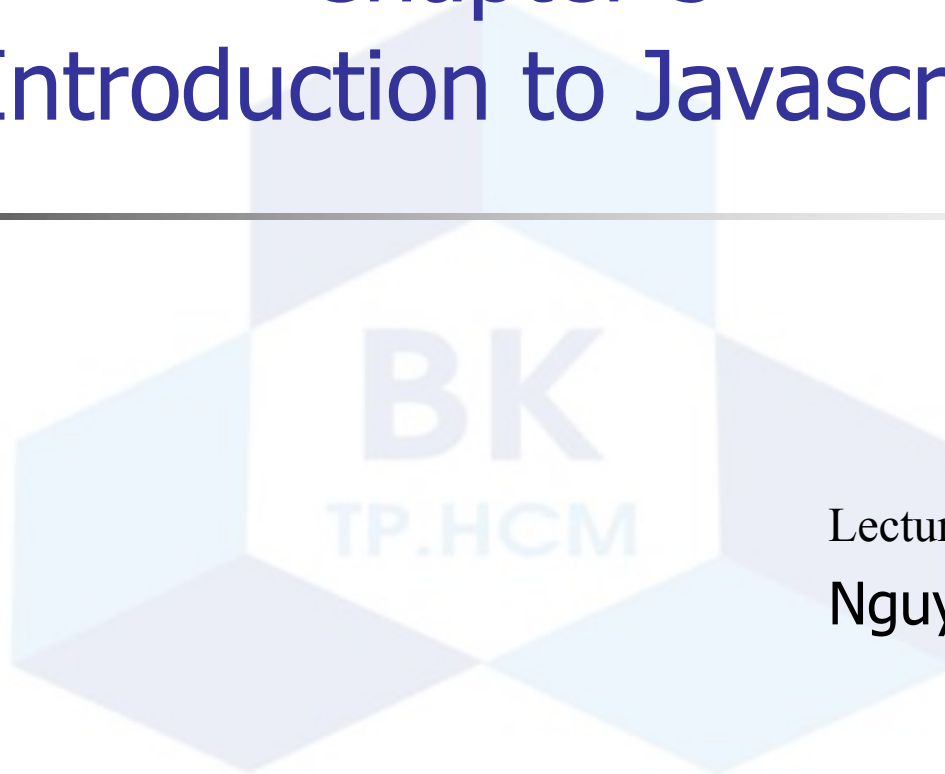


# Chapter 3

## Introduction to Javascript

---

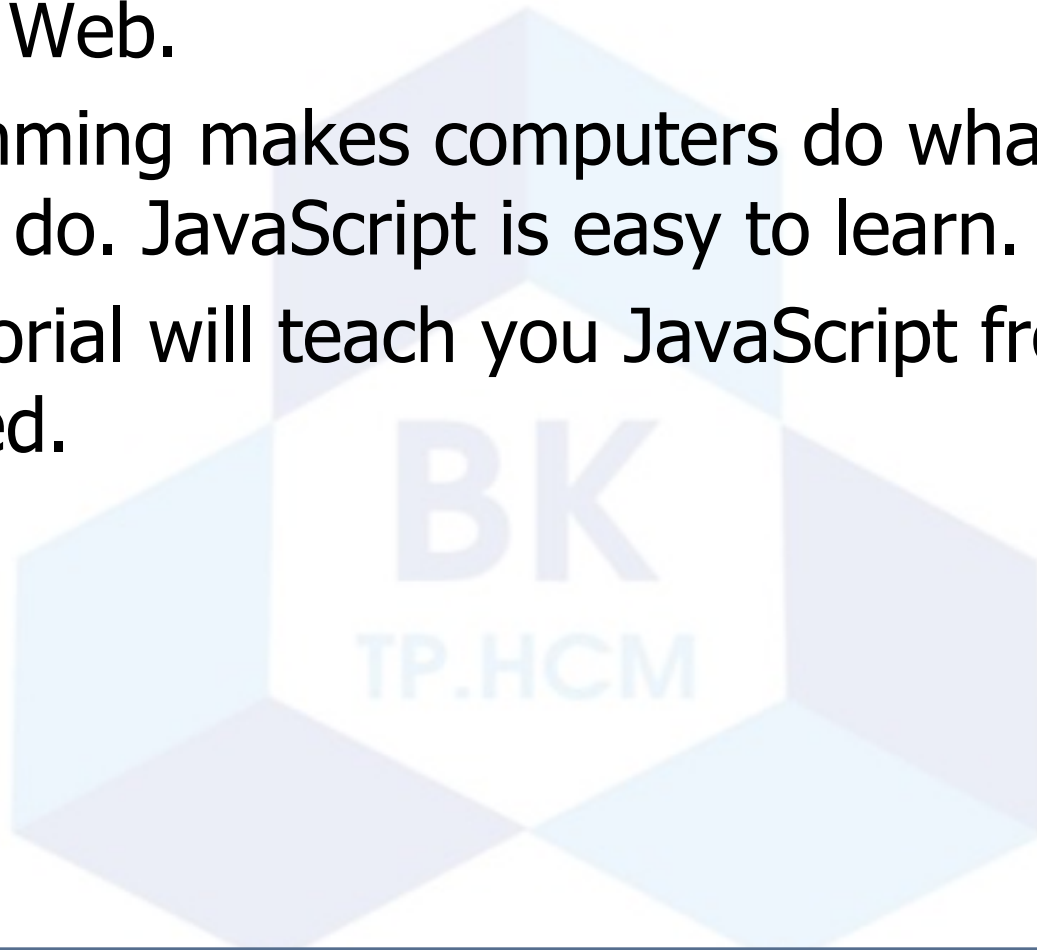


Lectured by:  
Nguyễn Hữu Hiếu

# JavaScript

---

- JavaScript is the programming language of HTML and the Web.
- Programming makes computers do what you want them to do. JavaScript is easy to learn.
- This tutorial will teach you JavaScript from basic to advanced.



# Why Study JavaScript?

---

- JavaScript is one of the 3 languages all web developers must learn:
  1. HTML to define the content of web pages
  2. CSS to specify the layout of web pages
  3. JavaScript to program the behaviour of web pages
- This lecture is about JavaScript, and how JavaScript works with HTML and CSS.

# JavaScript Introduction

- JavaScript Can Change HTML Content  
One of many HTML methods is `getElementById()`.
- This example uses the method to "find" an HTML element (with `id="demo"`), and changes the element content (`innerHTML`) to "Hello JavaScript":

```
document.getElementById( "demo" ).innerHTML  
    = "Hello JavaScript";
```

# Example

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<h1>What Can JavaScript Do?</h1>
```

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

```
<button type="button"
```

```
onclick="document.getElementById('demo').innerHTML =  
'Hello JavaScript!'">
```

Click Me!

```
</button>
```

```
</body>
```

```
</html>
```

# JavaScript Can Change HTML Attributes

**This example changes an HTML image, by changing the src attribute of an `<img>` tag:**

```
<!DOCTYPE html><html>
<body>
  <h1>JavaScript Can Change Images</h1>
  
  <p>Click the light bulb to turn on/off the light.</p>
  <script>
    function changeImage() {
      var image = document.getElementById('myImage');
      if (image.src.match("bulbon")) {
        image.src = "pic_bulboff.gif"; } else {
        image.src = "pic_bulbon.gif"; }
    }
  </script>
</body> </html>
```

# JavaScript Can Change HTML Styles (CSS)

