A guided tour to JavaScript

Oge Marques, PhD

Introduction

This document guides you through the most important aspects of JavaScript using the online version of the book "Eloquent JavaScript 2/e" (http://eloquentjavascript.net/) as a reference. It is structured as a step-by-step guide. It is best that you follow it in the intended sequence.

Part 1- The basics

1. Read the **Introduction** chapter (http://eloquentjavascript.net/00_intro.html).
2. (OPTIONAL) Play with the integrated "sandbox" (e.g., using the while loop example below), to get used to its interface and functionality.
   Click anywhere within the code to open the integrated editor / console and click on the top-right corner to perform one of the following operations: run code, revert to original code, reset sandbox, or deactivate editor.

   Here is the same program in JavaScript:

3. Read **Chapter 1** (http://eloquentjavascript.net/01_values.html) and try **all** of its examples, to make sure that you understand them.
4. Pay special attention to (and make sure that you understand) the examples below:

```
console.log(NaN == NaN)
// → false

console.log(8 * null)
// → 0
console.log("5" - 1)
// → 4
console.log("5" + 1)
// → 51
console.log("five" * 2)
// → NaN
console.log(false == 0)
// → true
```

```
console.log(null == undefined);
// → true
console.log(null == 0);
// → false
```

5. Read **Chapter 2** (http://eloquentjavascript.net/02_program_structure.html) and try **all** of its examples, to make sure that you understand them.
6. **Important!** ***Do not use any of the reserved keywords below*** as a name to your variables, functions, etc.

```
break case catch class const continue debugger
default delete do else enum export extends false
finally for function if implements import in
instanceof interface let new null package private
protected public return static super switch this
throw true try typeof var void while with yield
```

7. Make sure you understand the difference between **alert**, **prompt**, and **confirm**.
8. Pay special attention to (and make sure that you understand) the examples below:

```
var theNumber = Number(prompt("Pick a number", ""));
if (!isNaN(theNumber))
  alert("Your number is the square root of " +
        theNumber * theNumber);
else
  alert("Hey. Why didn't you give me a number?");
```

```
var num = Number(prompt("Pick a number", "0"));

if (num < 10)
  alert("Small");
else if (num < 100)
  alert("Medium");
else
  alert("Large");
```

```
var number = 0;
while (number <= 12) {
  console.log(number);
  number = number + 2;
}
```

```
var result = 1;
var counter = 0;
while (counter < 10) {
  result = result * 2;
  counter = counter + 1;
}
console.log(result);
// → 1024
```

```
for (var number = 0; number <= 12; number = number + 2)
  console.log(number);
// → 0
// → 2
//   … etcetera
```

```
var result = 1;
for (var counter = 0; counter < 10; counter = counter + 1)
  result = result * 2;
console.log(result);
// → 1024
```

```
for (var current = 20; ; current++) {
  if (current % 7 == 0)
    break;
}
console.log(current);
// → 21
```

```
for (var number = 0; number <= 12; number += 2)
  console.log(number);
```

```
switch (prompt("What is the weather like?")) {
  case "rainy":
    console.log("Remember to bring an umbrella.");
    break;
  case "sunny":
    console.log("Dress lightly.");
  case "cloudy":
    console.log("Go outside.");
    break;
  default:
    console.log("Unknown weather type!");
    break;
}
```

---

9. (OPTIONAL) Try the 3 exercises at the end of the chapter.
   IMPORTANT: refrain from looking at the official solution (no matter how easy / tempting this might be)! You will learn much more if you follow this advice!

Part 2- Functions, data structures, and arrays

1. Read **Chapter 3** (http://eloquentjavascript.net/03_functions.html) and try **all** of its examples, to make sure that you understand them.
2. Focus on the **differences between functions in JavaScript and other languages**, e.g., in JS functions can be anonymous.
3. Be mindful of the subtle way by which **local and global variable scopes** can be distinguished (what a difference the keyword 'var' makes…). You may want to play with the example below to fully understand it.

```
var x = "outside";

var f1 = function() {
  var x = "inside f1";
};
f1();
console.log(x);
// → outside

var f2 = function() {
  x = "inside f2";
};
f2();
console.log(x);
// → inside f2
```

4. For an additional twist on the topic (something technically known as *lexical scoping*), checkout the **landscape** function example.
5. Pay special attention to this remark: "People who have experience with **other programming languages** might expect that **any block of code between braces produces a new local environment**. But **in JavaScript, functions are the only things that create a new scope**."
6. Checkpoint – Answer the question: Why does the code below works, even though the function is declared **below** the point where it is used?

```
1 console.log("The future says:", future());
2
3 function future() {
4   return "We STILL have no flying cars.";
5 }
```

7. Checkpoint – Answer the question: Why is it dangerous (and not recommended!) to use the code below?

```
function example() {
  function a() {} // Okay
  if (something) {
    function b() {} // Danger!
  }
}
```

