Introduction

to

JavaScript

What is JavaScript?

- Cross-platform, Object-oriented scripting Language
- Invented by Brendan Eich in 1995 at Netscape Corporation (LiveScript), and became an ECMA standard in 1997
- Interpreted
- Weakly typed

History

- JavaScript was introduced as part of the Netscape 2.0 browser in 1995
- ECMA developed a standard language known as ECMAScript
- European Computer Manufacturers Association
- Also called as ECMAScript

Versions

Version	Official Name	Description
1	ECMAScript 1 (1997)	First Edition.
2	ECMAScript 2 (1998)	Editorial changes only.
3	ECMAScript 3 (1999)	Added Regular Expressions. Added try/catch.
4	ECMAScript 4	Never released.
5	ECMAScript 5 (2009)	Added "strict mode". Added JSON support. Added String.trim(). Added Array.isArray(). Added Array Iteration Methods.
5.1	ECMAScript 5.1 (2011)	Editorial changes.

Versions

Version	Official Name	Description
6	ECMAScript 2015	Added let and const. Added default parameter values. Added Array.find(). Added Array.findIndex().
7	ECMAScript 2016	Added exponential operator (**). Added Array.prototype.includes.
8	ECMAScript 2017	Added string padding. Added new Object properties. Added Async functions. Added Shared Memory.
9	ECMAScript 2018	Added rest / spread properties. Added Asynchronous iteration. Added Promise.finally(). Additions to RegExp.

Scope

- Desktop applications
  - Using HTML, CSS, JS and FS operations
  - e.g. NW.js and Electron
- Mobile applications
  - Using HTML, CSS, JS and additional platform specific APIs
- Server-side and Embedded applications
  - Using Node.js environment

Why Study JavaScript?

JavaScript is one of the 3 languages all web developers must learn:

1. HTML to define the content of web pages
2. CSS to specify the layout of web pages
3. JS to program the behavior of web pages

Applications

- JavaScript is used in million of web pages to
  - Improve design
  - Validate forms
  - Detect browsers
  - Create cookies, etc.

- Javascript is designed to add interactivity to HTML pages

Java vs. JavaScript

Java	JavaScript
Statically typed	Dynamically typed
Class based object model	Prototype based object model
Properties and methods cannot be added dynamically	Properties and methods can be added dynamically
Can automatically write to hard disk	Cannot automatically write to hard disk

HTML and JavaScript

- You can integrate javascript code into the HTML file in three ways

1. Integrating under <head> tag
2. Integrating under <body> tag
3. Importing the external JavaScript

HTML and JavaScript

1) Integrating script under the <head> tag

   – JavaScript in the HTML file will execute immediately while the web page loads into the web browser before anyone uses it

HTML and JavaScript

- Integrating script under the <head> tag
- Syntax :-

```
<html>
   <head>
          <script type="text/javascript" >
                -----------------
          </script>
     </head>
     <body>
   </body>
 <html>
```

HTML and JavaScript

- Including *type* attribute is a good practice but browsers like firefox, IE, etc. use javascript as their default script language, so if we don't specify type attribute it assumes that the scripting language is javascript

- However use of *type* attribute is specified as mandatory by W3C.

HTML and JavaScript

2) Integrating script under the <body> tag

   – this generates the content of the web page.

   – JavaScript code executes when the web page loads and so in the body section

```
<html>
   <head>
       <title>Hello JavaScript!!
       </title>
       <script>
           console.log("Hello");
       </script>
   </head>
   <body>
       <script>
           alert("JavaScript!!");
       </script>
   </body>
</html>
```

HTML and JavaScript

3) Importing the External JavaScript

- You can import an external JS file when you want to run the same JS file on several HTML files
- Save the external JS file with an extension **.js**
- The external JS file don't have a **<script>** tag

External JavaScript Advantages

- It separates HTML and code
- It makes HTML and JavaScript easier to read and maintain
- Cached JavaScript files can speed up page loads

HTML and JavaScript

```
<html>
       <head>
           <script src="first.js" >
                 -------------------
           </script>
       </head>
      <body>
                ------------------------
      </body>
 <html>
```