```
<!DOCTYPE html>
<html>
  <body>
    <h1>What Can JavaScript Do?</h1>
    <p id="demo">JavaScript can change the style of an HTML element.</p>
    <script>
      function myFunction() {
        var x = document.getElementById("demo");
        x.style.fontSize = "25px";
        x.style.color = "red";
      }
    </script>
    <button type="button" onclick="myFunction()">Click Me!</button>
  </body>
</html>
```

# JavaScript Can Validate Data

- JavaScript is often used to validate input



## Server-side validation

- 1) The user submits the form to the Web server.
- 2) The Web server validates the user's responses and, if necessary, returns the form to the user for correction.
- 3) After correcting any errors, the user resubmits the form to the Web server for another validation.



## Client-side validation

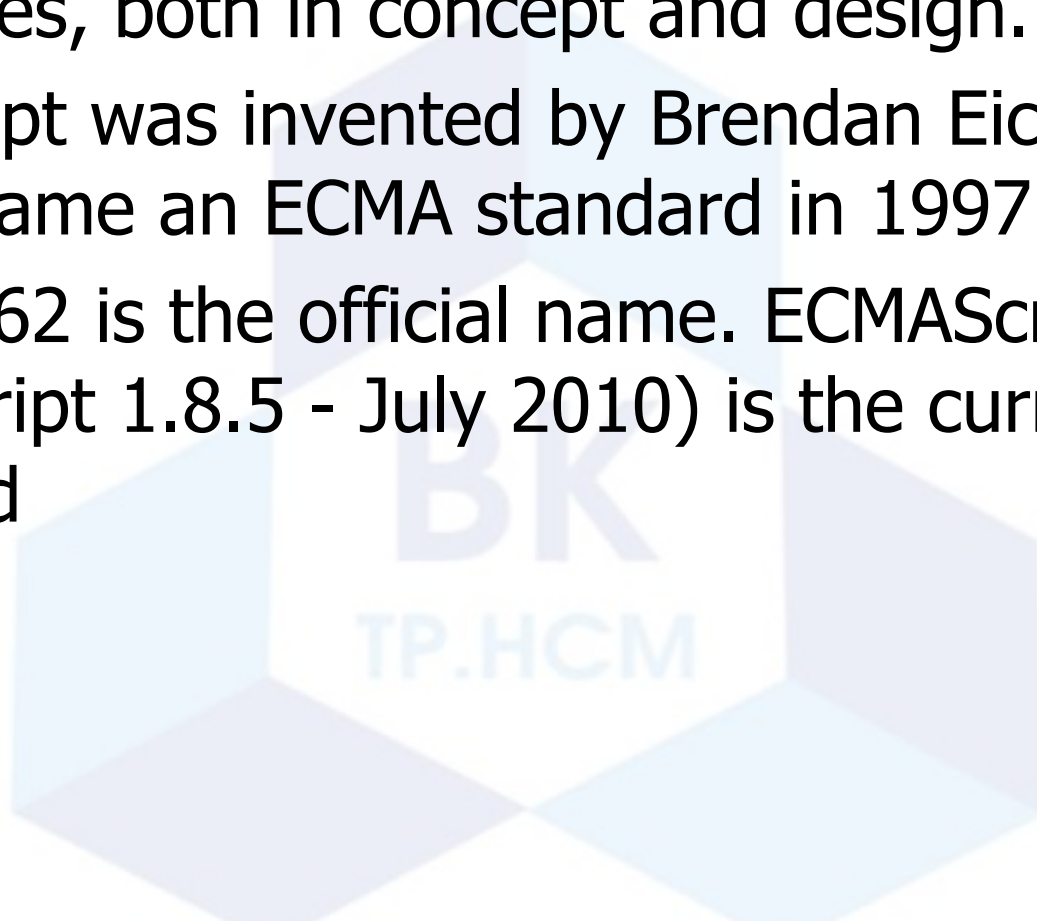
- 1) The user submits the form, and validation is performed on the user's computer.
- 2) After correcting any errors, the user submits the form to the Web server.



# Did You Know?

---

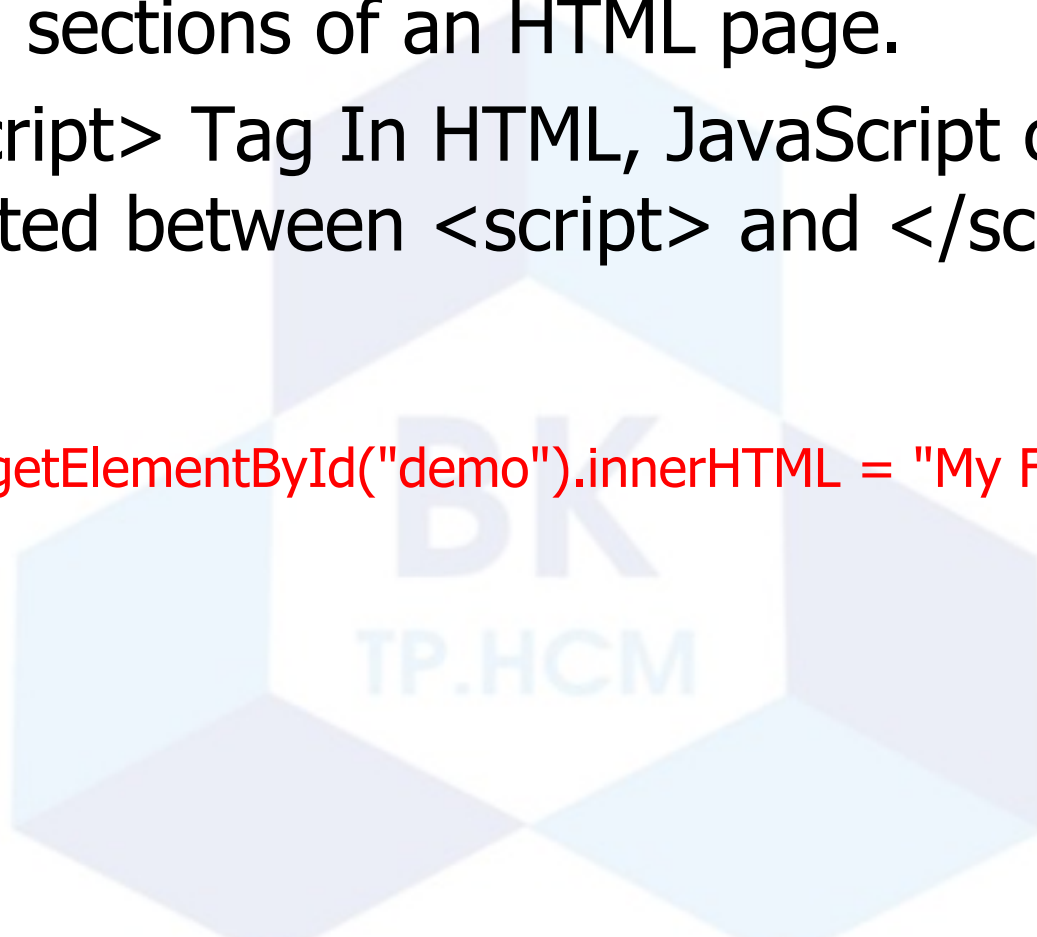
- JavaScript and Java are completely different languages, both in concept and design.
- JavaScript was invented by Brendan Eich in 1995, and became an ECMA standard in 1997.
- ECMA-262 is the official name. ECMAScript 5 (JavaScript 1.8.5 - July 2010) is the current standard



# JavaScript Where To

- JavaScript can be placed in the <body> and the <head> sections of an HTML page.
- The <script> Tag In HTML, JavaScript code must be inserted between <script> and </script> tag

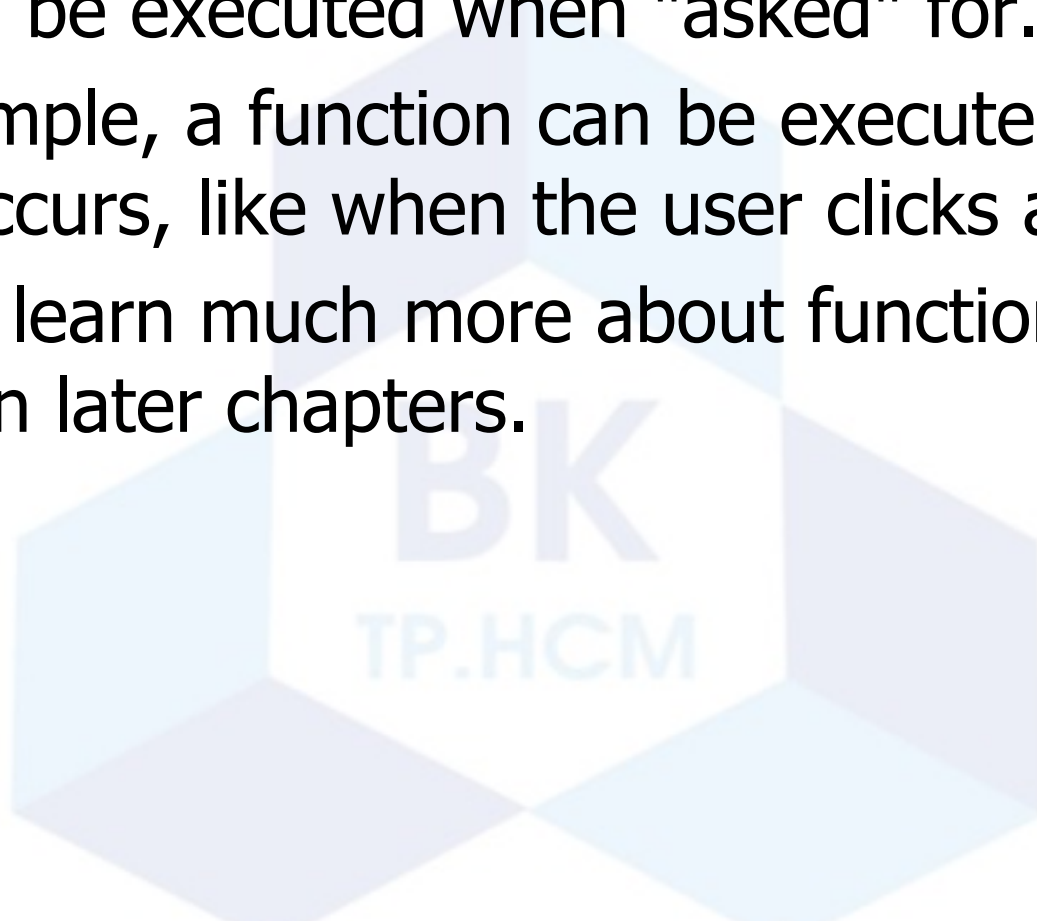
```
<script>  
    document.getElementById("demo").innerHTML = "My First JavaScript";  
</script>
```



# JavaScript Functions and Events

---

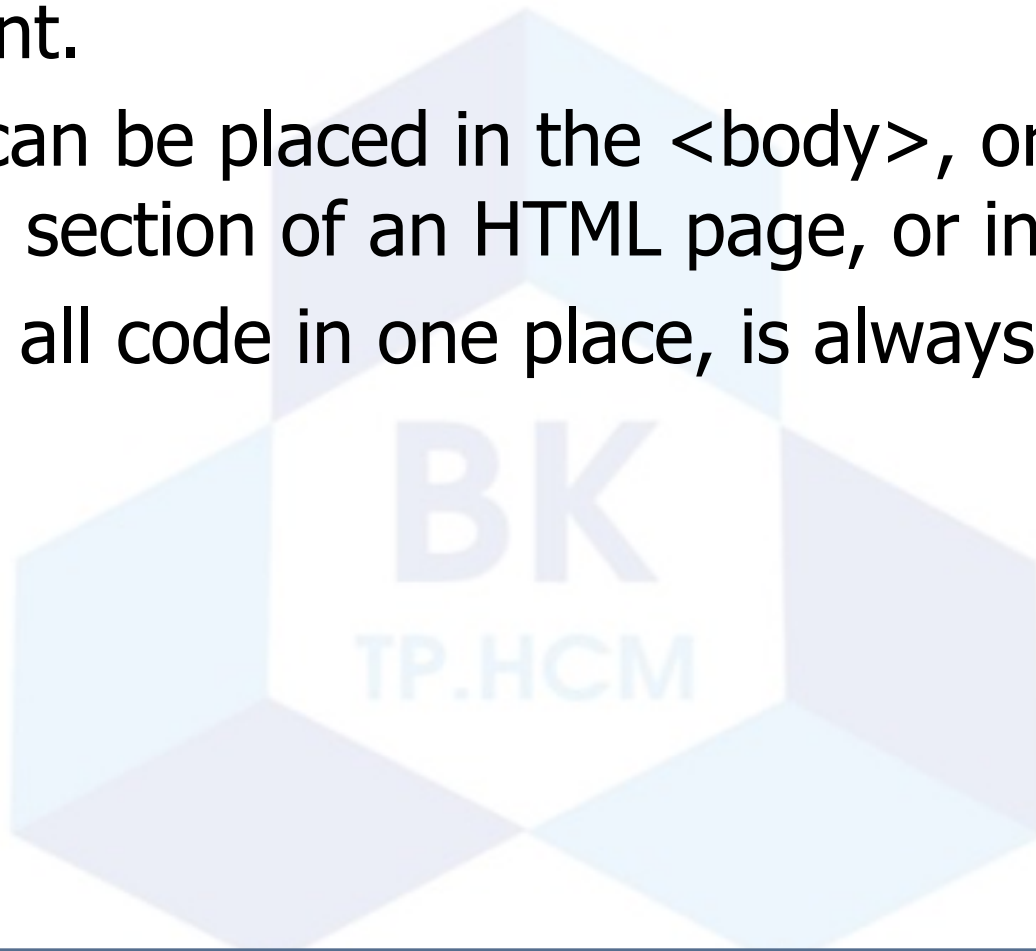
- A JavaScript function is a block of JavaScript code, that can be executed when "asked" for.
- For example, a function can be executed when an event occurs, like when the user clicks a button.
- You will learn much more about functions and events in later chapters.



# JavaScript in <head> or <body>

---

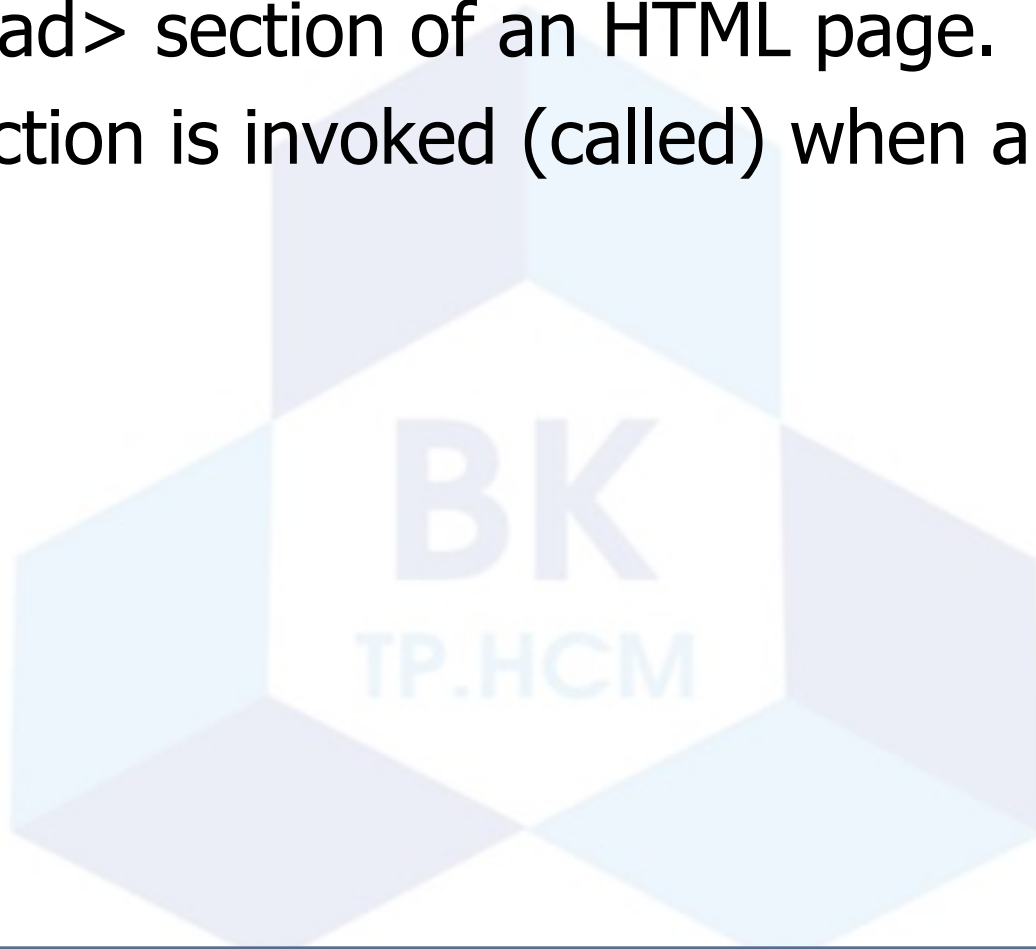
- You can place any number of scripts in an HTML document.
- Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both.
- Keeping all code in one place, is always a good habit.



# JavaScript in <head>

---

- In this example, a JavaScript function is placed in the <head> section of an HTML page.
- The function is invoked (called) when a button is clicked:



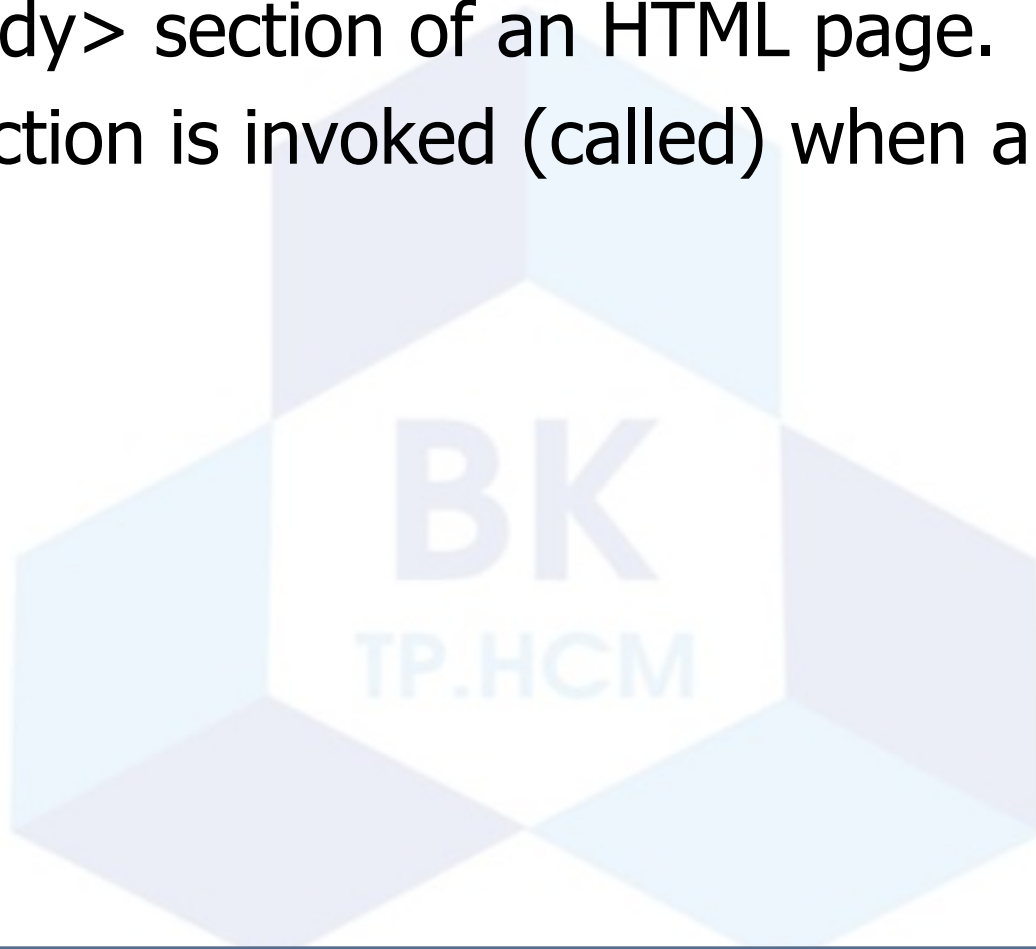
# Example

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      function myFunction() {
        document.getElementById("demo").innerHTML = "Paragraph
changed."; }
    </script>
  </head>
  <body>
    <h1>My Web Page</h1>
    <p id="demo">A Paragraph</p>
    <button type="button" onclick="myFunction()">Try it</button>
  </body>
</html>
```

# JavaScript in <body>

---

- In this example, a JavaScript function is placed in the <body> section of an HTML page.
- The function is invoked (called) when a button is clicked:



# Example

---

```
<!DOCTYPE html>
<html>
  <body>
    <h1>My Web Page</h1>
    <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>
```



# External JavaScript

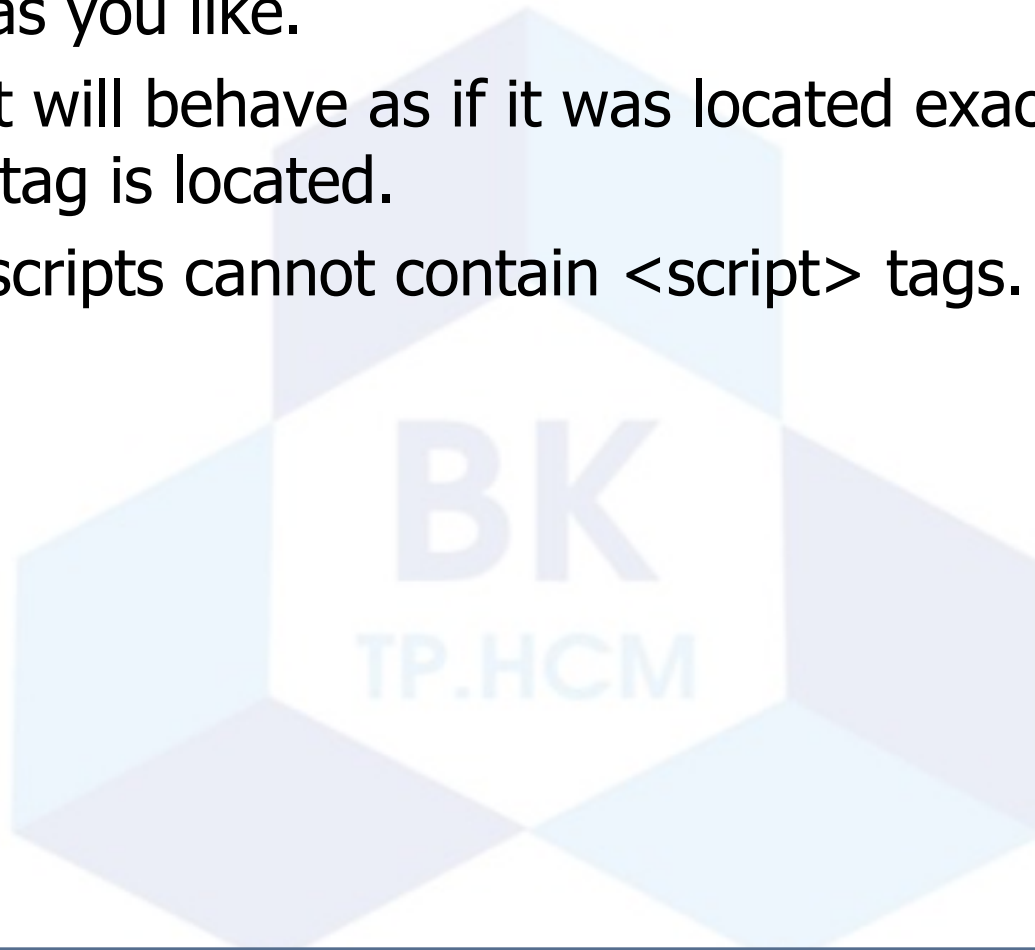
- Scripts can also be placed in external files.
- External scripts are practical when the same code is used in many different web pages.
- JavaScript files have the file extension .js.  
To use an external script, put the name of the script file in the src (source)
- attribute of the <script> tag:

```
<!DOCTYPE html>  
<html>  
  <body>  
    <script src="myScript.js"></script>  
  </body>  
</html>
```

