Core Java Fundamentals & Java Scripting

knowledge hut

Learning Objectives

- Basic Java commands and APIs using industry tools
- Foundational data organization and manipulation
- Code control structures, such as loops and if/else statements
- How to structure code using methods, parameters, and returns

Introduction to JavaScript

Fundamentals of JavaScript

- JavaScript is a very powerful **client-side scripting language**. JavaScript is used mainly for enhancing the interaction of a user with the webpage.

- JavaScript is a programming language used primarily by Web browsers to create a dynamic and interactive experience for the user. Most of the functions and applications that make the Internet indispensable to modern life are coded in some form of JavaScript.

- JavaScript is also being used widely in game development and Mobile application development.

- The language was initially called LiveScript and was later renamed JavaScript. There are many programmers who think that JavaScript and Java are the same. In fact, JavaScript and Java are very much unrelated. Java is a very complex programming language whereas JavaScript is only a scripting language. The syntax of JavaScript is mostly influenced by the programming language C.

Why Do We Need the JavaScript ?

- JavaScript is the programming language, There are three languages that every Web development programmer must know.
- HTML (Hypertext Markup Language), CSS (Cascading Style Sheets), and JavaScript.
- HTML tells a browser what content to show on your web pages.
- CSS tells the browser how to lay the content out and format it to look great.
- JavaScript controls the behavior of the page. All the popups, cool transitions, scrolling animations, and other attractive aspects of modern websites are created using JavaScript.

Reasons to Include the JavaScript

- **Client-side programming:** It's one of the few languages that isn't run on the server that hosts the web page. If you're a web developer, the time taken to load a page on a viewer's browser must be as low as possible. The higher the load time, the lower the user experience. JavaScript animations use the viewer's device processor to run themselves. This reduces the strain on the web server by a huge amount and cuts the page load time.

- **JavaScript** isn't the only programming language for the web. Others include Python, C++, and a few more. However, it's the most popular because it's easy to learn. The syntax JavaScript uses is very similar to English. Developers don't have to spend weeks figuring out what each snippet of code does when building a new site. In the fast-paced web development industry, every second counts. When a developer uses JavaScript, your site will be open to the public that much faster.

Reasons to Include the JavaScript

- **Compatible with other languages:** Web servers use different languages to run well. Python, PHP, Ruby, Rails, ASP.NET, and Java are a few examples. Regardless of the language used on the server, your viewers will always have a rich, responsive experience because JavaScript works well with all of these languages.

- **Compiler-free programming:** Traditional coding uses a compiler. A browser uses a compiler to translate raw code into readable syntax. This takes time and increases the load time of the page. It also increases the amount of time your site will be in development. On the other hand, browsers can natively read JavaScript. They can translate it the same way they do HTML. A compiler isn’t needed to make it readable.

Supported Browsers

- JavaScript was invented by Brendan Eich in 1995, and became an ECMA standard in 1997.
- ECMAScript is the official name of the language.
- From 2015 ECMAScript is named by year (ECMAScript 2015).
- ECMAScript 3 is fully supported in all browsers.
- ECMAScript 5 is fully supported in all **modern** browsers.

Chrome 23	IE10 / Edge	Firefox 21	Safari 6	Opera 15
Sep 2012	Sep 2012	Apr 2013	Jul 2012	Jul 2013

JS Syntax

JavaScript syntax is the set of rules, how JavaScript programs are constructed:

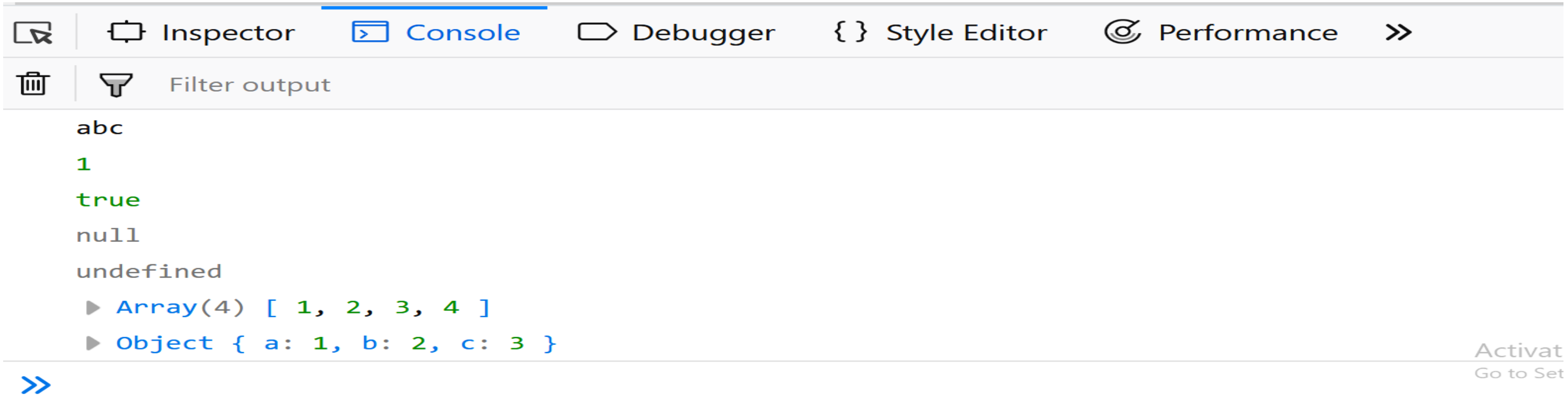
```
<html>
              <body>
                            <script language = javascript type = text/javascript>
                                          document.write(Hello World!);
                            </script>
              </body>
</html>
```

Using JavaScript Console

- We can open a console in web browser by using: Ctrl + Shift + J for windows and Command + Option + K for Mac. The console object provides us with several different methods, like :

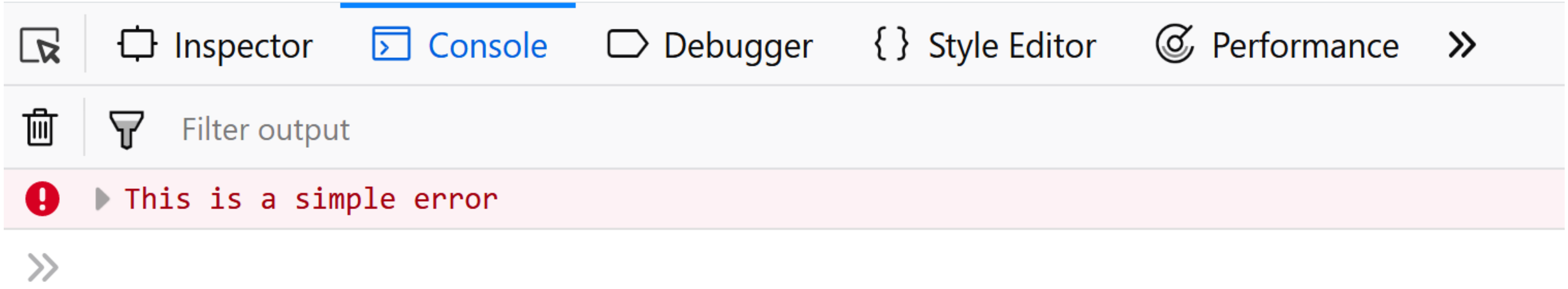
- log()
- error()
- warn()
- clear()
- time() and timeEnd()
- table()
- count()
- group() and groupEnd()

```
// console.log() method mainly used to log(print)
//the output to the console.
console.log('abc');
console.log(1);
console.log(true);
console.log(null);
console.log(undefined);
console.log([1, 2, 3, 4]); // array inside log
console.log({a:1, b:2, c:3}); // object inside log
```



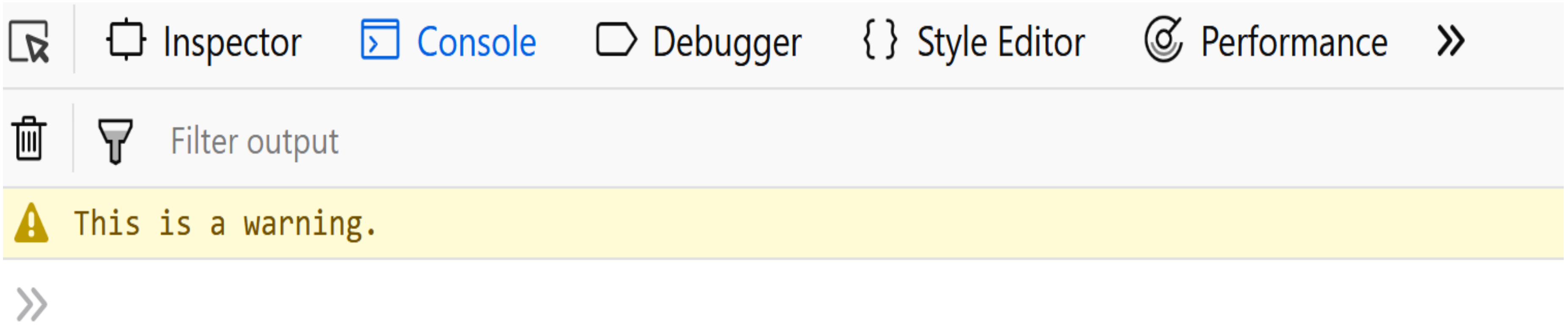
```
abc
1
true
null
undefined
▶ Array(4) [ 1, 2, 3, 4 ]
▶ Object { a: 1, b: 2, c: 3 }
```

```
// console.error() method is used to log error message to the console

console.error('This is a simple error');
```



