ADEPT PRACTICE

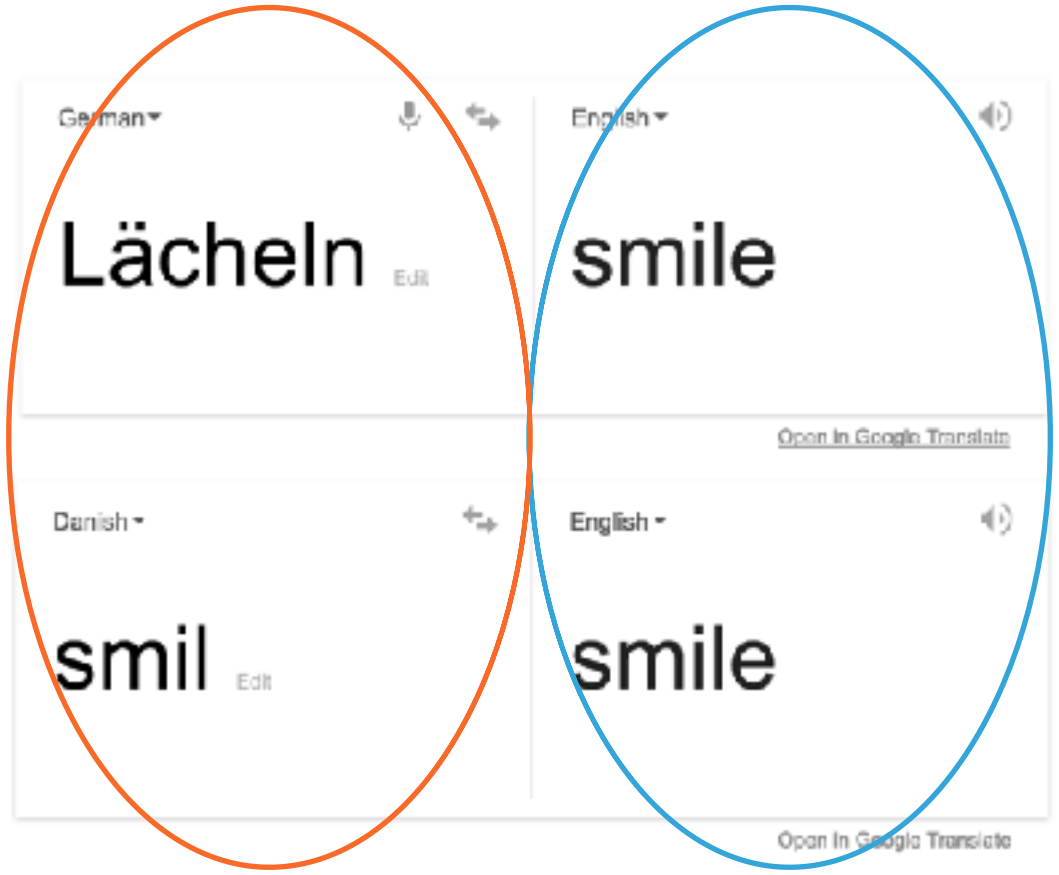
CHARACTER ENCODING

Character encodings are like foreign languages

And the character is the concept that a word in the language represents.

We understand the concept by translating words into some language representation that we understand (say, english).

Foreign word for "smile"