8. JavaScript allows a variable number of parameters to be passed to a function. Play with the example below to fully understand this aspect:

```
function power(base, exponent) {
  if (exponent == undefined)
    exponent = 2;
  var result = 1;
  for (var count = 0; count < exponent; count++)
    result *= base;
  return result;
}
```

9. The concept of **closure** is key to understanding functions in JS. Start with the example below, which is not surprising.

```
function wrapValue(n) {
  var localVariable = n;
  return function() { return localVariable; };
}

var wrap1 = wrapValue(1);
var wrap2 = wrapValue(2);
console.log(wrap1());
// → 1
console.log(wrap2());
// → 2
```

10. Then, move to the example below (which may cause your head to spin…) ☺

```
function multiplier(factor) {
  return function(number) {
    return number * factor;
  };
}

var twice = multiplier(2);
console.log(twice(5));
// → 10
```

11. Appreciate the evolution of the code for the “farm inventory” example, which goes from a rough first draft, through an improved second version, and finally an elegant third version.

12. Summarize with your own words (and try to fully understand) the author's advice on these two topics: (1) Do not add **cleverness** unless you are absolutely sure you're going to need it. (2) **Pure** functions are good, but **side effects** are a fact of life.
13. (OPTIONAL) Try the 3 exercises at the end of the chapter.
    IMPORTANT: refrain from looking at the official solution (no matter how easy / tempting this might be)! You will learn much more if you follow this advice!
14. Read **Chapter 4** (http://eloquentjavascript.net/04_data.html) and try **all** of its examples, to make sure that you understand them.
15. Get a good grasp of how **arrays** work in JS. They have similarities with other languages (e.g., they are 0-based), but also show some differences (e.g., the **length** property and the **push** method).
16. Moving on to **objects**, try to wrap your head around this statement (and its implications, especially if you have a C++, C#, or Java background): "Values of the type *object* are arbitrary collections of properties, and we can add or remove these properties as we please."
17. Play with the example below:

```
var day1 = {
  squirrel: false,
  events: ["work", "touched tree", "pizza", "running",
           "television"]
};
console.log(day1.squirrel);
// → false
console.log(day1.wolf);
// → undefined
day1.wolf = false;
console.log(day1.wolf);
// → false
```

18. Use the example below to understand the meaning of **delete** and **in**:

```
var anObject = {left: 1, right: 2};
console.log(anObject.left);
// → 1
delete anObject.left;
console.log(anObject.left);
// → undefined
console.log("left" in anObject);
// → false
console.log("right" in anObject);
// → true
```

19. Objects can be elements of an array. Use the example below to exercise your understanding of this.

```
var journal = [
  {events: ["work", "touched tree", "pizza",
            "running", "television"],
   squirrel: false},
  {events: ["work", "ice cream", "cauliflower",
            "lasagna", "touched tree", "brushed teeth"],
   squirrel: false},
  {events: ["weekend", "cycling", "break",
            "peanuts", "beer"],
   squirrel: true},
  /* and so on... */
];
```

20. The example below refers to the important aspects of mutability and aliases. Play with it to understand what it is trying to teach,

```
var object1 = {value: 10};
var object2 = object1;
var object3 = {value: 10};

console.log(object1 == object2);
// → true
console.log(object1 == object3);
// → false

object1.value = 15;
console.log(object2.value);
// → 15
console.log(object3.value);
// → 10
```

21. Feel free to skip the examples related to correlation and “The lycanthrope’s log”.
22. Array recap: make sure you understand the meaning and usage of: push, pop, shift, unshift, indexOf, and lastIndexOf.
23. The **slice** and **concat** functions can be very useful! Play with the example below to fully understand them.

```
function remove(array, index) {
  return array.slice(0, index)
    .concat(array.slice(index + 1));
}
console.log(remove(["a", "b", "c", "d", "e"], 2));
// → ["a", "b", "d", "e"]
```

24. Learn how to use length, toUpperCase, slice, indexOf, trim, and charAt with variables of type string.

25. Can you appreciate why the version of addEntry shown below is better than its predecessor (shown immediately after)?

```
function addEntry(squirrel) {
  var entry = {events: [], squirrel: squirrel};
  for (var i = 1; i < arguments.length; i++)
    entry.events.push(arguments[i]);
  journal.push(entry);
}
addEntry(true, "work", "touched tree", "pizza",
         "running", "television");
```

```
var journal = [];

function addEntry(events, didITurnIntoASquirrel) {
  journal.push({
    events: events,
    squirrel: didITurnIntoASquirrel
  });
}

addEntry(["work", "touched tree", "pizza", "running",
          "television"], false);
```

