ULTIMATE JAVASCRIPT OBJECTS

With Daniel Stern, Code Whisperer

INTRODUCTION

Ultimate JavaScript Objects

WHY JAVASCRIPT OBJECTS?

- JavaScript objects are incredibly flexible and easy to use
- Amazing for representing data reflecting real-life scenarios
  - Users in database
  - Diagnostic information from IoT
  - Server communications
- Format compliments JavaScript’s weakly-typed and dynamic style
- Useful for front-end applications (Angular, React)
- Integrates directly with Node.js-based back-ends
- Query certain database types (MongoDB, CouchDB) directly with JavaScript objects

COURSE ROADMAP

- Learn about objects – what, why and how
- Create and interact with objects in JavaScript via examples and demos
- Understand object scope and apply it in real JavaScript scenarios
- Create objects via innovative ES6 classes
- Work with JSON (JavaScript Object Notation) – a popular API and DB convention
- Use Lodash to streamline many common objects chores

BEFORE WE BEGIN

- All code demos will take place in **Google Chrome**
- Code along at home with all code demos
- **Do all the quizzes and exercises**
- PDFs for each chapter are provided

OBJECTS EXPOSED

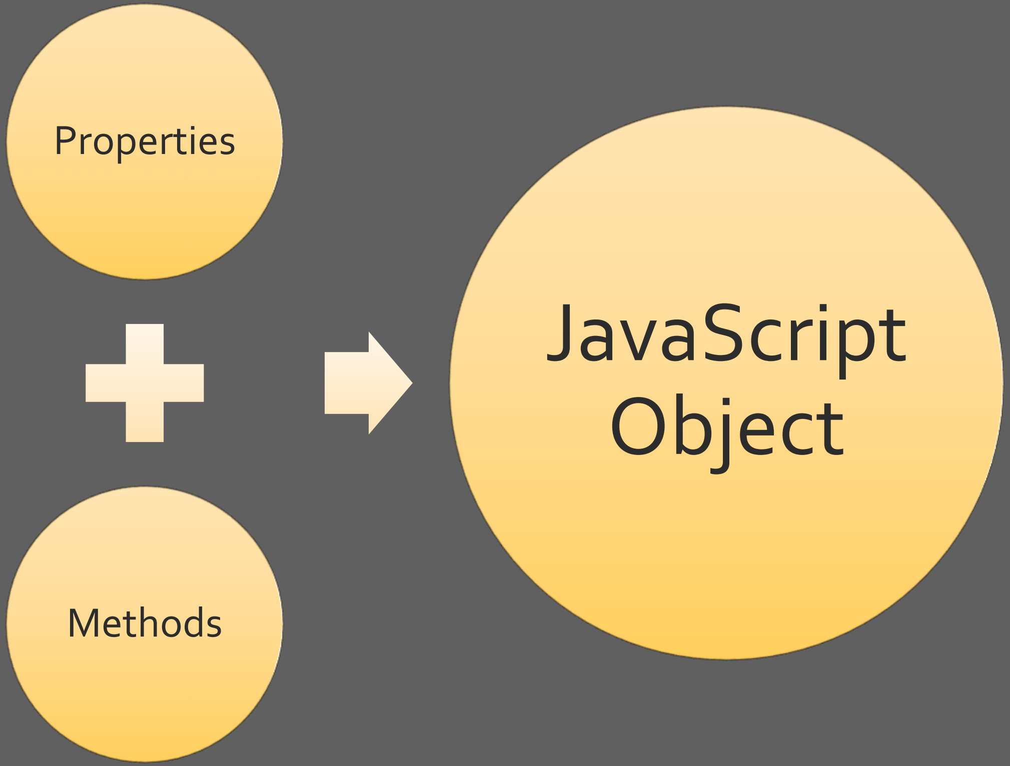
Ultimate JavaScript Objects

INTRODUCTION

- Cover all relevant aspects of Object theory
- Explain OOP
- Compare objects and arrays
- Discuss the many weird and interesting properties of JavaScript objects specifically
- Take a look at the colorful history of JavaScript objects

WHAT IS AN OBJECT?

