Java Script

Module 2

Content

- Introduction
- What can we do?
- Where to write?
- Generate output

Introduction

- JavaScript is the world's most popular programming language.
- JavaScript is the programming language of the Web.
- JavaScript is easy to learn.

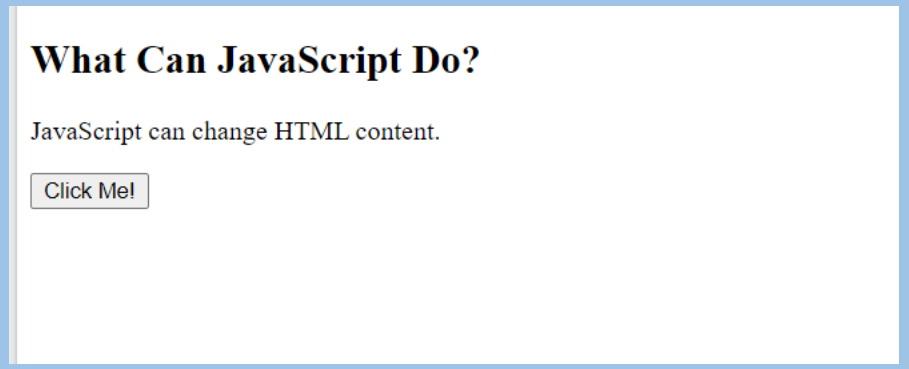
JavaScript is one of the 3 languages all web developers must learn:

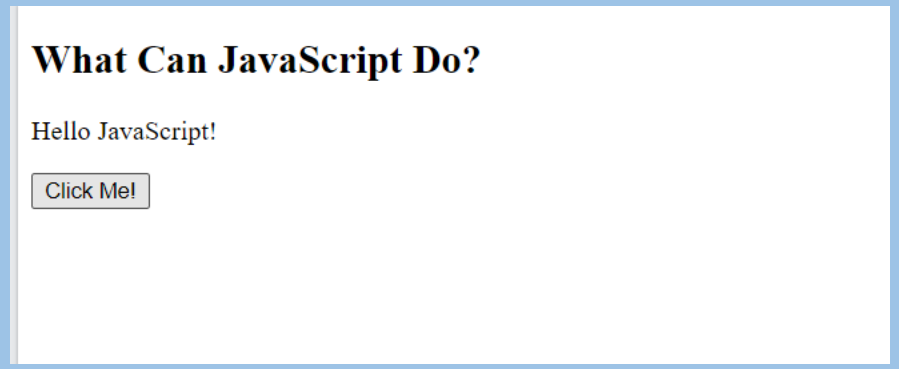
1. HTML to define the content of web pages
2. CSS to specify the layout of web pages
3. JavaScript to program the behavior of web pages

What JavaScript can do?

Change HTML Style (CSS)

```
<!DOCTYPE html>
<html>
<body>

<h2>What Can JavaScript Do?</h2>

<p id="demo">JavaScript can change HTML content.</p>

<button type="button" onclick="document.getElementById('demo').innerHTML = 'Hello JavaScript!'">Click Me!</button>

</body>
</html>
```

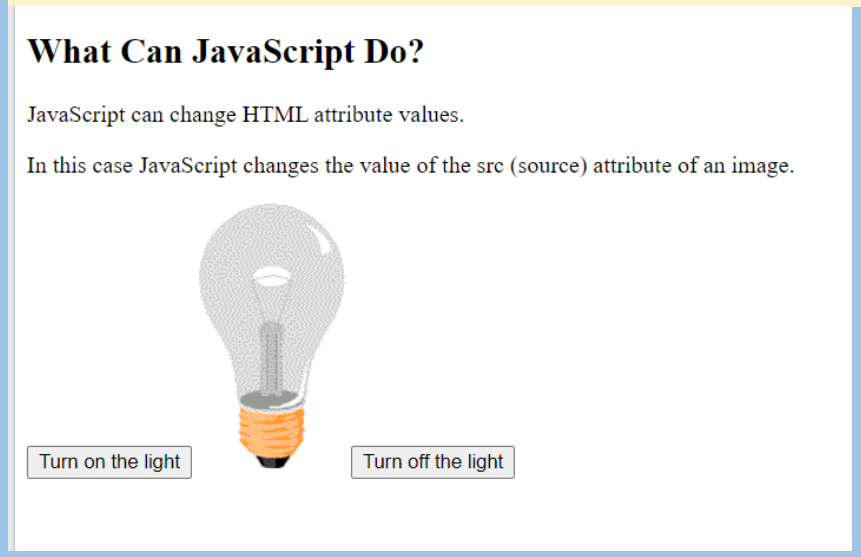


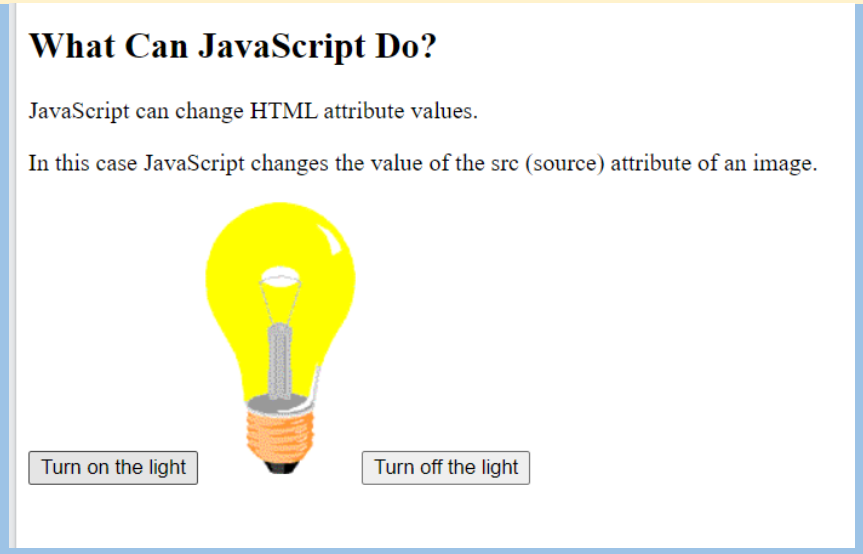


What JavaScript can do?

Change HTML Attribute Values

```
<!DOCTYPE html>
<html>
<body>

<h2>What Can JavaScript Do?</h2>
<p>JavaScript can change HTML attribute values.</p>
<p>In this case JavaScript changes the value of the src (source) attribute of an image.</p>

<button onclick="document.getElementById('myImage').src='pic_bulbon.gif'">Turn on the light</button>

<img id="myImage" src="pic_bulboff.gif" style="width:100px">

<button onclick="document.getElementById('myImage').src='pic_bulboff.gif'">Turn off the light</button>

</body>
</html>
```

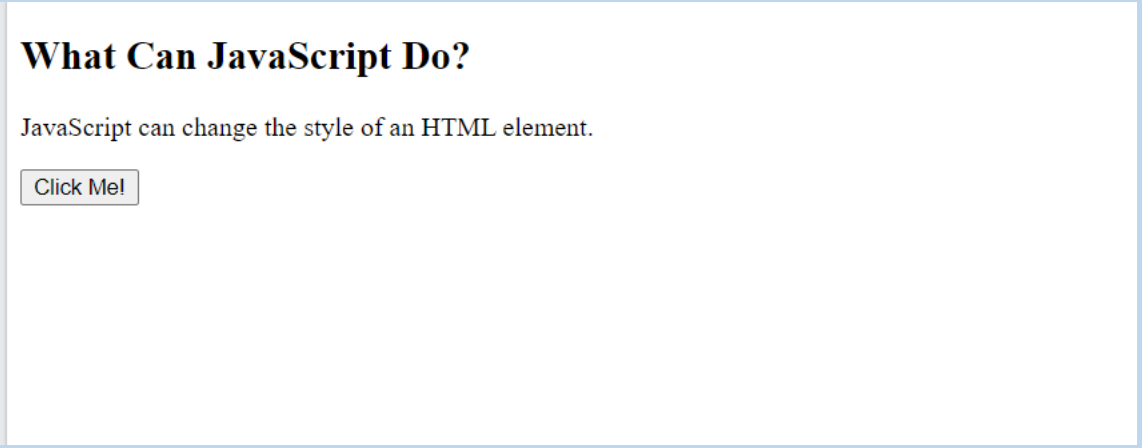


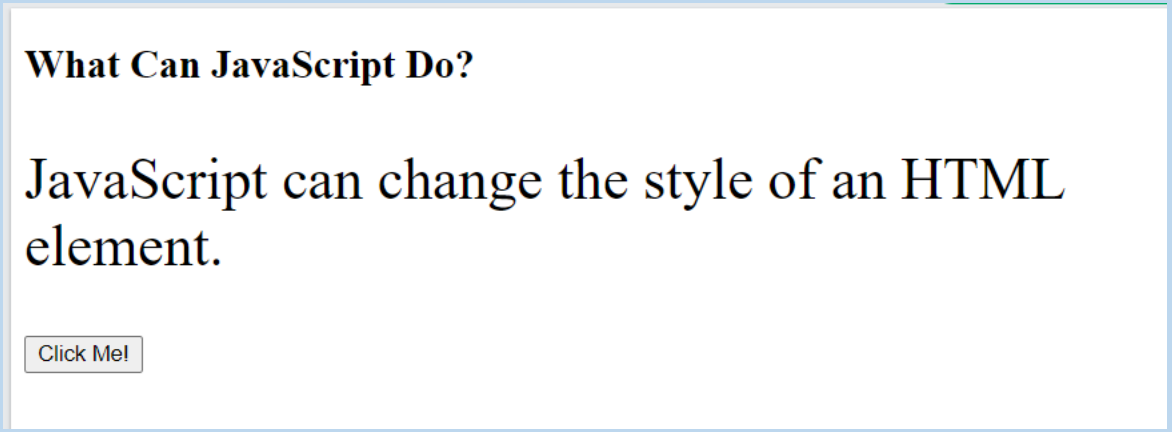


What JavaScript can do?

Change HTML Style (CSS)

```
<!DOCTYPE html>
<html>
<body>

<h2>What Can JavaScript Do?</h2>

<p id="demo">JavaScript can change the style of an HTML element.</p>

<button type="button" onclick="document.getElementById('demo').style.fontSize='35px'">Click Me!</button>

</body>
</html>
```

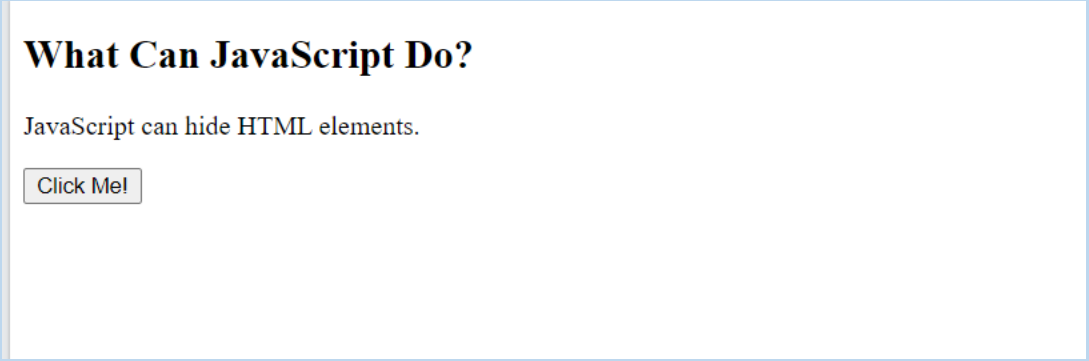


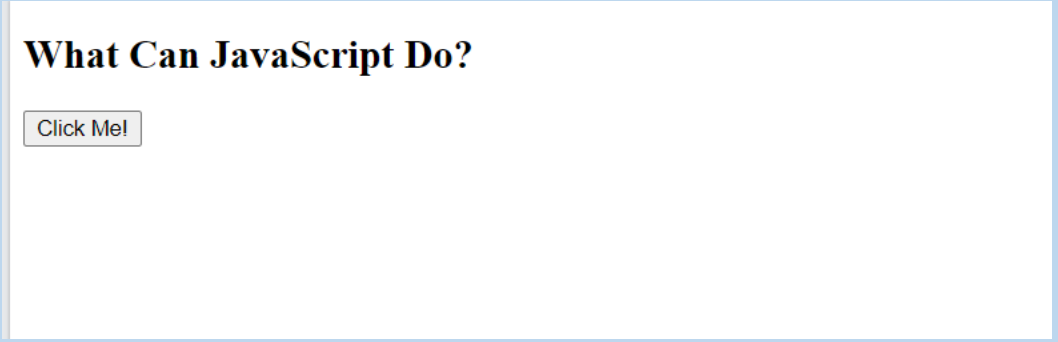


What JavaScript can do?

Hide HTML Elements

```
<!DOCTYPE html>
<html>
<body>

<h2>What Can JavaScript Do?</h2>

<p id="demo">JavaScript can hide HTML elements.</p>

<button type="button" onclick="document.getElementById('demo').style.display='none'">Click Me!</button>

</body>
</html>
```

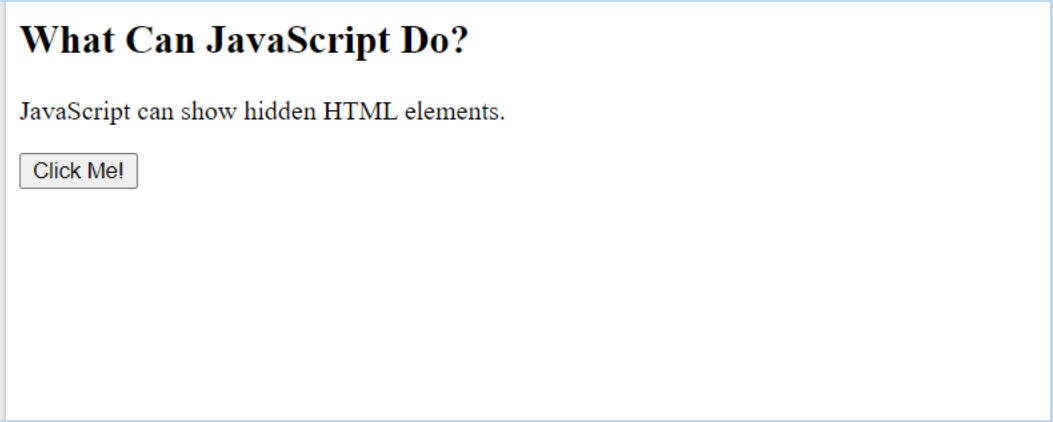


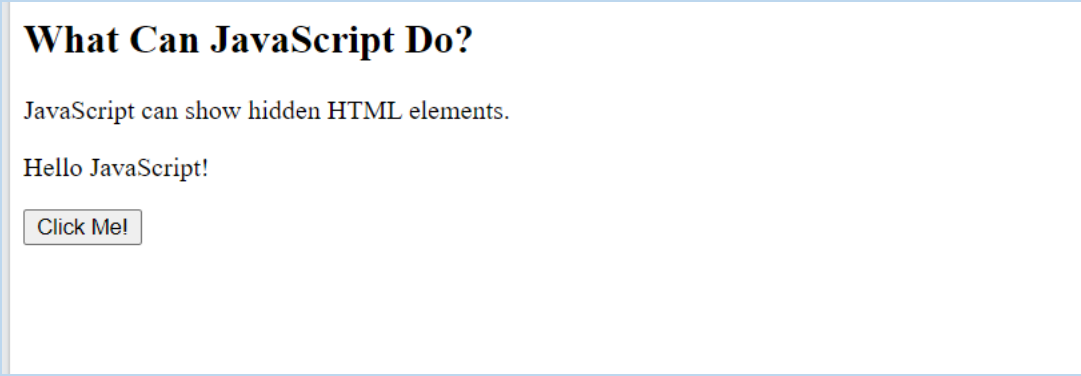


What JavaScript can do?

Show HTML Elements

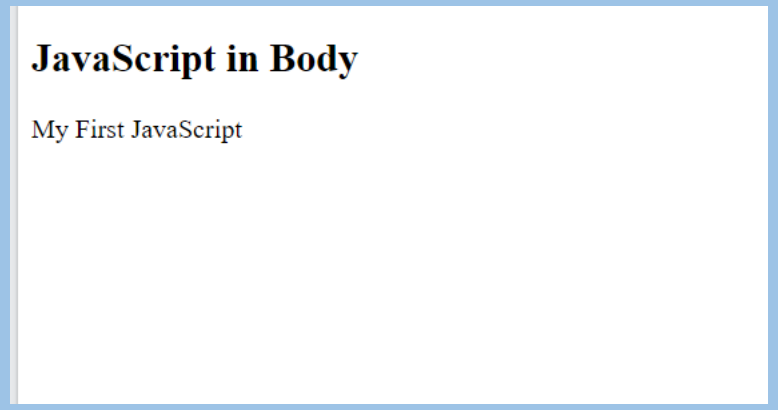
```
<!DOCTYPE html>
<html>
<body>

<h2>What Can JavaScript Do?</h2>

<p>JavaScript can show hidden HTML elements.</p>

<p id="demo" style="display:none">Hello JavaScript!</p>

<button type="button" onclick="document.getElementById('demo').style.display='block'">Click Me!</button>

</body>
</html>
```





Where to write JavaScript code?

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript in Body</h2>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = "My First JavaScript";
</script>

</body>
</html>
```

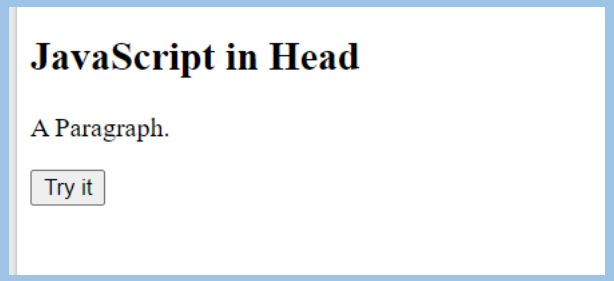


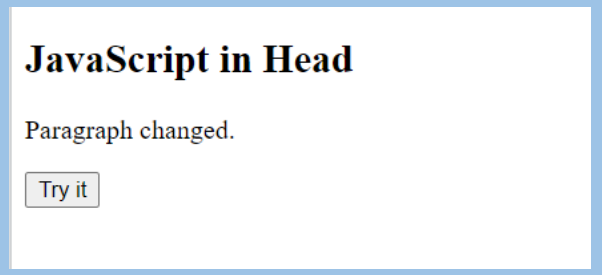
The <script> Tag

- In HTML, JavaScript code is inserted between <script> and </script> tags.
- We can place any number of scripts in an HTML document.
- Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both.

Where to write JavaScript code?