# External JavaScript

---

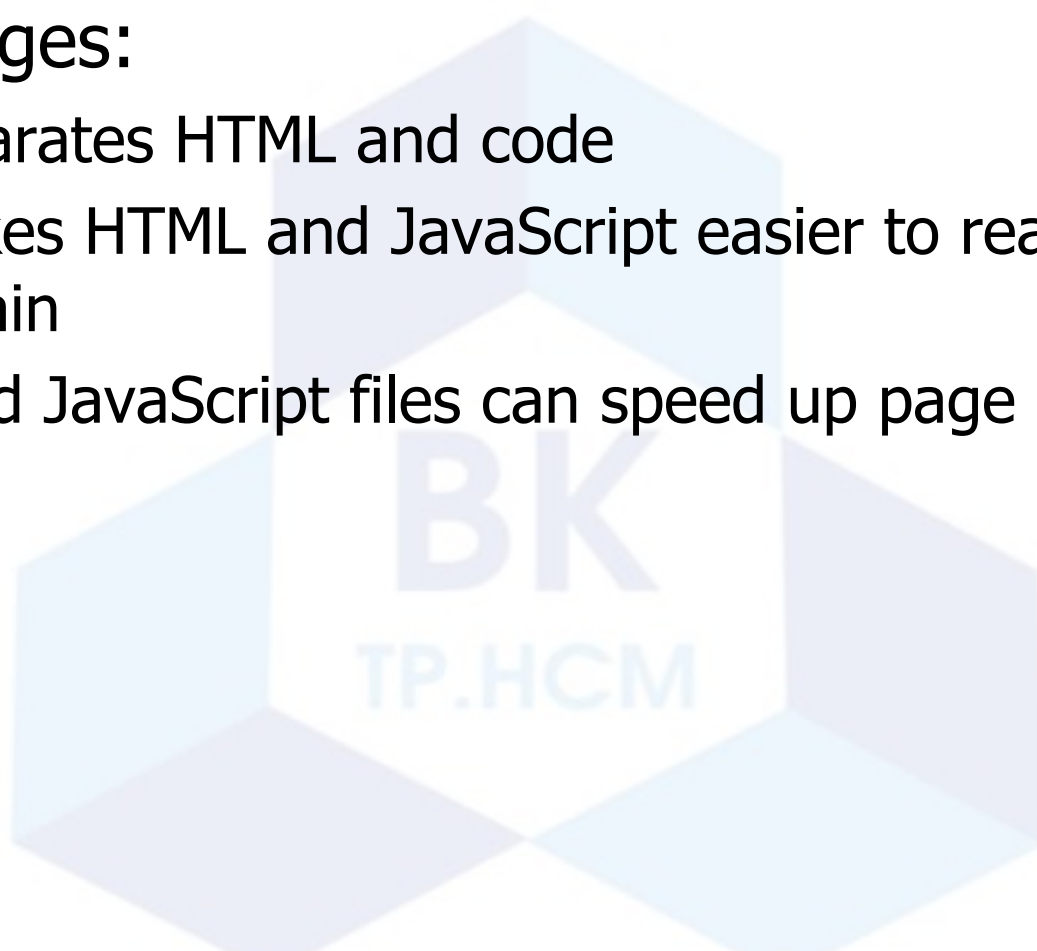
- You can place an external script reference in `<head>` or `<body>` as you like.
- The script will behave as if it was located exactly where the `<script>` tag is located.
- External scripts cannot contain `<script>` tags.



# External JavaScript Advantages

---

- Placing JavaScripts 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



# JavaScript Output

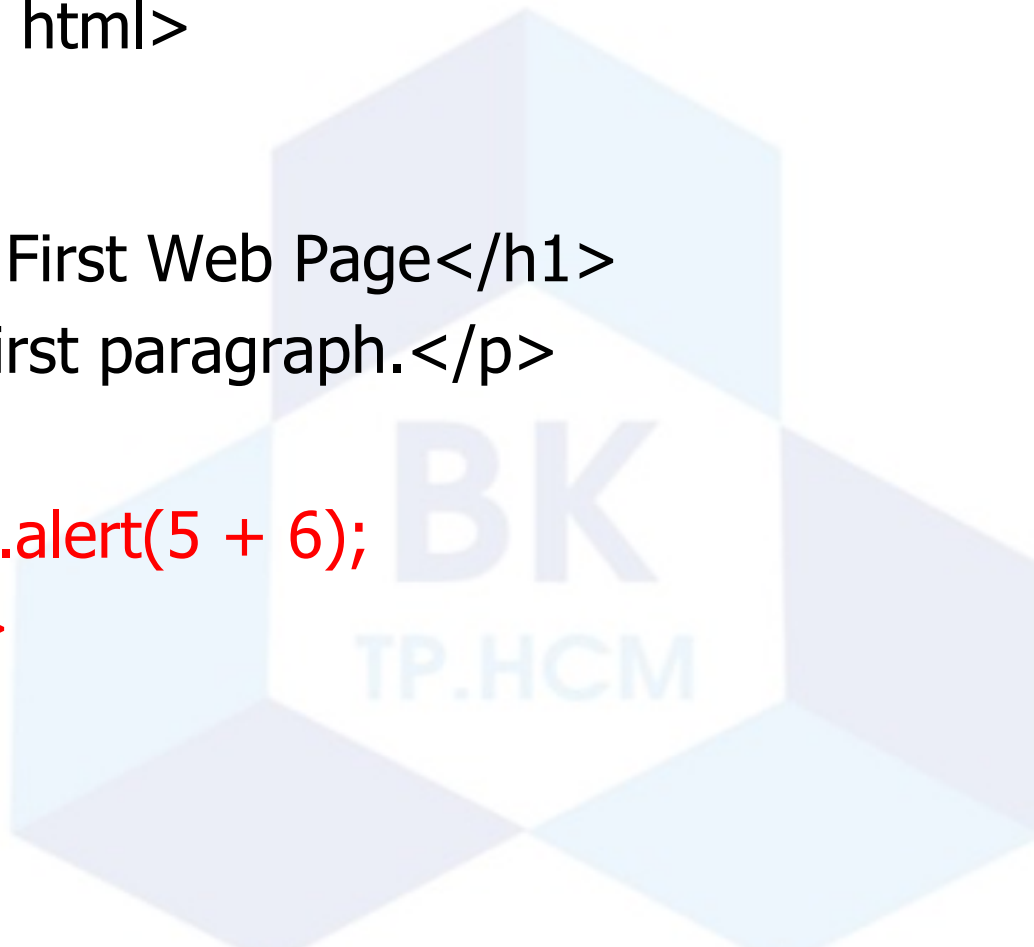
---

- JavaScript does not have any built-in print or display functions.
- JavaScript Display Possibilities JavaScript can "display" data in different ways:
  1. Writing into an alert box, using `window.alert()`.
  2. Writing into the HTML output using `document.write()`.
  3. Writing into an HTML element, using `innerHTML`.
  4. Writing into the browser console, using `console.log()`.

# Using window.alert()

**You can use an alert box to display:**

```
<!DOCTYPE html>
<html>
  <body>
    <h1>My First Web Page</h1>
    <p>My first paragraph.</p>
    <script>
      window.alert(5 + 6);
    </script>
  </body>
</html>
```



# Using document.write()

**For testing purposes, it is convenient to use document.write():**

```
<!DOCTYPE html>
<html>
  <body>
    <h1>My First Web Page</h1>
    <p>My first paragraph.</p>
    <script>
      document.write(5 + 6);
    </script>
  </body>
</html>
```

# Using document.write()

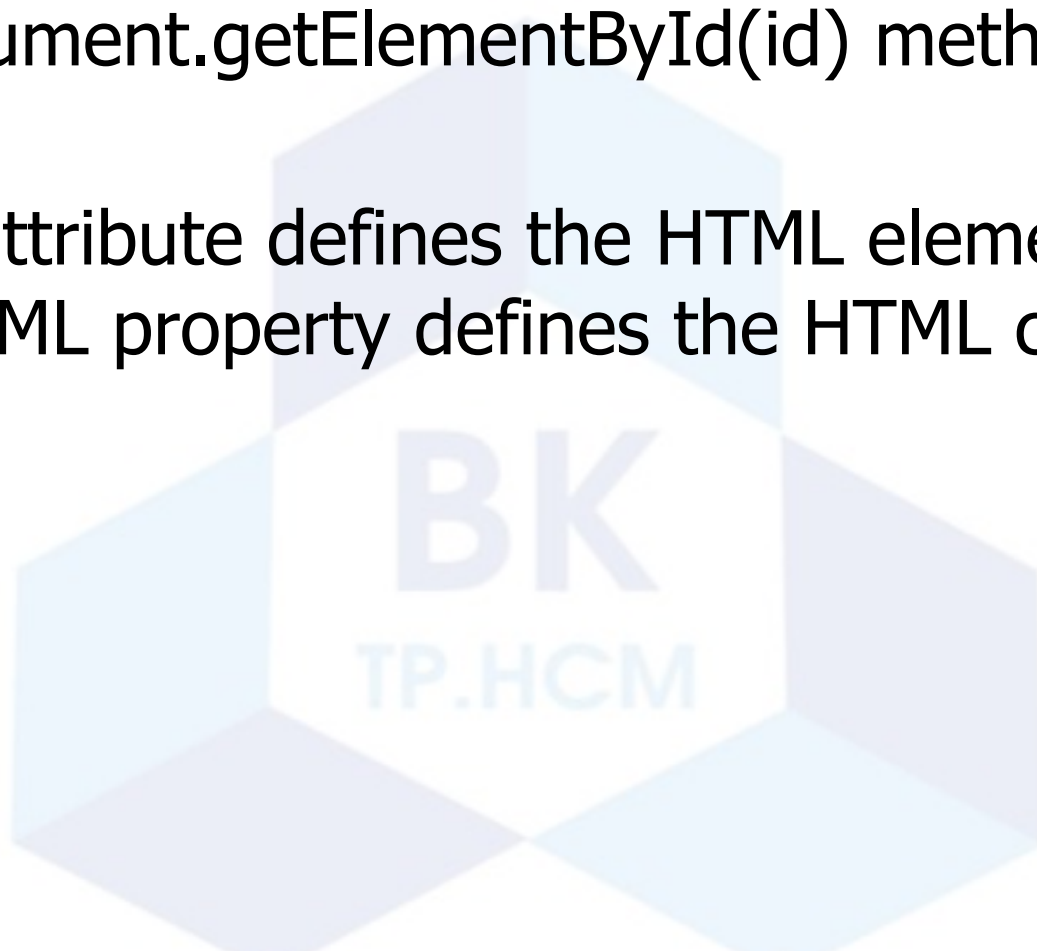
**Using document.write() after an HTML document is fully loaded, will delete all existing HTML:**

```
<!DOCTYPE html>
<html>
  <body>
    <h1>My First Web Page</h1>
    <p>My first paragraph.</p>
    <button onclick="document.write(5 + 6)">Try it</button>
  </body>
</html>
```

# Using 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:





# Example

```
<!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>
```

## Using console.log()

---

- In your browser, you can use the console.log() method to display data.
- Activate the browser console with F12, and select "Console" in the menu.



# Example

---

```
<!DOCTYPE html>
<html>
  <body>
    <h1>My First Web Page</h1>
    <p>My first paragraph.</p>
    <script>
      console.log(5 + 6);
    </script>
  </body>
</html>
```

# JavaScript Syntax

---

- JavaScript syntax is the set of rules, how JavaScript programs are constructed.
- A computer program is a list of "instructions" to be "executed" by the computer.
- In a programming language, these program instructions are called statements.
- JavaScript is a programming language.  
JavaScript statements are separated by semicolon.

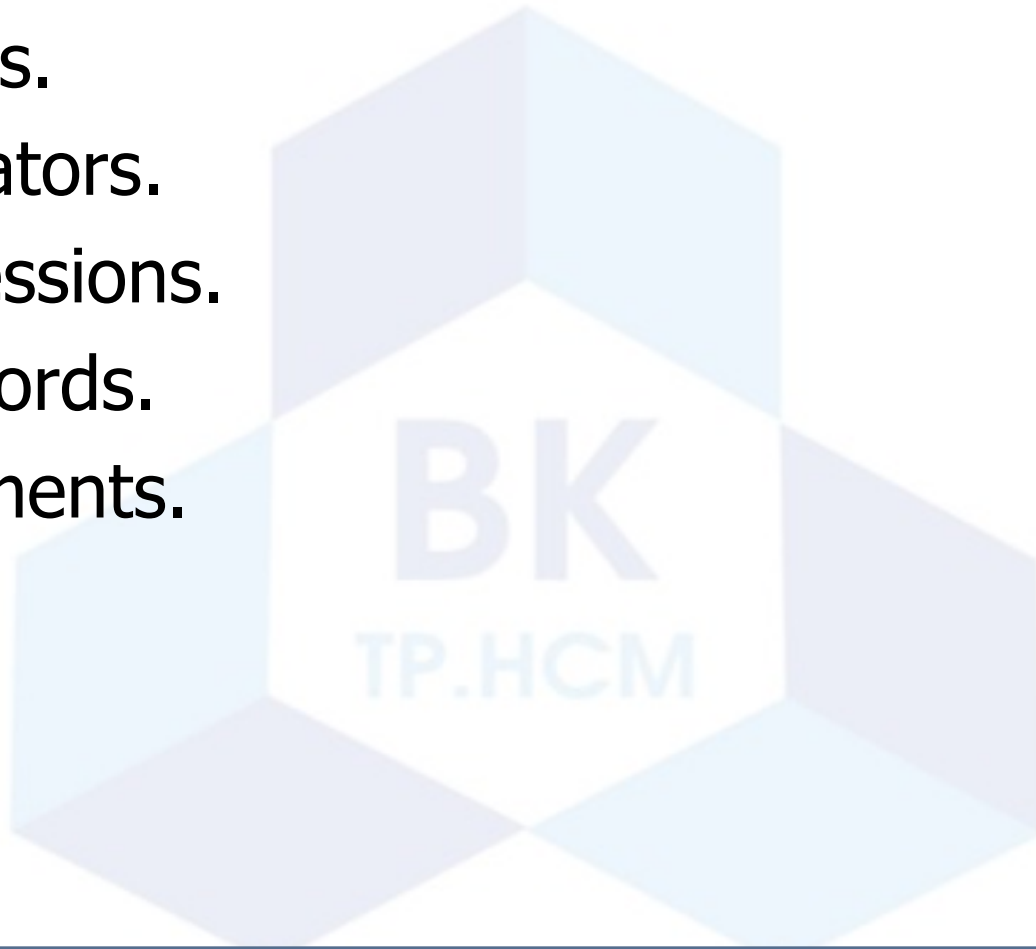
# Example

```
<!DOCTYPE html>
<html>
<body>
  <h1>JavaScript Statements</h1>
  <p>Statements are separated by semicolons.</p>
  <p>The variables x, y, and z are assigned the values 5, 6, and 11:</p>
  <p id="demo"></p>
  <script>
    var x = 5;
    var y = 6;
    var z = x + y;
    document.getElementById("demo").innerHTML = z;
  </script>
</body>
</html>
```

