[A]

JS

1

2

3

Binary

Binary can be intimidating to developers, although there is no need for it to be!

At the core of it, binary is simply a number system using only `0` and `1` as symbols.

As humans, we have our own number system of choice which has 10 symbols: **decimal**. Those 10 symbols are `0,1,2,3,4,5,6,7,8,9`.

Representing Values

How many values can these two number systems represent with **a single character**?

Decimal can represent **10 values** with its symbols `0` through `9`.

Binary can represent **2 values** with its symbols `0` and `1`.

What if we have multiple characters?

Multiple Characters in Decimal

How many values can **decimal** represent with **2 characters**?

Two characters in decimal allows you to count from `00` to `99`. In this range we can represent `100` unique values.

What if we had **3 characters**?

It would be `000` to `999`, representing `1000` possible values.

See a pattern here?

- One decimal character represents **10 values**
- Two decimal characters represents **100 values**
- Three decimal characters represents **1000 values**

The number of values we can represent in decimal is `10 ** n` where `n` is the number of characters!

Multiple Characters in Binary

How many values can **binary** represent with **2 characters**?

Two characters in binary gives us the unique values `00`, `01`, `10` and `11`. That's **four values**!

What about **3 characters**?

With three characters we can represent **eight values**: `000`, `001`, `010`, `011`, `100`, `101`, `110`, `111`.

A new pattern has emerged!

- One binary character represents **2 values**
- Two binary characters represents **4 values**
- Three binary characters represents **8 values**

The number of values we can represent in binary is `2 ** n` where `n` is the number of characters!

> If you've spent much time looking at computer specs, the numbers 256 and 1024 might look particularly familiar to you! These numbers are powers of two: 256 is `2 ** 8` and 1024 is `2 ** 10`. We'll talk about their significance a bit more below.

Counting

Intuitively, we know how to count in decimal. However, putting it into words can be surprisingly challenging!

Count from 8 to 12: `8`, `9`, `10`, `11`, `12`. Quite easy, right? How might you explain this to a **robot** or an **alien**?

> Give it a try for a moment! Design rules that will instruct someone to be able to count infinitely. Try to think generally so that when we reach `99`, the rules will instruct that we go to `100`. Similarly for `999` and `1000`.

Rules for Counting Decimal

For counting a single character, we might say:

1. Here are 10 symbols listed from lowest to highest, separated by a comma: `0,1,2,3,4,5,6,7,8,9`.

2. Start at the lowest symbol (`0`).

3. Count by moving up to the next highest symbol.

4. Repeat **step 3** until we have reached the highest symbol.

Now what happens when we reach the highest symbol: `9` ? We would go to `10` .

How do we explain this rule? It will help to take a moment to talk about **character significance**:

We can think of `9` as `09` where `0` is the **most significant** character and `9` is the **least significant** character. The further left the character is, the more significance it has in our number.

> A good example of this is to think of money. Would you rather have $109 or $901? This number involves the same symbols, except clearly we would want the higher value symbol in a place of **higher significance**.

Alright, so let's get back to counting! When we reach the highest value symbol in the **least significant** place, what do we do next?

A few examples:

- After `09` comes `10` .
- After `19` comes `20` .
- After `29` comes `30` .

What *exactly* is happening in these situations?

We are essentially wrapping around our symbol range in the least significant position and incrementing the next most significant number.

This process, of course, carries over if we reach the highest symbol in our next most significant position:

- After `099` comes `100`
- After `199` comes `200`
- After `299` comes `300`

In these cases, our 2nd most significant character also wraps around the symbol range and we increase the most significant character.

"Alright, alright I've had enough! Why do we need to explain counting so abstractly?"

The reason is, *these same counting rules apply for counting binary!*

Counting Binary

Now that we understand how to count abstractly in a number system we understand intuitively, counting binary should be a cinch!

Let's count **binary**: `0` , `1` , `10` , `11` , `100` , `101` , `110` , `111` , `1000`

Same rules as counting decimal, only our symbol range is shortened to just `0` and `1` !

> For a **robot**, binary is no more complicated than decimal! The same rules apply. Binary is used in classical computers because the input is electric current. The simplest input would be read as `0` (no current) or `1` (current).

Bits, Nibbles and Bytes

Now that we have established binary as a number system. Let's define a few keywords!

- **bit** - a single character in **binary** (a single character in decimal is called a **digit**!)
- **nibble** - a somewhat uncommon term for four bits together (i.e. `1011`)
- **byte** - eight bits together: `1000 1100` would be a **byte**!

The number `256` comes up a lot in computer science. Can you guess why this is?

This is the number of total distinct values we can represent with a **byte**! Remember our formula for distinct values in binary: `2 ** n` . Since a byte has eight bits, the total number of distinct values is `2 ** 8` or `256` !

With `256` distinct values we could choose what we'd like to represent. For example, we could choose to represent all the positive integers we can from `0` through `255` . If we wanted to include negative numbers, we might split the range in half representing from `-128` to `127`

> Depending on the implementation the size of the range may be equal on the negative and positive side, with a dual representation for zero. This would be called One's Complement.

Magnitudes

It is common to name larger numbers in decimal. For instance:

- 1_000 is referred to as a **thousand**
- 1_000_000 is referred to as a **million**
- 1_000_000_000 is referred to as a **billion**

In Binary, there are also names for large magnitudes:

- 1024 bits (`2 ** 10`) is referred to as a ***kilobit**
- 1024 kilobits is referred to as a ***megabit**
- 1024 megabits is referred to as a ***gigabit**

The same rule applies for bytes (i.e. 1024 bytes is a ***kilobyte**).

> In many cases the prefixes **kibi**, **mebi** and **gibi** will be used rather than **kilo**, **mega**, and **giga** respectively. The latter prefixes are used traditionally in the International System of Units to represent magnitudes in power of ten (`10**3` , `10**6` and `10**9`). Traditionally, a **kilobyte** referred to `1024` bytes, however this term is now potentially ambiguous. It may be proper to use the term **kibibyte** when referring to `1024` bytes to ensure there is no confusion.

Wrap Up

We covered quite a **bit** in the lesson!

In the next lesson we'll discuss **hexadecimal**: how to recognize it and convert it to binary. You'll see it quite a lot in crypto systems, so it is quite important to understand!

✓ **Complete**

Next: **Working with Hexadecimal** ›