Text

- Character sets
- Internationalization
- Validation
- Masking

Representing Text

- *Text* means a series of *characters*
  - alphabet, digits, whitespace, special characters, etc.
- *Sets of characters* form a *writing system* in human languages
  - e.g. Latin alphabet, Chinese characters, Arabic alphabet, Devanagari, Bengali, etc.,
- Need **standardized** encodings for characters in binary

https://www.smashingmagazine.com/2012/06/all-about-unicode-utf8-character-sets/

ASCII

- **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange
- Originally created in 1960s as a 7-bit encoding for teleprinters
  - 52 Latin chars, 10 digits, common symbols, control chars, ...
    e.g. **A** is #65; **0** is #48; **@** is #64; "carriage return" is #13
  - adopted as American standard in 1968 (i.e. a "world" standard)
- In 1970s, computers using 8-bit architectures became popular
  - extra bit meant space for another 128 characters
  - but initially no agreement for encoding
- In 1990s, ISO standardized 15 "code pages" for different encodings
  - e.g. the Cyrillic ISO-8859-5 code page encodes **Я** as #207
- Assumes using correct code page to exchange international text

https://www.smashingmagazine.com/2012/06/all-about-unicode-utf8-character-sets/

ASCII control characters

00	NULL	(Null character)
01	SOH	(Start of Header)
02	STX	(Start of Text)
03	ETX	(End of Text)
04	EOT	(End of Trans.)
05	ENQ	(Enquiry)
06	ACK	(Acknowledgement)
07	BEL	(Bell)
08	BS	(Backspace)
09	HT	(Horizontal Tab)
10	LF	(Line feed)
11	VT	(Vertical Tab)
12	FF	(Form feed)
13	CR	(Carriage return)
14	SO	(Shift Out)
15	SI	(Shift In)
16	DLE	(Data link escape)
17	DC1	(Device control 1)
18	DC2	(Device control 2)
19	DC3	(Device control 3)
20	DC4	(Device control 4)
21	NAK	(Negative acknowl.)
22	SYN	(Synchronous idle)
23	ETB	(End of trans. block)
24	CAN	(Cancel)
25	EM	(End of medium)
26	SUB	(Substitute)
27	ESC	(Escape)
28	FS	(File separator)
29	GS	(Group separator)
30	RS	(Record separator)
31	US	(Unit separator)
127	DEL	(Delete)

