

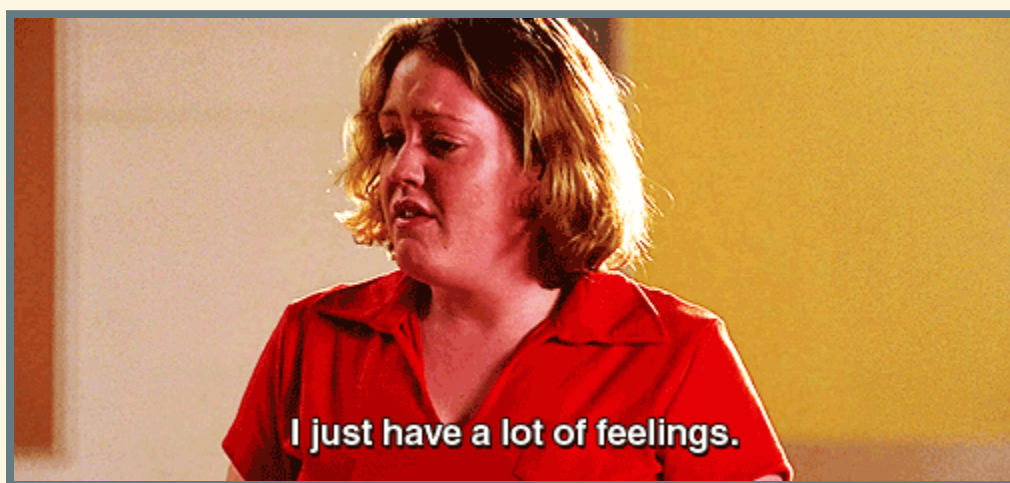
# UNICODE!

## (AND HOW ES6 CAN HELP)

By [Eddie Antonio Santos](#) / [@\\_eddieantonio](#)

Source: [eddieantonio.ca/unicode-es6](http://eddieantonio.ca/unicode-es6)

**(HYPOTHETICAL SCENARIO)**





bitter

(Thanks, [Kim!](#))

A man with a shaved head, wearing a dark t-shirt and a black beaded necklace with a silver pendant, is shown in a dimly lit room. He is holding a large stack of US dollar bills against his face, with his hand covering his eyes. The scene is dramatically lit from the side, creating strong shadows. The text "(DEMO)" is overlaid in the center of the image.

(DEMO)

# WHAT DOES `String.length` ACTUALLY MEASURE?

Let's consult the [standard](#)!

---

The String type is the set of all ordered sequences of zero or more **16-bit unsigned integer values** (“elements”)

---

The String type is generally used to represent textual data in a running ECMAScript program, in which case each element in the String is treated as a **UTF-16 code unit** value.

---

The length of a String is the **number** of elements (i.e., **16-bit values**) within it.

---



**WHAT THE IS A UTF-16 CODE UNIT VALUE?**

**UNICODE!**

# WHAT IS UNICODE?

- A mapping of numbers (*code points*) to every character. Ever.
- Database of properties for each character (e.g., name, general category).

# CODE POINTS

Unique number given to each character

They look like this:

*U+hhhh* or *U+hhhhhh*

Range from U+0000 to U+10FFFF

1,114,112 code points available in total

(See [Unicode Chapter 3](#))

# A TOUR OF THE UNICODE CHARACTER SPACE!

Code points are divided into 17 *planes*

# THE BASIC MULTILINGUAL PLANE

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F	
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F	
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F	
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F	
80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F	
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F	
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF	
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF	
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF	
D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF	
E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF	
F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF	

Latin script

non-Latin European scripts

African scripts

Middle Eastern and Southwest Asian scripts

South and Central Asian scripts

Southeast Asian scripts

East Asian scripts

Unified CJK Han

Indonesian and Oceanic scripts

American scripts

Symbols

Miscellaneous characters

Private use

UTF-16 surrogates

Unallocated code points

As of Unicode version 8.0

# THE BASIC MULTILINGUAL PLANE

## (PLANE 0)

- Characters from practically all widely-used modern-day scripts
- Code points are notated as U+*hhhh*
- Code points range from U+0000 to U+FFFF

