SWEN1005

MOBILE WEB PROGRAMMING

Session Seven

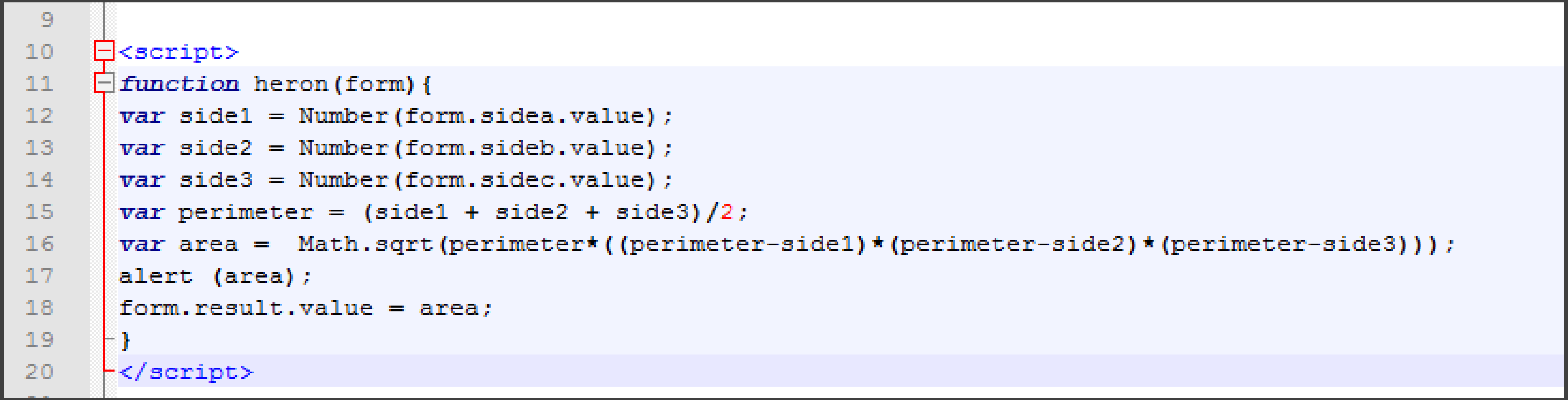
JAVASCRIPT PROGRAMMING CONTROL FLOW

What control flow?

- Sequential
- Selective
- Repetitive

Sequential control

- The default control flow of your JavaScript
- That is, your code will be interpreted and executed line by line in a sequential manner by default



```
<script>
function heron(form){
var side1 = Number(form.sidea.value);
var side2 = Number(form.sideb.value);
var side3 = Number(form.sidec.value);
var perimeter = (side1 + side2 + side3)/2;
var area =  Math.sqrt(perimeter*((perimeter-side1)*(perimeter-side2)*(perimeter-side3)));
alert (area);
form.result.value = area;
}
</script>
```

Selective

- Also called decision making structures or conditional statements
- There are two types in JavaScript:
  - If (condition) then …
  - Switch … case

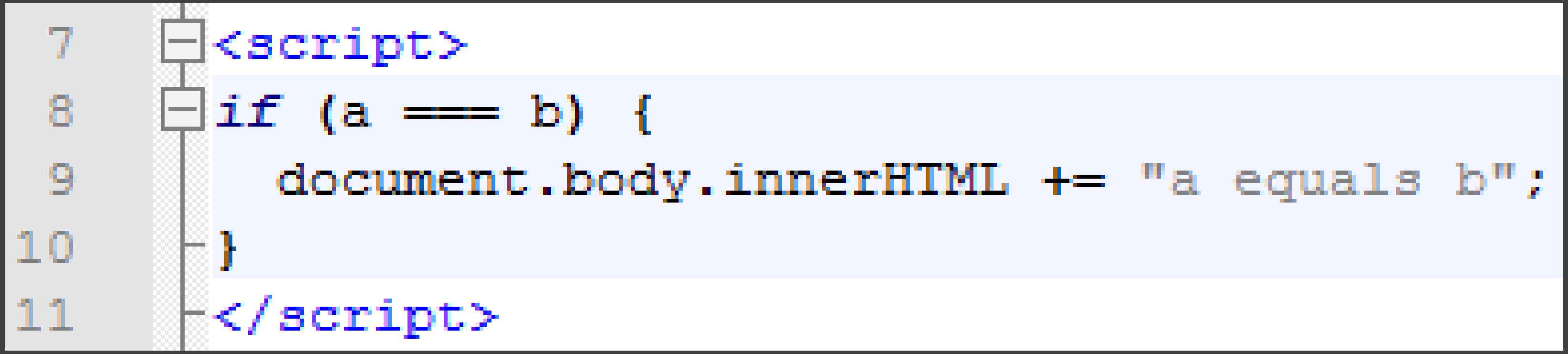
Comparison operators

```
var foo = 1;
var bar = 0;
var baz = '1';
var bim = 2;
foo == bar ; // returns false
foo != bar ; // returns true
foo == baz ; // returns true ; careful!
foo === baz ; // returns false (equal value and equal type)
foo !== baz ; // returns true (not equal value or not equal type)
foo === parseInt( baz ); // returns true
foo > bim ; // returns false
bim > baz ; // returns true
foo <= baz ; // returns true
```

Selective – If statements

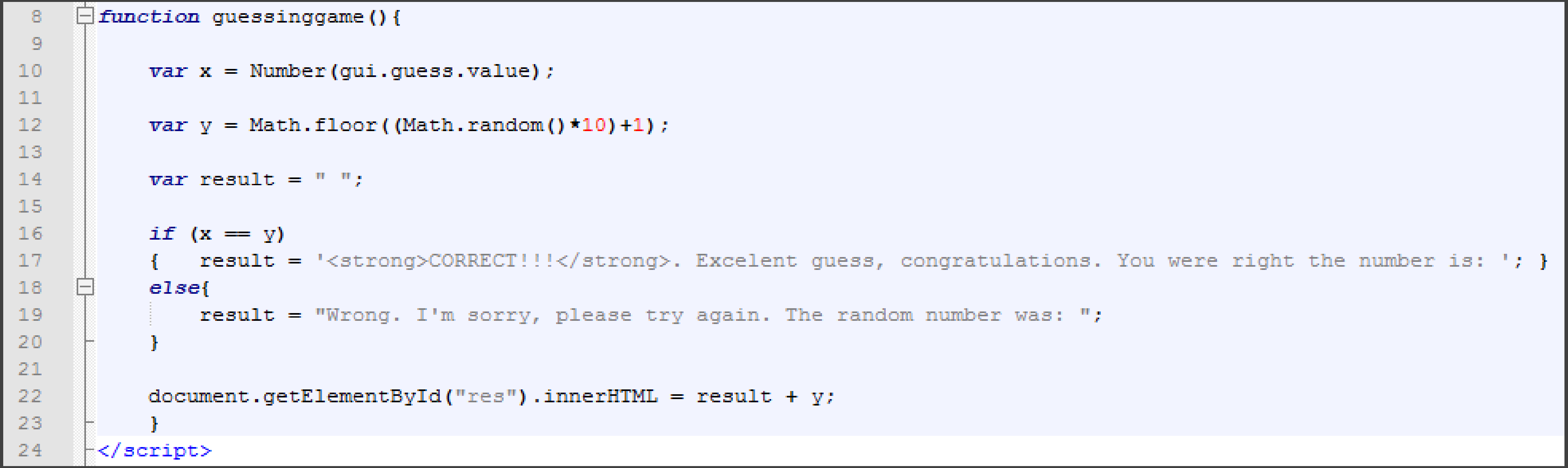
- These maybe
  - Single
  - Double
  - Cascading
  - Nested
  - Multiple

Selective – If statements - single



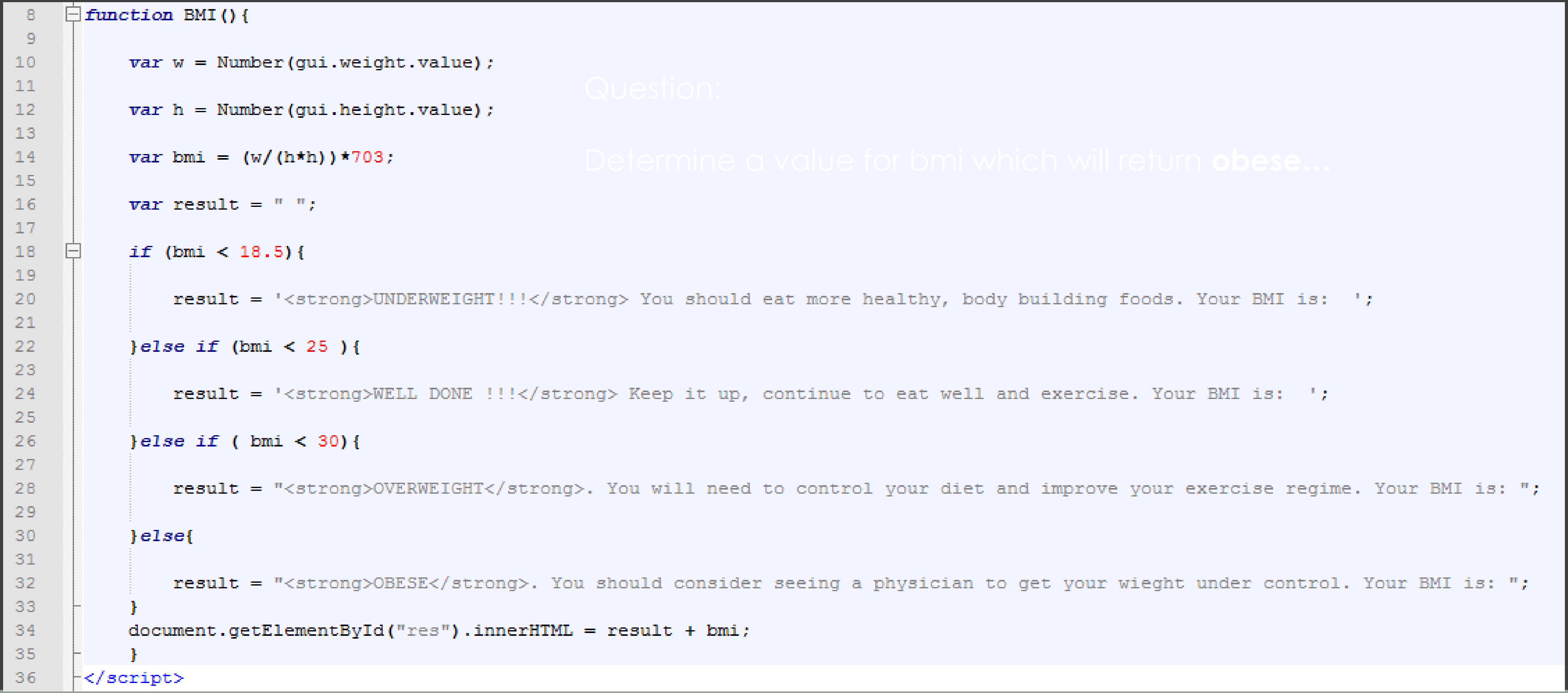
```
<script>
if (a === b) {
  document.body.innerHTML += "a equals b";
}
</script>
```

Selective – If statements - double



```
function guessinggame(){

    var x = Number(gui.guess.value);

    var y = Math.floor((Math.random()*10)+1);

    var result = " ";

    if (x == y)
    {   result = '<strong>CORRECT!!!</strong>. Excelent guess, congratulations. You were right the number is: '; }
    else{
        result = "Wrong. I'm sorry, please try again. The random number was: ";
    }

    document.getElementById("res").innerHTML = result + y;
    }
</script>
```

Selective – If statements - cascading

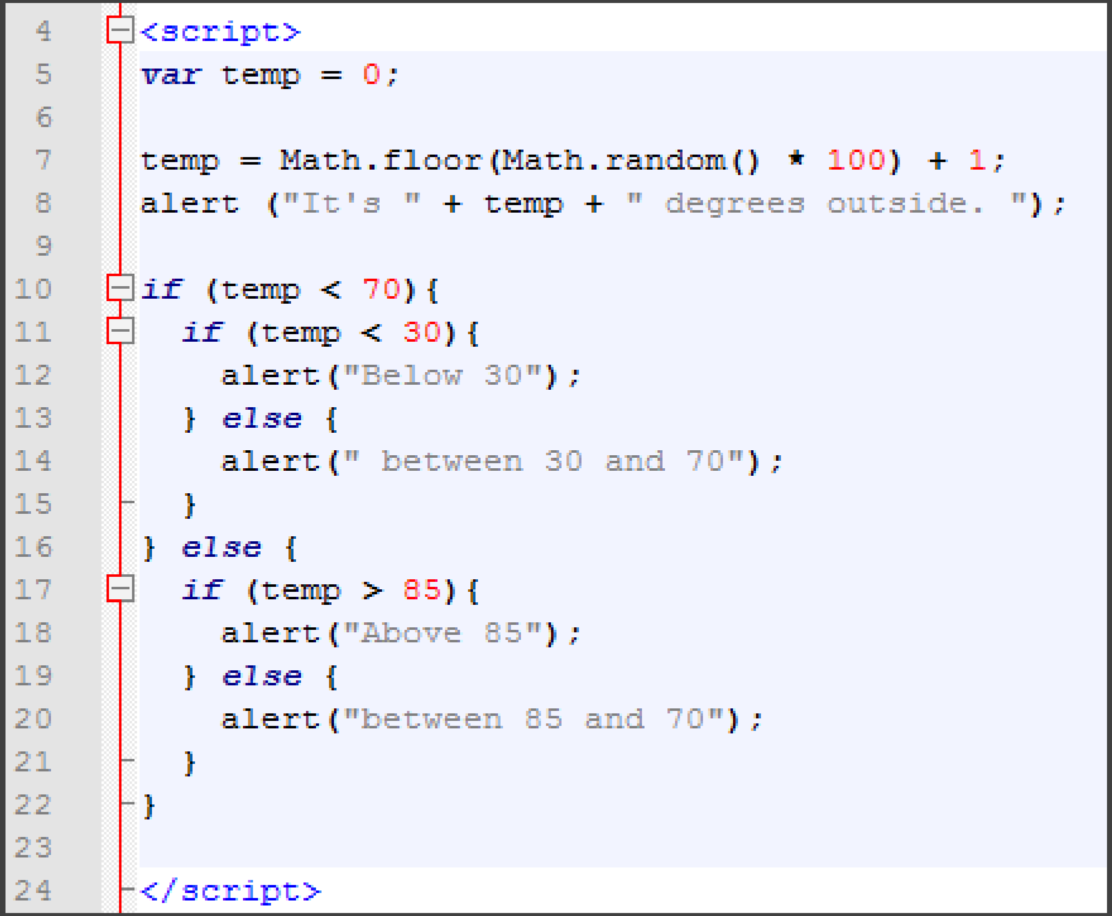


```
function BMI(){

    var w = Number(gui.weight.value);

    var h = Number(gui.height.value);

    var bmi = (w/(h*h))*703;

    var result = " ";

    if (bmi < 18.5){

        result = '<strong>UNDERWEIGHT!!!</strong> You should eat more healthy, body building foods. Your BMI is:  ';

    }else if (bmi < 25 ){

        result = '<strong>WELL DONE !!!</strong> Keep it up, continue to eat well and exercise. Your BMI is:  ';

    }else if ( bmi < 30){

        result = "<strong>OVERWEIGHT</strong>. You will need to control your diet and improve your exercise regime. Your BMI is: ";

    }else{

        result = "<strong>OBESE</strong>. You should consider seeing a physician to get your wieght under control. Your BMI is: ";
    }
    document.getElementById("res").innerHTML = result + bmi;
    }
</script>
```

