# W03 Reading Notes Mobile Application - CODE NINJA

## “this” in methods:

It’s common that an object method needs to access the information stored in the object to do its job.

For instance, the code inside user.sayHi() may need the name of the user.

**To access the object, a method can use the this keyword.**

The value of this is the object “before dot”, the one used to call the method.

For instance:

let user = {

name: "John",

age: 30,

sayHi() {

// "this" is the "current object"

alert(this.name);

}

};

user.sayHi(); // John

It is unreliable to use other things besides “this.”

## CH. 4 Functions (Recap from last week):

### **Function Declarations**

To define a function literal we can use a function declaration:

function hello(){

console.log('Hello World!');

}

This starts with the function keyword and is followed by the name of the function, which in this case is called ' hello ', followed by parentheses.

Calling the function is simple:

hello();

<< 'Hello world!'

**Parameters** and **arguments**

are terms that are often used interchangeably to represent values provided for the function as an input.

You can use as many parameters as you like when defining functions. For example, the following function finds the mean of any three numbers:

function mean(a,b,c){

return (a+b+c)/3;

}

mean(1, 3, 6);

<< 3.3333333333333335

* Functions are first-class objects that behave the same way as other values.
* Function literals can be defined using the function declaration, or by creating a*function expression*by assigning an anonymous function to a variable.
* All functions return a value. If this is not explicitly stated, the function will return undefined.
* A parameter is a value that is written in the parentheses of a function declaration and can be used like a variable inside the function’s body.
* An argument is a value that is provided to a function when it is invoked.
* The arguments variable is an array-like object that allows access to each argument provided to the function using index notation.
* The rest operator can be used to access multiple arguments as an array.
* Default arguments can be supplied to a function by assigning them to the parameters.
* Arrow functions are a new shorthand notation that can used for writing anonymous functions in ES6.
* Function declarations can be invoked before they are defined because they are hoisted to the top of the scope, but function expressions cannot be invoked until after they are defined.
* A*callback*is a function that is provided as an argument to another function.

Function Practice:

1

2

function isEven(a) {

3

if (a == 50) return true;

4

else if (a == 75) return false;

5

else return "??";

6

}

7

​

8

console.log(isEven(50));

9

// → true

10

console.log(isEven(75));

11

// → false

12

console.log(isEven(-1));

13

// → ??

true

false

??

## Chapter 5 Objects:

An object in JavaScript is a self-contained set of related values and functions. They act as a collection of named properties that map to any JavaScript value such as strings, numbers, booleans, arrays and functions. If a property’s value is a function, it is known as a **method**.

“To create an object literal, simply enter a pair of curly braces. The following example creates an empty object that is assigned to the variable spiderman :”

const spiderman = {};

This is the preferred way of doing it, you do not want to follow the “New Object()” way.

Accessing properties or an object are as follows:

superman.name

<< 'Superman'

“You can also access an object’s properties using bracket notation ― the property is represented by a string inside square brackets, so needs to be placed inside single or double quotation marks:”

superman['name']

<< 'Superman'

“Dot notation is much more common, but bracket notation has a few advantages: it's the only way to access nonstandard property and method names that don’t follow the variable naming rules. It also lets you evaluate an expression and use it as the property key:”

## **Calling Methods**

To call an object’s method we can also use dot or bracket notation. Calling a method is the same as invoking a function, so parentheses need to be placed after the method name:

superman.fly()

<< 'Up, up and away!'

superman['fly']()

<< 'Up, up and away!'

It is good for you to review the built in objects that Javascript has to offer.

### **JSON**

[JavaScript Object Notation, or JSON](http://www.json.org/),

It is an extremely popular lightweight data-storage format that is used by a large number of services for data serialization and configuration.

The parse() method takes a string of data in JSON format and returns a JavaScript object:

JSON.parse(batman);

<< { name: 'Batman',

'real name': 'Bruce Wayne',

height: 74,

weight: 210,

hero: true,

villain: false,

allies: [ 'Robin', 'Batgirl', 'Superman' ] }

“These methods are particularly useful when it comes to sending data to, and receiving data from, a web server using Ajax requests (see Chapter 13) – or when using localStorage to store data on a user’s machine (see Chapter 14). JSON data is easy to exchange between different services, as most languages and protocols are able to interpret data as strings of text – and they only need to be stored as a basic text file.”

Here are some of the useful built in objects from the reading that would be beneficial to review CTRL F in order to find them in the chapter 5 reading:

* **THE MATH OBJECT**
* **THE DATE OBJECT**
* **REGULAR EXPRESSIONS**

#### **Creating Regular Expressions**

There are two ways to create a regular expression. The first, and preferred way, is to use the literal notation of writing the regular expression between forward slashes that we’ve already seen:

const pattern = /[a-zA-Z]+ing$/;

The special characters associated with the regular expressions:

### **Special Characters**

In a regular expression, there are a number of characters that have a special meaning, commonly known as metacharacters:

* . matches any character, except line breaks
* \w matches any word character, and is equivalent to [A-Za-z0-9\_]
* \W matches any non-word character, and is equivalent to [\^A-Za-z0-9\_]
* \d matches any digit character, and is equivalent to [0-9]
* \D matches any non-digit character, and is equivalent to [^0-9]
* \s matches any whitespace character, and is equivalent to [ \t\r\n\f]
* \S matches any non-whitespace character, and is equivalent to [^ \t\r\n\f]

## Chapter 6: The Document Object Model:

### **What is the DOM?**

The Document Object Model, or DOM for short, represents an HTML document as a network of connected nodes that form a tree-like structure.

### **Legacy DOM Shortcut Methods**

There are some methods from DOM Level 0 that can still be employed to access commonly used elements. These include:

* Document.body returns the body element of a web page, as we saw in the previous example.
* Document.images returns a node list of all the images contained in the document.
* Document.links returns a node list of all the <a> elements and <area> elements that have an href attribute.
* Document.anchors returns a node list of all the <a> elements that have a name attribute.
* Document.forms returns a node list of all the forms in the document. This will be used when we cover forms in Chapter 8.

Different way to use the getElement is:

getElementById()

getElementByTagName()

getElementByClassName()

For example, the following could be used instead of document.getElementById() :

document.querySelector('#bats');

<< <li class="vigilante hero" id="bats">Batman</li>

And this could be used instead of document.getElementsByClassName :

document.querySelectorAll('.hero');

<< NodeList [<li class="hero">, <li id="bats">, <li class="hero">]

Note that this is not alivenode list. See the section later in this chapter for more details about live node lists.

## **Chapter 6 Summary**

* The Document Object Model is a way of representing a page of HTML as a tree of nodes.
* The document.getElementById() , document.getElementsByClassName() , document.getElementsByTagNames() and document.querySelector() can be used to access elements on a page.
* The parentNode() , previousSibling() , nextSibling() , childNodes() and children() methods can be used to navigate around the DOM tree.
* An element’s attributes can be accessed using the getAttribute() method, and updated using the setAttribute() method.
* The createElement() and createTextNode() methods can be used to create dynamic markup on the fly.
* Markup can be added to the page using the appendChild() and insertBefore() methods.
* Elements can be replaced using the replaceChild() method, and removed using the removeChild() method.
* The innerHTML property can be used to insert raw HTML directly into the DOM.
* The CSS properties of an element can be changed by accessing the style property.

## Chapter 7: Events

#### **The click Event**

The click event occurs when a user clicks with the mouse, presses the Enter key, or taps the screen, making it a very useful all-round event covering many types of interaction.

It would be beneficial for you Ben to review the many different types of event listeners.