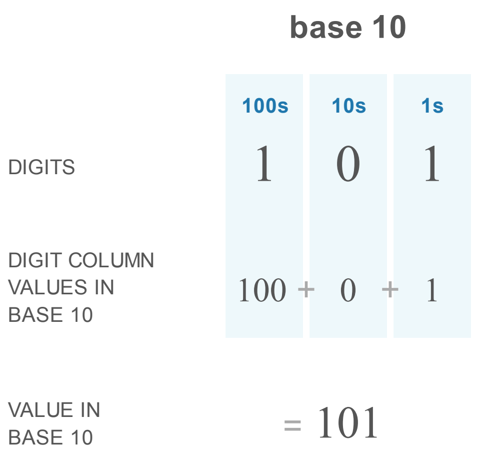
# Binary Numbers

When we use numbers, we usually use **decimal numbers** (or **base-10**), which are expressed using 10 values, $0 - 9$.

So our digit columns increase by 10 times (1s, 10s, 100s). For example, let's take the digits 101:

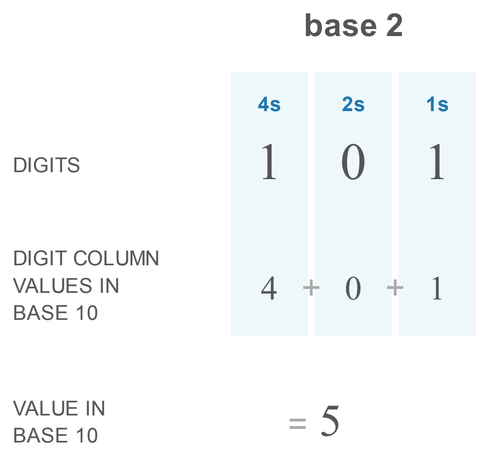


But using 10 values is arbitrary. We could mulitply our columns by *any* number and numbers would still work. Some languages spoken in Nigeria and India use **duodecimal** numbers, or **base-12**. So "eleven" and "twelve" aren't built using 1s and 2s, they're entirely different digits.

Some mathematicians argue that base-12 is a better system than our base-10, because 12 has more factors (1, 2, 3, 4, 6) than 10 does (1, 2, 5). We probably use decimal numbers because we have 10 fingers.

**Binary numbers** (or **base-2**) only use two values, 0 and 1. So binary digit columns increase by 2 times (1s, 2s, 4s).

Let's look at the same digits 101:



Binary numbers are nice for computers because they can easily be expressed as series of bits, which only have two states (think of them as "on" or "off", "open" or "closed", or 0 or 1).

Here are the base-10 numbers 0 through 10 in binary:

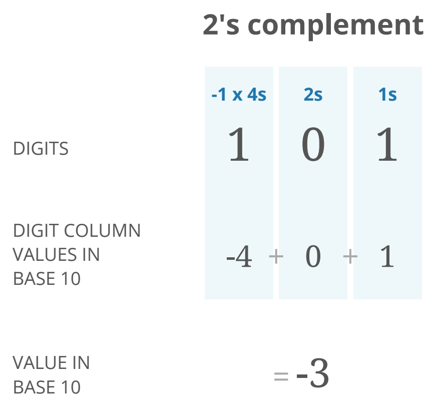
```java
// decimal  binary

   0        0000
   1        0001
   2        0010
   3        0011
   4        0100
   5        0101
   6        0110
   7        0111
   8        1000
   9        1001
  10        1010
```

**Negative numbers** are typically represented in binary using *two's complement* encoding. In two's complement, the leftmost digit is *negative*, and the rest of the digits are positive.

**Warning:** The leftmost digit is *not* the same as a negative sign. The absolute value of the leftmost digit is the same as described above, and each digit's value is double the value of the digit to the right.

To make this clearer, let's look at what happens when we interpret that “101” as two's complement:



> Fun computer systems trivia fact: two's complement isn't the only way negative numbers could be encoded. Other encodings tossed around in the 1960s included “one's complement” and “sign and magnitude” encodings. Of the three encodings, two's complement is the one still used today for a few reasons:
>
> 1. There is only one way to represent zero.
> 2. Basic operations like addition, subtraction, and multiplication are the same regardless of whether the numbers involved are positive or negative.
>
> Since two's complement provided these properties while the other representations didn't, it became the representation of choice and persists decades later.

Here are the base-10 numbers $-5$ through $5$ in two's complement, along with how we'd interpret each bit:

Java

```
// decimal  binary     interpretation


  -5       1011        -8 + 2 + 1
  -4       1100        -8 + 4
  -3       1101        -8 + 4 + 1
  -2       1110        -8 + 4 + 2
  -1       1111        -8 + 4 + 2 + 1
   0       0000        0
   1       0001        1
   2       0010        2
   3       0011        2 + 1
   4       0100        4
   5       0101        4 + 1
```

## See also:

- Bitwise AND (/concept/and)
- Bitwise OR (/concept/or)
- Bitwise NOT (/concept/not)
- Bitwise XOR (eXclusive OR) (/concept/xor)
- Bit Shifting (/concept/bit-shift)

## What's next?

If you're ready to start applying these concepts to some problems, check out our mock coding interview questions (/next).

They mimic a real interview by offering hints when you're stuck or you're missing an optimization.

**Try some questions now ➡**