Question:

Determine a value for bmi which will return **obese…**

Selective – If statements - nested



```
<script>
var temp = 0;

temp = Math.floor(Math.random() * 100) + 1;
alert ("It's " + temp + " degrees outside. ");

if (temp < 70){
  if (temp < 30){
    alert("Below 30");
  } else {
    alert(" between 30 and 70");
  }
} else {
  if (temp > 85){
    alert("Above 85");
  } else {
    alert("between 85 and 70");
  }
}

</script>
```

Question:

Determine the output if **temp** is 71

Selective – If statements – multiple

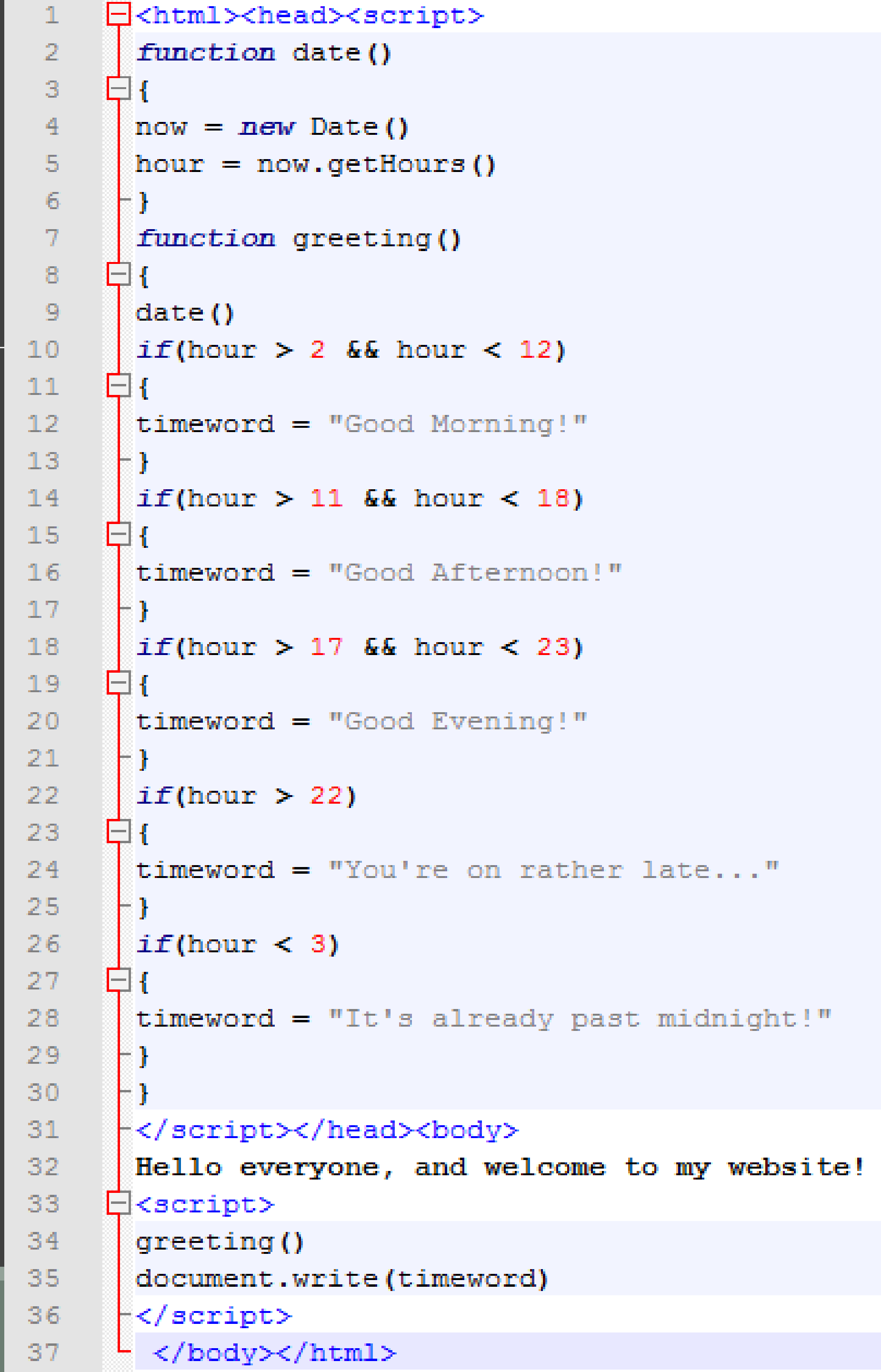
Question:

Determine the output if **the time is:**

2:31

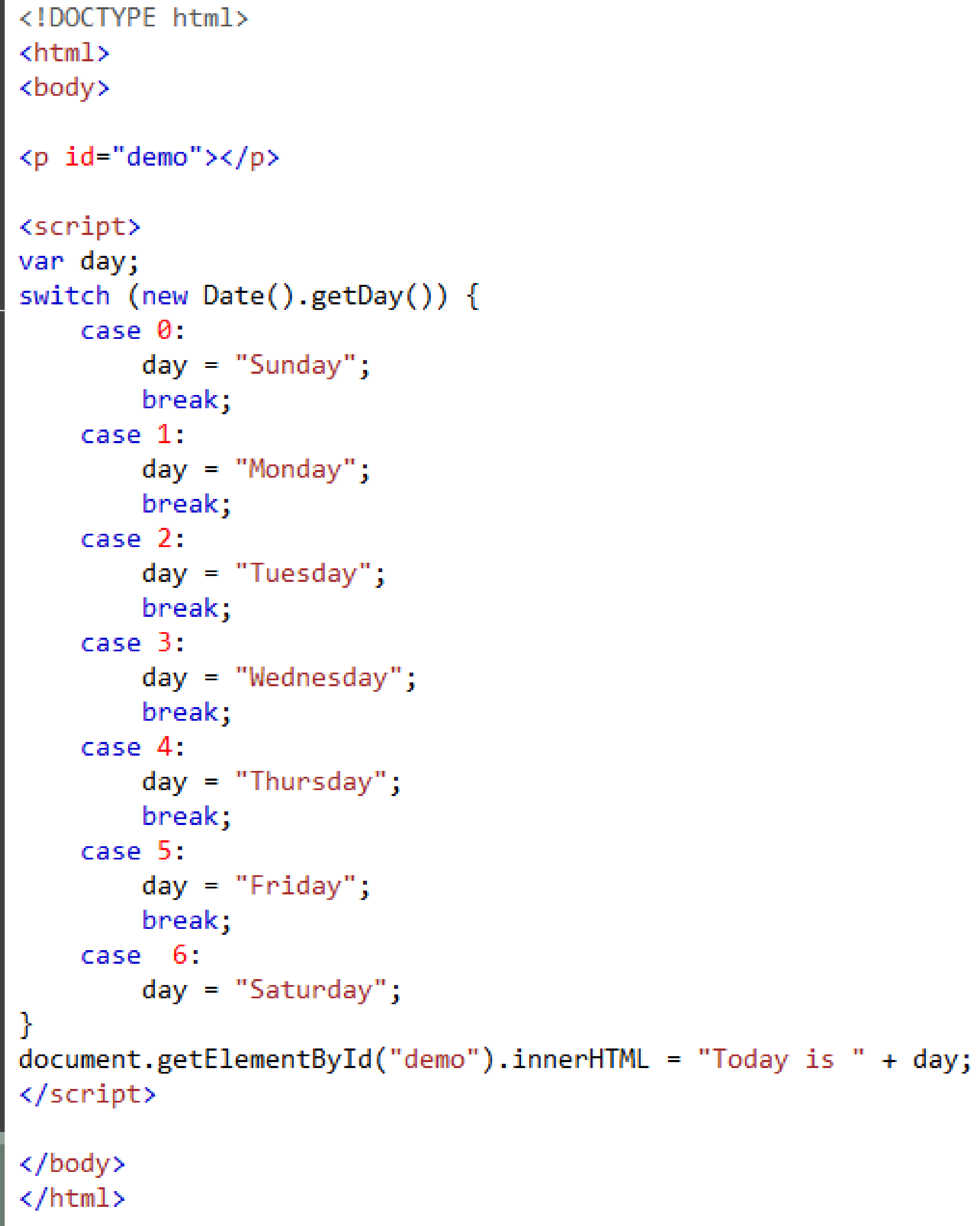
23:30

14:36



```
<html><head><script>
function date()
{
now = new Date()
hour = now.getHours()
}
function greeting()
{
date()
if(hour > 2 && hour < 12)
{
timeword = "Good Morning!"
}
if(hour > 11 && hour < 18)
{
timeword = "Good Afternoon!"
}
if(hour > 17 && hour < 23)
{
timeword = "Good Evening!"
}
if(hour > 22)
{
timeword = "You're on rather late..."
}
if(hour < 3)
{
timeword = "It's already past midnight!"
}
}
</script></head><body>
Hello everyone, and welcome to my website!
<script>
greeting()
document.write(timeword)
</script>
 </body></html>
```

Selective – Switch… case

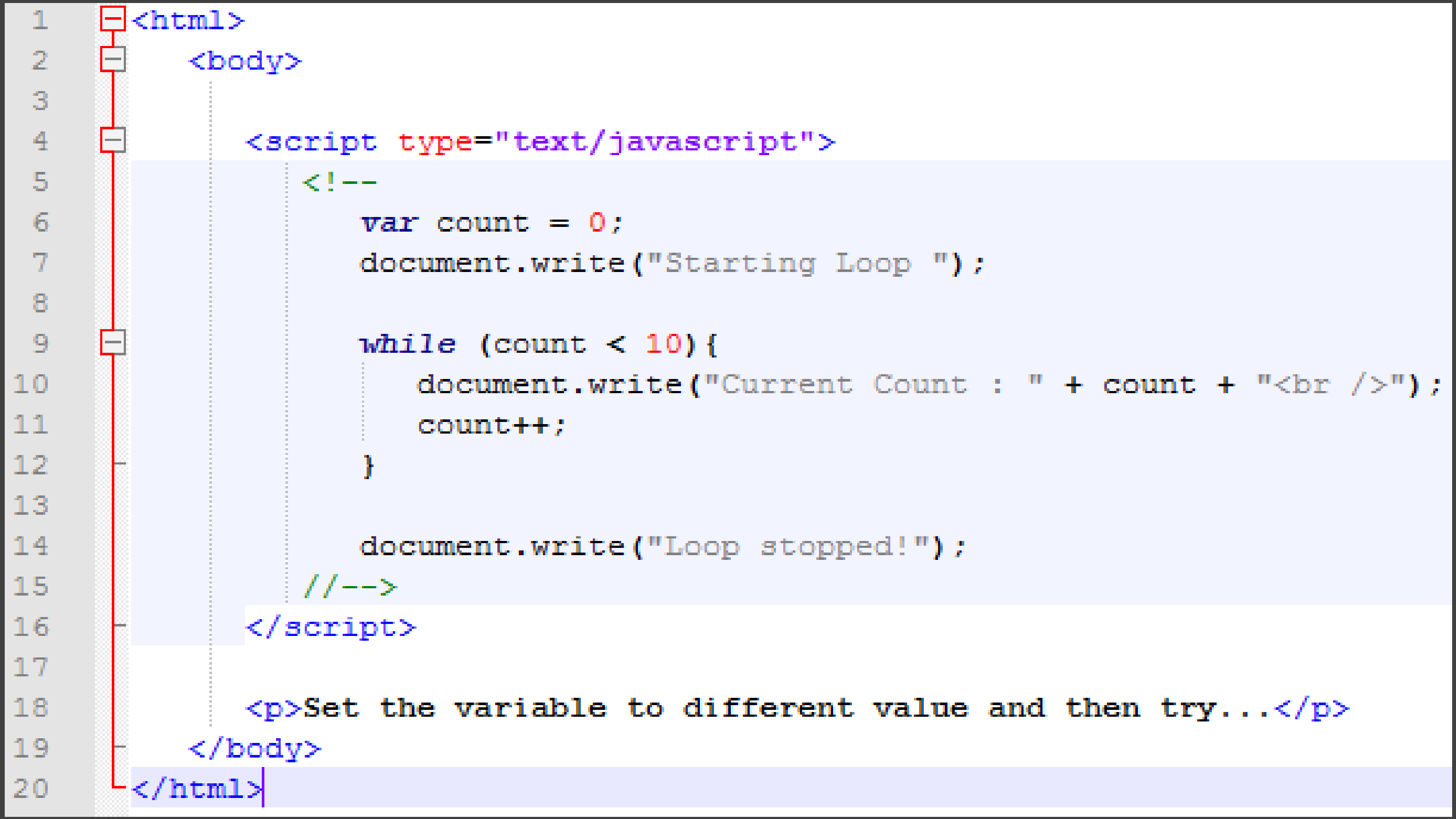


```
<!DOCTYPE html>
<html>
<body>

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

<script>
var day;
switch (new Date().getDay()) {
    case 0:
        day = "Sunday";
        break;
    case 1:
        day = "Monday";
        break;
    case 2:
        day = "Tuesday";
        break;
    case 3:
        day = "Wednesday";
        break;
    case 4:
        day = "Thursday";
        break;
    case 5:
        day = "Friday";
        break;
    case  6:
        day = "Saturday";
}
document.getElementById("demo").innerHTML = "Today is " + day;
</script>

</body>
</html>
```