JavaScript Output

- Writing into an HTML element, using innerHTML
- Writing into the HTML o/p using document.write()
- Writing into an alert box, using window.alert()
- Writing into the browser console, using console.log()

```
<html>
  <head>
     <title>Hello JavaScript!! </title>
     <script>
        document.write("Hello World!");
     </script>
  </head>
  <body>
     <h1>Learn JavaScript</h1>
  </body>
</html>
```

Variable Declarations

var: Declares a variable, optionally initializing it to a value

let: Declares a block-scoped, local variable, optionally initializing it to a value

const: Declares a block-scoped, read-only named constant

Variable Declarations

- Basic syntax of variable declaration
  - Syntax:-   var variablename;
- Naming conventions
  - Variable name can start with a alphabet or underscore. (Rest characters can be number, alphabets, dollar symbol, underscore)
  - Do not use any special character other than dollar sign ($), underscore (_)
  - Variable names are **case-sensitive.**
  - Cannot contain blank spaces.
  - Cannot contain any reserved word

Reserved Words

abstract
boolean
break
byte
case
catch
char
class
const
continue
debugger
default
delete
do
double
else
enum
export
extends
false
final
finally
float
for
function
goto
if
implements
import
in
instanceof
int
interface
long
native
new
null
package
private
protected
public
return
short
static
super
switch
synchronized
this
throw
throws
transient
true
try
typeof
var
void
volatile
while
with

Variables (var)

```
var a;
console.log('The value of a is ' + a);


console.log('The value of b is ' + b);


var b;

console.log('The value of c is ' + c);
```

Variables (var)

```
var a;
console.log('The value of a is ' + a);
// The value of a is undefined

console.log('The value of b is ' + b);
// The value of b is undefined

var b;

console.log('The value of c is ' + c);
//  Uncaught  ReferenceError:  c  is  not
defined
```

Use of **undefined**

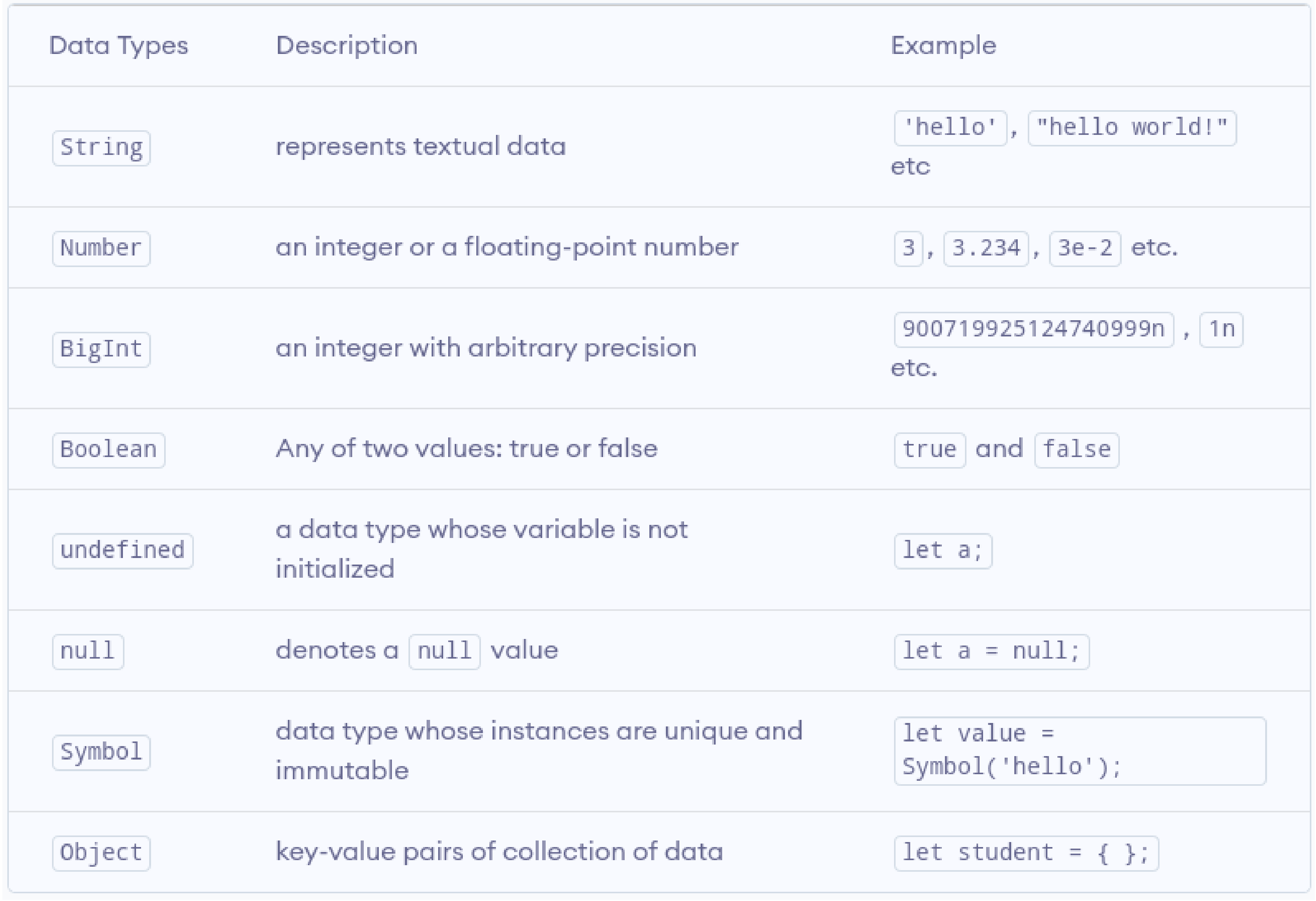
```
var input;

if (input === undefined)
{
   doThis();
} else
{
   doThat();
}
```

