Beyond Javascript

Using the features of tomorrow

Spread syntax

"Spread syntax allows an iterable such as an array expression or string to be expanded in places where zero or more arguments (for function calls) or elements (for array literals) are expected, or an object expression to be expanded in places where zero or more key-value pairs (for object literals) are expected."

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_syntax

```
const arr = [
  'one',
  'two',
  'three',
];

console.log(arr[0], arr[1], arr[2]);
```

```
const arr = [
  'one',
  'two',
  'three',
];

console.log(arr[0], arr[1], arr[2]);
```

```
console.log.apply(console, arr);
```

```javascript
const arr = [
  'one',
  'two',
  'three',
];

console.log(arr[0], arr[1], arr[2]);
```

```javascript
console.log(...arr);
```

Rest syntax

Reverses the effect of the spread operator to collect all other (the rest) of the arguments into a single variable.

```
let pokedex = {
  "Ash": [],
  "Gary": ["Machoke"],
}

function catchem(trainer, ...pokemon) {
  pokemon.forEach((p) => pokedex[trainer].push(p));

  console.log(trainer, "has now caught", pokedex[trainer])
}

catchem('Ash', 'Pikachu');
> Ash has now caught ["Pikachu"]
```

```
let pokedex = {
  "Ash": [],
  "Gary": ["Machoke"],
}

function catchem(trainer, ...pokemon) {
  pokemon.forEach((p) => pokedex[trainer].push(p));

  console.log(trainer, "has now caught", pokedex[trainer])
}

catchem('Ash', 'Pikachu');
> Ash has now caught ["Pikachu"]
```

```
catchem('Gary', 'Charmander', 'Squirtle', 'Bulbasaur');
> Gary has now caught ["Machoke", "Charmander", "Squirtle", "Bulbasaur"]
```

```
let pokedex = {
  "Ash": [],
  "Gary": ["Machoke"],
}

function catchem(trainer, ...pokemon) {
  pokemon.forEach((p) => pokedex[trainer].push(p));

  console.log(trainer, "has now caught", pokedex[trainer])
}

catchem('Ash', 'Pikachu');
> Ash has now caught ["Pikachu"]
```

```
catchem('Gary', 'Charmander', 'Squirtle', 'Bulbasaur');
> Gary has now caught ["Machoke", "Charmander", "Squirtle", "Bulbasaur"]
```

```
const fibonacci = [ 1, 1, 2, 3, 5, 8, 13, 21 ];

const [ first, second, third, ...others] = fibonacci;

console.log("Starts like", first, second, third);
console.log("Then goes", others);
> Starts like 1 1 2
> Then goes [3, 5, 8, 13, 21]
```

Works mostly everywhere except Internet Explorer.

String Literals

String Literals

```
console.log("Hello, World");
```

```
> Hello World
```

String Literals

```javascript
const greeting = "John";
console.log("Hello, " + greeting);
```

```
> Hello John
```

Template Literals

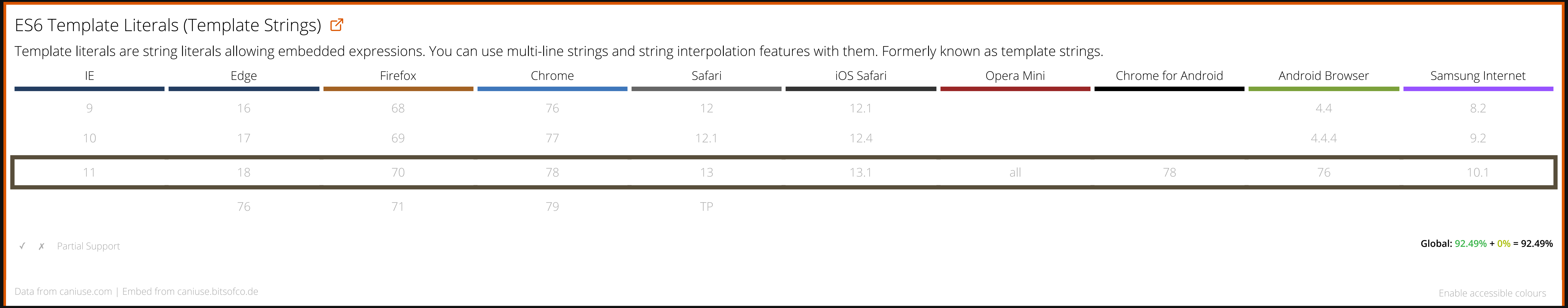
```
const greeting = "John";
console.log(`Hello, ${greeting}`);
```

```
> Hello John
```

Template Literals

```
const greeting = "John";
console.log(`Hello, ${greeting}`);
```

```
> Hello John
```



ES6 Template Literals (Template Strings)

Template literals are string literals allowing embedded expressions. You can use multi-line strings and string interpolation features with them. Formerly known as template strings.

