Tutorial 3: JS and ES6

Basic Javascript

Open any text editor, create a file namely index.html
Write the following code inside <script> tag

Open the file in a browser, ie. Chrome to see the result. Right click on the page and select Inspect/Console to see the log.

1. Fundamentals

Variable
To declare a variable:

```
var a  = 10;
var str = 'Hello'; //both single or double is fine
```

Alternatively, we can declare variable with let

```
let a = 10;
let str = 'Hello';
```

When we don't want to change the values of a variable, we can use const

```
const a = 10;
a = 15 //not possible as a is a const
```

Expression:

```
var name = 'Henry';
var str = 'Hello' + name;

var a = 5
var b = 6
c = a + b
console.log(c) // will be 11
```

Note that semicolon (;) after each statement is optional

If-else, for loop:
Very similar to Java

2. Hoisting

In JS function can be used after or even before it is declared. This feature is called hoisting, raising all the function declaration into the top of JS program. For example:

```
hello('Henry')

function hello(name){
	console.log('Hello '+name);
}
```

As JS also support anonymous function (function has no name), it is important to remember that with function expression, there is no hoisting. It means that a function expression must be defined before it can be called. The following is not working:

```
greeting('David')

const greeting = function(name){
	console.log('Greeting '+name);
}
```

This line `greeting('David')` must be called after the function expression.

3. Comparison

In JS, we compare values by using three equal signs, not two. Why?

I found on StackOverflow this best explanation why we should use === and !== instead of == and !=.

```
0 == false   // true
0 === false  // false, because they are of a different type
1 == "1"     // true, automatic type conversion for value only
1 === "1"    // false, because they are of a different type
null == undefined // true
null === undefined // false
'0' == false // true
'0' === false // false
```

4. Object

In Java, objects are typically created from class. In JS, objects are often initialized without any class declaration.

We can quickly have an object like this:

```
let student = {
        name: 'David',
        age: 30,
        display: function(){
                console.log(`Name: ${this.name}, age: ${this.age} `)
        }
}
```

This student object has 2 properties and 1 method.

We call method or access properties by:

```
console.log(student.age)
student.display()
```

Note that in the display function, we use String Template to format the output string. A string template uses a pair of backtick (`). For inserting a value, we use ${} expression. There are other ways to create an object such as Object.create or new Object() etc. But object initializer is a very handy way.

Exercises:

- Repeat all the example code above by your own
- Slightly change to see how it works

Tutorial 3. Advanced Javascript

In order to get started with modern framework such as React, Redux, Angular, you definitely need to upgrade yourself with new and modern knowledge of Javascript. Please don't go ahead until you really understand all the topics in this chapter. They all are prerequisites to start React-Redux.

Time flies and life changes dramatically. Do so frameworks. The same applies to JS. Dramatic changes happen to JS recently. Most of us who had worked with JS before became outdated with its new features. Learn it or not, that's the matter.

Continue to run the following code snippet:

1. **Arrow function**

This is one of the most fascinating of JS. Many times, I got mad when reading code that uses this new syntax.

Before:

```
function hello(name){
  alert('Hello '+ name);
}
```

Now:

```
(name)=>{
  alert('Hello '+ name);
}
```

What did we do? We strip off the keyword function and substitute it by a fat arrow (=>). We name it fat arrow because it is comprised of an equal sign (=) and a greater (>). Unlike, a thin arrow (->) which is a dash and a greater sign. After that, we placed the arrow between function params and body.

Wait, how can we call this function? We need to assign it to a variable:

```
const hello = (name)=> {alert('Hello '+name)}

hello('Thanh'); // will alert Hello Thanh
```

Wait, can we execute this anonymous function without assigning a variable pointing to it? Yes, please use the magic parenthesis (), like this:

```
(()=>{alert('Self running function')})()
```

But with this, we can't pass a param such as name.

Summary:

- Use arrow function when we don't want to have a function name (anonymous function). Have you been in trouble with thinking of a name for a function? We all are in such a situation.
- Arrow function makes our code cleaner and more natural.

2. Array: splice, join, push, map, filter

Array seems very trivial in JS. However, there are a couple of methods that are used a lot in many frameworks but cause us perplexed.