# JavaScript Statements

---

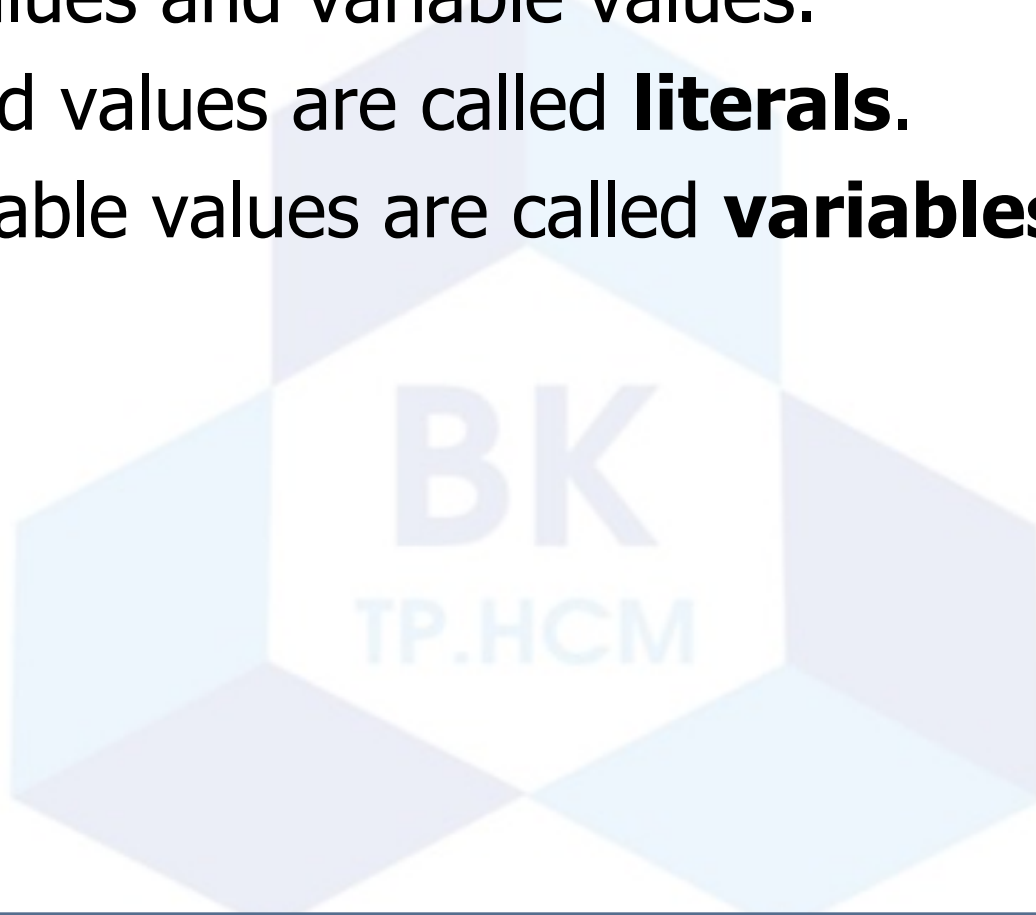
- JavaScript statements are composed of:
  - Values.
  - Operators.
  - Expressions.
  - Keywords.
  - Comments.



# 1-JavaScript Values

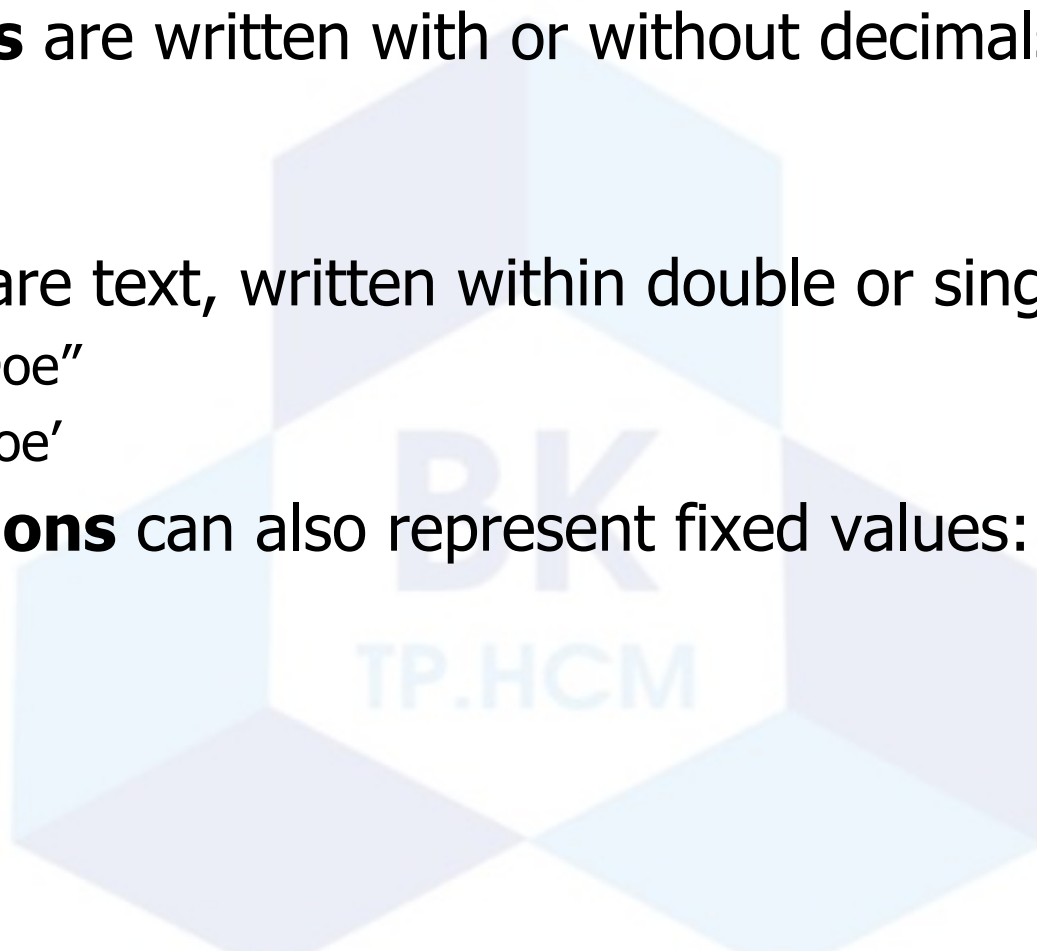
---

- The JavaScript syntax defines two types of values: Fixed values and variable values.
- 1.1-Fixed values are called **literals**.
- 2.1-Variable values are called **variables**.



# 1.1-JavaScript Literals

- The most important rules for writing fixed values are:
- **Numbers** are written with or without decimals:
  - 10.50
  - 1001
- **Strings** are text, written within double or single quotes:
  - "John Doe"
  - 'John Doe'
- **Expressions** can also represent fixed values:
  - 5+6
  - 5 \* 10



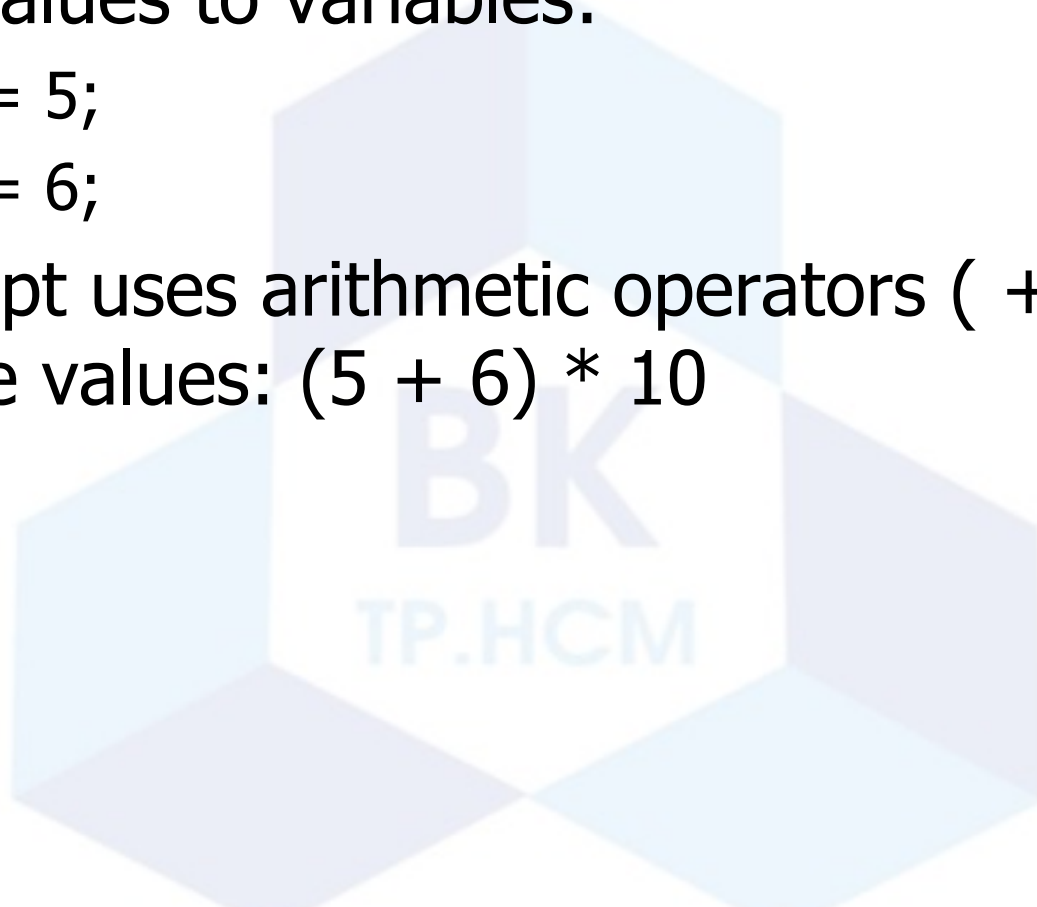


## 1.2-JavaScript Variables

- In a programming language, variables are used to store data values.
- JavaScript uses the `var` keyword to define variables.  
An equal sign is used to assign values to variables.
- In this example, `x` is defined as a variable. Then, `x` is assigned (given) the value 6:
  - `var x;`
  - `x = 6;`

## 2-JavaScript Operators

- JavaScript uses an assignment operator ( = ) to assign values to variables:
  - `var x = 5;`
  - `var y = 6;`
- JavaScript uses arithmetic operators ( + - \* / ) to compute values:  $(5 + 6) * 10$



# Object literals

- You don't declare the *types* of variables in JavaScript
- JavaScript has object *literals*, written with this syntax:
  - **`{ name1 : value1 , ... , nameN : valueN }`**
- Example (from Netscape's documentation):
  - `car = {myCar: "Saturn", 7: "Mazda",  
getCar: CarTypes("Honda"), special: Sales}`
    - The fields are `myCar`, `getCar`, `7` (this is a legal field name) , and `special`
    - `"Saturn"` and `"Mazda"` are Strings
    - `CarTypes` is a function call
    - `Sales` is a variable you defined earlier
  - Example use: `document.write("I own a " + car.myCar);`

# Three ways to create an object

- You can use an object literal:
  - `var course = { number: "CIT597", teacher: "Dr. Dave" }`
- You can use `new` to create a “blank” object, and add fields to it later:
  - `var course = new Object();`  
`course.number = "CIT597";`  
`course.teacher = "Dr. Dave";`
- You can write and use a constructor:
  - `function Course(n, t) { // best placed in <head>`  
`this.number = n; // keyword "this" is required, not optional`  
`this.teacher = t;`  
`}`
  - `var course = new Course("CIT597", "Dr. Dave");`



# Four ways to create an array

- You can use an array literal:  
`var colors = ["red", "green", "blue"];`
- You can use `new Array()` to create an empty array:
  - `var colors = new Array();`
  - You can add elements to the array later:  
`colors[0] = "red"; colors[2] = "blue"; colors[1]="green";`
- You can use `new Array(n)` with a single numeric argument to create an array of that size
  - `var colors = new Array(3);`
- You can use `new Array(...)` with *two or more* arguments to create an array containing those values:
  - `var colors = new Array("red", "green", "blue");`

# The length of an array

- If `myArray` is an array, its length is given by `myArray.length`
- Array length can be changed by assignment beyond the current length
  - Example: `var myArray = new Array(5); myArray[10] = 3;`
- Arrays are **sparse**, that is, space is only allocated for elements that have been assigned a value
  - Example: `myArray[50000] = 3;` is perfectly OK
  - But indices must be between 0 and  $2^{32}-1$
- As in C and Java, there are no two-dimensional arrays; but you can have an array of arrays: `myArray[5][3]`

# Arrays and objects

- Arrays *are* objects
- `car = { myCar: "Saturn", 7: "Mazda" }`
  - `car[7]` is the same as `car.7`
  - `car.myCar` is the same as `car["myCar"]`
- If you *know* the name of a property, you can use dot notation: `car.myCar`
- If you *don't know* the name of a property, but you have it in a variable (or can compute it), you *must* use array notation: `car["my" + "Car"]`



# Array functions

- If `myArray` is an array,
  - `myArray.sort()` sorts the array alphabetically
  - `myArray.sort(function(a, b) { return a - b; })` sorts numerically
  - `myArray.reverse()` reverses the array elements
  - `myArray.push(...)` adds any number of new elements to the end of the array, and increases the array's length
  - `myArray.pop()` removes and returns the last element of the array, and decrements the array's length
  - `myArray.toString()` returns a string containing the values of the array elements, separated by commas

# The for...in statement

- You can loop through all the properties of an object with **for (variable in object) statement**;
  - Example: 

```
for (var prop in course) {  
    document.write(prop + ": " + course[prop]);  
}
```
  - Possible output: 

```
teacher: Dr. Dave  
number: CIT597
```
  - The properties are accessed in an *undefined* order
  - If you add or delete properties of the object within the loop, it is *undefined* whether the loop will visit those properties
  - Arrays *are* objects; applied to an array, **for...in** will visit the "properties" `0, 1, 2, ...`
  - Notice that `course["teacher"]` is equivalent to `course.teacher`
    - You must use brackets if the property name is *in a variable*

