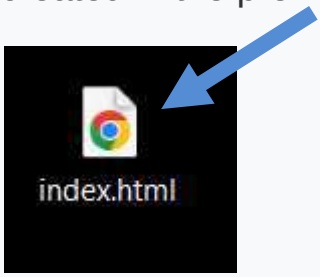
Javascript and HTML

Prerequisites for this JS Tutorial

Before starting with this JS tutorial, learners should complete the HTML and CSS tutorials provided earlier. You will need the `index.html` file created in the previous lessons.



Step 1: Sit at your computer with this mobile app and open the `index.html` file from the previous HTML tutorial.

JavaScript DOM (Document Object Model) manipulation technique. This technique is used for manipulating the content inside html tags and this technique is widely used in JavaScript Programming Language.

Basic Syntax of a Javascript DOM (Document Object Model)

```
document.getElementById("name_of_id").innerHTML = "New content!";
```

JavaScript Example

Type the below code in index.html file:

```
<!DOCTYPE html>
<html>
<head>
    <title>getElementById
 Example</title>
</head>
<body>
    <p>This is the original text.
 Noting Happens</p>
    <script>
document.getElementById("demo").innerHTML
= "New Text";
    </script>
</body>
</html>
```

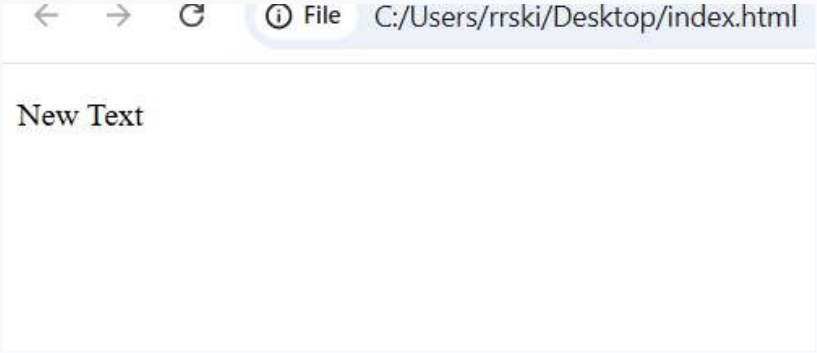
The Output of the above Program – "This is the original text and nothing happens."

Now add id="demo" inside the <p> html tag to activate the `document.getElementById("demo").innerHTML` Javascript Document Object Model.

Try to type the below code with id="demo" and execute

```
<!DOCTYPE html>
<html>
<head>
    <title>getElementById
 Example</title>
</head>
<body>
    <p id="demo">
This is the original text.
 Noting Happens</p>
    <script>
document.getElementById("demo").innerHTML
= "New Text";
    </script>
</body>
</html>
```

The output will be New Text.



Summary

Similar lines, javascript Document Object Model (DOM) manipulates the content inside html elements. Understanding `document.getElementById("demo").innerHTML="New Text"` will help in implementing Advanced Javascript Applications.

Advanced Tutorials

How to assign variables in Javascript

var x

var y

like this we can assign using Javascript

There are other types of declarations are present like let x; let y; or const x; const y; but later we can learn.

Arithmetic Operations in Javascript

1. Basic Arithmetic Operators

Operator	Description	Example	Result
+	Addition	5 + 3	8
-	Subtraction	5 - 3	2
*	Multiplication	5 * 3	15
/	Division	5 / 2	2.5
%	Modulus (Remainder)	5 % 2	1
**	Exponentiation (Power)	2 ** 3	8

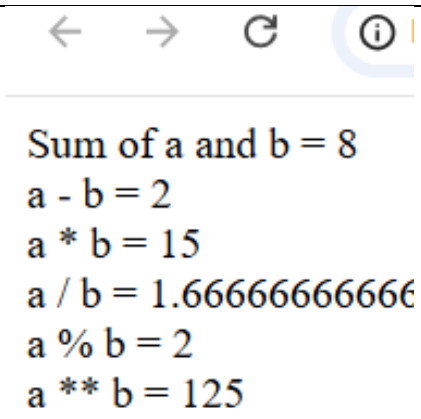
Example:

```
<html>
<head> <style> </style>
    </head>
<body>
    <p>
<script>
  let a = 5, b = 3;
  let c = a + b;
  let d = a - b;
  let e = a * b;
  let f = a / b;
  let g = a % b;
  let h = a ** b;

  document.write("Sum of a and b = " + c
+ "<br>");
  document.write("a - b = " + d +
"<br>");
  document.write("a * b = " + e +
"<br>");
  document.write("a / b = " + f +
"<br>");
  document.write("a % b = " + g +
```

