Summary ES6

Content

Introduction

What is ECMAScript?

ECMAScript (ES) is a scripting language specification standardized by ECMAScript International. It is used by applications to enable client-side scripting. The specification is influenced by programming languages like Self, Perl, Python, Java etc. Languages like JavaScript, Jscript and ActionScript are governed by this specification.

ECMAScript is a scripting language standard and specification

- JavaScript
- Jscript
- ActionScript

What is ES6?

The most recent version of ECMAScript / JavaScript
Very significant update with many changes
First major update since ES5 (2009)
ES6 and ES2015 are the same thing

Goals of ES6

- Be taken more seriously
- Fix some issues from ES5
- Backward compatibility – ES5 should work in ES6
- Modern syntax
- Better scaling & better for big applications
- New feature in standard library

Compatibility

- Still quite a ways to go for some browsers
- Latest versions of Chrome and Firefox are almost there
- Transpilers can be used to compile ES6 code to ES5
    - Babel
    - Traceur
    - Closure

Overview

❖ History

JavaScript was developed by Brendan Eich, a developer at Netscape Communications Corporation, in 1995.JavaScript started life with the name Mocha, and was briefly named LiveScript before being officially renamed to JavaScript. It is a scripting language that is executed by the browser, i.e. on the client's end. It is used in conjunction with HTML to develop responsive webpages.

❖ Feature

ECMA Script6's implementation discussed here covers the following new features:

- Support for constants
- Block Scope
- Arrow Functions
- Extended Parameter Handling
- Template Literals
- Extended Literals
- Enhanced Object Properties
- De-structuring Assignment
- Modules
- Classes
- Iterators
- Generators
- Collections
- New built in methods for various classes
- Promises

Theory and Examples

1. Syntax

Syntax defines the set of rules for writing programs. Every language specification defines its own syntax.

- Variables: Represents a named memory block that can store values for the program.
- Literals: Represents constant/fixed values.
- Operators: Symbols that define how the operands will be processed.
- Keywords: Words that have a special meaning in the context of a language.
- Modules: Represents code blocks that can be reused across different programs/scripts.
- Comments: Used to improve code readability. These are ignored by the JavaScript engine.
- Identifiers: These are the names given to elements in a program like variables, functions, etc.

2. Variables

A variable, by definition, is “a named space in the memory” that stores values. In other words, it acts as a container for values in a program. Variable names are called identifiers.

JavaScript Variable Scope

- Global scope: A variable with global scope can be accessed from within any part of the JavaScript code.
- Local Scope: A variable with a local scope can be accessed from within a function where it is declared.

3. Events

An event is an action or occurrence recognized by the software. It can be triggered by a user or the system. Some common examples of events include a user clicking on a button, loading the web page, clicking on a hyperlink and so on. Following are some of the common HTML events.

- Example:

```
<script type="text/javascript">
function sayHello() {  document.write ("Hello World")  }
</script>
<body>
<p> Click the following button and see result</p>
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
```

4. Cookies

Cookies are the default mechanism used by browsers to store data pertaining to a user's session.

- Example:

```
<html>
<head>
<script type="text/javascript">
function WriteCookie()
{
var now = new Date();
now.setMonth( now.getMonth() - 1 );
cookievalue = escape(document.myform.customer.value) + ";"
document.cookie="name=" + cookievalue;
document.cookie = "expires=" + now.toUTCString() + ";"
document.write("Setting Cookies : " + "name=" + cookievalue );
}
</script>
</head>
<body>
<form name="formname" action="">
Enter name: <input type="text" name="customer"/>
<input type="button" value="Set Cookie" onclick="WriteCookie()"/>
</form>
</body>
</html>
```

5. Objects

An object is an instance which contains a set of key value pairs. Unlike primitive data types, objects can represent multiple or complex values and can change over their life time. The values can be scalar values or functions or even array of other objects.

- Example:

```
"use strict"

var det = { name:"Tom", ID:"E1001" };
var copy = Object.assign({}, det);
console.log(copy);
for (let val in copy)
{
   console.log(copy[val])
}
```

6. Arrays

An array is a homogenous collection of values. To simplify, an array is a collection of values of the same data type. It is a user-defined type.

- Example:

```
var nums=new Array(12,13,14,15)
console.log("Printing original array.....")
nums.forEach(function(val,index){
    console.log(val)
})
nums.reverse()  //reverses the array element
console.log("Printing Reversed array....")
nums.forEach(function(val,index){
    console.log(val)
})
```

7. Date

The Date object is a datatype built into the JavaScript language. Date objects are created with the new Date () as shown in the following syntax.