Repetitive

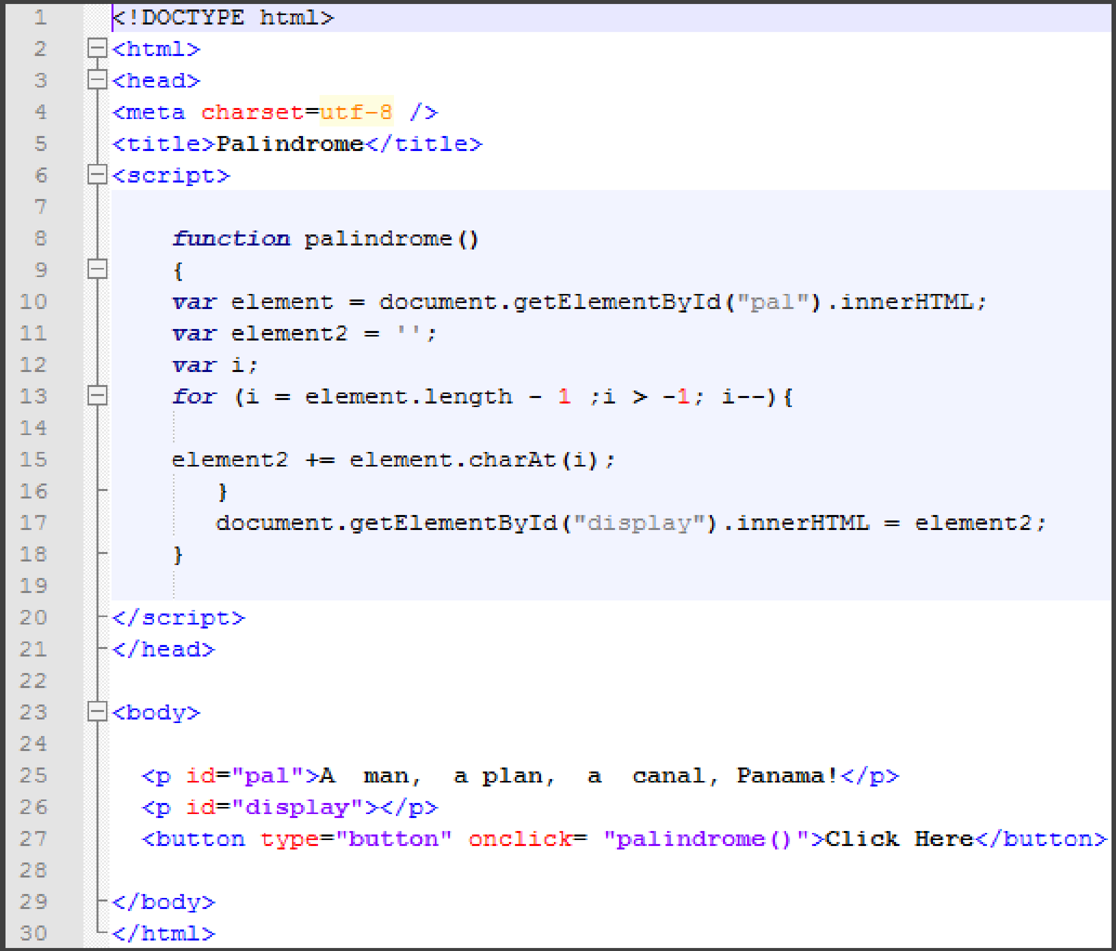
- Also called looping or iterative structures
- There are four generic types:
  - Top-tested
  - Bottom-tested
  - Sentinel (user controlled)
  - Finite (counter controlled)

Repetitive - Top-tested - Finite



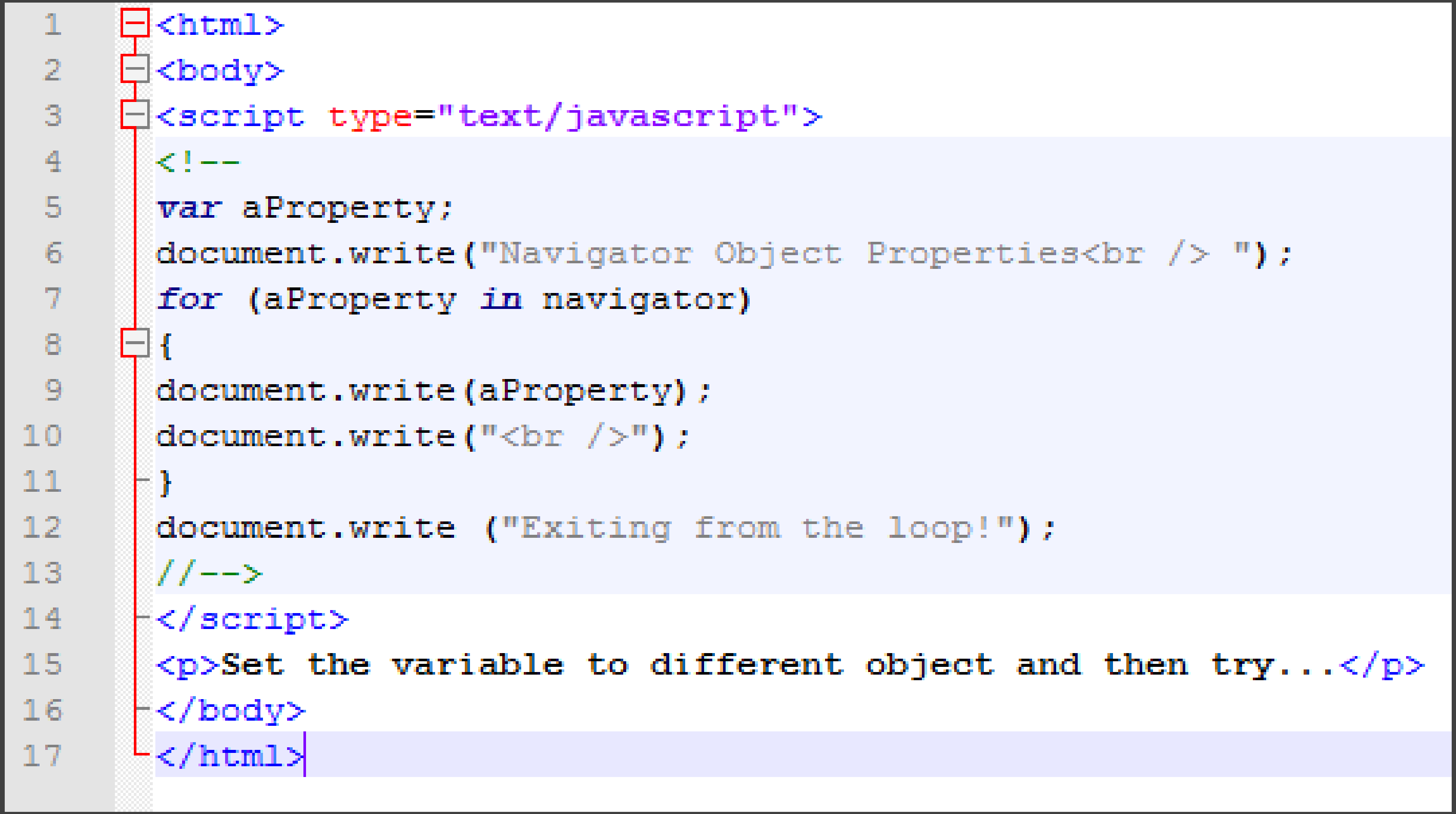
```
<html>
    <body>

        <script type="text/javascript">
            <!--
                var count = 0;
                document.write("Starting Loop ");

                while (count < 10){
                    document.write("Current Count : " + count + "<br />");
                    count++;
                }

                document.write("Loop stopped!");
            //-->
        </script>

        <p>Set the variable to different value and then try...</p>
    </body>
</html>
```

Repetitive - Top-tested - Finite



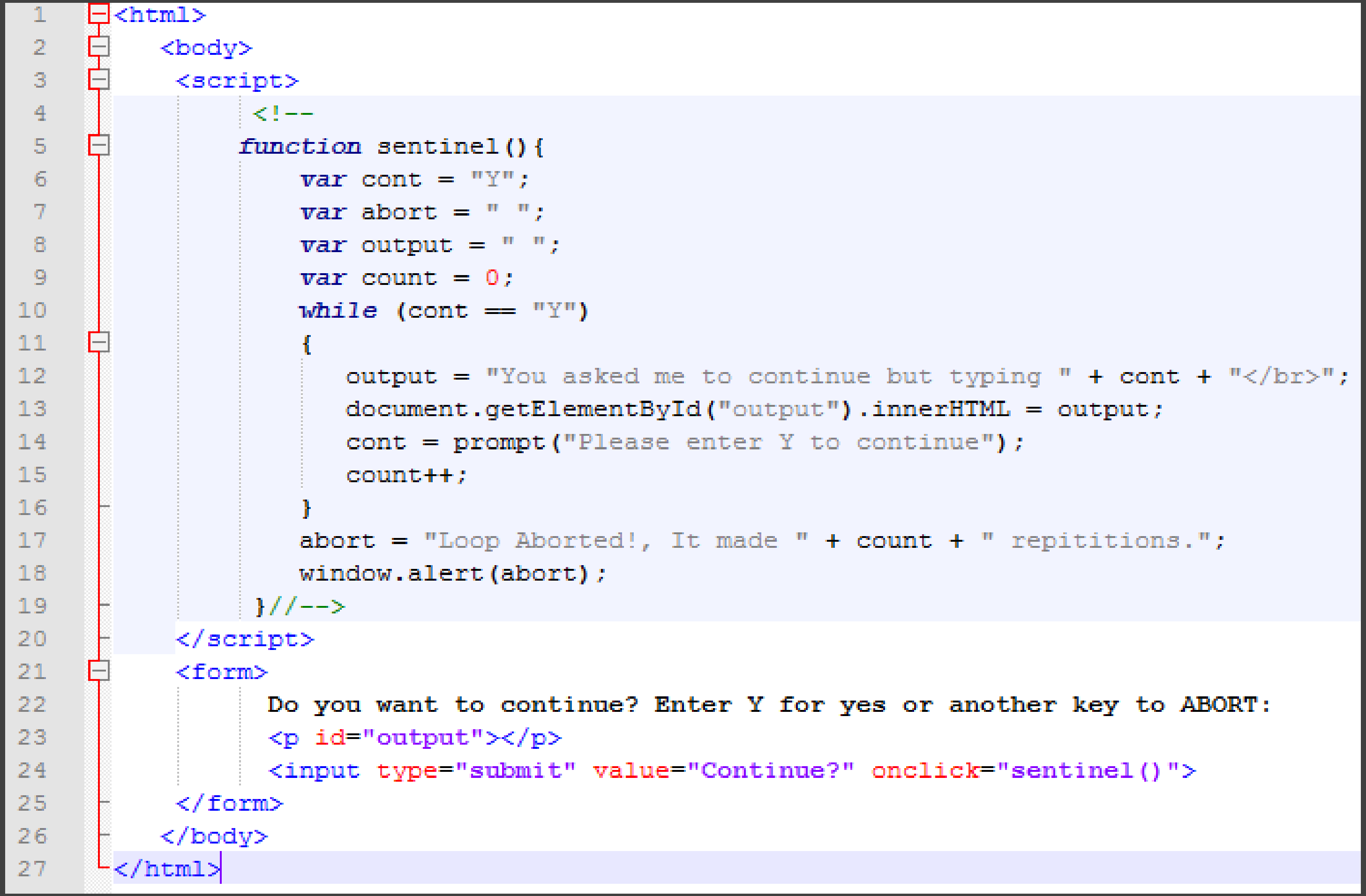
```
<!DOCTYPE html>
<html>
<head>
<meta charset=utf-8 />
<title>Palindrome</title>
<script>

    function palindrome()
    {
    var element = document.getElementById("pal").innerHTML;
    var element2 = '';
    var i;
    for (i = element.length - 1 ;i > -1; i--){

    element2 += element.charAt(i);
       }
       document.getElementById("display").innerHTML = element2;
    }

</script>
</head>

<body>

  <p id="pal">A  man,  a plan,  a  canal, Panama!</p>
  <p id="display"></p>
  <button type="button" onclick= "palindrome()">Click Here</button>

</body>
</html>
```