26. Explore the **Math** object and its built-in methods.
27. (OPTIONAL) Try all the exercises at the end of the chapter.
    IMPORTANT: refrain from looking at the official solution (no matter how easy / tempting this might be)! You will learn much more if you follow this advice!
28. Read **Chapter 5** (http://eloquentjavascript.net/05_higher_order.html) and try **the selected examples** mentioned below, to make sure that you understand them.
29. Make sure that you understand why the abstraction *forEach* is **better** than *logEach* in the code snippets below.

```
function logEach(array) {
  for (var i = 0; i < array.length; i++)
    console.log(array[i]);
}


function forEach(array, action) {
  for (var i = 0; i < array.length; i++)
    action(array[i]);
}
```

30. The JSON format for data representation is ubiquitous these days and you must understand its syntax and meaning, as well as the most important methods associated with a JSON object, such as *stringify* and *parse*. See example below:

```
var string = JSON.stringify({name: "X", born: 1980});
console.log(string);
// → {"name":"X","born":1980}
console.log(JSON.parse(string).born);
// → 1980
```

31. Download the ancestry.js (from http://eloquentjavascript.net/code/ancestry.js) and try the example below using your favorite editor (outside of the eBook):

```
var ancestry = JSON.parse(ANCESTRY_FILE);
console.log(ancestry.length);
// → 39
```

32. Using the same file (ancestry.js), try the example below:

```
function filter(array, test) {
  var passed = [];
  for (var i = 0; i < array.length; i++) {
    if (test(array[i]))
      passed.push(array[i]);
  }
  return passed;
}

console.log(filter(ancestry, function(person) {
  return person.born > 1900 && person.born < 1925;
}));
```

33. Using the same file (ancestry.js), try the example below:

```
console.log(ancestry.filter(function(person) {
  return person.father == "Carel Haverbeke";
}));
// → [{name: "Carolus Haverbeke", …}]
```

34. Using the same file (ancestry.js), try the example below, which shows how to use **map** ("a method that transforms an array by applying a function to all of its elements and building a new array from the returned values, 'mapped' to a new form by the function"):

```
function map(array, transform) {
  var mapped = [];
  for (var i = 0; i < array.length; i++)
    mapped.push(transform(array[i]));
  return mapped;
}

var overNinety = ancestry.filter(function(person) {
  return person.died - person.born > 90;
});
console.log(map(overNinety, function(person) {
  return person.name;
}));
// → ["Clara Aernoudts", "Emile Haverbeke",
//    "Maria Haverbeke"]
```

35. Using the same file (ancestry.js), try the example below, which shows how to use **reduce** ("a method that summarizes the contents of an array, by computing a single value from it"):

```
function reduce(array, combine, start) {
  var current = start;
  for (var i = 0; i < array.length; i++)
    current = combine(current, array[i]);
  return current;
}

console.log(reduce([1, 2, 3, 4], function(a, b) {
  return a + b;
}, 0));
// → 10
```

36. This is a challenging one! Carefully explore the example below to see *map* and *reduce* used together in a meaningful way (with some *filter* thrown into the mix for good measure…)

```
function average(array) {
  function plus(a, b) { return a + b; }
  return array.reduce(plus) / array.length;
}
function age(p) { return p.died - p.born; }
function male(p) { return p.sex == "m"; }
function female(p) { return p.sex == "f"; }

console.log(average(ancestry.filter(male).map(age)));
// → 61.67
console.log(average(ancestry.filter(female).map(age)));
// → 54.56
```

37. Explore the *bind* method, which "creates a new function that will call the original function but with some of the arguments already fixed." using the example below:

```
var theSet = ["Carel Haverbeke", "Maria van Brussel",
              "Donald Duck"];
function isInSet(set, person) {
  return set.indexOf(person.name) > -1;
}

console.log(ancestry.filter(function(person) {
  return isInSet(theSet, person);
}));
// → [{name: "Maria van Brussel", …},
//    {name: "Carel Haverbeke", …}]
console.log(ancestry.filter(isInSet.bind(null, theSet)));
// → … same result
```

38. (OPTIONAL) Try all the exercises at the end of the chapter.
IMPORTANT: refrain from looking at the official solution (no matter how easy / tempting this might be)! You will learn much more if you follow this advice!

Part 3- Objects

1. Read **Chapter 6** (http://eloquentjavascript.net/06_object.html) and try **the selected examples** mentioned below, to make sure that you understand them.
2. This chapter discusses objects in JS in greater depth. It will probably bring back memories of "Foundations of CS" class…
3. Start with the simple examples below and make sure you fully understand them:

```
var rabbit = {};
rabbit.speak = function(line) {
  console.log("The rabbit says '" + line + "'");
};

rabbit.speak("I'm alive.");
// → The rabbit says 'I'm alive.'
```

```
function speak(line) {
  console.log("The " + this.type + " rabbit says '" +
              line + "'");
}
var whiteRabbit = {type: "white", speak: speak};
var fatRabbit = {type: "fat", speak: speak};

whiteRabbit.speak("Oh my ears and whiskers, " +
                  "how late it's getting!");
// → The white rabbit says 'Oh my ears and whiskers, how
//   late it's getting!'
fatRabbit.speak("I could sure use a carrot right now.");
// → The fat rabbit says 'I could sure use a carrot
//   right now.'
```

4. Try to understand the difference between *apply* and *call*. See example below:

```
speak.apply(fatRabbit, ["Burp!"]);
// → The fat rabbit says 'Burp!'
speak.call({type: "old"}, "Oh my.");
// → The old rabbit says 'Oh my.'
```

5. Now we are entering the topic of **prototypes** in JavaScript. **This is extremely important!** Follow the steps below closely.