```
<!DOCTYPE html>
<html>
<head>

<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>

</head>
<body>

<h2>JavaScript in Head</h2>

<p id="demo">A Paragraph.</p>

<button type="button" onclick="myFunction()">Try it</button>

</body>
</html>
```





A JavaScript function

- It is a block of JavaScript code, that can be executed when "called" for.
  - Example: a function can be called when an event occurs, like when the user clicks a button.

Script is written in the head

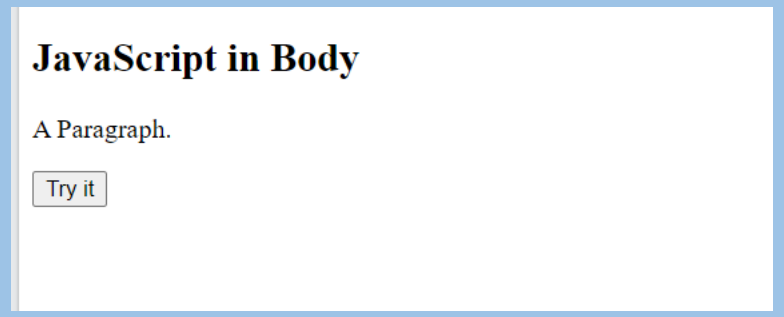
Where to write JavaScript code?

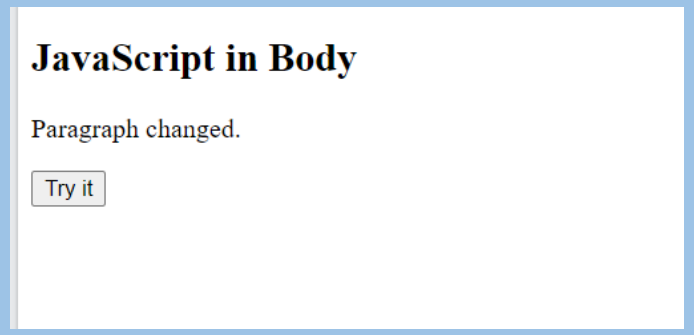
```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript in Body</h2>

<p id="demo">A Paragraph.</p>

<button type="button" onclick="myFunction()">Try it</button>

<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>

</body>
</html>
```

A JavaScript function

- It is a block of JavaScript code, that can be executed when "called" for.
  - Example: a function can be called when an event occurs, like when the user clicks a button.

Script is written in the body





Where to write JavaScript code?

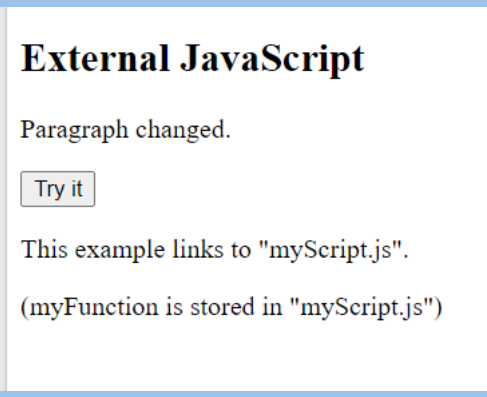
```
<!DOCTYPE html>
<html>
<body>

<h2>External JavaScript</h2>

<p id="demo">A Paragraph.</p>

<button type="button" onclick="myFunction()">Try it</button>

<p>This example links to "myScript.js".</p>
<p>(myFunction is stored in "myScript.js")</p>

<script src="myScript.js"></script>

</body>
</html>
```

```
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
```

myScript.js

External JavaScript

- Scripts can also be placed in external files
- External scripts are practical when the same code is used in many different web pages.
- JavaScript files have the file extension .js.
- To use an external script, put the name of the script file in the src (source) attribute of a <script> tag
- You can place an external script reference in <head> or <body> as you like.
- The script will behave as if it was located exactly where the <script> tag is located.
- External scripts cannot contain <script> tags.



Where to write JavaScript code?

External JavaScript

Placing scripts in external files has some advantages:

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

To add several script files to one page - use several script tags:

```
<script src="myScript1.js"></script>
<script src="myScript2.js"></script>
```

Where to write JavaScript code?

External References

An external script can be referenced in 3 different ways:

- With a full URL (a full web address)
  - Example: `<script src="https://www.w3schools.com/js/myScript.js"></script>`
- With a file path (like /js/)
  - Example: `<script src="/js/myScript.js"></script>`
- Without any path
  - Example: `<script src="myScript.js"></script>`

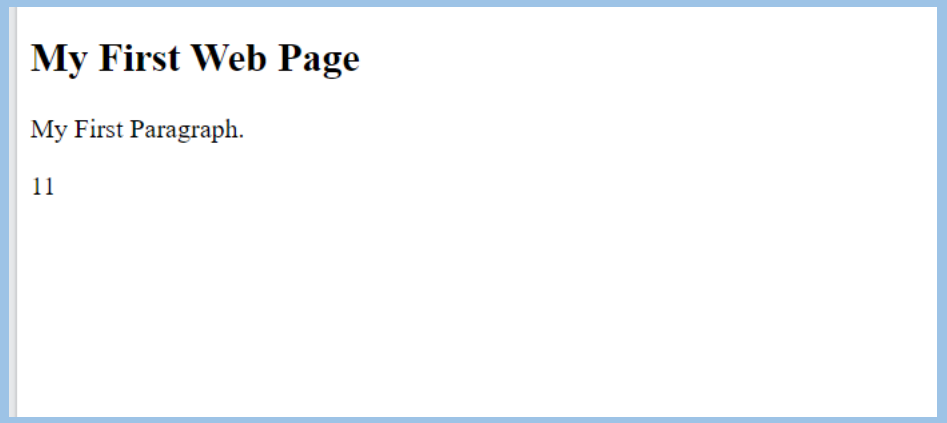
JavaScript Output

JavaScript can "display" data in different ways:

1. Writing into an HTML element, using innerHTML.
2. Writing into the HTML output using document.write().
3. Writing into an alert box, using window.alert().
4. Writing into the browser console, using console.log().

JavaScript Output

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My First Paragraph</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>

</body>
</html>
```

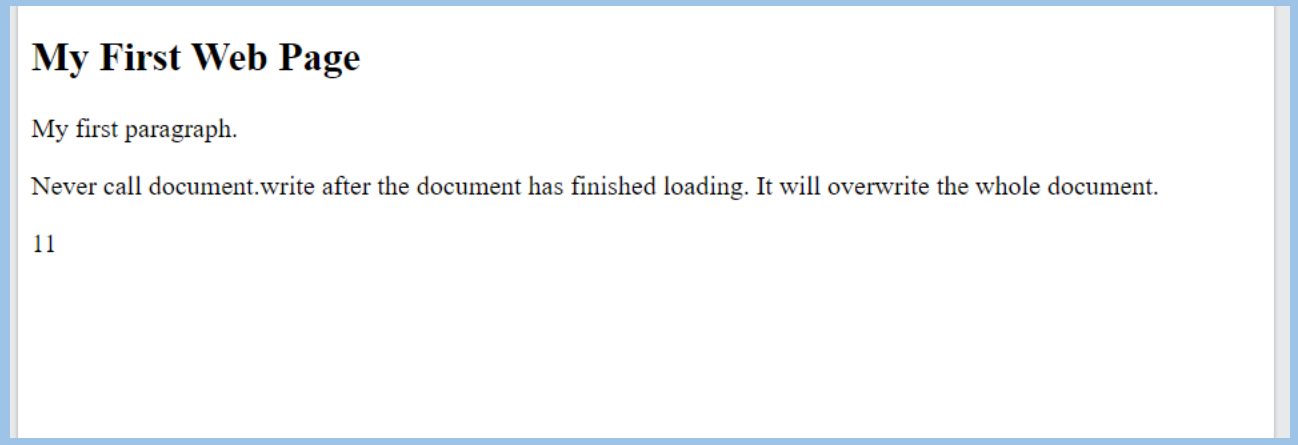


innerHTML

- To access an HTML element, JavaScript can use the document.getElementById(id) method.
- The id attribute defines the HTML element. The innerHTML property defines the HTML content:

JavaScript Output

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
document.write(5 + 6);
</script>

</body>
</html>
```



Document.write()

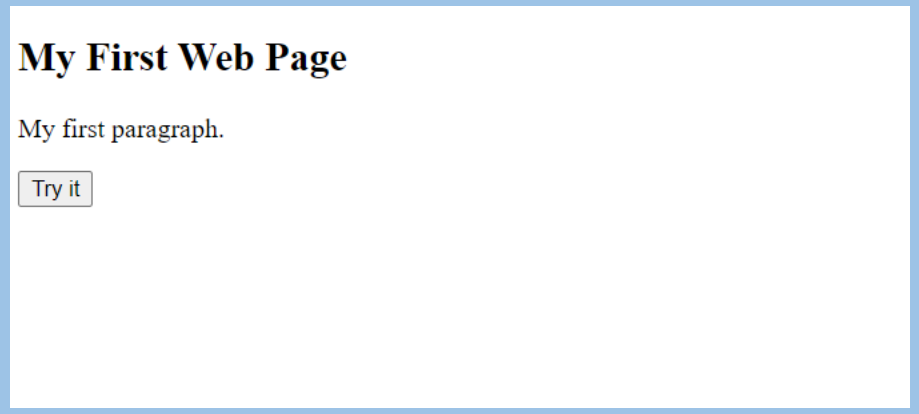
- For testing purposes, it is convenient to use document.write()
- Using document.write() after an HTML document is loaded, will delete all existing HTML

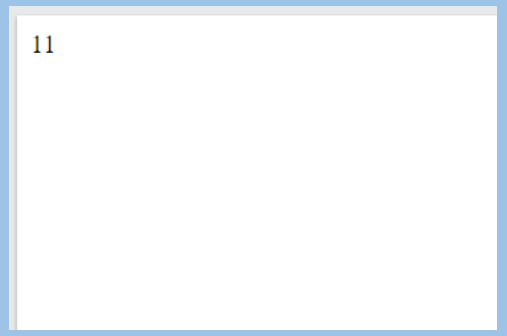
JavaScript Output

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<button type="button" onclick="document.write(5 + 6)">Try
it</button>

</body>
</html>
```

Document.write()

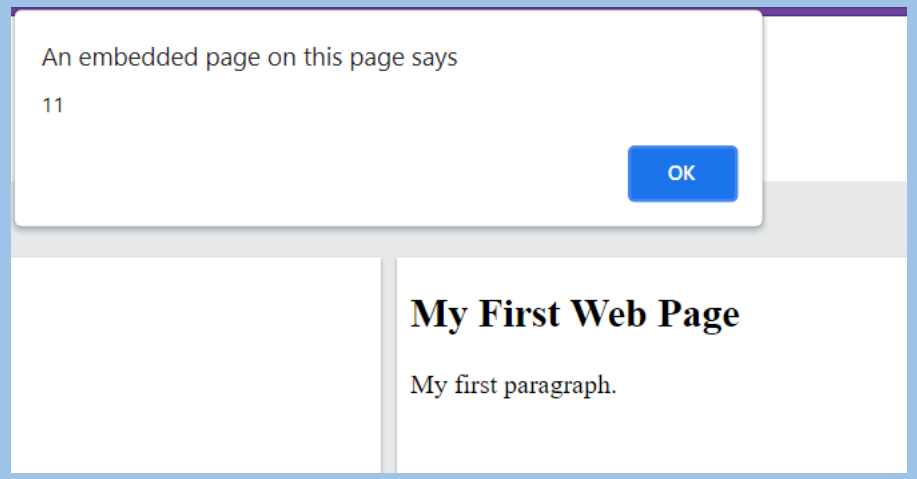
- For testing purposes, it is convenient to use document.write()
- Using document.write() after an HTML document is loaded, will delete all existing HTML





JavaScript Output

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
window.alert(5 + 6);
</script>

</body>
</html>
```



windows.alert()

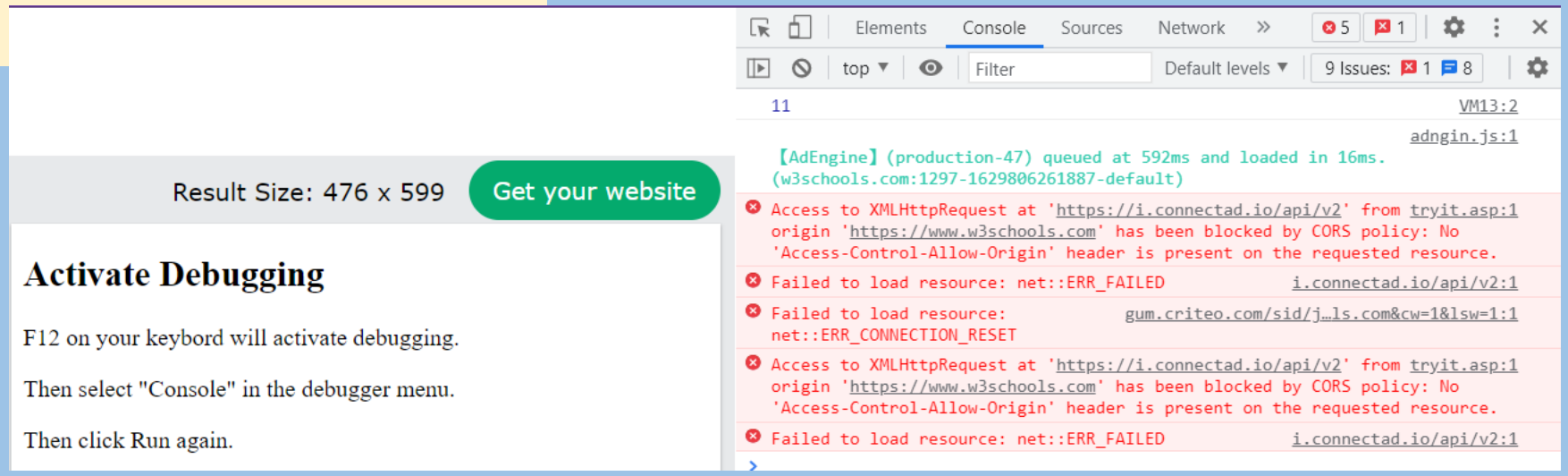
- We can use an alert box to display data
- We can skip the window keyword.
- In JavaScript, the window object is the global scope object,
- It means that variables, properties, and methods by default belong to the window object.
- This also means that specifying the window keyword is optional

JavaScript Output

```
<!DOCTYPE html>
<html>
<body>

<h2>Activate Debugging</h2>

<p>F12 on your keyboard will activate debugging.</p>
<p>Then select "Console" in the debugger menu.</p>
<p>Then click Run again.</p>

<script>
console.log(5 + 6);
</script>

</body>
</html>
```

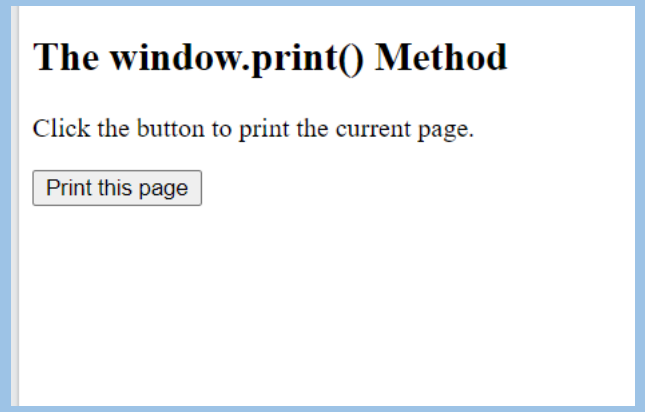
Console.log()

- For debugging purposes, we can call the console.log() method in the browser to display data



JavaScript Output

```
<!DOCTYPE html>
<html>
<body>

<h2>The window.print() Method</h2>

<p>Click the button to print the current page.</p>

<button onclick="window.print()">Print this page</button>

</body>
</html>
```

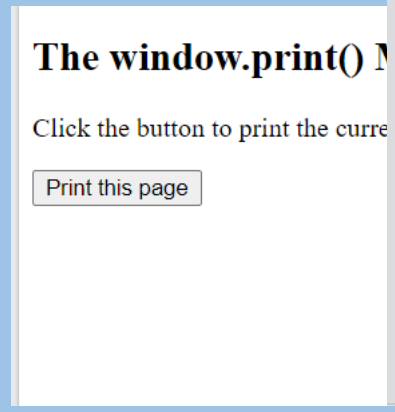


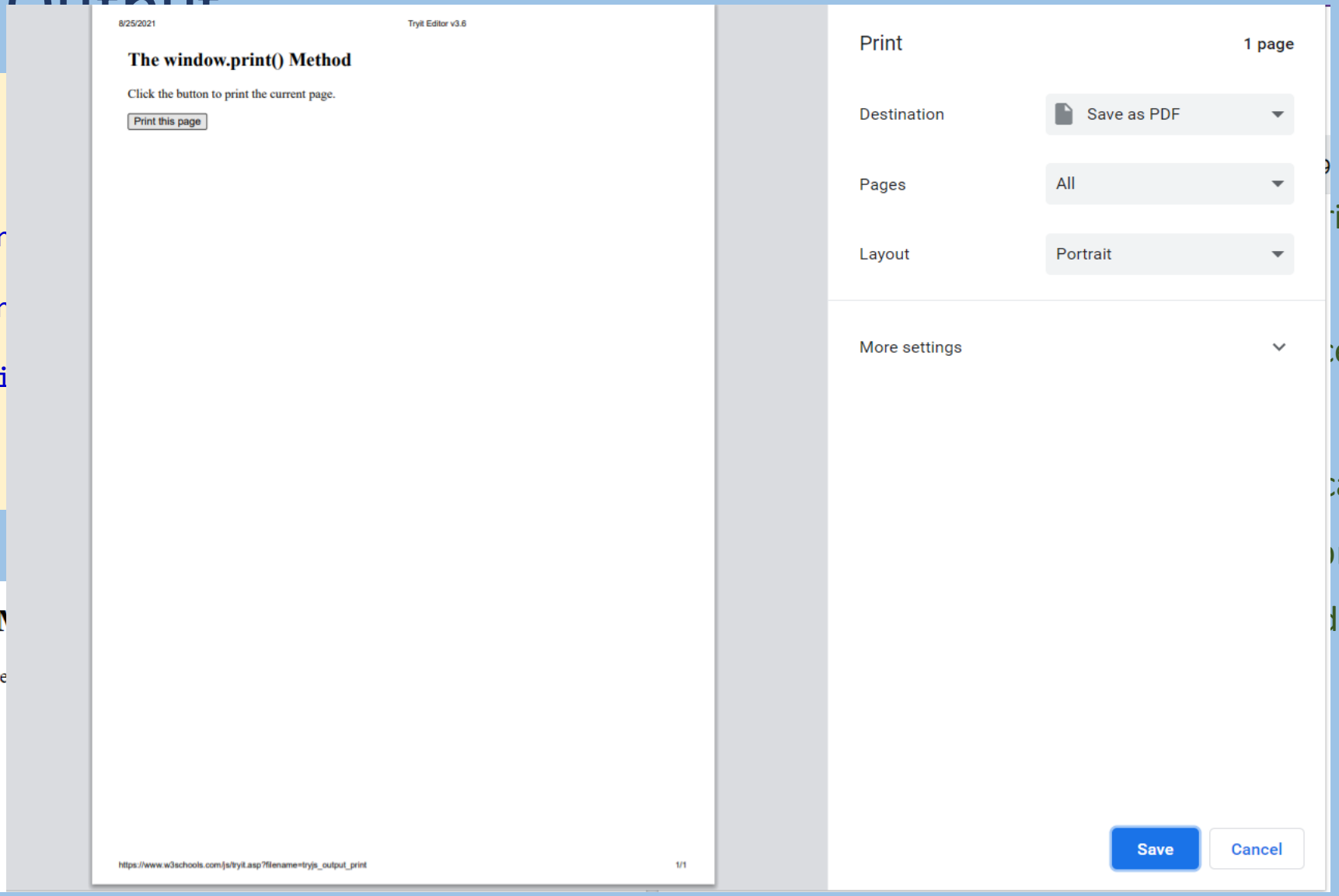
windows.print()

- JavaScript does not have any print object or print methods.
- You cannot access output devices from JavaScript.
- The only exception is that you can call the window.print() method in the browser to print the content of the current window.

JavaScript Output

```
<!DOCTYPE html>
<html>
<body>

<h2>The window.prir

<p>Click the buttor

<button onclick="wi

</body>
</html>
```





int object or print

es from

an call the

rowser to print

dow.

Java Script

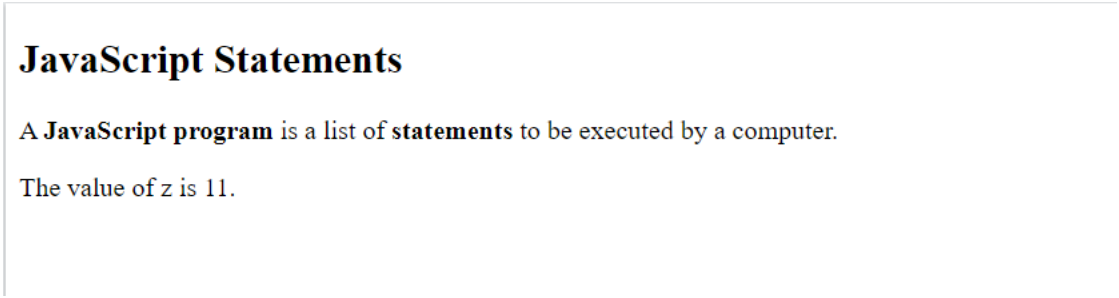
Module 2

Content

- Statements
- Syntax
- Function
- Objects

JavaScript Statements

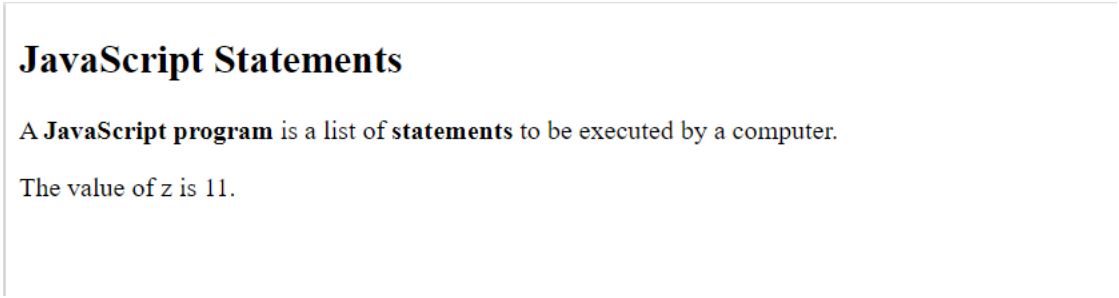
```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Statements</h2>

