SUMMARY OF UDACITY COURSE

ES6 JAVASCRIPT IMPROVED

RATINGS: 4/5

LESSON 1 – SYNTAX

UNIT 1: Harmony, ES6, ES2015

- Harmony, ES6 and ES2015 all mean the same thing
- There are now new additions to the JS language

UNIT 2: Let and Const

- Two new ways to declare variables in JS: “let” and “const”
- Hoisting: is a result of how JS is interpreted by your browser, before any JS is executed all variables declared with “var” are hoisted, they are raised to the top of the function scope
- “let” and “const” eliminate this specific issue of hoisting because they are scoped “to the block” not to the function
- Rule for using “let” and “const”: snap shot
- They suggest ditching “var” for “let” and “const”

UNIT 3: Quiz – Using Let and Const (1-1)

UNIT 4: Template Literals

- Template Literals are essentially string literals that include embedded expressions
- It is useful for easier string interpolation
- They denoted with “` `” back ticks instead of “ ”
- They can contain placeholders using “${ }”
- They really shine and are easier to read compared to using “+” to concatenate strings

UNIT 5: Quiz – Build an HTML Fragment (1-2)

UNIT 6: Destructuring

- Destructuring borrows inspiration from languages like Python by allowing you to specify elements you want to extract from an array of object on the “left side of an assignment”.
- Snap shot

UNIT 7: Quiz – Destructuring Arrays (1-3)

UNIT 8: Object Literal Shorthand

- A recurring trend in ES6 is to remove unnecessary repetition in your code. By removing unnecessary repetition, your code becomes easier to read and more concise

- “for…of loop”If object properties have the same name as the variables being assigned to them, then you can drop the duplicate variable names
- Snap shots

UNIT 9: Lesson 1 Checkup

UNIT 10: Iteration

- Iteration: is the process of getting one item after another
- New to ES6: “iterable protocol” – allows JS objects to define or customize their iteration behavior, “for…of loop” – a loop that iterates over iterable objects

UNIT 11: Family of For Loops

- “for…of loop”: combines the strength of its siblings the “for loop” and “for…in loop”
- “for loop” is the most common type of loop there is, the downside is having to keep track of the counter and exit condition
- “for…in loop” improves upon the weakness of the for loop by eliminating the counting logic and exit condition, snap shot, they are discouraged when looping over arrays

UNIT 12: For…of Loop

- “for…of loop” used to loop over any type of data that is “iterable”
- Snap shots
- “It is good practice to use plural names for objects that are collections of values”
- You can stop or break a “for…of loop” at any time, snap shot

UNIT 13: Quiz – Writing a For…of Loop (1-4)

- Snap shot

UNIT 14: Spread…Operator

- The “spread operator”, written with three consecutive dots (“ … ”), is new in ES6 and gives you the ability to expand, or spread, “iterable objects” into multiple elements
- Snap shots
- “concat()” method
- Snap shots

UNIT 15: …Rest Parameter

- The “rest parameter”, also written with three consecutive dots (“ … ”), allows you to represent an indefinite number of elements as an array
- Snap shot
- The “rest parameter” can be seen like the opposite of the “spread operator”
- “arguments object”: snap shot

UNIT 16: Quiz – Using the Rest Parameter (1-5)

UNIT 17: Lesson 1 Summary

- Checkout functions in the next lesson

LESSON 2 – FUNCTIONS

UNIT 1: Updates to Functions

- We now have “arrow functions” in JS

UNIT 2: Arrow Functions

- Very similar to regular functions in behavior, but are quite different syntactically
- Snap shot
- “(parameter_list) => function_body”
- “.filter( )”

UNIT 3: Using Arrow Functions

- Regular functions can be either “function declarations” or “function expressions”, however “arrow functions” are always “expressions”
- Where “arrow function” is valid: snap shot
- Concise and Block Body Syntax: snap shot
- “arrow functions”: syntax is a lot shorter, it is easier to write and read, they automatically return when using the concise body syntax
- “there are definitely times you might not want to use an arrow function”

UNIT 4: Quiz – Convert Function into an Arrow Function (2-1)

UNIT 5: Arrow Functions Recap

- “arrow functions” handle the “this” keyword in a different manner compared to “regular functions”

UNIT 6: Arrow Functions and the “this” keyword

- For the value of “this” in regular expressions, it depends on “how the function is called”
- In arrow functions, it depends on “where it is located in the code”

UNIT 7: “this” and Regular Functions

- Don’t really understand
- Snap shots

UNIT 8: “this” and Arrow Functions

- More confusion
- Arrow functions inherit their “this” value from the surrounding context

UNIT 9: Default Function Parameters

- Default function parameters are quite easy to read since they are placed in the function's parameter list
- Snap shot

UNIT 10: Defaults and Destructuring arrays

- Combine default function parameters with destructuring
- Snap shots
- A little twist with arrays: snap shot
- They recommend going with object defaults with object destructuring

