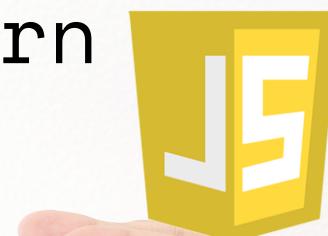


ARRAY CALLBACK METHODS

ARRAY

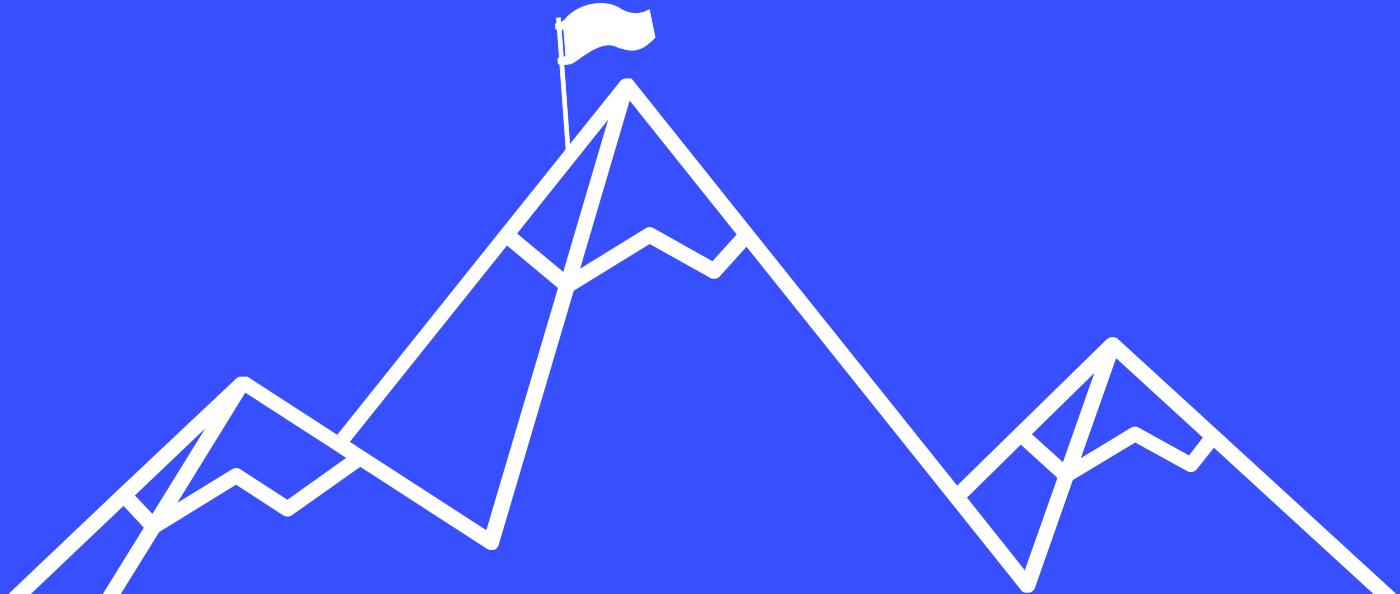
CALLBACK METHODS

- Arrays come with many built-in methods that accept callback functions
- Now that we've covered callbacks, we can learn these methods!
- SUPER USEFUL! 🎉



GOALS

- Use the new arrow function syntax
- Understand and use these methods:
 - forEach
 - map
 - filter
 - find
 - reduce
 - some
 - every