Repetitive - Top-tested – Finite – For…in



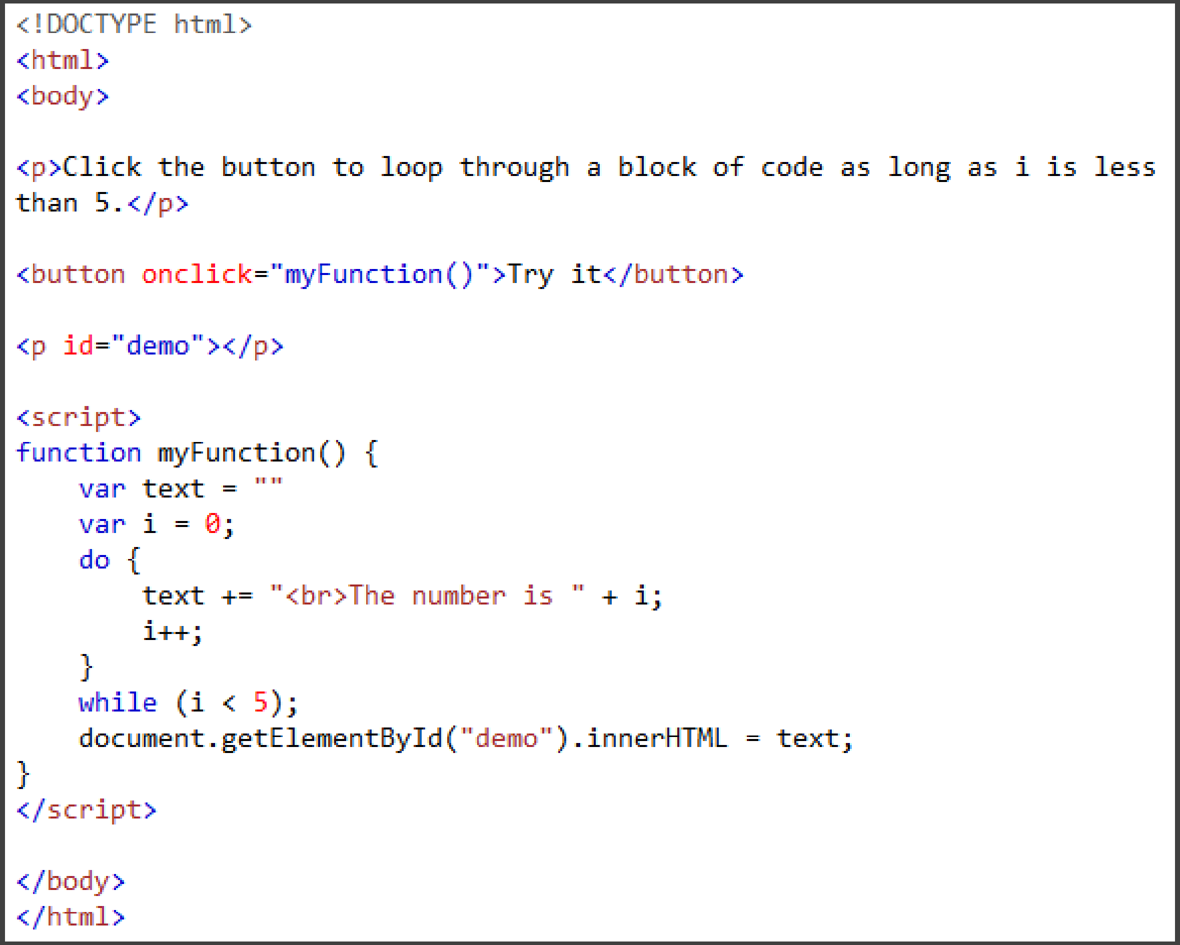
```
<html>
<body>
<script type="text/javascript">
<!--
var aProperty;
document.write("Navigator Object Properties<br /> ");
for (aProperty in navigator)
{
document.write(aProperty);
document.write("<br />");
}
document.write ("Exiting from the loop!");
//-->
</script>
<p>Set the variable to different object and then try...</p>
</body>
</html>
```

Repetitive - Top-tested - sentinel



```
<html>
   <body>
    <script>
        <!--
       function sentinel(){
           var cont = "Y";
           var abort = " ";
           var output = " ";
           var count = 0;
           while (cont == "Y")
           {
              output = "You asked me to continue but typing " + cont + "</br>";
              document.getElementById("output").innerHTML = output;
              cont = prompt("Please enter Y to continue");
              count++;
           }
           abort = "Loop Aborted!, It made " + count + " repititions.";
           window.alert(abort);
        }//-->
    </script>
    <form>
          Do you want to continue? Enter Y for yes or another key to ABORT:
          <p id="output"></p>
          <input type="submit" value="Continue?" onclick="sentinel()">
    </form>
   </body>
</html>
```

Repetitive - Bottom-tested – Finite???



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

<p>Click the button to loop through a block of code as long as i is less
than 5.</p>

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

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

<script>
function myFunction() {
    var text = ""
    var i = 0;
    do {
        text += "<br>The number is " + i;
        i++;
    }
    while (i < 5);
    document.getElementById("demo").innerHTML = text;
}
</script>

</body>
</html>
```

Session Eight

MORE ON HTML5 – WEB STORAGE

HTML5 Form Elements – Web Forms 2.0

- Web Forms 2.0 is an extension to the forms features found in HTML4.
- Form elements and attributes in HTML5 provide a greater degree of semantic mark-up than HTML4 and free us from a great deal of tedious scripting and styling that was required in HTML4.

HTML5 – Web Storage

- HTML5 introduces two mechanisms, similar to HTTP session cookies, for storing structured data on the client side and to overcome following drawbacks.
  - Cookies are included with every HTTP request, thereby slowing down your web application by transmitting the same data.
  - Cookies are included with every HTTP request, thereby sending data unencrypted over the internet.
  - Cookies are limited to about 4 KB of data. Not enough to store required data.

HTML5 – Web Storage

- The two storages are **session storage** and **local storage** and they would be used to handle different situations.
- The latest versions of pretty much every browser supports HTML5 Storage including Internet Explorer.

Session Storage

- ***Session Storage*** is designed for scenarios where the user is carrying out a single transaction, but could be carrying out multiple transactions in different windows at the same time.

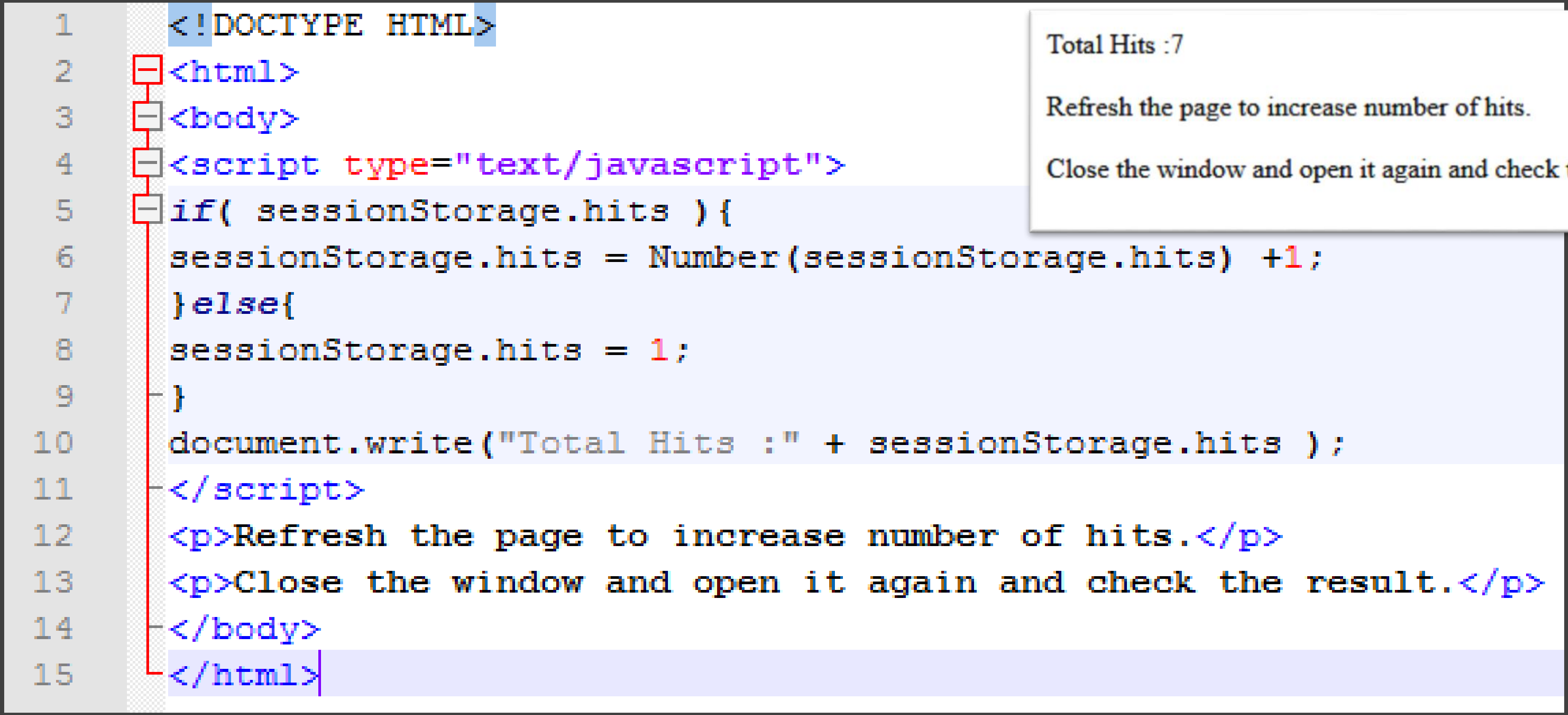
Session Storage - Problem

- *For example, if a user is buying plane tickets in two different windows, using the same site. If the site used cookies to keep track of which ticket the user was buying, then as the user clicked from page to page in both windows, the ticket currently being purchased would "leak" from one window to the other, potentially causing the user to buy two tickets for the same flight without really noticing.*

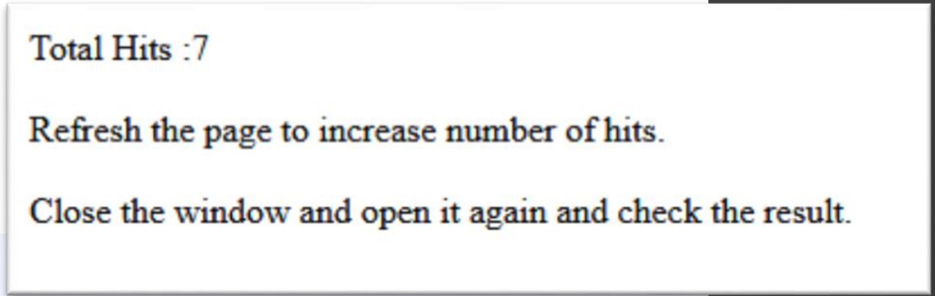
Session Storage - Solution

- HTML5 introduces the ***sessionStorage*** attribute which would be used by the sites to add data to the session storage, and it will be accessible to any page from the same site opened in that window, i.e., **session** and as soon as you close the window, the session would be lost.

Session Storage - Solution



```
<!DOCTYPE HTML>
<html>
<body>
<script type="text/javascript">
if( sessionStorage.hits ){
sessionStorage.hits = Number(sessionStorage.hits) +1;
}else{
sessionStorage.hits = 1;
}
document.write("Total Hits :" + sessionStorage.hits );
</script>
<p>Refresh the page to increase number of hits.</p>
<p>Close the window and open it again and check the result.</p>
</body>
</html>
```



```
Total Hits :7

Refresh the page to increase number of hits.

Close the window and open it again and check the result.
```

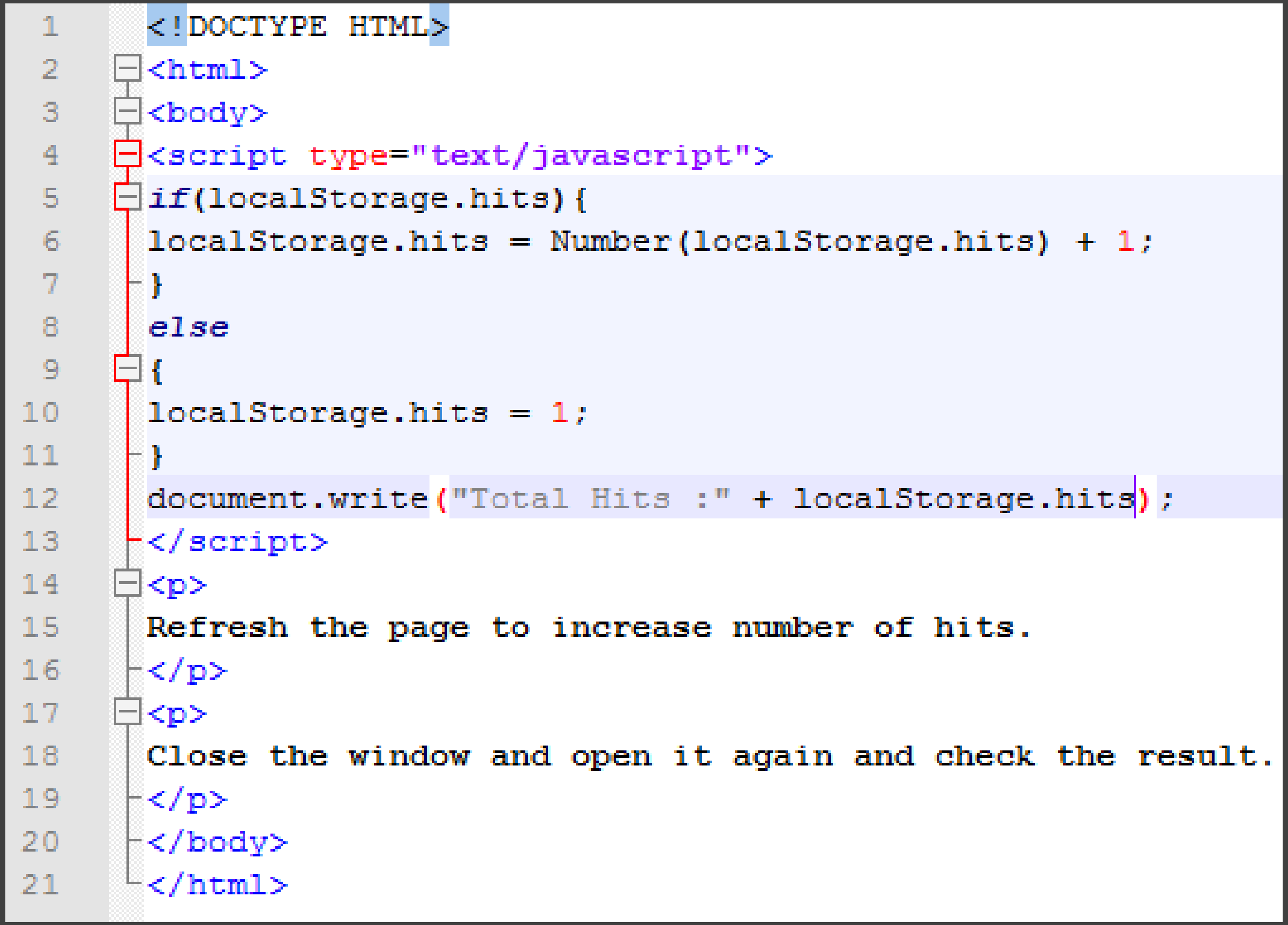
Local Storage

- The Local Storage is designed for storage that spans multiple windows, and lasts beyond the current session. In particular, Web applications may wish to store megabytes of user data, such as entire user-authored documents or a user's mailbox, on the client side for performance reasons.
- Again, cookies do not handle this case well, because they are transmitted with every request.

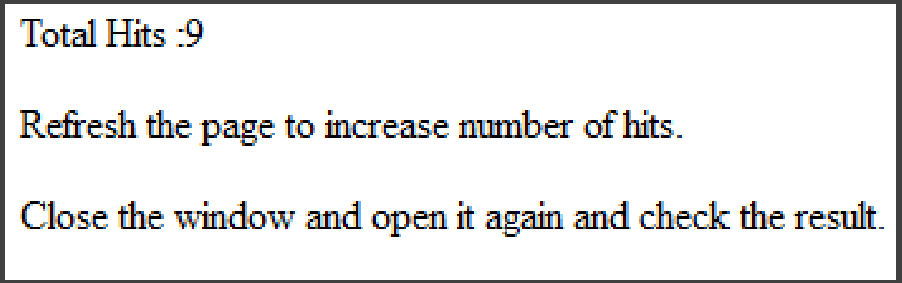
Local Storage

- HTML5 introduces the localStorage attribute which would be used to access a page's local storage area without a time limit and this local storage will be available whenever you use that page.
- The example on the next slide will set a local storage variable and access that variable every time this page is accessed, even next time you open the window

Local Storage - Example



