





Agenda

Activity	Time
Get Prepared: Log in to Nucamp Learning Portal • Slack • Screenshare	15 minutes
Introductions & Check-In	15 minutes
React Course Overview & Week 1 Review	90 minutes
BREAK	15 minutes
Finish Week 1 Review	20 minutes
Workshop Assignment	70 minutes
Check-Out	15 minutes



Introductions & Check-In

- Instructor introduction
- Student introductions – students should know each other already, but please introduce yourself briefly to your instructor if you have not met.
- Check-In:
 - How was this week? Any particular challenges or accomplishments?
 - Did you understand the Exercises and were you able to complete them?
 - You must complete all Exercises before beginning the Workshop Assignment. We will review some of the Exercises together next, along with this week's concepts.

React Course Overview

- This will be the hardest course in the bootcamp, be prepared!
- Expect to work on: Same NuCamp campsites project - *React*-ified
- Your same Portfolio Project – *React*-ified
- Give yourself at *least* 2 hours a day, M-F, to watch lectures, do exercises and challenges, do additional study & practice concepts – plus more time for your portfolio project.
- 20-minute rule - Reach out during the week if you'd like help understanding something, don't wait until the workshop



React Course Overview: Tips

- **Accept the notion of a “Black Box”** - you will find yourself using code without knowing exactly how it works at first
- Do your best, don't stress if it doesn't make sense. You will repeat the same tasks over and over again and come to understand them better... over time!
- Take it upon yourself to research more advanced JavaScript online, bit by bit, one concept at a time – build upon existing knowledge
- Difficult learning curve at first, very rewarding afterward!
- Even if you don't end up using React later, the concepts you will learn will still be useful to you overall as a web developer

Welcome back ... to more JavaScript!

Week 1 Recap – Overview: New Concepts You Learned This Week

- Data Type: Object
- Classes
- Class Inheritance
- *this* keyword

- Map
- Filter
- Reduce
- Arrays of Objects

Next slides will review these concepts.

There's Only Seven Data Types In JavaScript!

You already know:

- String (ex. 'Hello')
- Number (ex. 99.9)
- Boolean (True/False)

There's also:

- Undefined
- Null
- Symbol (new in ES6)
- BigInt (very new)

This week, you also learned about:

 **OBJECTS** 

*Objects are one of the original JavaScript data types.
They are not new in ES6. They've been here all along!*

Wait, what about arrays?

Good question... Arrays are not a data type.

Arrays are actually a special type of object.

So ... what *is* an **Object** again?

An **OBJECT** is:

- A collection of custom key/value pairs
- that store **properties** (such as name, height, color)
- and **methods** (actions – add(), delete(), log(), etc.).

Object Methods

Methods are functions attached to objects.

When called, methods run code and can have a return value.

You always call them like this: `<object>.<method>()`

If you pass in arguments, they go inside the parentheses.

You have already used a lot of methods, whether you realized it or not.

For example, whenever you wrote something to your console using `console.log('your text here')`, you were using the 'log' method of the global console object, passing in the argument 'your text here'.

Let's practice writing a custom object together,
with properties and a method.

Making Your Own Objects

Open your browser's Developer Console now in this tab. In your console, enter:



```
const bicycle = {  
  color: 'blue',  
  electric: false,  
  start() {  
    console.log('You begin to pedal the bike.');  }  
};
```

Next, in your console, type each line to the right. Hit enter after each line and look closely at each result before moving on to the next. Discuss your observations.

- `bicycle;`
- `bicycle.color;`
- `bicycle.start;`
- `bicycle.start();`



Manipulating Objects

Let's turn our bicycle into a green e-bike!



- Discuss together – what can you type into your console to change your bicycle object into a green electric bike?
- Next, change the `bicycle.start` method so that when called, it will log to the console: 'You start the engine on your bicycle.'
- Check the results in your console.



Manipulating Objects (cont)

Answers:

```
bicycle.color = 'green';  
bicycle.electric = true;  
bicycle.start = function() {  
  console.log('You start the engine on your  
bicycle.');
```



Arrow Function & Objects Practice

- Discuss as a class: How would you turn this function into the smallest possible (most concise) arrow function?

```
function hello(name) {  
    return { studentName: name };  
}
```



Arrow Function & Objects Practice: Answer

```
const hello = name => ({ studentName: name });
```

Discuss:

- Why do you need parentheses around the object literal?
- Why do you not need a return keyword or curly braces surrounding the return value?
- Why do you not need parentheses around the first occurrence of 'name'?



Review: Classes

- Classes are used in other programming languages extensively
- Class syntax was introduced to JavaScript in ES6
- You use classes to *instantiate* objects
 - a.k.a. making a new object, a.k.a. an *instance* of the class
 - A class can have a special method inside it called a **constructor**.
 - There can be only one constructor per class.
 - It creates and initializes any object made from that class.



Create a Class

Practice: In your Console, create this class:

```
class Book {  
  constructor(title, author, year, isRead=false) {  
    this.title = title;  
    this.author = author;  
    this.year = year;  
    this.isRead = isRead;  
  }  
}
```

Note: the *isRead* parameter has a default value of *false*, via default function parameters. Let's practice creating objects from a class. Enter the following and discuss each result.

```
const book1 = new Book('Steppenwolf', 'Herman Hesse', 1927, true);  
const book2 = new Book('Dune', 'Frank Herbert', 1965);
```

Now create a book3 object with a book of your choice, but only pass in two parameters: title and author. Discuss the result. What happened and why?



this and Class Constructors

The JavaScript keyword *this* inside an object refers to the object itself.

With classes, in the constructor, you will use the *this* keyword to initialize arguments passed in via the parameter list as properties of objects created from that class.

In the Book example, when you made your "book2" object using the Book class, you passed in the title "Dune" to the class constructor, right?

But just passing a value into the title argument isn't enough. In order to initialize it as the title property of the "book2" object being created, you wrote: *this.title = title;*

```
class Book {  
  constructor(title, author, year, isRead=false) {  
    this.title = title;  
    this.author = author;  
    this.year = year;  
    this.isRead = isRead;  
  }  
}  
  
const book2 = new Book('Dune', 'Frank Herbert');
```

That line is what makes it possible for you to now type: *book2.title* and get "Dune". Otherwise, *book2.title* would return *undefined*.



Discuss: What are some other uses of the *this* keyword you learned this week?

(if you can't think of any, the next slide contains a hint)



Hint: For example, you learned about using the *this* keyword in object methods in general, and also specifically in method chaining. How did you use it there?



Review: Class Inheritance

- You can inherit the properties and methods of an existing class into a new class using the *extends* keyword and the *super* method, then add more properties and methods to the new class only.
- In your Console, enter the below and observe the results:

```
class Audiobook extends Book {  
  constructor(title, author, year) {  
    super(title, author, year);  
  }  
  playAudio() {  
    console.log(`The audio recording of ${this.title} begins to play.`);  
  }  
}  
const book4 = new Audiobook('The Martian Chronicles', 'Ray Bradbury', 1950);  
book4.playAudio();
```



Review: Advanced Array Methods

Filter:

- Takes a callback function (very powerful – you can filter on anything)
- Returns a new, filtered array (same or less size than original array)
- Example:

```
const students = ['James', 'Jose', 'Katya', 'Kelsey', 'Sergey'];  
const newArray = students.filter(name => name[0] === 'K');
```

Wait... What's a callback function again?

Callback functions are functions that are passed as arguments into another function/method, then (generally) invoked (a.k.a. called) inside there.

In the example above, the callback function is the arrow function `name => name[0] === 'K'` - it is passed into the filter method as the argument.

Arrow functions are often used for callback functions.



Review: Advanced Array Methods (cont)

- **Map method:**

- Takes a callback function
- Returns a new array based on the result of the callback function (same size as original array)
- Try this in your console:
Then check the value of *squared*.

```
const myArr = [1, 2, 3];  
const squared = myArr.map(x => x * x);
```

- **Reduce method:**

- Takes a callback function (the “reducer”) and an initial value for the accumulator: `Array.reduce(reducer function, initial value)`
- Returns a single output value
- The reducer function requires two arguments: an accumulator and a current value
- Try these examples, using the same array as the example in Map above:

```
const sum = myArr.reduce((sum, curVal) => sum + curVal, 0);  
const mult = myArr.reduce((mult, curVal) => mult * curVal, 1);
```



Review: Advanced Array Methods (cont)

- The Filter, Map, and Reduce methods are all used on arrays and do not change (mutate) the original array.
- Filter and Map will always return a new array.
- Reduce will always return a single value, not an array.
- When used with arrays of objects, reduce typically requires an additional initialValue parameter to be set. The initialValue parameter can be used with arrays of non-objects also.



Extra: Chaining Array methods

```
const data = [  
  {  
    name: 'Butters',  
    age: 3,  
    type: 'dog'  
  },  
  {  
    name: 'Lizzy',  
    age: 6,  
    type: 'dog'  
  },  
  {  
    name: 'Red',  
    age: 1,  
    type: 'cat'  
  },  
  {  
    name: 'Joey',  
    age: 3,  
    type: 'dog'  
  }  
];
```

It's possible to chain array method calls to the result of a previous array method call (as long as that method returns a value), and you will see this often in practice. Let's take some data about animals and return the sum of all the dogs' ages in dog years. In your console:

1. Enter the data to the left.
2. Enter the code below.
3. Observe how the code:
 - Selects only the objects of type dog, using filter
 - Translates their ages into dog years (multiplies by 7), using map to create a new array
 - Combines the results into a single number, using reduce. The sum (stored in the 'ages' variable) should be 84.)

```
const ages = data  
  .filter(animal => animal.type === 'dog')  
  .map(dog => dog.age * 7)  
  .reduce((sum, dogyears) => sum + dogyears, 0);
```



Discussion

Time permitting, take some time to discuss any questions about this week's material, including going over the Code Challenges and Quiz.

Be sure to leave yourselves time to complete the Workshop Assignment and Check-Out – suggested ~85 minutes.



Workshop Assignment

- It's time to start the workshop assignment!
- Sit near your workshop partner.
 - Your instructor may assign partners, or have you choose.
- Work closely with each other.
 - 10-minute rule does *not* apply to talking to your partner, you should consult each other throughout.
- Follow the workshop instructions very closely
 - both the video and written instructions.
- Talk to your instructor if any of the instructions are unclear to you.



Check-Out

- Submit your final workshop files (both the HTML and JS files) via the Nucamp Learning Portal.
- Wrap up – Retrospective
 - What went well
 - What could improve
 - Action items
- Start on Week 2 if there's time, or continue discussing any Code Challenges/Quiz questions/any questions that remain from this week's materials.