FOREACH



```
const nums = [9, 8, 7, 6, 5, 4, 3, 2, 1];

nums.forEach(function (n) {
  console.log(n * n)
  //prints: 81, 64, 49, 36, 25, 16, 9, 4, 1
});

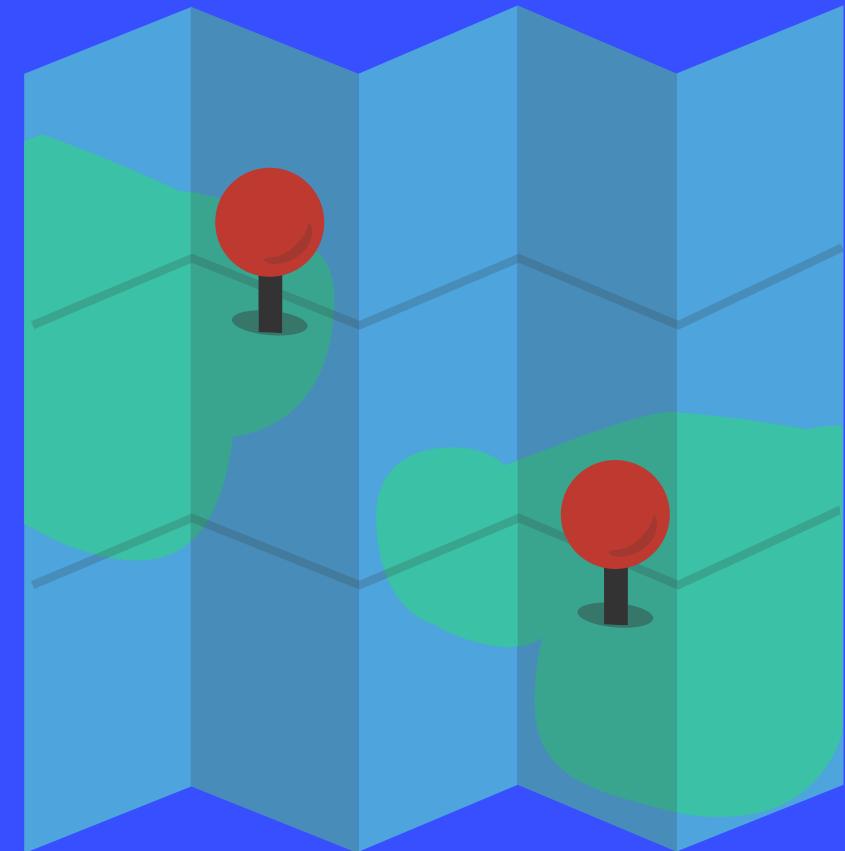
nums.forEach(function (el) {
  if (el % 2 === 0) {
    console.log(el)
    //prints: 8, 6, 4, 2
  }
})
```

Accepts a callback function.

Calls the function once per element in the array.

MAP

Creates a new array with the results of calling a callback on every element in the array



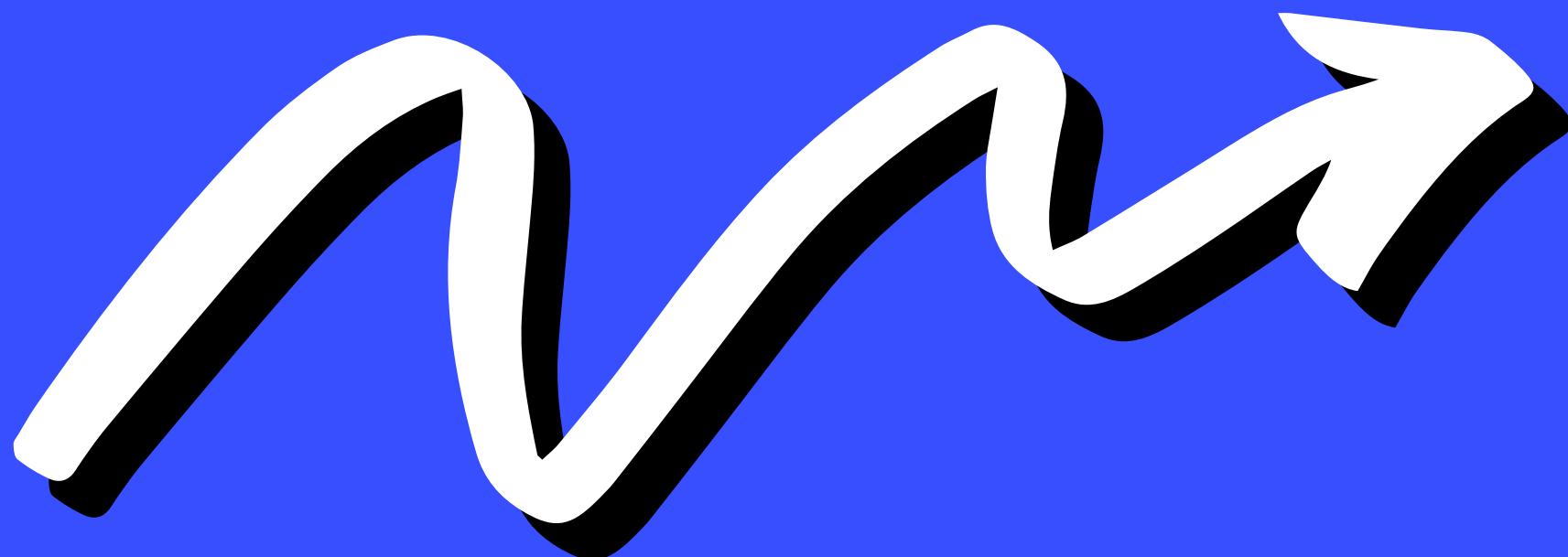
MAP



```
const texts = ['rofl', 'lol', 'omg', 'ttyl'];
const caps = texts.map(function (t) {
  return t.toUpperCase();
})
texts; //["rofl", "lol", "omg", "ttyl"]
caps; //["ROFL", "LOL", "OMG", "TTYL"]
```



ARROW FUNCTIONS!