## 3-JavaScript Keywords

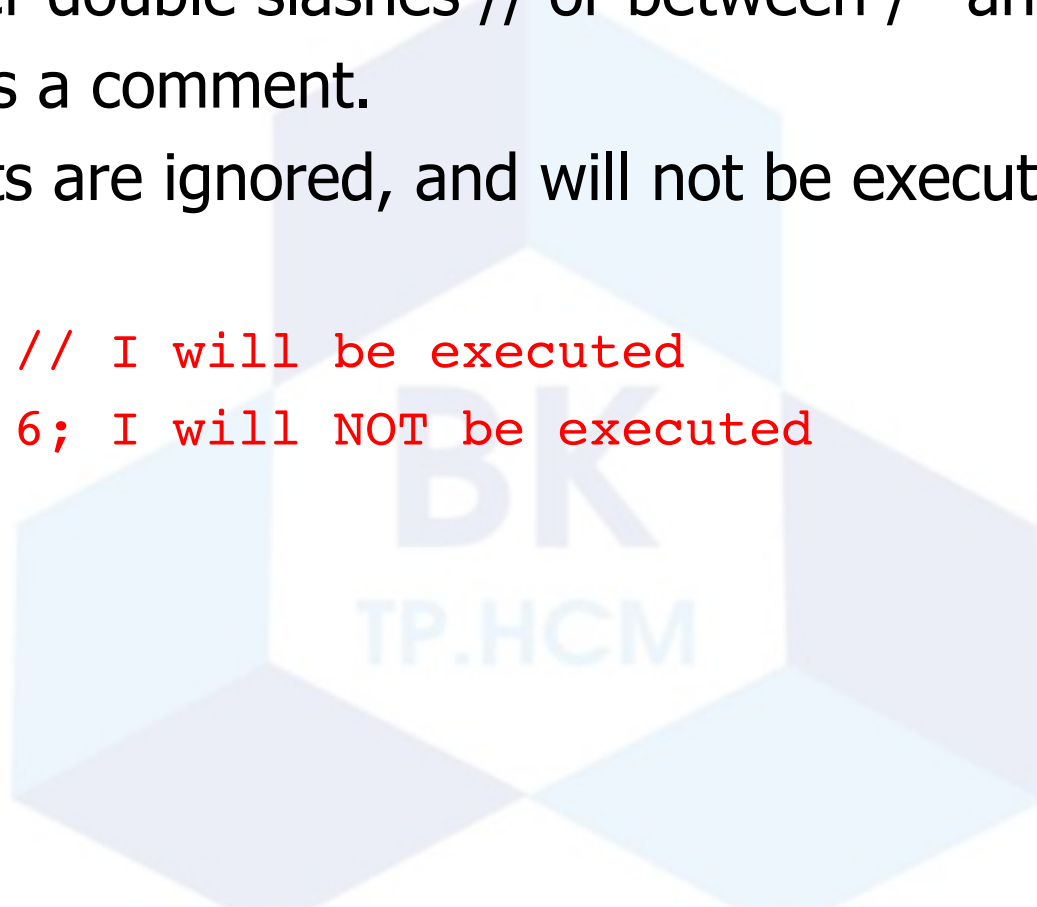
- JavaScript keywords are used to identify actions to be performed.
- The var keyword tells the browser to create a new variable:
  - `var x = 5 + 6;`
  - `var y = x * 10;`

BK  
TP.HCM

## 4-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:

```
var x = 5; // I will be executed  
// var x = 6; I will NOT be executed
```



# JavaScript is Case Sensitive

- All JavaScript identifiers are case sensitive. The variables `lastName` and `lastname`, are two different variables.

```
lastName = "Doe";  
lastname = "Peterson";
```



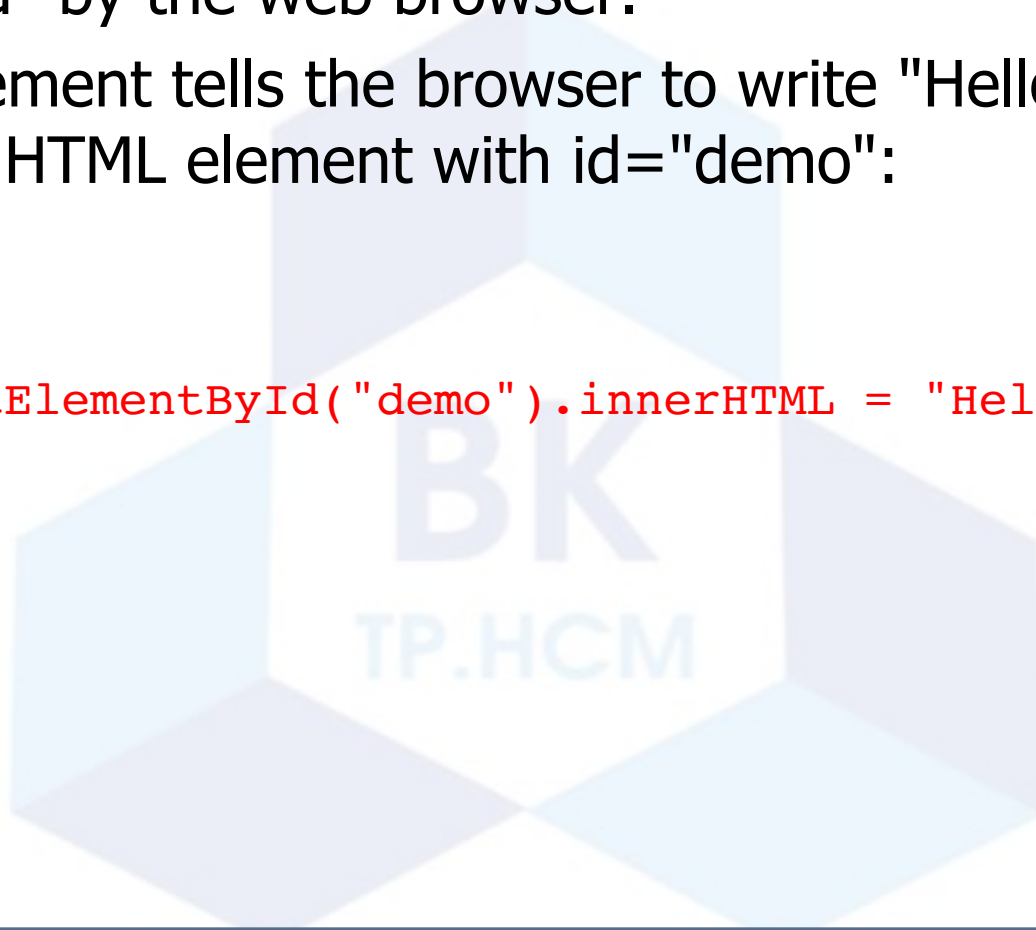
# JavaScript and Camel Case

- Historically, programmers have used three ways of joining multiple words into one variable name:
- Hyphens:
  - first-name, last-name, master-card, inter-city.
- Underscore:
  - first\_name, last\_name, master\_card, inter\_city.
- Camel Case:
  - FirstName, LastName, MasterCard, InterCity.
- Hyphens are not allowed in JavaScript. It is reserved for subtractions.

# JavaScript Statements

- In HTML, JavaScript statements are "instructions" to be "executed" by the web browser.
- This statement tells the browser to write "Hello Dolly." inside an HTML element with id="demo":
- Example:

```
document.getElementById("demo").innerHTML = "Hello Dolly.";
```



# JavaScript Programs

- Most JavaScript programs contain many JavaScript statements.
- The statements are executed, one by one, in the same order as they are written.
- In this example, x, y, and z is given values, and finally z is displayed: Example:

```
var x = 5;  
var y = 6;  
var z = x + y;  
document.getElementById("demo").innerHTML = z;
```

- 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:

```
a = 5;
```

```
b = 6;
```

```
c = a + b;
```

When separated by semicolons, multiple statements on one line are allowed: `a = 5; b = 6; c = a + b;`

- On the web, you might see examples without semicolons. Ending statements with semicolon is not required, but highly recommended.

# JavaScript White Space

- JavaScript ignores multiple spaces. You can add white space to your script to make it more readable.
- The following lines are equivalent: `var person = "Hege";`  
`var person="Hege";`
- A good practice is to put spaces around operators ( `= + - * /` ):

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

BK  
TP.HCM

# 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:

```
document.getElementById( "demo" ).innerHTML = "Hello  
Dolly.";
```



# JavaScript Code Blocks

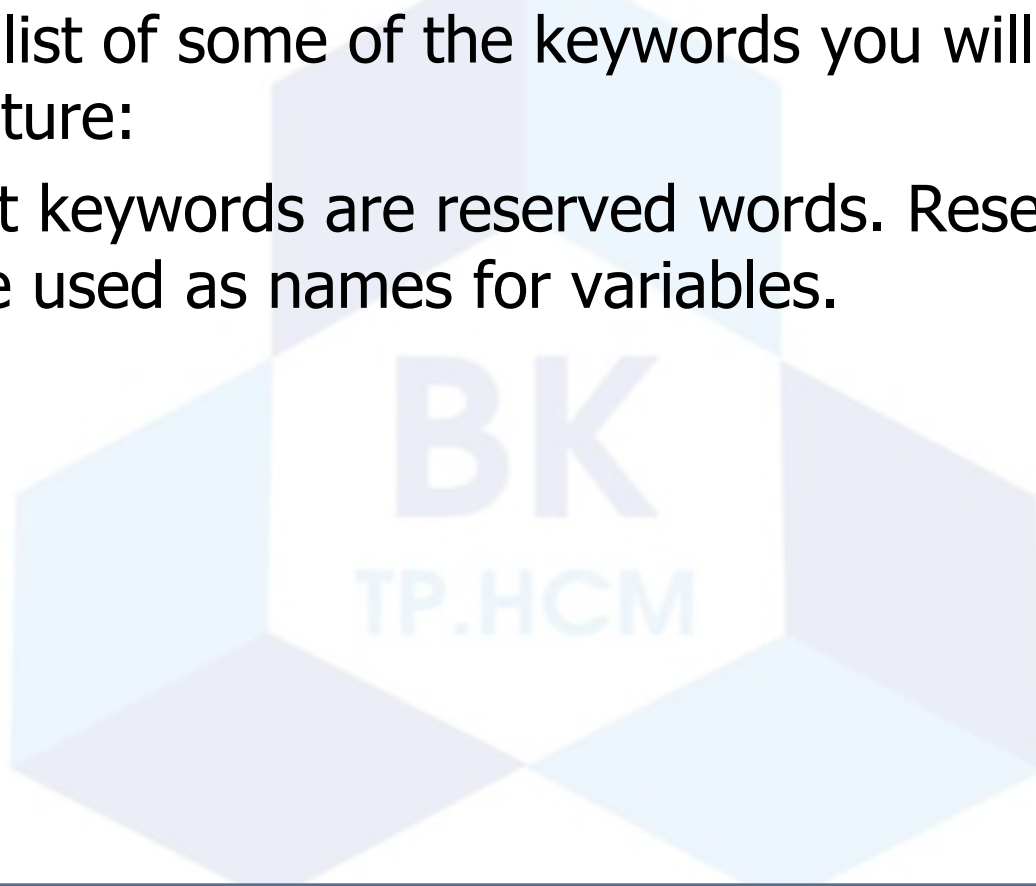
- JavaScript statements can be grouped together in code blocks, inside curly brackets {...}.
- The purpose of code blocks this is to define statements to be executed together.
- One place you will find statements grouped together in blocks, are in JavaScript functions:

```
function myFunction() {  
    document.getElementById("demo").innerHTML = "Hello Dolly.";  
    document.getElementById("myDIV").innerHTML = "How are you?";  
}
```

# JavaScript Keywords

---

- JavaScript statements often start with a keyword to identify the JavaScript action to be performed.
- Here is a list of some of the keywords you will learn about in this lecture:
- JavaScript keywords are reserved words. Reserved words cannot be used as names for variables.



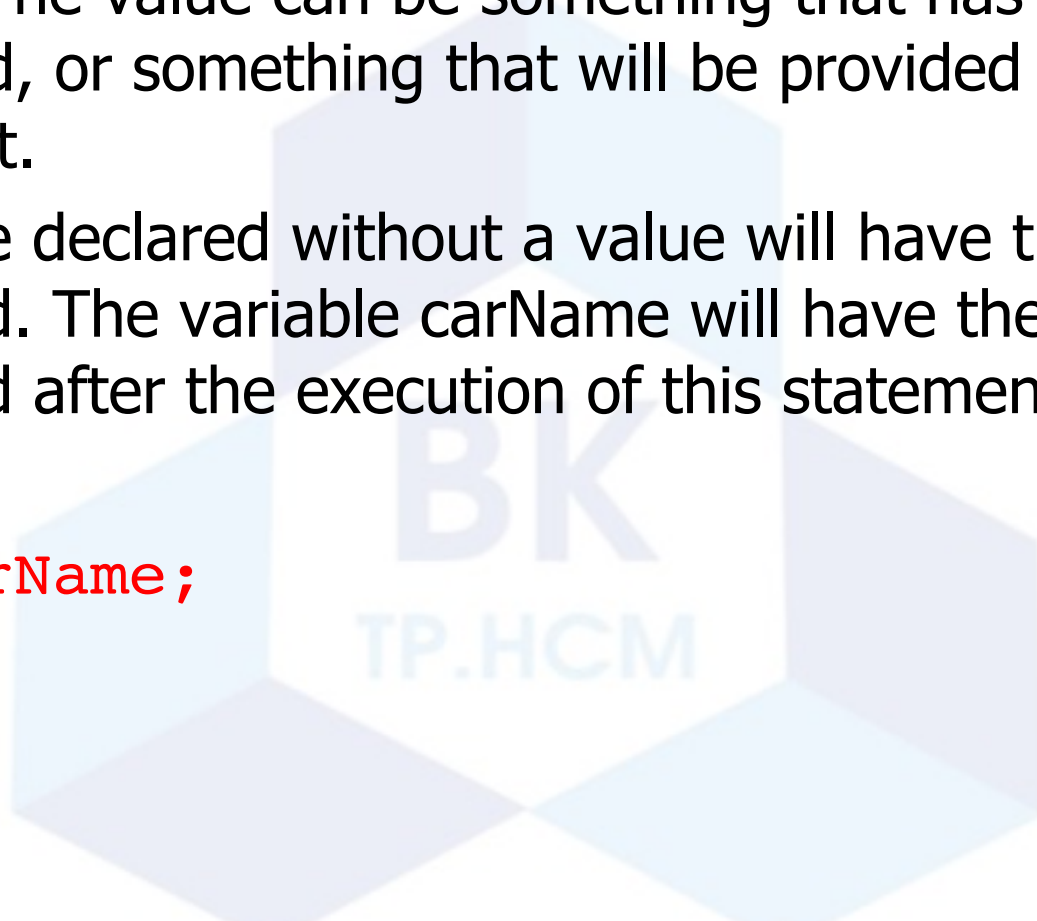
# JavaScript Keywords

Keyword	Description
break	Terminates a switch or a loop
continue	Jumps out of a loop and starts at the top
debugger	Stops the execution of JavaScript, and calls (if available) the debugging function
do ... while	Executes a block of statements, and repeats the block, while a condition is true
for	Marks a block of statements to be executed, as long as a condition is true
function	Declares a function
if ... else	Marks a block of statements to be executed, depending on a condition
return	Exits a function
switch	Marks a block of statements to be executed, depending on different cases
try ... catch	Implements error handling to a block of statements
var	Declares a variable