6. Try to understand the relationship between a “proto” rabbit and any rabbit derived from it, using the example below:

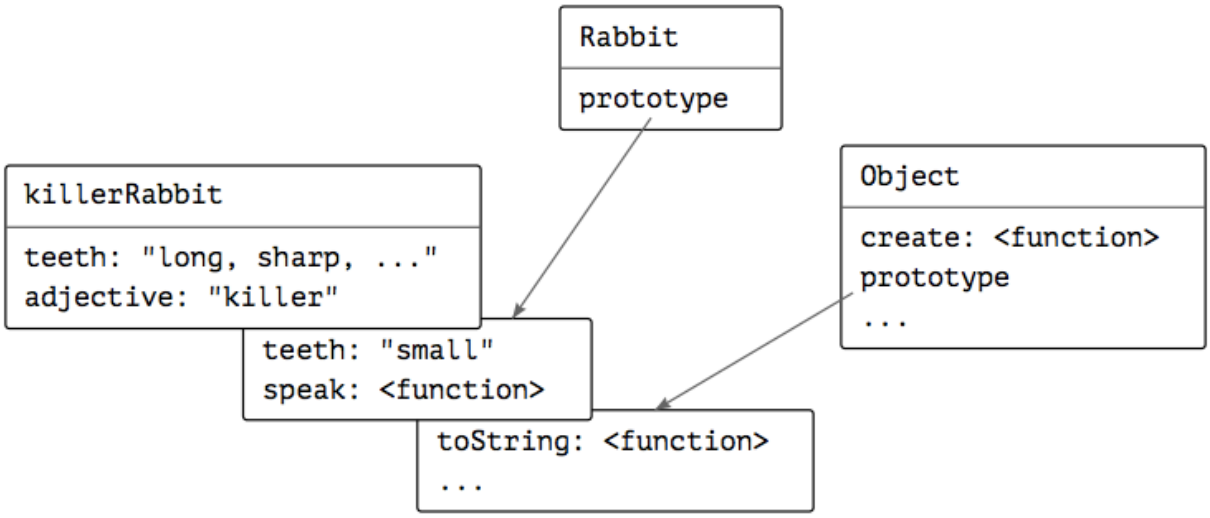
```
var protoRabbit = {
  speak: function(line) {
    console.log("The " + this.type + " rabbit says '" +
                line + "'");
  }
};
var killerRabbit = Object.create(protoRabbit);
killerRabbit.type = "killer";
killerRabbit.speak("SKREEEE!");
// → The killer rabbit says 'SKREEEE!'
```

7. Learn how constructors work in JS. See example below:

```
function Rabbit(type) {
  this.type = type;
}

var killerRabbit = new Rabbit("killer");
var blackRabbit = new Rabbit("black");
console.log(blackRabbit.type);
// → black
```

```
Rabbit.prototype.speak = function(line) {
  console.log("The " + this.type + " rabbit says '" +
              line + "'");
};
blackRabbit.speak("Doom...");
// → The black rabbit says 'Doom...'
```

8. The prototype mechanism is akin to the OOP concept of inheritance. The following diagram and example code illustrate that. Make sure you understand them!



```
Rabbit.prototype.teeth = "small";
console.log(killerRabbit.teeth);
// → small
killerRabbit.teeth = "long, sharp, and bloody";
console.log(killerRabbit.teeth);
// → long, sharp, and bloody
console.log(blackRabbit.teeth);
// → small
console.log(Rabbit.prototype.teeth);
// → small
```

9. (OPTIONAL) Try the remaining examples in the chapter. They are very relevant from an OOP / programming language viewpoint, but **not** essential to your projects in this course.
10. (OPTIONAL) Try all the exercises at the end of the chapter.
    IMPORTANT: refrain from looking at the official solution (no matter how easy / tempting this might be)! You will learn much more if you follow this advice!
11. (OPTIONAL) Read **Chapter 7** (http://eloquentjavascript.net/07_elife.html) and learn how to build a simple game in JS.
12.