- An object is a collection of values (properties)
- All properties have a unique key
- Order of properties cannot be trusted
- Some properties can be functions (methods)
- Can sometimes be serialized – or turned into a universal form of data like a string



WHAT IS OBJECT-ORIENTED PROGRAMMING?

- Termed by Alan Kays in the 60's, referring to biological cells
- Objects can be instances of classes
- Classes are like blueprints, objects are like the manufactured goods
- Each instance of a class is independent, has own properties
- JavaScript allows many implementations of OOP without being mandatory
- Useful for solving difficult problems or ones that are hard to conceptualize

WHAT CAN OBJECTS DO?

- Have any number of properties whose values are unique to that object
- Transfer data between functions or even programs
- Group complex data in ways easy for humans to understand
- Possess methods that work based on that object’s internal properties
- Have some properties accessible by other objects, some hidden
- Implement nearly all of the collection features found in arrays
- Represent classes that work together in grand and complex ways

WHEN TO USE OBJECTS?

- Collection of properties that all fit together thematically
- Each element has unique key
- Representing identical things inside a list
- Want methods that can operate on self in object-oriented fashion
- Elements in no particular order

OBJECTS VS ARRAYS

Objects

- Can have any number of properties accessible by indexes
- Copied by reference
- Primitive type
- No built in push, filter, etc.
- Order of properties doesn’t matter

Arrays

- Can have any number of properties accessible by indexes
- Copied by reference
- Inherits from Object
- Numerous built-in methods
- Order of elements always matters

THE SECRET LIFE OF JAVASCRIPT OBJECTS

- Can borrow methods from other functions using the mysterious *this* keyword
- Inside of objects, special scope applies
- Objects can be initialized by classes, but still have dynamic properties added later
- Are themselves a data-transfer format
- Both arrays and functions inherit from objects

EVOLUTION OF JAVASCRIPT OBJECTS EXPLAINED

- No classes before ES6
- OOP could still be implemented through libraries, manually
- Recently added Freeze, Observe and other interesting object methods
- Many methods based on implementations in Lodash, Prototype.js

CONCLUSION

- JavaScript objects are dynamic and powerful
- Can be used for most database or front-end purposes
- Objects are similar to arrays
- Objects work well when implemented using object-oriented programming
- Copied by reference

WORKING WITH OBJECTS

Ultimate JavaScript Objects

INTRODUCTION

- JavaScript objects are easy to learn, yet tough to master
- Numerous ways to create, edit and access – some with differing consequences
- New ES6 features add to required body of knowledge
- Some methods (creation, mapping) are used every day
- Some features (like Freezing and Copying) are more obscure but still useful

MODULE ROADMAP – WORKING WITH OBJECTS

- Create objects using object literal notation (and alternatives)
- Add and access object properties
- Iterate over properties
- Map arrays of objects
- Explore ES6 features – symbol, observe and freeze
- Learn about object.prototype

Create
Access
Control

CREATING OBJECTS

- Objects can be created using object literal syntax { }
- Objects can also be created with the new keyword
- Can also be created with Object.create
- No perfect solution for every situation
- Using *new* allows for classlike behavior and prototypical inheritance

ADDING AND ACCESSING OBJECT PROPERTIES

- Objects can be initialized by putting key value pairs inside curly brackets
- Multiple key-value pairs separated by commas
- Properties can be added to existing objects with dot or bracket notation
- Access also occurs via dot or bracket notation
- Some properties only accessible via bracket notation

```
{
  “name”:“Jon Snow”,
  “greatWolf”:“Ghost”
}
```

ADDING METHODS TO OBJECTS

- Objects can have properties which are functions
- These are called methods
- No difference between method and function except *this* keyword
- No practical reason to implement non-static methods without *this*
- The *this* keyword will be fully explained in the next chapter on Object Scope

REMOVING OBJECT PROPERTIES

- Property values can be removed without removing the key by setting them to *undefined*
- Both key and value can be removed by using the delete keyword