# Value = undefined

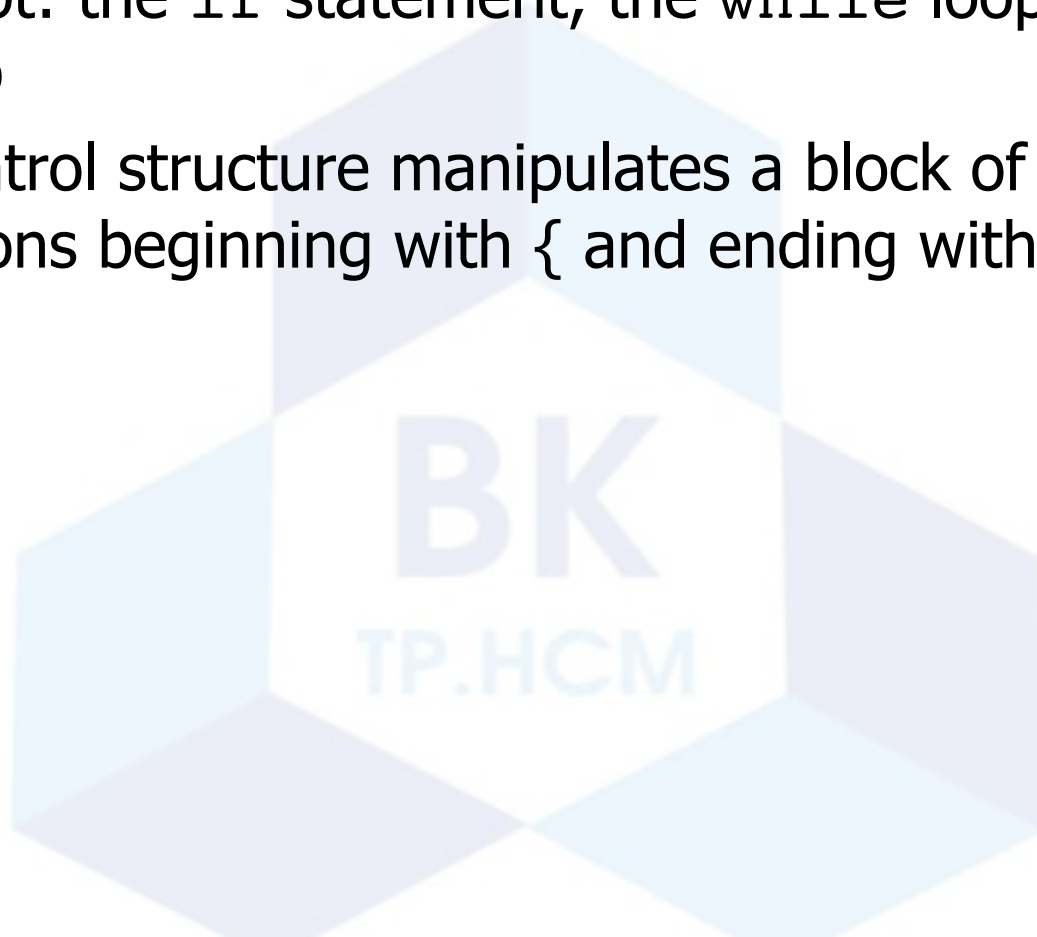
- In computer programs, variables are often declared without a value. The value can be something that has to be calculated, or something that will be provided later, like user input.
- A variable declared without a value will have the value undefined. The variable carName will have the value undefined after the execution of this statement:
- Example

```
var carName;
```



# Control Structures

- There are three basic types of control structures in JavaScript: the `if` statement, the `while` loop, and the `for` loop
- Each control structure manipulates a block of JavaScript expressions beginning with `{` and ending with `}`





# The If Statement

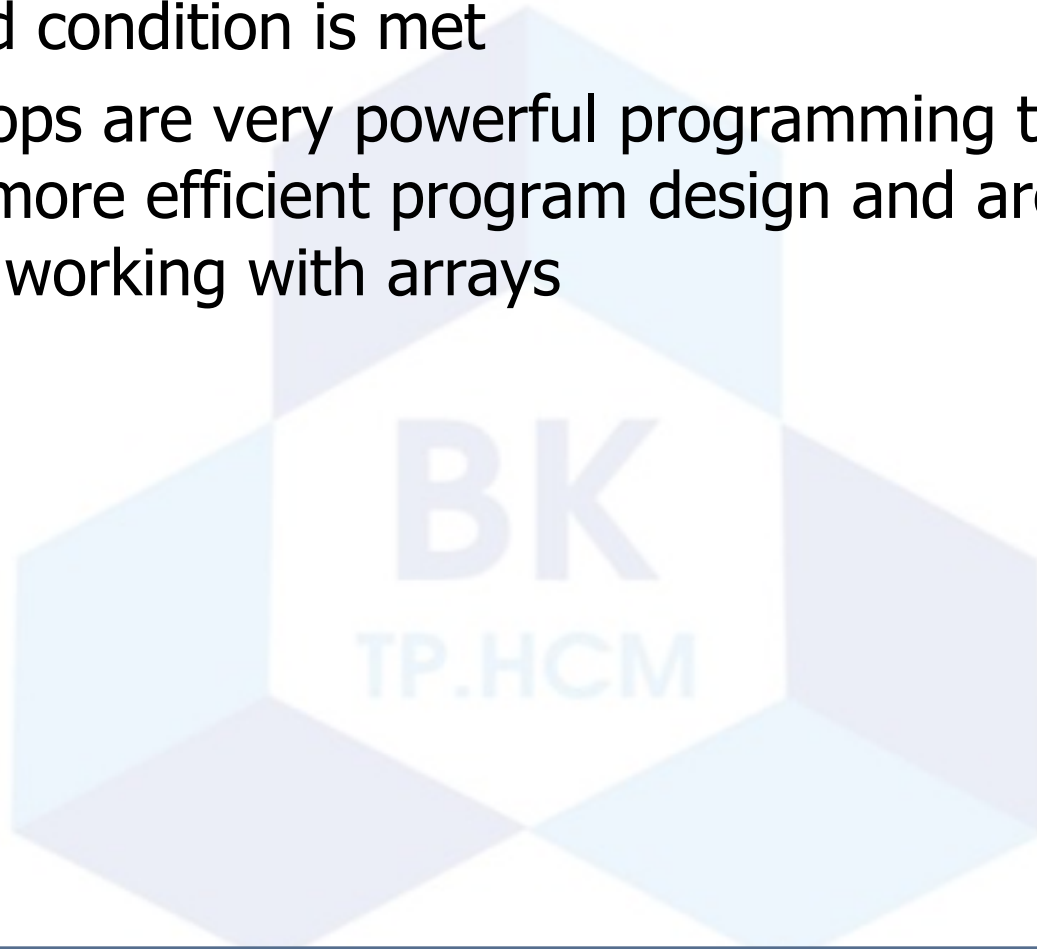
- The `if` statement allows JavaScript programmers to make a decision
- Use an `if` statement whenever you come to a “fork” in the program

```
If ( x == 10)
{
    y = x*x;
}
else
{
    x = 0;
}
```

# Repeat Loops

---

- A repeat loop is a group of statements that is repeated until a specified condition is met
- Repeat loops are very powerful programming tools; They allow for more efficient program design and are ideally suited for working with arrays



# The While Loop

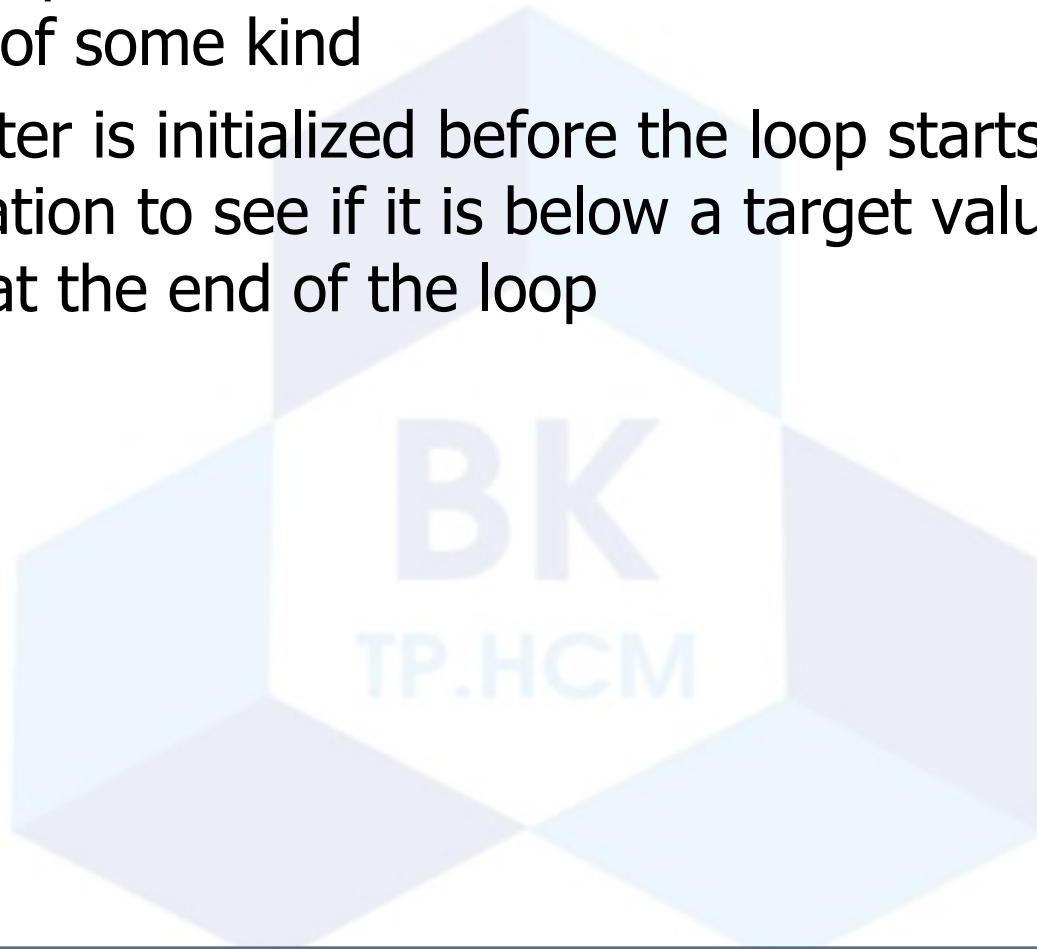
- The while loop is used to execute a block of code while a certain **condition** is true

```
count = 0;  
while (count <= 10) {  
    document.write(count);  
    count++;  
}
```

# The For Loop

---

- The for loop is used when there is a need to have a **counter** of some kind
- The counter is initialized before the loop starts, tested after each iteration to see if it is below a target value, and finally updated at the end of the loop



# Example: For Loop

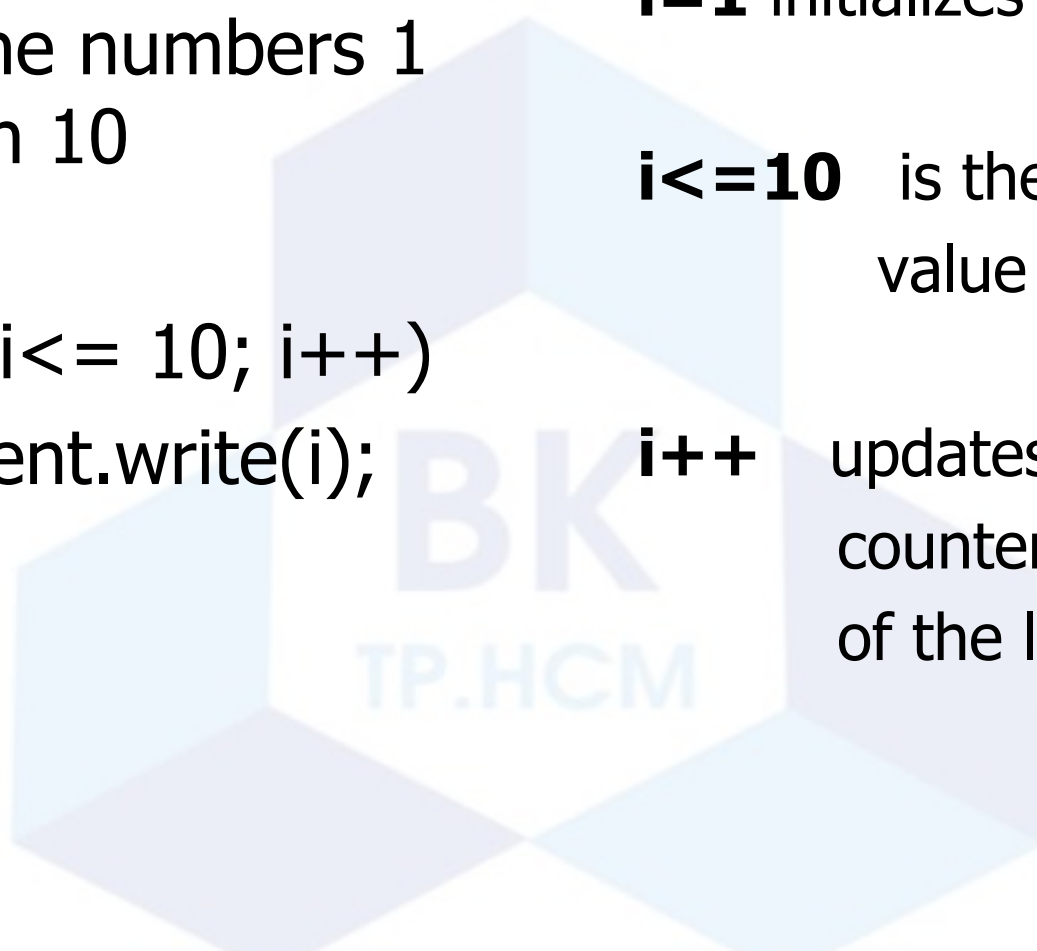
```
// Print the numbers 1  
    through 10
```

```
for (i=1; i<= 10; i++)  
    document.write(i);
```

**i=1** initializes the counter

**i<=10** is the target  
value

**i++** updates the  
counter at the end  
of the loop



# Example: For Loop

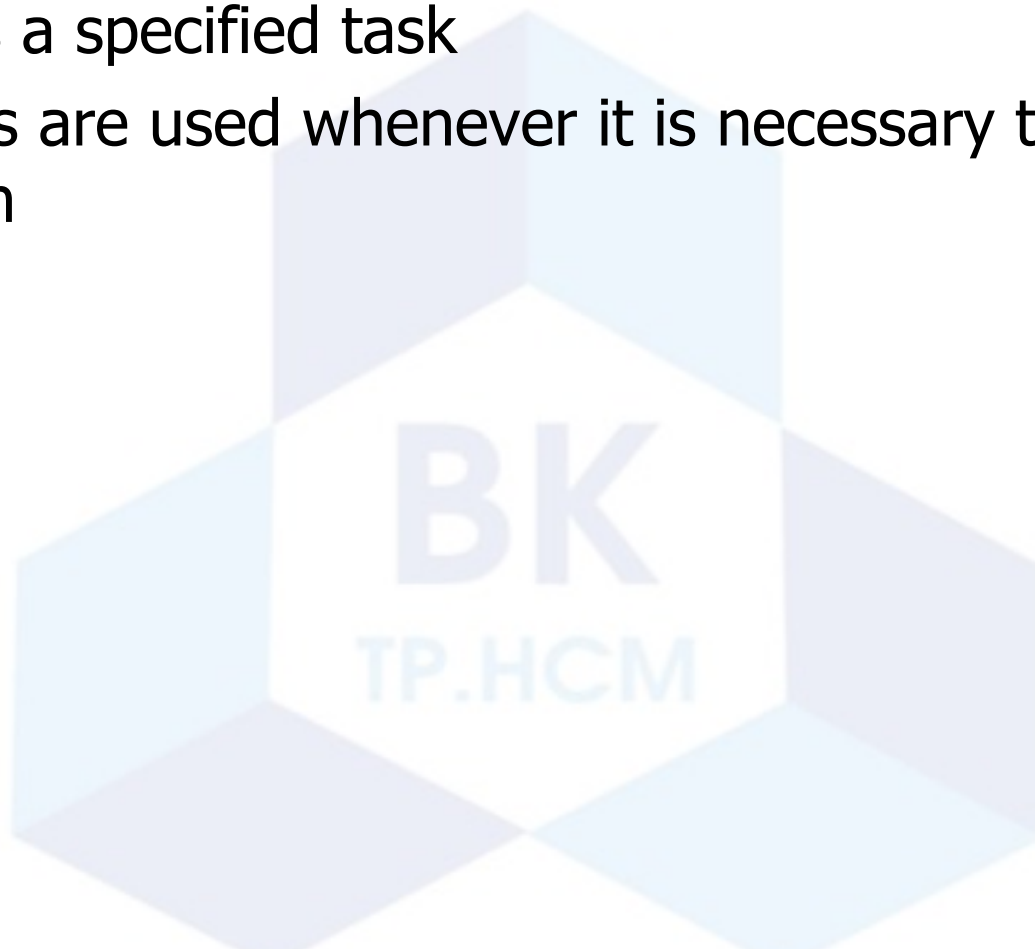
```
<SCRIPT>
document.write("1");
document.write("2");
document.write("3");
document.write("4");
document.write("5");
</SCRIPT>
```

```
<SCRIPT>
for (i=1; i<=5; i++) {
    document.write(i);
}
</SCRIPT>
```