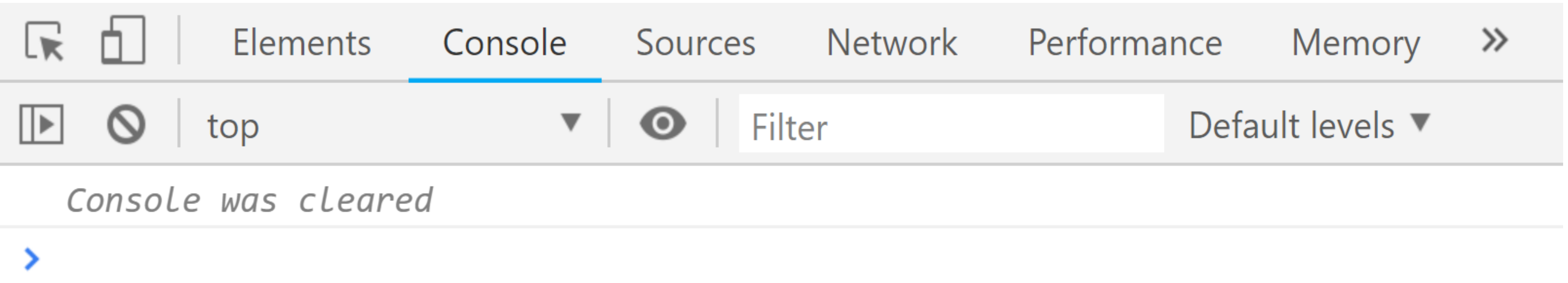
```
This is a simple error
```

```
// console.warn() method is used to log warning message to the console

console.warn('This is a warning.');
```



```
This is a warning.
```

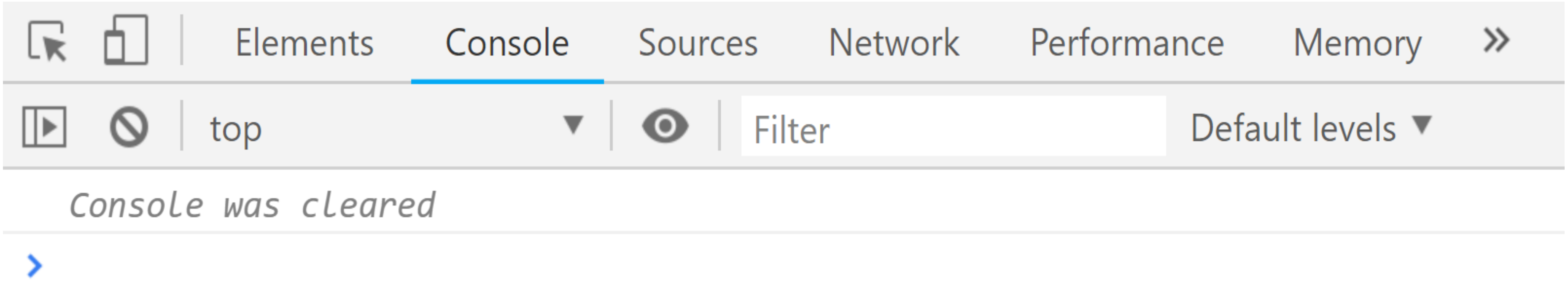
```
// console.clear() method is used to clear the console.

console.clear();
```



```
Console was cleared
>
```

// console.table() method allows us to generate a table inside a console

console.table({'a':1, 'b':2});



```
Console was cleared
>
```

JS Comments

JavaScript syntax is the set of rules, how JavaScript programs are constructed:

- JavaScript comments can be used to explain JavaScript code, and to make it more readable.
- JavaScript comments can also be used to prevent execution, when testing alternative code.
- Single line comments start with //.
- Multi-line comments start with /* and end with */.

JS Comments

- Single line Comment.

```
var x = 5;      // Declare x, give it the value of 5
```

- Multiple line Comments.

```
/*
The code below will change
the heading with id = myH
and the paragraph with id = myP
in my web page:
*/
document.getElementById(myH).innerHTML = My First Page;
```

Variables

- JavaScript variables are containers for storing data values.
- In the following example, x, y, and z, are variables:

```
var x = 5;
var y = 6;
var z = x + y;
```

Data Types

- JavaScript variables can hold many **data types**: numbers, strings, objects and more:

```
var length = 16;                                    //Number
var lastName = Johnson;                             // String
var x = {firstName:John, lastName:Doe};            // Object
```

Debugging Using Chrome Developer Tools

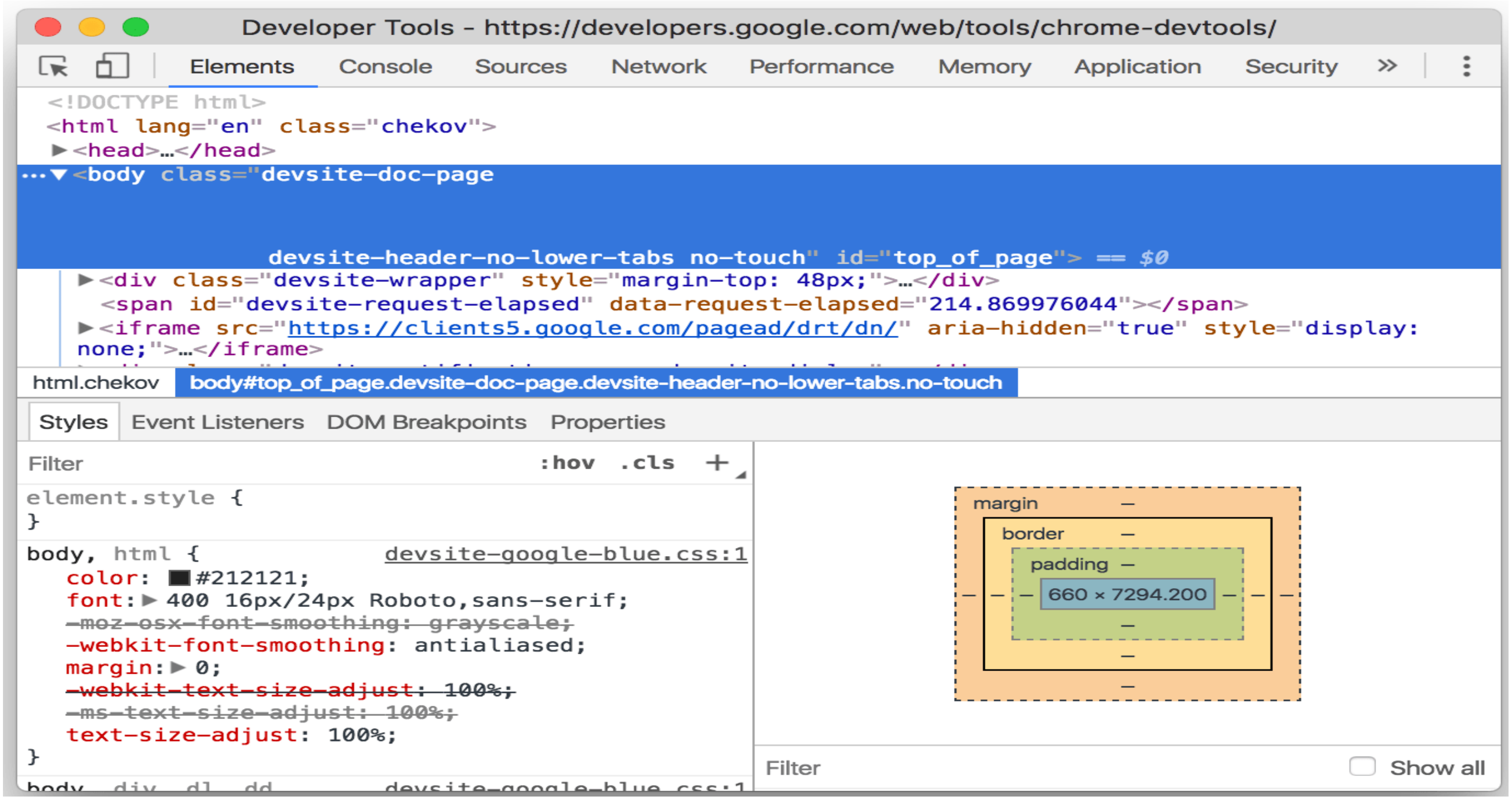
- Chrome DevTools is a set of web developer tools built directly into the Google Chrome browser. DevTools can help you edit pages on-the-fly and diagnose problems quickly, which ultimately helps you build better websites, faster.