IE	Edge	Firefox	Chrome	Safari	iOS Safari	Opera Mini	Chrome for Android	Android Browser	Samsung Internet
9	16	68	76	12	12.1			4.4	8.2
10	17	69	77	12.1	12.4			4.4.4	9.2
11	18	70	78	13	13.1	all	78	76	10.1
	76	71	79	TP					

✓ ✗ Partial Support

Global: 92.49% + 0% = 92.49%

Data from caniuse.com | Embed from caniuse.bitsofco.de

Enable accessible colours

Translations

```
/**
 * Translates a string and escapes arguments.
 *
 * @param string str
 *   The string with placeholders ({n}) to translate.
 * @param array args
 *   The arguments to replace in the translated string.
 *
 * @return string
 *   A translated string with escaped arguments.
 */
function t(str, args) {
  let translated = loadTranslation(str);

  for (const [key, value] of Object.entries(args)) {
    translated = translated.replace(`{${k}}`, htmlEscape(value));
  }

  return translated;
}
```

Translations

```
/**
 * Translates a string and escapes arguments.
 *
 * @param string str
 *   The string with placeholders ({n}) to translate.
 * @param array args
 *   The arguments to replace in the translated string.
 *
 * @return string
 *   A translated string with escaped arguments.
 */
function t(str, args) {
  let translated = loadTranslation(str);

  for (const [key, value] of Object.entries(args)) {
    translated = translated.replace(`{${k}}`, htmlEscape(value));
  }

  return translated;
}
```

```
t("Hello, {0}", "Javascript");
> Hoi, Javascript
```

Translations

```
/**
 * Translates a string and escapes arguments.
 *
 * @param string str
 *   The string with placeholders ({n}) to translate.
 * @param array args
 *   The arguments to replace in the translated string.
 *
 * @return string
 *   A translated string with escaped arguments.
 */
function t(str, args) {
  let translated = loadTranslation(str);

  for (const [key, value] of Object.entries(args)) {
    translated = translated.replace(`{${k}}`, htmlEscape(value));
  }

  return translated;
}
```

```
t("Hello, {0}", "Javascript");
> Hoi, Javascript
```

```
t("On a scale from {0} to {1}, what do you think of {2}?", 1, 10, t("French Fries"));
> Op een schaal van 1 to 10, wat vindt u van Franse frietjes?
```

Translations

```
/**
 * Translates a string and escapes arguments.
 *
 * @param string str
 *   The string with placeholders ({n}) to translate.
 * @param array args
 *   The arguments to replace in the translated string.
 *
 * @return string
 *   A translated string with escaped arguments.
 */
function t(str, args) {
  let translated = loadTranslation(str);

  for (const [key, value] of Object.entries(args)) {
    translated = translated.replace(`{${k}}`, htmlEscape(value));
  }

  return translated;
}
```

```
t("Hello, {0}", "Javascript");
> Hoi, Javascript
```

```
t("On a scale from {0} to {1}, what do you think of {2}?", 1, 10, t("French Fries"));
> Op een schaal van 1 to 10, wat vindt u van Franse frietjes?
```

```
t(`On a scale from ${1} to ${10}, what do you think of ${t("French Fries")}?`);
> On a scale from 1 to 10, what do you think of Franse frietjes?
```

```
/**
 * Retrieves the translation for a given string.
 *
 * @param string str
 *   The string to translate
 * @return string
 *   The translated string or the original string
 *   if no translation is available.
 */
function loadTranslation(str) {
  const language = window.navigator.userLanguage
                  || window.navigator.language;
  return
    str in translations && language in translations[str]
      ? translations[str][language]
      : str;
}
```

```
const translations = {
  "Hello World!": {
    "en-US": "Hello America!",
    "en-GB": "Hello World!",
    "nl": "Hallo Wereld!",
  },
  "Hello, {0}": {
    "en-US": "Howdy, {0}",
    "en-GB": "Hello, {0}",
    "nl": "Hoi, {0}",
  },
  "Do you like {0}?": {
    "en-US": "Who doesn't like football?",
    "en-GB": "Do you like {0}?",
    "nl": "Houd jij van {0}?",
  },
  "On a scale from {0} to {1}, what do you think of {2}?":
    "en-US": "On a scale from {0} to {1}, what do you think of {2}?",
    "en-GB": "On a scale from {0} to {1}, what do you think of {2}?",
    "nl": "Op een schaal van {0} to {1}, wat vindt u van {2}?",
  },
  "French Fries": {
    "en-US": "French Fries",
    "en-GB": "Chips",
    "nl": "Franse frietjes",
  },
};
```

```
/**
 * Uses a fake element to escape HTML entities.
 */
function htmlEscape(str) {
  let d = document.createElement('div');
  d.innerHtml = str;
  return d.innerText;
}
```

Tagged Template Literals

```
`On a scale from ${1} to ${10}, what do you think of ${     `French Fries`}?`
```

Tagged Template Literals

```
trans`On a scale from ${1} to ${10}, what do you think of ${trans`French Fries`}?`
```

```
function trans() {
  console.log(arguments);
}
```

Tagged Template Literals

```
trans`On a scale from ${1} to ${10}, what do you think of ${trans`French Fries`}?`
```

```
function trans() {
  console.log(arguments);
}
```

```
> [Arguments] { '0': [ 'French Fries' ] }
> [Arguments] {
  '0': [ 'On a scale from ', ' to ', ', what do you think of ', '?' ],
  '1': 1,
  '2': 10,
  '3': 'French Fries'
}
```

```
> [Arguments] {
'0': [ 'On a scale from ', ' to ', ', what do you think of ', '?' ],
'1': 1,
'2': 10,
'3': 'French Fries'
}
```

```
/**
* Translates a string and escapes arguments.
*







* @return string
*   A translated string with escaped arguments.
*/
function trans(literals,    args) {








  let translated = loadTranslation(literals);


  for (const [key, value] of Object.entries(args)) {
    translated = translated.replace(`{${k}}`, htmlEscape(value));
  }

  return translated;
}
```

```
> [Arguments] {
'0': [ 'On a scale from ', ' to ', ', what do you think of ', '?' ],
'1': 1,
'2': 10,
'3': 'French Fries'
}
```

```
/**
* Translates a string and escapes arguments.
*
* @param array literals
*   An array of string literals that surround the expressions
* @param {...} args
*   The arguments to replace in the translated string.
*
* @return string
*   A translated string with escaped arguments.
*/
function trans(literals, ...args) {
  // Add our placeholders for the translation loading.
  // idx starts at 0 if an initial value is provided to reduce.
  const translatableString = strings.slice(1).reduce(
    (acc, val, idx) => `${acc} {${idx}} ${val}`,
    strings[0]
  );


  let translated = loadTranslation(translatableString);

  for (const [key, value] of Object.entries(args)) {
    translated = translated.replace(`{${k}}`, htmlEscape(value));
  }

  return translated;
}
```

```
function trans(literals, ...args) {
  // Add our placeholders for the translation loading.
  // idx starts at 0 if an initial value is provided to reduce.
  const translatableString = strings.slice(1).reduce(
    (acc, val, idx) => `${acc} {${idx}} ${val}`,
    strings[0]
  );

  let translated = loadTranslation(translatableString);

  for (const [key, value] of Object.entries(args)) {
    translated = translated.replace(`{${k}}`, htmlEscape(value));
  }

  return translated;
}
```

```
// Alternate our rating type.
const getRateType = (() => { let i = 0; return () => ++i % 2 ? 'food' : 'drink'; })();
```

```
trans`On a scale from ${1} to ${10}, what do you think of ${getRateType() === 'food' ? trans`French Fries` : 'Coca Cola'}?`
> Op een schaal van 1 tot 10, wat vindt u van Franse frietjes?
```

```
trans`On a scale from ${1} to ${10}, what do you think of ${getRateType() === 'food' ? trans`French Fries` : 'Coca Cola'}?`
> Op een schaal van 1 tot 10, wat vindt u van Coca Cola?
```

Works out of the box in Chrome

Internet Explorer?

Unsupported function? Use `Modernizr!`

Unsupported function? Use `Modernizr!`

```
console.log(...arguments);
> Syntax Error
```

Unsupported function? Use ~~Modernizr!~~

```
console.log(...arguments);
> Syntax Error
```

Unsupported ~~function?~~ Syntax? Use ~~Modernizr!~~

```
console.log(...arguments);
> Syntax Error
```

Unsupported ~~function?~~ Syntax? Use ~~`Modernizr!`~~ `Babel!`

```
console.log(...arguments);
> Syntax Error
```

Babel is a JavaScript compiler.