Variables

```
<script type="text/javascript">
        var bookname="web tech applications";
        var bookprice=390;
             console.log("bookname is:
    ",bookname);
        console.log ("bookprice is: ", bookprice);
</script>
```

Datatypes



Data Types	Description	Example
`String`	represents textual data	`'hello'`, `"hello world!"` etc
`Number`	an integer or a floating-point number	`3`, `3.234`, `3e-2` etc.
`BigInt`	an integer with arbitrary precision	`900719925124740999n` , `1n` etc.
`Boolean`	Any of two values: true or false	`true` and `false`
`undefined`	a data type whose variable is not initialized	`let a;`
`null`	denotes a `null` value	`let a = null;`
`Symbol`	data type whose instances are unique and immutable	`let value = Symbol('hello');`
`Object`	key-value pairs of collection of data	`let student = { };`

String Datatype

- string values enclosed in single or double quotes.
- Examples:

```
var car_model = "Nexon";
var car_brand = 'Tata';
```

Number Datatype

- Integer literals can be represented in
  - decimal
  - hexadecimal and
  - octal form
- Floating literal consists of either a
  - number containing a decimal point or
  - an integer followed by an exponent

Boolean Datatype

- consists of logical values true and false
- JavaScript automatically converts logical values true and false to 1 and 0 when they are used in numeric expressions

Dynamic Types

```
var a = 10;

a = "hello";
```

Data Type Conversion

```
'hello' + 5   // 'hello5'

'37' – 7  // 30

'37' + 7  // '377'
```

Falsy values

The following values evaluate to false:

- false
- undefined
- null
- 0
- NaN
- the empty string ("")

All other values—including all objects—evaluate to true when passed to a conditional statement

Arithmetic Operators

Operator	Description
+	Adds two numbers together
-	Subtracts one number from another or changes a number to its negative
*	Multiplies two numbers together
/	Divides one number by another
%	Produces the remainder after dividing one number by another
**	Exponentiation (ES2016)
++	Increment [X++ is equivalent of X = X + 1;]
--	Decrement [X-- is equivalent of X = X - 1;]