- When you want to work with the DOM or CSS, right-click an element on the page and select **Inspect** to jump into the **Elements** panel. Or press Command+Option+C (Mac) or Control+Shift+C (Windows, Linux, Chrome OS).

- When you want to see logged messages or run JavaScript, press Command+Option+J (Mac) or Control+Shift+J (Windows, Linux, Chrome OS) to jump straight into the **Console** panel.

Elements Panel

View and change the DOM and CSS.

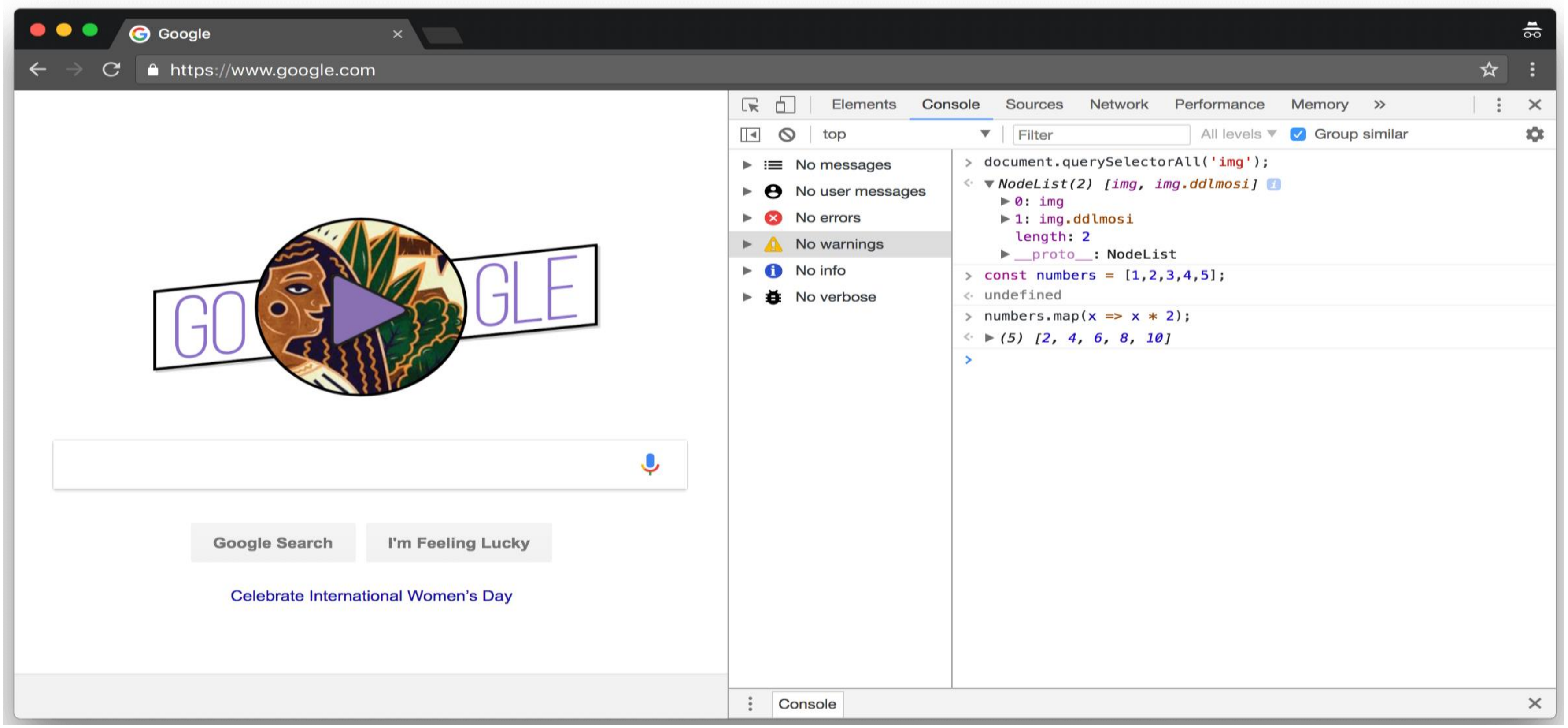


```
<!DOCTYPE html>
<html lang="en" class="chekov">
 ▶<head>…</head>
…▼<body class="devsite-doc-page

         devsite-header-no-lower-tabs no-touch" id="top_of_page"> == $0
   ▶<div class="devsite-wrapper" style="margin-top: 48px;">…</div>
    <span id="devsite-request-elapsed" data-request-elapsed="214.869976044"></span>
   ▶<iframe src="https://clients5.google.com/pagead/drt/dn/" aria-hidden="true" style="display:
   none;">…</iframe>
```

```
element.style {
}
body, html {                        devsite-google-blue.css:1
   color: ■#212121;
   font:▶ 400 16px/24px Roboto,sans-serif;
   ~~-moz-osx-font-smoothing: grayscale;~~
   -webkit-font-smoothing: antialiased;
   margin:▶ 0;
   ~~-webkit-text-size-adjust: 100%;~~
   ~~-ms-text-size-adjust: 100%;~~
   text-size-adjust: 100%;
}
```

Console Panel

View messages and run JavaScript from the Console



```
> document.querySelectorAll('img');
< ▼NodeList(2) [img, img.ddlmosi]
   ▶0: img
   ▶1: img.ddlmosi
    length: 2
   ▶__proto__: NodeList
> const numbers = [1,2,3,4,5];
< undefined
> numbers.map(x => x * 2);
< ▶(5) [2, 4, 6, 8, 10]
>
```

Expressions

- An *expression* is any valid unit of code that resolves to a value.
- There are two types of expressions: with side effects (for example: those that assign value to a variable) and those that in some sense evaluate and therefore resolve to a value.
- The expression x = 7 is an example of the first type.
- This expression uses the = *operator* to assign the value seven to the variable x. The expression itself evaluates to seven.
- The code 3 + 4 is an example of the second expression type.
- This expression uses the + operator to add three and four together without assigning the result, seven, to a variable.

Operators

JavaScript has the following types of operators. This section describes the operators and contains information about operator precedence.

1. Assignment operators
2. Comparison operators
3. Arithmetic operators
4. Bitwise operators
5. Logical operators
6. String operators
7. Conditional (ternary) operator
8. Comma operator
9. Unary operators
10. Relational operators

Assignment Operators

JavaScript includes assignment operators to assign values to variables with less key strokes.

Assignment operators	Description
=	Assigns right operand value to left operand.
+=	Sums up left and right operand values and assign the result to the left operand.
-=	Subtract right operand value from left operand value and assign the result to the left operand.
*=	Multiply left and right operand values and assign the result to the left operand.
/=	Divide left operand value by right operand value and assign the result to the left operand.
%=	Get the modulus of left operand divide by right operand and assign resulted modulus to the left operand.

Assignment Operators Example

```
var x = 5, y = 10, z = 15;

x = y; //x would be 10

x += 1; //x would be 6

x -= 1; //x would be 4
```

Comparison Operators

JavaScript language includes operators that compare two operands and return Boolean value true or false.

Operators	Description
==	Compares the equality of two operands without considering type.
===	Compares equality of two operands with type.
!=	Compares inequality of two operands.
>	Checks whether left side value is greater than right side value. If yes then returns true otherwise false.
<	Checks whether left operand is less than right operand. If yes then returns true otherwise false.
>=	Checks whether left operand is greater than or equal to right operand. If yes then returns true otherwise false.
<=	Checks whether left operand is less than or equal to right operand. If yes then returns true otherwise false.

Comparison Operators Example

```
var a = 5, b = 10, c = 5;

var x = a;

a == c; // returns true

a === c; // returns false

a == x; // returns true
```

Arithmetic Operators

Arithmetic operators are used to perform mathematical operations between numeric operands

Operator	Description
+	Adds two numeric operands.
-	Subtract right operand from left operand
*	Multiply two numeric operands.
/	Divide left operand by right operand.
%	Modulus operator. Returns remainder of two operands.
++	Increment operator. Increase operand value by one.
--	Decrement operator. Decrease value by one.