ITERATING THROUGH OBJECT PROPERTIES

- Object properties can be iterated through via For In Loop
- Used to be very challenging to do this in ES5, now is easy
- Looping doesn’t usually make sense since objects tend to have different types of values as variables
- Sometimes can be useful with array-like objects

MAPPING ARRAYS OF OBJECTS

- An extremely common problem is needing to transform (or map) an array of congruent objects into a different type of object
- E.g., database transfer script, React store
- Can easily be mapped with built-in array.map()
- Editing existing objects / copying both possible

SYMBOLLY AMAZING - ACCESSING OBJECT PROPERTIES WITH SYMBOLS

- Keeping references to object properties via strings is a time-honored and beloved hack
  - Using non-unique indexes can lead to catastrophically hard-to-debug errors
  - Two separate components of app may unwittingly use same property name, i.e., “name” or “health”
- Symbols are guaranteed to be unique
- Conceptually identical to a GUID (long, random string)

“A **symbol** is a unique and immutable data type and may be used as an identifier for object properties. ” - MDN

FREEZE & SEAL

- New to ES6
- Both restrict (control) ways an object can be modified
- Mostly used to prevent developer error
- Prevents new properties from being added (seal) or anything at all from changing (freeze)

OBSERVING CHANGES WITH PROXIES

- Developers often require code to run whenever an object changes or is accessed
- When working with asynchronous code (i.e., other user collaborating remotely) this can happen often but unpredictably
- Object.observe() was originally created for this purpose but was deprecated
- Proxies allow greatly increased oversight of objects
- Control how values are changed

PROTOTYPE

- Prototype is a powerful but highly confusing feature of objects
- When a function is invoked to create an object, that object has a prototype property which is the function itself
- All instances share prototype property as reference
- Highly confusing, difficult to use and can lead to disastrous scope confusion
- To be implemented only with great caution (i.e., library implementation) and only when no better solution exists

CONCLUSION

- Objects can be created various ways – literal syntax is recommended
- Objects can have any number of properties which are easily accessed or changed
- Can be iterated with For In Loop
- Symbols are the preferred way of accessing properties internally
- Freeze and Seal both prevent object from changing
- Proxies can add further control to objects

{ }

OBJECT QUICK REFERENCE

Technique	Code
Create an object	let obj = {}
Set an object property	obj.name = “objecto”
Get an object property (bracket notation)	let name = obj.name;
Prevent new properties from being added	Object.seal(obj);
Prevent properties from changing	Object.freeze(obj)
Iterate through object properties	for (var prop in obj) {…}

OBJECT SCOPE

Ultimate JavaScript Objects

INTRODUCTION

- JavaScript has three scopes – *window*, *function* and *eval*
- Object scope is function scope when that function is a method of an object
- All functions that contain the *this* keyword are basically methods as *this* does not evaluate logically outside of a method and usually creates a wrong result

UNDERSTANDING *THIS*

- Value of *this* keyword is equal to the object itself, when *this* is inside one of its bound methods
- Elsewhere, *this* is equal to the *window* variable (usually not what you want!)
- Value of *this* is always provided – no need to ever set it
- Though innocuous, a huge amount of new code constructs are possible
- The value of *this* can change again inside normal functions (but not arrow functions)

WHAT'S UP WITH… STRICT MODE

- Activates streamlined version of JavaScript
- Activated by putting "use strict" at the top of script or function
- The *this* keyword never refers to the window in strict mode

```
"use strict"
```

BINDING SCOPE

- All functions have a bind method which creates a copy of the function with a different value of *this*
- By binding a function to an object, we create a method that can refer to properties of the object itself and its other methods
- Variables available in parent scope of original function still available

OBJECT COMPOSITION

- Many object oriented languages offer inheritance – a hierarchical way of sharing methods between objects
- ES6 introduces classes with standard (classical) inheritance (see next chapter)
- JavaScript offers prototypical inheritance, which is useful but can have puzzling behaviors
- **Composition** is a third way of sharing methods between classes
- Shares methods between objects on a flat hierarchy using *this* and *bind*
- Not possible in many other languages and therefore worth investigating

CONCLUSION

- Using the *this* keyword allows methods to refer to the object of which they are part
- The value of *this* can be changed in any function using the function's bind method
- Unusual cases of the *this* keyword can be avoided by using strict mode
- Creating class-like objects by using many methods and the this keyword is called composition

ES6 CLASSES

Ultimate JavaScript Objects

WHY ES6 CLASSES?