Note: Always try to type your code manually. If you need to copy and paste, ensure you use a reliable source like ChatGPT. Errors often occur when learners copy and paste code, especially with single or double quotes. If an error arises, retype the single ‘ ’ or double “ ” quotes in your code before executing it.

```
"<br>");
  document.write("a ** b = " + h +
"<br>");
</script>
  </p>
</body>
</html>
```



Quick Reference – Javascript Operators

Operator	Description	Example	Result
ARITHMETIC OPERATORS			
+ (Addition)	Adds two numbers or concatenates strings.	5 + 3 or "Hello " + "World"	8 or "Hello World"
- (Subtraction)	Subtracts the second operand from the first.	5 - 3	2
* (Multiplication)	Multiplies two numbers.	5 * 3	15
/ (Division)	Divides the numerator by the denominator.	6 / 3	2
% (Modulus)	Returns the remainder of a division.	5 % 2	1
** (Exponentiation)	Raises the first operand to the power of the second operand.	2 ** 3	8
++ (Increment)	Increases the value of a variable by 1.	let a = 5; a++;	a = 6
-- (Decrement)	Decreases the value of a variable by 1.	let a = 5; a--;	a = 4
ASSIGNMENT OPERATORS			
= (Assignment)	Assigns the value of the right operand to the left operand.	let a = 10;	a = 10
+= (Add & Assign)	Adds the right operand to the left operand and assigns the result.	a += 5;	a = a + 5
-= (Subtract & Assign)	Subtracts the right operand from the left operand and assigns the result.	a -= 2;	a = a - 2
*= (Multiply & Assign)	Multiplies the left operand by the right operand and assigns the result.	a *= 3;	a = a * 3
/= (Divide & Assign)	Divides the left operand by the right operand and assigns the result.	a /= 2;	a = a / 2
%= (Modulus & Assign)	Computes the modulus and assigns the result to the left operand.	a %= 3;	a = a % 3
**= (Exponent & Assign)	Raises the left operand to the power of the right operand and assigns the result.	a **= 2;	a = a ** 2
COMPARISON OPERATORS			
== (Equality)	Compares two values for equality, ignoring type.	5 == '5'	true
=== (Strict Equality)	Compares two values and their types for equality.	5 === '5'	false
!= (Inequality)	Compares two values for inequality, ignoring type.	5 != '6'	true
!== (Strict Inequality)	Compares two values and their types for inequality.	5 !== '5'	true
> (Greater Than)	Checks if the left operand is greater than the right operand.	6 > 5	true
< (Less Than)	Checks if the left operand is less than the right operand.	5 < 6	true
>= (Greater or Equal)	Checks if the left operand is greater than or equal to the right operand.	5 >= 5	true
<= (Less or Equal)	Checks if the left operand is less than or equal to the right operand.	5 <= 6	true
LOGICAL OPERATORS			
&& (Logical AND)	Returns true if both operands are true.	true && false	false
` (Logical OR)	Returns true if at least one operand is true.		
! (Logical NOT)	Reverses the truth value of its operand.	!true	false
BITWISE OPERATORS			
& (Bitwise AND)	Performs a bitwise AND operation.	5 & 1	1
`	` (Bitwise OR)	Performs a bitwise OR operation.	`5

^ (Bitwise XOR)	Performs a bitwise XOR operation.	5 ^ 1	4
~ (Bitwise NOT)	Inverts the bits of its operand.	~5	-6
<< (Left Shift)	Shifts bits to the left, filling with zeros.	5 << 1	10
>> (Right Shift)	Shifts bits to the right, preserving the sign.	5 >> 1	2
>>> (Zero-Fill Right Shift)	Shifts bits to the right, filling with zeros.	5 >>> 1	2
OTHER OPERATORS			
?: (Ternary)	Shorthand for if-else.	a > b ? 'Yes' : 'No'	'Yes' (if a > b)
, (Comma)	Evaluates expressions from left to right, returning the last.	(1, 2, 3)	3
typeof	Returns the type of the operand.	typeof 42	'number'
delete	Deletes a property from an object.	delete obj.prop	true (if successful)
in	Checks if a property exists in an object.	'prop' in obj	true (if exists)
instanceof	Checks if an object is an instance of a constructor.	arr instanceof Array	true
?. (Optional Chaining)	Safely accesses nested properties.	obj?.prop	undefined (if prop doesn’t exist)

