Character Encoding

Instructor: Pramod Kumar Jena

What is Character Encoding?

Character encoding is a system that assigns numbers (binary values) to characters so they can be stored and processed by computers.

Example:

- The letter 'A' is stored as **65** in ASCII (or **01000001** in binary).
- The letter 'अ' (Devanagari Hindi) is stored as **2309** in UTF-8 (or **E0 A4 85** in hexadecimal).

---

1. ASCII (American Standard Code for Information Interchange)

Overview:

- Basic encoding for English characters (0-127 in decimal).
- Uses **7 bits** (0 to 127 values).
- Supports only English letters, numbers, and common symbols.

Examples:

Character	Decimal	Binary Representation
'A'	65	01000001
'B'	66	01000010
'a'	97	01100001
'9'	57	00111001

Real-Life Example:

If you open a plain text (.txt) file in Notepad, the text is stored in ASCII if it contains only English letters and numbers.

Limitations:

- Cannot store non-English characters like é, ñ, ü, च, 北京.

---

2. Extended ASCII (8-bit Encoding)

Overview:

- Expands ASCII to **256 characters** (0-255 in decimal).
- Uses **8 bits** (adds 128 extra symbols).
- Supports some European accented characters like é, ñ, ö.

Examples:

Character	Decimal	Binary Representation
'é'	130	10000010
'Ö'	153	10011001

Real-Life Example:

If you type é in a text document and save it in Western European encoding, it uses Extended ASCII.

Limitations:

- Still not enough for languages like Hindi, Chinese, or Arabic.

---

3. Unicode (The Global Standard)

Overview:

- A universal character set that supports **all languages**.

- Unicode assigns a **unique number (code point)** to every character, regardless of language.

Unicode Encoding Types:

UTF-8 (Most Popular)

- Uses **8-bit, 16-bit, 24-bit, or more per character**.
- **Backward compatible with ASCII** (first 128 values are same as ASCII).
- Efficient for English text (**1 byte per character**) but expands for other languages.
- Used in **HTML, JSON, APIs, Databases, and Web Apps**.

UTF-8 Encoding Examples:

Character	Bytes Used	Encoding (Binary / Hex)
'A'	1 byte	01000001 (same as ASCII)
'é'	2 bytes	11000011 10101001
'न'	3 bytes	E0 A4 A8
'😊'	4 bytes	F0 9F 98 8A

Why is UTF-8 the Best Choice?

- Compact for English (**1 byte per character**).
- Supports all languages (expands when needed).
- Default encoding for Web & Databases.

Real-Life Example:

All modern websites, including Google, Facebook, and Wikipedia, use UTF-8 to support multiple languages.

UTF-16 (Used in Windows & Java)

- Uses **16 bits (2 bytes) or more per character**.
- Efficient for **Asian languages** (Chinese, Japanese, Korean).
- Used in **Windows applications, Microsoft Word, Java**.

Real-Life Example:

If you save a Microsoft Word document with Hindi text, it may use UTF-16 internally.

Limitations:

- Takes more space for English text (**2 bytes per character**).
- Not widely used in web development.

UTF-32 (Fixed 4-byte Encoding)

- Every character is stored in **4 bytes (32-bit)**.
- Simple but very inefficient (wastes space).
- Used in some internal programming systems.

Real-Life Example:

Some databases use UTF-32 for internal text processing, but it is rarely used in web applications.

Limitations:

- Too much storage waste for simple text.

---

4. Which Encoding Should You Use?

Use Case	Recommended Encoding
Websites & Web Apps	**UTF-8**
Windows & Java Apps	**UTF-16**
Legacy Text Files	**ASCII** (if only English is used)
Databases (Best Practice)	**UTF-8**

---

5. Final Summary Table

Encoding	Bits Used	Supports	Best For	Example Characters
ASCII	7-bit	English only	Simple text files	A = 65, B = 66
Extended ASCII	8-bit	Some European characters	Legacy systems	é = 130, Ö = 153
UTF-8	8-32 bits	All languages	Websites, APIs, Databases	A = 65, 😊 = F0 9F 98 8A
UTF-16	16-32 bits	Best for Asian languages	Windows, Java	न = 0928, 中 = 4E2D
UTF-32	32-bit	All languages	Some databases	😊 = 0001F60A

---

6. Final Takeaway

If you are building modern applications, especially for the web, always use **UTF-8** because it is:

- **Efficient** (small size for English, flexible for other languages).
- **Compatible** (works with ASCII).
- **Universal** (supports all languages and special symbols).