- Many attempts have been made over the years to give JavaScript OOP features like classes and inheritance
- JavaScript functions are class-like without supporting inheritance
- Support for features like private and public variables, as well as constructors, has long been hacked into existing structures
- ES6 Classes resolve all this by providing actual classes
- Use an ES6 to make lots of similar objects

WHAT IS AN ES6 CLASS?

- Any ES6 class can be compiled down into a JavaScript function with a large amount of sugar to make things like inherited methods possible
  - ES6 classes are therefore just fancy functions
- Way to access high-level functionality without libraries or hacks
- Can be more easy to understand than functions
- Suitable for many situations where the end result is a complex object with many methods and properties

SETTING UP ATOM FOR NODE DEVELOPMENT

- Install Atom (v1.8.0 or compatible)
- Install platformio-ide-terminal (2.1.0 or compatible plugin)
- Install NodeJS
- NPM install Nodemon

CREATING AN ES6 CLASS

- ES6 classes can be created with the *class* keyword
- Can also be created with *class expressions*
- Class cannot be defined twice
- Classes have a special property called constructor that is a function which runs whenever the class is created

CLASS METHODS AND PROPERTIES

- Classes can have properties and methods just like objects
- No truly private properties
- Methods have convenient syntax without the word *function*

INHERITANCE

- Classes can inherit from other classes
  - Gain access to methods and properties
- Classes that inherit from classes can themselves be inherited from
- Best used sparingly
- Indicate inheritance with the *extends* keyword
- Access parent methods with the *super* keyword

CLASS CONCLUSION

- Classes are versatile and convenient
- Not necessary to use
- At its heart, a function that returns objects
- Only works in ES6+ but can be readily compiled to ES5

COPYING JAVASCRIPT OBJECTS

Ultimate JavaScript Objects

COPYING JAVASCRIPT OBJECTS

- Learn about the complexities of copying JavaScript objects
- Learn several strategies for achieving end goals
- Apply strategies in a few demos
- Resolve complex copying situations involving nested children

COPYING VS REFERENCING

Copying

- Default for strings, numbers, Booleans
- Changing the original will not change any other copies
- Creates another variable in memory
- Destroying a reference will not destroy the object in memory

Making a Reference

- Default for arrays and objects
- Changing the original will change any other references
- Requires minimal additional memory

COPYING CONFUSION

- Classes sharing references to an object may affect each-other
  - i.e., global configuration object
- Some values (references to other objects) cannot be truly copied to another object
- Truly copying requires fully copying all children and their children, which may not be the intended result
  - May also be impossible – e.g., any object with a reference to self
- Objects with only primitives (strings, numbers, etc.) as properties can be copied without any complication at all

COPYING AN OBJECT

- An object can be copied in a number of ways
  - Create an empty object and loop through the properties of the original, adding them to the copy
  - Create an object with object literal syntax and include the copied variables as key-value pairs
- Object properties will be copied as references only (fine if this is the intended result)

INTO THE DEEP
COPYING NESTED PROPERTIES

- A copy of an object which also attempts to copy referenced values is called a “deep” copy
- Deep copying is not possible if loops exist within references
- Can lead to confusing results
  - i.e., reference to config object being deep copied, then the config is changed but the object’s behavior does not change
- OK as long as all references are eventually resolved
- Code must recurse through all children until each is resolved
- Can be handled by external libraries (next chapters)

COPYING CONCLUSION

- Copying objects is more difficult than copying strings or floats
- Objects are not copied when creating a new reference
- Copying objects with no references as properties is straightforward
- Copying objects with nested properties is challenging but not impossible
- Objects with loop references cannot be deep copied

JSON

Ultimate JavaScript Objects

WHAT IS JSON? (*JAY–SONN*)

- **J**avaScript **O**bject **N**otation
- Means of encoding data in object or array format
- Looks identical to Object Literal Notation
- Can store certain properties…
  - String
  - Float
  - Null / Undefined
  - Boolean
  - Object / Array
- But not…
  - Functions
  - References

```
{ "a" : 1 , "b" : "Jon" }
```

WHY JSON?