JavaScript Statements

Statement	Description	Syntax	Example	Output in Browser
Variable Declaration	Declares variables using `var`, `let`, or `const`.	`let varName;`	`<script>let x = 5; document.write("x = " + x);</script>`	`x = 5`
If Statement	Executes a block of code if a specified condition is true.	`if (condition) { ... }`	`<script>let x = 10; if (x > 5) { document.write("x is greater than 5"); }</script>`	`x is greater than 5`
If-Else Statement	Executes one block of code if a condition is true and another if it is false.	`if (condition) { ... } else { ... }`	`<script>let x = 3; if (x > 5) { document.write("x is greater"); } else { document.write("x is smaller"); }</script>`	`x is smaller`
Else If	Adds more conditions to an `if` statement.	`if (...) { ... } else if (...) { ... }`	`<script>let x = 5; if (x > 5) { document.write("x is large"); } else if (x === 5) { document.write("x is equal to 5"); } else { document.write("small"); }</script>`	`x is equal to 5`
Switch	Executes one block of code based on a variable's value, matching against `case` labels.	`switch (variable) { case value1: ...; break; ... }`	`<script>let x = 2; switch (x) { case 1: document.write("One"); break; case 2: document.write("Two"); break; default: document.write("Other"); }</script>`	`Two`

Javascript Loops

Statement	Description	Syntax	Example	Output in Browser
For Loop	Loops through a block of code a fixed number of times.	`for (initialization; condition; increment) { ... }`	`<script>for (let i = 1; i <= 5; i++) { document.write("Number: " + i + " "); }</script>`	`Number: 1 Number: 2 Number: 3 Number: 4 Number: 5`
While Loop	Repeats a block of code while a condition is true.	`while (condition) { ... }`	`<script>let i = 1; while (i <= 5) { document.write("Count: " + i + " "); i++; }</script>`	`Count: 1 Count: 2 Count: 3 Count: 4 Count: 5`
Do-While Loop	Executes a block of code at least once, then repeats while a condition is true.	`do { ... } while (condition);`	`<script>let i = 1; do { document.write("Iteration: " + i + " "); i++; } while (i <= 5);</script>`	`Iteration: 1 Iteration: 2 Iteration: 3 Iteration: 4 Iteration: 5`
Break	Exits a loop or `switch` statement immediately.	`break;`	`<script>for (let i = 1; i <= 5; i++) { if (i === 3) break; document.write("i: " + i + " "); }</script>`	`i: 1 i: 2`
Continue	Skips the current iteration of a loop and continues with the next.	`continue;`	`<script>for (let i = 1; i <= 5; i++) { if (i === 3) continue; document.write("i: " + i + " "); }</script>`	`i: 1 i: 2 i: 4 i: 5`
Function Declaration	Declares a reusable block of code that can be called.	`function functionName(params) { ... }`	`<script>function greet() { document.write("Hello! "); } greet(); greet();</script>`	`Hello! Hello!`

Return	Exits from a function and optionally returns a value.	`return value;`	`<script>function add(a, b) { return a + b; } document.write("Sum: " + add(5, 3));</script>`	`Sum: 8`
Try-Catch	Handles errors in a `try` block by running alternative code in a `catch` block.	`try { ... } catch (error) { ... }`	`<script>try { let x = y; } catch (e) { document.write("Error: " + e.message); }</script>`	`Error: y is not defined`

Functions in Javascript

1. Basic Function Declaration

```
<script>
function greet() {
    document.write("Hello, World!<br>");
}
greet(); // Calling the function
greet(); // Calling it again
</script>
```

Output:

Explanation: This is a simple function that prints a message to the webpage.

- **`function greet()`**: Declares a function named `greet`.
- **`document.write()`**: Prints "Hello, World!" to the webpage.
- **`greet();`**: Executes the function. You can call it multiple times.