Part 4- Programming in JavaScript: practical (and advanced) tips

1. Read **Chapter 8** (http://eloquentjavascript.net/08_error.html) and learn more about testing and debugging in JS.
2. Use the example below to understand 'use strict':

```
function canYouSpotTheProblem() {
  "use strict";
  for (counter = 0; counter < 10; counter++)
    console.log("Happy happy");
}

canYouSpotTheProblem();
// → ReferenceError: counter is not defined
```

3. Use the example below to understand how to throw and catch exceptions in JS:

```
function promptDirection(question) {
  var result = prompt(question, "");
  if (result.toLowerCase() == "left") return "L";
  if (result.toLowerCase() == "right") return "R";
  throw new Error("Invalid direction: " + result);
}

function look() {
  if (promptDirection("Which way?") == "L")
    return "a house";
  else
    return "two angry bears";
}

try {
  console.log("You see", look());
} catch (error) {
  console.log("Something went wrong: " + error);
}
```

4. Learn how to use a 'finally' block finally to ensure a piece of code is *always* run when a block finishes. Refer to the example below:

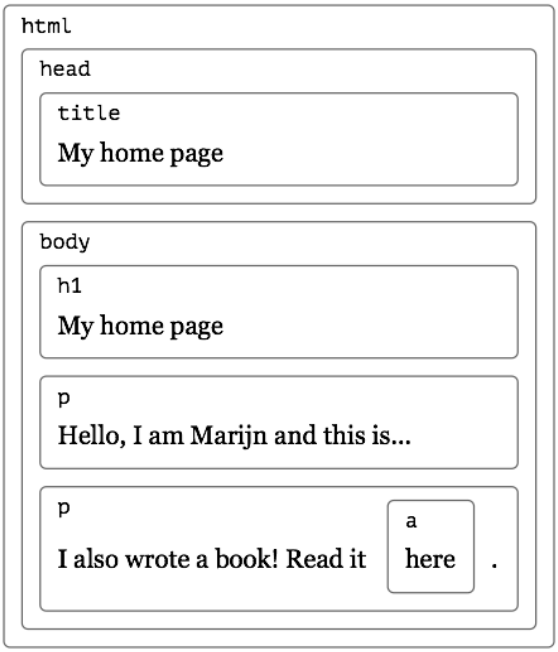
```
function withContext(newContext, body) {
  var oldContext = context;
  context = newContext;
  try {
    return body();
  } finally {
    context = oldContext;
  }
}

try {
  withContext(5, function() {
    if (context < 10)
      throw new Error("Not enough context!");
  });
} catch (e) {
  console.log("Ignoring: " + e);
}
// → Ignoring: Error: Not enough context!

console.log(context);
// → null
```

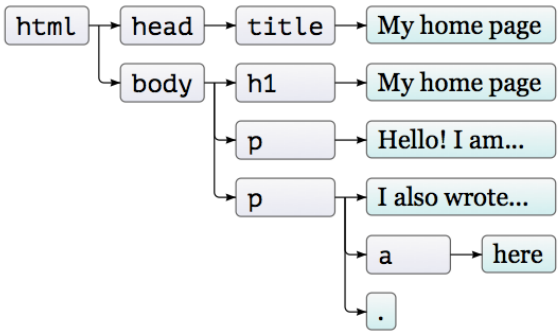
5. (OPTIONAL) Try all the exercises at the end of the chapter.
   IMPORTANT: refrain from looking at the official solution (no matter how easy / tempting this might be)! You will learn much more if you follow this advice!
6. (OPTIONAL) Read **Chapter 9** (http://eloquentjavascript.net/09_regexp.html) and learn more regular expressions in JS.
7. (OPTIONAL) Read **Chapter 10** (http://eloquentjavascript.net/10_modules.html) and learn more how to organize your code into **modules (and associated interfaces)**, which "provide structure to bigger programs by separating the code into different files and namespaces."
8. (OPTIONAL) Read **Chapter 11** (http://eloquentjavascript.net/11_language.html) **only if** you want to learn more about how to build an entirely new programming language using JS.

Part 5- JavaScript and the browser
(extremely important!!)

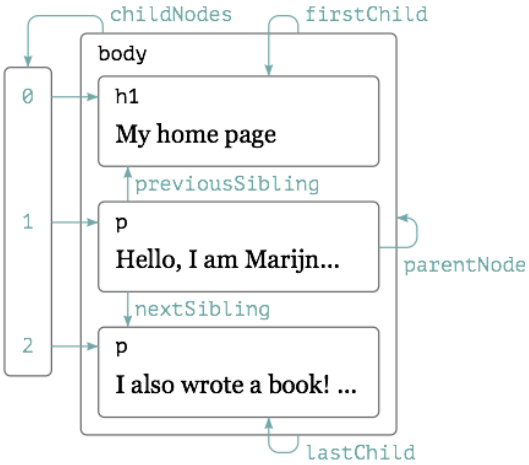
1. (OPTIONAL) Read **Chapter 12** (http://eloquentjavascript.net/12_browser.html) to learn more about the relationships between JS and the browser, or "what happens under the hood" when your browser interprets and executes JS code.
2. Read **Chapter 13** (http://eloquentjavascript.net/13_dom.html) to learn more about the DOM (Document Object Model). Manipulating, accessing, and traversing the DOM are essential tasks for building rich, interactive, client-side apps using JavaScript (and jQuery).
3. As a refresher, look at the diagrams below to understand the relationship between HTML markup and two possible graphical representations of the DOM: "floor plan" and tree.

```
<!doctype html>
<html>
  <head>
    <title>My home page</title>
  </head>
  <body>
    <h1>My home page</h1>
    <p>Hello, I am Marijn and this is my home page.</p>
    <p>I also wrote a book! Read it
      <a href="http://eloquentjavascript.net">here</a>.</p>
  </body>
</html>
```