ASCII printable characters

32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Extended ASCII characters

128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

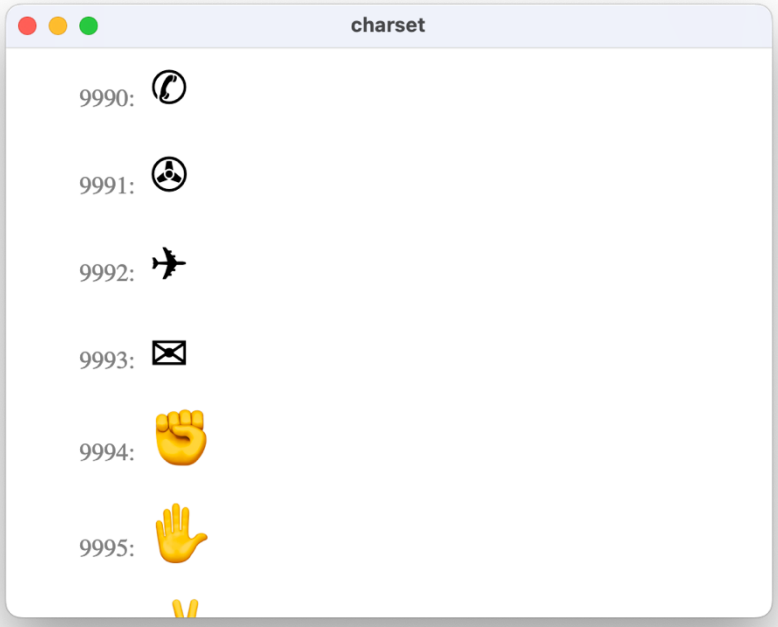
Unicode

- **Unicode** is a *superset* of ASCII
  - Capacity is up to 1,114,112 characters
  - Version 6.1 actually encodes only 110,000 characters
  - Every *character* in *every language* has a unique encoding
  - Unicode has replaced ASCII in common use
- General structure
  - #0 to #127 have same meaning as ASCII (e.g. Latin A is #65)
  - #128 to #256 are common signs and accented characters
  - after #256 many more accented characters
  - after #800 Greek alphabet, then Cyrillic, etc.
- Unicode character codes written as "U+" then 4-digit hexadecimal
  - e.g. **H** is **U+0048** in Unicode (instead of decimal **72** in ASCII)
- Unicode encoding needs more than 1 byte
  - the *implementation* of Unicode is not defined by Unicode

https://www.w3.org/International/articles/definitions-characters/index

charset

- Displays first 10,000 Unicode characters
- Uses `String.fromCharCode(i)`



UTF-8

- **U**niversal Character Set **T**ransformation **F**ormat **8** bit
- Internally, web browsers use 4-byte "wide characters"
  - in C/C++ it's the `wchart_t` type
- The problem is sending, receiving, and storing text
  - some existing software sends/receives in 1-byte units
  - using 4 bytes for each Latin character would bloat storage
- UTF-8 uses a multi-byte variable width encoding:

Code point ↔ UTF-8 conversion

First code point		Last code point		Byte 1	Byte 2	Byte 3	Byte 4
0	U+0000	127	U+007F	0xxxxxxx			
128	U+0080	2047	U+07FF	110xxxxx	10xxxxxx		
2048	U+0800	65535	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
65536	U+10000	[b]	U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

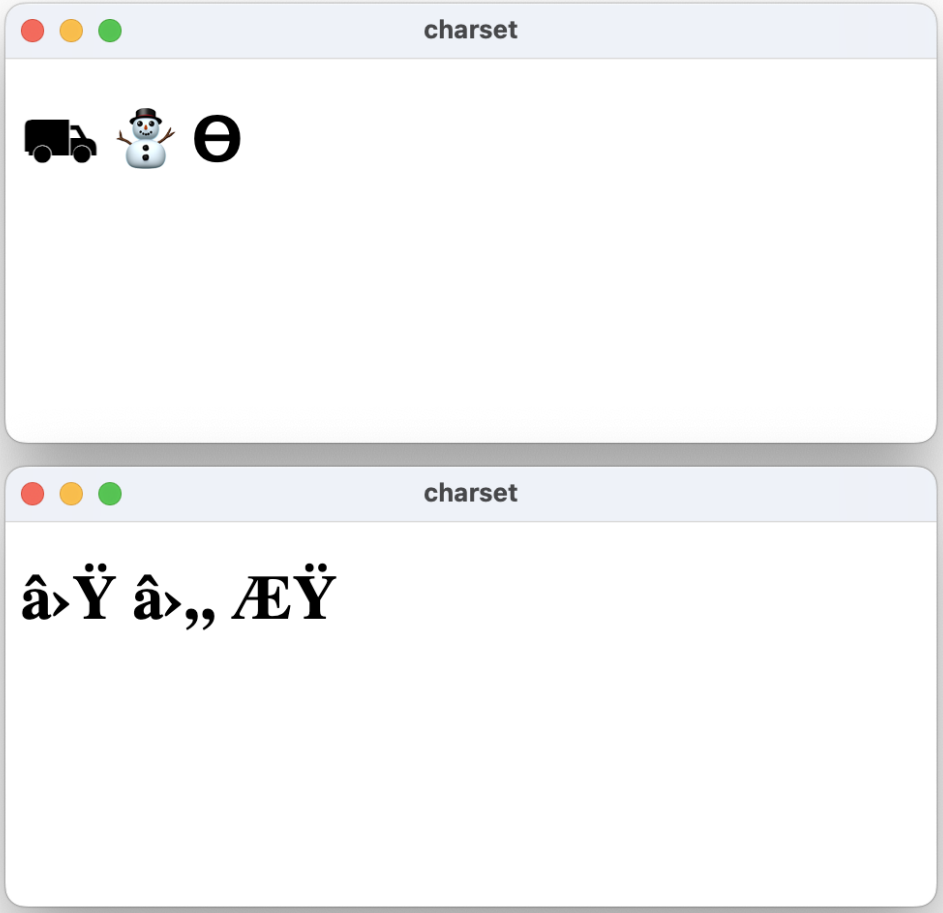
how many bytes to expect (e.g. 4)

identifies this as a "continuation" byte

https://en.wikipedia.org/wiki/UTF-8#Examples

charset

- Uncomment <h1> to display Unicode characters
- Change charset from "UTF-8" to "ascii"



