A study in UTF-8 on OsX

Prelude

UTF-8 stands for Unicode Transform Format. It can store regular 8-bits ASCII characters and is backwards compatible with the ASCII table. For a study of the ASCII table, use "man ascii".

For more complex characters likle kanji characters or emoji, more than 8-bits are used within UTF-8.
I always found this "UNICODE" stuff overly complex and stayed away from it.. until today. Below shows my search for the truth.. conclusion: as usual in technology, it's not that hard once you understand it.

The contents of a file in UNIX with an emoji

Here is an example of a text file with the following contents:
AB😀

Note: to type an emoji in OsX, use <command><control><SPACE>

The HEX characters in the file are:

```
$ od -x a
0000000       4241     9ff0     8098     000a
```

Here 0x42 = "B" and 0x41 = "A", the byte ordering (Little-endian) orders them this way. (Lowest 16 -bits first) The rest is used for the Emoji..

Big Endian / Little Endian Byte ordering

To test whether my OsX system (UNIX) is BIG / little Endian, this Python code will reveal it:

```
from sys import byteorder
print(byteorder)
```

Output: **little**

→ so my system is **LITTLE** Endian, as Intel systems generally are!

The search for the truth

A file that solely contains the emoji, reveals the following HEX content:

```
0x F0 9F 98 80 00 0a
```

The 0x 00 0A = New line.

An example of another Emoji and its Unicode-codepoint:

emoji	unicode	utf-8	another utf-8
😀	U+1F601	\xf0\x9f\x98\x81	\xed\xa0\xbd\xed\xb8\x81

Definition from the net:

A study in UTF-8 on OsX

"UTF-8 uses 1 byte to represent characters in the ASCII set, 2 bytes for characters in several more alphabetic blocks, 3 bytes for the rest of the BMP, and 4 bytes as needed for supplementary characters."

Clearly my emoji character requires 4B here.

To see the UTF-8 codepoint for the emoji, refer to this website:
https://unicode.org/emoji/charts/full-emoji-list.html

face-smiling

№	Code	Browser
1	U+1F600	😀

So 0x **F0 9F 98 80** in my file should match UNICODE Codepoint U+1F600.

From Wikipedia:

Layout of UTF-8 byte sequences

Number of bytes	First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
1	U+0000	U+007F	0xxxxxxx			
2	U+0080	U+07FF	110xxxxx	10xxxxxx		
3	U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
4	U+10000	[nb 2]U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

My codepoint U+1F600 complies with the last line as it lies between U+100000 and U+10FFFF so requires 4Bytes to store the Emoji.

Where does the U stand for? These are the FIXED bits in the table above. The "UNICODE" bits.

From a file 'b' that only contains "€", the HEX contents is:

```
$ od -x b
0000000       82e2    0aac
0000004
```

Skipping the 0a which is NewLine, the contents reads:
0x E2 82 AC

Wikipedia to the rescue

The explanation for this character with codepoint U+20AC from Wikipedia:

Consider the encoding of the Euro sign, €:

1. The Unicode code point for "€" is U+20AC.
2. As this code point lies between U+0800 and U+FFFF, this will take three bytes to encode.

3. Hexadecimal 20AC is binary 0010 0000 1010 1100. The two leading zeros are added because a three-byte encoding needs exactly sixteen bits from the code point.
4. Because the encoding will be three bytes long, its leading byte starts with three 1s, then a 0 (1110...)
5. The four most significant bits of the code point are stored in the remaining low order four bits of this byte (1110 0010), leaving 12 bits of the code point yet to be encoded (...0000 1010 1100).
6. All continuation bytes contain exactly six bits from the code point. So the next six bits of the code point are stored in the low order six bits of the next byte, and 10 is stored in the high order two bits to mark it as a continuation byte (so 1000 0010).
7. Finally the last six bits of the code point are stored in the low order six bits of the final byte, and again 10 is stored in the high order two bits (1010 1100).

The three bytes 1110 0010 1000 0010 1010 1100 can be more concisely written in hexadecimal, as E2 82 AC.

The proof is in the pudding

So good for Wikipedia, but how do I apply this for my file that only contains a 😬?

Codepoint U+1F600 = 0x 0001 1111 0110 0000 0000

Note, an extra Binary "0" is pre-pended (see Wikipedia) so:
U+1F600 = 0x 0 0001 1111 0110 0000 0000

If we use the last table of UTF-8: (nr of 'x' bit is: 21)
1111 0xxx 10xx xxxx 10xx xxxx 10xx xxxx

Here the RED BITS are fixed / "UNICODE" bits. If you substitute the "x" for the bits of my Unicode emoji:
1111 0000 1001 1111 1001 1000 1000 0000

This amounts to:
0x F 0 9 F 9 8 8 0

QED:
0x F0 9F 98 80