<p>A <b>JavaScript program</b> is a list of <b>statements</b> to be executed by a
computer.</p>

<p id="demo"></p>

<script>
let x, y, z;  // Statement 1
x = 5;        // Statement 2
y = 6;        // Statement 3
z = x + y;    // Statement 4

document.getElementById("demo").innerHTML = "The value of z is " + z + ".";
</script>

</body>
</html>
```



- A computer program is a list of "instructions" to be "executed" by a computer.
- In a programming language, these programming instructions are called statements.
- A JavaScript program is a list of programming statements.
- In HTML, JavaScript programs are executed by the web browser.
- JavaScript statements are composed of:
  - Values
  - Operators
  - Expressions
  - Keywords
  - Comments.

JavaScript Statements

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Statements</h2>

<p>A <b>JavaScript program</b> is a list of <b>statements</b> to be executed by a
computer.</p>

<p id="demo"></p>

<script>
let x, y, z;  // Statement 1
x = 5;        // Statement 2
y = 6;        // Statement 3
z = x + y;    // Statement 4

document.getElementById("demo").innerHTML = "The value of z is " + z + ".";
</script>

</body>
</html>
```



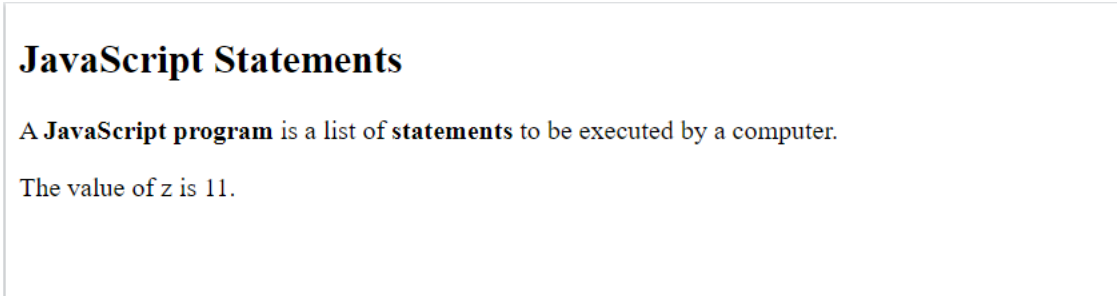
- Most JavaScript programs contain many JavaScript statements.
- The statements are executed, one by one, in the same order as they are written.
- JavaScript programs (and JavaScript statements) are often called JavaScript code

Semicolons:

- Semicolons separate JavaScript statements.
- Add a semicolon at the end of each executable statement
- When separated by semicolons, multiple statements on one line are allowed
- Ending statements with semicolon is not required, but highly recommended

JavaScript Statements

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Statements</h2>

<p>A <b>JavaScript program</b> is a list of <b>statements</b> to be executed by a
computer.</p>

<p id="demo"></p>

<script>
let x, y, z;  // Statement 1
x = 5;        // Statement 2
y = 6;        // Statement 3
z = x + y;    // Statement 4

document.getElementById("demo").innerHTML = "The value of z is " + z + ".";
</script>

</body>
</html>
```



JavaScript White Space

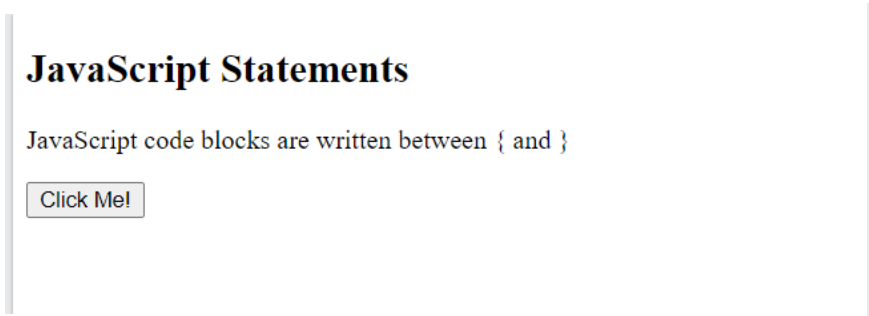
- JavaScript ignores multiple spaces.
- We can add white space to your script to make it more readable
- A good practice is to put spaces around operators (= + - * /)

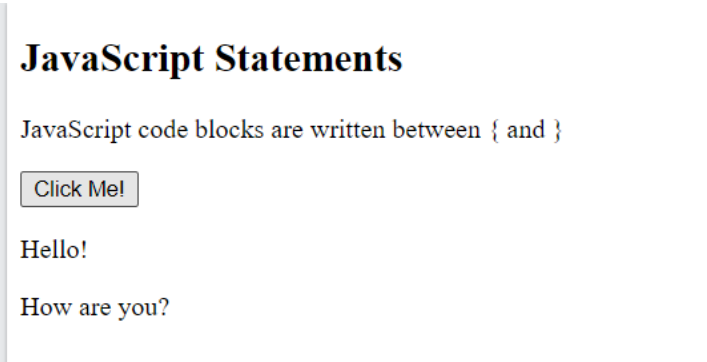
JavaScript Line Length and Line Breaks

- For best readability, programmers often like to avoid code lines longer than 80 characters.
- If a JavaScript statement does not fit on one line, the best place to break it is after an operator:

JavaScript Statements

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Statements</h2>

<p>JavaScript code blocks are written between { and }</p>

<button type="button" onclick="myFunction()">Click Me!</button>

<p id="demo1"></p>
<p id="demo2"></p>

<script>
function myFunction() {
  document.getElementById("demo1").innerHTML = "Hello!";
  document.getElementById("demo2").innerHTML = "How are you?";
}
</script>

</body>
</html>
```





JavaScript Code Blocks

- JavaScript statements can be grouped together in code blocks, inside curly brackets {...}.
- The purpose of code blocks is to define statements to be executed together.
- One place you will find statements grouped together in blocks, is in JavaScript functions

JavaScript Keywords

JavaScript statements often start with a keyword to identify the JavaScript action to be performed.

Some of the keywords:

Keyword	Description
var	Declares a variable
let	Declares a block variable
const	Declares a block constant
if	Marks a block of statements to be executed on a condition
switch	Marks a block of statements to be executed in different cases
for	Marks a block of statements to be executed in a loop
function	Declares a function
return	Exits a function
try	Implements error handling to a block of statements

JavaScript Syntax

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

JavaScript Values

The JavaScript syntax defines two types of values:

- Fixed values – Literals
- Variable values – Variables

JavaScript Literals

The two most important syntax rules for fixed values are:

1. Numbers are written with or without decimals
2. Strings are text, written within double or single quotes

JavaScript Variables

- In a programming language, variables are used to store data values.
- JavaScript uses the keywords var, let and const to declare variables.
- An equal sign is used to assign values to variables.

JavaScript Syntax

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

JavaScript Operators

- Arithmetic operators: + , - , * , / , ** , % , ++ , --
- Assignment operator: = , += , -= , *= , /= , %= , **=
- String operator: + , +=
- Comparison Operator: == (Equal to) , === (Equal Value and Equal Type), != , < , > , <= , >= , ?
- Logical Operator: && , || , !
- Type Operators: typeof, instanceof
- Bitwise Operators: & , | , ~ , ^ , << , >>

JavaScript Syntax

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

JavaScript Operators

Arithmetic operators: Addition (+), Subtraction(-), Multiplication(*), Division (/)

Assignment operator: (=)

JavaScript Expressions

- An expression is a combination of values, variables, and operators, which computes to a value.
- The computation is called an evaluation.
  - For example, 5 * 10 evaluates to 50
- Expressions can also contain variable values
- The values can be of various types, such as numbers and strings.
  - For example, "John" + " " + "Doe", evaluates to "John Doe"

JavaScript Syntax

JavaScript Comments

Not all JavaScript statements are "executed".

Code after double slashes // or between /* and */ is treated as a comment.

Comments are ignored, and will not be executed

JavaScript Identifiers

- Identifiers are names.
- In JavaScript, identifiers are used to name variables (and keywords, and functions, and labels).
- The rules for legal names are much the same in most programming languages.
- In JavaScript, the first character must be a letter, or an underscore (_), or a dollar sign ($).
- Numbers are not allowed as the first character.
- Subsequent characters may be letters, digits, underscores, or dollar signs.
- All JavaScript identifiers are case sensitive
- JavaScript programmers tend to use camel case that starts with a lowercase letter:
    - Example: firstName, lastName, masterCard, interCity

JavaScript Functions

- It is a block of code designed to perform a particular task
- It is executed when "something" invokes it (calls it)

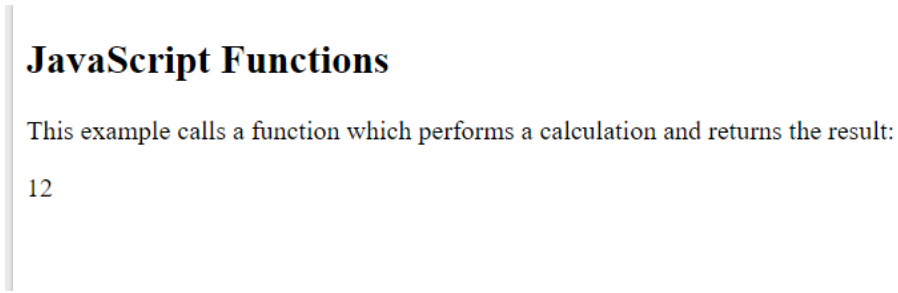
Syntax:

- It 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).
- The parentheses may include parameter names separated by commas:
- (parameter1, parameter2, ...)
- The code to be executed, by the function, is placed inside curly brackets: { }
- Function parameters are listed inside the parentheses () in the function definition.
- Function arguments are the values received by the function when it is invoked.
- Inside the function, the arguments (the parameters) behave as local variables.
- A Function is much the same as a Procedure or a Subroutine, in other programming languages.

```
function name(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

JavaScript Functions

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Functions</h2>

<p>This example calls a function which performs a calculation and returns the
result:</p>

<p id="demo"></p>

<script>
var x = myFunction(4, 3);
document.getElementById("demo").innerHTML = x;

function myFunction(a, b) {
  return a * b;
}
</script>

</body>
</html>
```



Function Invocation

The code inside the function will execute when "something" invokes (calls) the function:

- When an event occurs (when a user clicks a button)
- When it is invoked (called) from JavaScript code
- Automatically (self invoked)

Function Return

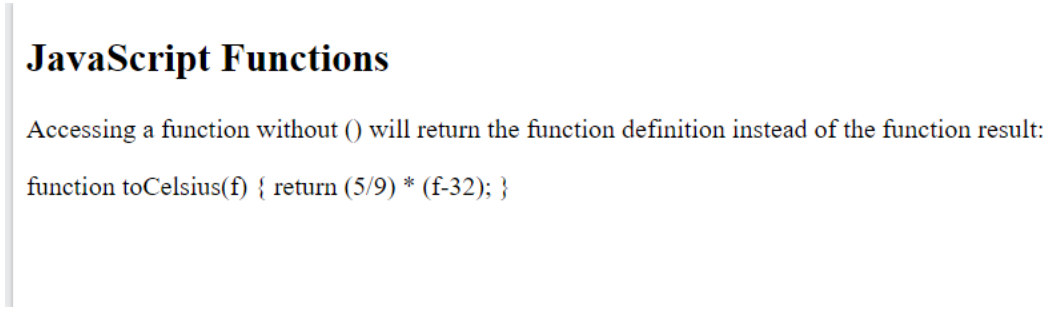
- When JavaScript reaches a return statement, the function will stop executing.
- If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.
- Functions often compute a return value. The return value is "returned" back to the "caller

JavaScript Functions

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Functions</h2>

<p>Accessing a function without () will return the function definition instead of the
function result:</p>
<p id="demo"></p>

<script>
function toCelsius(f) {
  return (5/9) * (f-32);
}
document.getElementById("demo").innerHTML = toCelsius;
</script>

</body>
</html>
```

Function Invocation

Accessing a function without () will return the function object instead of the function result.



JavaScript Objects

Real Life Objects, Properties, and Methods

- In real life, a car is an object.
- A car has properties like weight and color, and methods like start and stop
- All cars have the same properties, but the property values differ from car to car.
- All cars have the same methods, but the methods are performed at different times.

JavaScript Objects

- Objects are variables but it contain many values.
- The values are written as name:value pairs (name and value separated by a colon).
- It is a common practice to declare objects with the const keyword.
- We can define (and create) a JavaScript object with an object literal
- Spaces and line breaks are not important. An object definition can span multiple lines

JavaScript Objects

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Objects</h2>

<p id="demo"></p>

<script>
// Create an object:
const car = {type:"Fiat", model:"500", color:"white"};

// Display some data from the object:
document.getElementById("demo").innerHTML = "The car type is " + car.type;
</script>

</body>
</html>
```

Function Invocation

Accessing a function without () will return the function object instead of the function result.

JavaScript Functions

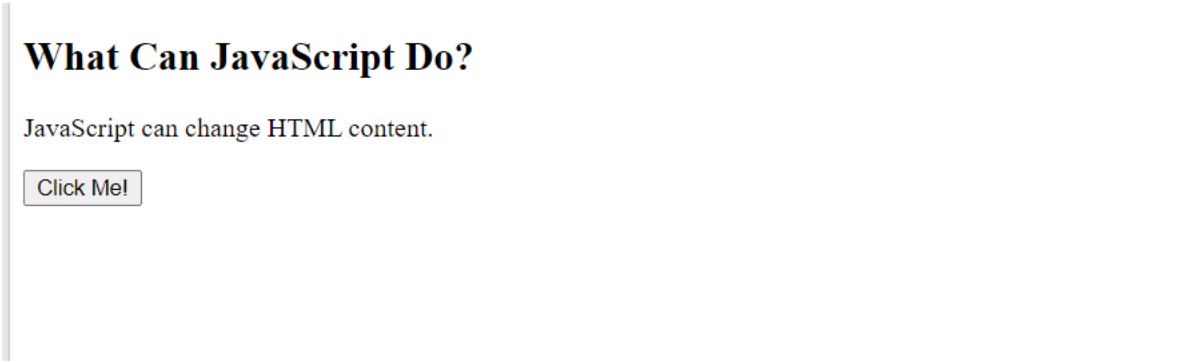
Accessing a function without () will return the function definition instead of the function result:

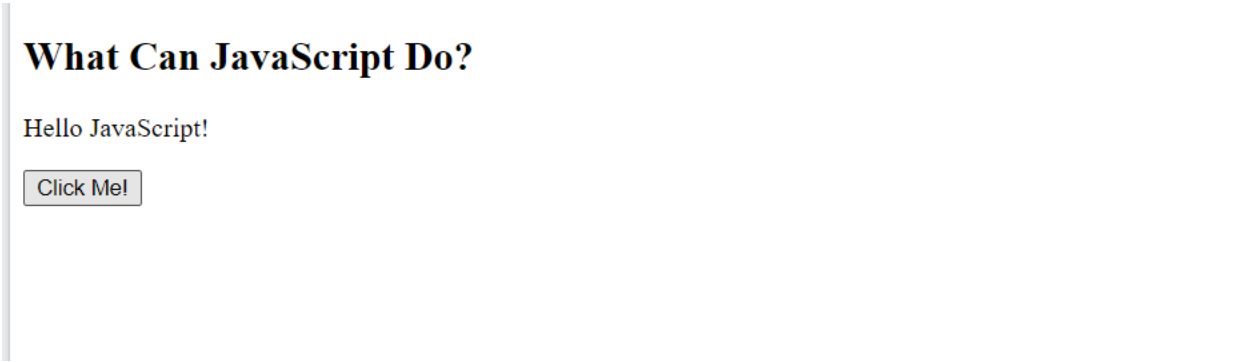
function toCelsius(f) { return (5/9) * (f-32); }

What JavaScript can do?

Change HTML Style (CSS)

```
<!DOCTYPE html>
<html>
<body>

<h2>What Can JavaScript Do?</h2>

<p id="demo">JavaScript can change HTML content.</p>

<button type="button" onclick="document.getElementById('demo').innerHTML = 'Hello JavaScript!'">Click Me!</button>

</body>
</html>
```

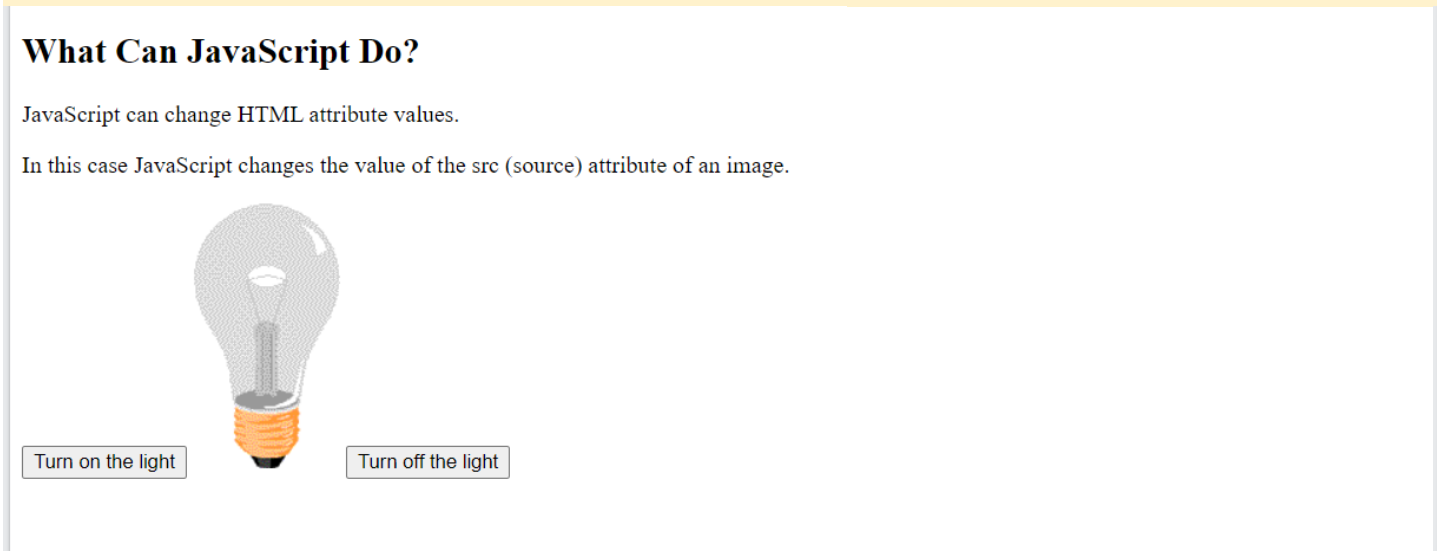


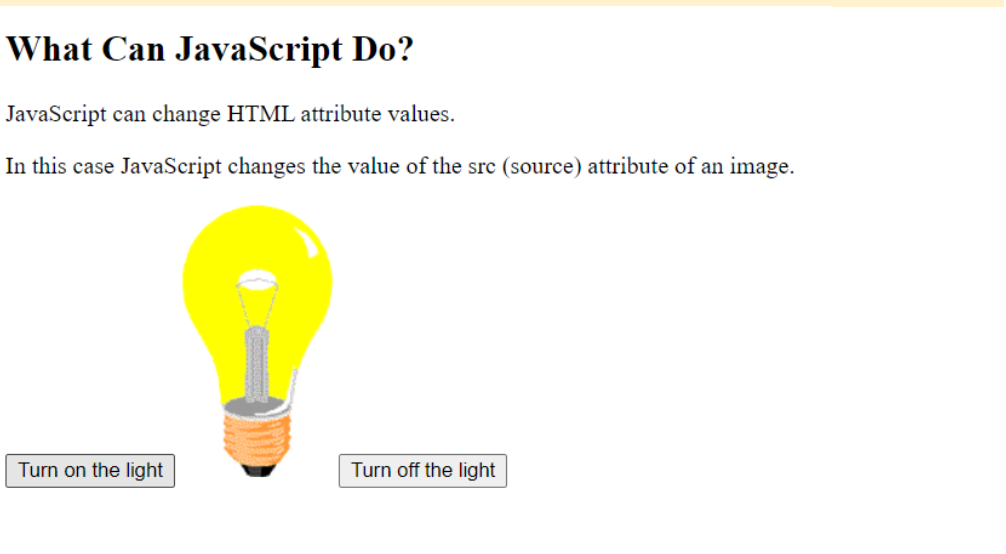


What JavaScript can do?