# ALL UNICODE CODE POINTS

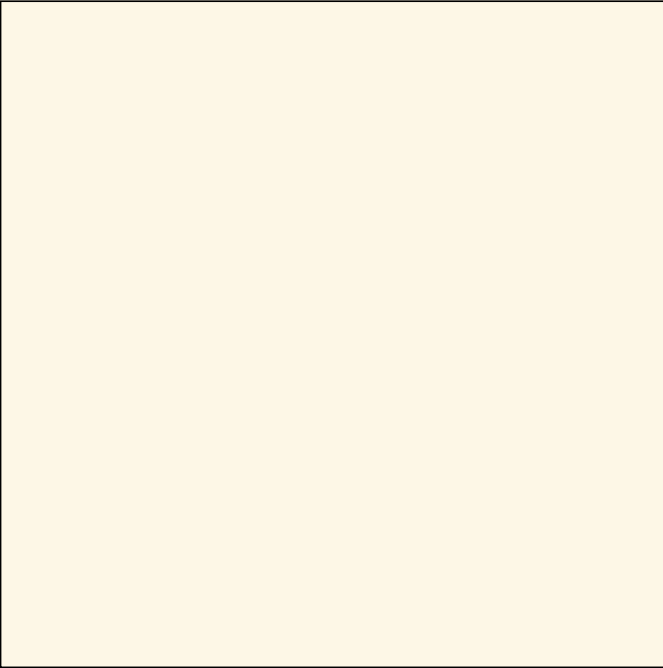


00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF

BMP

100	101	102	103	104	105	106	107	108	109	10A	10B	10C	10D	10E	10F
110	111	112	113	114	115	116	117	118	119	11A	11B	11C	11D	11E	11F
120	121	122	123	124	125	126	127	128	129	12A	12B	12C	12D	12E	12F
130	131	132	133	134	135	136	137	138	139	13A	13B	13C	13D	13E	13F
140	141	142	143	144	145	146	147	148	149	14A	14B	14C	14D	14E	14F
150	151	152	153	154	155	156	157	158	159	15A	15B	15C	15D	15E	15F
160	161	162	163	164	165	166	167	168	169	16A	16B	16C	16D	16E	16F
170	171	172	173	174	175	176	177	178	179	17A	17B	17C	17D	17E	17F
180	181	182	183	184	185	186	187	188	189	18A	18B	18C	18D	18E	18F
190	191	192	193	194	195	196	197	198	199	19A	19B	19C	19D	19E	19F
1A0	1A1	1A2	1A3	1A4	1A5	1A6	1A7	1A8	1A9	1AA	1AB	1AC	1AD	1AE	1AF
1B0	1B1	1B2	1B3	1B4	1B5	1B6	1B7	1B8	1B9	1BA	1BB	1BC	1BD	1BE	1BF
1C0	1C1	1C2	1C3	1C4	1C5	1C6	1C7	1C8	1C9	1CA	1CB	1CC	1CD	1CE	1CF
1D0	1D1	1D2	1D3	1D4	1D5	1D6	1D7	1D8	1D9	1DA	1DB	1DC	1DD	1DE	1DF
1E0	1E1	1E2	1E3	1E4	1E5	1E6	1E7	1E8	1E9	1EA	1EB	1EC	1ED	1EE	1EF
1F0	1F1	1F2	1F3	1F4	1F5	1F6	1F7	1F8	1F9	1FA	1FB	1FC	1FD	1FE	1FF

SMP



# THE ASTRAL PLANES

- Everything else (Planes 1-16)
- Characters from ancient scripts, alternative scripts, pictograms, and rare and archaic CJK(V) ideograms (Chinese-style characters). Also, (most) Emoji.
- Two entire planes devoted to *private use characters*
- Code points are notated as U+*hhhhhh*
- Code points range from U+010000 to U+10FFFF

**PUA-B**

# WHAT IS *NOT* UNICODE?

- ~~a character encoding~~ It's several character encodings!
- Code points  $\neq$  Bytes

# CODE UNIT

Smallest unit of storage required to store or transmit a single character in an encoding scheme

# WAYS OF TRANSMITTING CODE POINTS

- UTF-8
- UTF-16
- UTF-32/UCS-4



**UTF-16 NEEDS TWO CODE UNITS TO REPRESENT  
ONE ASTRAL CODE POINT**

# BACK TO OUR PROBLEM...

We want to count **code points** and not **code units**

**ENTER:**

`String.prototype[@@iterator]`

`(String.prototype.codePointAt() exists too)`



---

When the `@@iterator` method is called it returns an `Iterator` object ([25.1.1.2](#)) that **iterates over the code points** of a `String` value, returning each code point as a `String` value.

---

# COMPARE

```
let a = [];  
for (let c of s) {  
  a.push(c);  
}
```

VS.

```
var i, a = [],  
for (i = 0; i < s.length; i++) {  
  a.push(s[i]);  
}
```

# LET'S FIX OUR CODE!

Trick: Use `Array#from`

(it just does this:)

```
function (s) {  
  let a = [];  
  for (let c of s) {  
    a.push(c);  
  }  
  return a;  
}
```

# CHANGE OF PLANS



bitter

(Thanks, [Kim!](#))

**(DEMO)**

# THREE DIFFERENT WAYS OF WRITING

- pho = o + o + o
- pho = Matthew + o
- pho = in



# NORMALIZATION FORMS!

Useful for comparing *different representations* of the same abstract character sequence

- **NFD** Canonical decomposition
- **NFC** Canonical decomposition, followed by Canonical Composition
- **NFKD** Compatibility Decomposition
- **NFKC** Compatibility Decomposition, followed by Canonical Composition

(See [UAX # 15](#) )

# CANONICAL (DE)COMPOSITION

- NFD :  $\Rightarrow \text{ø} \text{ ǒ}$
- NFC : on  $\Rightarrow$  in

**ENTER:**

`String.prototype.normalize()`

When in doubt, use

**NFC**

**LET'S FIX OUR APP!**

# COMPATIBILITY

Check the [compatibility table](#)!

**ASK ME QUESTIONS**



# RESOURCES

# STANDARDS

- Unicode 8.0
- ECMAScript 6

# UNICODE

- Mathias Bynens: JavaScript ❤️ Unicode
- Tom Scott: Accidental Emoji Expert
- Tom Scott: Characters, Symbols and the Unicode Miracle
- Tom Scott: Why You Can Tweet More In Japanese: What Counts As A Character?

# OTHER

- UTF-8 Everywhere
- JavaScript has a Unicode problem