# UTF-8 Encoding

The essence of **UTF (Unicode Transformation Format)** encoding lies in how we can shorten the 32-bit encoding of Unicode characters. UTF-8 is the most widely used encoding method today, especially in web applications and information systems. It efficiently compresses the encoding of Unicode characters to reduce storage space while ensuring compatibility with older standards like ASCII.

## Structure of UTF-8

UTF-8 is a **variable-length encoding** that uses 1 to 6 bytes to represent a character. The number of bytes used depends on the value of the Unicode character:

- Characters from the ASCII range (U+0000 to U+007F) are encoded using **1 byte**.

- Characters beyond the ASCII range (U+0080 to U+07FF) use **2 bytes**.

- For larger Unicode values (U+0800 to U+FFFF), **3 bytes** are used.

- Characters in the supplementary planes (U+10000 to U+10FFFF) are encoded using **4 bytes**.

Here is the UTF-8 encoding scheme:

| **Unicode (bits)** | **UTF-8** |
|---|---|
| 00000000 00000000 00000000 0xxxxxxx | 0xxxxxxx |
| 00000000 00000000 00000yyy yyxxxxxx | 110yyyyy 10xxxxxx |
| 00000000 00000000 xxxxxxxx xxxxxxxx | 1110xxxx 10xxxxxx 10xxxxxx |
| 00000000 000xxxxx xxxxxxxx xxxxxxxx | 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx |
| 000000xx xxxxxxxx xxxxxxxx xxxxxxxx | 111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx |
| 0xxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx | 1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx |

### Explanation of Encoding

1.) **1-byte encoding**: If a character's Unicode value is within the range of U+0000 to U+007F (which corresponds to standard ASCII characters), it is directly represented in UTF-8 using a single byte. For example:

1. The letter 'A' has a Unicode code point of **U+0041**, which is encoded as **01000001** in binary, or **0x41** in hexadecimal, and remains the same in UTF-8.

2.) **2-byte encoding**: Characters outside of the ASCII range require more bytes. For example, the Unicode character 'é' has the code point **U+00E9**:

- In binary: 00000000 11101001
- UTF-8 encoding: 11000011 10101001 (C3 A9 in hexadecimal).

3.) **3-byte encoding**: Characters with even larger code points require 3 bytes. For instance, the character 'अ' (from the Devanagari script) has the code point **U+0905**:

- In binary: 00000000 00001001 00000101
- UTF-8 encoding: 1110 0000 1010 0001 1000 0101 (E0 A5 85 in hexadecimal).

4.) **4-byte encoding**: Characters from supplementary planes, such as certain emojis or rare historical scripts, use 4 bytes. For example, the emoji '' (unicorn) has the Unicode code point **U+1F984**:

- In binary: 00000001 11111001 10000100
- UTF-8 encoding: 11110000 10011111 10011000 10000100 (F0 9F A6 84 in hexadecimal).

#### Example: Encoding the Character 'ó'

The Unicode code point for 'ó' is **U+00F3**, which is decimal 243 or hexadecimal 0x00F3. Since this is more than 7 bits, it fits the second rule for UTF-8 encoding (2-byte encoding).

- Unicode binary: 00000000 00000000 11110011 - UTF-8 binary: 11000011 10110011 (C3 B3 in hexadecimal).

Thus, 'ó' in UTF-8 is encoded as two bytes: **C3 B3**.

## UTF-16 Encoding

**UTF-16** is another encoding standard that uses either 2 or 4 bytes to represent characters. Unlike UTF-8, UTF-16 uses **a minimum of 2 bytes** for each character, which simplifies the encoding of characters but can be less space-efficient for texts containing many ASCII characters.

For characters outside the Basic Multilingual Plane (BMP), UTF-16 requires **4 bytes** to encode using a technique called **surrogate pairs**.

#### Byte Order in UTF-16

One important aspect of UTF-16 is the order of bytes, also known as **endianness**. UTF-16 files typically start with a **Byte Order Mark (BOM)** to indicate the byte order. The BOM is represented by the special Unicode character **U+FEFF**, which is a “zero-width non-breaking space” that doesn't appear in text but signals the byte order:

- **FE FF**: Big-endian (most significant byte first)
- **FF FE**: Little-endian (least significant byte first)

For example, in Windows text files or Microsoft Office documents, you may encounter this BOM at the beginning, especially when opening files in text editors like Notepad.

#### Conclusion

UTF-8 has become the dominant encoding standard because it is backward-compatible with ASCII, space-efficient for texts that are predominantly ASCII, and can represent any character in the Unicode

standard. Meanwhile, UTF-16 is commonly used in environments like Windows, where it handles 2-byte characters efficiently.

In summary: - **UTF-8** is optimal for web content and multi-language support. - **UTF-16** is preferred in some system environments, particularly in Windows applications.

—

With this explanation, your students will gain a clearer understanding of how UTF-8 encoding works and how it differs from UTF-16. If you'd like, I can also add some more specific real-world examples.