```
<!DOCTYPE HTML>
<html>
<body>
<script type="text/javascript">
if(localStorage.hits){
localStorage.hits = Number(localStorage.hits) + 1;
}
else
{
localStorage.hits = 1;
}
document.write("Total Hits :" + localStorage.hits);
</script>
<p>
Refresh the page to increase number of hits.
</p>
<p>
Close the window and open it again and check the result.
</p>
</body>
</html>
```



Total Hits :9

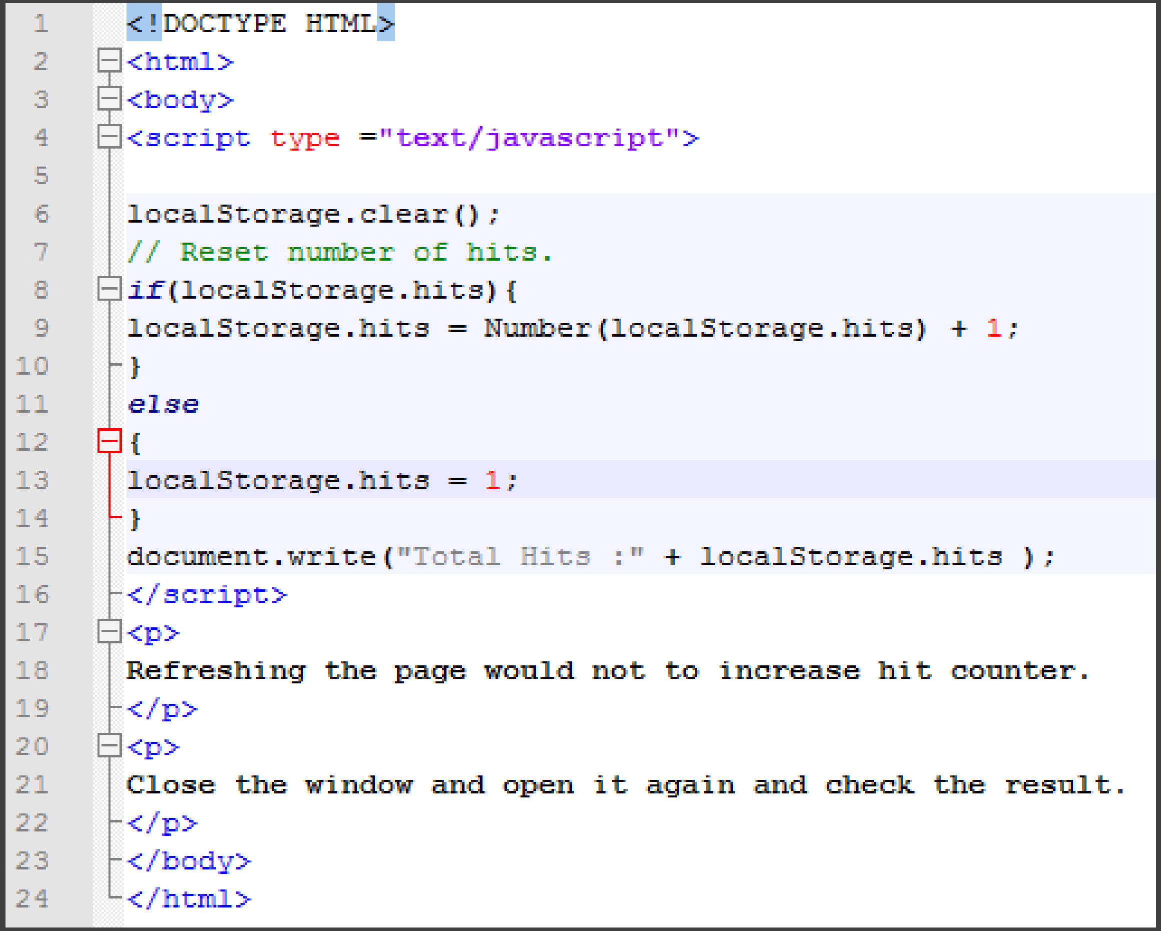
Refresh the page to increase number of hits.

Close the window and open it again and check the result.

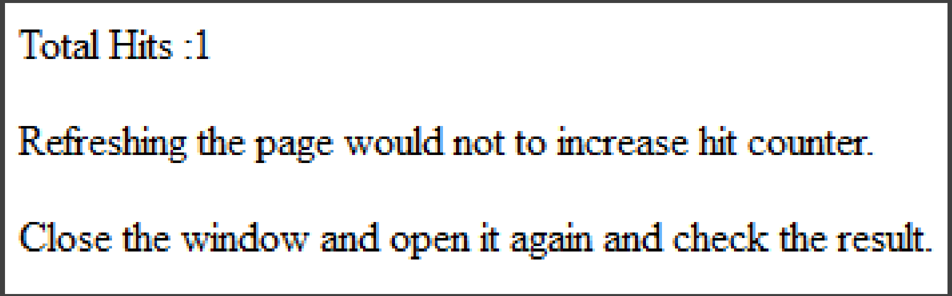
Delete Web Storage

- Storing sensitive data on local machine could be dangerous and could leave a security hole.
- The Session Storage Data would be deleted by the browsers immediately after the session gets terminated.
- To clear a local storage setting you would need to call **localStorage.remove('key');** where 'key' is the key of the value you want to remove. If you want to clear all settings, you need to call **localStorage.clear()** method.

Delete Web Storage



```
<!DOCTYPE HTML>
<html>
<body>
<script type ="text/javascript">

localStorage.clear();
// Reset number of hits.
if(localStorage.hits){
localStorage.hits = Number(localStorage.hits) + 1;
}
else
{
localStorage.hits = 1;
}
document.write("Total Hits :" + localStorage.hits );
</script>
<p>
Refreshing the page would not to increase hit counter.
</p>
<p>
Close the window and open it again and check the result.
</p>
</body>
</html>
```



```
Total Hits :1

Refreshing the page would not to increase hit counter.

Close the window and open it again and check the result.
```

Session 8

HTML5 - GEOLOCATION

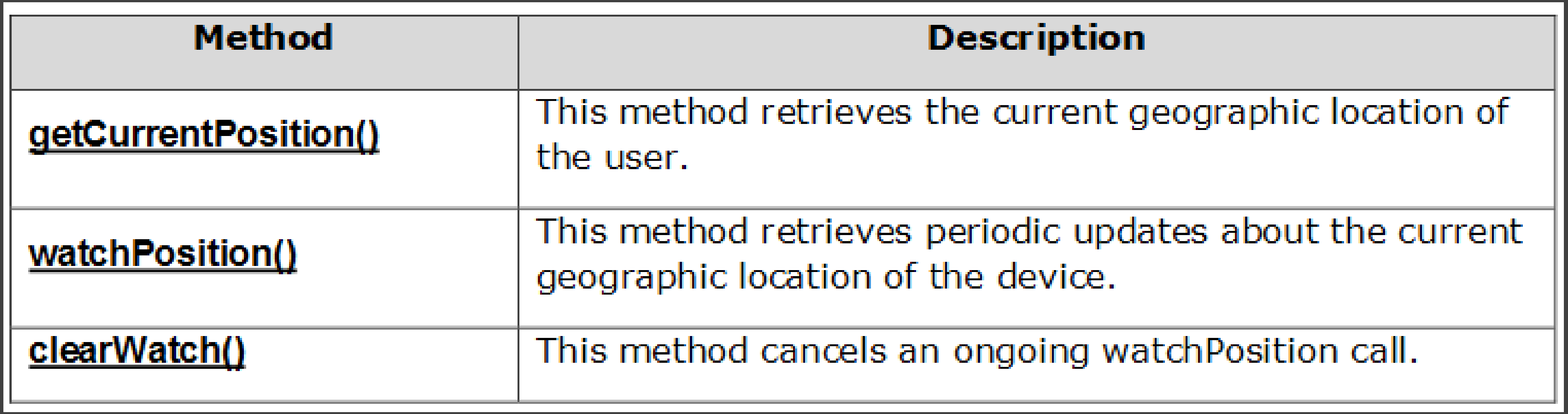
HTML5 Geolocation API

- HTML5 Geolocation API lets you share your location with your favorite web sites. A JavaScript can capture your latitude and longitude and can be sent to backend web server and do fancy location-aware things like finding local businesses or showing your location on a map.

HTML5 Geolocation API

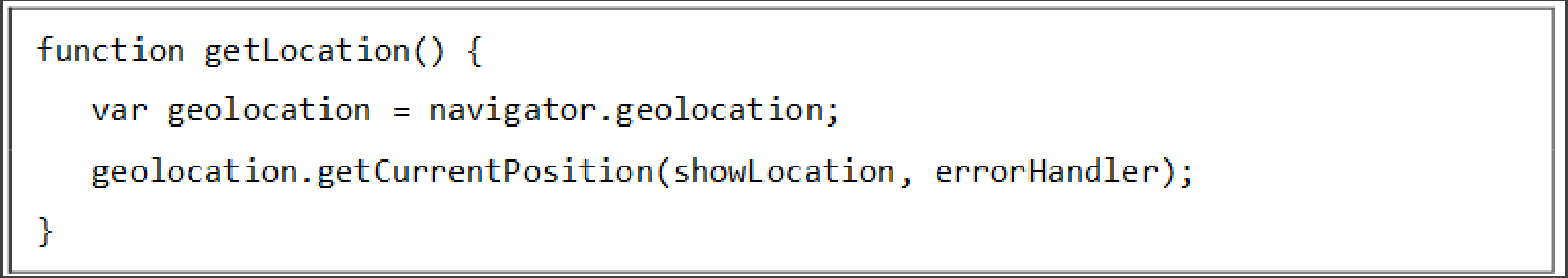
- Today most of the browsers and mobile devices support Geolocation API. The geolocation APIs work with a new property of the global navigator object ie. Geolocation object which can be created as follows:
  - `var geolocation = navigator.geolocation;`
- The geolocation object is a service object that allows widgets to retrieve information about the geographic location of the device.

Geolocation Methods



Method	Description
getCurrentPosition()	This method retrieves the current geographic location of the user.
watchPosition()	This method retrieves periodic updates about the current geographic location of the device.
clearWatch()	This method cancels an ongoing watchPosition call.

Geolocation - Example



```
function getLocation() {
   var geolocation = navigator.geolocation;
   geolocation.getCurrentPosition(showLocation, errorHandler);
}
```

showLocation and **errorHandler** are callback methods which would be used to get actual position and to handle errors if there are any.

Geolocation getCurrentPosition() API

- The getCurrentPosition method retrieves the current geographic location of the device.
- The location is expressed as a set of geographic coordinates together with information about heading and speed.
- The location information is returned in a Position object

Geolocation getCurrentPosition() API

getCurrentPosition(showLocation, ErrorHandler, options);

- **showLocation**: This specifies the callback method that retrieves the location information. This method is called asynchronously with an object corresponding to the **Position** object which stores the returned location information.

Geolocation getCurrentPosition() API

```
getCurrentPosition(showLocation, ErrorHandler, options);
```

- **ErrorHandler:** This optional parameter specifies the callback method that is invoked when an error occurs in processing the asynchronous call. This method is called with the PositionError object that stores the returned error information.

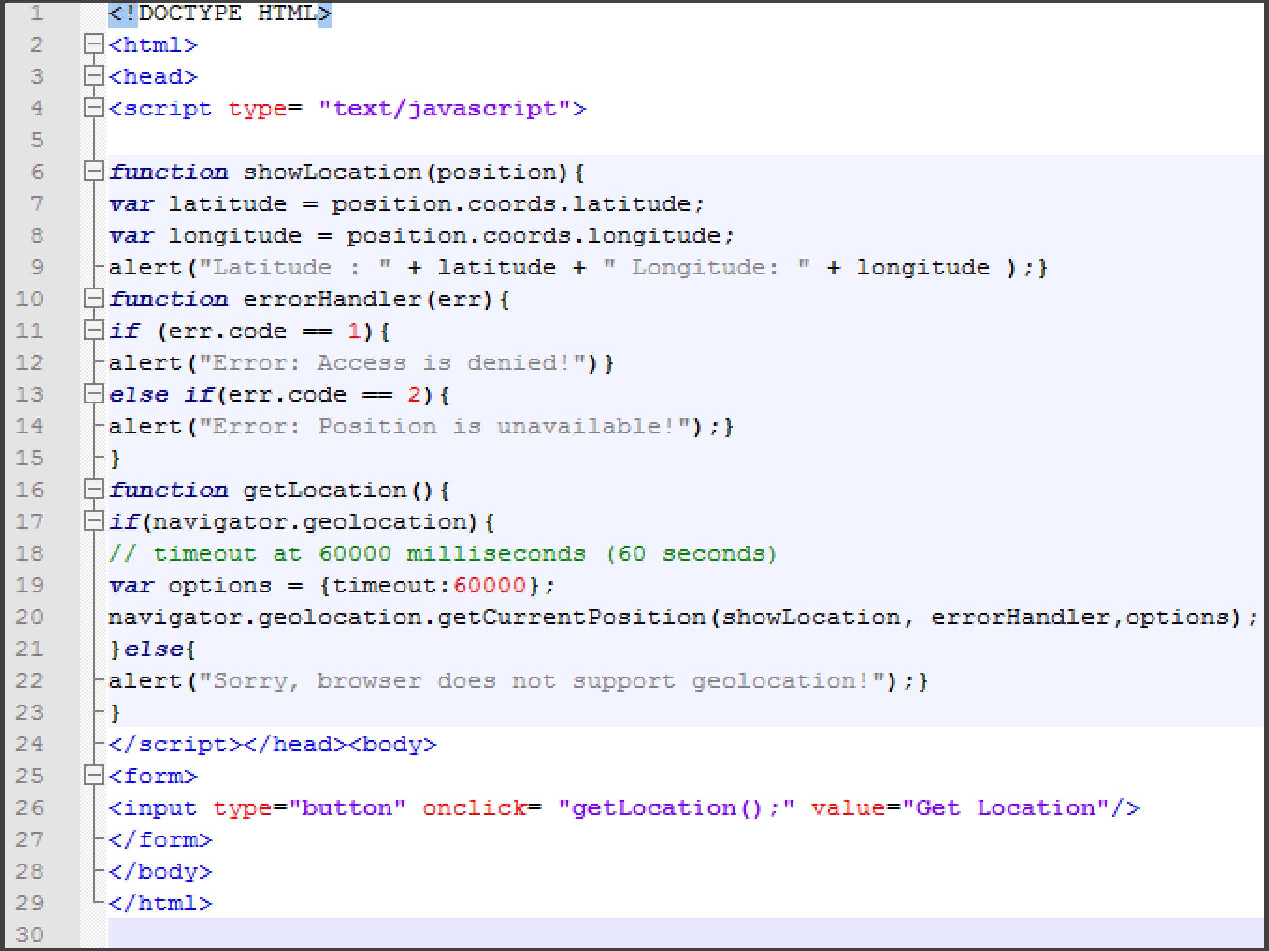
Geolocation getCurrentPosition() API

