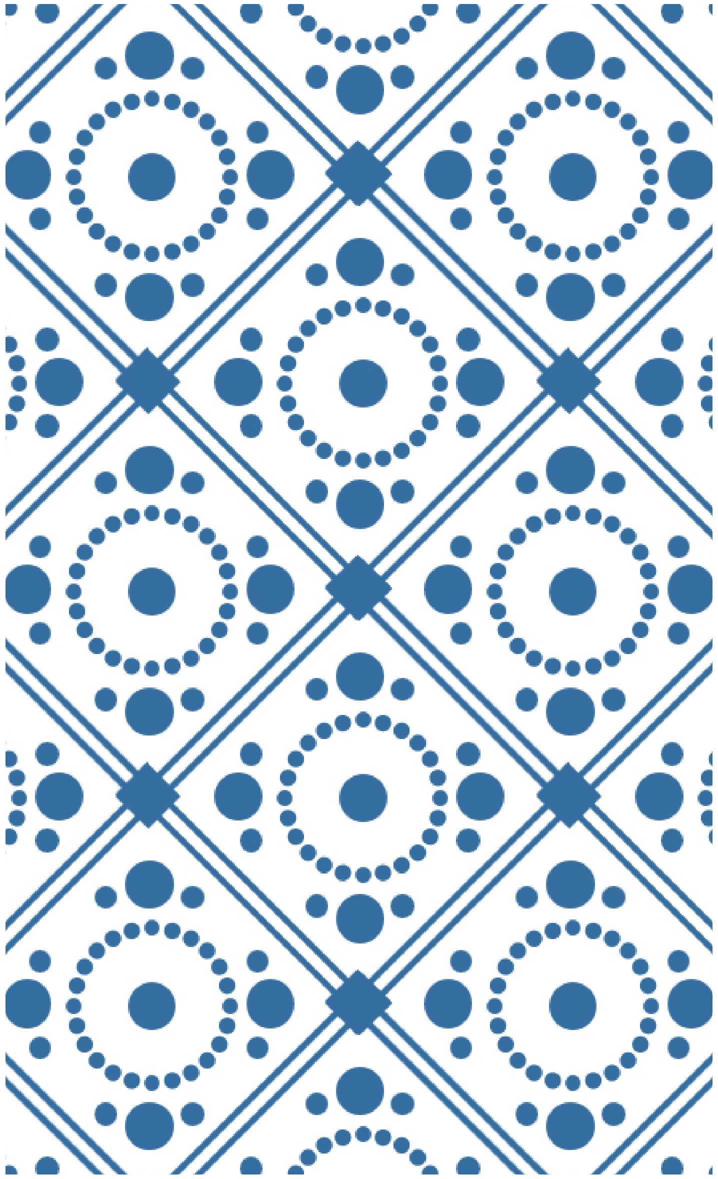
DATA: SPREAD OPERATOR

Andrew Sheehan

Boston University
Computer Science/MET



THE SPREAD OPERATOR WAS INTRODUCED WITH ES6

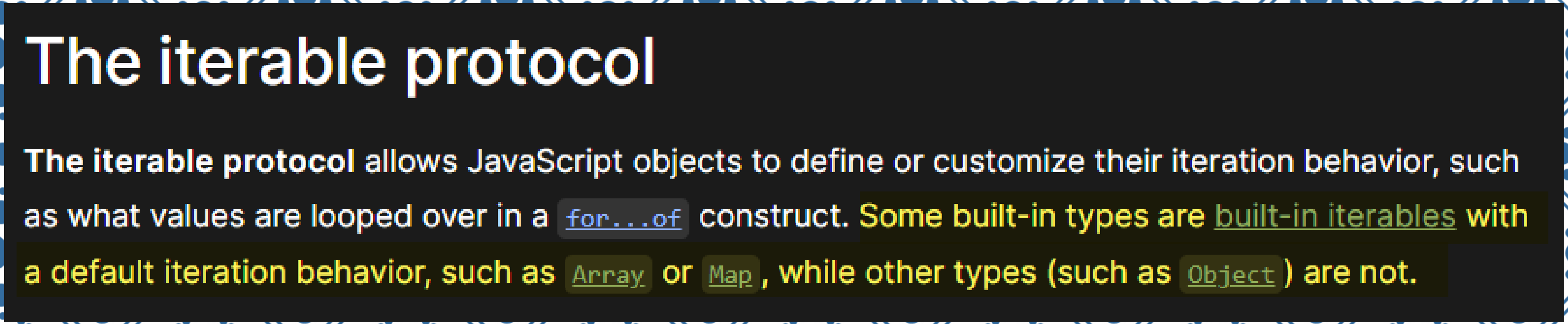
It "spreads" the values across one or more arguments

SPREAD OPERATOR:

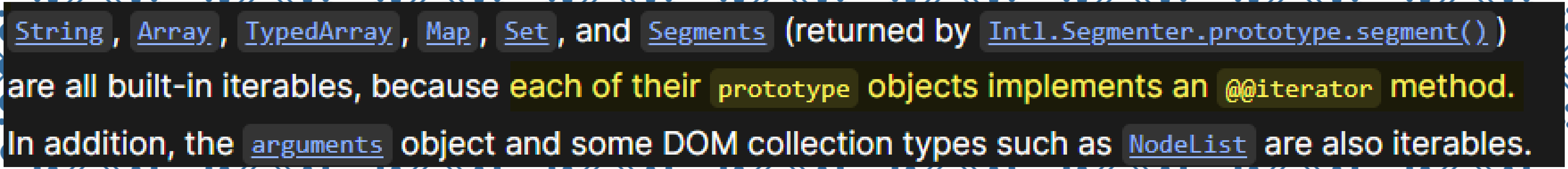
TURNS THE ITEMS OF AN ITERABLE INTO ARGUMENTS OF A FUNCTION CALL OR INTO ELEMENTS OF AN ARRAY.

