# **ASCII**

**What is ASCII?**

* ASCII is a character encoding standard
* ASCII uses 7 bits to represent a character.
* By using 7 bits, we can have a maximum of 2^7 distinct combinations.
* This means, we can represent 128 different characters, which map to numbers 0 – 127.

**Mapping Characters**

* We can map characters to a number.
* Likewise, we use a number to map a character.
* We can use binary, decimal, or hexadecimal values to map characters:
  + Binary: 00000000–01111111
  + Decimal: 0–127
  + Hexadecimal: 0–7F

**Wait, 7 bits? But why not 1 byte (8 bits)?**

* The last bit (8th) is used for avoiding errors as [parity bit](https://en.wikipedia.org/wiki/Parity_bit). This was relevant years ago.

**ASCII Characters**

* Most ASCII characters are
  + alphabet characters such as abc or ABC,
  + numbers such as 123,
  + punctuation such as ?&!,
  + control characters such as [carriage return, line feed](https://stackoverflow.com/a/12747850/1132522), tab, etc.

A picture containing graphical user interface

Description automatically generated

As you can see, ASCII was originally only meant for English.

* **What? Why English only? There are so many languages out there!**
* Because the center of the computer industry was in the USA at that time.
* Consequently, they didn’t need to include variants of the ASCII characters with accents and other modifications marks such as á, ü, ç, ñ, etc.
* There are, of course, many other characters used in non-English languages, completely unrelated to ASCII and English, including Arabic, Chinese, Hebrew, Hindi, Japanese, and Korean
* **So how was this problem for a general lack of universal encodings solved?**
* ASCII solves the problem for languages that are based on the Latin alphabet... what about the others needing a completely different language’s alphabet?
* We would have needed an entirely new character sets... that's the rationale behind **Unicode**.
* Unicode doesn't contain every character from every language, but it contains a very large amount of characters ([see this table](https://unicode-table.com/en/#cjk-unified-ideographs-extension-a)).

# **Unicode**

**What is Unicode?**

* Before Unicode, there were many different standards: ASCII in the United States, ISO 8859-1 for Western European languages, KOI-8 for Russian, GB18030 and BIG-5 for Chinese, and so on.
* The fact that there were many different standards caused two problems

1. A particular code value corresponds to different letters in the different encoding schemes.
2. Moreover, the encodings for languages with large character sets have variable length: Some common characters are encoded as single bytes, others require two or more bytes.

* Unicode was invented to overcome the many limitations of these traditional character encoding schemes.
* Unicode defines (less than) 221 characters, which, similarly, map to numbers 0–221 (though not all numbers are currently assigned, and some are reserved).

**Encodings: UTF-8 vs UTF-16 vs UTF-32**

* [This answer](https://stackoverflow.com/a/22404874/1132522) does a pretty good job at explaining the basics:
  + UTF-8 and UTF-16 are variable length encodings.
  + In UTF-8, a character may occupy a minimum of 8 bits.
  + In UTF-16, a character length starts with 16 bits.
  + UTF-32 is a fixed length encoding of 32 bits.
* UTF-8 uses the ASCII set for the first 128 characters. That's handy because it means ASCII text is also valid in UTF-8.
* Unicode is a superset of ASCII, meaning ASCII characters exist in Unicode.
* In ASCII, they are represented as 0–127 or 0-7F.
* This is equivalent to Unicode codepoints U+0000 - U+007F, inclusive.
* Because Unicode characters don't generally fit into one 8-bit byte, there are numerous ways of storing Unicode characters in byte sequences, such as UTF-32, UTF-16, and UTF-8.
* Java strings are represented in UTF-16, which is a 16-bit byte encoding of Unicode.
* Each Java char is a UTF-16 code unit. Unicode codepoints U+0000 - U+FFFF use 1 UTF-16 code unit and thus fit in a single char, whereas Unicode codepoints U+10000 and higher require a UTF-16 *surrogate pair* and thus need two chars.
* You cannot save text to your hard drive as "Unicode". Unicode is an abstract representation of the text. You need to "encode" this abstract representation. That's where an [encoding](https://en.wikipedia.org/wiki/Character_encoding) comes into play.

**Character Encoding**

Character encoding is the process of assigning numbers to graphical characters,

Unicode can be stored using several different [encodings](https://en.wikipedia.org/wiki/Comparison_of_Unicode_encodings), which translate the character codes into sequences of bytes. The Unicode standard defines three and several other encodings exist, all in practice [variable-length encodings](https://en.wikipedia.org/wiki/Variable-length_encoding).

Char Data Type

* The **char** data type was originally intended to describe **individual characters**.
* However, this is no longer the case.
* Nowadays, some Unicode characters can be described with one char value, and other Unicode characters require two char values.
* Literal values of type char are enclosed in single quotes.
* For example, 'A' is a character constant with value 65.
* It is different from "A", a string containing a single character.
* Values of type char can be expressed as hexadecimal values that run from \u0000 to \uFFFF. For example, \u2122 is the trademark symbol (TM) and \u03C0 is the Greek letter pi (π).

Besides the \u escape sequences, there are several escape sequences for special characters, as shown in Table 3.3. You can use these escape sequences inside quoted character literals and strings, such as '\u2122' or "Hello\n". The \u escape se- quence (but none of the other escape sequences) can even be used *outside* quoted character constants and strings. For example,

Literals of types char and String may contain any Unicode (UTF-16) characters. If your editor and file system allow it, you can use such characters directly in your code. If not, you can use a "Unicode escape" such as '\u0108' (capital C with circumflex), or "S\u00ED Se\u00F1or" (Sí Señor in Spanish). Always use 'single quotes' for charliterals and "double quotes" for String literals. Unicode escape sequences may be used elsewhere in a program (such as in field names, for example), not just in char or String literals.

The Java programming language also supports a few special escape sequences for char and String literals: \b (backspace), \t (tab), \n (line feed), \f (form feed), \r(carriage return), \" (double quote), \' (single quote), and \\ (backslash).

There's also a special null literal that can be used as a value for any reference type. null may be assigned to any variable, except variables of primitive types. There's little you can do with a null value beyond testing for its presence. Therefore, null is often used in programs as a marker to indicate that some object is unavailable.

Finally, there's also a special kind of literal called a *class literal*, formed by taking a type name and appending ".class"; for example, String.class. This refers to the object (of type Class) that represents the type itself.