- Turns any object into a string
- Can be transmitted efficiently across HTTP
- Responses sent from server in JSON do not need to be heavily parsed by JavaScript
- Data models resemble related code (less confusion)
- Format of choice for REST applications

JSON VS XML

JSON

- Brief
- Easy to read
- Non-repetitive
- Drag and drop into any JavaScript application

XML

- Verbose
- Confusing
- Must repeat element names
- Requires hefty XML parsing applications
- Extra layer of confusion if not transformed correctly

SERIALIZING JAVASCRIPT OBJECTS

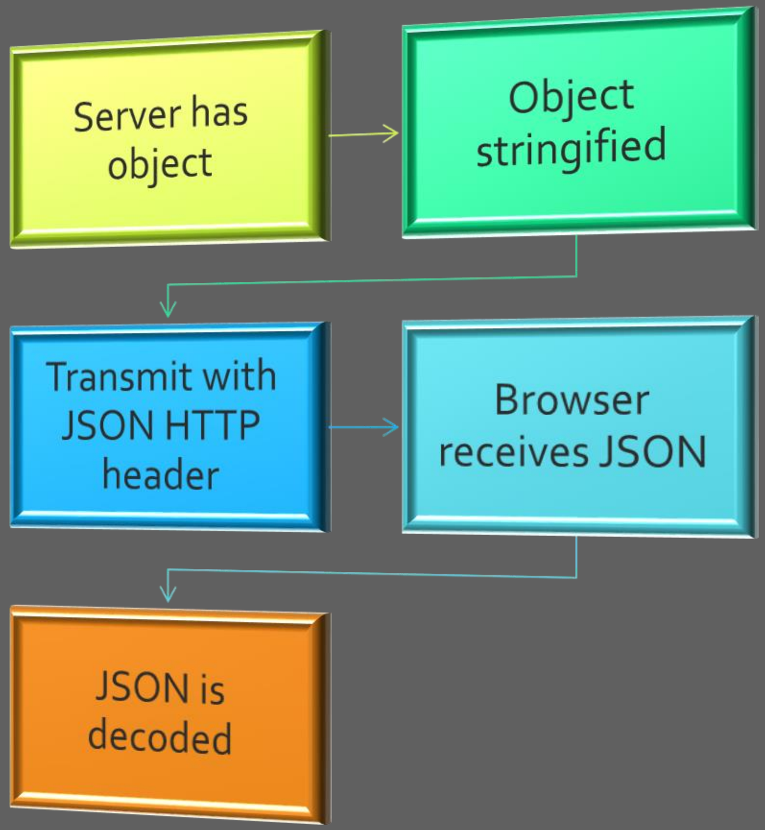
- Turning JavaScript objects into strings (or JSON) is called serializing
- Can be done with built-in JSON module
- Some properties cannot be serialized
- Serialized JavaScript objects can be sent across HTTP connections

PARSING JSON

- Parsing (or de-serializing) JSON turns it into a JavaScript object
- Only valid JSON can be parsed
- Also accomplished with built-in JSON module

TRANSMITTING JAVASCRIPT OBJECTS WITH HTTP

- Perfect for Node.js backend
- Minimal formatting required
- Express can be used to transmit JSON with built-in functions



JSON CONCLUSION

- JSON is lightweight and versatile
- Extremely easy to use in any JavaScript environment
- Good choice for communication between browser and server
- Easily parsed and stringified with JSON module

USING LODASH

Ultimate JavaScript Objects

WHAT IS LODASH?