Internationalization and Localization

- **Internationalization (i18n)** is designing and developing software so it can be adapted to different cultures and languages
  - use i18n features like Unicode characters, bidirectional text, etc.
  - support locale formats for numbers, currency, date, time, etc.
  - plan for regional differences in storing information
  - separate localization elements from source code and content
- **Localization (l10n)** is the *act of implementing* i18n
  - **locale** means the region and language, e.g. **en_CA** or **fr_CA**

https://phrase.com/blog/posts/step-step-guide-javascript-localization/

Implementing Localization

Steps for localization in the browser:

1. Use HTML data attribute to identify i18n text elements

```
<label data-i18n="label-name" ...
```

2. Create JSON translation data structures

```
en: { label-name: "Name" ...
fr: { label-name: "Nom" ...
```

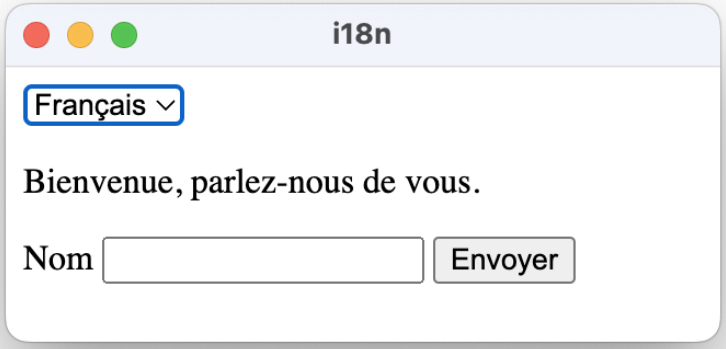
3. Use preferred browser locale

```
navigator.language
```

4. Add a locale switcher (recommended)

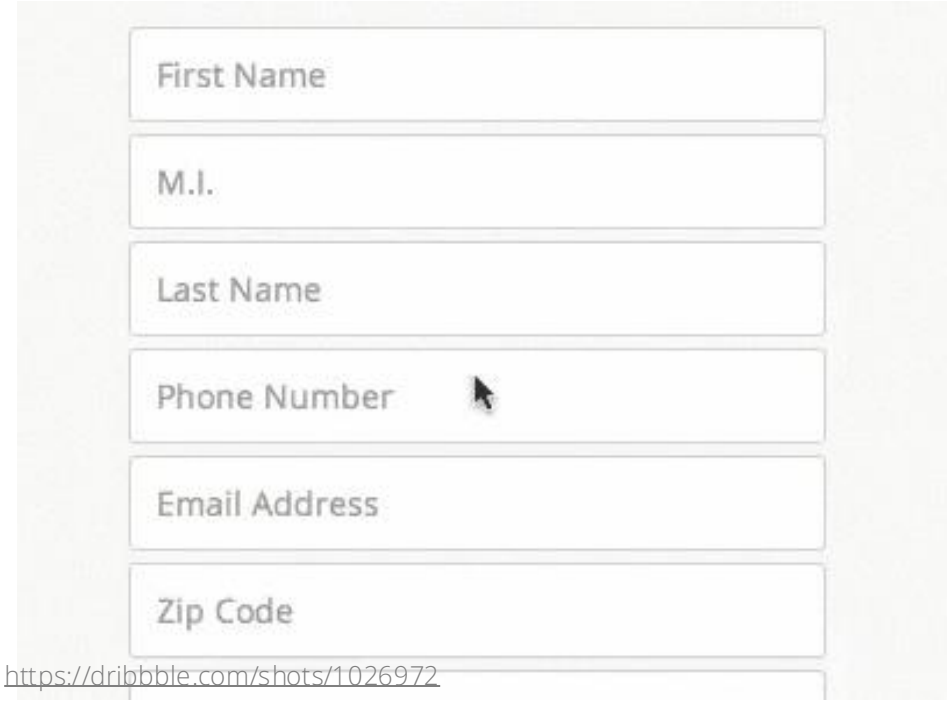
i18n

- TypeScript types for translation table
- structure of translation table
- setting translations to all i18n elements
- using default browser locale
- locale switcher
  - select input widget
  - "change" event