+ push: yes, we can insert an element to an array using push. Let's take an example:

```
var a = ['car', 'phone', 'clothes']
a.push('book');
console.log(a); //book is inserted
```

We can get the most recently inserted item by calling pop()

```
console.log(a.pop()); //book
console.log(a); //book is removed
```

+ join: this is an easy method but sometimes confusing. join() concatenates all items of an array and convert them into a string. Take the following example:

```
let b = ['math', 'english', 'science'];
console.log(b.join('-')); // math-english-science
```

+ splice vs filter

JS does not provide a remove() method to delete an item of an array directly. You must use splice(index, len) with index as a position to remove and len as number of items to be removed.

```
let c = ['fish', 'dog', 'cat']
c.splice(1,1); //dog will be removed
console.log(c);
```

splice() will modify the array but there is a method that does not update the array, i.e. filter.

```
let e = ['carnation', 'rose', 'daisy']
let f = e.filter((flower, i)=>{
        return flower!=='rose'
})
console.log(e); //unchanged
console.log(f); //rose is removed
```

In this example, we pass an anonymous function using arrow to the filter method. We return only flowers that are NOT rose. All the returned values will be added up to form a new array (array f).

```
let g = e.map(
(flower, i) =>{
        return 'flower '+ flower;
}
)
console.log(g); // ["flower carnation", "flower rose", "flower daisy"]
```

We can see that all items are prefixed with 'flower'.

+ **Apply, call | bind**

In JS, we execute a function by putting parentheses after the function name. How can we call a function that is a member of an object like the following:

```
var student = {
        name: 'Henry',
        age: 20,
        showDetail: function(){
                console.log(`This is ${this.name}, ${this.age} years old`);
        }
}
```

We call showDetail function by:

```
student.showDetail();
```

It is very straight-forward. But if it is just like that, JS is boring and we can't reuse the showDetail function for other objects. Call and Bind function helps reuse a function with other objects. Cool?

It is just like this:

```
student.showDetail.apply({name: 'David', age: 30})
```

You can see that the object {name: 'David', age: 30} has been passed into the showDetail function. And because showDetail does not depend on any context (object), we can declare it independently:

```
showDetail: function(){
        console.log(`This is ${this.name}, ${this.age} years old`);
}
```

We can put any object into this function. The object will become **this.**

call() function works more or less the same way. What we need to care about is the powerful function named bind().

bind() function returns a new function. Remember it WON'T execute the function. If we want to execute it, we have to put ()

```
var newfunction = student.showDetail.bind({name: 'David', age: 30}) //to bind it
newfunction(); //to run it
```

Another way to use call/apply/bind is described belows

```
function theFunction(name, profession) {
   console.log("My name is " + name + " and I am a " + profession +".");
}
theFunction("John", "fireman");
```

```
theFunction.apply(undefined, ["Susan", "school teacher"]);
theFunction.call(undefined, "Claude", "mathematician");
theFunction.call(undefined, ...["Matthew", "physicist"]); // used with the spread
operator
```

Source:
https://stackoverflow.com/questions/1986896/what-is-the-difference-between-call-and-apply

Exercises:

1. **Repeat**
   For each of the code snippets above, repeat with a slight change to see their effect.
2. **Arrow function.**
   Define the following functions using arrows
   a. A function to add 2 numbers
   b. A function to multiply 2 numbers
   c. A function to check if age of someone is sufficient to drive car (i.e. above 18)
   d. A function to calculate area of a square
   e. A function to calculate circumference of a circle
   f. A function to calculate diagonal of a rectangle
   g. A function to check if a name starts with a character
   h. A function to get all the first initial characters of a string and return that as a word

3. Array functions

a. Define an array named flowers with 3 flowers.
   - Push one more flower
   - Pop 2 flowers

b. Define an array named animals with 3 animals.
   - Push one more animal
   - Pop 1 animal
   - Run a for loop to pop all of animals

c. Define an array named flowers with 3 flowers.
   - Use map function to return all flowers with prefix beautiful
   - Use splice to return 2 items starting at position 1
   - Use filter to remove all items starting with r

d. Define an array named animals with 3 animals.
   - Use map function to return all flowers with prefix cute
   - Use splice to return 1 items starting at position 2
   - Use filter to remove all items starting with d

4. Apply, call, bind

a. Define a student object with 2 properties id, name, and a function hello() to print Hello + name
    - Use apply and call to apply the function with a new object
    - Use bind to assign it to a variable and execute it

b. Define a car object with 2 properties brand, color, and a function paint() to print Paint + color
    - Use apply and call to apply the function with a new object
    - Use bind to assign it to a variable and execute it