Comparison Operators

Operator	Description
==	**Equal** operator. *value1 == value2* Auto type conversion is done while testing
===	**Strict Equal** operator. *value1 === value2* No type conversion is done
!=	**Not Equal** operator. *value1 != value2* *Tests whether value1* is different from *value2*.
<	**Less Than** operator. *value1 < value2* *Tests whether value1* is less than *value2*.
>	**Greater Than** operator. *value1 > value2* *Tests whether value1* is greater than *value2*.
<=	**Less Than or Equal To** operator. *value1 <= value2* *Tests whether value1* is less than or equal to *value2*.
>=	**Greater Than or Equal To** operator. *Tests whether value1* is greater than or equal to *value2*.

Strict Equality using ===

0==false

// **true**, because false is equivalent of 0

0===false

// **false**, both operands are of different type

2=="2"

// **true**, auto type coercion, string converted to number

2==="2"

// **false**, since both operands are not of same type

Logical (Relational) Operators

Operator	Description
&&	**And** operator. If both the operands are non-zero, then the condition becomes true.
||	**Or** operator. If any of the two operands are non-zero, then the condition becomes true.
!	**Not** operator. Reverses the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false.

Bitwise Operators

Operator	Description
&	**Bitwise AND** It performs a Boolean AND operation on each bit of its integer arguments.
|	**Bitwise OR** It performs a Boolean OR operation on each bit of its integer arguments.
^	**Bitwise XOR** It performs a Boolean exclusive OR operation on each bit of its integer arguments. Exclusive OR means that either operand one is true or operand two is true, but not both.
~	**Bitwise Not** It is a unary operator and operates by reversing all the bits in the operand.

Bitwise Operators (Cont..)

Operator	Description
<<	**Left Shift** It moves all the bits in its first operand to the left by the number of places specified in the second operand. New bits are filled with zeros. Shifting a value left by one position is equivalent to multiplying it by 2, shifting two positions is equivalent to multiplying by 4, and so on.
>>	**Right Shift** Binary Right Shift Operator. The left operand's value is moved right by the number of bits specified by the right operand.
>>>	**Right Shift with Zero** This operator is just like the >> operator, except that the bits shifted in on the left are always zero.

Bitwise Operators (Examples)

Operator	Description	Example	Same as	Result	Dec.
&	AND	5 & 1	0101 & 0001	0001	1
|	OR	5 | 1	0101 | 0001	0101	5
~	NOT	~ 5	~0101	1010	10
^	XOR	5 ^ 1	0101 ^ 0001	0100	4
<<	Zero fill left shift	5 << 1	0101 << 1	1010	10
>>	Signed right shift	5 >> 1	0101 >> 1	0010	2
>>>	Zero fill right shift	5 >>> 1	0101 >>> 1	0010	2

Assignment Operators

Operator	Description
=	Assigns right operand value to left operand.
+=	X += Y (The equivalent of X = X + Y;)
-=	X -= Y (The equivalent of X = X - Y;)
***=**	X *= Y (The equivalent of X = X * Y;)
/=	X /= Y (The equivalent of X = X / Y;)
%=	X %= Y (The equivalent of X = X % Y;)
****=**	X **= Y (The equivalent of X = X ** Y;)

Note − Same logic applies to Bitwise operators so they will become like <<=, >>=, >>>=, &=, \|= and ^=.

Ternary Operators

- JavaScript includes special operator called ternary operator **?:** that assigns a value to a variable based on some condition. This is like short form of if-else condition.

- **Syntax:** <condition> **?** <value1> **:** <value2>;

- Ternary operator starts with conditional expression followed by ? operator. Second part (after ? and before : operator) will be executed if condition turns out to be true. If condition becomes false then third part (after :) will be executed.

