# What are encodings?

## A beginner's guide to Binary, Hex, Base64, and ASCII

By Rodrigo Espinosa de los Monteros

# What are encodings?

- Different ways to display the same information.
- Encodings specify how bits turn into symbols or text.
- Bits are bits are bits - **bits are not encodings**.
- ASCII is an encoding, it's the encoding that's used to turn bits into symbols that represent (some) human readable languages. Like this one, English.

# Bits are bits are bits

- Computers don't understand text. But they do understand bits. And they can display bits as symbols or text.
- A bit is a single point of information.
- A bit has two states, it can be `off` or it can be `on`.
- Usually if you want to display the value of individual bits you use the Binary encoding.
- If you want to represent two bits, one `off` and one `on`, you can do that with the Binary encoding like this: "01".

# Bits are bits are bits

Going forward in this presentation we will represent bits using the Binary encoding.

- So when you see a Binary string like this: `01000001`
  Remember that means we're talking about 8 bits with the following states:
  `off`, `on`, `off`, `off`, `off`, `off`, `off`, `on`
- 8 bits is 1 byte.
- `01000001` is 1 byte worth of bits, represented in Binary.

> Computers generally don't work with bits individually, instead they work with bytes. Hold that thought...

# Different encodings display bits differently

- This is how you display this byte in different encodings: `01000001`

```
Binary:               01000001
Hexadecimal or Hex:   41
Base64:               QQ
ASCII:                A
```

- In terms of the actual bits your computer is using: "01000001", "41", "QQ", and "A" are all the exact same.
    - They just look different to us because they're using different encodings.

> For Base64 you could also display those bits as "QQ==" but "=" is just a padding character.

# Not all encodings use 8-bits per symbol

- Binary encodes 1 bit at a time
- Hex encodes 4 bits at a time
- Base64 encodes 6 bits at a time
- ASCII encodes 8 bits (1 byte) at a time

> All encoding are displayed in ASCII for human readability
>
> Wait what? Bear with me...

# Not all encodings use 8-bits per symbol (Binary)

The Binary encoding encodes 1 bit per symbol
This is how you encode this byte, `01000001`, into Binary:

1. "01000001"

Encoding bits into binary is really easy because Binary uses 1 bit per symbol.

> A bit can only have two separate states `on` or `off`.
> So the binary encoding only has two symbols in its alphabet, "1" and "0".

# Not all encodings use 8-bits per symbol (Hex)

The Hex encoding encodes 4 bits at a time.

This is how you encode this byte, `01000001`, into Hex:

1. Split the byte into 4-bit chunks: `01000001` = `0100` + `0001`
2. Lookup `0100` in the Hex alphabet.
    i. `0100` -> "4"
3. Lookup `0001` in the Hex alphabet.
    i. `0001` -> "1"
4. Put them together "4" + "1" = "41"

> Hex has 16 symbols in its alphabet
> 0123456789ABCDEF

# Not all encodings use 8-bits per symbol (Base64)

Base64 encoding encodes 6 bits at a time.
This is how you encode this byte, `01000001`, into Base64:

1. Split the byte into 6-bit chunks: `01000001` = `010000` + `01`
    i. Wait, the second part is only 2 bits! We'll just add `0`s to make it 6 bits.
    ii. `01000001` = `010000` + `010000`
2. Lookup `010000` in the Base64 alphabet.
    i. `010000` -> "Q"
3. Lookup `000000` in the Base64 alphabet.
    i. `010000` -> "Q"
4. Put them together "Q" + "Q" = "Q"

> Base64 has 64 symbols in its alphabet
> ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/

# Not all encodings use 8-bits per symbol (Base64 Part II)

> You can gloss over this for now...

We mentioned earlier that you can encode `01000001` into Base64 as "QQ==" but that "=" is just a padding character.

These padding characters are used because, as you noticed in the previous slide, we couldn't split a byte (8 bits) evenly into 6-bit chunks so we ended up padding the last chunk with `0`s until it was 6 bits.

We also mentioned that Base64 encodes 6 bits per symbol. So in Base64, four symbols, "QQ==", corresponds to 24 bits (6 * 4 = 24).

And we want 24 bits because 24 is evenly divisible by both 6 *and* 8.

24 bits is four 6-bit chunks and also three bytes (8 bit chunks).

# Not all encodings use 8-bits per character (ASCII)

ASCII encoding encodes a full byte (8 bits) at a time.
This is how you encode this byte, `01000001`, into ASCII:

1. Look up `01000001` in the ASCII alphabet:
    i. `01000001` -> "0"
2. "0"

Encoding bytes into ASCII is just as easy as encoding bits into Binary because each byte has a one to one mapping to an ASCII symbol.

> The ASCII alphabet has 128 symbols in its alphabet
> The *extended* ASCII alphabet has 256 symbols in its alphabet
> https://www.ascii-code.com/

# Not all encodings use 8-bits per character

## All together now!

Lets encode this byte into all four different encodings:

`01000001`

```
|=======================|
|ASCII  |       A       | <- 8 bits worth of information per symbol
|-----------------------|
|HEX    |   4   |   1   | <- 4 bits worth of information per symbol
|-----------------------|
|BINARY |0|1|0|0|0|0|0|1| <- 1 bit worth of information per symbol
|-----------------------|
|BASE64 |     Q     | Q | <- 6 bits worth of information per symbol
|=======================|    Remember we turn `01` into `000000` for Base64
```

# All encodings are displayed with the ASCII alphabet

Wait what?