- Example:

```
var dt = new Date(1993, 6, 28, 14, 39, 7);
console.log( "Formated Date : " + dt.toDateString() )
```

8. Math

The math object provides you properties and methods for mathematical constants and functions. Unlike other global objects, Math is not a constructor. All the properties and methods of Math are static and can be called by using Math as an object without creating it.

- Example:

```
var value1 = Math.random();
console.log("First Test Value : " + value1 );
var value2 = Math.random();
console.log("Second Test Value : " + value2 );
var value3 = Math.random();
console.log("Third Test Value : " + value3 );
var value4 = Math.random();
console.log("Fourth Test Value : " + value4 );
```

9. RegExp

A regular expression is an object that describes a pattern of characters. Regular expressions are often abbreviated “regex” or “regexp”.

- Example:

```
var re = new RegExp( "string" );

        if ( re.ignoreCase ){
           console.log("Test1-ignoreCase property is set");
        }
        else
        {
          console.log("Test1-ignoreCase property is not set");
        }
        re = new RegExp( "string", "i" );

        if ( re.ignoreCase ){
           console.log("Test2-ignoreCase property is set");
        }
        else
        {
          console.log("Test2-ignoreCase property is not set");
        }
```

10. HTML DOM

A document object represents the HTML document that is displayed in that window. The document object has various properties that refer to other objects which allow access to and modification of the document content.

The way a document content is accessed and modified is called the Document Object Model, or DOM. The objects are organized in a hierarchy. This hierarchical structure applies to the organization of objects in a web document.

- Example:

```
<script type="text/javascript">
function myFunc() {
var ret = document.title;
alert("Document Title : " + ret );
var ret = document.URL;
alert("Document URL : " + ret );
var ret = document.forms[0];
alert("Document First Form : " + ret );
var ret = document.forms[0].elements[1];
alert("Second element : " + ret );
}  </script>
<h1 id="title">This is main title</h1>
<form name="FirstForm">
<input type="button" value="Click Me" onclick="myFunc();" />
<input type="button" value="Cancel">
</form>
<form name="SecondForm">
<input type="button" value="Don't ClickMe"/>
</form>
```

11. Collections

ES6 introduces two new data structures: Maps and Sets.

- Maps: This data structure enables mapping a key to a value.
- Sets: Sets are similar to arrays. However, sets do not encourage duplicates.

❖ Maps

The Map object is a simple key/value pair. Keys and values in a map may be primitive or objects.

- Example:

```
var myMap = new Map();
myMap.set("id", "admin");
myMap.set("pass", "admin@123");
var itr=myMap.keys();
console.log(itr.next().value);
console.log(itr.next().value);
```

❖ Sets

A set is an ES6 data structure. It is similar to an array with an exception that it cannot contain duplicates. In other words, it lets you store unique values. Sets support both primitive values and object references.

- Example:

```
var set = new Set();
set.add(10);
set.add(20);
set.add(30);
console.log(`Size of Set before delete() :${set.size}`);
console.log(`Set has 10 before delete() :${set.has(10)}`);
set.delete(10)
console.log(`Size of Set after delete() :${set.size}`);
console.log(`Set has 10 after delete() :${set.has(10)}`);
```

12. Classes

Object Orientation is a software development paradigm that follows real-world modelling. Object Orientation, considers a program as a collection of objects that communicates with each other via mechanism called methods. ES6 supports these object-oriented components too.

Class: A class in terms of OOP is a blueprint for creating objects. A class encapsulates data for the object.

- Example :

```
'use strict'

class PrinterClass
{
doPrint()
{console.log("doPrint() from Parent called…") }
}
class StringPrinter extends PrinterClass
{
doPrint()
{
super.doPrint()
console.log("doPrint() is printing a string…") }
}
var obj= new StringPrinter()
obj.doPrint()
```

13. Promises

Promises are a clean way to implement async programming in JavaScript (ES6 new feature). Prior to promises, Callbacks were used to implement async programming. Let's begin by understanding what async programming is and its implementation, using Callbacks.

- Example:

```
<script>
function notifyAll(fnSms, fnEmail)
{
   setTimeout(function()
   {
      console.log('starting notification process');
      fnSms();
      fnEmail();
   }, 2000);
}
notifyAll(function()
{
   console.log("Sms send ..");
},
function()
{
   console.log("email send ..");
});
console.log("End of script"); //executes first or not blocked by
others
</script>
```

14. Error Handing

There are three types of errors in programming: Syntax Errors, Runtime Errors, and Logical Errors.