Form Validation

- Interfaces often need to *validate* text input typed by the user
- For example:
  - a required field (e.g. credit card number)
  - a certain format (e.g. numeric, postal code, phone number)
  - within a certain range (e.g. number between 0 and 100)
  - unique (e.g. choose a username to one else has used)



https://dribbble.com/shots/1026972

Why Form Validation

1. The system needs the right data, in the right format
   - The model expects certain kinds of data, or logic won't work
2. To guide users
   - e.g. force them to use secure passwords
3. Protect the system
   - attacks mounted through unprotected text submission

Unprotected Text Submission Attack

In practice, input sanitization is also done on the server

- SQL Injection: when user input is passed directly to SQL statement

Username bird2002

```
sql = `SELECT * FROM users WHERE name = '${userName}';`
```

If a real username is entered everything is fine

…. but a malicious user could enter:

Username a';DROP TABLE users; SELECT * FROM userinfo WHERE 't' = 't

```
sql = `SELECT * FROM users WHERE name = '${userName}';`
```

```
SELECT * FROM users WHERE name = 'a';DROP TABLE users; SELECT *
FROM userinfo WHERE 't' = 't';
```

https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Website_security

Guidelines for Form Validation

- Place error messages near fields
- Use colour to differentiate errors from normal field states
- When possible, accept data formatted in different ways
- When possible, filter invalid characters from being entered
- When possible, validate a field before input is complete

https://www.nngroup.com/articles/errors-forms-design-guidelines/

HTML Form Basics

- <form> to group different input widgets together
- <label> to associate text with an input field
  - association by element id

```
<label for="name">Name</label>
<input type="text" id="name" />
```

- <input> placeholder text
  - `placeholder` to display an example value (or as a compact label)

```
<input type="text" placeholder="Name" id="name" />
```

Built-in HTML Validation

- Modern HTML5 widgets can validate many kinds of input
- Use specific type of input

  `type="number"`

  `type="email"`
- Use attributes to configure validation

  `required`

  `minlength, maxlength`

  `min, max`

  `pattern`
- Use CSS pseudo-class selectors to provide validation feedback

  `:required`

  `:invalid`

Regular Expressions ("regex", "re")

A sequence of characters that specifies a search pattern in text

- from language theory and theoretical computer science
- a regex pattern describes a deterministic finite automaton (DFA)

Used in form validation to “test” if string is in correct format:

Postal Code (upper case only with optional space in between)

```
[A-Z]\d[A-Z]\s*\d[A-Z]\d
```

Number (decimals allowed, positive only, optional leading 0)

```
\d*\.?\d*
```

Phone Number (10 digit North American with formatting options)

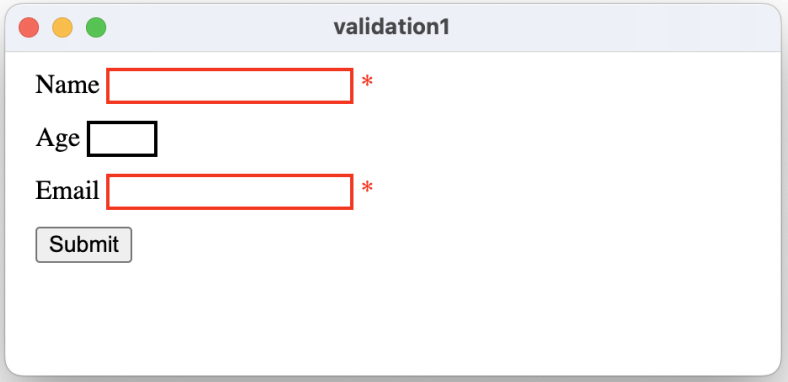
```
\(?\d{3}\)?[\s.\-]?\d{3}[\s.\-]?\d{4}
```