4. Study the diagram below to ensure that you understand parent/children/siblings relationships in the DOM.



5. **Important remark:** for the remaining examples in this Part, **do not get too attached to the syntax! Focus on the concepts!** Soon enough, you will use **jQuery** to do these types of DOM manipulations… and jQuery's syntax is infinitely friendlier than original (raw) JavaScript.

6. Use the example below to understand document.getElementById():

```
<p>My ostrich Gertrude:</p>
<p><img id="gertrude" src="img/ostrich.png"></p>

<script>
  var ostrich = document.getElementById("gertrude");
  console.log(ostrich.src);
</script>
```

7. Use the example below to understand insertBefore() (and modify it to achieve other changes in the DOM):

```
<p>One</p>
<p>Two</p>
<p>Three</p>

<script>
  var paragraphs = document.body.getElementsByTagName("p");
  document.body.insertBefore(paragraphs[2], paragraphs[0]);
</script>
```

8. Explain this particular modification to the example (results at the bottom):

```
<ul id="myList">
    <li>One</li>
    <li>Two</li>
    <li>Three</li>
</ul>

<script>
  var items = document.getElementsByTagName("li");
  var list = document.getElementById("myList");
  list.insertBefore(items[2], items[0]);

  var node = document.createElement("li");
  var textNode = document.createTextNode("Four");
  node.appendChild(textNode);
  list.appendChild(node);
</script>
```

- Three
- One
- Two
- Four

9. Run (and explain) the “Cat in the Hat” example below:

```
<p>The <img src="img/cat.png" alt="Cat"> in the
  <img src="img/hat.png" alt="Hat">.</p>

<p><button onclick="replaceImages()">Replace</button></p>

<script>
  function replaceImages() {
    var images = document.body.getElementsByTagName("img");
    for (var i = images.length - 1; i >= 0; i--) {
      var image = images[i];
      if (image.alt) {
        var text = document.createTextNode(image.alt);
        image.parentNode.replaceChild(text, image);
      }
    }
  }
</script>
```

10. This is a more challenging example of DOM manipulation. After running it, try to draw the DOM tree before and after the code executes, to ensure that you fully understood what it does.

```
<blockquote id="quote">
  No book can ever be finished. While working on it we learn
  just enough to find it immature the moment we turn away
  from it.
</blockquote>

<script>
  function elt(type) {
    var node = document.createElement(type);
    for (var i = 1; i < arguments.length; i++) {
      var child = arguments[i];
      if (typeof child == "string")
        child = document.createTextNode(child);
      node.appendChild(child);
    }
    return node;
  }

  document.getElementById("quote").appendChild(
    elt("footer", "—",
        elt("strong", "Karl Popper"),
        ", preface to the second editon of ",
        elt("em", "The Open Society and Its Enemies"),
        ", 1950"));
</script>
```

11. Use the example below to understand the concept of **attributes**.

```
<p data-classified="secret">The launch code is 00000000.</p>
<p data-classified="unclassified">I have two feet.</p>

<script>
  var paras = document.body.getElementsByTagName("p");
  Array.prototype.forEach.call(paras, function(para) {
    if (para.getAttribute("data-classified") == "secret")
      para.parentNode.removeChild(para);
  });
</script>
```

12. Skip the remaining of Chapter 13. It contains CSS manipulations that are better done using jQuery. More on that later.
13. Read **Chapter 14** (http://eloquentjavascript.net/14_event.html) to learn more about event handling.
14. Try the example below and explain what it does.

```
<p>Click this document to activate the handler.</p>
<script>
  addEventListener("click", function() {
    console.log("You clicked!");
  });
</script>
```

15. Try the example below and explain the differences between this example and the one in the previous step.

```
<button>Click me</button>
<p>No handler here.</p>
<script>
  var button = document.querySelector("button");
  button.addEventListener("click", function() {
    console.log("Button clicked.");
  });
</script>
```

16. Use the example below to understand the concept of **adding and removing event listeners**.

```
<button>Act-once button</button>
<script>
  var button = document.querySelector("button");
  function once() {
    console.log("Done.");
    button.removeEventListener("click", once);
  }
  button.addEventListener("click", once);
</script>
```