Arithmetic Operators Example

```
var x = 5, y = 10, z = 15;

x + y; //returns 15

y - x; //returns 5

x * y; //returns 50
```

Bitwise Operators

In JavaScript, a number is stored as a 64-bit floating point number but the bit-wise operation is performed on a 32-bit binary number

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shifts left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shifts right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off
>>>	Zero fill right shift	Shifts right by pushing zeros in from the left, and let the rightmost bits fall off

Bitwise Operators Example

```
var a = 6;
var b = 1;

// AND Operation
document.write(A & B =  + (a & b) + '<br>');            A & B = 0

// OR operation
document.write(A | B =  + (a | b) + '<br>');            A | B = 7

// NOT operation
document.write(~A =  + (~a) + '<br>');                  ~A = -7

// Sign Propagating Right Shift
document.write(A >> B =  + (a >> b) + '<br>');          A >> B = 3

// Zero Fill Right Shift
document.write(A >>> B =  + (a >>> b) + '<br>');        A >>> B = 3

// Left Shift
document.write(A << B =  + (a << b) + '<br>');          A << B = 12
```

Logical Operators

Logical operators are used to combine two or more conditions.
JavaScript includes following logical operators.

Operator	Description
&&	&& is known as AND operator. It checks whether two operands are non-zero (0, false, undefinednull or are considered as zero), if yes then returns 1 otherwise 0.
||	|| is known as OR operator. It checks whether any one of the two operands is non-zero (0, false, undefined, null or is considered as zero).
!	! is known as NOT operator. It reverses the boolean result of the operand (or condition)

Logical Operators Example

```
var a = 5, b = 10;

(a != b) && (a < b); // returns true

(a > b) || (a == b); // returns false

(a < b) || (a == b); // returns true
```

String Operators

String operators concatenate values.

Operator	Description
string1 + string2	Concatenates **string1** and **string2** so the beginning character of **string2** follows the ending character of **string1**.
string1 += string2	Concatenates **string1** and **string2**, and assigns the result to **string1**.

String Operator Example:

```
var a = mohan;

var b = jayabalan;

var result = a + b;        //mohanjayabalan will be stored in result
```

Ternary Operator

The **conditional (ternary) operator** is the only JavaScript operator that takes three operands: a condition followed by a question mark (?), then an expression to execute if the condition is truthy followed by a colon (:), and finally the expression to execute if the condition is falsy.

voteable = (age < 18) ? Too young : Old enough;

Conditions & Loops

Conditions

1. if statement
2. else statement
3. else if statement

Loops

1. while
2. Do....while
3. for

If Condition

- Used to check for a condition.
- Expression is solved first and it returns true or false accordingly.

```
<html>
   <body>
      <script type=text/javascript>
            var student_age = 22;
            if( student_age > 10 ){
               document.write(<b>This is my first program
</b>);
            }
      </script>
    </body>
</html>
```

If else Condition

- Used when we want any logic if an if statement is false.
- When condition does not satisfies then else statement are executed.

```
<html>
  <body>
    <script type=text/javascript>
        var student_age = 22;
        if( student_age > 10 ){
          document.write(<b> Age is greater than
10 </b>);
        }
                else{
                  document.write(<b>Age is less than
10 </b>);
                }
    </script>
   </body>
</html>
```

If…elseif Condition

- Used when we have many conditions.

```
<html>
  <body>
    <script type=text/javascript>
        var student_age = 22;
        if( student_age > 10 ){
           document.write(<b> Age is greater than 10 </b>);
        }
                else if( student_age < 10 ){
                                        document.write(<b> Age is less than 10  </b>);
                }
                else {
                                        document.write(<b> Other Age </b>);
                }
    </script>
   </body>
</html>
```

Switch Statement

- The switch statement evaluates an expression, matching the expression's value to a case clause, and executes statements associated with that case, as well as statements in cases that follow the matching case.

```
var x = 0;
switch (x) {
  case 0:
    text = Off;
    break;
  case 1:
    text = On;
    break;
  default:
    text = No value found;
}
```

For statement

- A for loop repeats until a specified condition evaluates to false. The JavaScript for loop is similar to the Java and C for loop
- Syntax-

```
for ([initialExpression]; [condition]; [incrementExpression])

        statement
```

- Example-

```
function print() {
    for (let i = 0; i <10; i++){
        document.write(i);
    }
}
```

Do…while statement

- Do…while statement repeats until a specified condition evaluates to false

```
do
statement
while (condition);
```

- Example-

```
let i = 0;
do {
   i += 1;
   document.write(i);
  }
while (i < 5);
```

while statement

- A while statement executes its statements as long as a specified condition evaluates to true. A while statement looks as follows:

```
while (condition)

   statement
```

- Example-

```
function print() {
    let i= 0;
        while (i <10){
                document.write(i);
                i = i +1;
        }
}
```

Break Statement

- Use the break statement to terminate a loop, switch, or in conjunction with a labeled statement.

```
for (let i = 0; i < a.length; i++) {
  if (a[i] === theValue) {
    break;
  }
}
```

Continue Statement

- The continue statement can be used to restart a while, do-while, for, or label statement.

```
let i = 0;
let n = 0;
while (i < 5) {
    i++;
    if (i === 3) {
        continue;
    }
    n += i;
    document.write(n);
}
```

For … in Statement

- The for...in statement iterates a specified variable over all the enumerable properties of an object. For each distinct property, JavaScript executes the specified statements. A for...in statement looks as follows:

```
var person = { fname:John,  lname:Doe,  age:25};

var text = ;
var x;
for (x in person) {
  text += person[x] +  ;
}
```

For … of Statement

- The for…of statement creates a loop Iterating over iterable objects (including Array, Map, Set, arguments object and so on), invoking a custom iteration hook with statements to be executed for the value of each distinct property.

```
var cars = ['BMW', 'Volvo', 'Mini'];
var x;

for (x of cars) {
  document.write(x + <br >);
}
```

Functions

Functions

What is JS Function:

- A JavaScript function is a block of code designed to perform a particular task.
- A JavaScript function is executed when something invokes it (calls it).

Syntax :

- A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses ().
- Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

Functions Parameters

- Function parameters are the names listed in the function definition.
- Function arguments are the real values passed to (and received by) the function.

```
myFunction(10,10);


function myFunction(x, y) {
  if (x === y) {
    document.write(Both are same);
  }
}
```

Functions Scope

- Scope determines the **accessibility (visibility)** of variables.
- In JavaScript there are **two** types of scope:

1. Local scope
2. Global scope

1.1 Variables declared within a JavaScript function, become **LOCAL** to the function.
1.2 Local variables have **Function scope**: They can only be accessed from within the function.

2.1 A variable declared outside a function, becomes **GLOBAL**.
2.2 A global variable has **global scope**: All scripts and functions on a web page can access it.

Hoisting

- Hoisting is JavaScript's default behavior of moving declarations to the top.
- In JavaScript, a variable can be declared after it has been used.
- In other words; a variable can be used before it has been declared.

```
x = 5; // Assign 5 to x

elem = document.getElementById(demo); // Find an element
elem.innerHTML = x;                     // Display x in the element

var x; // Declare x
```

Anonymous Function

```
var x = function (a, b) {return a * b};
var z = x(4, 3);
```

- The function above is actually an **anonymous function** (a function without a name).
- Functions stored in variables do not need function names. They are always invoked (called) using the variable name.
- The function above ends with a semicolon because it is a part of an executable statement.

Function Invocation

- The code inside a function is not executed when the function is **defined**.
- The code inside a function is executed when the function is **invoked**.
- It is common to use the term **call a function** instead of **invoke a function**

```
function myFunction(a, b) {
  return a * b;
}
myFunction(10, 2);          // Will return 20
```

Call, apply and bind

- The **call()** method calls a function with a given this value and arguments provided individually.
- call() and apply() serve the **exact same purpose.**
- The ***only difference between how they work is that*** call() expects all parameters to be passed in individually, whereas apply() expects an array of all of our parameters.
- The **bind()** method creates a new function that, when called, has its this keyword set to the provided value.