Regex Tutorial
https://regexone.com

Regex Testing, Explanation, Reference
https://regex101.com

validation1

- Attributes
  - `required`
  - `pattern=" ... "`
  - `type="number", type="email", etc.`
- CSS pseudo-classes
  - `:invalid`
  - `:valid`
  - `:required`
- Using div containers for form elements
- Advanced CSS to add * after
  - `:has()`
  - `::after`



Custom Validation with Constraint Validation API

- Only available on some widgets
  - button, input, select, ...
- form `novalidate` to turn off standard validation messages
- API properties and methods

```
validity: ValidityState
checkValidity(): boolean
```

validation2

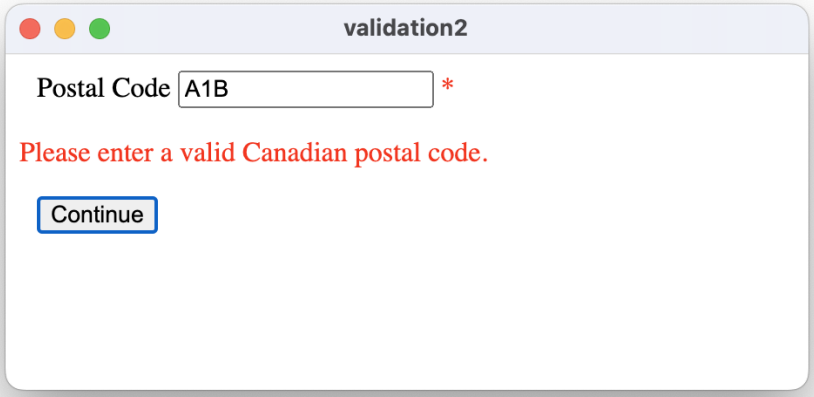
- `novalidate` attribute for form
  - to turn off built-in HTML validation messages
- Error message in p tag

```
<p id="pcode-error" class="error">Error message</p>
```

- CSS uses `active` class to show error message in p tag, e.g.

```
classList.add("active");
```

- main.ts sets up two listeners
  - form "submit" event when form button is pressed
  - postal code field "input" event when user changes text
- Constraint API usage
  - `validity.valid`
  - `validity.valueMissing`
  - `validity.patternMismatch`
- `onSubmitOnly` flag



https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation#the_constraint_validation_api

Custom Validation without Built-in API

- For custom input widgets, you must write a custom validator
  - create classes for invalid, etc.
  - listen to input event for custom widget
  - test against conditions (usually a regex)

https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation#validating_forms_without_a_built-in_api

Input Formatting and Masking

- When form text is formatted as it's typed
- *Input formatting* updates the string in textfield as user types
  - "input" event listener re-writes textfield with standard formatting

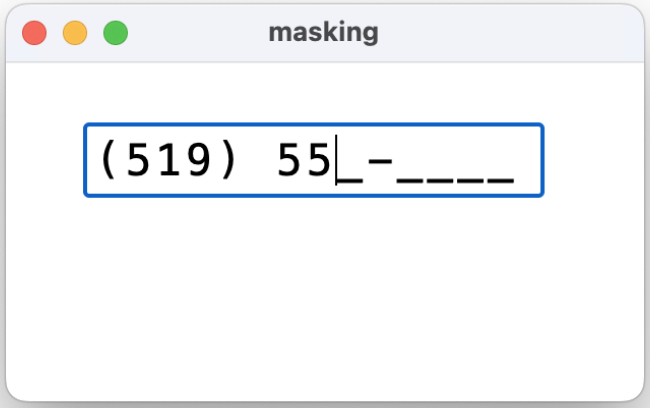
User types: `519 5555` ***Textfield displays:*** +1 (519) 555-5