- Library for manipulating collections in JavaScript
- Functionally identical to Underscore but better maintained
- Implements many useful functions for operating on objects
- Usable in any front-end or back-end JavaScript project

WHY LODASH?

- Objects are very complicated
  - Prototype creates countless corner-cases which are difficult to plan for
  - Scope and references add additional complexity
- Implementing a function to copy an object that covers every corner case is non-trivial
- Even if utility is implemented correctly, performance issues are also a challenge
- Don’t implement yourself – use Lodash!
- Backed by massive battery of tests and performance benchmarks - > 6,566 tests

IMPLEMENTING LODASH IN A PROJECT

- Lodash can be included as a script on any front-end page
- Can be installed via NPM for NodeJS server
- Once installed via NPM, can be accessed with *require* keyword
- Can eventually be packaged and sent to front-end by back-end

ASSIGNING OBJECTS WITH LODASH

- Copies the properties of any number of objects to another object
- Can be used to create a mutated or hybrid copy of an object
- Can be used to directly copy one object easily

```
_.assign({},{a:1})
```

INVERSION

- Used to swap all an object‘s keys and values
- Useful when dealing with malformed server data

PICKING VALUES

- Takes only the chosen properties from an object and creates a new object from that copy
- Used to create a limited (controlled) copy of an object
- Does not affect the original

MERGING

- Merges two arrays of objects into an array of combinations
- Good for dealing with multiple data sources from server
- Mutates an object

CONCLUSION

- Lodash methods exist for almost any object operation necessary
- Methods especially exist for those operations that are tricky or error-prone
- Using all methods effectively is key to become a very skilled developer
- Successful use of Lodash can save hundreds of hours on a project

CONCLUSION

Ultimate JavaScript Objects

REVIEW – ALL ABOUT OBJECTS

- Objects contain pairs of key-value pairs
- They can refer to themselves with *this* which allows object-oriented programming (this)
- The order of keys in an object can't be depended on
- All JavaScript primitives extend object
- ES6 introduces classes to create objects

REVIEW – WORKING WITH OBJECTS

- Objects can be created with object literal syntax or Object.create
- Properties can be easily accessed or modified using bracket or dot notation
- Objects can be iterated through with the *for in* loop
- Arrays of objects can be transformed with *array.map*
- Symbols make ideal keys as they cannot be duplicated accidentally
- Objects can frozen or sealed which restricts how they can be changed
- Object.prototype is confusing and should be avoided in favor of newer constructs (classes)

REVIEW – OBJECT SCOPE

- The *this* keyword changes in value based on where it is
- Functions must be bound to objects for the *this* keyword to have expected value
- Function, eval and global make up 3 kinds of scope in JavaScript
- Strict Mode prevents common this-related errors
- Classes can be built of many functions bound to their scope, which is called compositing

ES6 CLASSES

- ES6 classes incorporate decades worth of hacks into a clean interface
- Class is a function which produces an object (often containing methods and properties) when invoked
- Allows public properties and methods, constructors and inheritance
- Easy to use

COPYING OBJECT

- Copying objects presents numerous technical and logical complications
- String, Boolean and number properties can be copied easily
- Deep copying an object means to copy it *and* all of its descendent children
- Looping references make an object not deep copiable
- Ultimately best done with Lodash

JSON

- JavaScript Object Notation
- All JSON can be converted into a JavaScript object
- Most objects can be converted to JSON
- JSON strings can transfer data efficiently between front- and back-end
- Arguably better than XML
- Handled with the built-in JSON module

LODASH

- Lodash (similar to Underscore) is library of helper methods for JavaScript
- Countless methods for working with objects
- Reliable and well-tested
- Recommended for any tricky or copying actions needed

CONTINUE YOUR EDUCATION

- MDN Reference
- Lodash Reference
- JavaScript Weekly
- *Ultimate JavaScript Arrays* on Udemy (50% off!)

THANK YOU!

Ultimate JavaScript Objects