- Example:

```
var a = 100;
var b = 0;
try
{
if (b == 0 )
{
throw(“Divide by zero error.”);
}
else
{
var c = a / b;
}
}
catch( e )
{
console.log("Error: " + e );
}
```

15. Validations

Form validation normally used to occur at the server, after the client had entered all the necessary data and then pressed the Submit button. If the data entered by the client was incorrect or was simply missing, the server would have to send all the data back to the client and request that the form be resubmitted with the correct information. This was really a lengthy process which used to put a lot of burden on the server.

- Basic Validation: First of all, the form must be checked to make sure all the mandatory fields are filled in. It would require just a loop through each field in the form and check for data.
- Data Format Validation: Secondly, the data that is entered must be checked for correct form and value. Your code must include appropriate logic to test the correctness of data.

```
<script type="text/javascript"> function validate()  {
   if( document.myForm.Name.value == "" )  {
     alert( "Please provide your name!" );
document.myForm.Name.focus() ;      return false;  }
   if( document.myForm.EMail.value == "" )  {
     alert( "Please provide your Email!" );
document.myForm.EMail.focus() ;      return false; }
   if( document.myForm.Zip.value == "" ||           isNaN(
document.myForm.Zip.value ) ||
document.myForm.Zip.value.length != 5 )     {
     alert( "Please provide a zip in the format #####." );
document.myForm.Zip.focus() ;      return false;   }
   if( document.myForm.Country.value == "-1" )   {
     alert( "Please provide your country!" );      return false;   }
   return( true ); }
</script>
```

```
<script type="text/javascript">
function validateEmail()
{
var emailID = document.myForm.EMail.value;
atpos = emailID.indexOf("@");    dotpos =
emailID.lastIndexOf(".");    if (atpos < 1 || ( dotpos -
atpos < 2 ))   {
alert("Please enter correct email ID")
document.myForm.EMail.focus() ;
return false;
}
return( true );
</script>
```

16. Image Map

The image that is going to form the map is inserted into the page using the <img /> element as normal, except that it carries an extra attribute called usemap. The value of the usemap attribute is the value of the name attribute on the <map> element, which you are about to meet, preceded by a pound or a hash sign.

Example:

```
<html>
<head>
<title>Using JavaScript Image Map</title><script type="text/javascript">
function showTutorial(name){
 document.myform.stage.value = name
}
</script>
</head>
<body>
<form name="myform">
<input type="text" name="stage" size="20" />
</form>
<img src="/images/usemap.gif" alt="HTML Map" border="0" usemap="#tutorials"/>
<map name="tutorials">
<area shape="poly" coords="74,0,113,29,98,72,52,72,38,27"  href="/perl/index.htm" alt="Perl Tutorial"
target="_self" onMouseOver="showTutorial('perl')" onMouseOut="showTutorial('')"/>
<area shape="rect"
        coords="22,83,126,125"
        href="/html/index.htm" alt="HTML Tutorial"   target="_self"
        onMouseOver="showTutorial('html')"  onMouseOut="showTutorial('')"/>
<area shape="circle"  coords="73,168,32"
        href="/php/index.htm" alt="PHP Tutorial"  target="_self"
        onMouseOver="showTutorial('php')"  onMouseOut="showTutorial('')"/>
</map>
</body></html>
```

17. Browsers

It is important to understand the differences between different browsers in order to handle each in the way it is expected. So it is important to know which browser your web page is running in. To get information about the browser your webpage is currently running in, use the built-in navigator object.

```
<body>
<script type="text/javascript">
var userAgent   = navigator.userAgent;
var opera       = (userAgent.indexOf('Opera') != -1); var ie          = (userAgent.indexOf('MSIE') != -1); var gecko       =
(userAgent.indexOf('Gecko') != -1); var netscape    = (userAgent.indexOf('Mozilla') != -1); var version     =
navigator.appVersion;
if (opera){
  document.write("Opera based browser");    // Keep your opera specific URL here.
}else if (gecko){
document.write("Mozilla based browser");   // Keep your gecko specific URL here.
}else if (ie){
  document.write("IE based browser");    // Keep your IE specific URL here.
}else if (netscape){
  document.write("Netscape based browser");
}else{
  document.write("Unknown browser");
}
document.write("<br /> Browser version info : " + version );
</script>
</body>
```