Type Operators

Operator	**Description**
typeof	Returns the type of a variable
instanceof	Returns true if an object is an instance of an object type

If condition

- **Syntax:-**

  if ***(conditional expression)***

  { *do this…* }

If – else condition

- **Syntax:-**

```
if (conditional expression)
        {   do this...   }
   else
        {   do this...   }
```

Nested if condition

- **Syntax:-**

```
if (conditional expression)    {
            if (conditional expression)
                {    do this…   }
           else
                {    do this…   }
        }
     else {
            if (conditional expression)
                {    do this…   }
           else
                {    do this…   }
    }
```

If..else if condition

- **Syntax:-**

```
if (conditional expression1)
    {   do this…   }
else if (conditional expression2)
    {   do this…   }
else if (conditional expression3)
    {   do this…   }
…
else
    {   do this…   }
```

The Switch Statement

- **Syntax:-**

```
switch (expression)
{
    case "value1":    do this...
                      break
    case "value2":    do this...
                      break
    ...
    default:    do this...
}
```

While Loop

while statement:-

```
    while (conditional expression)
     {
            do this…
     }
```

Do….While Loop

do …. while statement

```
do
{
    do this…
}while (conditional  expression)
```

For Loop

For statement:-

```
    for (exp1;exp2;exp3)
     {
           do this…
     }
```

exp1: ***initial expression***

exp2: ***conditional expression***

exp3: ***incremental expression***

Variable Scope

- Global Scope:
- Variable declared outside a function have Global Scope
- Local Scope:
- Variable declared inside a function have Local Scope
- Block Scope:
- JavaScript before ECMAScript 2015 did not have block scope
- Variables declared inside a block cannot be accessed from outside the block
- **let** and **const** are two keywords that provide Block Scope in JavaScript.

Variable Scope (var)

```
if(true) {
    var x = 5;
}
console.log(x);
// Output?
// 5
```

Variable Scope (var)

```
function newFunction() {
    var msg = "hello";
}
console.log(msg);
// Output?
//Uncaught ReferenceError: hello is
not defined
```

Variable Scope (let)

```
if(true) {
    let y = 5;
}
console.log(y);
// Output?
// Uncaught ReferenceError: y is not
defined
```

Variable Scope

```
function newFunction() {
    msg = "hello";
}
newFunction();
console.log(msg);
// "hello"
```

Note: variables declared inside a function without var keyword also become global variables

Problem with "var"

```
var greeter = "hi";
var times = 4;
if (times > 3) {
    var greeter = "hello";
}
console.log(greeter);
// Output?
// hello
```

Another example of "let"

```
let greeting = "Hi";
if (true) {
    let greeting = "Hello";
    console.log(greeting);
}
console.log(greeting);
// "Hello"
// "Hi"
```

Variable Hoisting

```
console.log (greeter); // undefined
var greeter = "hello";
```

```
var greeter;
console.log (greeter); // undefined
var greeter = "hello";
```

Variable is hoisted to the top of its scope and initialized with value "undefined"

var vs. let

var	let
Not block-scoped	Block scoped
Can be re-declared within same scope	Cannot be re-declared within same scope
When hoisted initialized to "undefined"	When hoisted variables are not initialized

const keyword

- You can create a read-only, named constant with the const keyword

```
const PI = 3.14;
```

- scope can be either global or local to the block in which it is declared

- Global constants do not become properties of the window object, unlike var variables

const keyword

- It defines a constant reference to a value
- It does not mean the value it holds is immutable—just that the variable identifier cannot be reassigned

```
const arr = [1, 2, 3]

arr = "hello"

//Uncaught TypeError: invalid assignment
//to const 'arr'

arr[1] = 10  // [1, 10, 3]
```

const keyword

- It cannot be reassigned a new value
- It cannot be re-declared
- It must be initialized
- Block scoped
- You cannot declare a constant with the same name as a function or variable in the same scope

const keyword

the properties of objects assigned to constants are not protected

```
const MY_OBJECT = {'key': 'value'};
MY_OBJECT.key = 'otherValue';
```

const keyword

Contents of an array are not protected

```
const MY_ARRAY = ['HTML','CSS'];
MY_ARRAY.push('JAVASCRIPT');
console.log(MY_ARRAY );
//logs ['HTML','CSS','JAVASCRIPT'];
```

References

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/