Callbacks

- Callbacks are a great way to handle something after something else has been completed.
- By something here we mean a function execution.
- If we want to execute a function right after the return of some other function, then callbacks can be used.

```
// add() function is called with arguments a, b
// and callback, callback will be executed just
// after ending of add() function
function add(a, b , callback){
		document.write(`The sum of ${a} and ${b} is ${a+b}.` +<br>);
callback();
}
// add() function is called with arguments given below
add(5,6,function disp(){
		document.write('This must be printed after addition.');
});
```

Arrays

Array

- JavaScript arrays are used to store multiple values in a single variable.
- An array is a special variable, which can hold more than one value at a time.
- If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
var car1 = Saab;
var car2 = Volvo;
var car3 = BMW;
```

What if you had not 3 cars, but 300? **The solution is an array!**

```
var cars = [Saab, Volvo, BMW];
```

CRUD Operations on Array:

Creating an Array

- Using an array literal is the easiest way to create a JavaScript Array.
- Syntax:
    - var *array_name* = [*item1*, *item2*, ...];
- Example:

```
var cars = [Saab, Volvo, BMW];
```

Read an Array

- You access an array element by referring to the **index number**.
- This statement accesses the value of the first element in cars:

- Example:

```
var cars = [Saab, Volvo, BMW];

     var name = cars[0];
```

Update an Array

- You can update an array element by referring to the **index number**.
- This statement updates the value of the first element in cars:
- Example:

```
var cars = [Saab, Volvo, BMW];

          cars[0] = Opel;
```

Delete Elements from Array

1. Using *pop() - pop()* method will remove one element from end.

2. Using shift() - shift() command will remove the first element of the array and the *unshift()* command will add an element to the beginning of the array.

3. Using splice() - The inputs to the *splice()* function are the index point to start at and the number of elements to remove.

Delete Elements from Array

1. **Using *pop()* -**

```
list = [bar, baz, foo, qux]

list.pop()

Output: [bar, baz, foo]
```

Delete Elements from Array

2. Using shift() -

```
list = [bar, baz, foo, qux]

list.shift()

Output: [baz, foo, qux]
```

Delete Elements from Array

3. Using slice() -

```
list = [bar, baz, foo, qux]

list.splice(0, 2)

Output: [foo, qux]
```

Most Commonly used Array Methods:

1. **push(); -** Helps in adding elements to the end of an array.
2. **pop(); -** Helps in removing elements at the end of the array, and returns the element.
3. **shift(); -** Helps in removing elements from the beginning of the array, and returns the element.
4. **unshift(); -** Helps in adding elements to the beginning an array, and returns the length of the array.
5. **concat(); -** This helps to concatenate(join/link) two arrays together.

Most Commonly used Array Methods:

6. **join(); -** This joins all the elements in an array to create a string, and does not affect or modify the array.
7. **reverse(); -** This basically takes in an array, and reverses it.
8. **sort(); -** This basically helps in sorting arrays to order, however causing permanent change to the array.
9. **slice(); -** This basically targets an array and returns the targeted array,
   and the original array remains unchanged.
10. **splice(); -** Just like the slice(); method, this also targets an array and returns the targeted array,
    moreover it causes a permanent change in the array.

Strings & Dates

Strings

- JavaScript strings are used for storing and manipulating text.
- A JavaScript string is zero or more characters written inside quotes.
- You can use single or double quotes:

```
var x = "John Doe";

var x = ’John Doe’;
```

Most Commonly used String Methods:

1. The length property returns the length of a string.
2. The indexOf() method returns the index of (the position of) the first occurrence of a specified text in a string.
3. The lastIndexOf() method returns the index of the **last** occurrence of a specified text in a string.

Both indexOf(), and lastIndexOf() return -1 if the text is not found.

4. The search() method searches a string for a specified value and returns the position of the match .
5. slice() extracts a part of a string and returns the extracted part in a new string.

Dates

- By default, JavaScript will use the browser's time zone and display a date as a full text string
- Date objects are created with the new Date() constructor.
- There are **4 ways** to create a new date object:

new Date()
new Date(*year, month, day, hours, minutes, seconds, milliseconds*)
new Date(*milliseconds*)
new Date(*date string*)

Most Commonly used Dates Methods:

Method	Description
getFullYear()	Get the year as a four digit number (yyyy)
getMonth()	Get the month as a number (0-11)
getDate()	Get the day as a number (1-31)
getHours()	Get the hour (0-23)
getMinutes()	Get the minute (0-59)
getSeconds()	Get the second (0-59)
getMilliseconds()	Get the millisecond (0-999)
getTime()	Get the time (milliseconds since January 1, 1970)
getDay()	Get the weekday as a number (0-6)
Date.now()	Get the time. ECMAScript 5.

Objects

Objects

- A javaScript object is an entity having state and behavior (properties and method).
- JavaScript is an object-based language. Everything is an object in JavaScript.
- JavaScript is template based not class based. Here, we don't create class to get the object. But we direct create objects.

- There are 3 ways to create objects.

  1. By object literal
  2. By creating instance of Object directly (using new keyword)
  3. By using an object constructor (using new keyword)

Objects

1. By object literal

The syntax of creating object using object literal is given below:

```
object={property1:value1,property2:value2.....propertyN:valueN}
```

As you can see, property and value is separated by : (colon).
Let’s see the simple example of creating object in JavaScript.

```
<script>
        emp={id:102,name:“Mohan Jayabalan",salary:40000}
        document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
```

Objects

2. By creating instance of Object

The syntax of creating object directly is given below:

```
var objectname=new Object();
```

Here, **new keyword** is used to create object.
Let’s see the example of creating object directly.

```
<script>
var emp=new Object();
emp.id=101;
emp.name="Mohan Jayabalan";
emp.salary=50000;
document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
```

Objects

3. By using an Object constructor

- Here, you need to create function with arguments. Each argument value can be assigned in the current object by using this keyword.
- The **this keyword** refers to the current object.
- The example of creating object by object constructor is given below.

```
<script>
        function emp(id,name,salary){
                this.id=id;
                this.name=name;
                this.salary=salary;
        }
        e=new emp(103,"Vimal Jaiswal",30000);
        document.write(e.id+" "+e.name+" "+e.salary);
</script>
```

Creating and using constructors

- As we previously discussed, Constructors are like regular functions, but we use them with the new keyword.

```
function Book() {
                // unfinished code
}
var myBook = new Book();
```

- The last line of the code creates an instance of Book and assigns it to a variable.
- Although the Book constructor doesn't do anything, myBook is still an instance of it. As you can see,
- there is no difference between this function and regular functions except that it's called with the new keyword and the function name is capitalized.

Determining the type of an instance

- To find out whether an object is an instance of another one, we use the instanceof operator:

```
myBook instanceof Book // true

myBook instanceof String // false
```

Accessing Properties on an Object

There are two ways to access properties on an object:

1. Dot Notation
2. Bracket Notation

You can access properties on an object by specifying the name of the object, followed by a dot (period) followed by the property name. This is the syntax:

```
let obj = {
      cat: 'meow',
    dog: 'woof'
    };
    let sound = obj.cat;
    console.log(sound);
// meow
```

Accessing Properties on an Object

1. Dot Notation

- You can access properties on an object by specifying the name of the object, followed by a dot (period) followed by the property name.
- This is the syntax:
    - objectName.propertyName;
- Example -

```
let obj = {
      cat: 'meow',
    dog: 'woof'
    };
let sound = obj.cat;
console.log(sound);

Output: meow
```

Accessing Properties on an Object

2. Bracket Notation

- You can access properties on an object by specifying the name of the object followed by the property name in brackets.
- Here's the syntax:
    - objectName["propertyName"]

- Example -

```
let arr = ['a','b','c'];
let letter = arr[1];
console.log(letter);
Output: b
```

Getters

- Getters and setters allow you to define Object Accessors (Computed Properties).
- This example uses a lang property to get the value of the language property.

```
// Create an object:
var person = {
    firstName: "John",
    lastName : "Doe",
    language : "en",
    get lang() {
        return this.language;
    }
};
// Display data from the object using a getter:
document.getElementById("demo").innerHTML
= person.lang;
```

Setters

- This example uses a lang property to set the value of the language property.

```
var person = {
    firstName: "John",
    lastName : "Doe",
    language : "",
    set lang(lang) {
        this.language = lang;
    }
};
// Set an object property using a setter:
person.lang = "en";
// Display data from the object:
document.getElementById("demo").innerHTML
= person.language;
```

This Keyword

- ***this*** keyword refers to an object, that object which is executing the current bit of **javascript** code.

```
function bike() {
   console.log(this.name);
}

var name = "Ninja";
var obj1 = { name: "Pulsar", bike: bike };
var obj2 = { name: "Gixxer", bike: bike };

bike();                         // "Ninja"
obj1.bike();                    // "Pulsar"
obj2.bike();                    // "Gixxer"
```

Object Constructors

- Object is the collection of related data or functionality in the form of **key**. This functionalities are usually consists of several functions and variables. All JavaScript values are objects except primitives.