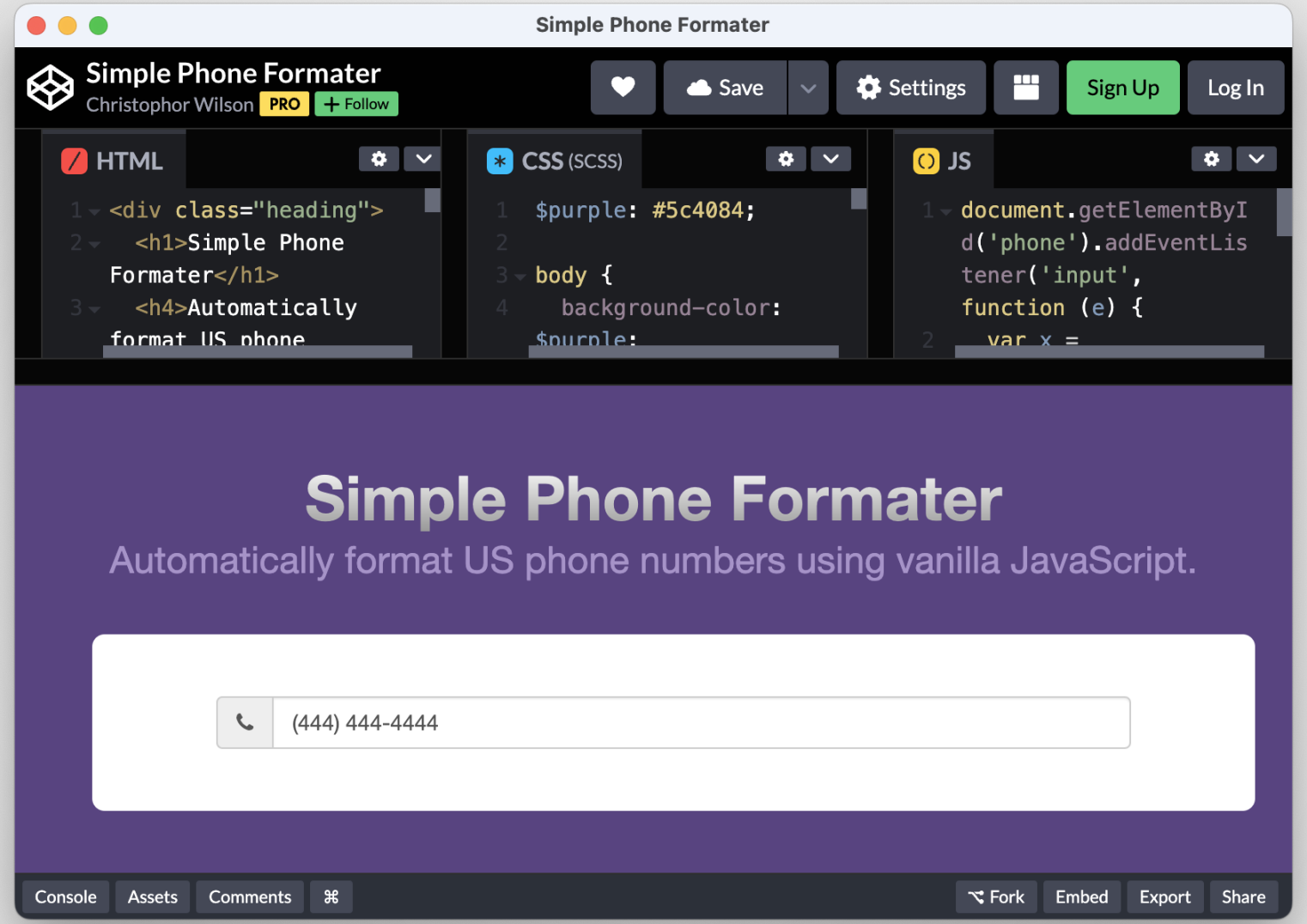
- *Input masking* provides a graphical representation of the final format and fills it in as the user types
  - Input handler re-writes textfield with formatted placeholders
  - More elaborate formatting possible with custom element

User types: `5195555` ***Textfield displays:*** `+1 (519) 555-5___`

masking

- Uses monospace font with letter-spacing
- Filter non-numbers
- Max 10 numbers
- Build string of numbers and blanks
- Set cursor position
  - Doesn't handle many edge cases





https://codepen.io/CSWApps/pen/EZxwMY