Output:

```
Hello, World!
Hello, World!
```

2. Function with Parameters

```
<script>
function greet(name) {
    document.write("Hello, " + name + "!<br>");
}
greet("Alice");
greet("Bob");
</script>
```

Explanation: This function takes a parameter to make it dynamic.

- **Parameter (`name`)**: Acts as a placeholder for the value passed to the function.
- **Arguments**: `"Alice"` and `"Bob"` are passed to the `greet` function.

Output:

```
Hello, Alice!
Hello, Bob!
```

3. Function with Return Value

```
<script>
function add(a, b) {
    return a + b; // Returns the sum of a and b
}
let result = add(5, 3);
document.write("The sum is: " + result + "<br>");
</script>
```

Explanation: This function calculates and returns a value.

- `return`: Outputs the result of the calculation.
- **Storing in a variable**: The returned value is stored in `result`.

Output:

```
The sum is: 8
```

4. Function Expression

```
<script>
let multiply = function(a, b) {
    return a * b;
};
document.write("Product: " + multiply(4, 5) + "<br>");
</script>
```

Explanation: A function is assigned to a variable.

- **Function Expression**: A function is assigned to the variable `multiply`.
- You call it using the variable name (`multiply(4, 5)`).

Output:

```
Product: 20
```

5. Arrow Functions (ES6)

```
<script>
const divide = (a, b) => a / b;
document.write("Quotient: " + divide(10, 2) + "<br>");
</script>
```

Explanation: Arrow functions provide a concise syntax for writing functions.

Arrow Function: `=>` replaces the `function` keyword for a concise syntax.

- The single line automatically returns the value.

Output:

```
Quotient: 5
```

6. Default Parameters

```
<script>
function greet(name = "Guest") {
    document.write("Hello, " + name + "!<br>");
}
greet("Alice");
greet(); // No argument passed, uses default value
</script>
```

Explanation: Default parameters are used when no arguments are provided.

- **Default Parameter (`name = "Guest"`)**: Used if no argument is provided during the function call.

Output:

```
Hello, Alice!
Hello, Guest!
```

7. Functions with Objects

```
<script>
function showPerson(person) {
    document.write("Name: " + person.name + ", Age: " + person.age +
"<br>");
}
let person = { name: "Alice", age: 25 };
showPerson(person);
</script>
```

Explanation: This function demonstrates passing an object as an argument.

- **Object**: `person` contains properties like `name` and `age`.
- The function accesses object properties using `person.name` and `person.age`.

Output:

```
Name: Alice, Age: 25
```

8. Callback Functions

```
<script>
function calculate(a, b, operation) {
    return operation(a, b);
}
let sum = calculate(5, 3, (x, y) => x + y);
document.write("Sum: " + sum + "<br>");
</script>
```

Explanation: Callback functions are passed as arguments to other functions.

- **Callback Function**: `(x, y) => x + y` is passed as the `operation` argument.
- The `calculate` function executes the callback with `a` and `b`.

Output:

```
Sum: 8
```

9. Recursive Function

```
<script>
function factorial(n) {
    if (n === 0) return 1; // Base case
    return n * factorial(n - 1); // Recursive case
}
document.write("Factorial of 5: " + factorial(5) + "<br>");
</script>
```

Explanation: A recursive function calls itself to solve smaller subproblems.

- **Base Case**: Stops recursion when `n === 0`.
- **Recursive Case**: Calls `factorial` with `n - 1`.

Output:

```
Factorial of 5: 120
```

10. Anonymous Function (Self-Executing Function)

```
<script>
(function() {
    document.write("This runs immediately!<br>");
})();
</script>
```

Explanation: An anonymous function runs immediately after being defined.

- **Anonymous Function**: Defined without a name.
- **Self-Executing**: Runs as soon as it's defined using `()`.

Output:

```
This runs immediately!
```

11. Higher-Order Functions

```
<script>
function applyOperation(a, b, operation) {
    return operation(a, b);
}
function multiply(x, y) {
    return x * y;
}
document.write("Result: " + applyOperation(4, 5, multiply) + "<br>");
</script>
```

Explanation: Higher-order functions take other functions as arguments or return them

- **Higher-Order Function**: `applyOperation` takes `multiply` as an argument.
- **Reusable Logic**: You can pass different operations to `applyOperation`.

Output:

```
Result: 20
```