- Bits are not encodings, instead *chunks* of bits are encodings.
- Encodings tell us how many bits are in a chunk and also what *symbol* a given chunk represents.

> The most common symbols we use in the English language are the Latin Alphabet, a bunch of punctuation symbols like `.`, `?`, and `,`, and the Decimal Numeral System (the numbers `0` through `9`).

# All encodings are displayed with the ASCII alphabet (continued...)

**Binary**

A chunk is one bit, an `off` bit represents the digit "0", an `on` bit represents "1".

**Hex**

A single `off` bit doesn't represent anything, but four `off` bits represent the symbol "0".

**Base64**

You need 6 bits to represent a symbol, six `off` bits represent the symbol "A".

**ASCII**

You need a byte (8 bits) to represent a symbol, eight `off` bits represent... actually ASCII is weird (the first 32 symbols in ASCII are not text but we won't get into that here).

To represent the first text symbol in ASCII, a space, " ", you use 8 bits that can be represented in Binary like so: `00100000`

# All encodings are displayed with the ASCII alphabet (continued...)

Do you remember that thought you were holding? No? Here it is:

> Computers generally don't work with bits individually, instead they work with bytes.

- A byte is 8 bits.
- A computer can't work with less than 8 bits at a time because it works with bytes.
- Some encodings use less than 8 bits for symbols!
- That's ok. We can just transform the symbols for the different encodings into the same symbols in ASCII to display them.

> Was that confusing? Go to the next slide.

# All encodings are displayed with the ASCII alphabet (Binary)

The Binary encoding encodes 1 bit per symbol
This is how you *display* the byte that is composed of 8 bits with the these states- `off`, `on`, `off`, `off`, `off`, `off`, `off`, `on`:

1. You lookup the symbol for an `off` bit in the Binary alphabet.
    i. An `off` bit maps to the symbol "0".
2. So you lookup the symbol "0" in the ASCII alphabet.
    i. You do this because the ASCII alphabet uses 1 byte per symbol and a computer can't think in chunks that are smaller than 1 byte (8 bits).
3. The ASCII alphabet says the symbol "0" is this sequence of bits: `01000001`
4. You tell the computer to display those bits using the ASCII encoding.
5. The computer displays the symbol "0".
6. You repeat this process for each bit in the sequence above.

# All encodings are displayed with the ASCII alphabet (Hex)

The Hex encoding encodes 4 bits per symbol
This is how you *display* this byte, `01000001`, in Hex:

1. You split the byte into chunks of 4 bits: `01000001` -> `0100` + `0001`.
2. You lookup the symbol for `0100` in the Hex alphabet:
    i. `0100` -> "3"
3. You lookup the byte for the symbol "3" in the ASCII alphabet
    i. "3" -> `00110011`
4. You tell the computer to display the byte `00110011` using the ASCII encoding.
5. You repeat that process for the second 4 bit chunk.

> You can do this for any number of bits. You just split all the bits into chunks of 4 and repeat this process for each chunk.

# All encodings are displayed with the ASCII alphabet (ASCII)

The ASCII encoding encodes 8 bits per symbol

This is how you *display* this byte, `01000001`, in ASCII:

1. You tell the computer to display the bits `01000001` in ASCII.
2. The computers looks up the symbol that corresponds to that sequence of bits
    i. `01000001` -> "A"
3. It displays "A".

# All encodings are displayed with the ASCII alphabet (Base64)

Base64 encoding encodes 6 bits at a time.
This is how you *display* this byte, `01000001`, in Base64:

1. You split up `01000001` into 6-bit chunks: `01000001` -> `010000` + `01`
    i. Pad the second chunk with `0`s -> `010000`
2. Lookup the symbol for `010000` in the Base64 alphabet.
    i. `010000` -> "Q"
3. You lookup the byte for the symbol "Q" in the ASCII alphabet
    i. "Q" -> `01010001`
4. You tell the computer to display that byte as ASCII.
5. You repeat that for the rest of the chunks, padding with `0` where necessary to complete 6-bit chunks.

# Remember

- Bits are not encodings.
- An encodings tell us how to group bits into chunks of information.
- An encodings also provides an alphabet that maps a chunk of bits to a symbol.
- Encodings are generally displayed in ASCII because computers can only deal with 1 byte (8 bits) at a time. And ASCII encodes each symbol using 1 byte.

# Further reading

- Binary code
- Hexadecimal
- Base64
- ASCII
- Character Encodings
- If this presentation was interesting to you, go read this now!
  - What Every Programmer Absolutely, Positively Needs To Know About Encodings And Character Sets To Work With Text
- I tricked you, most computers now-a-days display things using the Unicode alphabet and the UTF-8 encoding, not ASCII.
  - But UTF-8 is a superset of ASCII and Unicode is a superset of the ASCII alphabet.

# Thanks!

Reach me at rodesp@hey.com if you have any feedback!