# Functions

---

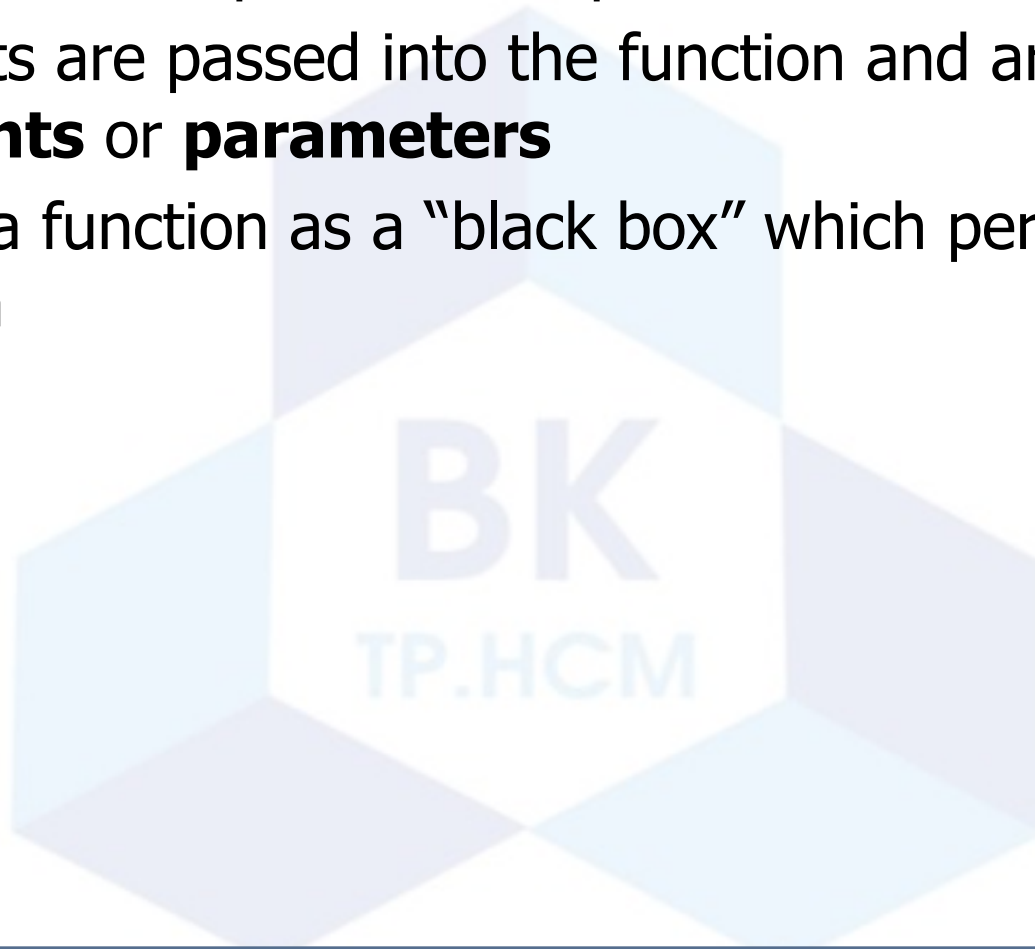
- Functions are a collection of JavaScript statement that performs a specified task
- Functions are used whenever it is necessary to repeat an operation



# Functions

---

- Functions have inputs and outputs
- The inputs are passed into the function and are known as **arguments** or **parameters**
- Think of a function as a “black box” which performs an operation

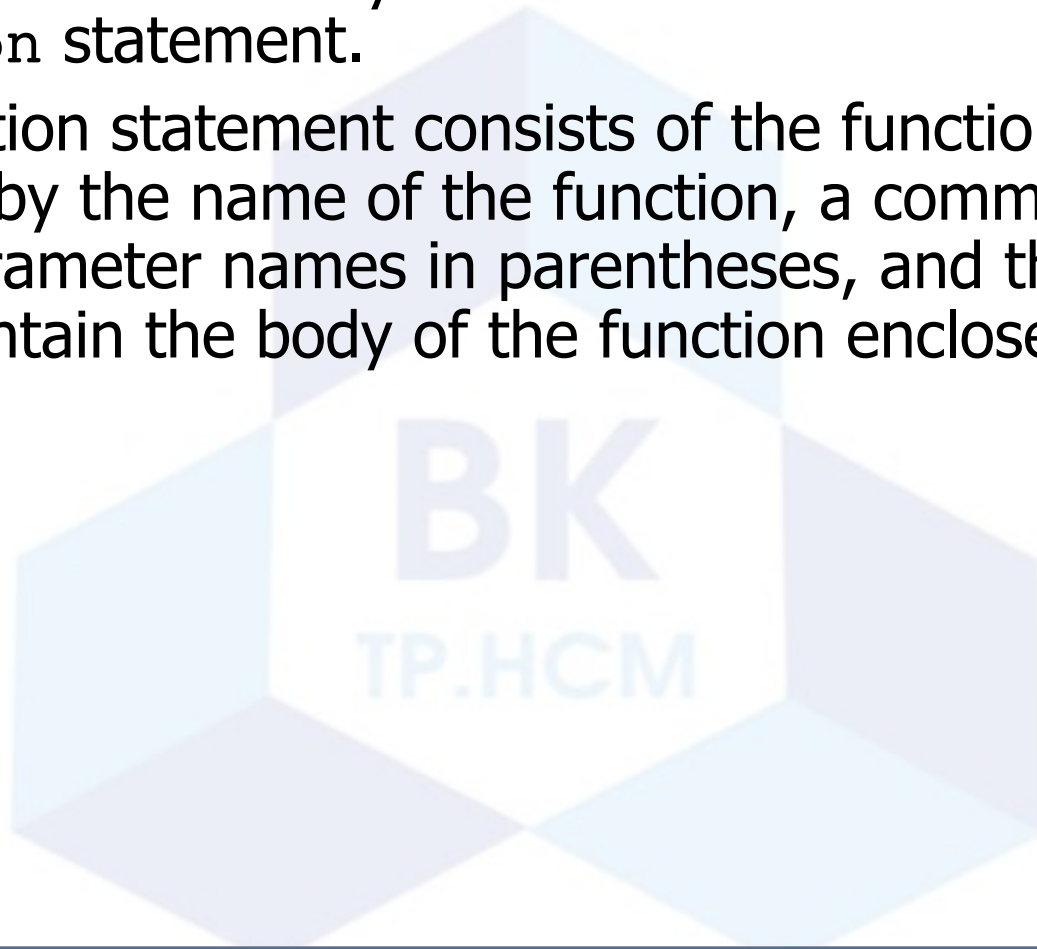




# Defining Functions

---

- The most common way to define a function is with the `function` statement.
- The function statement consists of the function keyword followed by the name of the function, a comma-separated list of parameter names in parentheses, and the statements which contain the body of the function enclosed in curly braces



# Example: Function

```
function square(x) {  
    return x*x;  
}
```

**Name of Function:**  
square

**Input/Argument:** x

```
z = 3;  
sqr_z = square(z);
```

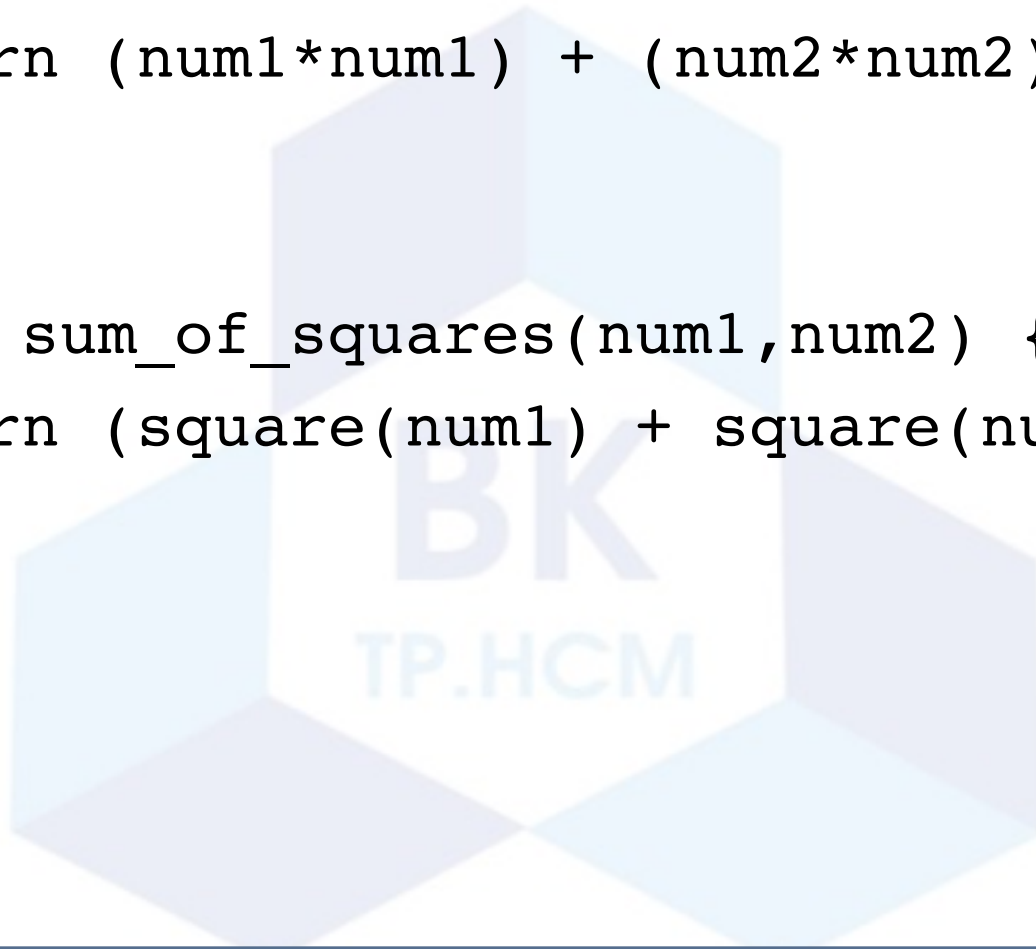
**Output:**  $x*x$

# Example: Function

---

```
function sum_of_squares(num1,num2) {  
    return (num1*num1) + (num2*num2);  
}
```

```
function sum_of_squares(num1,num2) {  
    return (square(num1) + square(num2));  
}
```



# Javascript Events

- Events are actions that can be detected by Javascript
- Every element on a web page has certain events which can trigger Javascript functions
- Often placed within the HTML tag
  - `<tag attribute1 attribute2 onEventName="javascript code;">`
  - `<a href="" onMouseOver="popupFunc();">`
- The set of all events which may occur and the page elements on which they can occur is part of the Document Object Model(DOM) not Javascript
  - Browsers don't necessarily share the same set of events

# Common Javascript Events

Event	Occurs when...	Event Handler
■ click	User clicks on form element or link	onClick
■ change	User changes value of text, textarea, or select element	onChange
■ focus	User gives form element input focus	onFocus
■ blur	User removes input focus from form element	onBlur
■ mouseover	User moves mouse pointer over a link or anchor	onMouseOver
■ mouseout	User moves mouse pointer off of link or anchor	onMouseOut
■ select	User selects form element's input field	onSelect
■ submit	User submits a form	onSubmit
■ resize	User resizes the browser window	onResize
■ load	User loads the page in the Navigator	onLoad
■ unload	User exits the page	onUnload

## 5.2 DOM (Document Object Model)

---



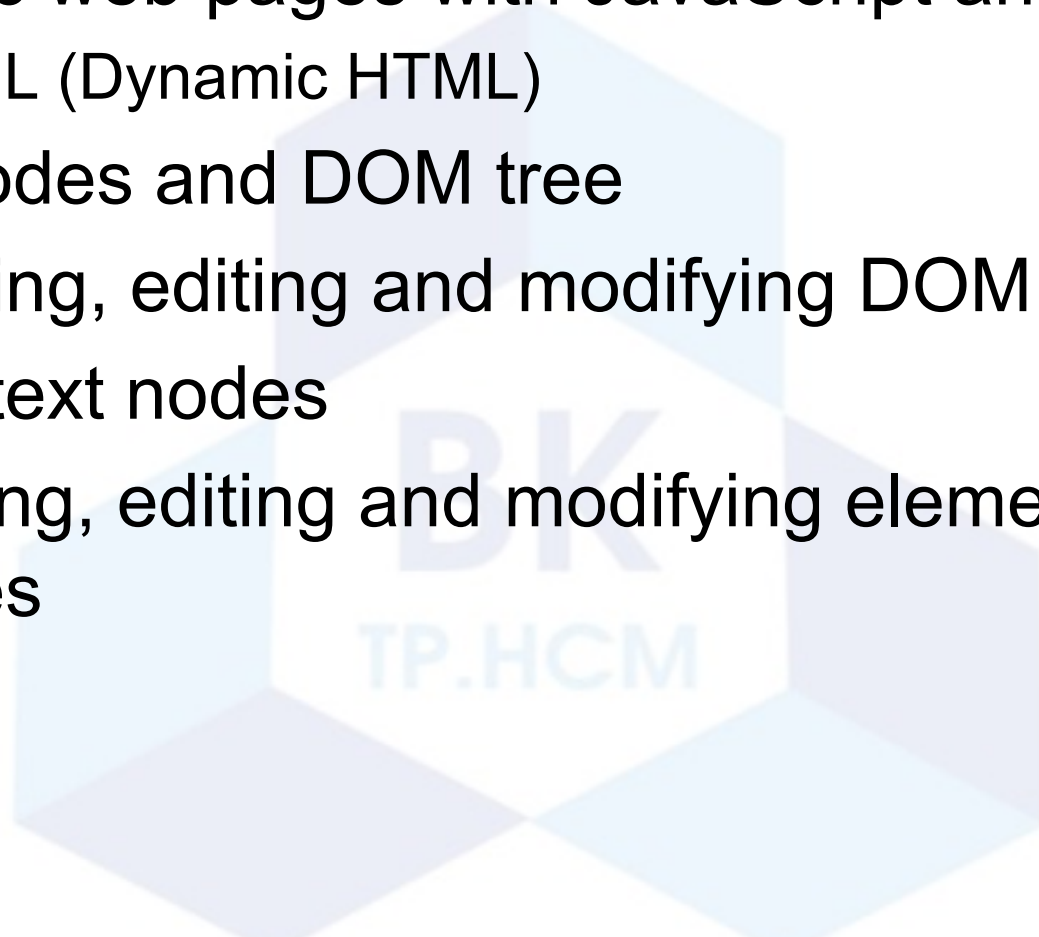
# The Document Object Model

- When a document is loaded in the web browser, a number of objects are created.
  - Most commonly used are window and document
- Window
  - `open()`, `close()`, `alert()`, `confirm()`, `prompt()`
- Document
  - Contains arrays which store all the components of your page
  - You can access and call methods on the components using the arrays
  - An object may also be accessed by its name
    - `document.myform.address.value = "123 Main"`
    - `document.myform.reset()`
  - Can also search for element by name or id
    - `document.getElementById("myelementid")`
    - `document.getElementsByName("myelementname")`