Change HTML Attribute Values

```
<!DOCTYPE html>
<html>
<body>

<h2>What Can JavaScript Do?</h2>
<p>JavaScript can change HTML attribute values.</p>
<p>In this case JavaScript changes the value of the src (source) attribute of an image.</p>

<button onclick="document.getElementById('myImage').src='pic_bulbon.gif'">Turn on the light</button>

<img id="myImage" src="pic_bulboff.gif" style="width:100px">

<button onclick="document.getElementById('myImage').src='pic_bulboff.gif'">Turn off the light</button>

</body>
</html>
```

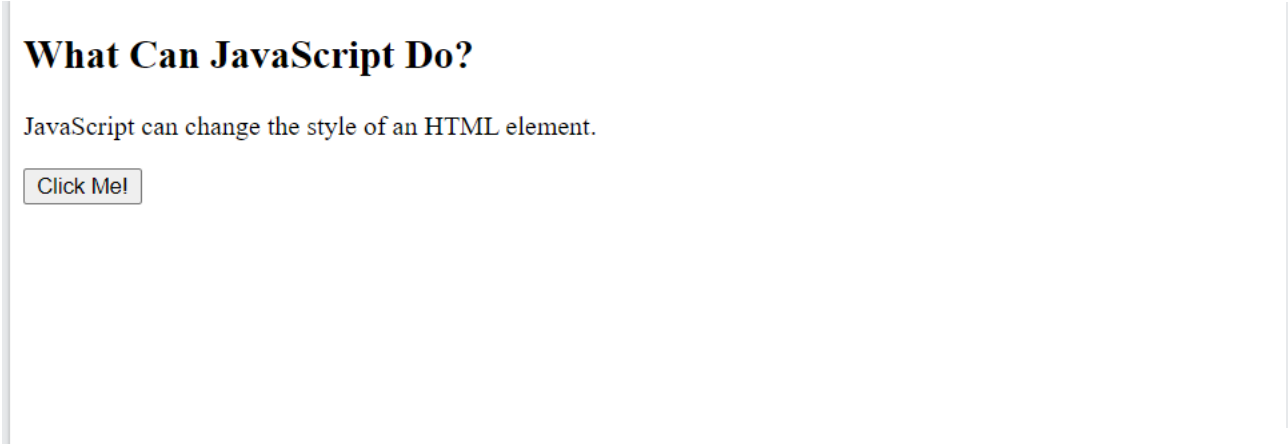


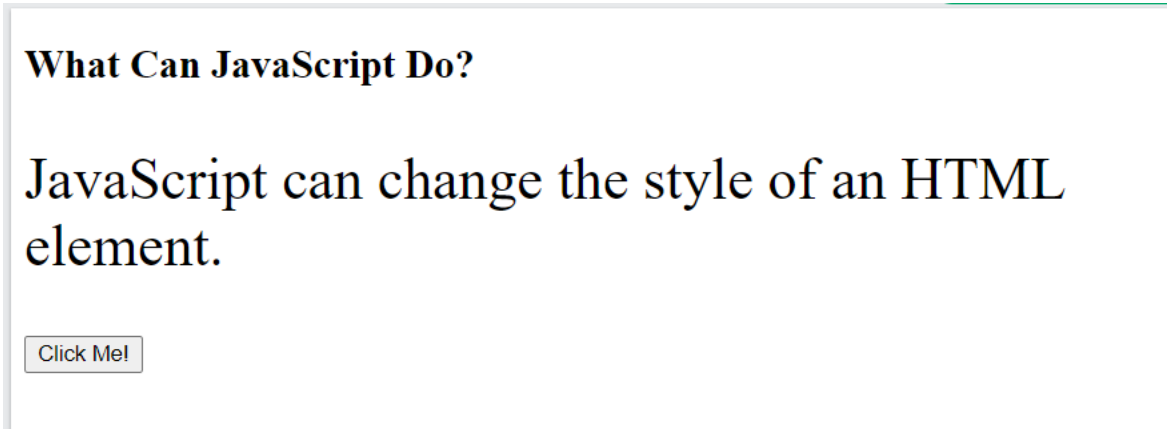


What JavaScript can do?

Change HTML Style (CSS)

```
<!DOCTYPE html>
<html>
<body>

<h2>What Can JavaScript Do?</h2>

<p id="demo">JavaScript can change the style of an HTML element.</p>

<button type="button" onclick="document.getElementById('demo').style.fontSize='35px'">Click Me!</button>

</body>
</html>
```

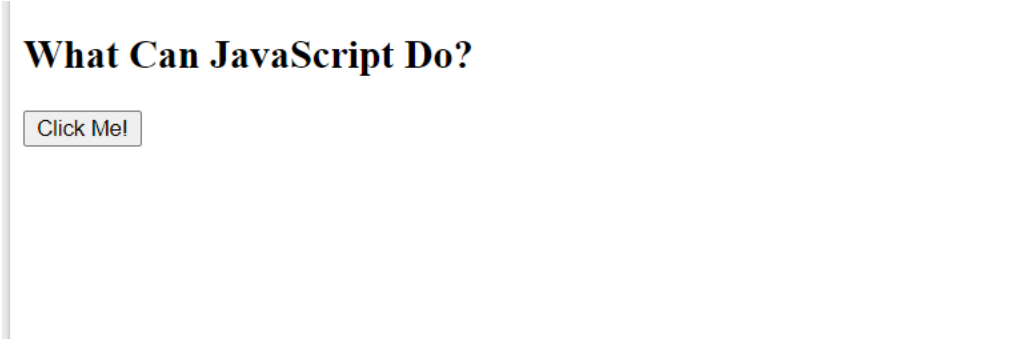




What JavaScript can do?

Hide HTML Elements

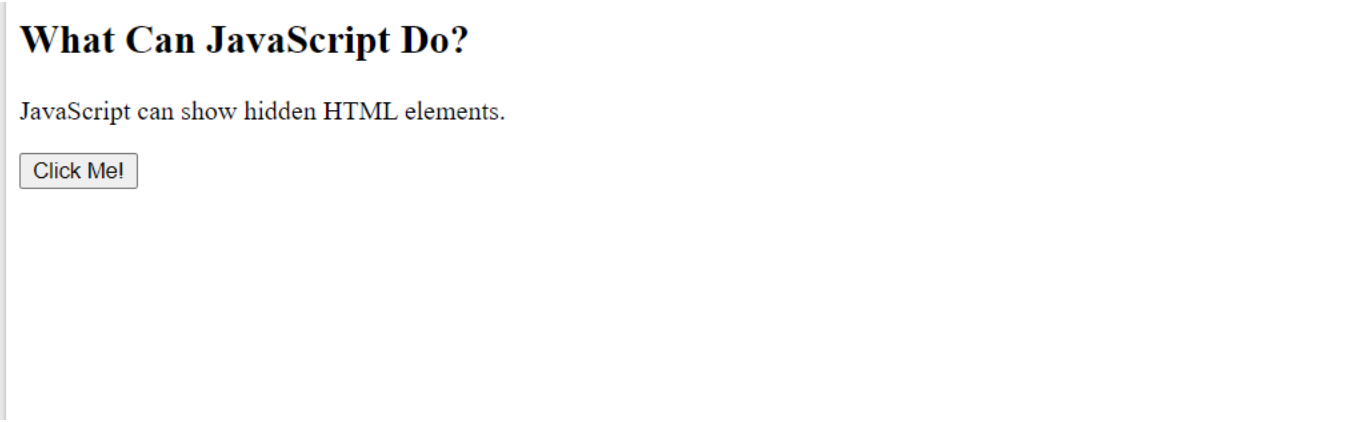
```
<!DOCTYPE html>
<html>
<body>

<h2>What Can JavaScript Do?</h2>

<p id="demo">JavaScript can hide HTML elements.</p>

<button type="button" onclick="document.getElementById('demo').style.display='none'">Click Me!</button>

</body>
</html>
```

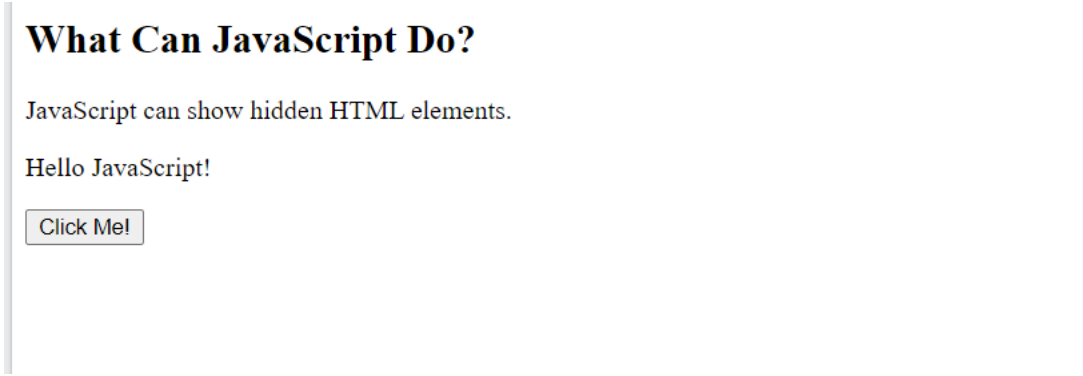


What JavaScript can do?

Show HTML Elements

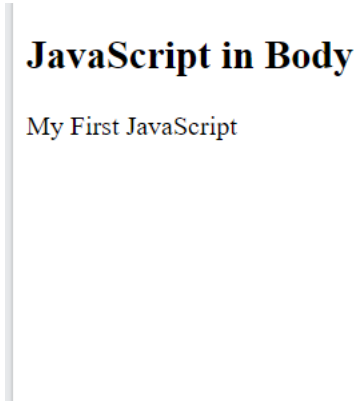
```
<!DOCTYPE html>
<html>
<body>

<h2>What Can JavaScript Do?</h2>

<p>JavaScript can show hidden HTML elements.</p>

<p id="demo" style="display:none">Hello JavaScript!</p>

<button type="button" onclick="document.getElementById('demo').style.display='block'">Click Me!</button>

</body>
</html>
```





Where to write JavaScript code?

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript in Body</h2>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = "My First JavaScript";
</script>

</body>
</html>
```

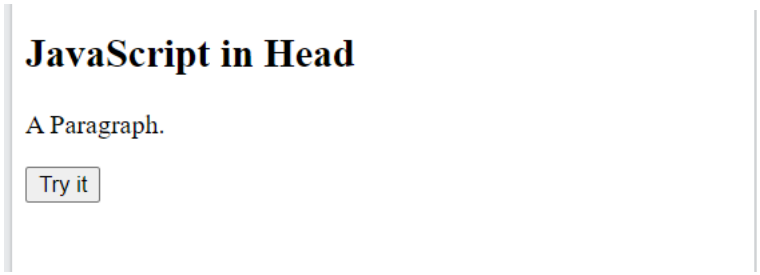


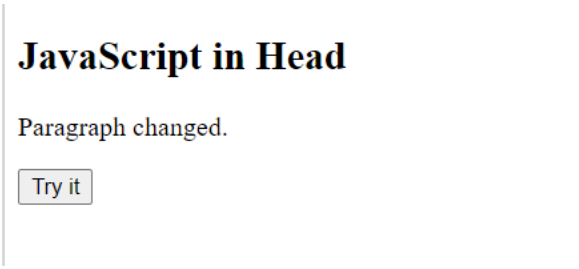
The <script> Tag

- In HTML, JavaScript code is inserted between <script> and </script> tags.
- We can place any number of scripts in an HTML document.
- Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both.

Where to write JavaScript code?

```
<!DOCTYPE html>
<html>
<head>

<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>

</head>
<body>

<h2>JavaScript in Head</h2>

<p id="demo">A Paragraph.</p>

<button type="button" onclick="myFunction()">Try it</button>

</body>
</html>
```





A JavaScript function

- It is a block of JavaScript code, that can be executed when "called" for.
  - Example: a function can be called when an event occurs, like when the user clicks a button.

Script is written in the head

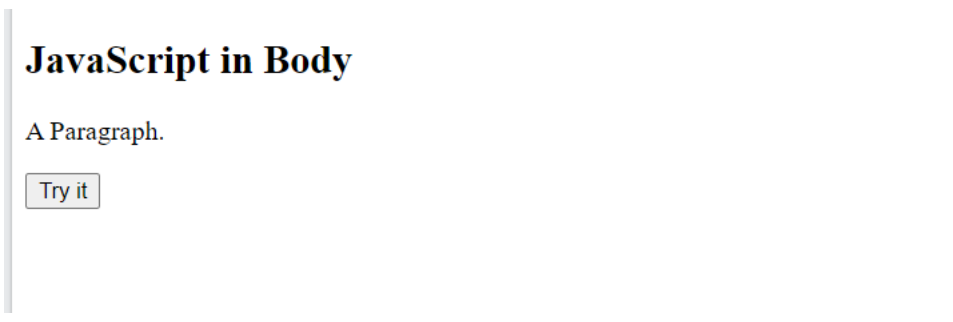
Where to write JavaScript code?

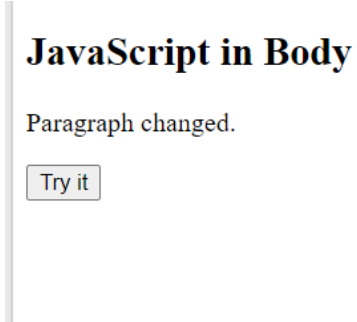
```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript in Body</h2>

<p id="demo">A Paragraph.</p>

<button type="button" onclick="myFunction()">Try it</button>

<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>

</body>
</html>
```

A JavaScript function

- It is a block of JavaScript code, that can be executed when "called" for.
  - Example: a function can be called when an event occurs, like when the user clicks a button.

Script is written in the body





Where to write JavaScript code?

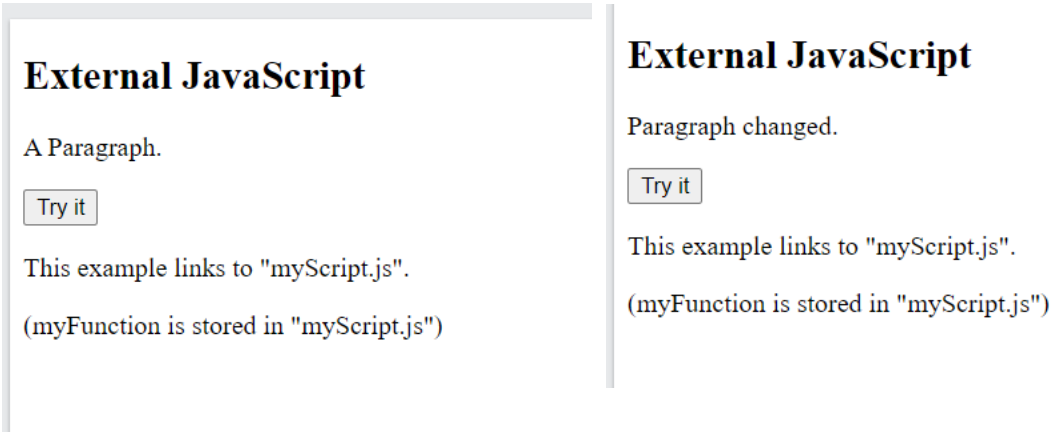
```
<!DOCTYPE html>
<html>
<body>

<h2>External JavaScript</h2>

<p id="demo">A Paragraph.</p>

<button type="button" onclick="myFunction()">Try it</button>

<p>This example links to "myScript.js".</p>
<p>(myFunction is stored in "myScript.js")</p>

<script src="myScript.js"></script>

</body>
</html>
```

```
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
```

myScript.js

External JavaScript

- Scripts can also be placed in external files
- External scripts are practical when the same code is used in many different web pages.
- JavaScript files have the file extension .js.
- To use an external script, put the name of the script file in the src (source) attribute of a <script> tag
- You can place an external script reference in <head> or <body> as you like.
- The script will behave as if it was located exactly where the <script> tag is located.
- External scripts cannot contain <script> tags.



Where to write JavaScript code?

External JavaScript

Placing scripts in external files has some advantages:

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

To add several script files to one page - use several script tags:

```
<script src="myScript1.js"></script>
<script src="myScript2.js"></script>
```

Where to write JavaScript code?

External References

An external script can be referenced in 3 different ways:

- With a full URL (a full web address)
  - Example: `<script src="https://www.w3schools.com/js/myScript.js"></script>`
- With a file path (like /js/)
  - Example: `<script src="/js/myScript.js"></script>`
- Without any path
  - Example: `<script src="myScript.js"></script>`

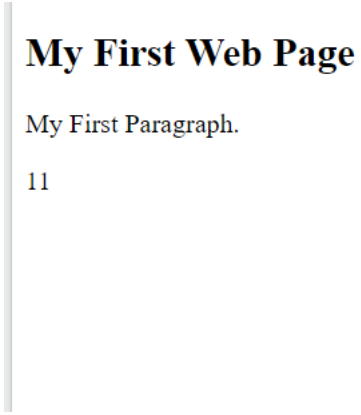
JavaScript Output

JavaScript can "display" data in different ways:

1. Writing into an HTML element, using innerHTML.
2. Writing into the HTML output using document.write().
3. Writing into an alert box, using window.alert().
4. Writing into the browser console, using console.log().

JavaScript Output

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My First Paragraph</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>

</body>
</html>
```

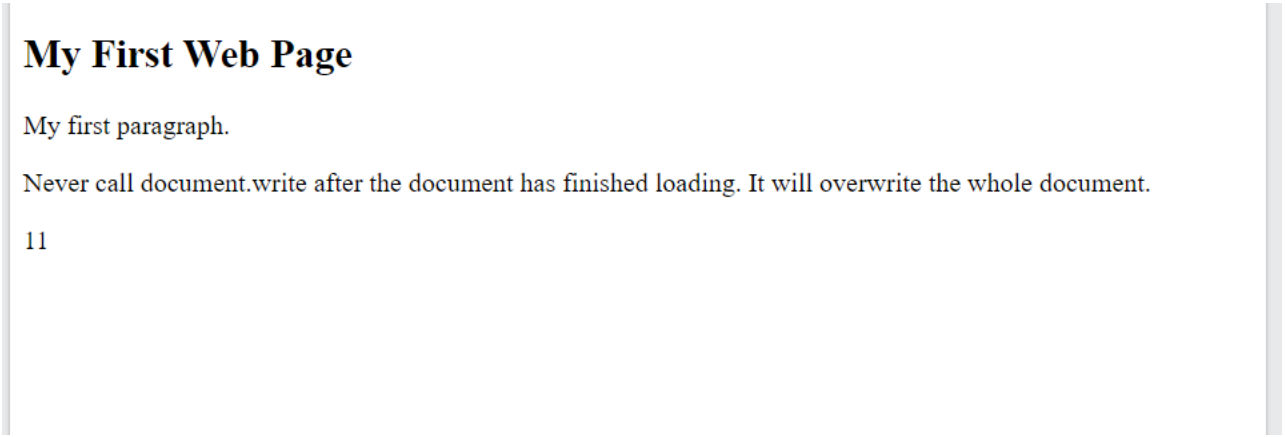


innerHTML

- To access an HTML element, JavaScript can use the document.getElementById(id) method.
- The id attribute defines the HTML element. The innerHTML property defines the HTML content:

JavaScript Output

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
document.write(5 + 6);
</script>

</body>
</html>
```



Document.write()

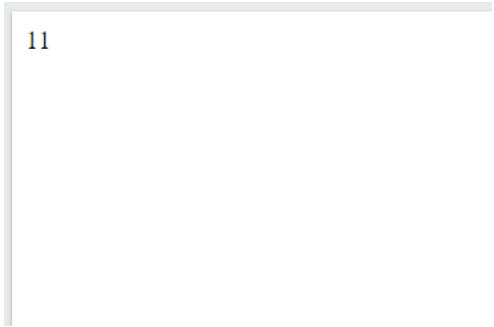
- For testing purposes, it is convenient to use document.write()
- Using document.write() after an HTML document is loaded, will delete all existing HTML

JavaScript Output

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<button type="button" onclick="document.write(5 + 6)">Try
it</button>

</body>
</html>
```

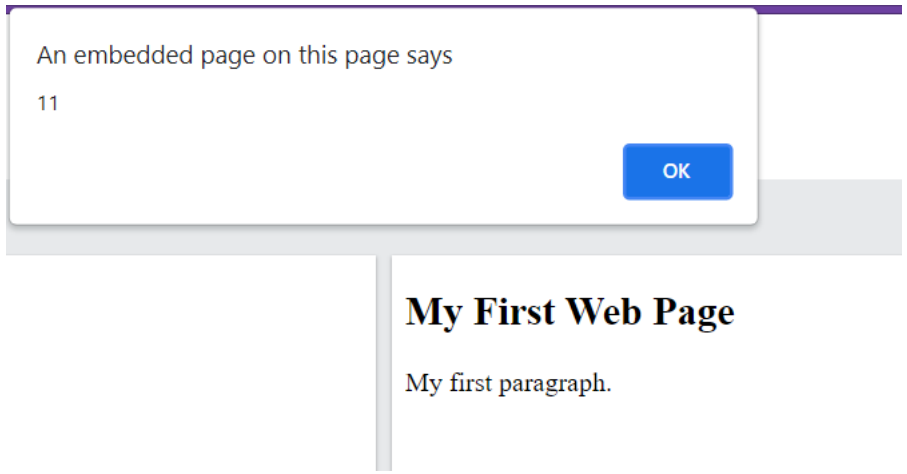
Document.write()

- For testing purposes, it is convenient to use document.write()
- Using document.write() after an HTML document is loaded, will delete all existing HTML



JavaScript Output

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
window.alert(5 + 6);
</script>

</body>
</html>
```



windows.alert()

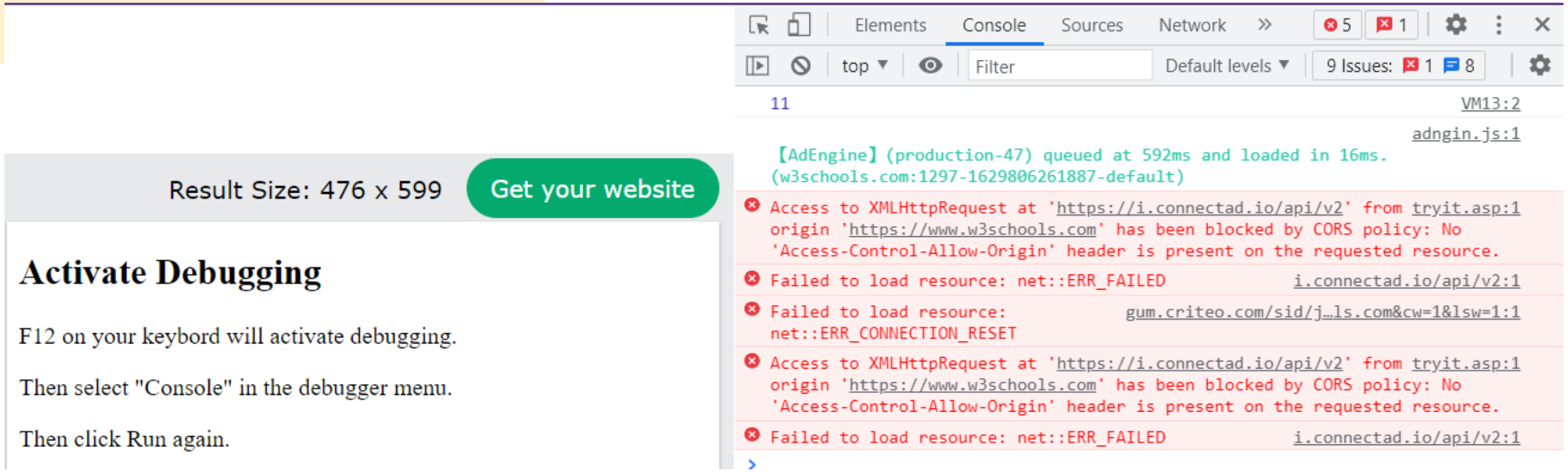
- We can use an alert box to display data
- We can skip the window keyword.
- In JavaScript, the window object is the global scope object,
- It means that variables, properties, and methods by default belong to the window object.
- This also means that specifying the window keyword is optional

JavaScript Output

```
<!DOCTYPE html>
<html>
<body>

<h2>Activate Debugging</h2>

<p>F12 on your keyboard will activate debugging.</p>
<p>Then select "Console" in the debugger menu.</p>
<p>Then click Run again.</p>

<script>
console.log(5 + 6);
</script>

</body>
</html>
```

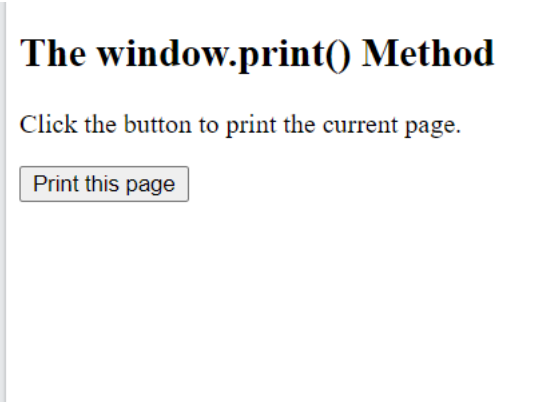
Console.log()

- For debugging purposes, we can call the console.log() method in the browser to display data



JavaScript Output

```
<!DOCTYPE html>
<html>
<body>

<h2>The window.print() Method</h2>

<p>Click the button to print the current page.</p>

<button onclick="window.print()">Print this page</button>

</body>
</html>
```

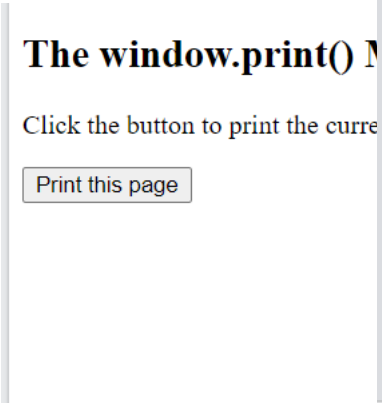


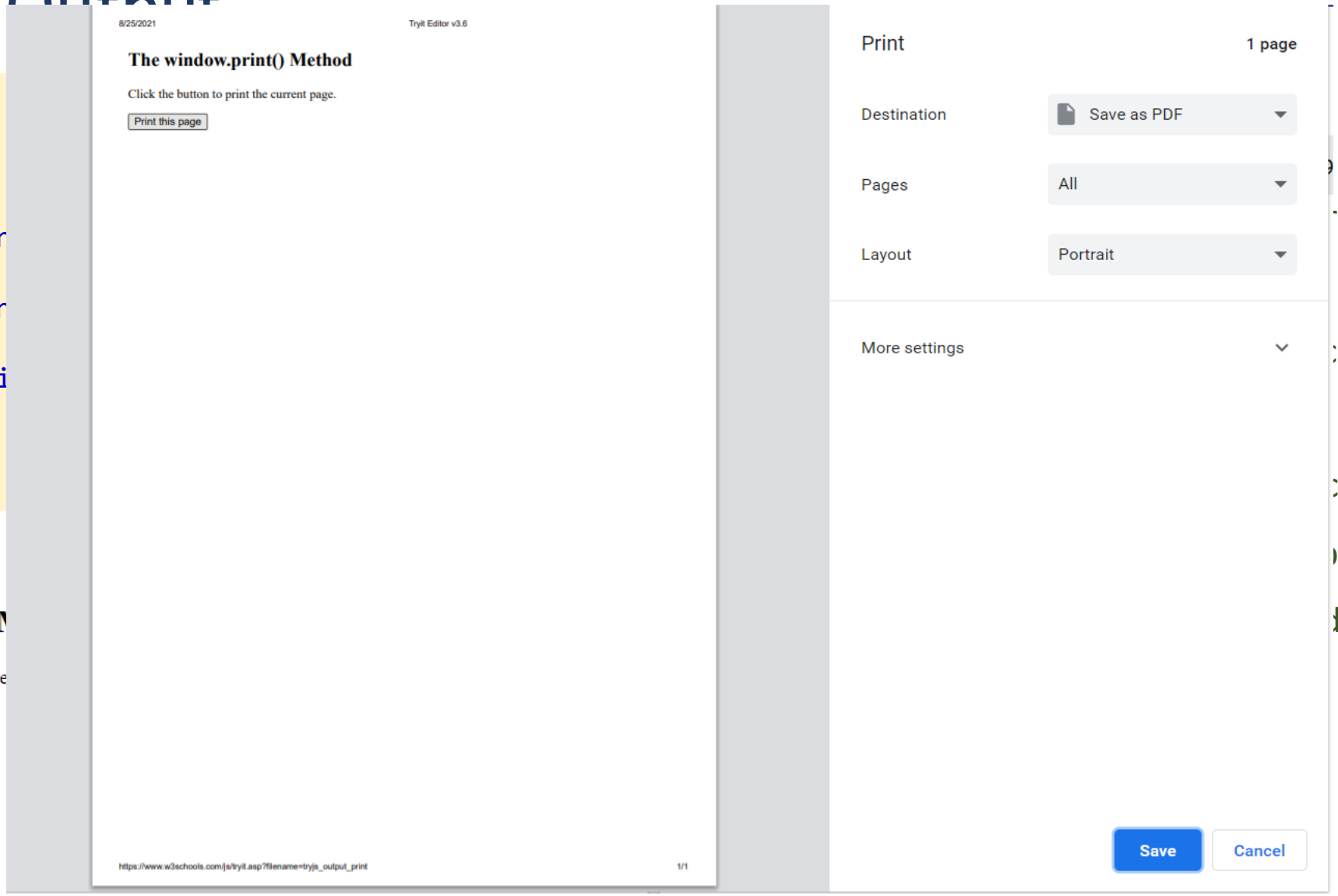
windows.print()

- JavaScript does not have any print object or print methods.
- You cannot access output devices from JavaScript.
- The only exception is that you can call the window.print() method in the browser to print the content of the current window.

JavaScript Output

```
<!DOCTYPE html>
<html>
<body>

<h2>The window.prir

<p>Click the buttor

<button onclick="wi

</body>
</html>
```





int object or print

es from

an call the

rowser to print

low.

Java Script

Module 2

Content

- JavaScript Errors
- JavaScript Exception Handling
- Document Object Model

JavaScript Errors

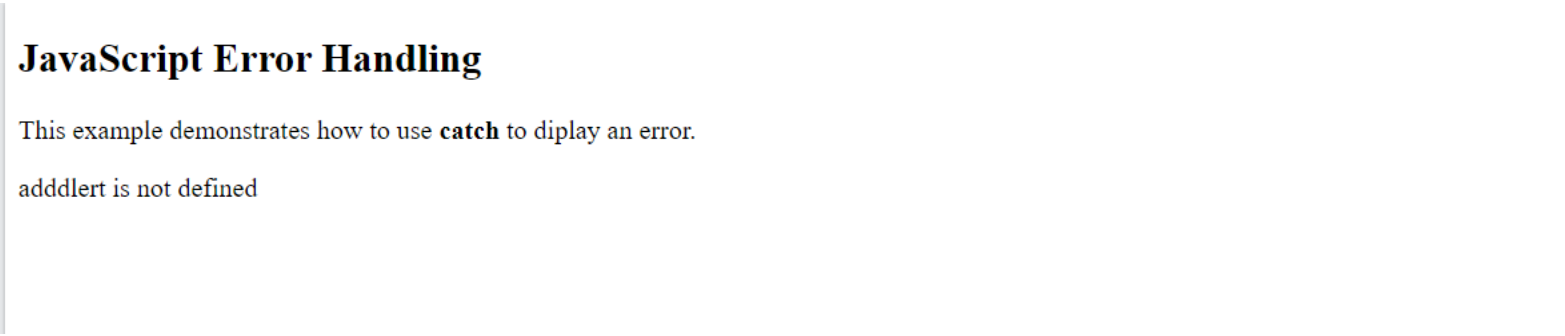
- The try statement lets you test a block of code for errors.
- The catch statement lets you handle the error.
- The throw statement lets you create custom errors.
- The finally statement lets you execute code, after try and catch, regardless of the result.

JavaScript Errors

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Error Handling</h2>

<p>This example demonstrates how to use <b>catch</b> to diplay an error.</p>

<p id="demo"></p>

<script>
try {
  adddlert("Welcome guest!");
}
catch(err) {
  document.getElementById("demo").innerHTML = err.message;
}
</script>

</body>
</html>
```

- When executing JavaScript code, different errors can occur.
- Errors can be coding errors made by the programmer, errors due to wrong input, and other unforeseeable things.
- Example

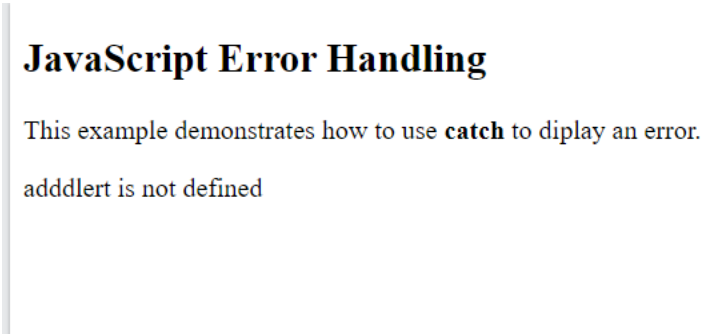
In this example we misspelled "alert" as "adddlert" to deliberately produce an error



JavaScript catches adddlert as an error, and executes the catch code to handle it.

JavaScript try and catch

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Error Handling</h2>

<p>This example demonstrates how to use <b>catch</b> to diplay an error.</p>

<p id="demo"></p>

<script>
try {
  adddlert("Welcome guest!");
}
catch(err) {
  document.getElementById("demo").innerHTML = err.message;
}
</script>

</body>
</html>
```



- The try statement allows you to define a block of code to be tested for errors while it is being executed.
- The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.
- The JavaScript statements try and catch come in pairs

```
try {
  Block of code to try
}
catch(err) {
  Block of code to handle errors
}
```

JavaScript Throws Errors

- When an error occurs, JavaScript will normally stop and generate an error message.
- The technical term for this is: **JavaScript will throw an exception (throw an error).**
- JavaScript will actually create an Error object with two properties:
  - name
  - message.

The throw Statement

- The throw statement allows you to create a custom error.
- Technically you can throw an exception (throw an error).
- The exception can be a JavaScript String, a Number, a Boolean or an Object:
- throw "Too big"; // throw a text
- throw 500; // throw a number
- If you use throw together with try and catch, you can control program flow and generate custom error messages.

The throw Statement

- The throw statement allows you to create a custom error.
- Technically you can throw an exception (throw an error).
- The exception can be a JavaScript String, a Number, a Boolean or an Object:

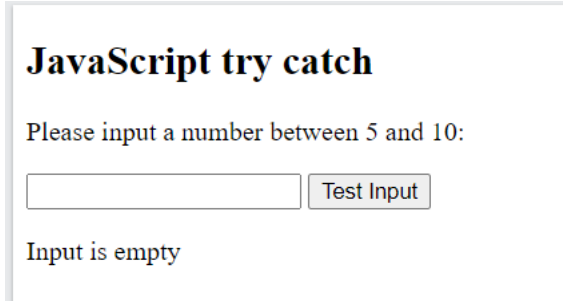
```
throw "Too big";    // throw a text

throw 500;          // throw a number
```

- If you use throw together with try and catch, you can control program flow and generate custom error messages.

Input Validation

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript try catch</h2>

<p>Please input a number between 5 and 10:</p>

<input id="demo" type="text">
<button type="button" onclick="myFunction()">Test Input</button>
<p id="p01"></p>

<script>
function myFunction() {
  const message = document.getElementById("p01");
  message.innerHTML = "";
  let x = document.getElementById("demo").value;
  try {
    if(x == "")  throw "empty";
    if(isNaN(x)) throw "not a number";
    x = Number(x);
    if(x < 5)  throw "too low";
    if(x > 10)   throw "too high";
  }
  catch(err) {
    message.innerHTML = "Input is " + err;
  }
}
</script>

</body>
</html>
```

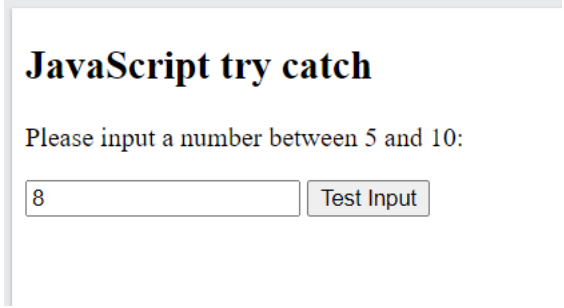
- This example examines input. If the value is wrong, an exception (err) is thrown.
- The exception (err) is caught by the catch statement and a custom error message is displayed:

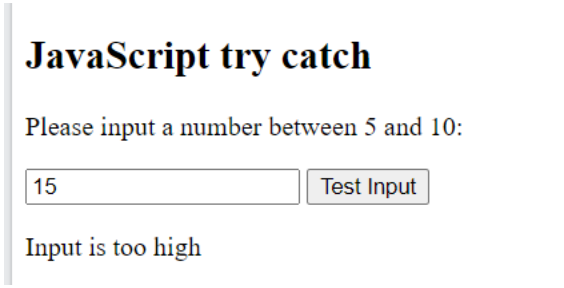


Input Validation

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript try catch</h2>

<p>Please input a number between 5 and 10:</p>

<input id="demo" type="text">
<button type="button" onclick="myFunction()">Test Input</button>
<p id="p01"></p>

<script>
function myFunction() {
  const message = document.getElementById("p01");
  message.innerHTML = "";
  let x = document.getElementById("demo").value;
  try {
    if(x == "")  throw "empty";
    if(isNaN(x)) throw "not a number";
    x = Number(x);
    if(x < 5)  throw "too low";
    if(x > 10)   throw "too high";
  }
  catch(err) {
    message.innerHTML = "Input is " + err;
  }
}
</script>

</body>
</html>
```

- This example examines input. If the value is wrong, an exception (err) is thrown.
- The exception (err) is caught by the catch statement and a custom error message is displayed:





```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript try catch</h2>

<p>Please input a number between 5 and 10:</p>

<input id="demo" type="text">
<button type="button" onclick="myFunction()">Test Input</button>

<p id="p01"></p>

<script>
function myFunction() {
  const message = document.getElementById("p01");
  message.innerHTML = "";
  let x = document.getElementById("demo").value;
  try {
    if(x == "")  throw "is empty";
    if(isNaN(x)) throw "is not a number";
    x = Number(x);
    if(x > 10)   throw "is too high";
    if(x < 5)  throw "is too low";
  }
  catch(err) {
    message.innerHTML = "Input " + err;
  }
  finally {
    document.getElementById("demo").value = "";
  }
}
</script>

</body>
</html>
```

finally statement

- The finally statement lets you execute code, after try and catch, regardless of the result:

Syntax

```
try {
  Block of code to try
}
catch(err) {
  Block of code to handle errors
}
finally {
  Block of code to be executed regardless of the try / catch result
}
```

finally statement

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript try catch</h2>

<p>Please input a number between 5 and 10:</p>

<input id="demo" type="text">
<button type="button" onclick="myFunction()">Test Input</button>

<p id="p01"></p>

<script>
function myFunction() {
  const message = document.getElementById("p01");
  message.innerHTML = "";
  let x = document.getElementById("demo").value;
  try {
    if(x == "")  throw "is empty";
    if(isNaN(x)) throw "is not a number";
    x = Number(x);
    if(x > 10)   throw "is too high";
    if(x < 5)  throw "is too low";
  }
  catch(err) {
    message.innerHTML = "Input " + err;
  }
  finally {
    document.getElementById("demo").value = "";
  }
}
</script>

</body>
</html>
```

The Error Object

- JavaScript has a built in error object that provides error information when an error occurs.
- The error object provides two useful properties: name and message.