UNIT 11: Quiz – Using Default Function Parameters (2-2)

- Snap shot

UNIT 12: Class Preview

- "class", "extend", "constructor()" and "super" are a bunch of new keywords and syntax to play around with when creating JS classes
- Snap shot

UNIT 13: JS Illusion of Classes

- "class" in JS can be quite different from the way of other programming work
- JS is not an "class based language", uses functions to create objects, links objects together by "prototype"/ "inheritance"
- JS classes are a mirage of functions and prototype inheritance

UNIT 14: JS Classes

- ES6 classes are just a mirage and hide the fact that prototypal inheritance is actually going on under the hood
- Create a class in ES5: Snap shot
- ES5 Constructors, things to note: snap shot
- Create a class in ES6 using "class" and "constructor()" keywords: snap shot

UNIT 15: Convert a Function to a Class

- "constructor()" method is called when objects are instantiated

UNIT 16: Working with JS Classes

- JS classes are simply functions
- "," commas are not used to separate properties or methods in a Class
- "static method_name() { … }": used to create static method (method that can be accessed directly on the Class), snap shot

- Benefits of classes: less setup, clearly defined constructor function, everything is contained
- “under the hood JS class just uses prototypal inheritance”
- Use “new” keyword when instantiating a “class”

UNIT 17: Super and Extends

- Use “extends” keyword to set “subclass”
- Snap shots

UNIT 18: Extending Classes from ES5 to ES6

- Less code using ES6 class definitions compared to ES5 class definition

UNIT 19: Working with JavaScript Subclasses

- There is a lot less setup code and it is a lot cleaner syntax to create a subclass using “class”, “super”, “extends”
- In a subclass constructor function, before “this” can be used, a call to the super class must be made “super”

UNIT 20: Quiz – Building Classes and Subclasses (2-3)

- Snap shot

UNIT 21: Lesson 2 Summary

- Arrow functions, default values, JS classes

LESSON 3 – BUILT-INS

UNIT 1: New Built-ins

- We have a new set of Built-ins they make code easier

UNIT 2: Symbols Intro

- “symbols” are a new JS primitive data type
- It is a unique identifier, most often used to uniquely identify properties in an Object

UNIT 3: Symbols

- A symbol is a unique and immutable data type that is often used to identify object properties
- Create a symbol: “Symbol(optional_string_for_description)”
- Snap shot

UNIT 4: Iteration and Iterable Protocols

- “Iterable Protocol”: is used for defining and customizing the iteration behavior of Objects
- Some objects like arrays and string come built-in with this behavior
- “Iterator Object”: an object that conforms to the “iterator protocol”
- “Iterator Protocol”: is used to define a standard way that an object produces a sequence of values, it defines how an object will iterate, done using the “.next( )” method
- Snap shot

UNIT 5: Sets

- Set behaves like a mathematical set (unique values) and works similar to an array
- Set is not indexed-based, items can’t be accessed individually
- Set is basically an object that lets you store unique items
- Create set: “new Set([…])”

UNIT 6: Modifying Sets

- “.add()” / “.delete()”: methods used to add / delete items from the Set
- “.clear()”: used to delete all the items in a Sets
- “.add()” returns the Set if an item is successfully added, “.delete()” returns a Boolean depending on successful deletion

UNIT 7: Working with Sets

- “.size”: returns the length (number of items) of a Set
- “.has(item_name)”: return a Boolean if item_name is in the Set
- “.values()”: returns the values in a Set, it return a “SetIterator Object”

UNIT 8: Sets and Iterators

- Sets are built-in iterables, so we can use “for … of” loop
- Using “SetIterator”: snap shot

UNIT 9: Quiz – Using Sets

UNIT 10: WeakSets

- “WeakSet”: can only contain objects, are not iterable, does not have a “.clear()” method
- Create a “WeakSet”: “new WeakSet([…])
- “Garbage Collection”: the process of freeing up memory after it is no longer needed, snap shot

UNIT 11: Quiz – Working with WeakSets

UNIT 12: Maps

- Maps and WeakMaps
- Maps are to “objects” as Sets are to “array”

UNIT 13: Creating and Modifying Maps

- Map is an object that lets you store “key-value” pairs where both keys and the values can be objects, primitive values, or a combination of the two
- Create Map: “new Map()”
- “.set(key, value)”: to add key-values to the Map, snap shot
- “.delete(key)”: to remove a key-values
- “.clear()”: to empty the Map

UNIT 14: Working with Maps

- “.has(key)”: return a Boolean if key-value is in Map
- “.get (key)”: returns the value attacked to the key

UNIT 15: Looping Through Maps

- “MapIterator”: snap shot
- “.forEach()”: snap shot

UNIT 16: WeakMaps

- “WeakMap”: can only contain object as keys, not iterable, does not have a “.clear( )” method
- Some things about “Garbage Collection”