```
var GFG = {
    subject : "programming",
    language : "JavaScript",
}
```

Here, *subject* and *language* are
the **keys** and *programming* and *JavaScript* are the **values**.

Using the Prototype Property

The JavaScript **prototype** property allows you to add new properties to object constructors:

```
function Person(first, last, age, eyecolor) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eyecolor;
}

Person.prototype.nationality = "English";
```

Using the Prototype Property

The JavaScript **prototype** property allows you to add new methods to objects constructors:

```
function Person(first, last, age, eyecolor) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eyecolor;
}

Person.prototype.name = function() {
  return this.firstName + " " + this.lastName;
};
```

Prototypal Inheritance

HTML DOM

HTML DOM

In the HTML DOM (Document Object Model), everything is a **node**:

The DOM is a W3C (World Wide Web Consortium) standard

The HTML DOM defines a standard way for accessing and manipulating HTML and XML documents.

- ➢ The document itself is a document node
- ➢ All HTML elements are element nodes
- ➢ All HTML attributes are attribute nodes
- ➢ Text inside HTML elements are text nodes
- ➢ Comments are comment nodes

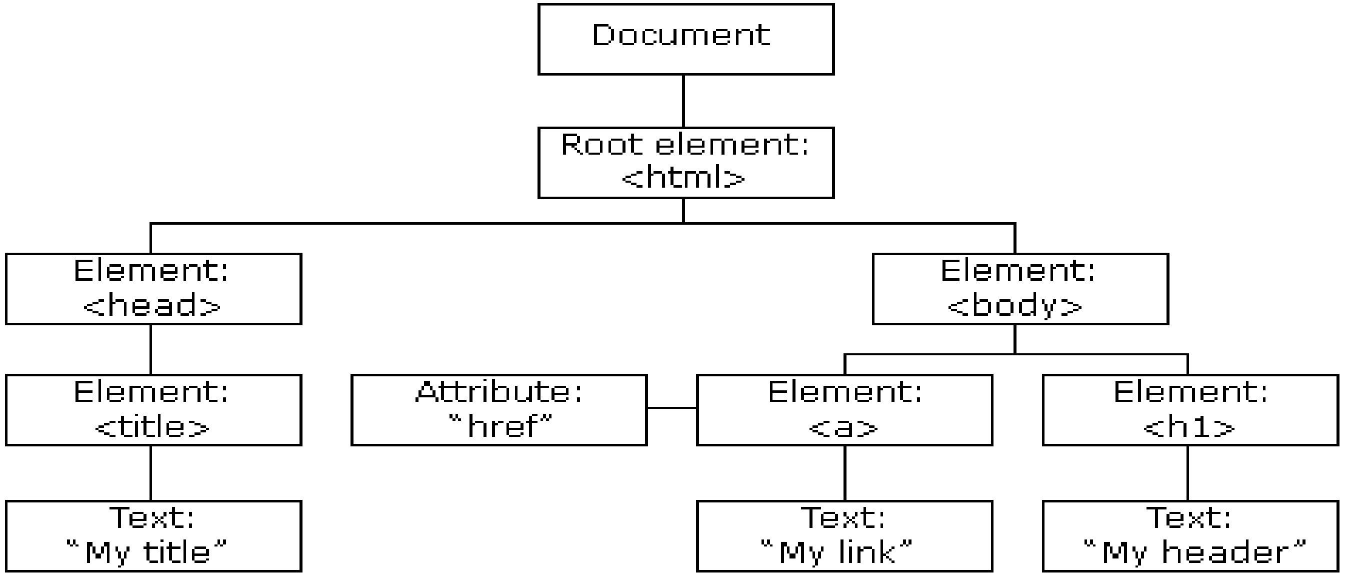
WHY DOM?

DOCUMENT OBJECT MODEL

is a programmatic representation to the documentary of HTML and XML documents. DOM is uses to interact web page. simply say it used when we need to interact with web pages like ADD/EDIT/DELETE contents on HTML Documents.

Understanding Document Object

The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document.



CRUD With DOM

1. Create an Element

To create an element use **document.createElement(<*elmName*>)** method. This method receives one parameter that is the name of element that will be created. DOM method **document.createTextNode(<*text*>)** is similar to document.createElement but will create a text node instead of an element.

```
var ul = document.getElementsByTagName("ul")[0];
var li = document.createElement("li");
var text = document.createTextNode("'li' element.");
li.appendChild(text);
ul.appendChild(li);
```

CRUD With DOM

2. Read Text of an Element

To create an element use **document.createElement(<*elmName*>)** method. This method receives one parameter that is the name of element that will be created. DOM method **document.createTextNode(<*text*>)** is similar to document.createElement but will create a text node instead of an element.

```
var ul = document.getElementsByTagName("ul")[0];
var li = document.createElement("li");
var text = document.createTextNode("'li' element.");
li.appendChild(text);
ul.appendChild(li);
```

CRUD With DOM

3. Update an Element Content

Update operation really similar to read operation, except updating an element means we change the node value with the new one. If ‘elm’ point to a <li> element, the following code:

```
var newValue = "New text node valuel";
elm.firstChild.nodeValue = newValue ;
```

CRUD With DOM

3. Delete an Element

Here, delete an element has two meanings. First delete the text node inside an element, second delete the element itself. Deleting an element can be accomplished using **removeChild(<*elm*>)** method. This method receives one parameter, that is child element that will be deleted.

```
var newValue = "New text node valuel";
elm.parentNode.removeChild(elm) ;
```

Traversing DOM

You can traverse in three directions:

1. Downwards-
   1. element.querySelector or element.querySelectorAll
   2. Children
2. Upwards-
   1. parentElement
   2. closest
3. Sideaways-
   1. nextElementSibling
   2. previousElementSibling
   3. Combining parentElement, children, and index

Working with Events

1. onclick Event Type
   - This is the most frequently used event type which occurs when a user clicks the left button of his mouse. You can put your validation, warning etc., against this event type.

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

Working with Events

- Other methods-

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

Browser Object Model

Browser Object Model

- There are no official standards for the Browser Object Model (BOM).
- Since modern browsers have implemented (almost) the same methods and properties for JavaScript interactivity,
- it is often referred to, as methods and properties of the BOM.

Window Object

- The **window** object is supported by all browsers. It represents the browser's window.
- All global JavaScript objects, functions, and variables automatically become members of the window object.
- Global variables are properties of the window object.
- Global functions are methods of the window object.
- Even the document object (of the HTML DOM) is a property of the window object:

```
window.document.getElementById("header");
```

is the same as:

```
document.getElementById("header");
```

Window Object

Window Size

- Two properties can be used to determine the size of the browser window.
- Both properties return the sizes in pixels:
    1. window.innerHeight - the inner height of the browser window (in pixels)
    2. window.innerWidth - the inner width of the browser window (in pixels)

Window Methods –

1. window.open() - open a new window
2. window.close() - close the current window
3. window.moveTo() - move the current window
4. window.resizeTo() - resize the current window

History Object

- The **JavaScript history object** represents an array of URLs visited by the user. By using this object, you can load previous, forward or any particular page.
- The history object is the window property, so it can be accessed by:
- **window.history** or **history.**

Methods-

1. history.back();//for previous page
2. history.forward();//for next page
3. history.go(2);//for next 2nd page
4. history.go(-2);//for previous 2nd page

Navigator Object

The **JavaScript navigator object** is used for browser detection. It can be used to get browser information such as appName, appCodeName, userAgent etc.
The navigator object is the window property, so it can be accessed by:
window.navigator or **navigator**

```
<script>
document.writeln("<br/>navigator.appCodeName: "+navigator.appCodeName);
document.writeln("<br/>navigator.appName: "+navigator.appName);
document.writeln("<br/>navigator.appVersion: "+navigator.appVersion);
document.writeln("<br/>navigator.cookieEnabled: "+navigator.cookieEnabled);
document.writeln("<br/>navigator.language: "+navigator.language);
document.writeln("<br/>navigator.userAgent: "+navigator.userAgent);
document.writeln("<br/>navigator.platform: "+navigator.platform);
document.writeln("<br/>navigator.onLine: "+navigator.onLine);
</script>
```

Timers

- A timer is a function that enables us to execute a function at a particular time.
- Using timers you can delay the execution of code so that it does not get done at the exact moment an event is triggered or the page is loaded.
- For example, you can use timers to change the advertisement banners on your website at regular intervals, or display a real-time clock, etc.
- There are two timer functions in JavaScript:
    1. setTimeout()
    2. setInterval()

Timers

```
<script>
function myFunction() {
		alert('Hello World!');
}
</script>
<button onclick="setTimeout(myFunction,
2000)">Click Me</button>
```

Cookies

Working with Forms in JavaScript

Working with Form in JavaScript

- Javascript wears many hats. You can use JavaScript to create special effects.

- You can use JavaScript to make your HTML pages "smarter" by exploiting its decision-making capabilities. And you can use JavaScript to enhance HTML forms.

- Of all the hats JavaScript can wear, its form processing features are among the most sought and used.

Setting Values from Form Element

For setting value to individual elements, we use the following code:

```
document.getElementById("mytext").value = "My value";
```

Getting Values from Form Element

The syntax for accessing the form object is as below:

```
oFormObject = document.forms['myform_id'];
```

For accessing individual elements, we use the following code:

```
oformElement = oFormObject.elements[index];
```

OR

```
oFormElement = oFormObject.elements["element_name"];
```

Form Validation

- HTML form validation can be done by JavaScript.

- If a form field (fname) is empty, this function alerts a message, and returns false, to prevent the form from being submitted:

```
function validateForm() {
  var x = document.forms["myForm"]["fname"].value;
  if (x == "") {
    alert("Name must be filled out");
    return false;
  }
}
```

Remote communication in JS

AJAX Programming

AJAX = **A**synchronous **J**avaScript **A**nd **X**ML.
AJAX is not a programming language.

AJAX just uses a combination of:

- A browser built-in XMLHttpRequest object (to request data from a web server)
- JavaScript and HTML DOM (to display or use the data)

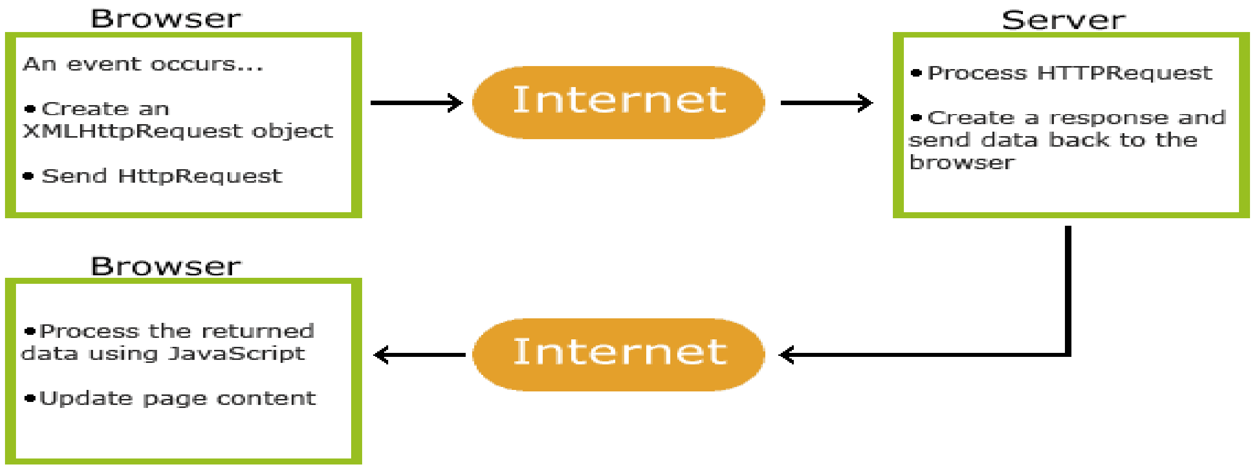
AJAX is a developer's dream, because you can:

- Update a web page without reloading the page
- Request data from a server - after the page has loaded
- Receive data from a server - after the page has loaded
- Send data to a server - in the background

AJAX Programming

```
<button type="button" onclick="loadDoc()">Change</button>


  function loadDoc() {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
      if (this.readyState == 4 && this.status == 200) {
       document.getElementById("demo").innerHTML = t
  his.responseText;
      }
    };
    xhttp.open("GET", "ajax_info.txt", true);
    xhttp.send();
  }
```

AJAX Programming


Browser
An event occurs...
• Create an XMLHttpRequest object
• Send HttpRequest
Internet
Server
• Process HTTPRequest
• Create a response and send data back to the browser
Browser
• Process the returned data using JavaScript
• Update page content
Internet

1. An event occurs in a web page (the page is loaded, a button is clicked)
2. An XMLHttpRequest object is created by JavaScript
3. The XMLHttpRequest object sends a request to a web server
4. The server processes the request
5. The server sends a response back to the web page
6. The response is read by JavaScript
7. Proper action (like page update) is performed by JavaScript

Working with Promises

- **Promises** are the ideal choice for handling asynchronous operations in the simplest manner. They can handle multiple asynchronous operations easily and provide better error handling than callbacks and events.

- **Benefits of Promises**
  1. Improves Code Readability
  2. Better handling of asynchronous operations
  3. Better flow of control definition in asynchronous logic
  4. Better Error Handling

- **A Promise has four states:**
  1. **fulfilled**: Action related to the promise succeeded
  2. **rejected**: Action related to the promise failed
  3. **pending**: Promise is still pending i.e not fulfilled or rejected yet
  4. **settled**: Promise has fulfilled or rejected

Working with Promises

Syntax:

```
var promise = new Promise(function(resolve, reject){
                //do something
});
```

- Promise constructor takes only one argument,a callback function.
- Callback function takes two arguments, *resolve* and *reject*
- Perform operations inside the callback function and if everything went well then call resolve.
- If desired operations do not go well then call reject.

Working with Promises

Example:

```
var promise = new Promise(function(resolve, reject)
{
  const x = "mohanjayabalan";
  const y = "mohanjayabalan"
  if(x === y) {
    resolve();
  } else {
    reject();
  }
});
```

```
promise.
    then(function () {
        console.log('Success, You are a Equal');
    }).
    catch(function () {
        console.log('Some error has occured');
    })
```

Working in Fetch

The Fetch API is a newer built-in feature of JavaScript that makes working with requests and responses easier.

```
// Replace ./data.json with your JSON feed
fetch('./data.json')
    .then(response => {
        return response.json()
    })
    .then(data => {
        // Work with JSON data here
        console.log(data)
    })
    .catch(err => {
    // Do something for an error here
})
```

What is JSON

- JSON stands for **J**ava**S**cript **O**bject **N**otation
- JSON is a lightweight data interchange format
- JSON is language independent
- JSON is "self-describing" and easy to understand

- JSON objects are written inside curly braces.
- Just like in JavaScript, objects can contain multiple name/value pairs:

```
{"firstName":"John", "lastName":"Doe"}
```

JSON arrays are written inside square brackets.

```
"employees":[
  {"firstName":"John", "lastName":"Doe"},
  {"firstName":"Anna", "lastName":"Smith"},
  {"firstName":"Peter", "lastName":"Jones"}
]
```

Why JSON?

- JSON is JavaScript Object Notation.
- It is a much-more compact way of transmitting sets of data across network connections as compared to XML.
- JSON is a light weight data.
- Its in key value pair as the JavaScript is.
- For REST API its widely used for data transfer from server to client.
- Nowadays many of the social media sites are using this.

Data Types

In JSON, values must be one of the following data types:

- a string
- a number
- an object (JSON object)
- an array
- a boolean
- *null*

JSON values cannot be one of the following data types:

- a function
- a date
- *undefined*

Parse and Stringify JSON

The JSON object, available in all modern browsers, has two very useful methods to deal with JSON-formatted content: parse and stringify. JSON.parse() takes a JSON string and transforms it into a JavaScript object.
JSON.stringify() takes a JavaScript object and transforms it into a JSON string.

```
const myObj = {
  name: 'Skip',
  age: 2,
  favoriteFood: 'Steak'
};
const myObjStr = JSON.stringify(myObj);
console.log(myObjStr);
// "{"name":"Skip","age":2,"favoriteFood":"Steak"}"
console.log(JSON.parse(myObjStr));
// Object {name:"Skip",age:2,favoriteFood:"Steak"}
```

JSON Objects

- JSON objects are surrounded by curly braces {}.
- JSON objects are written in key/value pairs.
- Keys must be strings, and values must be a valid JSON data type (string, number, object, array, boolean or null).
- Keys and values are separated by a colon.
- Each key/value pair is separated by a comma.

```
{ "name":"John", "age":30, "car":null }
```

You can access the object values by using dot (.) notation:

```
myObj = { "name":"John", "age":30, "car":null };
x = myObj.name;
```

JSON Arrays

- Arrays in JSON are almost the same as arrays in JavaScript.
- In JSON, array values must be of type string, number, object, array, boolean or *null*.
- In JavaScript, array values can be all of the above, plus any other valid JavaScript expression, including functions, dates, and *undefined*.

```
myObj={
        "name":"John",
        "age":30,
        "cars":[ "Ford", "BMW", "Fiat" ]
}


x = myObj.cars[0];
```

ES6 and Beyond

ES6 and Beyond

- **ECMAScript (ES)** is a scripting language specification standardized by ECMAScript International.
- It is used by applications to enable client-side scripting.
- Languages like JavaScript, Jscript and ActionScript are governed by this specification.
- This next will introduces you to ES6 implementation in JavaScript.

Introduction to Babel

- Babel is an awesome entry in the Web Developer toolset.

- It's an awesome tool, and it’s been around for quite some time, but nowadays almost every JavaScript developer relies on it, and this will continue going on, because Babel is now indispensable and has solved a big problem for everyone.

- The problem that every Web Developer has surely had: a feature of JavaScript is available in the latest release of a browser, but not in the older versions. Or maybe Chrome or Firefox implement it, but Safari iOS and Edge do not.

let and const

- ***let*** - is a signal that **the variable may be reassigned**, such as a counter in a loop, or a value swap in an algorithm.

It also signals that the variable will be used **only in the block it's defined in**, which is not always the entire containing function.

- ***const*** is a signal that **the identifier won't be reassigned.**

Temporal Dead Zone

In case of let and const variables, Basically, Temporal Dead Zone is a zone "before your variable is declared",
i.e where you can not access the value of these variables, it will throw an error.
ex.

