Huge number of character representations (encodings) exist you need know only two:

- **ASCII (ISO 646)**
  - Single byte values, only low 7-bit used, top bit always 0
  - Can encode roman alphabet a-zA-Z, digits 0-9, punctuation, control chars
  - Complete alphabet for English, Bahasa
  - No diacritics, e.g.: ç, so missing a little of alphabet for other latin languages, e.g.: German, French, Spanish, Italian, Swedish, Tagalog, Swahili
  - Characters for most of world’s languages completely missing

- **UTF-8 (Unicode)**
  - Contains all ASCII (single-byte) values
  - Also has 2-4 byte values, top bit always 1 for bytes of multi-byte values
  - Contains symbols for essentially all human languages plus other symbols, e.g.: √ ∑ ∀ ∃

### ASCII Character Encoding

- Uses values in the range 0x00 to 0x7F (0..127)
- Characters partitioned into sequential groups
  - Control characters (0..31) e.g. \n
  - Punctuation chars (32..47, 91..96, 123..126)
  - Digits (48..57) e.g. '0'..'9'
  - Upper case alphabetic (65..90) e.g. 'A'..'Z'
  - Lower case alphabetic (97..122) e.g. 'a'..'z'
- Sequential nature of groups allow ordination e.g. 'J' - 'A' == 10
- See `man 7 ascii`

### Unicode

- Widely-used standard for expressing “writing systems”
  - Not all writing systems use a small set of discrete symbols
- Basically, a 32-bit representation of a wide range of symbols
  - Around 140K symbols, covering 140 different languages
- Using 32-bits for every symbol would be too expensive
  - E.g. standard roman alphabet + punctuation needs only 7-bits
  - Several Unicode encodings have been developed
  - UTF-8 most widely used encoding, dominates web-use
- Designed by Ken Thompson on napkin in New Jersey diner
The 127 1-byte codes are compatible with ASCII
The 2048 2-byte codes include most Latin-script alphabets
The 65536 3-byte codes include most Asian languages
The 2097152 4-byte codes include symbols and emojis and ...

<table>
<thead>
<tr>
<th>ch</th>
<th>code-point</th>
<th>unicode binary</th>
<th>UTF-8 encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>U+0024</td>
<td>0100100</td>
<td>00100100</td>
</tr>
<tr>
<td>ç</td>
<td>U+00A2</td>
<td>00010100000001000</td>
<td>11000010 10100010</td>
</tr>
<tr>
<td>€</td>
<td>U+20AC</td>
<td>00100000101011000</td>
<td>11000010 10000010 101100</td>
</tr>
</tbody>
</table>

Printing UTF-8 in a C program

```c
printf("The unicode code point U+1F600 encodes in UTF-8\n");
printf("as 4 bytes: 0xF0 0x9F 0x98 0x80\n");
printf("We can output the 4 bytes like this: \xF0\x9F\x98\x80\n");
printf("Or like this: ");
putchar(0xF0);
putchar(0x9F);
putchar(0x98);
putchar(0x80);
putchar('n');
```

Converting Unicode Codepoints to UTF-8

```c
uint8_t encoding[5] = {0};
if (code_point < 0x80) {
    encoding[0] = code_point;
} else if (code_point < 0x800) {
    encoding[0] = 0xC0 | (code_point >> 6);
    encoding[1] = 0x80 | (code_point & 0x3f);
} else if (code_point < 0x10000) {
    encoding[0] = 0xE0 | ((code_point >> 12) & 0x3f);
    encoding[1] = 0x80 | ((code_point >> 6) & 0x3f);
    encoding[2] = 0x80 | (code_point & 0x3f);
} else if (code_point < 0x200000) {
    encoding[0] = 0xF0 | ((code_point >> 18) & 0x3f);
    encoding[1] = 0x80 | ((code_point >> 12) & 0x3f);
    encoding[2] = 0x80 | ((code_point >> 6) & 0x3f);
    encoding[3] = 0x80 | (code_point & 0x3f);
}
```

source code for utf8_encode.c
Converting Unicode Codepoints to UTF-8

```c
int main(void) {
    printf("U+%0x UTF-8: ", code_point);
    for (uint8_t *s = encoding; *s != 0; s++) {
        printf("0x%02x ", *s);
    }
    printf(" %s\n", encoding);
}
```

source code for utf8_encode.c

Summary of UTF-8 Properties

- Compact, but not minimal encoding; encoding allows you to resync immediately if bytes lost from a stream.
- ASCII is a subset of UTF-8 - complete backwards compatibility!
- All other UTF-8 bytes > 127 (0x7f)
  - no byte of multi-byte UTF-8 encoding is valid ASCII.
- No byte of multi-byte UTF-8 encoding is 0
  - can still use store UTF-8 in null-terminated strings.
- 0x2F (ASCII /) and 0x00 can not appear in multi-byte characters
  - hence can use UTF-8 for Linux/Unix filenames
- C programs can treat UTF-8 similarly to ASCII.
- Beware: number of bytes in UTF-8 string != number of characters.