ARROW FUNCTIONS



"syntactically compact alternative"
to a regular function expression



```
const square = (x) => {
  return x * x;
}
```

```
const sum = (x, y) => {
  return x + y;
}
```

ARROW FUNCTIONS



```
//parens are optional if there's only one parameter:  
const square = x => {  
    return x * x;  
}  
  
//Use empty parens for functions w/ no parameters:  
const singASong = () => {  
    return "LA LA LA LA LA LA";  
}
```

IMPLICIT RETURN

All these functions do the same thing:



```
const isEven = function (num) { //regular function expression
    return num % 2 === 0;
}
const isEven = (num) => { //arrow function with parens around param
    return num % 2 === 0;
}
const isEven = num => { //no parens around param
    return num % 2 === 0;
}
const isEven = num => ( //implicit return
    num % 2 === 0
);
const isEven = num => num % 2 === 0; //one-liner implicit return
```

FIND

returns the value of the **first element** in the array that satisfies the provided testing function.



```
let movies = [
  "The Fantastic Mr. Fox",
  "Mr. and Mrs. Smith",
  "Mrs. Doubtfire",
  "Mr. Deeds"
]
let movie = movies.find(movie => {
  return movie.includes('Mrs.')
}) // "Mr. and Mrs. Smith"

let movie2 = movies.find(m => m.indexOf('Mrs') === 0);
// "Mrs. Doubtfire"
```

FILTER

Creates a new array with all elements that pass the test implemented by the provided function.



```
const nums = [9, 8, 7, 6, 5, 4, 3, 2, 1];
const odds = nums.filter(n => {
    return n % 2 === 1; //our callback returns true or false
    //if it returns true, n is added to the filtered array
})
//[9, 7, 5, 3, 1]

const smallNums = nums.filter(n => n < 5);
//[4, 3, 2, 1]
```

EVERY

tests whether **all** elements in the array pass the provided function. It returns a Boolean value.



```
const words = ["dog", 'dig', 'log', 'bag', 'wag'];

words.every(word => {
  return word.length === 3;
}) //true

words.every(word => word[0] === 'd'); //false

words.every(w => {
  let last_letter = w[w.length - 1];
  return last_letter === 'g'
}) //true
```

SOME

Similar to `every`, but returns true if ANY of the array elements pass the test function



```
const words = [ 'dog', 'jello', 'log', 'cupcake', 'bag', 'wag' ];

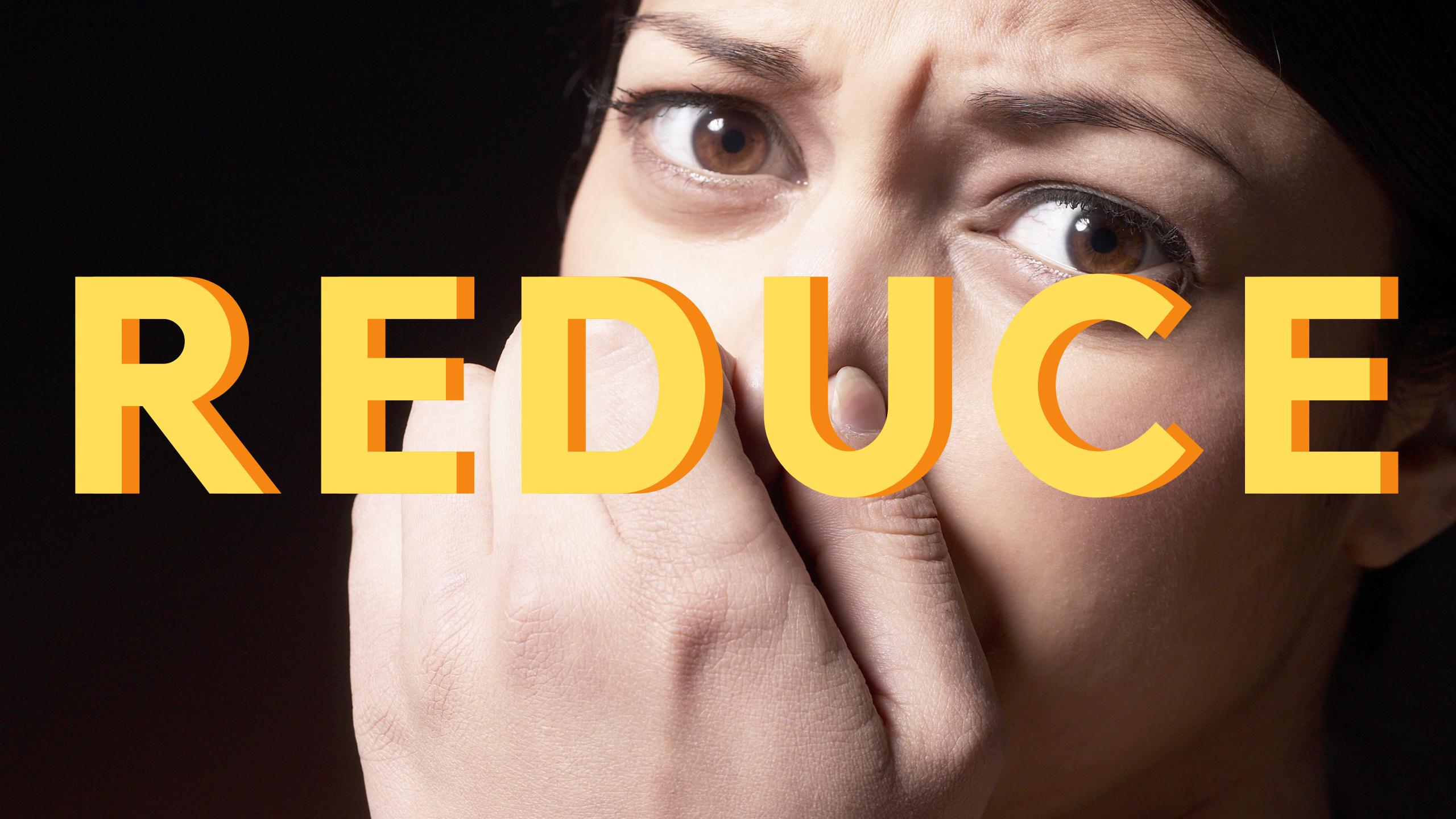
//Are there any words longer than 4 characters?
words.some(word => {
  return word.length > 4;
}) //true

//Do any words start with 'Z'?
words.some(word => word[0] === 'Z'); //false

//Do any words contain 'cake'?
words.some(w => w.includes('cake')) //true
```