```
getCurrentPosition(showLocation, ErrorHandler, options);
```

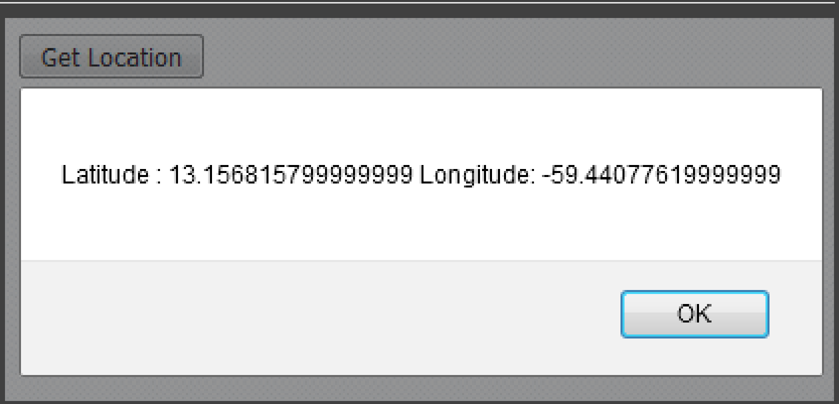
- **options**–This optional parameter specifies a set of options for retrieving the location information. You can specify (a) Accuracy of the returned location information (b) Timeout for retrieving the location information and (c) Use of cached location information.

Note: The **getCurrentPosition** method does not return a value

Geolocation - Example



```
<!DOCTYPE HTML>
<html>
<head>
<script type= "text/javascript">

function showLocation(position){
var latitude = position.coords.latitude;
var longitude = position.coords.longitude;
alert("Latitude : " + latitude + " Longitude: " + longitude );}
function errorHandler(err){
if (err.code == 1){
alert("Error: Access is denied!")}
else if(err.code == 2){
alert("Error: Position is unavailable!");}
}
function getLocation(){
if(navigator.geolocation){
// timeout at 60000 milliseconds (60 seconds)
var options = {timeout:60000};
navigator.geolocation.getCurrentPosition(showLocation, errorHandler,options);
}else{
alert("Sorry, browser does not support geolocation!");}
}
</script></head><body>
<form>
<input type="button" onclick= "getLocation();" value="Get Location"/>
</form>
</body>
</html>
```



```
Latitude : 13.156815799999999 Longitude: -59.44077619999999
```

Geolocation watchPosition() API

- The **watchPosition** method retrieves periodic updates about the current geographic location of the device. The location is expressed as a set of geographic coordinates together with information about heading and speed.
- The location information is returned in a Position object. Each update returns a new Position object.

Geolocation watchPosition() API

```
watchPosition(showLocation, ErrorHandler, options);
```

- showLocation–This specifies the callback method that retrieves the location information. This method is called asynchronously with an object corresponding to the Position
- object which stores the returned location information.

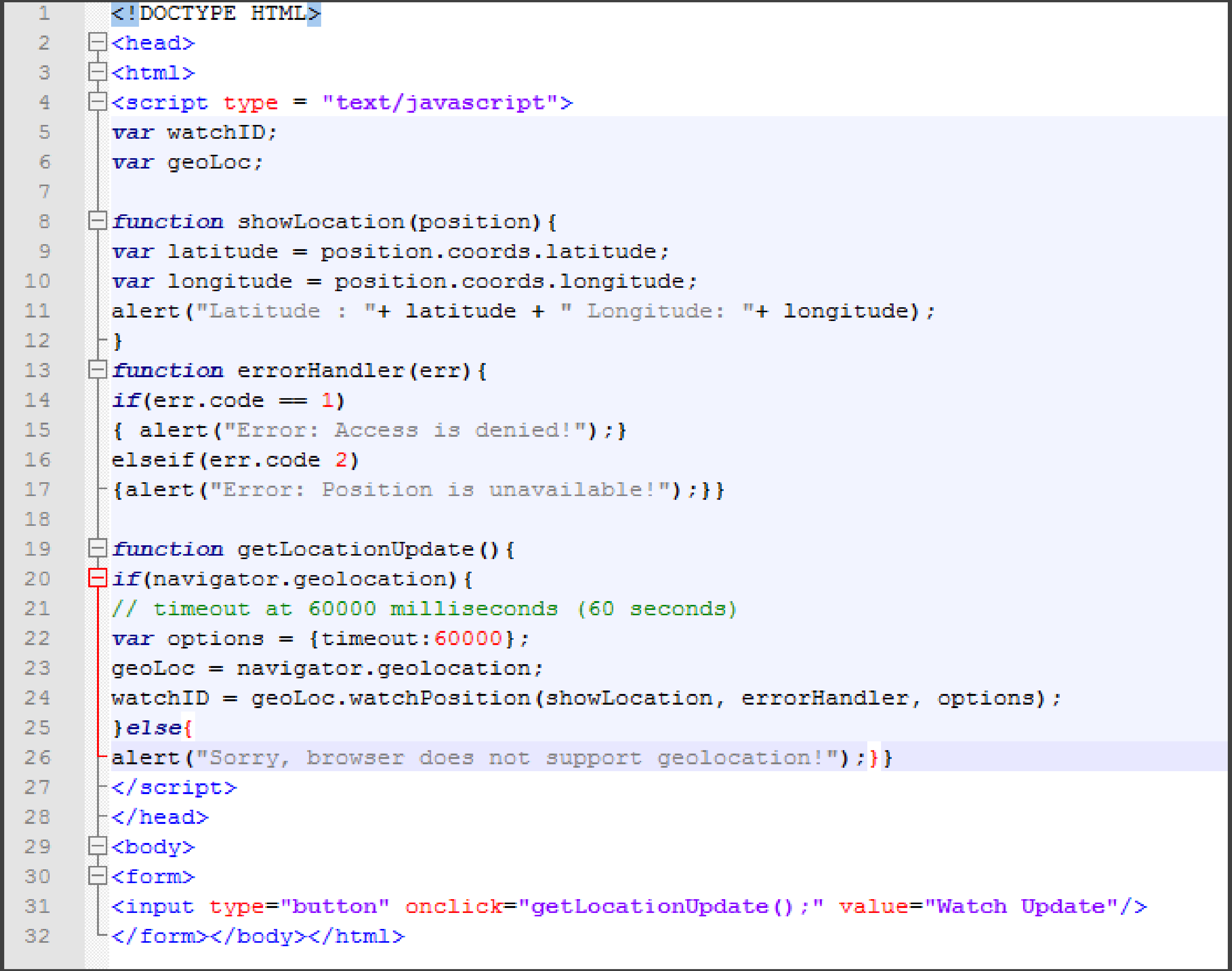
Geolocation watchPosition() API

- **`watchPosition(showLocation, ErrorHandler, options);`**
- ErrorHandler: This optional parameter specifies the callback method that is invoked when an error occurs in processing the asynchronous call. This method is called with the PositionError object that stores the returned error information.
- Options: This optional parameter specifies a set of options for retrieving the location information. You can specify
  (a) Accuracy of the returned location information
  (b) Timeout for retrieving the location information and
  (c) Use of cached location information.

Geolocation watchPosition() API

- `watchPosition(showLocation, ErrorHandler, options);`
- The watchPosition method returns a unique transaction ID (number) associated with the asynchronous call. Use this ID to cancel the watchPosition call and to stop receiving location updates.

Geolocation watchPosition() Example



```
<!DOCTYPE HTML>
<head>
<html>
<script type = "text/javascript">
var watchID;
var geoLoc;

function showLocation(position){
var latitude = position.coords.latitude;
var longitude = position.coords.longitude;
alert("Latitude : "+ latitude + " Longitude: "+ longitude);
}
function errorHandler(err){
if(err.code == 1)
{ alert("Error: Access is denied!");}
elseif(err.code 2)
{alert("Error: Position is unavailable!");}}

function getLocationUpdate(){
if(navigator.geolocation){
// timeout at 60000 milliseconds (60 seconds)
var options = {timeout:60000};
geoLoc = navigator.geolocation;
watchID = geoLoc.watchPosition(showLocation, errorHandler, options);
}else{
alert("Sorry, browser does not support geolocation!");}}
</script>
</head>
<body>
<form>
<input type="button" onclick="getLocationUpdate();" value="Watch Update"/>
</form></body></html>
```

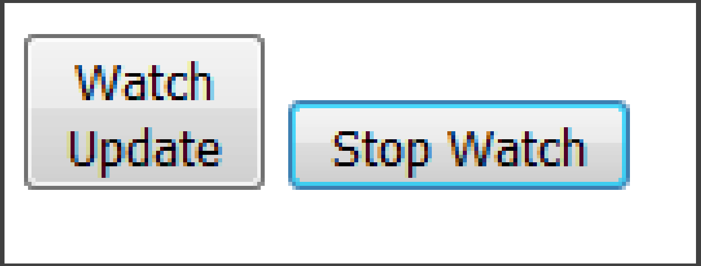
Geolocation clearWatch() API

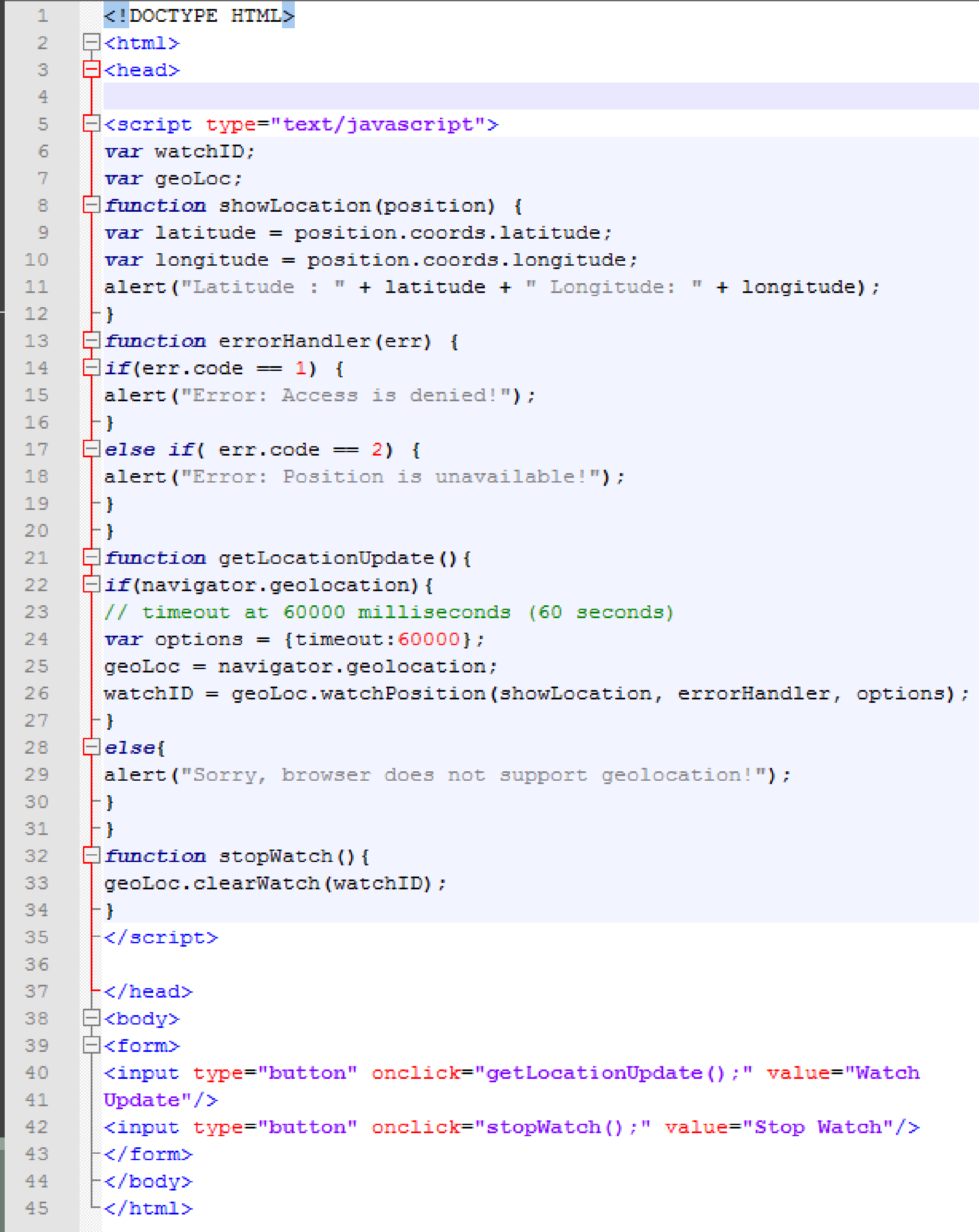
- The clearWatch method cancels an ongoing watchPosition call. When cancelled, the watch Position call stops retrieving updates about the current geographic location of the device.

Geolocation clearWatch() API - Syntax

- `clearWatch(watchId);`
- **watchId** – This specifies the unique ID of the watchPosition call to cancel. The IDis returned by the watchPosition call.

Geolocation clearWatch() - Example





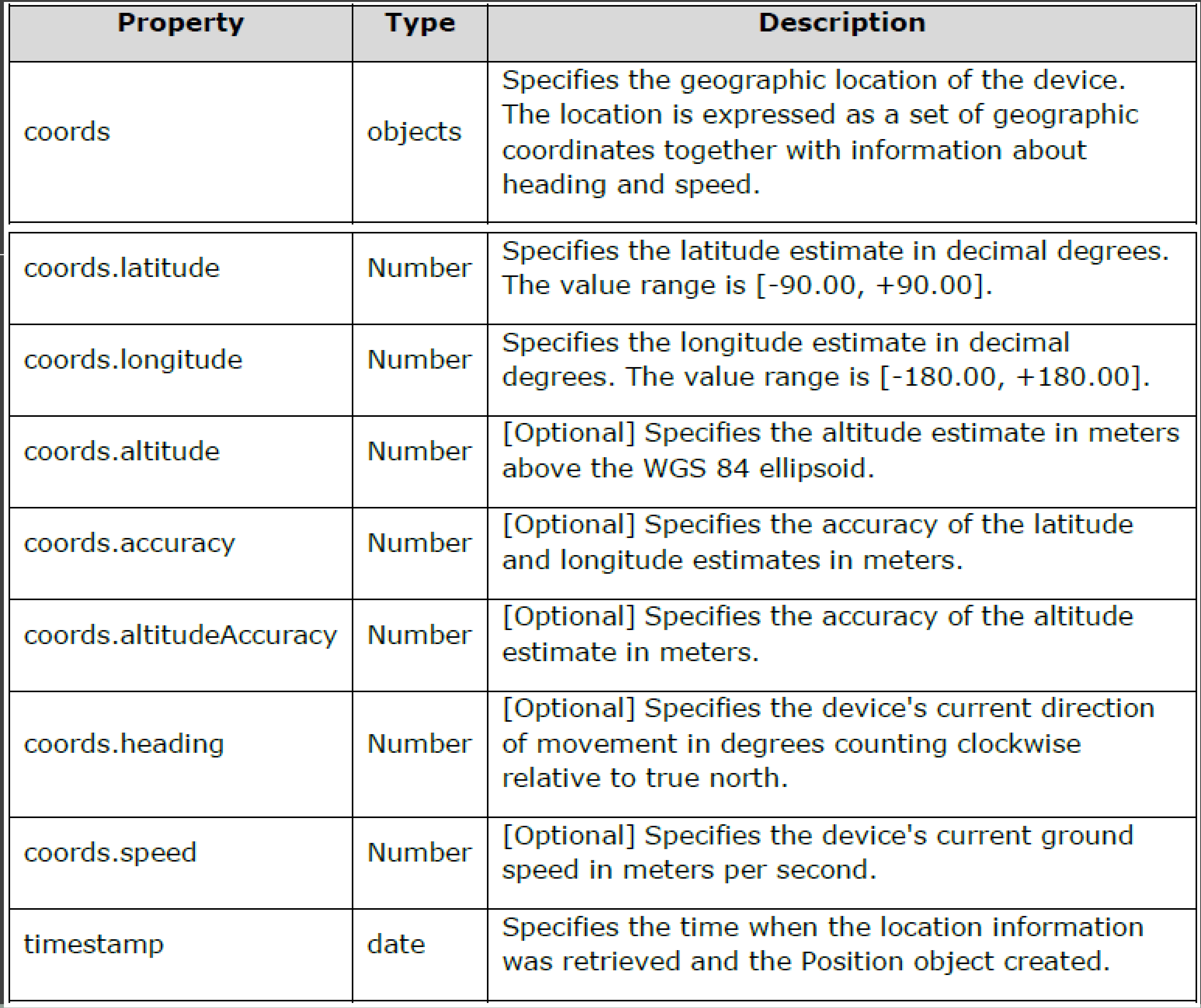
```
<!DOCTYPE HTML>
<html>
<head>