17. Use the example below to understand the concept of **event objects**. [*Yes, middle mouse buttons are a thing of the past, but this doesn't make the example any less useful. ;-)*]

```
<button>Click me any way you want</button>
<script>
  var button = document.querySelector("button");
  button.addEventListener("mousedown", function(event) {
    if (event.which == 1)
      console.log("Left button");
    else if (event.which == 2)
      console.log("Middle button");
    else if (event.which == 3)
      console.log("Right button");
  });
</script>
```

18. Stopping undesired event propagation (often called *event bubbling*) is a useful skill to master. There are multiple ways of achieving it[1]. This is but one example.

```
<p>A paragraph with a <button>button</button>.</p>
<script>
  var para = document.querySelector("p");
  var button = document.querySelector("button");
  para.addEventListener("mousedown", function() {
    console.log("Handler for paragraph.");
  });
  button.addEventListener("mousedown", function(event) {
    console.log("Handler for button.");
    if (event.which == 3)
      event.stopPropagation();
  });
</script>
```

19. Use the example below to understand the *target* property of the *event* object.

```
<button>A</button>
<button>B</button>
<button>C</button>
<script>
  document.body.addEventListener("click", function(event) {
    if (event.target.nodeName == "BUTTON")
      console.log("Clicked", event.target.textContent);
  });
</script>
```

[1] In fact, the topic of event propagation is full of details, several of which are browser-dependent.

20. Use the example below to understand the concept of *preventing the browser's default actions* when an event occurs.

```
<a href="https://developer.mozilla.org/">MDN</a>
<script>
  var link = document.querySelector("a");
  link.addEventListener("click", function(event) {
    console.log("Nope.");
    event.preventDefault();
  });
</script>
```

21. Use the example below to understand **keyboard-based events**.

```
<p>This page turns violet when you hold the V key.</p>
<script>
  addEventListener("keydown", function(event) {
    if (event.keyCode == 86)
      document.body.style.background = "violet";
  });
  addEventListener("keyup", function(event) {
    if (event.keyCode == 86)
      document.body.style.background = "";
  });
</script>
```

22. (OPTIONAL) Use the example below to understand **mouse-based events** in the context of a very primitive drawing program in JS.

```
<style>
  body {
    height: 200px;
    background: beige;
  }
  .dot {
    height: 8px; width: 8px;
    border-radius: 4px; /* rounds corners */
    background: blue;
    position: absolute;
  }
</style>
<script>
  addEventListener("click", function(event) {
    var dot = document.createElement("div");
    dot.className = "dot";
    dot.style.left = (event.pageX - 4) + "px";
    dot.style.top = (event.pageY - 4) + "px";
    document.body.appendChild(dot);
  });
</script>
```

23. Before you get carried away by cool examples such as the resizable bar, be mindful that graphics programming for things like drawing with JavaScript can be substantially assisted by jQuery, jQueryUI, several libraries[2], and good knowledge of the HTML5 <canvas> element.
24. Use the example below to understand the concepts of *focus* and *blur*.

```
<p>Name: <input type="text" data-help="Your full name"></p>
<p>Age: <input type="text" data-help="Age in years"></p>
<p id="help"></p>

<script>
  var help = document.querySelector("#help");
  var fields = document.querySelectorAll("input");
  for (var i = 0; i < fields.length; i++) {
    fields[i].addEventListener("focus", function(event) {
      var text = event.target.getAttribute("data-help");
      help.textContent = text;
    });
    fields[i].addEventListener("blur", function(event) {
      help.textContent = "";
    });
  }
</script>
```

25. Glance through the examples related to setting and clearing **timeOut** timers.
26. (OPTIONAL) Read **Chapter 15** (http://eloquentjavascript.net/15_game.html) to learn how to build a rudimentary platform game in JS.

Part 6- Forms, canvas and more!

1. (OPTIONAL) Read **Chapter 16** (http://eloquentjavascript.net/16_canvas.html) to learn about the HTML5 <canvas> element.
2. (OPTIONAL) Take the "Draw with JavaScript" lesson at Codecademy: https://www.codecademy.com/en/courses/web-beginner-en-SWM11/0/1 (It should take 15 minutes or less).
3. Read **Chapter 17** (http://eloquentjavascript.net/17_http.html) to learn more about the HTTP protocol, the GET and POST methods, the the XMLHttpRequest object (responsible for the "AJAX magic"), and HTTPS.
4. Run and try to understand the two variants (XML and JSON) of the **fruits** example, below.

```
var req = new XMLHttpRequest();
req.open("GET", "example/fruit.xml", false);
req.send(null);
console.log(req.responseXML.querySelectorAll("fruit").length);
```

[2] See http://modeling-languages.com/javascript-drawing-libraries-diagrams/ for a recently compiled list.

```
var req = new XMLHttpRequest();
req.open("GET", "example/fruit.json", false);
req.send(null);
console.log(JSON.parse(req.responseText));
// → {banana: "yellow", lemon: "yellow", cherry: "red"}
```