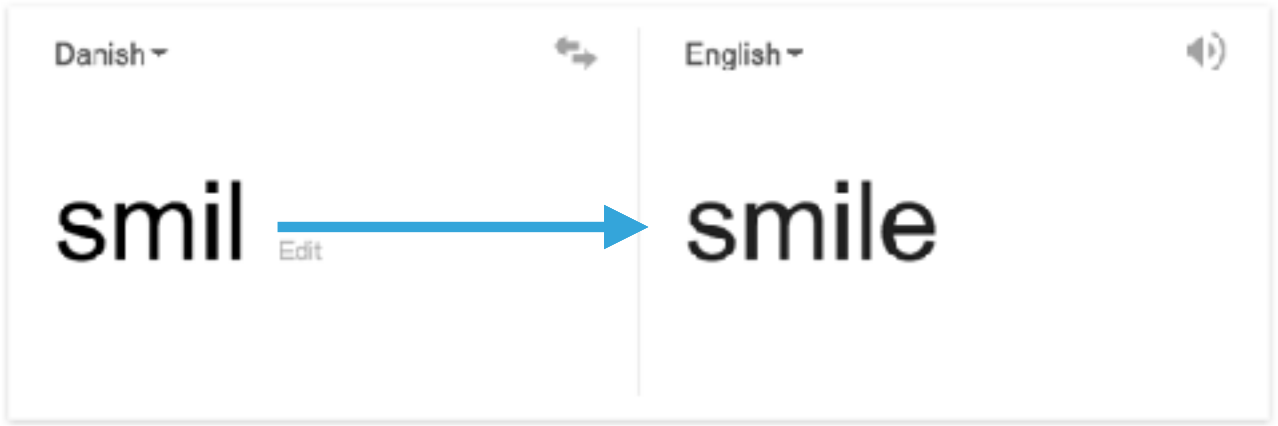


The concept of "😃"

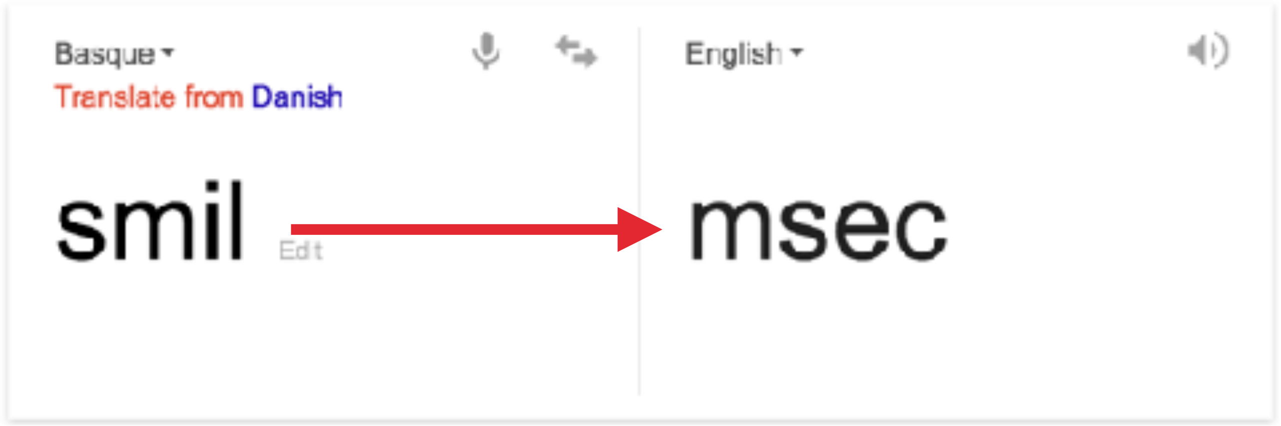
ANALOGY

To understand the word, we need to know how to translate the foreign word into English. ***We need the right dictionary.***

To understand the encoded data, we need to know how to translate ("decode") it into the concept (the "code point" of a specific character). ***We need the right encoding.***



We can translate "smil" from Danish to English if we use a Danish dictionary.