<script type="text/javascript">
var watchID;
var geoLoc;
function showLocation(position) {
var latitude = position.coords.latitude;
var longitude = position.coords.longitude;
alert("Latitude : " + latitude + " Longitude: " + longitude);
}
function errorHandler(err) {
if(err.code == 1) {
alert("Error: Access is denied!");
}
else if( err.code == 2) {
alert("Error: Position is unavailable!");
}
}
function getLocationUpdate(){
if(navigator.geolocation){
// timeout at 60000 milliseconds (60 seconds)
var options = {timeout:60000};
geoLoc = navigator.geolocation;
watchID = geoLoc.watchPosition(showLocation, errorHandler, options);
}
else{
alert("Sorry, browser does not support geolocation!");
}
}
function stopWatch(){
geoLoc.clearWatch(watchID);
}
</script>

</head>
<body>
<form>
<input type="button" onclick="getLocationUpdate();" value="Watch
Update"/>
<input type="button" onclick="stopWatch();" value="Stop Watch"/>
</form>
</body>
</html>
```

Location Properties

- Geolocation methods getCurrentPosition() and getPositionUsingMethodName() specify the callback method that retrieves the location information.
- These methods are called asynchronously with an object **Position** which stores the complete location information.
- The **Position** object specifies the current geographic location of the device. The location is expressed as a set of geographic coordinates together with information about heading and speed.

Location Properties

- This table describes the properties of the Position object. For the optional properties if the system cannot provide a value, the value of the property is set to null.



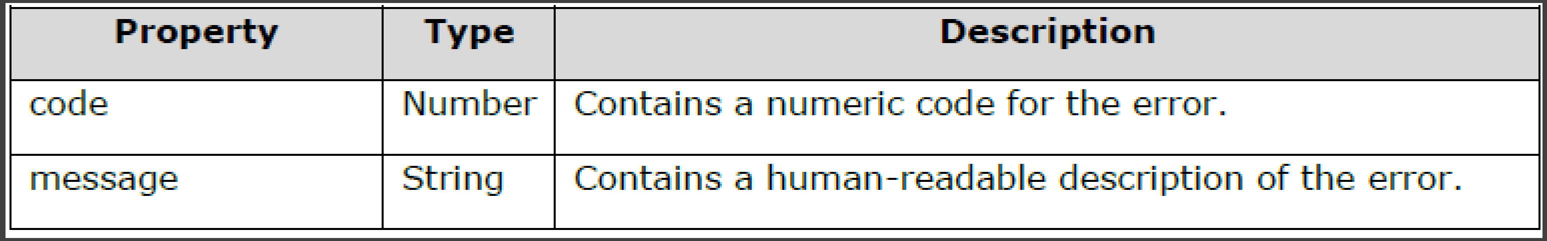
Property	Type	Description
coords	objects	Specifies the geographic location of the device. The location is expressed as a set of geographic coordinates together with information about heading and speed.
coords.latitude	Number	Specifies the latitude estimate in decimal degrees. The value range is [-90.00, +90.00].
coords.longitude	Number	Specifies the longitude estimate in decimal degrees. The value range is [-180.00, +180.00].
coords.altitude	Number	[Optional] Specifies the altitude estimate in meters above the WGS 84 ellipsoid.
coords.accuracy	Number	[Optional] Specifies the accuracy of the latitude and longitude estimates in meters.
coords.altitudeAccuracy	Number	[Optional] Specifies the accuracy of the altitude estimate in meters.
coords.heading	Number	[Optional] Specifies the device's current direction of movement in degrees counting clockwise relative to true north.
coords.speed	Number	[Optional] Specifies the device's current ground speed in meters per second.
timestamp	date	Specifies the time when the location information was retrieved and the Position object created.

Handling Errors

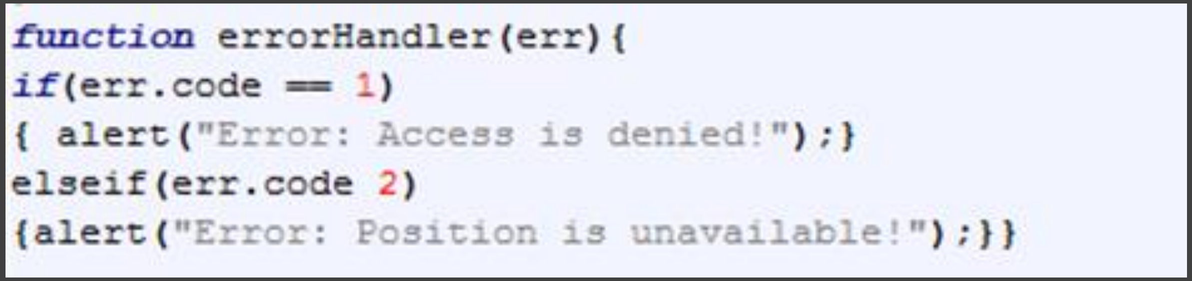
- Geolocation is complicated, and it is very much required to catch any error and handle it gracefully.
- The geolocations methods getCurrentPosition() and watchPosition() make use of an errorhandler callback method which gives **PositionError** object.

Handling Errors

This object has following two properties:



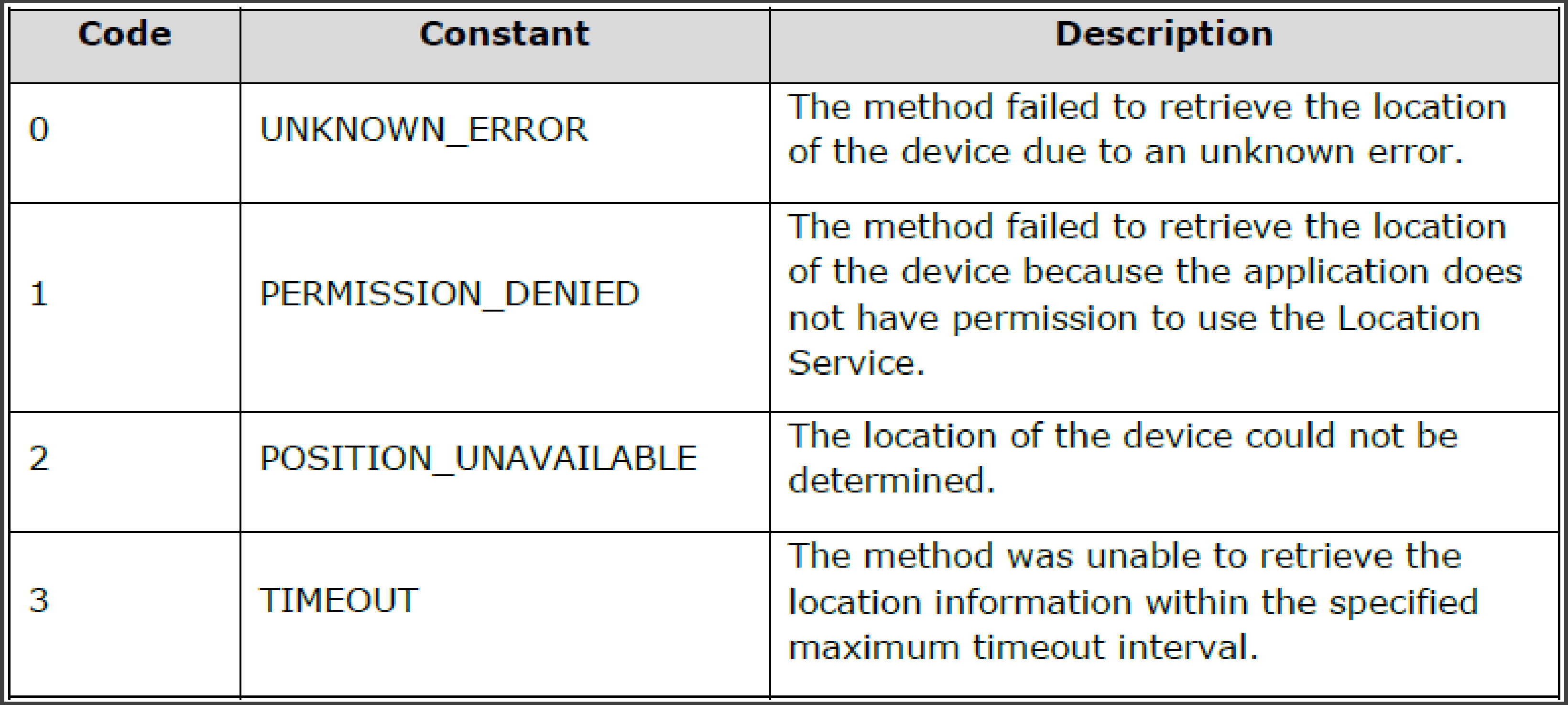
Property	Type	Description
code	Number	Contains a numeric code for the error.
message	String	Contains a human-readable description of the error.