REDUCE
REDUCE
REDUCE
REDUCE
REDUCE
REDUCE
REDUCE
REDUCE
REDUCE
REDUCE



A close-up photograph of a woman's face. Her eyes are looking directly at the camera with a neutral expression. Her hands are positioned in front of her face, with her fingers partially covering her mouth and nose, suggesting a gesture of silence or distress. The lighting is dramatic, highlighting the contours of her face and the texture of her skin.

REDUCE

REDUCE

Executes a reducer function
on each element of the array,
resulting in a single value.



SUMMING AN ARRAY



```
[3, 5, 7, 9, 11].reduce((accumulator, currentValue) => {  
  return accumulator + currentValue;  
});
```

Callback	accumulator	currentValue	return value
first call	3	5	8
second call	8	7	15
third call	15	9	24
fourth call	24	11	35

FINDING MAX VAL



```
let grades = [89, 96, 58, 77, 62, 93, 81, 99, 73];

const topScore = grades.reduce((max, currVal) => {
  if (currVal > max) return currVal;
  return max;
})
topScore; //99
```

```
//A shorter option w/ Math.max & implicit return
const topScore = grades.reduce((max, currVal) => (
  Math.max(max, currVal)
))
```

INITIAL VALUE



```
[4, 5, 6, 7, 8].reduce((accumulator, currentValue) => {  
    return accumulator + currentValue;  
});  
//RETURNS: 30
```

```
[4, 5, 6, 7, 8].reduce((accumulator, currentValue) => {  
    return accumulator + currentValue;  
}, 100);  
//RETURNS: 130
```

TALLYING



```
const votes =  
['y','y','n','y','n','y','n','y','n','n','y','y'];  
const tally = votes.reduce((tally, vote) => {  
  tally[vote] = (tally[vote] || 0) + 1;  
  return tally;  
}, {}); //INITIAL VALUE: {}  
  
tally; //{y: 7, n: 6}
```

SORT PT. 2

arr.sort(compareFunc(a,b))

- If *compareFunc(a,b)* returns less than 0
 - Sort *a* before *b*
- If *compareFunc(a,b)* returns 0
 - Leave *a* and *b* unchanged with respect to each other
- If *compareFunc(a,b)* returns greater than 0
 - Sort *b* before *a*

SORT



```
const prices = [400.50, 3000, 99.99, 35.99, 12.00, 9500];
```

```
prices.sort();
//[12, 3000, 35.99, 400.5, 9500, 99.99]
```

```
prices.sort((a, b) => a - b)
//[12, 35.99, 99.99, 400.5, 3000, 9500]
```

```
prices.sort((a, b) => b - a);
//[9500, 3000, 400.5, 99.99, 35.99, 12]
```