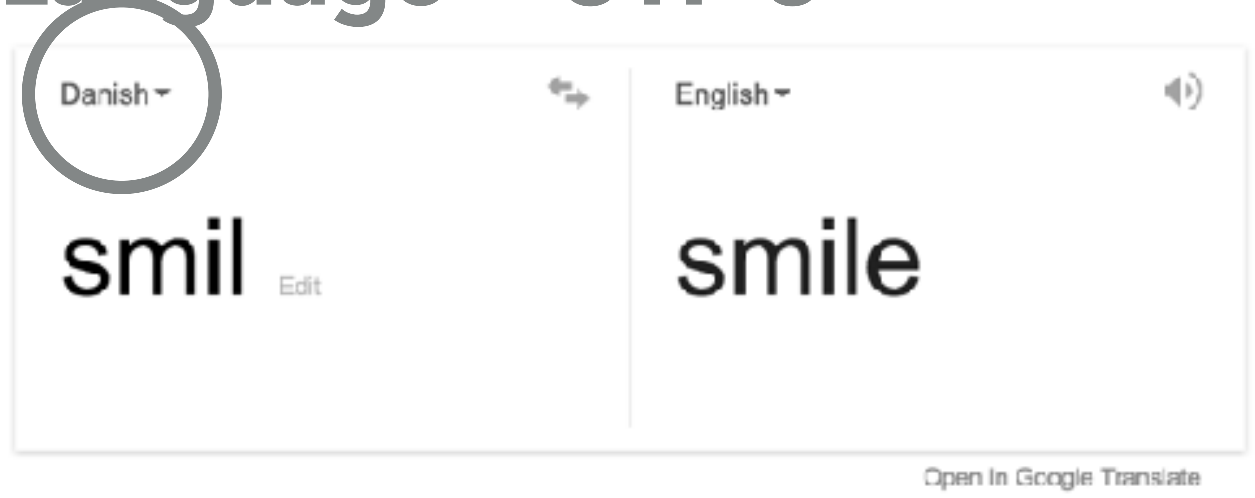
We can't get the right meaning if we try to use a different dictionary ("decoder") to translate the word.

Stored on your computer:

11110000 10011111 10011000 10000011

Get the right dictionary:

Language = UTF-8

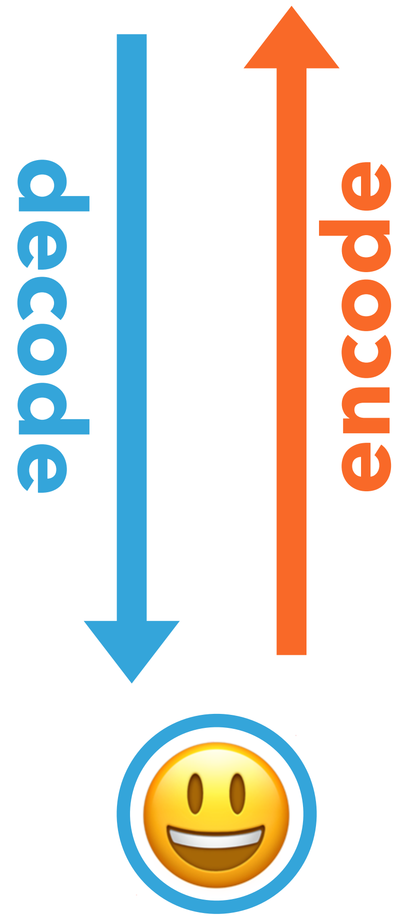


Danish

smil

English

smile



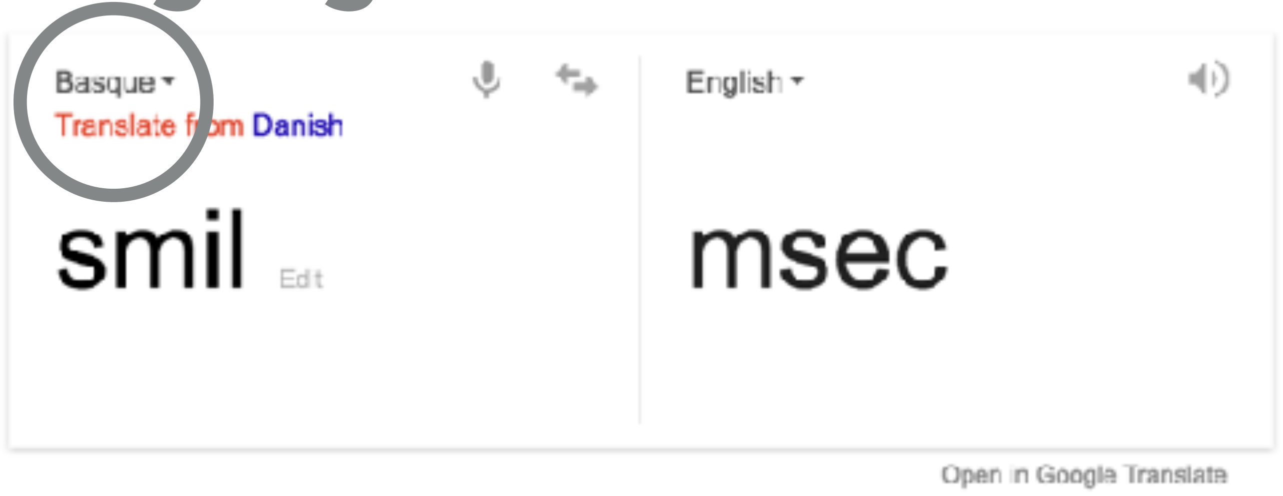
DIAGRAM

Stored on your computer:

11110000 10011111 10011000 10000011

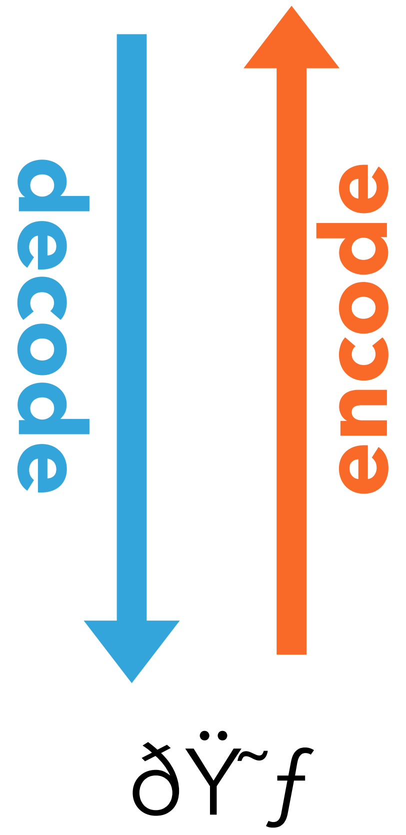
Getting the wrong dictionary:

Language = LATIN-1



smil

msec



PLAIN ENGLISH

Translating from characters to bytes is encoding.

Translating from bytes to characters is decoding.

TECHNICAL DEFINITION

- A computer can only store information as **bytes** (sets of 8 bits, where a bit is a "1" or a "0")
- There are 255 bytes, and many more characters
- Group bytes together to get numbers greater that 255
- Then we need a translation between the numbers represented by groups of bytes ("code points"), and text characters

$$\begin{tikzcd}
\textbf{Bytes} \arrow[r, leftrightarrow, "\text{how groups of bytes correspond to numbers}"'] & \textbf{Numbers} \arrow[r, leftrightarrow, "\text{how numbers correspond to characters}"'] & \textbf{Characters}
\end{tikzcd}$$

TECHNICAL EXAMPLE

- Some encodings (ex: ASCII) take exactly one byte per character, and can represent only 255 characters. Other single-byte encodings simply assign a different set of 255 characters (cyrillic characters, or arabic characters) to those 255 numbers.
- Other encodings (ex: UTF-16) take a larger (ex: 2 bytes, 16 bits) number of bytes to represent a character.
- UTF-8 takes a variable number of bytes to represent a character. This allows us to not waste 2 whole bytes to represent a common character ("e"), while allowing us to represent very uncommon characters ("😃","🔣") when we need to.

TECHNICAL EXAMPLE

- There are many different encodings (just as there are many different languages).
- I’m going to walk through the most common web-text encoding: **UTF-8**.
- **UTF** stands for: **U**nicode **T**ransformation **F**ormat
  - Transformation is that “translation” between bytes and numbers (remember: 1s and 0s to numbers). In “UTF-**8**”, the “**8**” tells us specifically about the kind of translation.
  - Unicode refers to a mapping of numbers (“code points”) to characters, not a method of reading bytes. UTF-_ are all unicode encodings (remember: numbers to characters)

How UTF-8 works:

What is stored on your computer:

11110000 10011111 10011000 10000011

The translation between this binary representation and “128515” is UTF-**8** specific.

The first 4 bits of the first byte tell you how many bytes form one code point. In this case, we have 4 ones–that means 4 bytes, or 32 bits.

The leading two bits of the next bytes (the “continuation bytes”) are always “10.”

The remaining bytes, in sequence, are the number of the unicode code point, represented in binary.

0000011111011000000011 = 128515 = 😃

The translation between “128515” and “😃” is the **unicode** (**U**TF-_) character mapping.