More Array Knowledge

Array.sort()

- Doesn't work the way you think it does

```
var sports = ["soccer", "hockey",
"football"];
var numbers = [2, 100, 15, 67, 33];
```

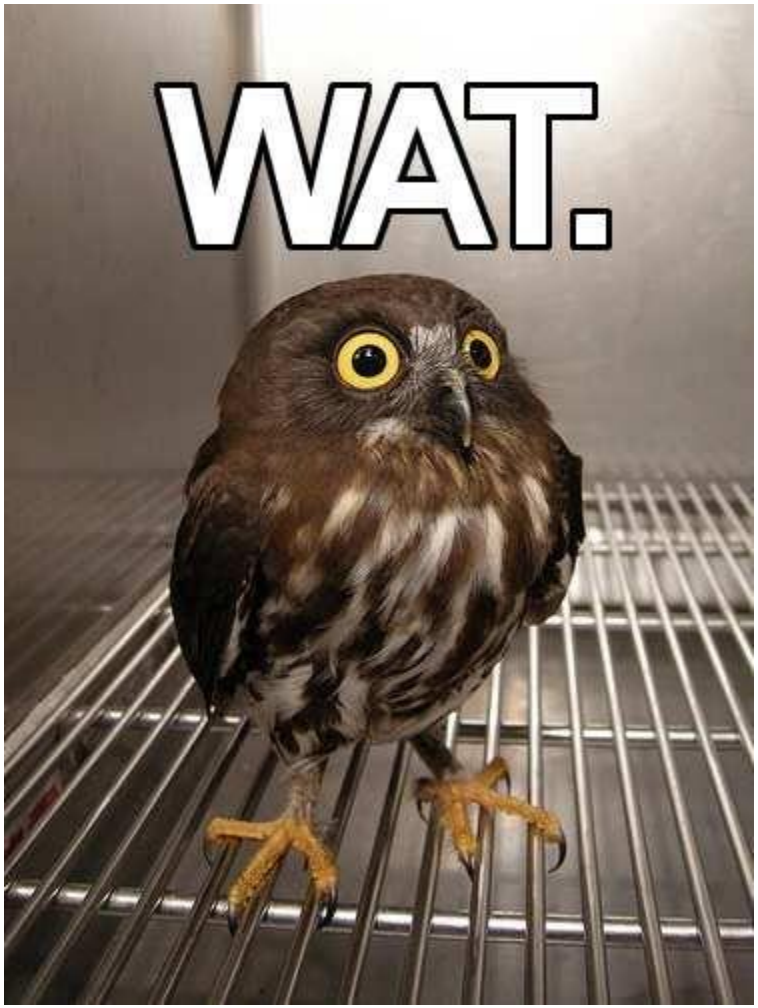
- What do you think the result of sports.sort() is?
- `["football”,"hockey","soccer"]`

Array.sort()

- Doesn't work the way you think it does

```
var sports = ["soccer", "hockey",
"football"];
var numbers = [2, 100, 15, 67, 33];
```

- What do you think the result of numbers.sort() is?
- `[100, 15, 2, 33, 67]`


WAT.

Array.sort()

- So the sort method pretends your numbers are strings and sorts them “alphabetically”
- It doesn’t convert them to strings, it leaves them as numbers

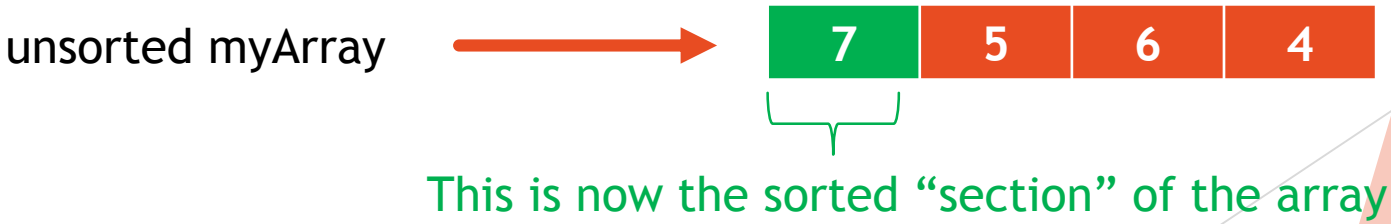
- So how do we sort things array elements numerically?
- We have to specify that we want to sort numerically (ascending or descending) using functions!

Insertion Sort

- There are many different sorting algorithms
- It appears that when sorting numbers JavaScript in Chrome uses insertion sort to sort arrays
- insertion sort works by separating an array into two sections, a sorted section and an unsorted section
- Initially, of course, the entire array is unsorted, so the sorted section of the array is considered to be empty

Insertion Sort

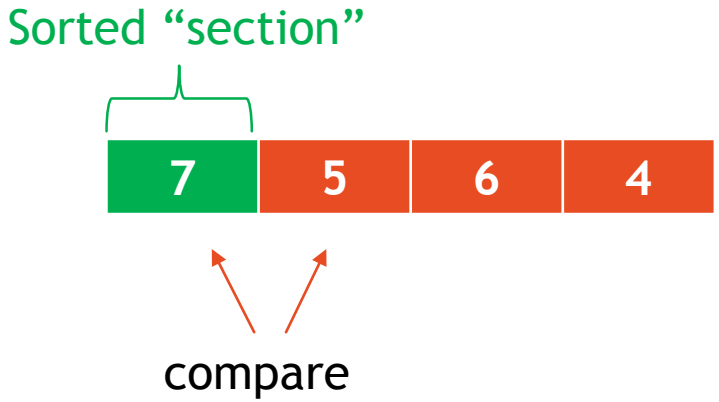
- The first step is to add a value to the sorted section, so the first item in the array is used
- This works because a list of one item is always sorted

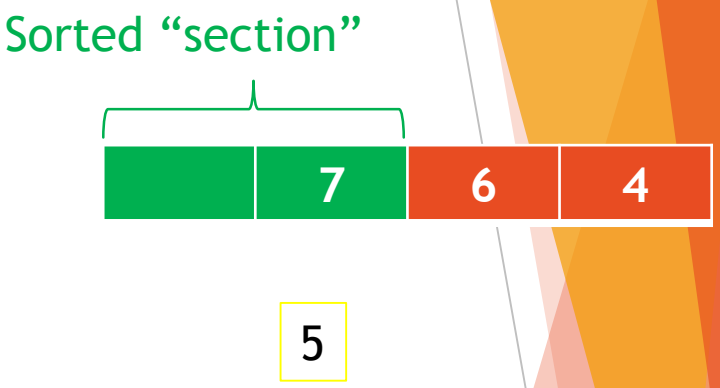


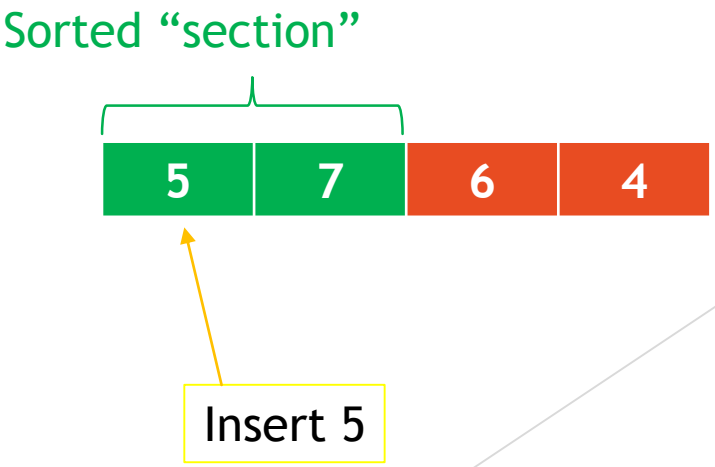
Insertion Sort

- Then at each item in the unsorted section:
  - If the item value goes after the last item in the sorted section, then do nothing
  - If the item value goes before the last item in the sorted section, remove the item value from the array and shift the last sorted item into the now-vacant spot
  - Compare the item value to the previous value (second to last) in the sorted section
  - If the item value goes after the previous value and before the last value, then place the item into the open spot between them, otherwise, continue this process until the start of the array is reached

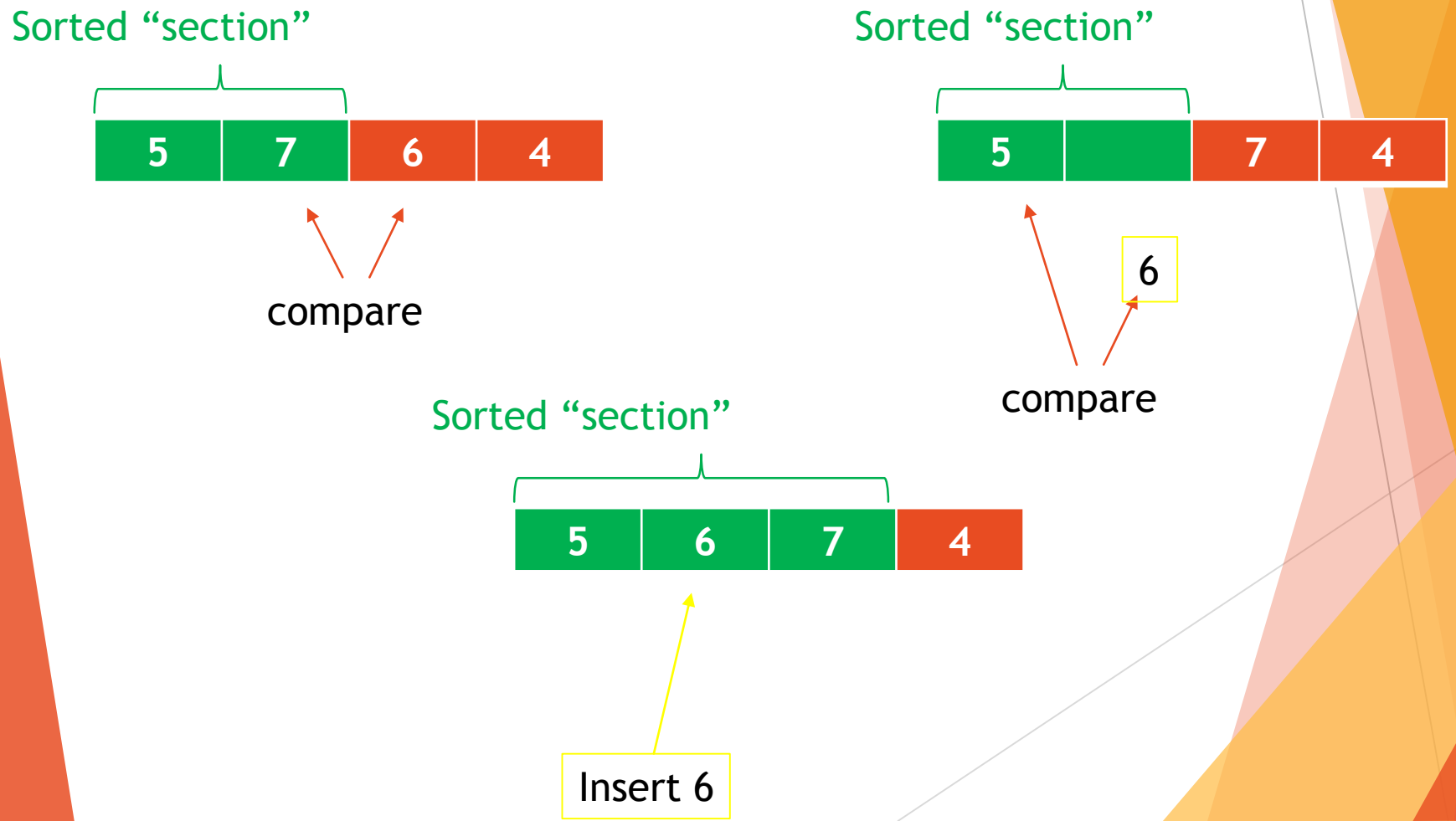
Example


Sorted “section”
7
5
6
4
compare

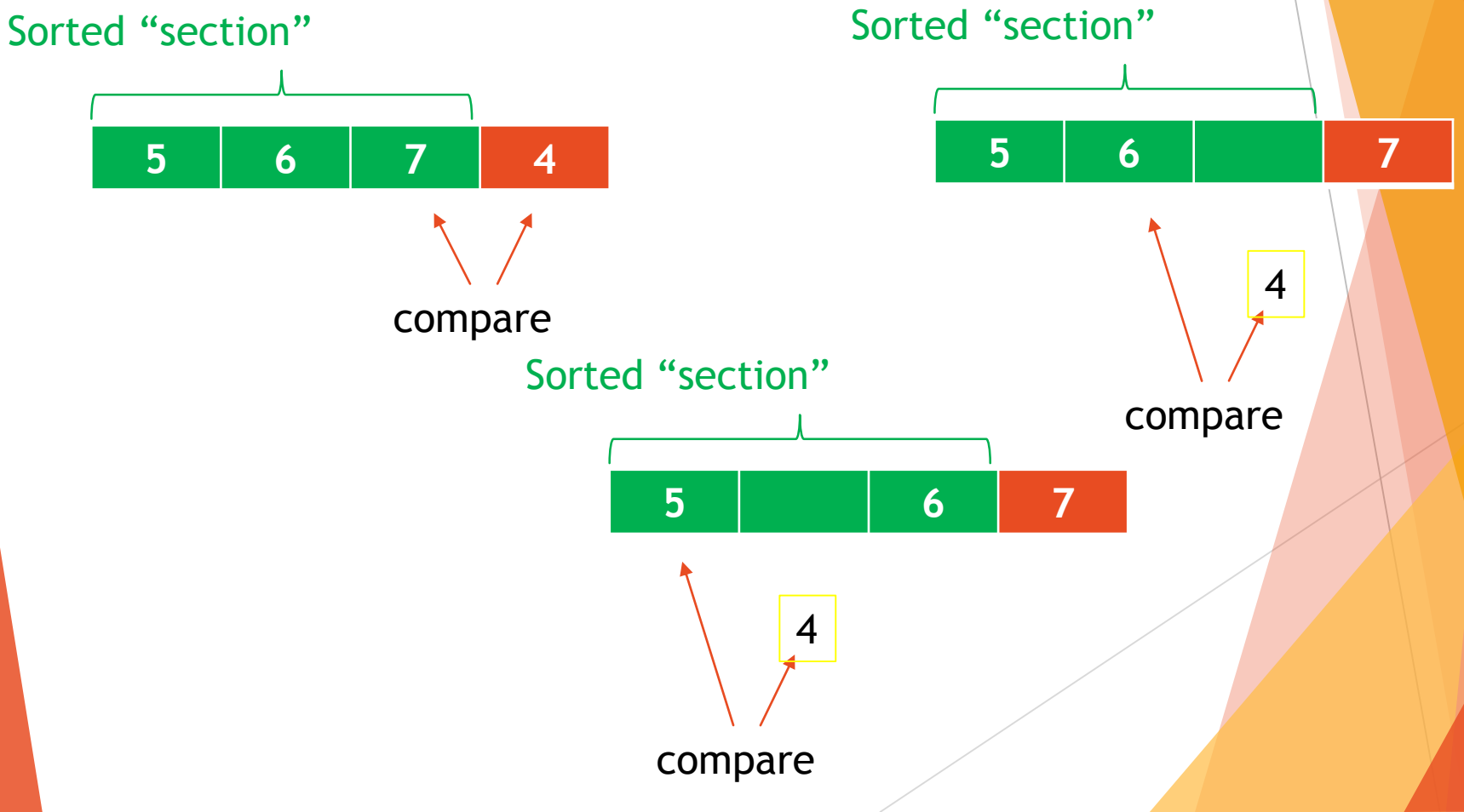

Sorted “section”
7
6
4
5


Sorted “section”
5
7
6
4
Insert 5

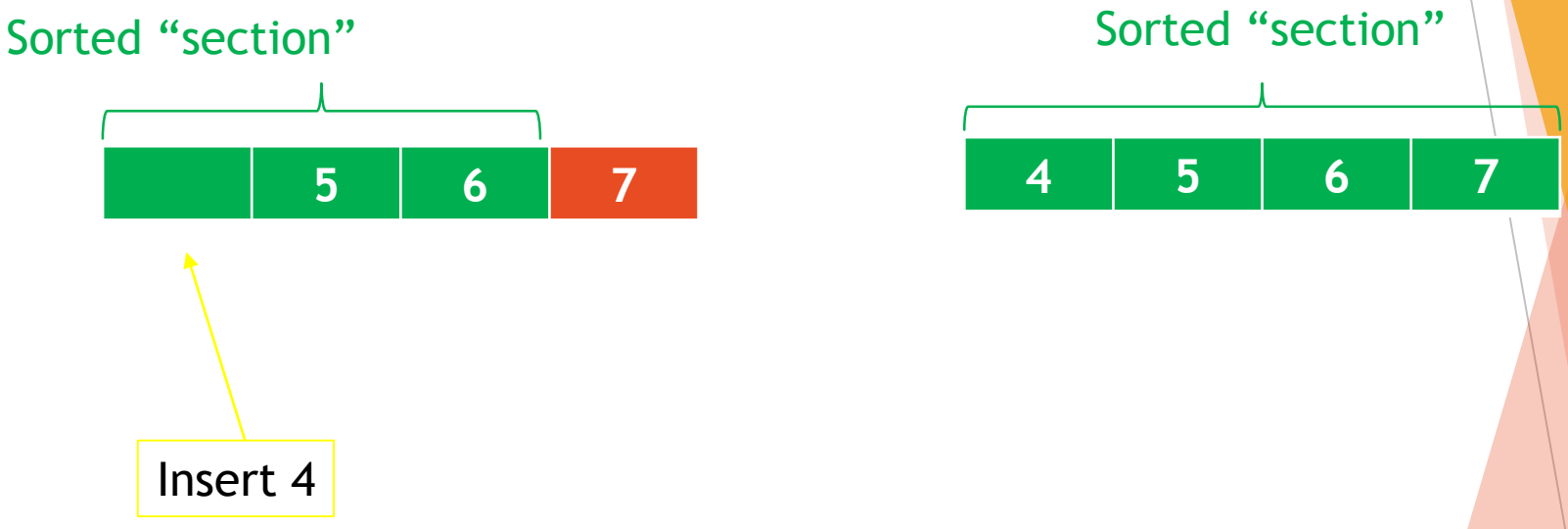
Example


Sorted “section”
5
7
6
4
compare
Sorted “section”
5
7
4
6
compare
Sorted “section”
5
6
7
4
Insert 6

Example


Sorted “section”
5
6
7
4
compare
Sorted “section”
5
6
7
4
compare
Sorted “section”
5
6
7
4
compare

Example



Review

- So we know JavaScript uses insertion sort to sort arrays of numbers
- We know JavaScript sorts our numbers “alphabetically”
- We know that to sort numerically (ascending or descending) we have to specify which function to use to compare the numbers
  - The compareNumbers(a,b) function
- If compareFunction is supplied, the array elements are sorted according to the return value of the compare function

Function to Compare Numbers

- If a and b are two elements being compared, then:
  - If compareNumbers(a, b) is less than 0, sort a to a lower index than b, i.e. a comes first.
  - If compareNumbers(a, b) returns 0, leave a and b unchanged with respect to each other, but sorted with respect to all different elements.
  - If compareNuumbers(a, b) is greater than 0, sort b to a lower index than a.

Ascending

```
function compareNumbers(a, b)
{
    return a - b;
}
```

Descending

```
function compareNumbers(a, b)
{
    return b - a;
}
```

So How Do We Use It?

```
var numbers = [2, 100, 15, 67, 33];
numbers.sort(compareNumbers);
```

Ascending

```
function compareNumbers(a, b)
{
    return a - b;
}
```

Descending

```
function compareNumbers(a, b)
{
    return b - a;
}
```

Ex.

```
> function compareNumbers(a, b) {
  console.log (a + " " + b + " " + (a - b))
  console.log(numbers)
    return a - b ;
  }
<- undefined
> numbers = [2,100,15,67,33]
<- [2, 100, 15, 67, 33]
> numbers.sort(compareNumbers)
  2 100 -98
  [2, 100, 15, 67, 33]
  100 15 85
  [2, 100, 15, 67, 33]
  2 15 -13
  [2, 100, 100, 67, 33]
  100 67 33
  [2, 15, 100, 67, 33]
  15 67 -52
  [2, 15, 100, 100, 33]
  100 33 67
  [2, 15, 67, 100, 33]
  67 33 34
  [2, 15, 67, 100, 100]
  15 33 -18
  [2, 15, 67, 67, 100]
<- [2, 15, 33, 67, 100]
```

```
> function compareNumbers(a, b) {
  console.log (a + " " + b + " " + (b - a))
  console.log(numbers)
    return b - a;
  }
<- undefined
> numbers = [2,100,15,67,33]
<- [2, 100, 15, 67, 33]
> numbers.sort(compareNumbers)
  2 100 98
  [2, 100, 15, 67, 33]
  2 15 13
  [100, 2, 15, 67, 33]
  100 15 -85
  [100, 2, 2, 67, 33]
  2 67 65
  [100, 15, 2, 67, 33]
  15 67 52
  [100, 15, 2, 2, 33]
  100 67 -33
  [100, 15, 15, 2, 33]
  2 33 31
  [100, 67, 15, 2, 33]
  15 33 18
  [100, 67, 15, 2, 2]
  67 33 -34
  [100, 67, 15, 15, 2]
<- [100, 67, 33, 15, 2]
```

Let's Try It

- Write a JavaScript function called find(myArray). This function should take in an array of numbers as a parameter and find the second lowest and second greatest numbers, respectively, and print them to the console
  - Example:
    find([6, 8, 22, 1, 44, 13]) should print out
    "Next Largest: 22 Next Smallest: 6"

Exercise

- Write out the steps of insertion sort for the following unsorted array in ascending order
  - var numbers = [6, 26, 2, 94, 88, 40, 54, 67];

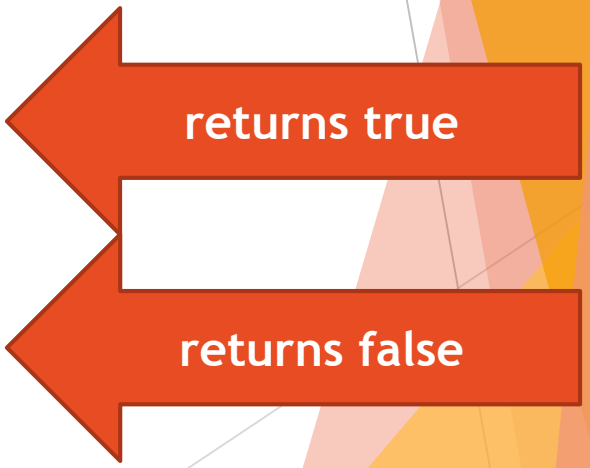
Every

- If you've ever wanted to check and see if every item in array meets some criteria, you're in luck
- The every() method checks if all elements in an array pass a test (provided as a function)
- The every() method executes the function once for each element present in the array:
  - If it finds an array element where the function returns a *false* value, every() returns false (and does not check the remaining values)
  - If no false occur, every() returns true

Example with every()

```
var numberArray = [40, 1, 5, 200];
var otherArray = [“this“, “is”, “a”, “string”, “array”];

numberArray.every(function(a){
   return typeof(a) === "number“
});
otherArray.every(function(a){
   return typeof(a) === "number“
});
```



Example with every()

- This notation looks ugly, but is (kind of?) understandable

```
numberArray.every(function(a){
    return typeof(a) === "number"
});
```

- We could rewrite this as a separate function and then refer to it:

```
function checkTypeOf(a){
    return typeof(a) === "number"
}
numberArray.every(checkTypeOf);
```

Every's function

- You must pass a function into every()
- The function passed into every() has the following parameters
- *function(currentValue, index, arr)*
  - currentValue is required, it represents the value of the current element
  - index is optional, it represents array index of the current element
  - arr is optional, it is the array object the current element belongs to

Example

```
function isBigEnough(element, index, array)
{
  return element >= 10;
}


[12, 5, 8, 130, 44].every(isBigEnough);
// false
[12, 54, 18, 130, 44].every(isBigEnough);
// true
```