- Error Object Properties

Property	Description
name	Sets or returns an error name
message	Sets or returns an error message (a string)
Error	Name Values

The Error Object

Error Name Values

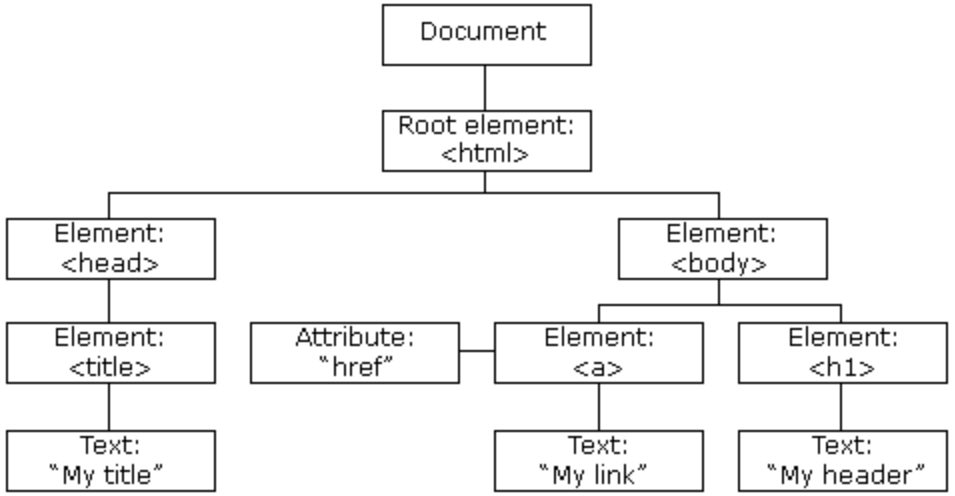
Six different values can be returned by the error name property:

Error Name	Description
EvalError	An error has occurred in the eval() function
RangeError	A number "out of range" has occurred
ReferenceError	An illegal reference has occurred
SyntaxError	A syntax error has occurred
TypeError	A type error has occurred
URIError	An error in encodeURI() has occurred

DOM (Document Object Model)

DOM (Document Object Model)

- With the HTML DOM, JavaScript can access and change all the elements of an HTML document.
- When a web page is loaded, the browser creates a Document Object Model of the page.
- The HTML DOM model is constructed as a tree of Objects:



DOM (Document Object Model)

With the object model, JavaScript gets all the power it needs to create dynamic HTML:

JavaScript can

- Change all the HTML elements in the page
- Change all the HTML attributes in the page
- Change all the CSS styles in the page
- Remove existing HTML elements and attributes
- Add new HTML elements and attributes
- Can react to all existing HTML events in the page
- Create new HTML events in the page

What is DOM (Document Object Model)

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

The DOM defines a standard for accessing documents:

"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."

The W3C DOM standard is separated into 3 different parts:

1. Core DOM - standard model for all document types
2. XML DOM - standard model for XML documents
3. HTML DOM - standard model for HTML documents

What is HTML DOM?

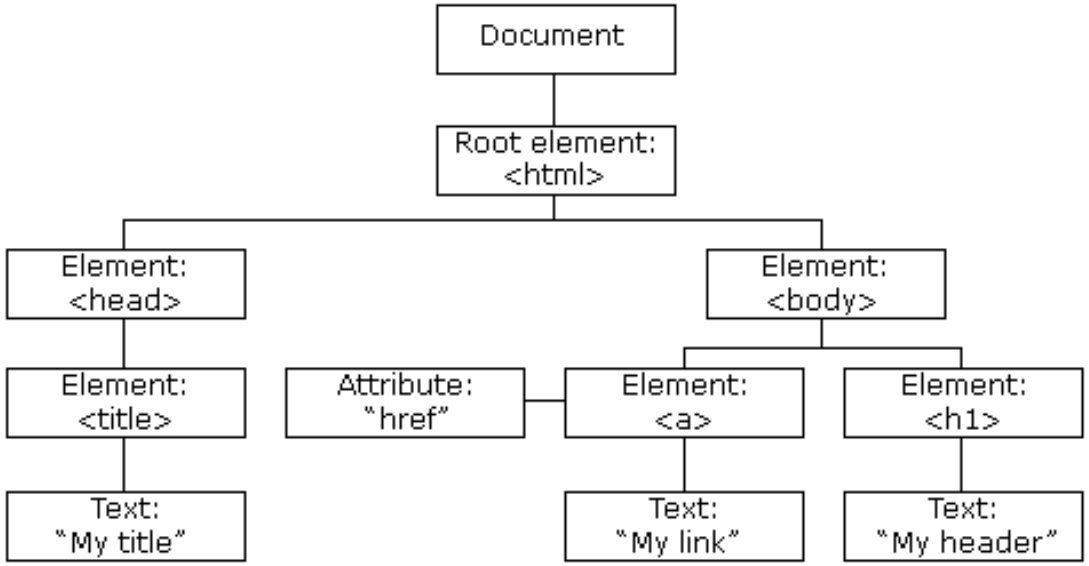
The HTML DOM is a standard object model and programming interface for HTML.

It defines:

- The HTML elements as objects
- The properties of all HTML elements
- The methods to access all HTML elements
- The events for all HTML elements

In other words: **The HTML DOM is a standard for how to get, change, add, or delete HTML elements.**

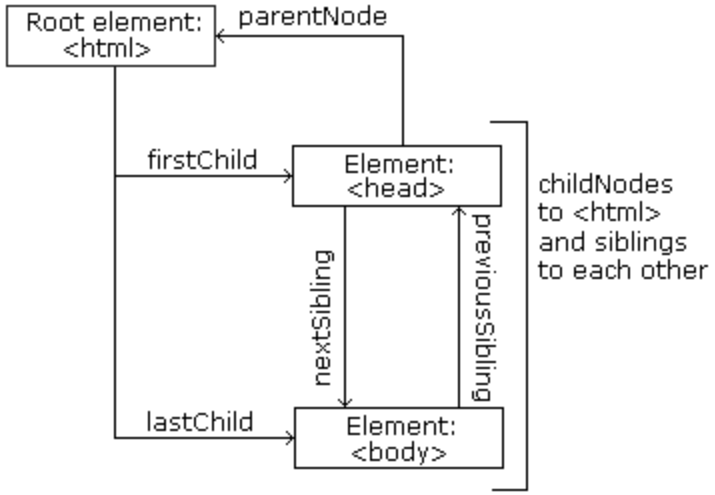
DOM Nodes



- According to the W3C HTML DOM standard, everything in an HTML document is a node:
  - The entire document is a document node
  - Every HTML element is an element node
  - The text inside HTML elements are text nodes
  - Every HTML attribute is an attribute node (deprecated)
  - All comments are comment nodes
- With the HTML DOM, all nodes in the node tree can be accessed by JavaScript.
- New nodes can be created, and all nodes can be modified or deleted.

Node Relationships

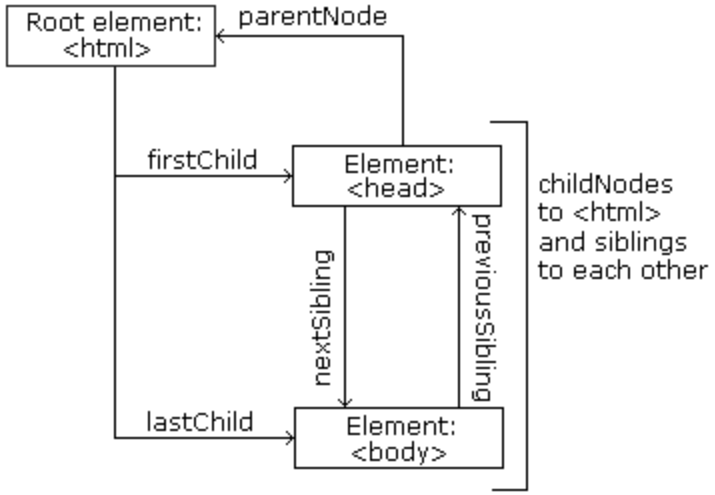
```
<html>

  <head>
    <title>DOM Tutorial</title>
  </head>

  <body>
    <h1>DOM Lesson one</h1>
    <p>Hello world!</p>
  </body>

</html>
```



- The nodes in the node tree have a hierarchical relationship to each other.
- The terms parent, child, and sibling are used to describe the relationships.
- In a node tree, the top node is called the root (or root node)
- Every node has exactly one parent, except the root (which has no parent)
- A node can have a number of children
- Siblings (brothers or sisters) are nodes with the same parent

Node Relationships

```
<html>

  <head>
    <title>DOM Tutorial</title>
  </head>

  <body>
    <h1>DOM Lesson one</h1>
    <p>Hello world!</p>
  </body>

</html>
```



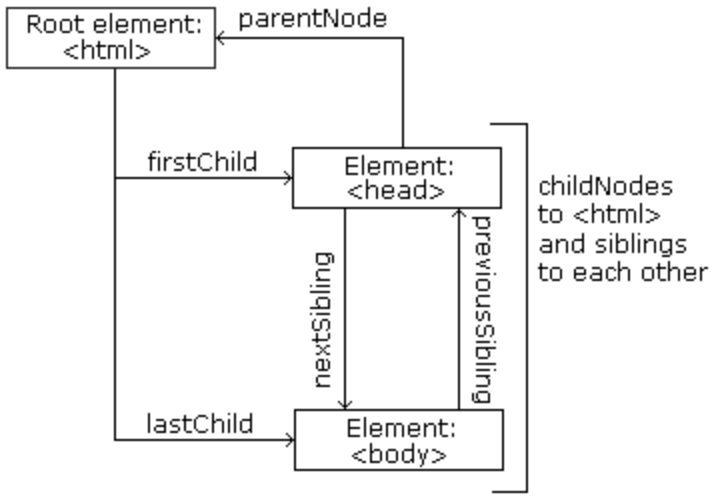
From the HTML given we can read

- <html> is the root node
- <html> has no parents
- <html> is the parent of <head> and <body>
- <head> is the first child of <html>
- <body> is the last child of <html>

- <head> has one child: <title>
- <title> has one child (a text node): "DOM Tutorial"
- <body> has two children: <h1> and <p>
- <h1> has one child: "DOM Lesson one"
- <p> has one child: "Hello world!"
- <h1> and <p> are siblings

Navigation between Nodes

```
<html>

  <head>
    <title>DOM Tutorial</title>
  </head>

  <body>
    <h1>DOM Lesson one</h1>
    <p>Hello world!</p>
  </body>

</html>
```



We can use the following node properties to navigate between nodes with JavaScript:

- parentNode
- childNodes[nodenumber]
- firstChild
- lastChild
- nextSibling
- previousSibling

Form Validation

```
<!DOCTYPE html>
<html>
<head>

<script>
function validateForm() {
  let x = document.forms["myForm"]["fname"].value;
  if (x == "") {
    alert("Name must be filled out");
    return false;
  }
}
</script>

</head>
<body>

<h2>JavaScript Validation</h2>

<form name="myForm" action="/action_page.php" onsubmit="return validateForm()" method="post">
  Name: <input type="text" name="fname">
  <input type="submit" value="Submit">
</form>

</body>
</html>
```

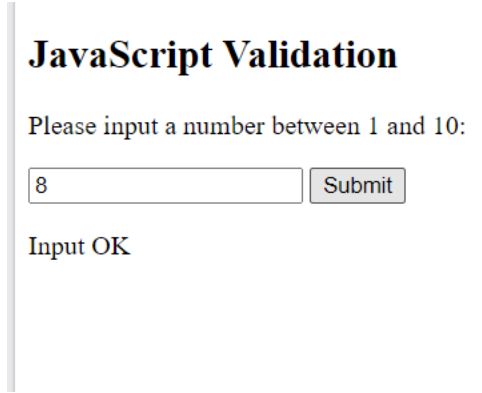
Form Validation

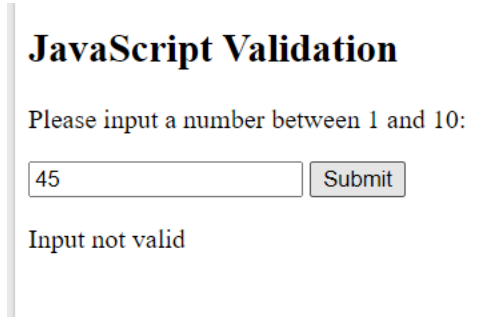
- HTML form validation can be done by JavaScript.
- If a form field (fname) is empty, this function alerts a message, and returns false, to prevent the form from being submitted
- The function can be called when the form is submitted

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Validation</h2>

<p>Please input a number between 1 and 10:</p>

<input id="numb">

<button type="button" onclick="myFunction()">Submit</button>

<p id="demo"></p>

<script>
function myFunction() {
  // Get the value of the input field with id="numb"
  let x = document.getElementById("numb").value;
  // If x is Not a Number or less than one or greater than 10
  let text;
  if (isNaN(x) || x < 1 || x > 10) {
    text = "Input not valid";
  } else {
    text = "Input OK";
  }
  document.getElementById("demo").innerHTML = text;
}
</script>

</body>
</html>
```

Form Validation

- Validation of Numeric Inputs

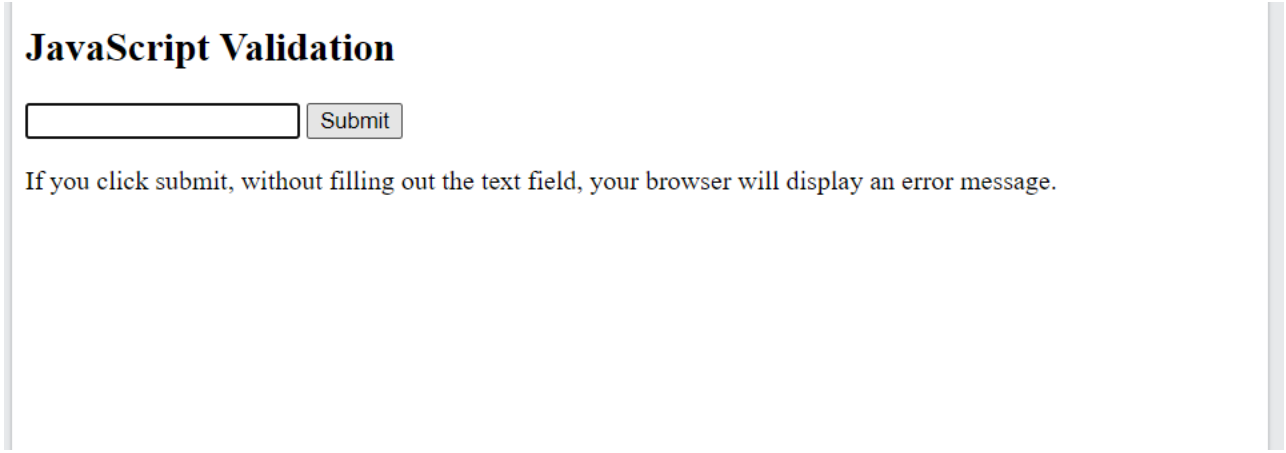


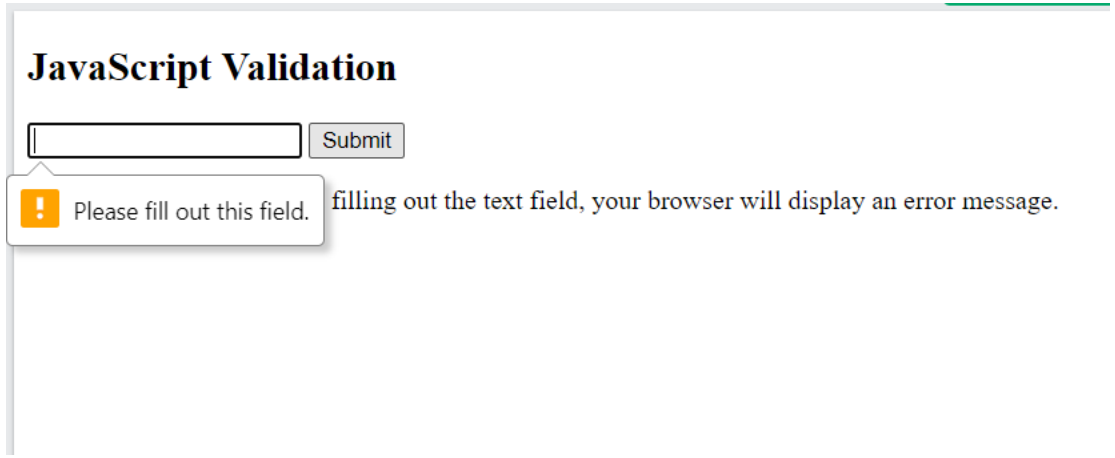


Automatic HTML Form Validation

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Validation</h2>

<form action="/action_page.php" method="post">
  <input type="text" name="fname" required>
  <input type="submit" value="Submit">
</form>

<p>If you click submit, without filling out the text field,
your browser will display an error message.</p>

</body>
</html>
```

- HTML form validation can be performed automatically by the browser:
- If a form field (fname) is empty, the required attribute prevents this form from being submitted





Data Validation

- Data validation is the process of ensuring that user input is clean, correct, and useful.
- Typical validation tasks are:
  - has the user filled in all required fields?
  - has the user entered a valid date?
  - has the user entered text in a numeric field?
- Most often, the purpose of data validation is to ensure correct user input.
- Validation can be defined by many different methods, and deployed in many different ways.
- **Server side validation:** is performed by a web server, after input has been sent to the server.
- **Client side validation:** is performed by a web browser, before input is sent to a web server.

Constraint Validation

- HTML5 introduced a new HTML validation concept called constraint validation.
- HTML constraint validation is based on:
  - Constraint validation HTML Input Attributes
  - Constraint validation CSS Pseudo Selectors
  - Constraint validation DOM Properties and Methods

Constraint Validation HTML Input Attributes

Attribute	Description
disabled	Specifies that the input element should be disabled
max	Specifies the maximum value of an input element
min	Specifies the minimum value of an input element
pattern	Specifies the value pattern of an input element
required	Specifies that the input field requires an element
type	Specifies the type of an input element

Constraint Validation CSS Pseudo Selectors

Selector	Description
:disabled	Selects input elements with the "disabled" attribute specified
:invalid	Selects input elements with invalid values
:optional	Selects input elements with no "required" attribute specified
:required	Selects input elements with the "required" attribute specified
:valid	Selects input elements with valid values

DOM Events

JavaScript HTML DOM Events

HTML DOM allows JavaScript to react to HTML events

A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.