# DOM & DHTML

---

- Dynamic web pages with JavaScript and DOM
  - DHTML (Dynamic HTML)
- DOM nodes and DOM tree
- Traversing, editing and modifying DOM nodes
- Editing text nodes
- Accessing, editing and modifying elements' attributes

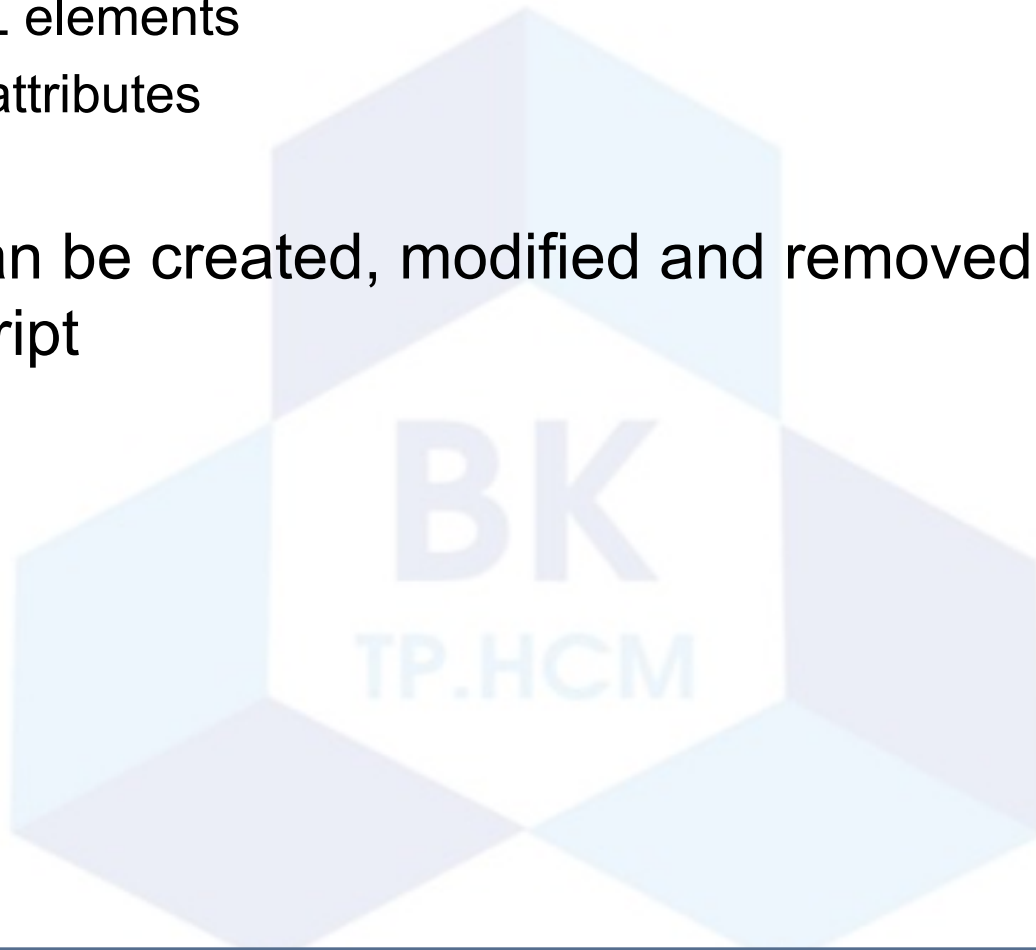




# DOM Concept

---

- DOM makes all components of a web page accessible
  - HTML elements
  - their attributes
  - text
- They can be created, modified and removed with JavaScript



# DOM Objects

- DOM components are accessible as objects or collections of objects
- DOM components form a tree of nodes
  - relationship parent node – children nodes
  - **document** is the root node
- Attributes of elements are accessible as text
- Browsers can show DOM visually as an expandable tree
  - Firebug for Firefox
  - in IE -> Tools -> Developer Tools

# DOM Standards

- W3C [www.w3.org](http://www.w3.org) defines the standards
- DOM Level 3 recommendation
  - [www.w3.org/TR/DOM-Level-3-Core/](http://www.w3.org/TR/DOM-Level-3-Core/)
- DOM Level 2 HTML Specification
  - [www.w3.org/TR/DOM-Level-2-HTML/](http://www.w3.org/TR/DOM-Level-2-HTML/)
  - additional DOM functionality specific to HTML, in particular objects for XHTML elements
- **But**, the developers of web browsers
  - **don't** implement all standards
  - implement some standards **differently**
  - implement some **additional features**

# Accessing Nodes by `id`

- Access to elements by their `id`
  - `document.getElementById(<id>)`
    - returns the element with `id <id>`
  - `id` attribute can be defined in each start tag
    - `div` element with `id` attribute can be used as an root node for a dynamic DOM subtree
    - `span` element with `id` attribute can be used as a dynamic inline element
- The preferred way to access elements

# Other Access Methods

- Access by elements' tag

- there are typically several elements with the same tag
- `document.getElementsByTagName(<tag>)`
  - returns the collection of all elements whose tag is <tag>
  - the collection has a `length` attribute
  - an item in the collection can be reached by its index
- e.g.
  - `var html = document.getElementsByTagName("html")[0];`

- Access by elements' `name` attribute

- several elements can have the same name
- `document.getElementsByName(<name>)`
  - returns the collection of elements with `name` <name>

# Traversing DOM tree

- Traversal through node properties
  - **childNodes** property
    - the value is a collection of nodes
      - has a **length** attribute
      - an item can be reached by its index
    - e.g. `var body = html.childNodes[1];`
  - **firstChild**, **lastChild** properties
  - **nextSibling**, **previousSibling** properties
  - **parentNode** property

# Other Node Properties

- **nodeType** property
  - **ELEMENT\_NODE**: HTML element
  - **TEXT\_NODE**: text within a parent element
  - **ATTRIBUTE\_NODE**: an attribute of a parent element
    - attributes can be accessed another way
  - **CDATA\_SECTION\_NODE**
    - CDATA sections are good for unformatted text
- **nodeName** property
- **nodeValue** property
- **attributes** property
- **innerHTML** property
  - not standard, but implemented in major browsers
  - very useful
- **style** property
  - object whose properties are all style attributes, e.g., those defined in CSS

# Accessing JS Object's Properties

- There are two different syntax forms to access object's properties in JS (
  - `<object>.<property>`
    - dot notation, e.g., `document.nodeType`
  - `<object>[<property-name>]`
    - brackets notation, e.g., `document["nodeType"]`
    - this is used in `for-in` loops
- this works for properties of DOM objects, too



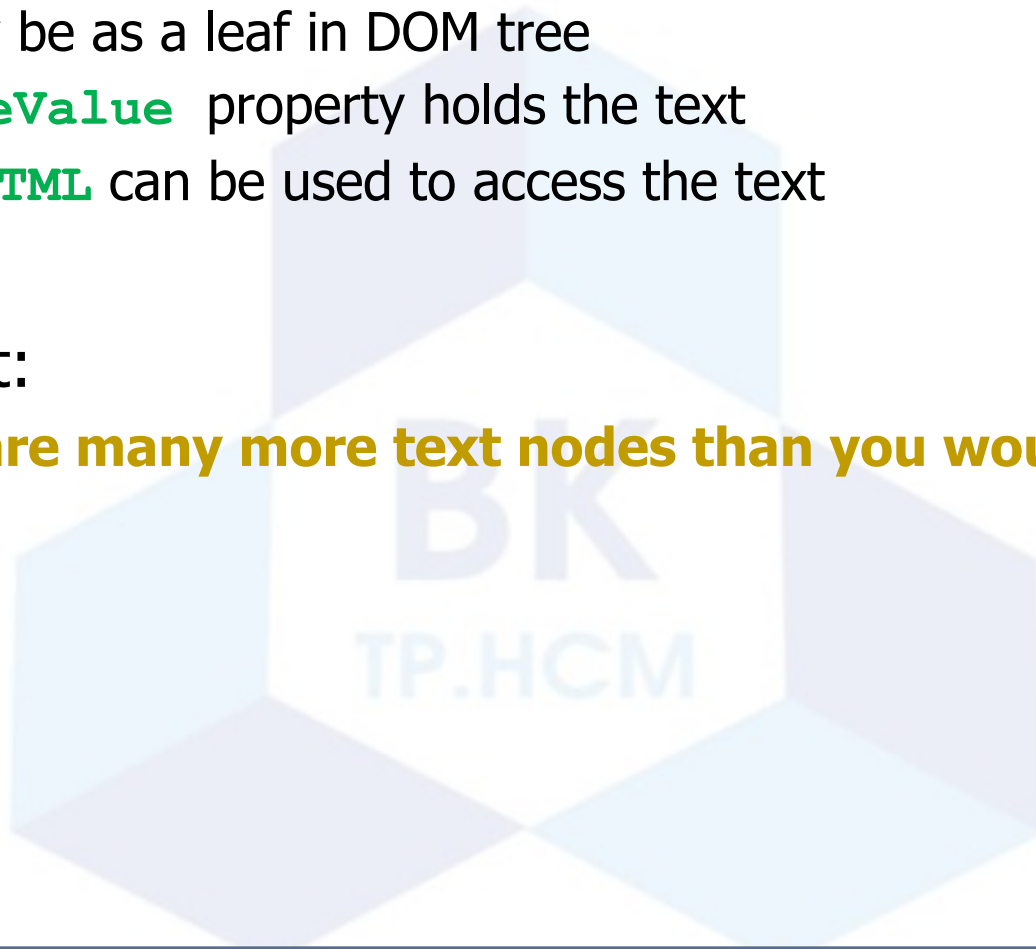
# Attributes of Elements

- Access through **attributes** property
  - **attributes** is an array
  - has a **length** attribute
  - an item can be reached by its index
  - an item has the properties **name** and **value**
  - e.g.
    - `var src = document.images[0].attributes[0].value;`
- Access through function **getAttribute(<name>)**
  - returns the value of attribute **<name>**
  - e.g.
    - `var src = document.images[0].getAttribute("src");`

# Text Nodes

---

- Text node
  - can only be as a leaf in DOM tree
  - it's `nodeValue` property holds the text
  - `innerHTML` can be used to access the text
- Watch out:
  - **There are many more text nodes than you would expect!**



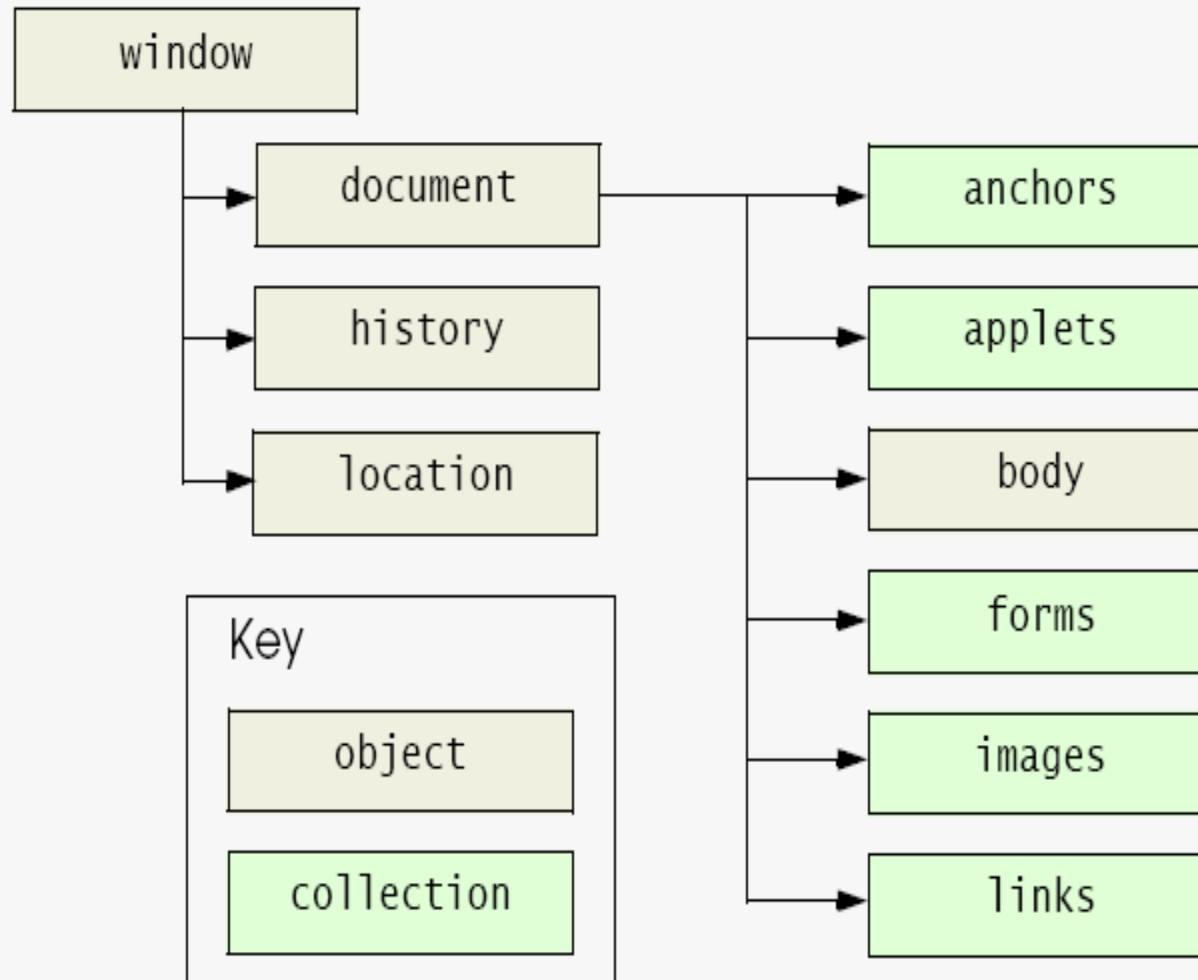
# Modifying DOM Structure

- **document.createElement(<tag>)**
  - creates a new DOM element node, with <tag> tag.
  - the node still needs to be inserted into the DOM tree
- **document.createTextNode(<text>)**
  - creates a new DOM text with <text>
  - the node still needs to be inserted into the DOM tree
- **<parent>.appendChild(<child>)**
  - inserts <child> node behind all existing children of <parent> node
- **<parent>.insertBefore(<child>, <before>)**
  - inserts <child> node before <before> child within <parent> node
- **<parent>.replaceChild(<child>, <instead>)**
  - replaces <instead> child by <child> node within <parent> node
- **<parent>.removeChild(<child>)**
  - removes <child> node from within <parent> node

# Modifying Node Attributes

- `<node>.setAttribute(<name>,<value>)`
  - sets the value of attribute `<name>` to `<value>`
  - e.g.
    - `document.images[0].setAttribute("src","keiki.jpg");`
- That's the standard
  - but it doesn't work in IE, there you have to use
    - `setAttribute(<name=value>)`
  - e.g.
    - `document.images[0].setAttribute("src=\"keiki.jpg\"");`

# W3C Document Object Model



# Special DOM Objects

- **window**
  - the browser window
  - new popup **windows** can be opened
- **document**
  - the current web page inside the **window**
- **body**
  - **<body>** element of the **document**
- **history**
  - sites that the user visited
  - makes it possible to go back and forth using scripts
- **location**
  - URL of the **document**
  - setting it goes to another page

# Tài Liệu Tham Khảo

---

- [1] Stepp, Miller, Kirst. Web Programming Step by Step. ( 1st Edition, 2009) Companion Website:  
<http://www.webstepbook.com/>
- [2] W3Schools,  
<http://www.w3schools.com/html/default.asp>

