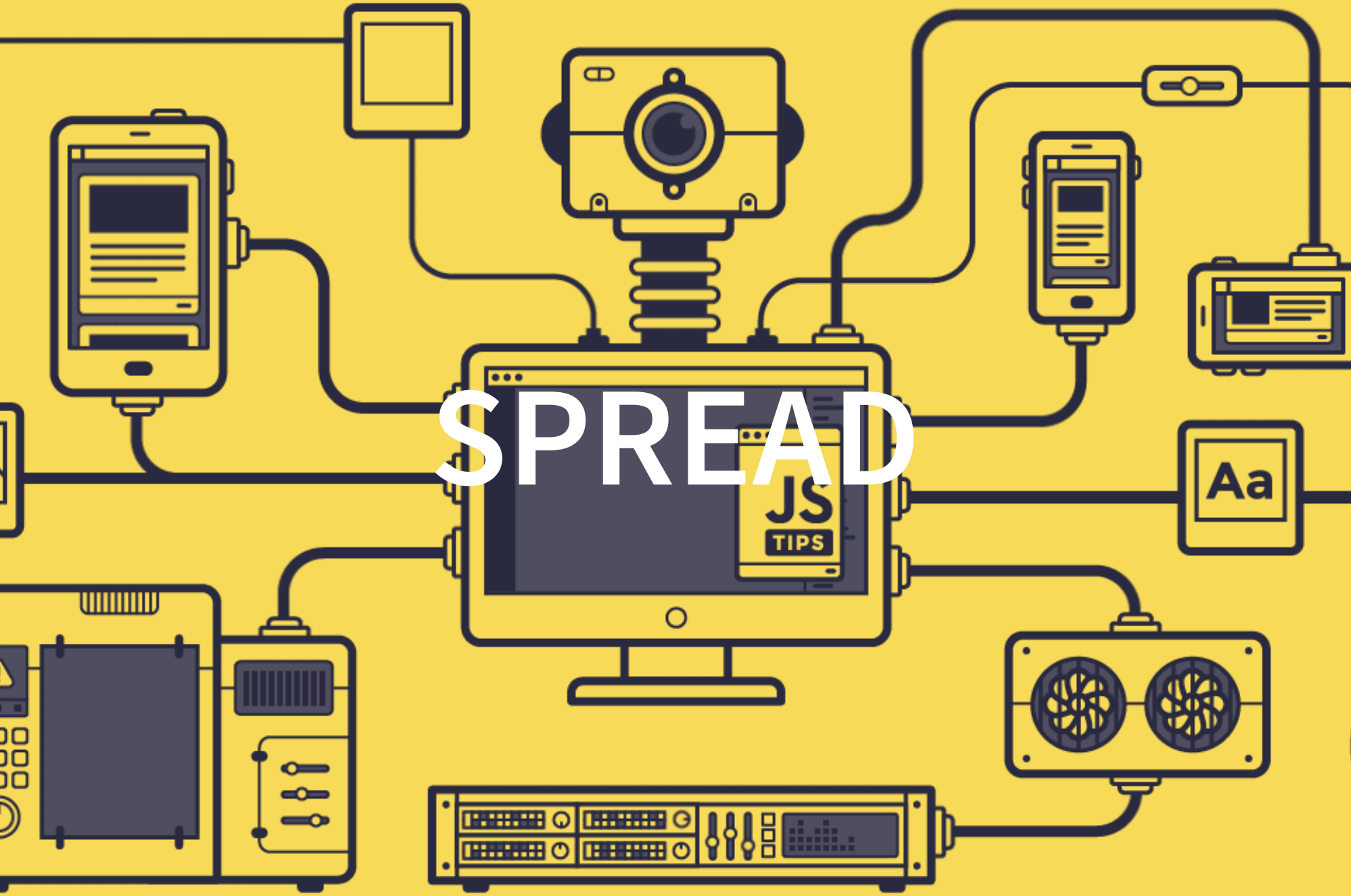
SPREAD
JS
TIPS
Aa

WHAT IS IT?

Operator which forces subject to iterate

...

MOST COMMON USAGE

In functions

```
function func (dataFirst, dataNext, dataSome) {
    console.log(dataSome); // 'dataSome'
}

func(...['dataFirst', 'dataNext', 'dataSome']);
```

In array literals

```
function func (dataFirst, dataNext, dataSome) {
    console.log(dataSome); // 'my friend'
}

const literal = ['my friend'];
func(...['dataFirst', 'dataNext', ...literal]);
```

In array literals to copy

```
1 const original = [12, 13];
2 const copyOfOriginal = [...original];
```

In array literals to copy but NOT deep

```
1 const a = [12, 13];
2 const b = [a, 14, 15];
3 const c = [...b];
4 c[0] === b[0] // true
```

In array literals to copy but NOT deep

```
1 const a = [12, 13];
2 const b = [a, 14, 15];
3 const c = [...b];
4 c[0] === b[0] // true
```

In string literals

```
1 const hiGuys = 'greet';
2 [...hiGuys] // ["g", "r", "e", "e", "t"]
```

In object literals

```
1 const obj = {prop: true, and: 1, you: []};
2 const result = {...obj, good: false};
3 // {prop: true, and: 1, you: [], good: false}
4 result.you === obj.you;
5 // true
```

In object literals – new object is created

```
1 const obj = {prop: true, and: 1, you: []};
2 const result = {...obj, you: [123]};
3 // result does not have obj `prototype`
4 result.you !== obj.you;
5 // true
6 result.you
7 // [123]
```

In object literals – new object is created

```
1 const obj = {prop: true, and: 1, you: []};
2 const result = {...obj, you: [123]};
3 // result does not have obj `prototype`
4 result.you !== obj.you;
5 // true
6 result.you
7 // [123]
```

In object literals – new object is created

```
1 const obj = {prop: true, and: 1, you: []};
2 const result = {...obj, you: [123]};
3 // result does not have obj `prototype`
4 result.you !== obj.you;
5 // true
6 result.you
7 // [123]
```

In object literals – new object is created

```
1 const obj = {prop: true, and: 1, you: []};
2 const result = {...obj, you: [123]};
3 // result does not have obj `prototype`
4 result.you !== obj.you;
5 // true
6 result.you
7 // [123]
```

In object literals – new object is created

```
1 const obj = {prop: true, and: 1, you: []};
2 const result = {...obj, you: [123]};
3 // result does not have obj `prototype`
4 result.you !== obj.you;
5 // true
6 result.you
7 // [123]
```

In object literals – does not run setters

```
const a = Object.defineProperty(
    {},
    'p',
    { set () { console.log(123) } }
);
const b = {...a, p: 2};
b.p === 2 // true;
const assigned = Object.assign(a, {p: 2});
// log: 123
assigned.p === undefined // true
```

Map, Set and DOM collections

```
[ ...new Set([1,2,3]) ]; // [1,2,3]
[ ...new Map([[1, 'one']]) ]; // [[1, 'one']]
[ ...document.querySelectorAll('div') ] // [ div, div, div]
```

IRREPLACEABLE SOMETIMES

```
1 const arr = [1, 2];
2 const resultArr = [...arr, 3, 4];
3 console.log(resultArr); // [1, 2, 3, 4]
```

```
1 const arr = [1, 2];
2 const resultArr = [...arr, 3, 4];
3 console.log(...resultArr); // 1 2 3 4
```

```
1 const arr = [1, 2];
2 const resultArr = [...arr, 3, 4];
3 console.log(...arr, ...resultArr); // 1 2 1 2 3 4
```

Could not be easily achieved before

```
1 const ES2018ReleaseDate = [2018, 5, 1];  // 1 Jun 2018
2 const d = new Date(...ES2018ReleaseDate);
```

ITERABLES

Works only with iterable entities

```
1 const obj = {P: 1};
2 [...obj] // TypeError: obj is not iterable
```

You can make anything iterable

```
const obj = {P: 1};
obj[Symbol.iterator] = function() {
  return {
    next: function() {
      if (!this._last) {
        this._last = true;
        return { value: ['P', 1], done: false };
      } else {
        return { done: true };
      }
    }
  };
};
[...obj] // ["P", 1]
```

@@ITERATOR METHOD

- String
- Array
- TypedArray
- Map
- Set

DESTRUCTING

```
let myArray = [1,2,3];
let [a, b, c] = myArray;

console.log(a); // 1
console.log(b); // 2
console.log(c); // 3
```

```
let myArray = [1, 2, 3, 4, 5];
let [a, b, c, ...d] = myArray;

console.log(a); // 1
console.log(b); // 2
console.log(c); // 3
console.log(d); // [4, 5]
```

```
let myObject = { a: 1, b: 2, c: 3, d: 4};
let {b, d, ...remaining} = myObject;

console.log(b); // 2
console.log(d); // 4
console.log(remaining); // { a: 1, c: 3 }
```