```
let sum = a + 5;
console.log(sum)
let a = 5;
```

Above code gives an error, the same code will not give an error when we use var for variable 'a',
ex.

```
var sum = a;
console.log(sum) //prints undefined
var a = 5;
```

Template String

Template Strings use back-ticks (` `) rather than the single or double quotes we're used to with regular strings. A template string could thus be written as follows:

```
var greeting = `Yo World!`;
```

Template Strings can contain placeholders for string substitution using the ${ } syntax, as demonstrated below:

```
// Simple string substitution
var name = “Mohan";
console.log(`Yo, ${name}!`);

// => "Yo, Mohan!"
```

Array Destructuring

Destructuring is a JavaScript expression that makes it possible to unpack values from arrays, or properties from objects, into distinct variables. That is, we can extract data from arrays and objects and assign them to variables.

```
var introduction = ["Hello", "I" , "am", “Mohan"];
var greeting = introduction[0];
var name = introduction[3];
console.log(greeting);//"Hello"
console.log(name);//“Mohan"
```

If we want to extract data using arrays, it's quite simple using destructuring assignment. Let's refer to our first example for arrays. Instead of going through that repetitive process, we'll do this.

```
var introduction = ["Hello", "I" , "am", "Sarah"];
var [greeting, pronoun] = introduction;
console.log(greeting);//"Hello"
console.log(pronoun);//"I"
```

Object Destructuring

We want to extract data from an object and assign to new variables. See how tedious it is to extract such data. We have to repeatedly do the same thing.

```
var person = {name: “Mohan", country: “India", job: “Tester"};
var name = person.name;
var country = person.country;
var job = person.job;
console.log(name);//“Mohan"
console.log(country);//“India"
console.log(job);//Tester"
```

ES6 comes with destructuring to save the day.
Let's jump right into it.

Object Destructuring

Let us repeat the above example with ES6. Instead of assigning it one by one, we can use an object on the left to extract the data.

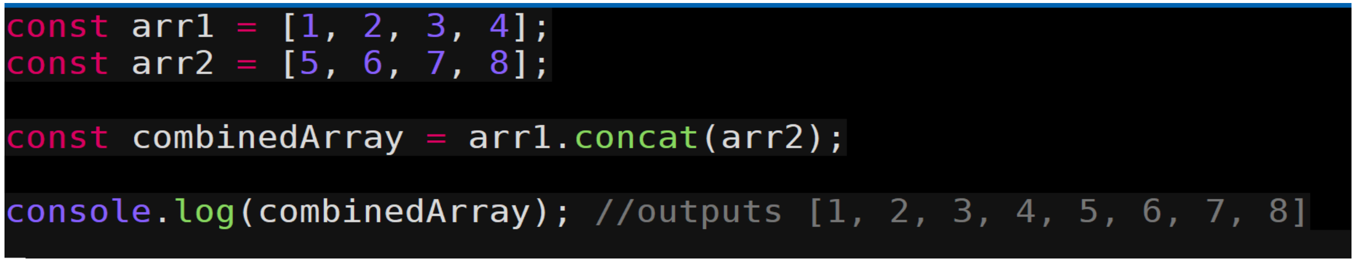
```
var person = {name: “Mohan", country: “India", job: “Tester"};
var {name, country , job} = person;
console.log(name);//“Mohan"
console.log(country);//“India"
console.log(job);//Tester"
```

One more way-

```
var {name, country, job} = {
            name: "Sarah",
            country: "Nigeria",
            job: "Developer“
};
console.log(name);//"Sarah"
console.log(country);//"Nigeria"
console.log(job);//Developer"
```

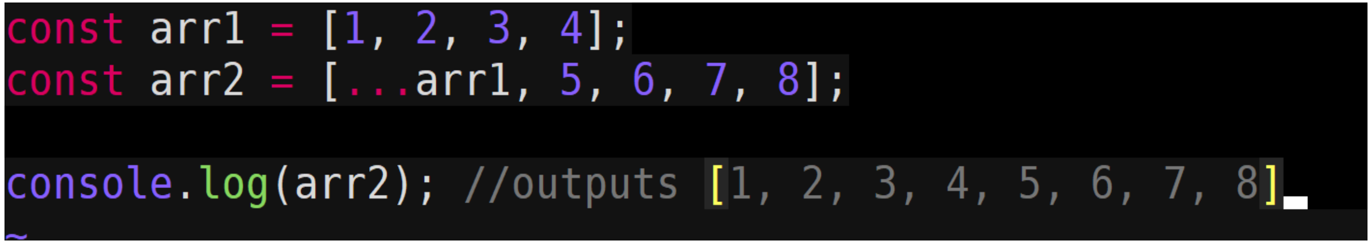
Spread Operator

The spread operator “spreads” the values in an iterable (arrays, strings) across zero or more arguments or elements. I’ll explain exactly what that means later on. But first, let’s look at how we might combine two arrays:



```
const arr1 = [1, 2, 3, 4];
const arr2 = [5, 6, 7, 8];

const combinedArray = arr1.concat(arr2);

console.log(combinedArray); //outputs [1, 2, 3, 4, 5, 6, 7, 8]
```

The spread operator takes an array (or any iterable) and spreads it values. Let’s take a look at how it works:

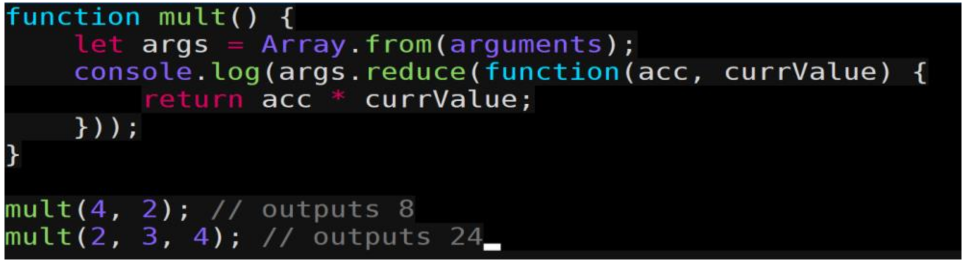


```
const arr1 = [1, 2, 3, 4];
const arr2 = [...arr1, 5, 6, 7, 8];

console.log(arr2); //outputs [1, 2, 3, 4, 5, 6, 7, 8]
```

That is much more simpler! The spread operator (...) takes the values of *arr1* and spreads them across *arr2.*

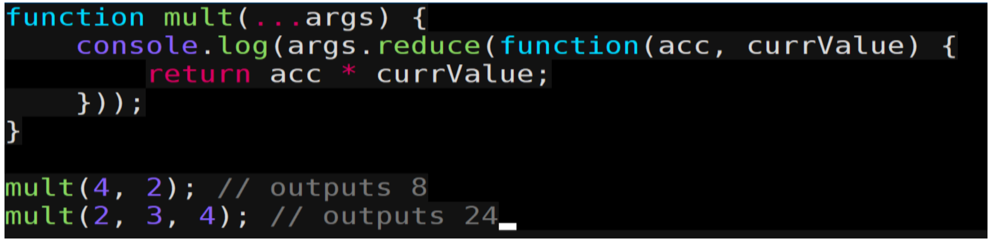
Rest Operator

We have a function that multiples the arguments we pass it and we need to be able to pass it any number of arguments. Here’s how we would write that function before ES6:



```
function mult() {
    let args = Array.from(arguments);
    console.log(args.reduce(function(acc, currValue) {
        return acc * currValue;
    }));
}

mult(4, 2); // outputs 8
mult(2, 3, 4); // outputs 24
```

Since the *argument* object isn’t an array, we first have to convert it into an array using the *Array.from* method before we can use the *reduce* method



```
function mult(...args) {
    console.log(args.reduce(function(acc, currValue) {
        return acc * currValue;
    }));
}

mult(4, 2); // outputs 8
mult(2, 3, 4); // outputs 24
```

The rest parameter gives us an easier and cleaner way of working with an indefinite number of parameters.

Arrow Functions

Arrow functions were introduced in ES6. Arrow functions allow us to write shorter function syntax Before:

```
hello = function() {
  return "Hello World!";
}
```

With Arrow Function:

```
hello = () => {
  return "Hello World!";
}
```

It gets shorter! If the function has only one statement, and the statement returns a value, you can remove the brackets *and* the return keyword:

```
hello = () => "Hello World!";
```

Default Function Arguments

- You just assign a value to the parameter when initializing its parameters.
- It is important to note that parameters are set from left to right.
- So the overwriting of default values will occur based on the position of your input value when you call the function.

```
function add(a=3, b=5) {
    return a + b;
}

add(4,2) // 6
add(4) // 9
add() // 8
```

https://www.w3schools.com/tags/ref_attributes.asp

https://www.javatpoint.com/html-attributes

https://www.w3schools.com/tags/att_type.asp

https://www.w3schools.com/html/html_basic.asp knowledge hut

thank you!