2.9 Application: Representing Text

We have mentioned that computers use a series of 0s and 1s to store data. These 0s and 1s represent numbers. So then, how can numbers represent textual data (i.e., a string)? The answer is functions.

Once upon a time, humans interacted with computers through punched paper tape (or simply punched tape). A hole (or the lack of a hole) at a particular location on the tape represented a 0 or a 1 (i.e., binary). Today we would call each 0 or 1 a **bit**. Obviously, this is much more tedious than using our modern input peripherals: keyboards, mice, touch screens, etc. Eventually, a standard for representing characters (e.g., letters, numbers) with holes was settled on. Using only 7 locations on the tape, 128 different characters could be represented ($2^7 = 128$).

The standard was called ASCII (pronounced ass-key) and it persists to this day. You can think of the ASCII standard as a function with domain $\{0, 1, \ldots, 127\}$, whose codomain is the set of all possible characters. This function is *one-to-one*, meaning no two numbers map to the same character—this would be redundant for the purpose of encoding the characters. This standard covered all English letters (lowercase and uppercase), digits, punctuation, and various others (e.g., to communicate a new line). For example, the number 65 mapped to the letter `'A'` and the number 33 mapped to the punctuation mark `'!'`.

But what about other languages? Computer scientists extended ASCII from length-7 to length-8 sequences of bits, and hence its domain increased to size 256 ($\{0, 1, \ldots, 255\}$). This allowed "extended ASCII" to support some other characters used in similar Latin-based languages, such as `'é'` (233), `'ö'` (246), `'¥'` (165), and other useful symbols like `'©'` (169) and `'½'` (189). But what about characters used in very different languages (e.g., Greek, Mandarin, Arabic)?

The latest standard, Unicode, uses **up to 32 bits** that gives us a domain of $\{0, 1, \ldots, 2^{32} - 1\}$, over 4 billion different numbers. This number is in fact larger than the number of distinct characters in use across all different languages! There are several *unused numbers* in the domain of Unicode—Unicode is not technically a function defined over $\{0, 1, \ldots, 2^{32} - 1\}$ because of this.

But with the pervasiveness of the Internet, these unused numbers are being used to map to emojis. Of course, this can cause some lost-in-translation issues. The palm tree emoji may appear different on your device than a friend's. In extreme cases, your friend's device may not see a palm tree at all or see a completely different emoji. Part of the process involves submitting a proposal for a new emoji. But the second half of that process means that computer scientists need to support newly approved emojis by updating their software. And, of course, in order to do that computer scientists need to have a firm understanding of functions!

Python's Unicode conversion functions

Python has two built-in functions that implement the (partial) mapping between characters and their Unicode number. The first is `ord`, which takes a single-character string and returns its Unicode number as an `int`.

```
>>> ord('A')
65
>>> ord('é')
233
>>> ord('♥')
9829
```

The second is `chr`, which computes the *inverse* of `ord`: given an integer representing a Unicode number, `chr` returns a string containing the corresponding character.

```
>>> chr(65)
'A'
>>> chr(233)
'é'
>>> chr(9829)
'♥'
```

Unicode representations are a source of one common source of surprise for Python programmers: string ordering comparisons (`<`, `>`) are based on Unicode numeric values! For example, the Unicode value of `'Z'` is 90 and the Unicode value of `'a'` is 97, and so the following holds:

```
>>> 'Z' < 'a'
True
>>> 'Zebra' < 'animal'
True
```

This means that sorting a collection of strings can seem alphabetical, but treats lowercase and uppercase letters differently:

```
>>> sorted({'David', 'Mario', 'Jacqueline'})
['David', 'Jacqueline', 'Mario']
>>> sorted({'david', 'Mario', 'Jacqueline'})
['Jacqueline', 'Mario', 'david']
```

References

- ASCII Code: The extended ASCII table
- Unicode Character Table