5. Read **Chapter 18** (http://eloquentjavascript.net/18_forms.html) to learn more about (client-side) **form processing**.
6. Play with the example below to get acquainted with the most common form elements.

```
<p><input type="text" value="abc"> (text)</p>
<p><input type="password" value="abc"> (password)</p>
<p><input type="checkbox" checked> (checkbox)</p>
<p><input type="radio" value="A" name="choice">
   <input type="radio" value="B" name="choice" checked>
   <input type="radio" value="C" name="choice"> (radio)</p>
<p><input type="file"> (file)</p>
```

7. Play with the example below and try to submit the form. What happened?

```
<form action="example/submit.html">
  Name: <input type="text" name="name"><br>
  Password: <input type="password" name="password"><br>
  <button type="submit">Log in</button>
</form>
<script>
  var form = document.querySelector("form");
  console.log(form.elements[1].type);
  // → password
  console.log(form.elements.password.type);
  // → password
  console.log(form.elements.name.form == form);
  // → true
</script>
```

8. Contrast the example below with the one in the previous step. What has changed? Why? How is it relevant?

```
<form action="example/submit.html">
  Value: <input type="text" name="value">
  <button type="submit">Save</button>
</form>
<script>
  var form = document.querySelector("form");
  form.addEventListener("submit", function(event) {
    console.log("Saving value", form.elements.value.value);
    event.preventDefault();
  });
</script>
```

9. Run the example below and suggest a scenario in which it could be useful.

```
<input type="text"> length: <span id="length">0</span>
<script>
  var text = document.querySelector("input");
  var output = document.querySelector("#length");
  text.addEventListener("input", function() {
    output.textContent = text.value.length;
  });
</script>
```

10. After running the example below in its original form, modify it to make it do other (visible) things.

```
<input type="checkbox" id="purple">
<label for="purple">Make this page purple</label>
<script>
  var checkbox = document.querySelector("#purple");
  checkbox.addEventListener("change", function() {
    document.body.style.background =
      checkbox.checked ? "mediumpurple" : "";
  });
</script>
```

11. Try the example below and explain how it works.

```
Color:
<input type="radio" name="color" value="mediumpurple"> Purple
<input type="radio" name="color" value="lightgreen"> Green
<input type="radio" name="color" value="lightblue"> Blue
<script>
  var buttons = document.getElementsByName("color");
  function setColor(event) {
    document.body.style.background = event.target.value;
  }
  for (var i = 0; i < buttons.length; i++)
    buttons[i].addEventListener("change", setColor);
</script>
```

12. Try the example below and explain how it works.

```
<select multiple>
  <option value="1">0001</option>
  <option value="2">0010</option>
  <option value="4">0100</option>
  <option value="8">1000</option>
</select> = <span id="output">0</span>
<script>
  var select = document.querySelector("select");
  var output = document.querySelector("#output");
  select.addEventListener("change", function() {
    var number = 0;
    for (var i = 0; i < select.options.length; i++) {
      var option = select.options[i];
      if (option.selected)
        number += Number(option.value);
    }
    output.textContent = number;
  });
</script>
```

13. Try the example below and explain how it works.

```
<input type="file">
<script>
  var input = document.querySelector("input");
  input.addEventListener("change", function() {
    if (input.files.length > 0) {
      var file = input.files[0];
      console.log("You chose", file.name);
      if (file.type)
        console.log("It has type", file.type);
    }
  });
</script>
```

14. Try the example below and explain how it works.

```
<input type="file" multiple>
<script>
  var input = document.querySelector("input");
  input.addEventListener("change", function() {
    Array.prototype.forEach.call(input.files, function(file) {
      var reader = new FileReader();
      reader.addEventListener("load", function() {
        console.log("File", file.name, "starts with",
                    reader.result.slice(0, 20));
      });
      reader.readAsText(file);
    });
  });
</script>
```

15. The next examples will teach you about local storage.
16. Try the example below, modify it, and use the browser's developer tools to better understand how it works.

```
localStorage.setItem("username", "marijn");
console.log(localStorage.getItem("username"));
// → marijn
localStorage.removeItem("username");
```

17. Study the "a simple note-taking application" example carefully and try to modify it in meaningful ways.
18. (OPTIONAL) Read **Chapter 19** (http://eloquentjavascript.net/19_paint.html) to learn how to create a simple "paint" program in JavaScript.

Part 7- Node.js (the server side of JavaScript)

(outside the scope of this class – included here just for the sake of completeness)

1. (OPTIONAL) Read **Chapter 20** (http://eloquentjavascript.net/20_node.html) to learn about node.js and npm.
2. (OPTIONAL) Check out http://davidherron.com/book/2015-09-14/books-and-videos-so-you-can-easily-learn-nodejs-programming for a list of books, tutorials, videos, and resources on node.js.
3. (OPTIONAL) Read **Chapter 21** (http://eloquentjavascript.net/21_skillsharing.html) to learn how to build a fully functional app .

THE END