To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute:

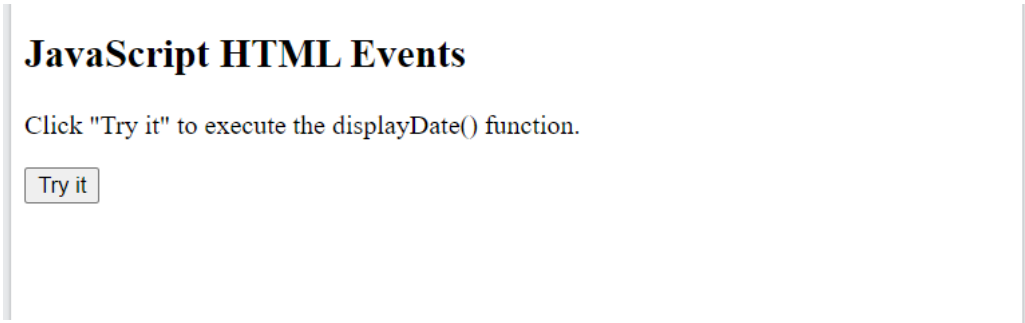
onclick=JavaScript

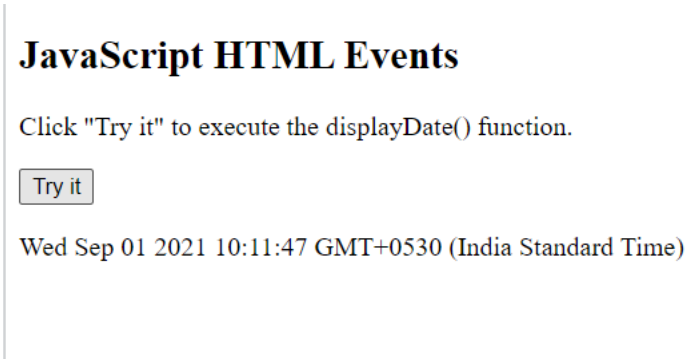
Examples of HTML events:

- When a user clicks the mouse
- When a web page has loaded
- When an image has been loaded
- When the mouse moves over an element
- When an input field is changed
- When an HTML form is submitted
- When a user strokes a key

Assign events using HTML DOM

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML Events</h2>
<p>Click "Try it" to execute the displayDate() function.</p>

<button id="myBtn">Try it</button>

<p id="demo"></p>

<script>
document.getElementById("myBtn").onclick = displayDate;

function displayDate() {
  document.getElementById("demo").innerHTML = Date();
}
</script>

</body>
</html>
```





Java Script

Module 2

Content

- DOM Events
- DOM Eventhandlers

DOM Events

JavaScript HTML DOM Events

HTML DOM allows JavaScript to react to HTML events

A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.

To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute:

onclick=JavaScript

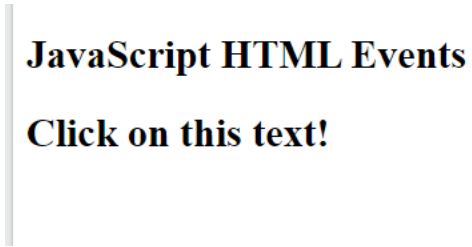
Examples of HTML events:

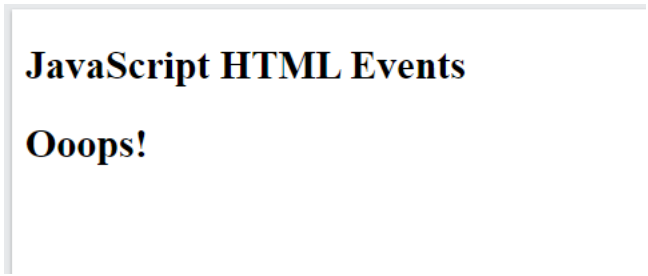
- When a user clicks the mouse
- When a web page has loaded
- When an image has been loaded
- When the mouse moves over an element
- When an input field is changed
- When an HTML form is submitted
- When a user strokes a key

Changing Element with Click

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML Events</h2>
<h2 onclick="this.innerHTML='Ooops!'">Click on this text!</h2>

</body>
</html>
```

The content of the <h1> element is changed when a user clicks on it



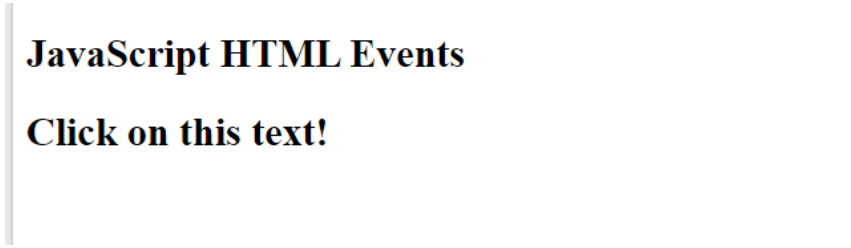


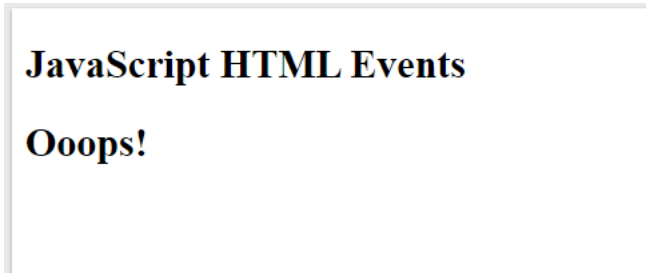
Changing Element with Click

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML Events</h2>
<h2 onclick="changeText(this)">Click on this text!</h2>

<script>
function changeText(id) {
  id.innerHTML = "Ooops!";
}
</script>

</body>
</html>
```

The content of the <h1> element is changed when a user clicks on it.

Here function is called from the event handler

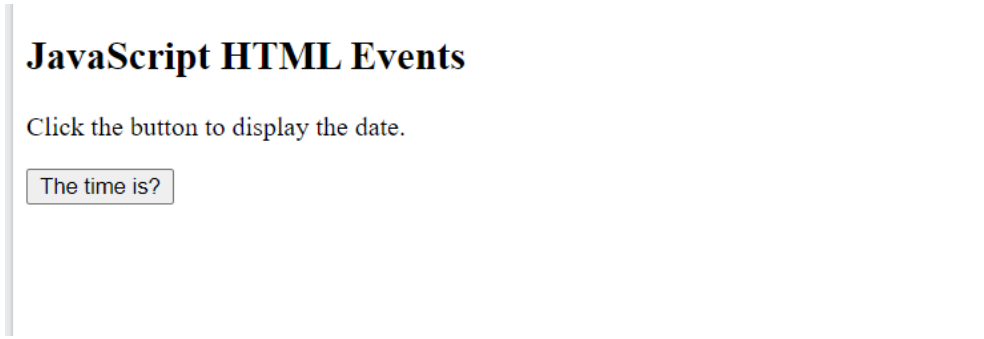


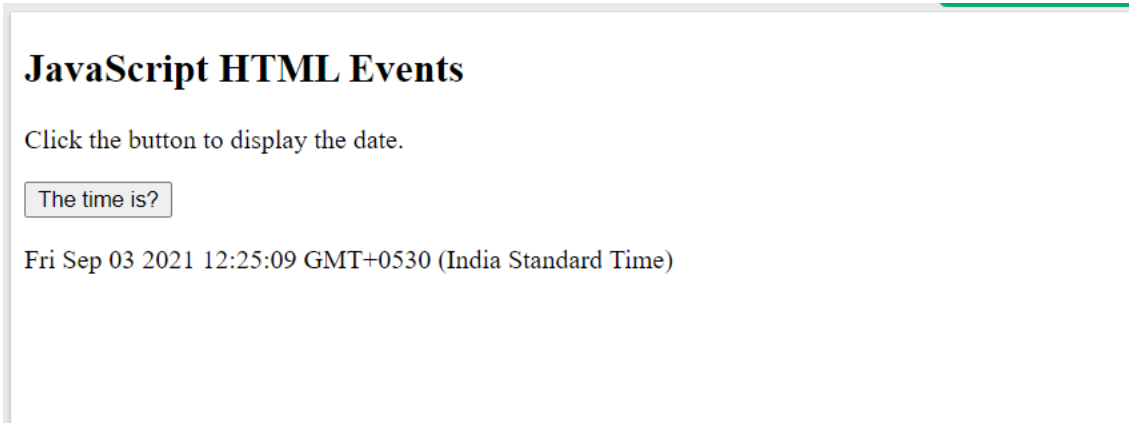


Assign events using Element Attributes

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML Events</h2>
<p>Click the button to display the date.</p>

<button onclick="displayDate()">The time is?</button>

<script>
function displayDate() {
  document.getElementById("demo").innerHTML = Date();
}
</script>

<p id="demo"></p>

</body>
</html>
```

- To assign events to HTML elements you can use event attributes.
- Assign an onclick event to a button element
- For this example, a function named displayDate() will be executed when the button is clicked.

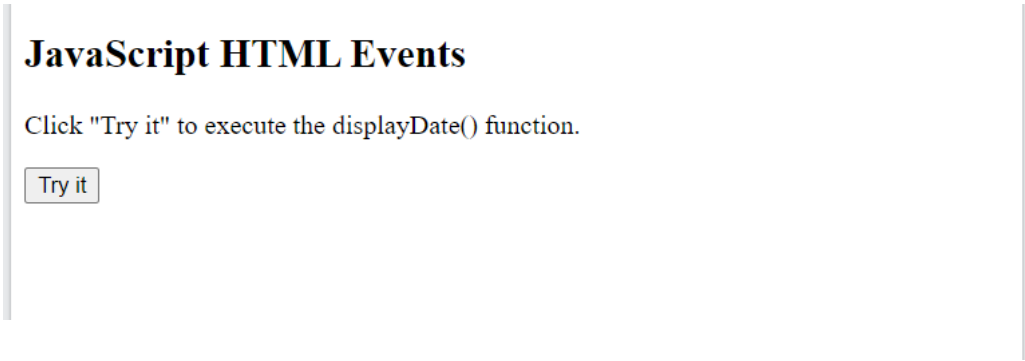




Assign events using HTML DOM

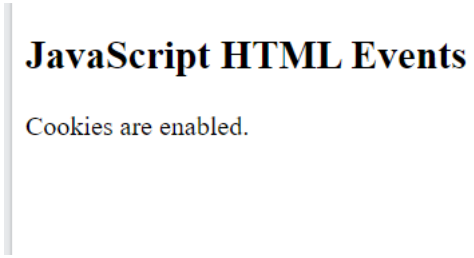
```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML Events</h2>
<p>Click "Try it" to execute the displayDate() function.</p>

<button id="myBtn">Try it</button>

<p id="demo"></p>

<script>
document.getElementById("myBtn").onclick = displayDate;

function displayDate() {
  document.getElementById("demo").innerHTML = Date();
}
</script>

</body>
</html>
```

- The HTML DOM allows you to assign events to HTML elements using JavaScript
- In the example above, a function named displayDate is assigned to an HTML element with the id="myBtn"



The onload and onunload Events

```
<!DOCTYPE html>
<html>
<body onload="checkCookies()">

<h2>JavaScript HTML Events</h2>

<p id="demo"></p>

<script>
function checkCookies() {
  var text = "";
  if (navigator.cookieEnabled == true) {
    text = "Cookies are enabled.";
  } else {
    text = "Cookies are not enabled.";
  }
  document.getElementById("demo").innerHTML = text;
}
</script>

</body>
</html>
```



- The onload and onunload events are triggered when the user enters or leaves the page.

The onload event:

- Used to check the visitor's browser type and browser version
- load the proper version of the web page based on the information.

- The onload and onunload events can be used to deal with cookies.

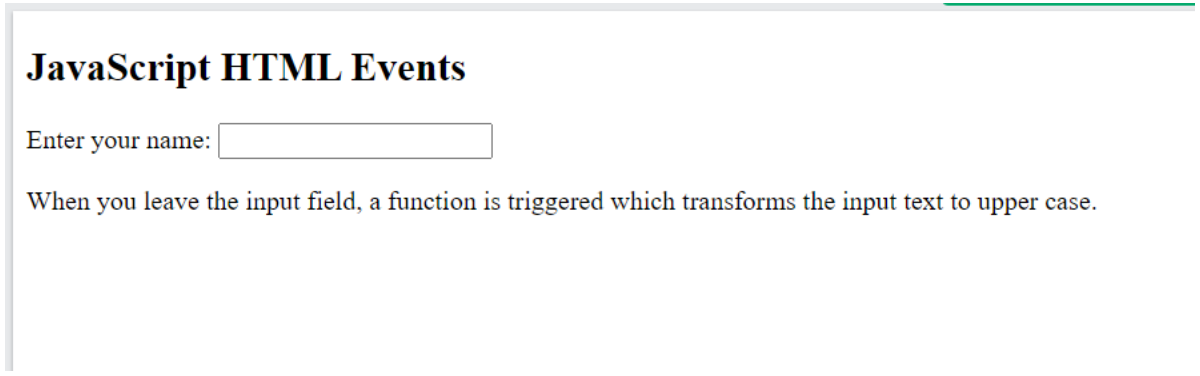
The onchange Events

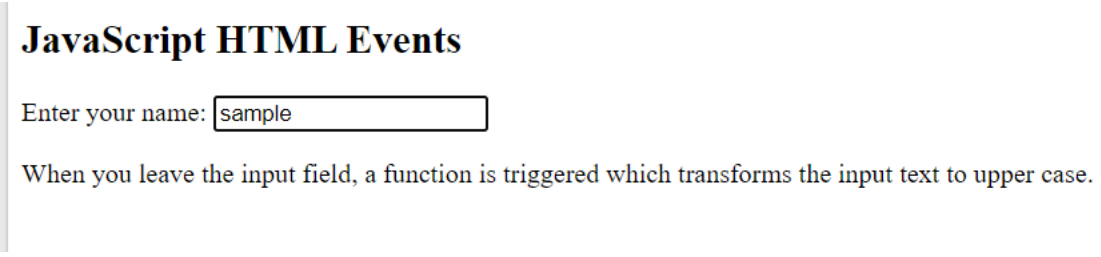
- The onchange event is often used in combination with validation of input fields.

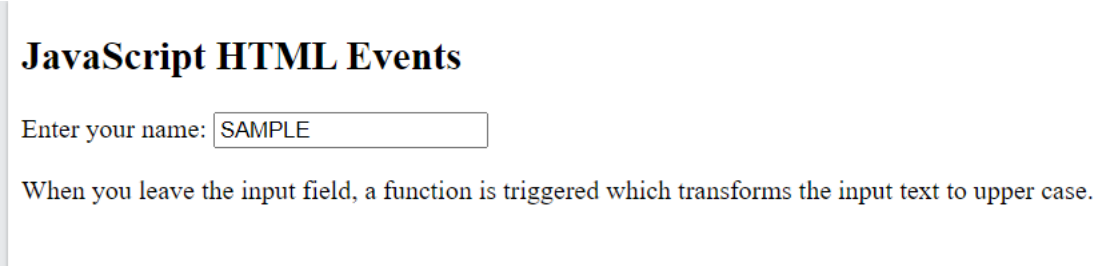
```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML Events</h2>
Enter your name: <input type="text" id="fname" onchange="upperCase()">
<p>When you leave the input field, a function is triggered which transforms the input text to upper case.</p>

<script>
function upperCase() {
  const x = document.getElementById("fname");
  x.value = x.value.toUpperCase();
}
</script>

</body>
</html>
```

In the example

The upperCase() function will be called when a user changes the content of an input field.





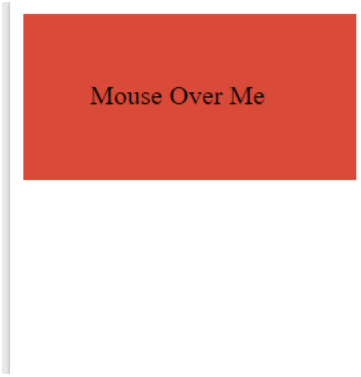


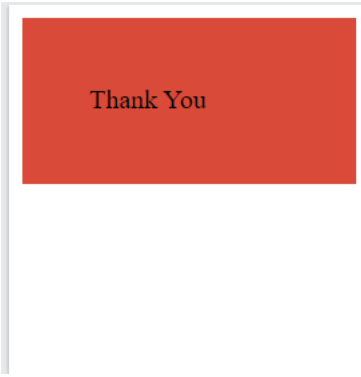
The Mouse Events

```
<!DOCTYPE html>
<html>
<body>

<div onmouseover="mOver(this)" onmouseout="mOut(this)"
style="background-color:#D94A38;width:120px;height:20px;padding:40px;">
Mouse Over Me</div>

<script>
function mOver(obj) {
  obj.innerHTML = "Thank You"
}

function mOut(obj) {
  obj.innerHTML = "Mouse Over Me"
}
</script>

</body>
</html>
```

- The onmouseover event
- The onmouseout events

They can be used to trigger a function when the user mouses over, or out of, an HTML element

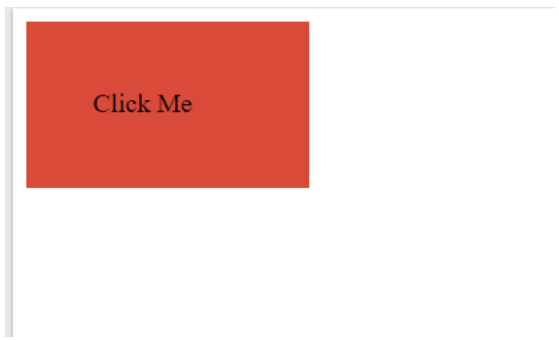


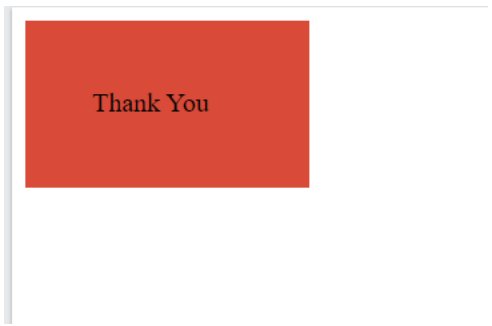


The Mouse Events

```
<!DOCTYPE html>
<html>
<body>

<div onmousedown="mDown(this)" onmouseup="mUp(this)"
style="background-color:#D94A38;width:90px;height:20px;padding:40px;">
Click Me</div>

<script>
function mDown(obj) {
  obj.style.backgroundColor = "#1ec5e5";
  obj.innerHTML = "Release Me";
}

function mUp(obj) {
  obj.style.backgroundColor="#D94A38";
  obj.innerHTML="Thank You";
}
</script>

</body>
</html>
```

- **onmousedown event:** when a mouse-button is clicked, the onmousedown event is triggered
- **onmouseup event:** when the mouse-button is released, the onmouseup event is triggered
- **onclick event:** when the mouse-click is completed, the onclick event is triggered

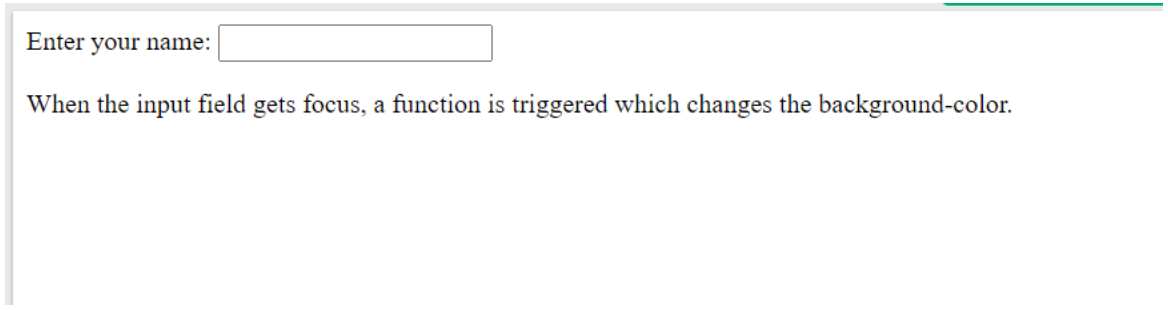


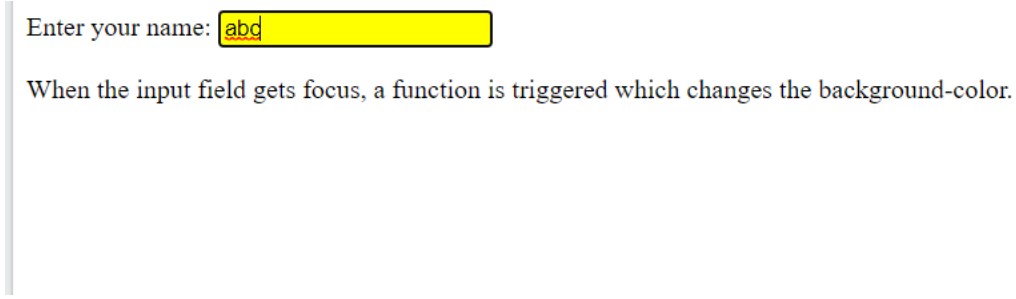


The onfocus Events

```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction(x) {
  x.style.background = "yellow";
}
</script>
</head>
<body>

Enter your name: <input type="text" onfocus="myFunction(this)">

<p>When the input field gets focus, a function is triggered which changes the background-color.</p>

</body>
</html>
```

- **onfocus event:** Change the background-color of an input field when it gets focus.