REST OPERATOR:

COLLECTS THE REMAINING ITEMS OF AN ITERABLE INTO AN ARRAY AND IS USED FOR REST PARAMETERS AND DESTRUCTURING.


The iterable protocol

The iterable protocol allows JavaScript objects to define or customize their iteration behavior, such as what values are looped over in a `for...of` construct. Some built-in types are built-in iterables with a default iteration behavior, such as `Array` or `Map`, while other types (such as `Object`) are not.


`String`, `Array`, `TypedArray`, `Map`, `Set`, and `Segments` (returned by `Intl.Segmenter.prototype.segment()`) are all built-in iterables, because each of their `prototype` objects implements an `@@iterator` method. In addition, the `arguments` object and some DOM collection types such as `NodeList` are also iterables.

WHAT IS ITERABLE

SPREAD MORE DETAIL

In a spread operation, you're trying to take an array/object and place it's items into a new array or object

SAID DIFFERENTLY: WHAT SPREAD DOES

The spread operator allows us to copy all or part of an existing array or object **into another**.

Similar to splice()

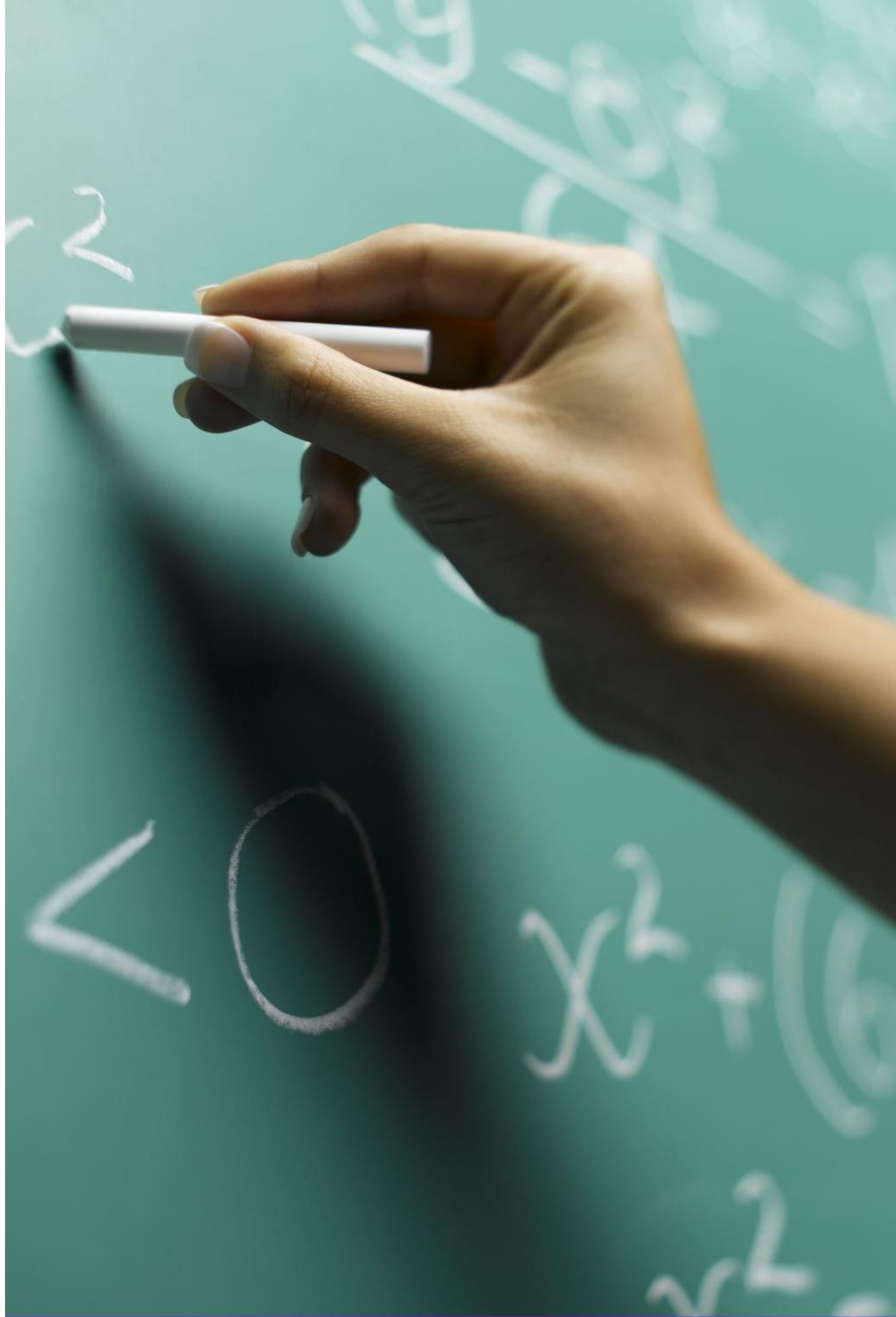
https://www.w3schools.com/jsref/jsref_splice.asp

Src: https://www.w3schools.com/react/react_es6_spread.asp

SPREAD: DATE EXAMPLE

```
let fields = [1980, 10, 23];

//spreads into constructor.
let birthdate = new Date(…fields);
```



MATH.MAX()

EXAMPLE

```
Ok: Math.max(-1, 5, 8, 11);

Ok: Math.max(…[-1,5,8,11]);
```

SPREAD WITH OBJECT LITERALS

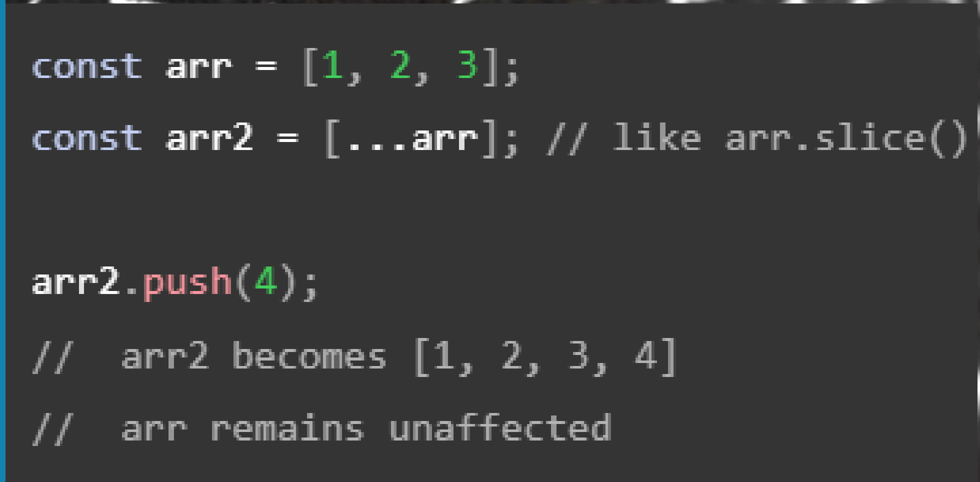
```
const user = {
  fullName: “Andrew Sheehan”,
  userId: “rush2112”
}
const pref = {
  band: “Pop Evil”,
  currentSong: “Trenches”,
  link: “https://www.youtube.com/watch?v=DWdtN7pCZug”
}
const account = {…user, …pref}
```

MAKING COPIES (1D ARRAY)

```
// this is a single dimensional array
let firstNames =
   [‘drew’, ‘je’, ‘pena’];

let copyOfFirstNames =

   […firstNames];
```

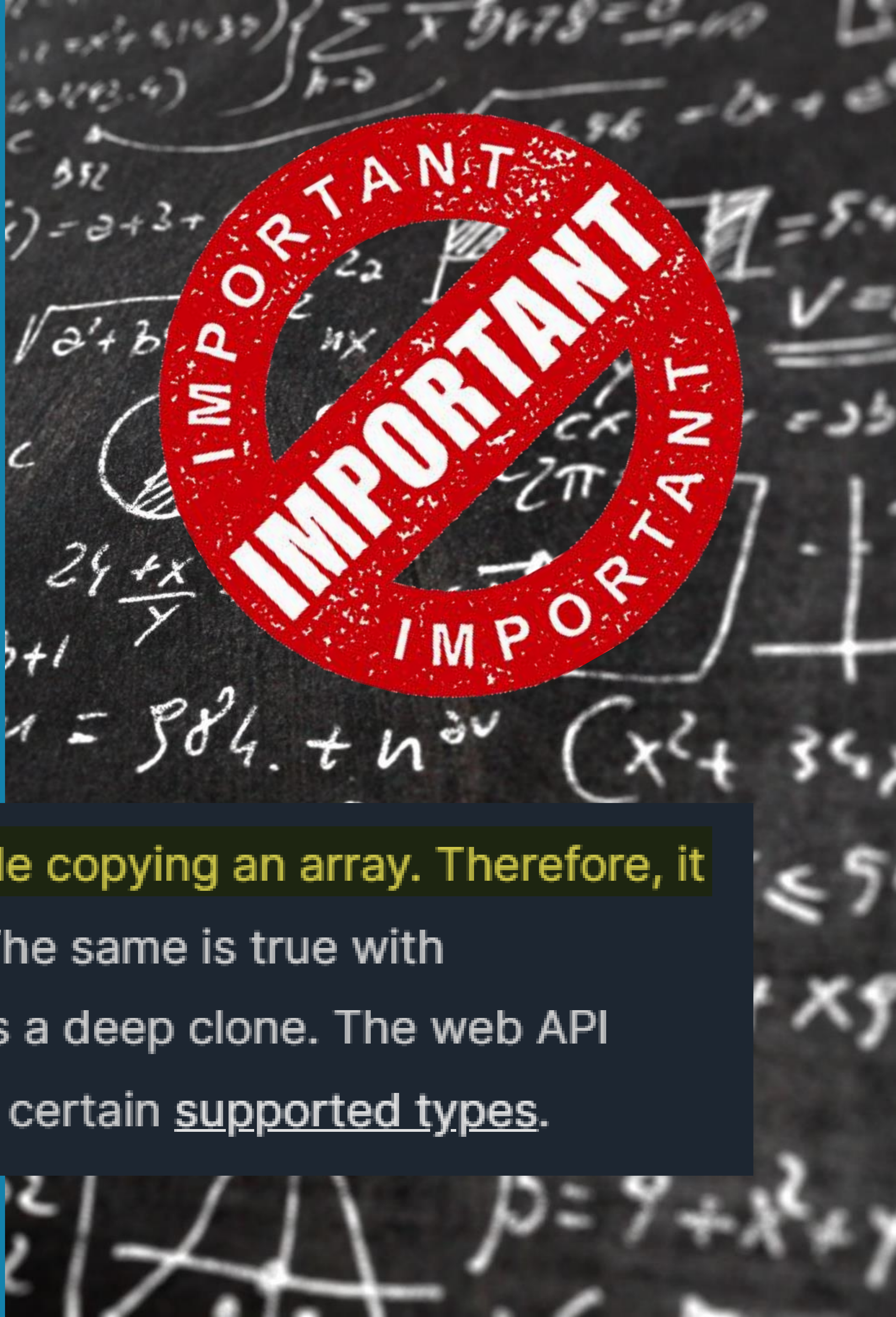


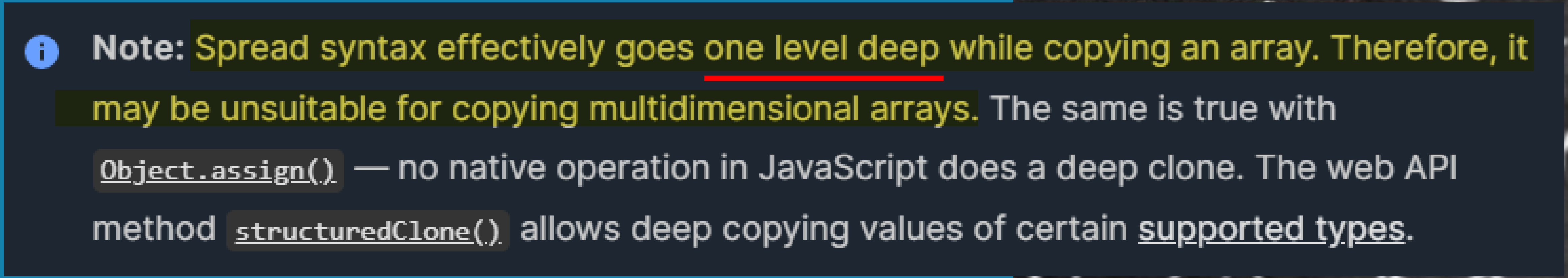
```
const arr = [1, 2, 3];
const arr2 = [...arr]; // like arr.slice()


arr2.push(4);
//  arr2 becomes [1, 2, 3, 4]
//  arr remains unaffected
```

MAKING ARRAY COPIES OF 1D ARRAY

```
let firstNames = ['drew', 'je', 'pena'];
let copyOfFirstNames = [...firstNames];
```



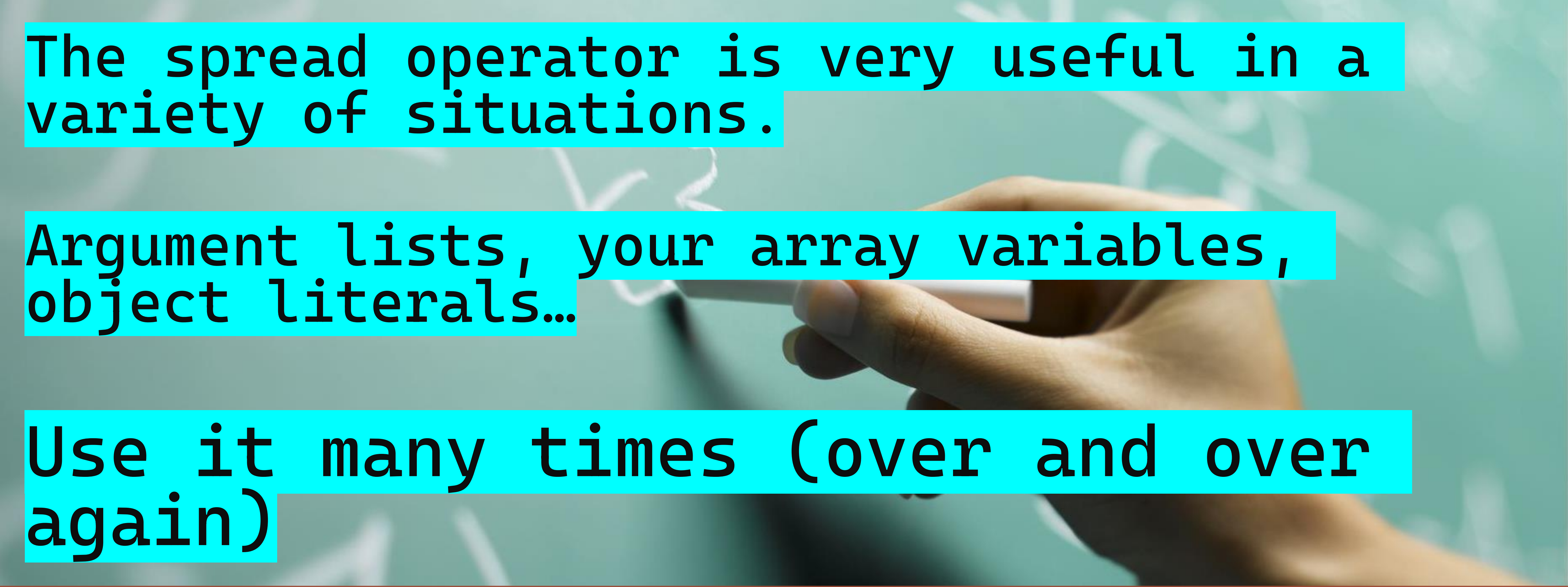


Note: Spread syntax effectively goes one level deep while copying an array. Therefore, it may be unsuitable for copying multidimensional arrays. The same is true with `Object.assign()` — no native operation in JavaScript does a deep clone. The web API method `structuredClone()` allows deep copying values of certain supported types.

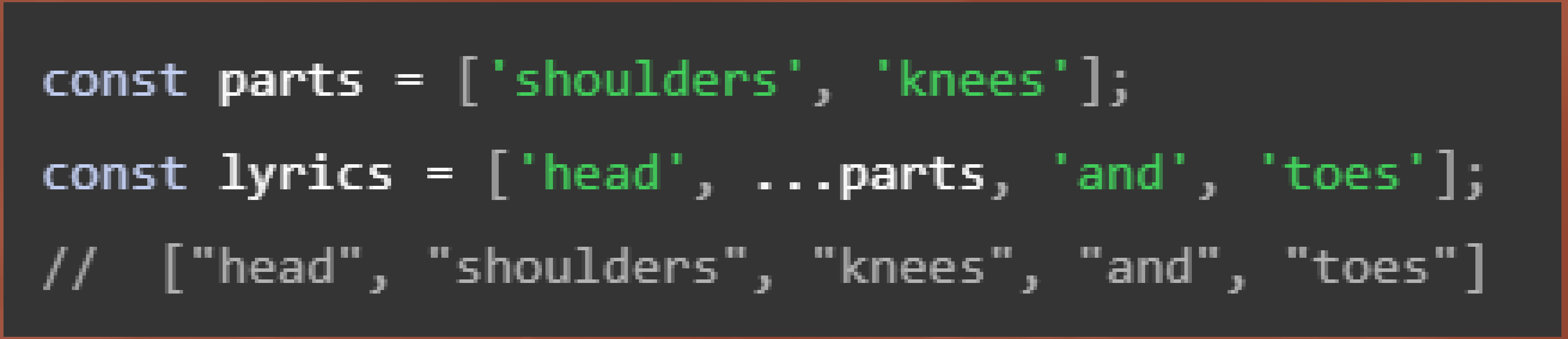
The spread operator is very useful in a variety of situations.

Argument lists, your array variables, object literals…

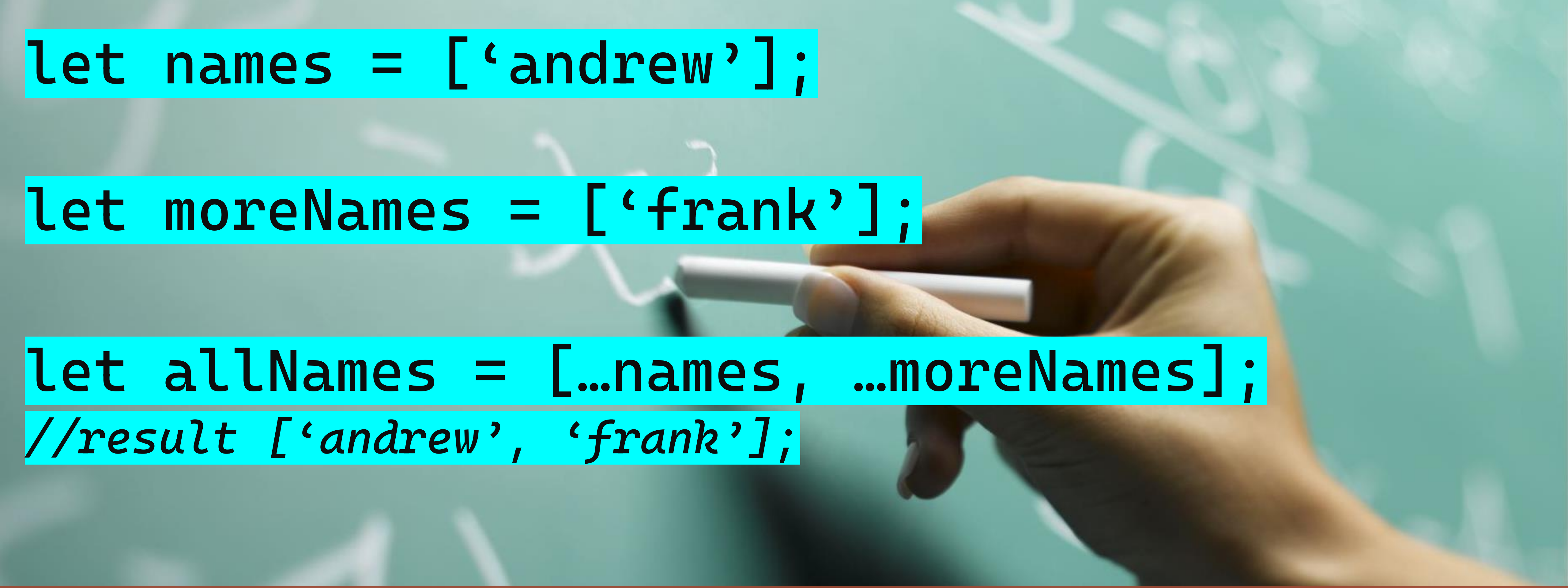
Use it many times (over and over again)



USE CASES



```
const parts = ['shoulders', 'knees'];
const lyrics = ['head', ...parts, 'and', 'toes'];
//  ["head", "shoulders", "knees", "and", "toes"]
```

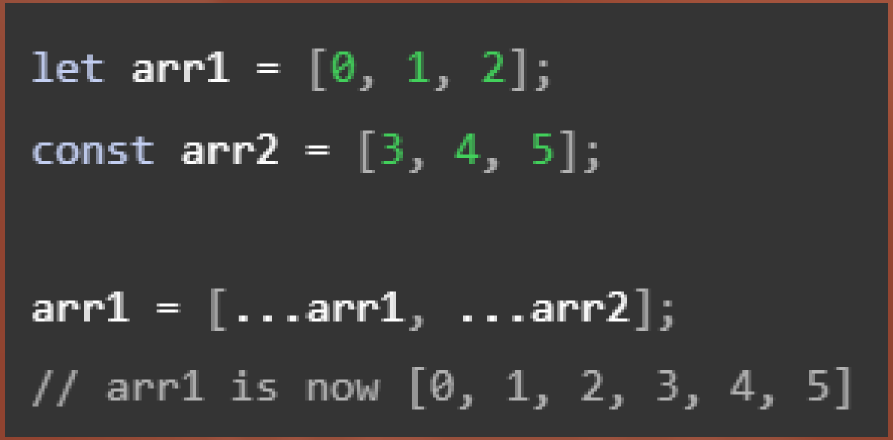


```
let names = [‘andrew’];

let moreNames = [‘frank’];

let allNames = […names, …moreNames];
//result [‘andrew’, ‘frank’];
```

JOINING ARRAYS



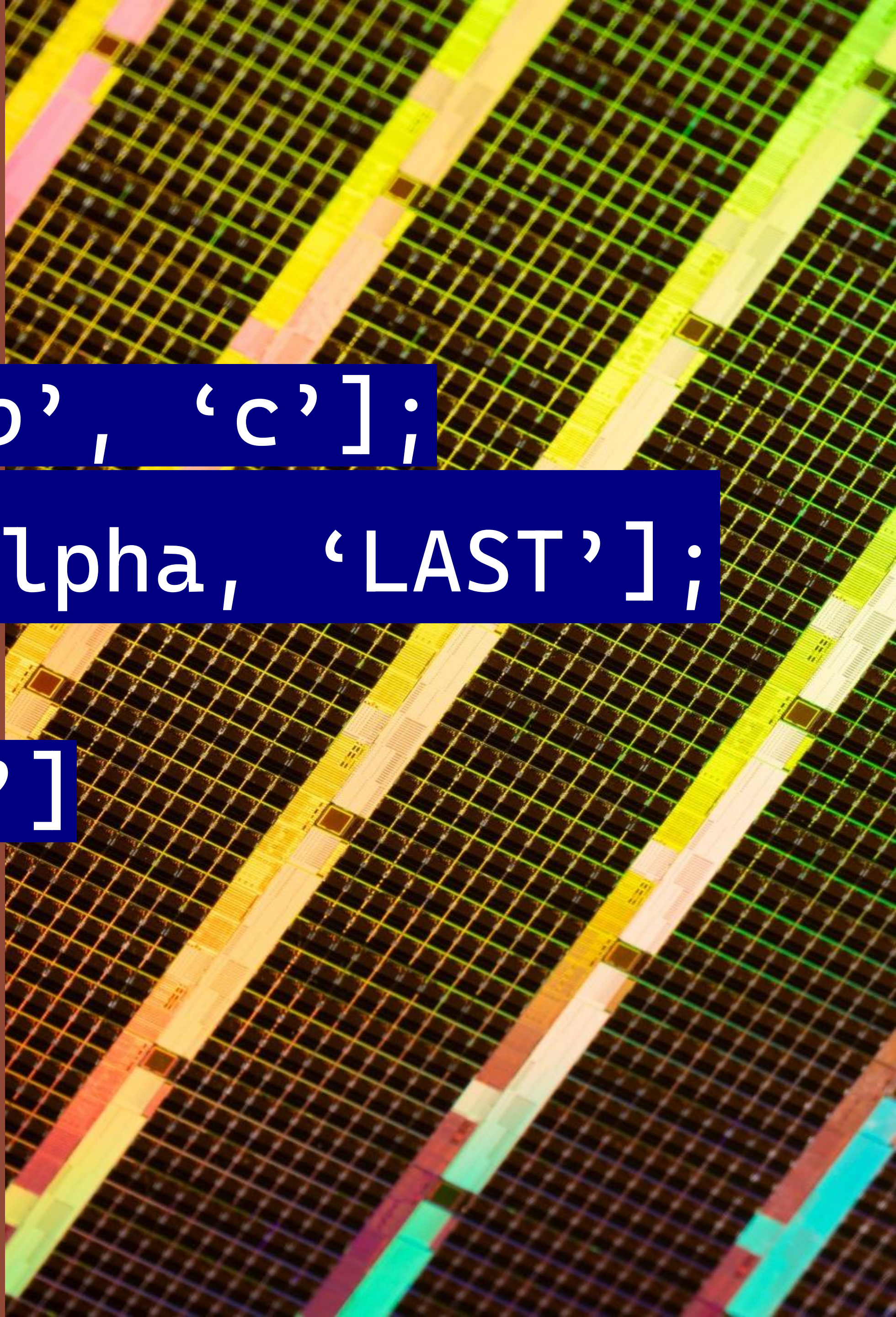
```
let arr1 = [0, 1, 2];
const arr2 = [3, 4, 5];


arr1 = [...arr1, ...arr2];
// arr1 is now [0, 1, 2, 3, 4, 5]
```

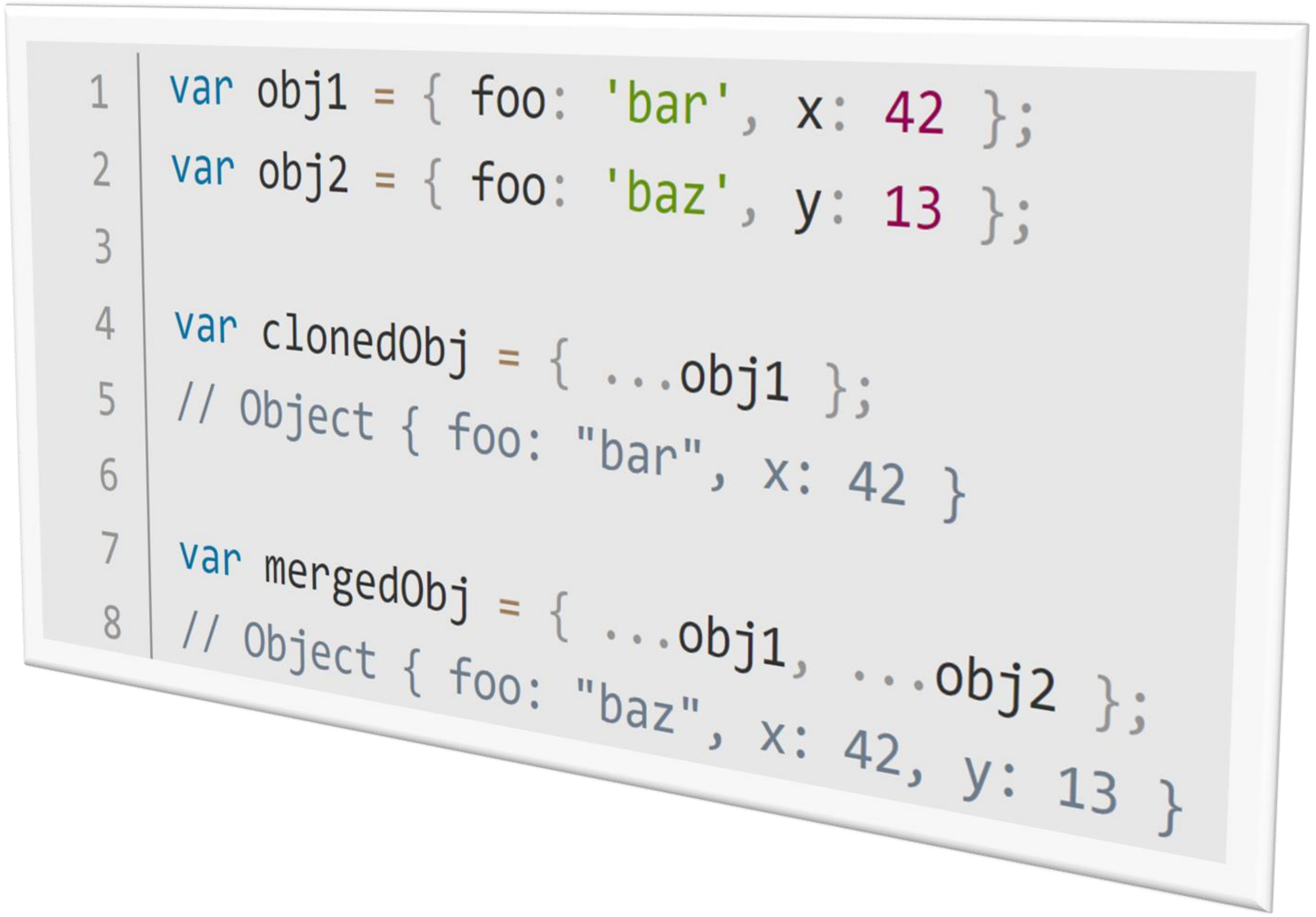
COPY THEN ADD

```
const alpha = [‘a’, ‘b’, ‘c’];
const moreAlpha = […alpha, ‘LAST’];

// [‘a’,’b’,’c’,’LAST’]
```



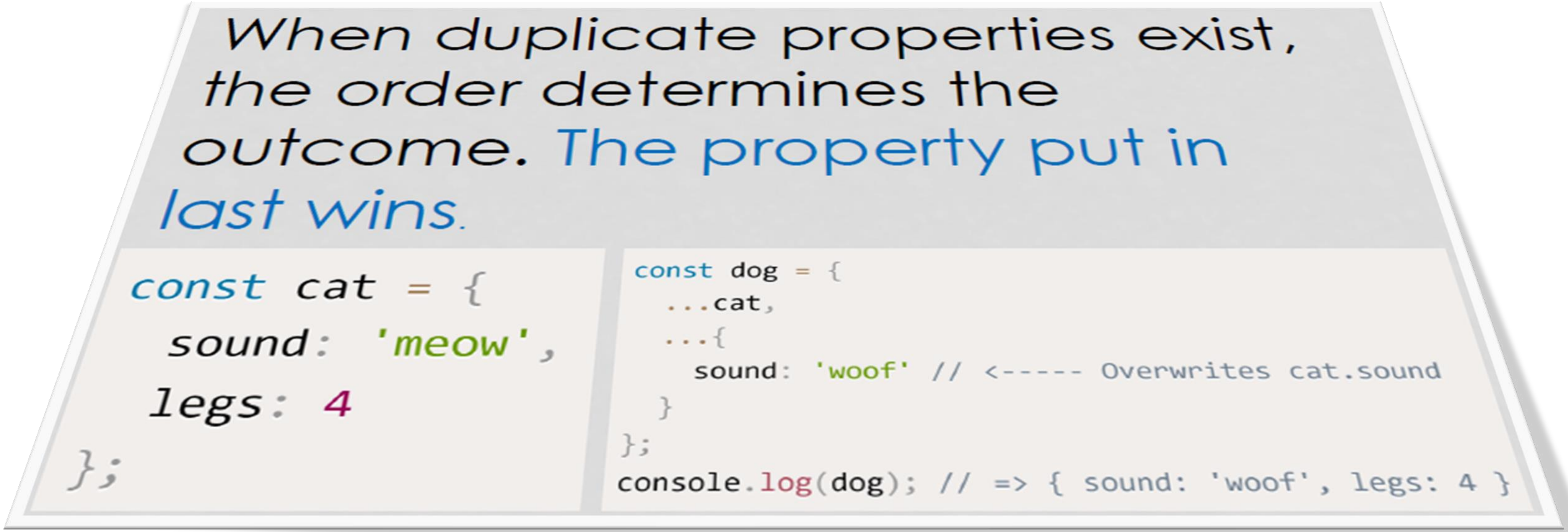
EXAMPLE: WITH OBJECT LITERALS



```
var obj1 = { foo: 'bar', x: 42 };
var obj2 = { foo: 'baz', y: 13 };

var clonedObj = { ...obj1 };
// Object { foo: "bar", x: 42 }

var mergedObj = { ...obj1, ...obj2 };
// Object { foo: "baz", x: 42, y: 13 }
```

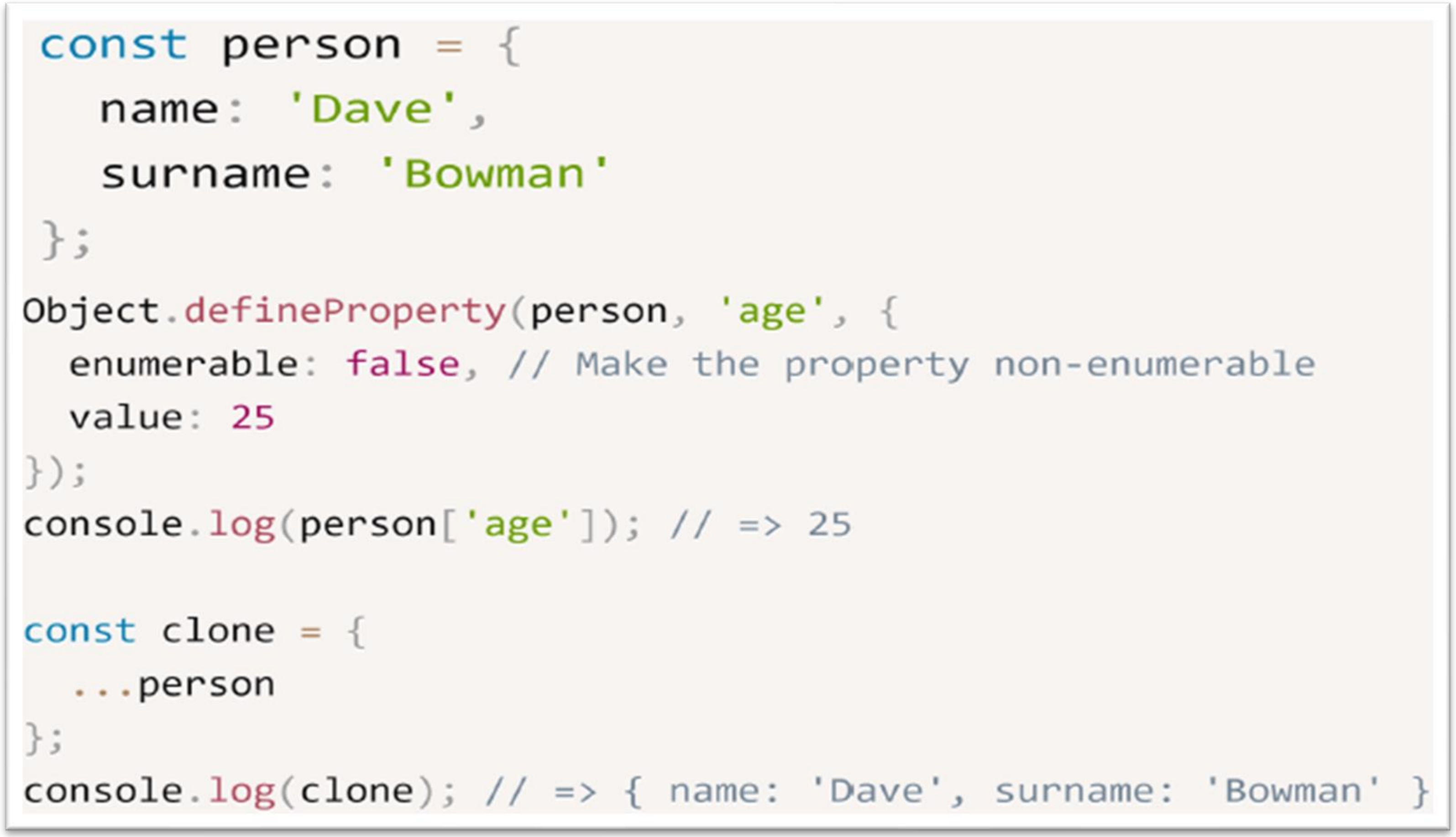


When duplicate properties exist, the order determines the outcome. The property put in last wins.

```
const cat = {
  sound: 'meow',
 legs: 4
};
```

```
const dog = {
  ...cat,
  ...{
    sound: 'woof' // <----- Overwrites cat.sound
  }
};
console.log(dog); // => { sound: 'woof', legs: 4 }
```

LAST IN: WINS


```
const person = {
   name: 'Dave',
   surname: 'Bowman'
};
Object.defineProperty(person, 'age', {
  enumerable: false, // Make the property non-enumerable
  value: 25
});
console.log(person['age']); // => 25

const clone = {
  ...person
};
console.log(clone); // => { name: 'Dave', surname: 'Bowman' }
```

EXAMPLE WITH NON-ENUMERABLES