Put in next-gen JavaScript. Get browser-compatible JavaScript out

Babel is a JavaScript compiler.

Put in next-gen JavaScript. Get browser-compatible JavaScript out

```
console.log(...arguments);
```

Babel is a JavaScript compiler.

Put in next-gen JavaScript. Get browser-compatible JavaScript out

```
console.log(...arguments);
```

```
var _console;

(_console = console).log.apply(_console, arguments);
```

Need a Babel configuration quick-start?

`@babel/preset-env` is for you.

- Use the latest JavaScript without micromanaging syntax transforms
- Use browserlist expressions

  ```
  > 0,25%, not dead
  ```

  Compile to work in any browser with more than 0,25% marketshare (according to Can I Use)

Some next-gen JavaScript

Decorators

Decorators

- A proposal for ESNext by Ecma Technical Committee (TC) 39
- https://github.com/tc39/proposal-decorators/

Decorators

- A proposal for ESNext by Ecma Technical Committee (TC) 39
- https://github.com/tc39/proposal-decorators/
- Makes metaprogramming in JavaScript more explicit
- Replaces code such as `Object.defineProperty`

Decorators

- A proposal for ESNext by Ecma Technical Committee (TC) 39
- https://github.com/tc39/proposal-decorators/
- Makes metaprogramming in JavaScript more explicit
- Replaces code such as `Object.defineProperty`
- Can also manipulate private fields/methods
- Manipulations run at time of definition
- Can be used for a whole class or an individual element

Why?

```javascript
class Counter {
  count = 0;





  handleClick() {
    this.count++;
    console.log(this, this.count);
  }

  render() {
    let button = document.createElement('button');
    button.innerText = "Increment";
    button.onclick = this.handleClick;
    document.body.append(button);
  }
}

let counter = new Counter();
counter.render();
```

Why?

```
class Counter {
  count = 0;






  handleClick() {
    this.count++;
    console.log(this, this.count);
  }

  render() {
    let button = document.createElement('button');
    button.innerText = "Increment";
    button.onclick = this.handleClick;
    document.body.append(button);
  }
}

let counter = new Counter();
counter.render();
```

```
> <button>Increment</button> NaN
```

Why?

```
class Counter {
  count = 0;

  constructor() {
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    this.count++;
    console.log(this, this.count);
  }

  render() {
    let button = document.createElement('button');
    button.innerText = "Increment";
    button.onclick = this.handleClick;
    document.body.append(button);
  }
}

let counter = new Counter();
counter.render();
```

```
> Counter {count: 1, handleClick: ƒ} 1
> Counter {count: 2, handleClick: ƒ} 2
```

Why?

```
class Counter {
  count = 0;




  @bound
  handleClick() {
    this.count++;
    console.log(this, this.count);
  }

  render() {
    let button = document.createElement('button');
    button.innerText = "Increment";
    button.onclick = this.handleClick;
    document.body.append(button);
  }
}

let counter = new Counter();
counter.render();
```

```
> Counter {count: 1, handleClick: ƒ} 1
> Counter {count: 2, handleClick: ƒ} 2
```

How?

The documentation[1] is better than my funny code snippets.

1. https://github.com/tc39/proposal-decorators/blob/master/METAPROGRAMMING.md#basic-usage

A decorator is a function. Usable in three places

```
/**
 * A counter element.
 *
 * Use in your HTML
 */

class Counter extends HTMLElement {

  count = 0;

  constructor() {
    super();
    this.onclick = this.handleClick;
  }

  connectedCallback() { this.render(); }


  handleClick() {
    this.count++;
  }


  render() {
    this.textContent = this.count.toString();
  }
}
```

A decorator is a function. Usable in three places

- Class

```
/**
 * A counter element.
 *
 * Use in your HTML
 */
@defineElement
class Counter extends HTMLElement {

  count = 0;

  constructor() {
    super();
    this.onclick = this.handleClick;
  }

  connectedCallback() { this.render(); }


  handleClick() {
    this.count++;
  }


  render() {
    this.textContent = this.count.toString();
  }
}
```

A decorator is a function. Usable in three places

- Class

```
/**
 * A counter element.
 *
 * Use in your HTML as <num-counter />.
 */
@defineElement('num-counter')
class Counter extends HTMLElement {

  count = 0;

  constructor() {
    super();
    this.onclick = this.handleClick;
  }

  connectedCallback() { this.render(); }


  handleClick() {
    this.count++;
  }


  render() {
    this.textContent = this.count.toString();
  }
}
```

A decorator is a function. Usable in three places

- Class
- Property

```
/**
 * A counter element.
 *
 * Use in your HTML as <num-counter />.
 */
@defineElement('num-counter')
class Counter extends HTMLElement {
  @logged
  count = 0;

  constructor() {
    super();
    this.onclick = this.handleClick;
  }

  connectedCallback() { this.render(); }


  handleClick() {
    this.count++;
  }


  render() {
    this.textContent = this.count.toString();
  }
}
```

A decorator is a function. Usable in three places

- Class
- Property
- Method

```
/**
 * A counter element.
 *
 * Use in your HTML as <num-counter />.
 */
@defineElement('num-counter')
class Counter extends HTMLElement {
  @logged
  count = 0;

  constructor() {
    super();
    this.onclick = this.handleClick;
  }

  connectedCallback() { this.render(); }

  @bound
  handleClick() {
    this.count++;
  }

  @bound
  render() {
    this.textContent = this.count.toString();
  }
}
```

Method

```
// arguments
{
  kind: "method"
  key: String, Symbol or Private Name,
  placement: "static", "prototype" or "own",
  descriptor: Property Descriptor (argument to Object.defineProperty),

}

// return
{ kind, key, placement, descriptor,              , extras?, finisher? }
```

https://github.com/tc39/proposal-decorators/blob/master/METAPROGRAMMING.md#api

Field

```
// arguments
{
  kind:         "field"
  key: String, Symbol or Private Name,
  placement: "static", "prototype" or "own",
  descriptor: Property Descriptor (argument to Object.defineProperty),
  initializer: A method used to set the initial state of the field,
}

// return
{ kind, key, placement, descriptor, initializer, extras?, finisher? }
```

https://github.com/tc39/proposal-decorators/blob/master/METAPROGRAMMING.md#api

Field

```
// arguments
{
  kind:         "field"
  key: String, Symbol or Private Name,
  placement: "static", "prototype" or "own",
  descriptor: Property Descriptor (argument to Object.defineProperty),
  initializer: A method used to set the initial state of the field,
}

// return
{ kind, key, placement, descriptor, initializer, extras?, finisher? }
```

```
extras : [ { kind, key, placement, descriptor, initializer?}, ... ]
```

https://github.com/tc39/proposal-decorators/blob/master/METAPROGRAMMING.md#api

Field

```
// arguments
{
  kind:          "field"
  key: String, Symbol or Private Name,
  placement: "static", "prototype" or "own",
  descriptor: Property Descriptor (argument to Object.defineProperty),
  initializer: A method used to set the initial state of the field,
}

// return
{ kind, key, placement, descriptor, initializer, extras?, finisher? }
```

```
extras : [ { kind, key, placement, descriptor, initializer?}, ... ]
```

https://github.com/tc39/proposal-decorators/blob/master/METAPROGRAMMING.md#api

Property descriptor documentation: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/defineProperty#Description

Class

```
 // arguments
{
  kind: "class"
  elements: Array of all class elements
}

// return
{
  kind: "class"
  elements: Possibly modified class elements (can include additional class elements)
  finisher: (optional) A callback that is called at the end of class creation
}
```

Logging property changes

```
class Counter {
  @logged
  count = 0;
}

let counter1 = new Counter();
counter1.count++;
counter1.count++;
counter1.count++;

let counter2 = new Counter();
counter2.count++;
counter2.count++;
```

```javascript
// Logs the new value of a variable to the console whenever it changes.
function logged({kind, key, placement, descriptor, initializer}) {
  assert(kind == "field");
  assert(placement == "own");
  // Create a new private name as a key for a class element
  let storage = Symbol(key);
  let underlyingDescriptor = { enumerable: false, configurable: false, writable: true };
  let underlying = { kind, key: storage, placement, descriptor: underlyingDescriptor, initializer };
  return {
    kind: "method",
    key,
    placement,
    descriptor: {
      get() { return this[storage]; },
      set(value) {
        this[storage] = value;
        // Log the change of value
        console.log("Value of", key, "changed to", value);
      },
      enumerable: descriptor.enumerable,
      configurable: descriptor.configurable
    },
    extras: [underlying]
  };
}
```

```
Value of count changed to 1
Value of count changed to 2
Value of count changed to 3
Value of count changed to 1
Value of count changed to 2
```

This won't even run in Chrome (yet)

Babel to the rescue

- @babel/plugin-proposal-decorators
- @babel/plugin-proposal-class-properties

Note: In your config add the classProperties plugin after decorators.

`@babel/plugin-proposal-*` **vs** `@babel/plugin-syntax-*`

Have fun!

With great power comes great responsibility.