2 and find remainders to calculate binary (read bottom to top for left to right)
The **length** of a binary number is the amount of 1s and 0s it has

each 1 or 0 is called a **bit**
a group of 4 bits is called a **nibble**
8-bits, or 2 nibbles make a **byte**
word can mean longer bits (16+ long)

Leading zeros can add information about whether you are working with nibbles or bytes. But don't change the value
In addition to basic mathematical manipulation like addition, subtraction, etc, you can use the bitwise **operators** to perform functions bit-by-bit.

1) Complement (**NOT**) means finding the exact opposite of everything (switching 1s and 0s) and is the only bitwise operator that operates on a *single* binary value
2) **OR** takes two numbers and produces the union of them. This operator has four possible outcomes:

1. 0 OR 0 = 0
2. 1 OR 1 = 1
3. 0 OR 1 = 1

4. 1 OR 0 = 1

OR is like addition. 0 + 0 = 0, but 1 + anything will always = 1

3) **AND** takes two binaries and produces the conjunction of them. If either or both bits are 0 then the resulting bit is 0. If both values are 1, the resulting bit is also 1:

1. 0 AND 0 = 0
2. 0 AND 1 = 0
3. 1 AND 0 = 0
4. 1 AND 1 = 1

AND is like multiplication

4) **XOR** is an exclusive OR because it behaves the same but will only produce a 1 if either one or the other number has a 1 in the position:

1. 0 XOR 0 = 0
2. 0 XOR 1 = 1
3. 1 XOR 0 = 1
4. 1 XOR 1 = 0 (both are 1 in this case so it doesn't work!)

Bit Shifts are handy for manipulating a single binary value. Each shift has two components: **direction** and **amount** of bits that are shifted.

Right: one or more of the least significant digits on the far right will get cut off. Adding leading zeroes will keep the bit-length the same (same as dividing by 2^n)
Left: all of the bits are pushed towards the most significant digits on the far left, and for each shift a 0 must be added to the least significant position on the far right. (same as multiplying by 2^n)

Binary is what drives all electronics!

In most programming languages, binary numbers are represented by a **0b** preceding the number, without this it would just be a decimal number.

Bitwise operators can be used in programming:
& = AND
| = OR
~ = NOT
^ = XOR
Left shift: <<n
Right shift: >>n