```
function errorHandler(err){
if(err.code == 1)
{ alert("Error: Access is denied!");}
elseif(err.code 2)
{alert("Error: Position is unavailable!");}}
```

Handling Errors

- These are the possible error codes returned in the PositionError object.



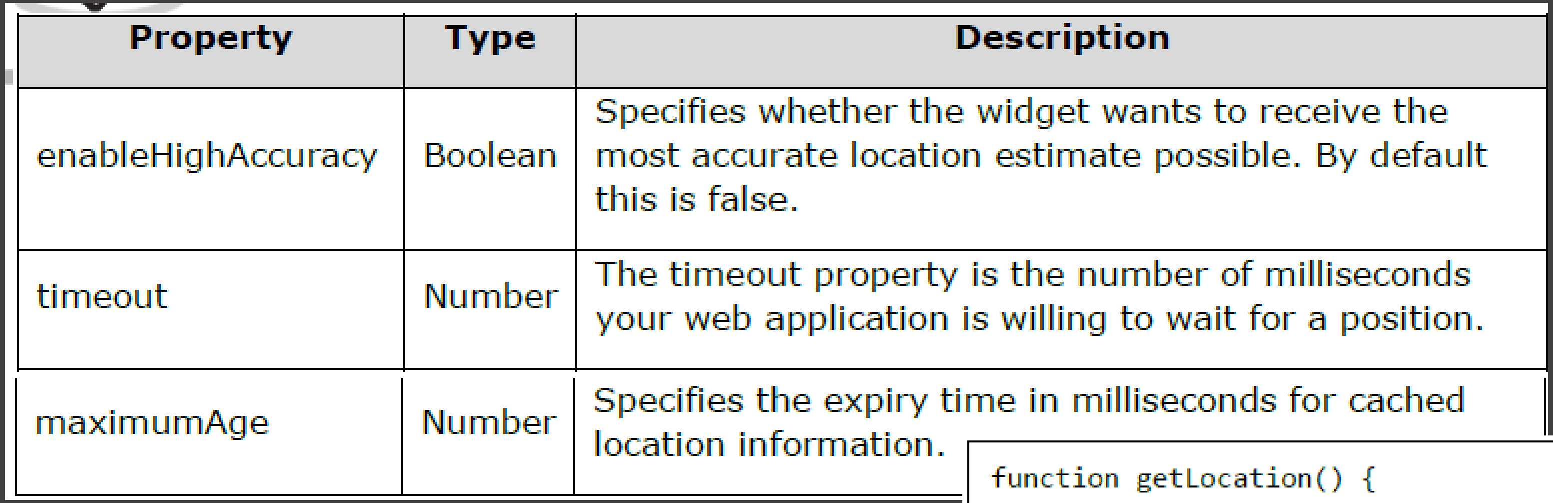
Code	Constant	Description
0	UNKNOWN_ERROR	The method failed to retrieve the location of the device due to an unknown error.
1	PERMISSION_DENIED	The method failed to retrieve the location of the device because the application does not have permission to use the Location Service.
2	POSITION_UNAVAILABLE	The location of the device could not be determined.
3	TIMEOUT	The method was unable to retrieve the location information within the specified maximum timeout interval.

Position Options

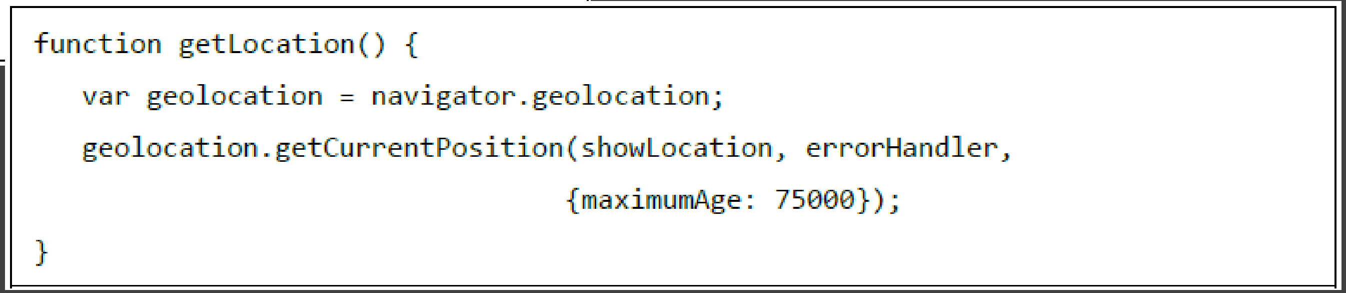
- Recall the syntax:
  - getCurrentPosition(callback, ErrorCallback, options)
- Here the third argument is the **PositionOptions** object which specifies a set of options for retrieving the geographic location of the device.

Position Options

- **`getCurrentPosition(callback, ErrorCallback, options)`**
- Here are the options which can be specified as third argument:



Property	Type	Description
enableHighAccuracy	Boolean	Specifies whether the widget wants to receive the most accurate location estimate possible. By default this is false.
timeout	Number	The timeout property is the number of milliseconds your web application is willing to wait for a position.
maximumAge	Number	Specifies the expiry time in milliseconds for cached location information.

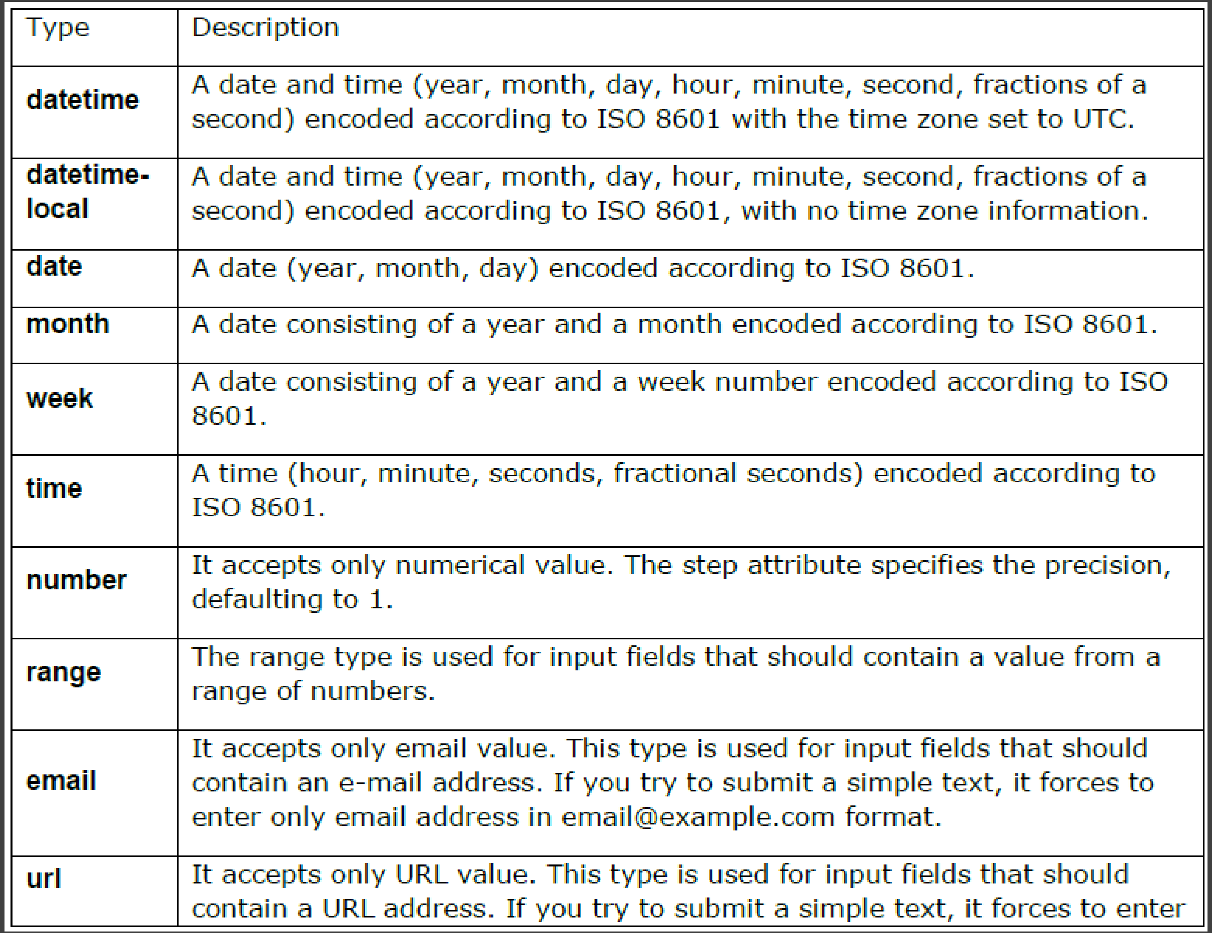


```
function getLocation() {
   var geolocation = navigator.geolocation;
   geolocation.getCurrentPosition(showLocation, errorHandler,
                                  {maximumAge: 75000});
}
```

Session Eight

A BIT MORE ON FORMS

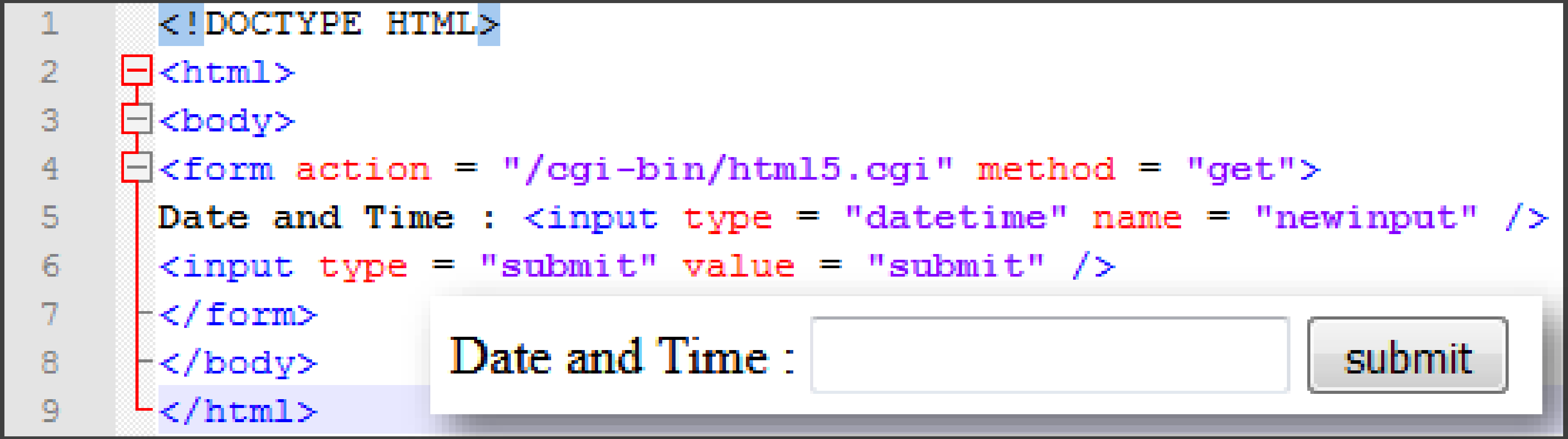
HTML5 Form Elements



Type	Description
datetime	A date and time (year, month, day, hour, minute, second, fractions of a second) encoded according to ISO 8601 with the time zone set to UTC.
datetime-local	A date and time (year, month, day, hour, minute, second, fractions of a second) encoded according to ISO 8601, with no time zone information.
date	A date (year, month, day) encoded according to ISO 8601.
month	A date consisting of a year and a month encoded according to ISO 8601.
week	A date consisting of a year and a week number encoded according to ISO 8601.
time	A time (hour, minute, seconds, fractional seconds) encoded according to ISO 8601.
number	It accepts only numerical value. The step attribute specifies the precision, defaulting to 1.
range	The range type is used for input fields that should contain a value from a range of numbers.
email	It accepts only email value. This type is used for input fields that should contain an e-mail address. If you try to submit a simple text, it forces to enter only email address in email@example.com format.
url	It accepts only URL value. This type is used for input fields that should contain a URL address. If you try to submit a simple text, it forces to enter

datetime set to UTC

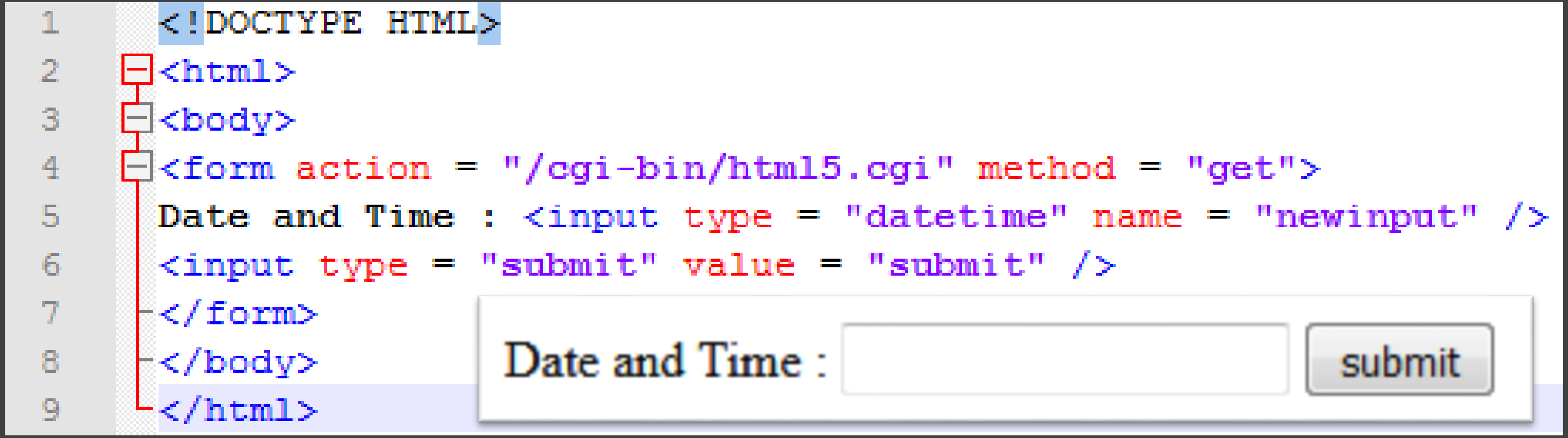
- **UTC** / GMT is the basis for local times worldwide. Other names: Universal Time Coordinated / Universal Coordinated Time. Successor to: Greenwich Mean Time (GMT) Military name: “Zulu” Military Time. Longitude: 0° (Prime Meridian)



```
<!DOCTYPE HTML>
<html>
<body>
<form action = "/cgi-bin/html5.cgi" method = "get">
Date and Time : <input type = "datetime" name = "newinput" />
<input type = "submit" value = "submit" />
</form>
</body>
</html>
```

datetime set to UTC

- **UTC** / GMT is the basis for local times worldwide. Other names: Universal Time Coordinated / Universal Coordinated Time. Successor to: Greenwich Mean Time (GMT) Military name: “Zulu” Military Time. Longitude: 0° (Prime Meridian)



```
<!DOCTYPE HTML>
<html>
<body>
<form action = "/cgi-bin/html5.cgi" method = "get">
Date and Time : <input type = "datetime" name = "newinput" />
<input type = "submit" value = "submit" />
</form>
</body>
</html>
```

Quick Review

- JavaScript Programming Constructs
- Web Storage
- Geolocation
- A bit more on forms