UNIT 17: Promises Intro

- Promises help us to handle Asynchronous request better

UNIT 18: Promises

- A promise will let you start some work that will be done “asynchronously” and let you get back to your regular work
- Create a promise: “new Promise (function_to_run_async)”, snap shot
- “reject”, “resolve”: snap shots
- A Promise will immediately return an object, can use “.then( )” method on it to notify us if the request made was successful or not
- “promise_obj.then (function_successful, function_failed)”: snap shot

UNIT 19: More Promises

- Being able to write JS Promises is very important, checkout the standalone course on promises
- Promises make it easier to do Asynchronous Coding easier to read and debug

UNIT 20: Proxies Intro

- Proxies will let one object stand in for another object

UNIT 21: Proxies

- To create a Proxy: “new proxy(object_it_will_to_be_proxy_for, list_of_methods_to_handle)”
- Snap shot
- “get” trap: it intercepts calls to “the properties”, snap shots
- “set” trap: it intercepts code that will “change a property”, snap shots
- There are a total of 13 traps that can be used by in a handler, snap shot
- “get” and “set” are the most used

UNIT 22: Proxies vs ES5 Getter/Setter

- Some functionality of proxy objects may be similar to existing ES5 getter/setter methods. But with proxies, you do not need to initialize the object with getters/setters for each property when the object is initialized
- Snap shots

UNIT 23: Proxies Recap

- “A proxy sits between a real object and the calling code”
- A trap is a function that will intercept calls to properties, if no trap is defined the default behavior is sent to the target object
- Proxies are a powerful new way to create and manage the interactions between objects

UNIT 24: Generators

- There is no way to pause a “function” in the middle of its execution
- “generator functions” new in ES6 they are “pausable functions”
- Create a generator function: “function* func_name() { }”, an asterisk after “function” keyword, snap shot

UNIT 25: Generators and Iterators

- When a “generator function” is invoked it doesn’t run the any of the code, instead it creates and returns an “iterator”, the iterator can then be used to execute the actual generator’s inner code, snap shot
- “yield” keyword new in ES6 can only be used inside “generator functions”, snap shot
- “yield data to the outside world”: snap shot

UNIT 26: Sending Data into/out of a Generator

- “.next( )” is used to pass data into the generator, snap shots
- Generators are a powerful new kind of function that is able to pause its execution while also maintaining its own state

UNIT 27: Lesson 3 Summary

- Some browsers have not caught up with some of this improvements to JS

LESSON 4 – PROFESSIONAL DEVELOPER – FU

UNIT 1: The Web is growing up

- The Web had a beginning but no end

UNIT 2: Old and New Browsers

- If you try to run ES6 code in an older browser it will not work
- ECMA International is an industry association that develops and overseas standards like JS and JSON

UNIT 3: ES6 Specification

- The specification lists the set of rules and guidelines on how the language is supposed to function
- Check it out: http://www.ecma-international.org/ecma-262/6.0/index.html
- You can also check out MDN for further clarifications on how things are meant to work

UNIT 4: Supported Features

- Each browser apart from Safari has a website that tracks its development status
- Snap shot
- Check out: http://kangax.github.io/compat-table/es6/

UNIT 5: The Web is Eternal

- You have to embrace the new ES6 technology, use the tools

UNIT 6: Polyfills

- Polyfill is a JS file that patches a hole by replicating some native feature that is missing

UNIT 7: Using Polyfills

- Polyfill is a piece of code (or plugiin) that provides the technology that you, the developer, expect the browser to provide natively
- Check out: https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-Browser-Polyfills
- Snap shot

UNIT 8: Polyfill Walkthrough

UNIT 9: Other Uses of Polyfills

- JS is the language used to create a polyfill, but a polyfill doesn't just patch up missing JS features, there are polyfills for all sorts of browser features

- Check out: https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-Browser-Polyfills

UNIT 10: Transpiling

- Compiler: changes a language level of abstraction, it turns a "source code" to a "lower level language (machine code)"
- Transpiler: does not change the language level of abstraction, it turns a "source code" to a "target code"
- Transpile ES6 code to ES5 code

UNIT 11: Using Babel

- Babel is the most popular JS transpiler
- Check out: https://babeljs.io/repl/#?browsers=defaults%2C%20not%20ie%2011%2C%20not%20ie_mob%2011&build=&builtIns=false&spec=false&loose=false&code_lz=Q&debug=false&forceAllTransforms=false&shippedProposals=false&circleciRepo=&evaluate=true&fileSize=false&timeTravel=false&sourceType=module&lineWrap=false&presets=es2015&prettier=false&targets=&version=7.10.5&externalPlugins=
- Babel transforms code from one language to another through "plugins", it has a "preset" which is a set of "plugins" bundled together
- Babel uses both Node and NPM to distribute its plugins

UNIT 12: Transpiling Walkthrough

UNIT 13: Transpiling Recap

- It is important to stay on top of all the changes JS is going through

UNIT 14: Course Summary