DOM EventListener

The addEventListener() method

- It attaches an event handler to the specified element.
- It attaches an event handler to an element without overwriting existing event handlers.
- We can add many event handlers to one element.
- We can add many event handlers of the same type to one element, i.e two "click" events.
- We can add event listeners to any DOM object not only HTML elements. i.e the window object.
- The addEventListener() method makes it easier to control how the event reacts to bubbling.
- When using the addEventListener() method, the JavaScript is separated from the HTML markup, for better readability and allows you to add event listeners even when you do not control the HTML markup.
- We can easily remove an event listener by using the removeEventListener() method.

The addEventListener() method

Syntax:

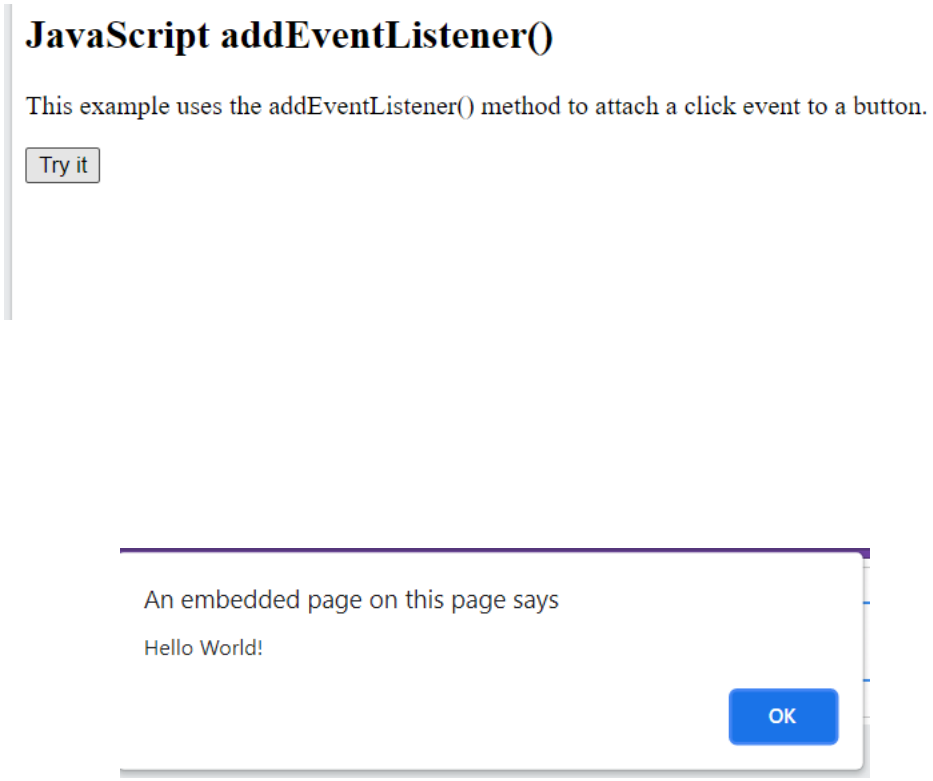
```
element.addEventListener(event, function, useCapture);
```

- The first parameter is the type of the event (like "click" or "mousedown" or any other HTML DOM Event.)
- The second parameter is the function we want to call when the event occurs.
- The third parameter is a boolean value specifying whether to use event bubbling or event capturing. This parameter is optional.

Note that you don't use the "on" prefix for the event; use "click" instead of "onclick".

Example

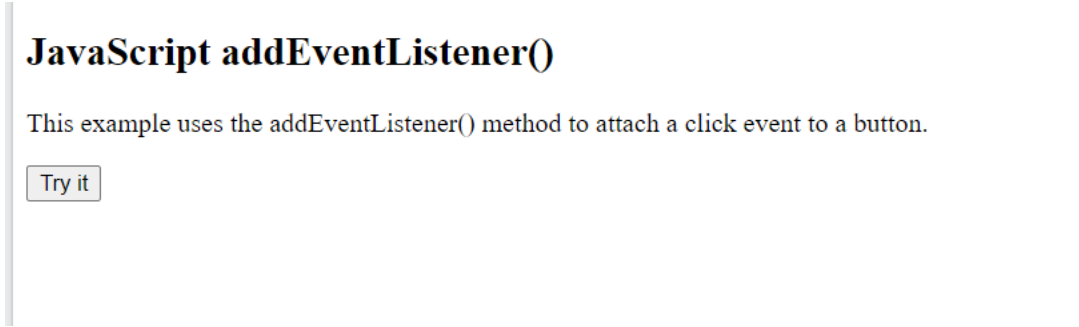
```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript addEventListener()</h2>

<p>This example uses the addEventListener() method to attach a click event to a button.</p>

<button id="myBtn">Try it</button>

<script>
document.getElementById("myBtn").addEventListener("click", function() {
  alert("Hello World!");
});
</script>

</body>
</html>
```

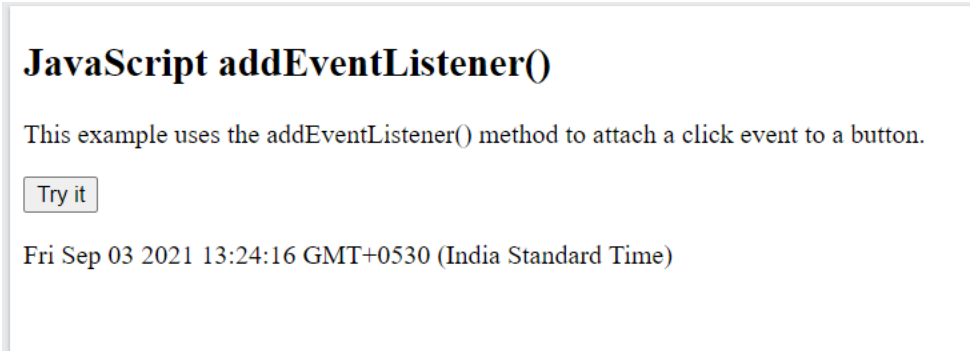


Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript addEventListener()</h2>

<p>This example uses the addEventListener() method to attach a click event to a button.</p>

<button id="myBtn">Try it</button>

<p id="demo"></p>

<script>
document.getElementById("myBtn").addEventListener("click", displayDate);

function displayDate() {
  document.getElementById("demo").innerHTML = Date();
}
</script>

</body>
</html>
```

- **Here external function displaydate( ) is written.**

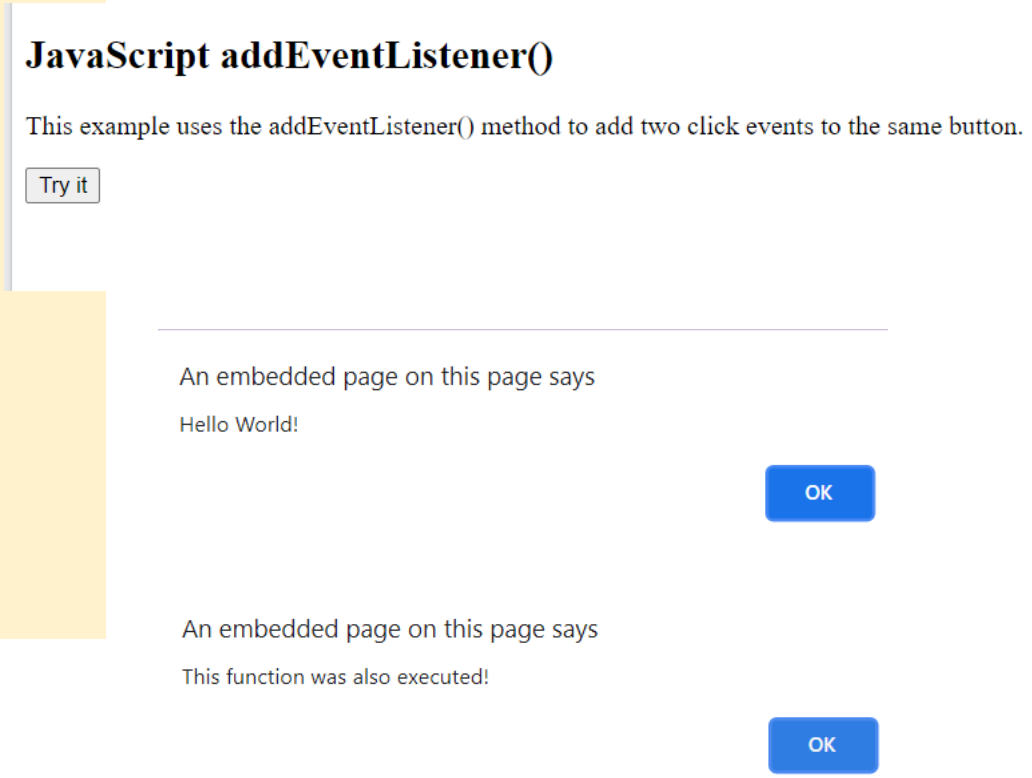




Many Event Handler to same element

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript addEventListener()</h2>

<p>This example uses the addEventListener() method to add two click events to the same
button.</p>

<button id="myBtn">Try it</button>

<script>
var x = document.getElementById("myBtn");
x.addEventListener("click", myFunction);
x.addEventListener("click", someOtherFunction);

function myFunction() {
  alert ("Hello World!");
}

function someOtherFunction() {
  alert ("This function was also executed!");
}
</script>

</body>
</html>
```

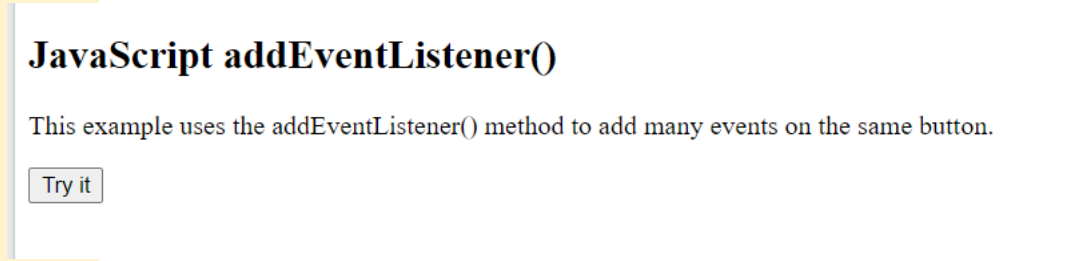
- **Two events are defined for button click**

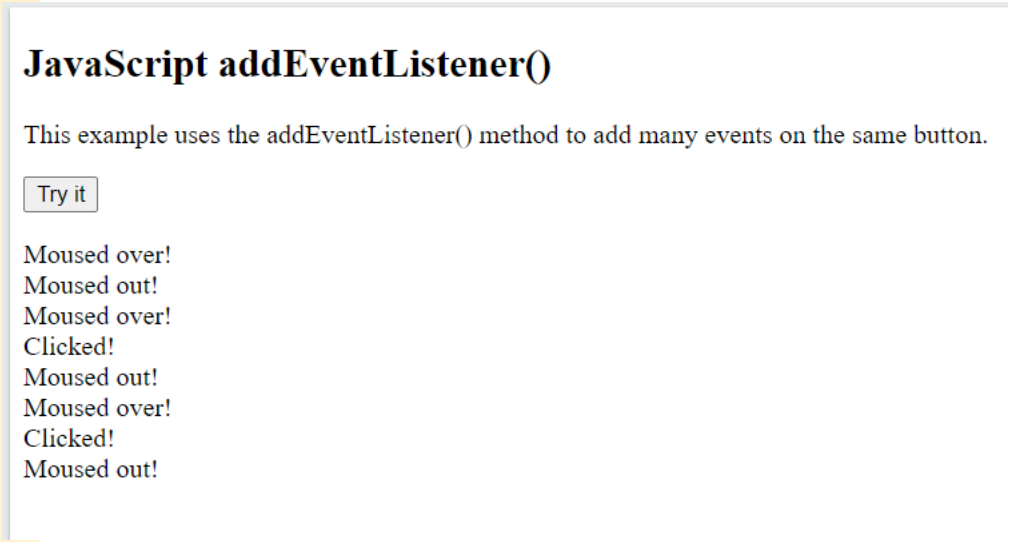


Many Event Handler to same element

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript addEventListener()</h2>

<p>This example uses the addEventListener() method to add many events on the same button.</p>

<button id="myBtn">Try it</button>

<p id="demo"></p>

<script>
var x = document.getElementById("myBtn");
x.addEventListener("mouseover", myFunction);
x.addEventListener("click", mySecondFunction);
x.addEventListener("mouseout", myThirdFunction);

function myFunction() {
  document.getElementById("demo").innerHTML += "Moused over!<br>";
}

function mySecondFunction() {
  document.getElementById("demo").innerHTML += "Clicked!<br>";
}

function myThirdFunction() {
  document.getElementById("demo").innerHTML += "Moused out!<br>";
}
</script>

</body>
</html>
```

- **You can add events of different types to the same element**

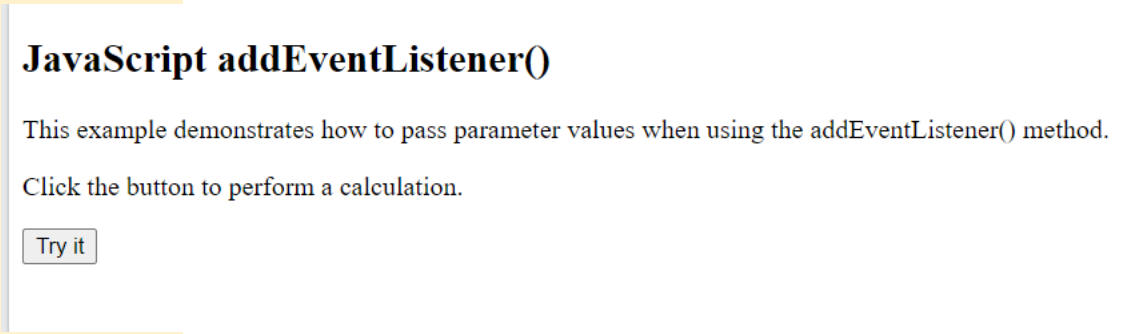


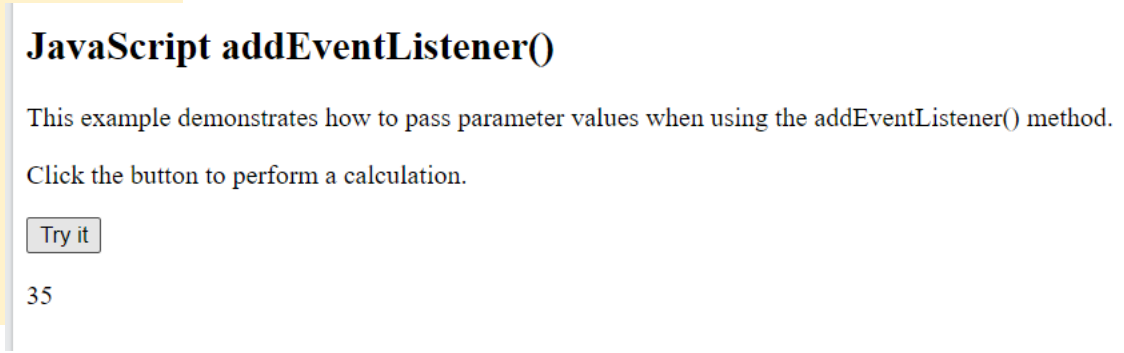


Passing Parameters

- When passing parameter values, use an "anonymous function" that calls the specified function with the parameters

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript addEventListener()</h2>

<p>This example demonstrates how to pass parameter values when using the addEventListener()
method.</p>

<p>Click the button to perform a calculation.</p>

<button id="myBtn">Try it</button>

<p id="demo"></p>

<script>
let p1 = 5;
let p2 = 7;
document.getElementById("myBtn").addEventListener("click", function() {
  myFunction(p1, p2);
});

function myFunction(a, b) {
  document.getElementById("demo").innerHTML = a * b;
}
</script>

</body>
</html>
```





Event Bubbling or Event Capturing?

There are two ways of event propagation in the HTML DOM

- Bubbling: In bubbling the inner most element's event is handled first and then the outer
- Capturing: In capturing the outer most element's event is handled first and then the inner

Event propagation is a way of defining the element order when an event occurs.

- If you have a <p> element inside a element, and the user clicks on the <p> element, which element's "click" event should be handled first?
- In bubbling: the <p> element's click event is handled first, then the element's click event.
- In capturing: the element's click event will be handled first, then the <p> element's click event.

Event Bubbling or Event Capturing?

Syntax:

With the addEventListener() method we can specify the propagation type by using the "useCapture" parameter:

```
addEventListener(event, function, useCapture);
```

- The default value is false, which will use the bubbling propagation,
- when the value is set to true, the event uses the capturing propagation.

Event Bubbling and Event Capturing

```
<!DOCTYPE html>
<html>
<head>
<style>
#myDiv1, #myDiv2 {
  background-color: coral;
  padding: 50px;
}

#myP1, #myP2 {
  background-color: white;
  font-size: 20px;
  border: 1px solid;
  padding: 20px;
}
</style>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type">
</head>
<body>

<h2>JavaScript addEventListener()</h2>

<div id="myDiv1">
  <h2>Bubbling:</h2>
  <p id="myP1">Click me!</p>
</div><br>

<div id="myDiv2">
  <h2>Capturing:</h2>
  <p id="myP2">Click me!</p>
</div>
```

```
<script>
document.getElementById("myP1").addEventListener("click", function() {
  alert("You clicked the white element!");
}, false);

document.getElementById("myDiv1").addEventListener("click", function() {
  alert("You clicked the orange element!");
}, false);

document.getElementById("myP2").addEventListener("click", function() {
  alert("You clicked the white element!");
}, true);

document.getElementById("myDiv2").addEventListener("click", function() {
  alert("You clicked the orange element!");
}, true);
</script>

</body>
</html>
```

Event Bubbling and Event Capturing

```
<script>
document.getElementById("myP1").addEventListener("click", function() {
  alert("You clicked the white element!");
}, false);

document.getElementById("myDiv1").addEventListener("click", function() {
  alert("You clicked the orange element!");
}, false);

document.getElementById("myP2").addEventListener("click", function() {
  alert("You clicked the white element!");
}, true);

document.getElementById("myDiv2").addEventListener("click", function() {
  alert("You clicked the orange element!");
}, true);
</script>

</body>
</html>
```

The removeEventListener() method

```
<!DOCTYPE html>
<html>
<head>
<style>
#myDIV {
  background-color: coral;  border: 1px solid;  padding: 50px;  color: white;  font-size: 20px;
}
</style>
</head>
<body>

<h2>JavaScript removeEventListener()</h2>

<div id="myDIV">
  <p>This div element has an onmousemove event handler that displays a random number every time
you move your mouse inside this orange field.</p>
  <p>Click the button to remove the div's event handler.</p>
  <button onclick="removeHandler()" id="myBtn">Remove</button>
</div>

<p id="demo"></p>

<script>
document.getElementById("myDIV").addEventListener("mousemove", myFunction);

function myFunction() {
  document.getElementById("demo").innerHTML = Math.random();
}

function removeHandler() {
  document.getElementById("myDIV").removeEventListener("mousemove", myFunctior
}
</script>
</body>
</html>
```

It removes event handlers that have been attached with the addEventListener() method: