Mansoura University
Faculty of Computers and Information
Department of Information System
Second Semester- 2021-2022

Web Engineering / E-Commerce

Level 3

Dr. Nabila Eladawi

& Dr. Sara Shaker

Web Design

Lecture3: JavaScript

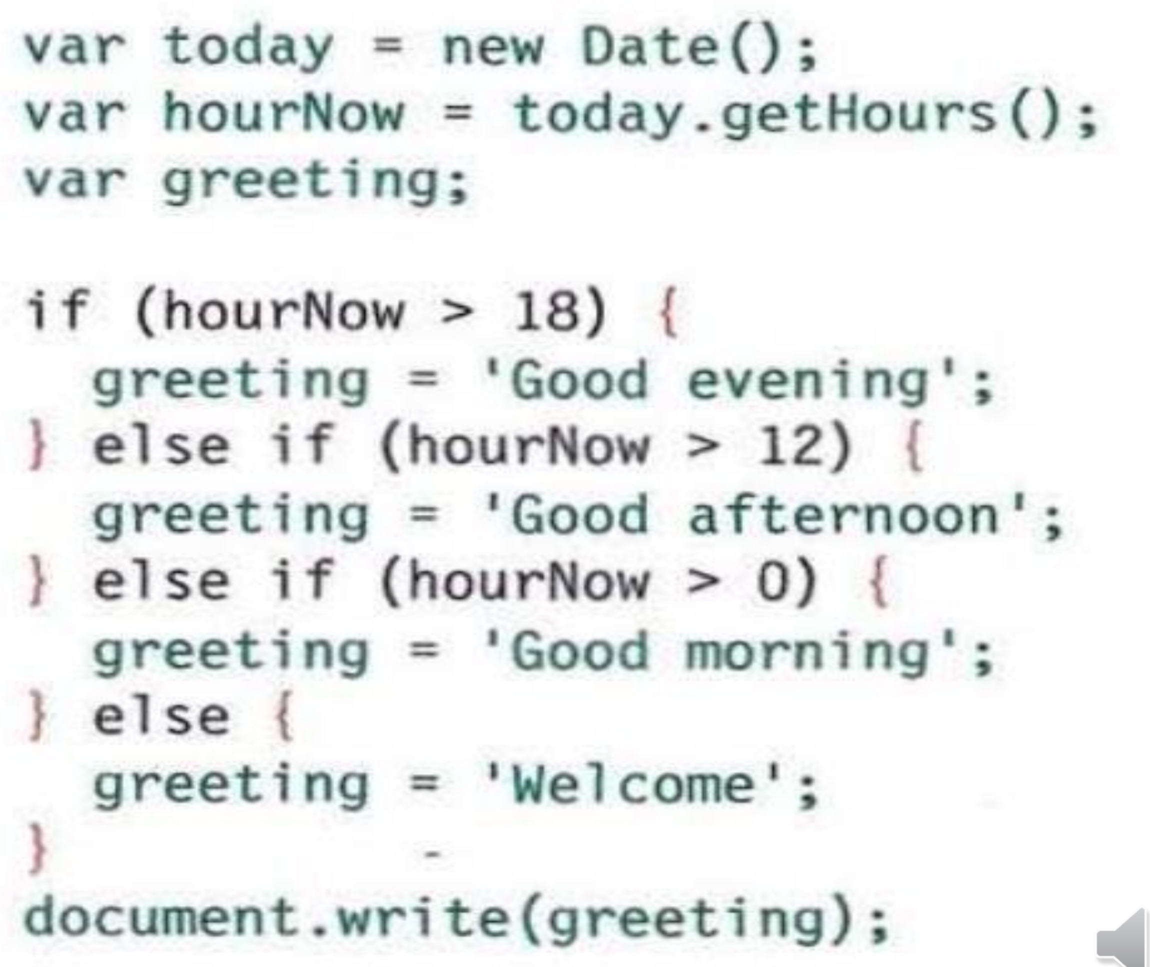
Mansoura University

Faculty of Computers and Information

Dept. of Information System

Script

- ➢ A series of instructions that the computer can follow in order to achieve a goal.
- ➢ Each individual step is known as a statement.
- ➢ A group is called code Block



```
var today = new Date();
var hourNow = today.getHours();
var greeting;

if (hourNow > 18) {
  greeting = 'Good evening';
} else if (hourNow > 12) {
  greeting = 'Good afternoon';
} else if (hourNow > 0) {
  greeting = 'Good morning';
} else {
  greeting = 'Welcome';
}
document.write(greeting);
```

How to add Script to Web Page?

- It is preferred to keep the three languages in separate files, with the HTML page linking to CSS and JavaScript files
  - Content Layer (.html files)
  - Presentation Layer (. css files)
  - Behavior Layer (. js files)

JavaScript Where To

- In HTML, JavaScript code is inserted between <script> and </script> tags.

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

- 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.

JavaScript Where To

```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph 
changed.";}
</script>
</head>
<body><h2>Demo JavaScript in Head</h2>
<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try it</button>
</body>
</html>
```

External JavaScript

- 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.

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

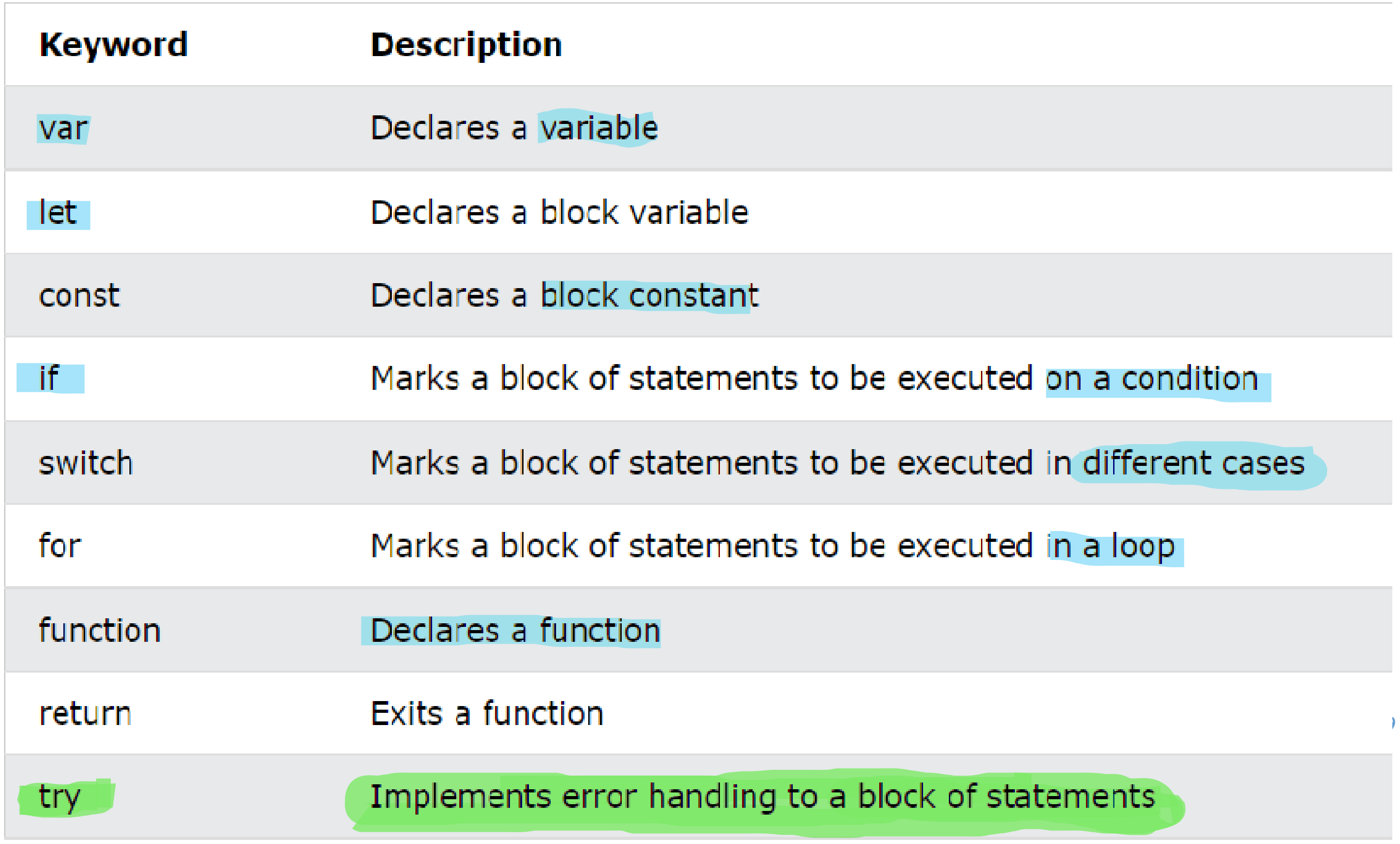
JavaScript Statements

- JavaScript statements are composed of:
- Values, Operators, Expressions, Keywords, and Comments.
- Semicolons separate JavaScript statements.
- This statement tells the browser to write "Hello Dolly." inside an HTML element with id="demo":

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

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.



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 Variables

- There are 3 ways to declare a JavaScript variable:
  - Using var
  - Using let
  - Using const

```
var price1 = 5;
var price2 = 6;
var total = price1 + price2;
```

- The general rules for constructing names for variables (unique identifiers) are:
  - Names can contain letters, digits, underscores, and dollar signs.
  - Names must begin with a letter
  - Names can also begin with $ and _
  - Names are case sensitive (y and Y are different variables)
  - Reserved words (like JavaScript keywords) cannot be used as names

- When declaring a variable in JavaScript, you do not need to specify the type of data.

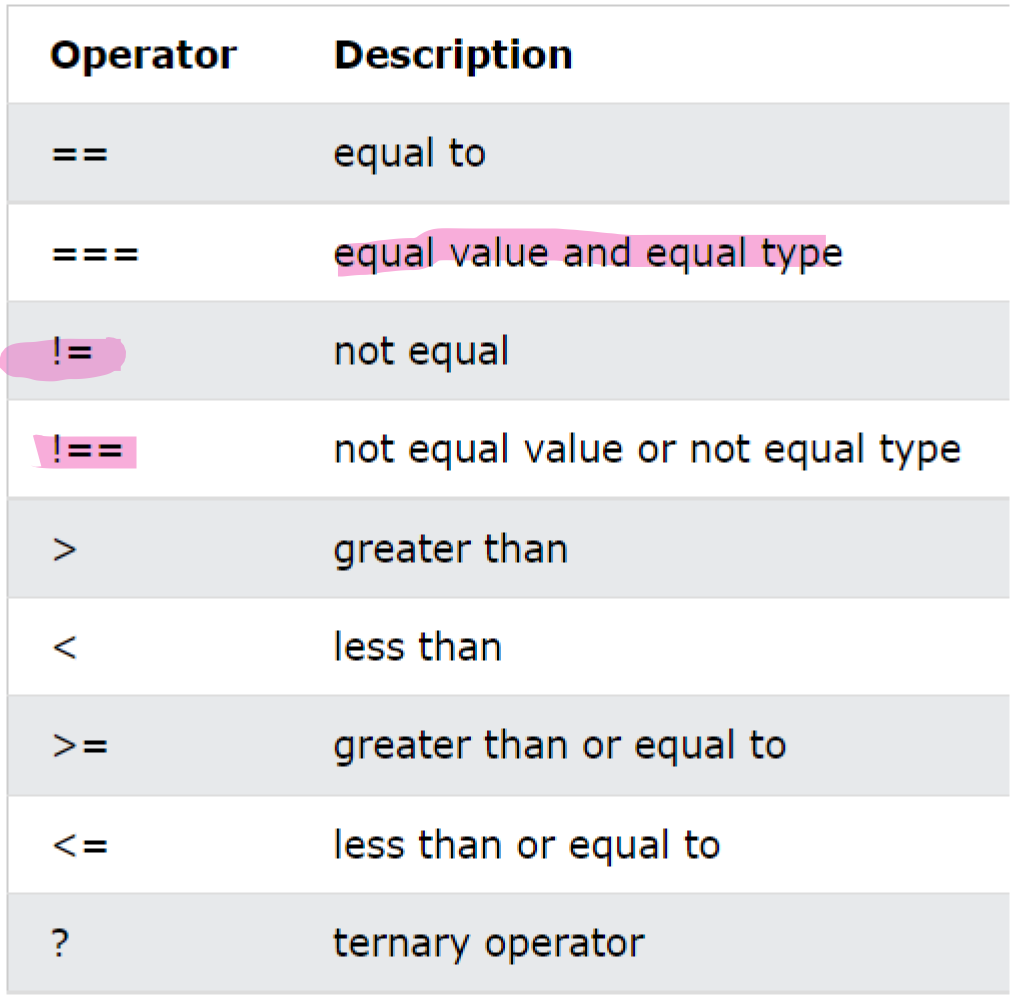
JavaScript Arithmetic Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES2016)
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

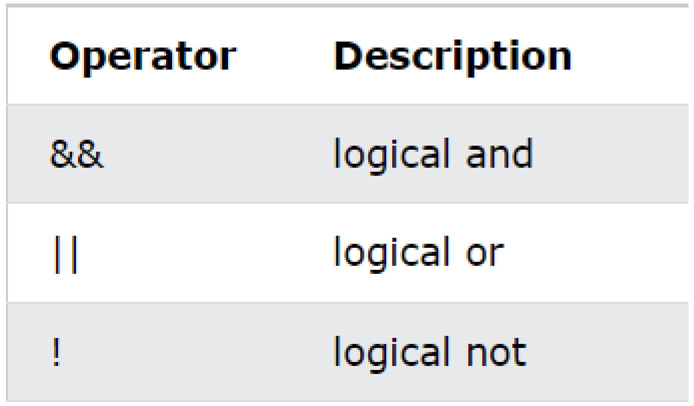
JavaScript Assignment Operators

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

JavaScript Comparison and Logical Operators



Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator



Operator	Description
&&	logical and
||	logical or
!	logical not

JavaScript Functions

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

```
function myFunction(p1, p2) {
  return p1 * p2;   // The function returns the
product of p1 and p2
}
```

- JavaScript function is defined with the function keyword, followed by a name, followed by parentheses ().
- Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).
- The parentheses may include parameter names separated by commas: (parameter1, parameter2, ...).
- The code to be executed, by the function, is placed inside curly brackets: {}

JavaScript Functions

- With functions:
  - You can reuse code: Define the code once, and use it many times.
  - You can use the same code many times with different arguments, to produce different results.

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}
document.getElementById("demo").innerHTML = toCelsius(77);
```

- Functions can be used the same way as you use variables, in all types of formulas, assignments, and calculations.

```
let x = toCelsius(77);
let text = "The temperature is " + x + " Celsius";
let text = "The temperature is " + toCelsius(77)
+ " Celsius";
```

JavaScript Functions

- Variables declared within a JavaScript function, become LOCAL to the function.
- Local variables can only be accessed from within the function.

```
// code here can NOT use carName

function myFunction() {
  let carName = "Volvo";
  // code here CAN use carName
}

// code here can NOT use carName
```

- Since local variables are only recognized inside their functions, variables with the same name can be used in different functions.
- Local variables are created when a function starts, and deleted when completed.

JavaScript Objects

- 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.
- Objects are variables too. But objects can contain many values.
- This code assigns many values (Fiat, 500, white) to a variable named car:

```
const car = {type:"Fiat", model:"500",
color:"white"};
```

JavaScript Objects

- The name:values pairs in JavaScript objects are called properties
- You can access object properties in two ways:
  - *objectName.propertyName*
  - *objectName["propertyName"]*

```
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};
person.lastName;
```

Object Methods

- Objects can also have methods.
- Methods are actions that can be performed on objects.
- Methods are stored in properties as function definitions.
- In a function definition, *this* refers to the "owner" of the function.
- In the example, *this* is the person object that "owns" the fullName function.
- In other words, this.firstName

means the firstName property

of this object.

```
const person = {
  firstName: "John",
  lastName : "Doe",
  id       : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
```

JavaScript Events

- HTML events are "things" that happen to HTML elements.
- When JavaScript is used in HTML pages, JavaScript can "react" on these events.
- An HTML event can be something the browser does, or something a user does.
- Here are some examples of HTML events:
  - An HTML web page has finished loading
  - An HTML input field was changed
  - An HTML button was clicked
- Often, when events happen, you may want to do something.
- JavaScript lets you execute code when events are detected.
- HTML allows event handler attributes, with JavaScript code, to be added to elements.

JavaScript Events

- In this example,

the JavaScript code will execute

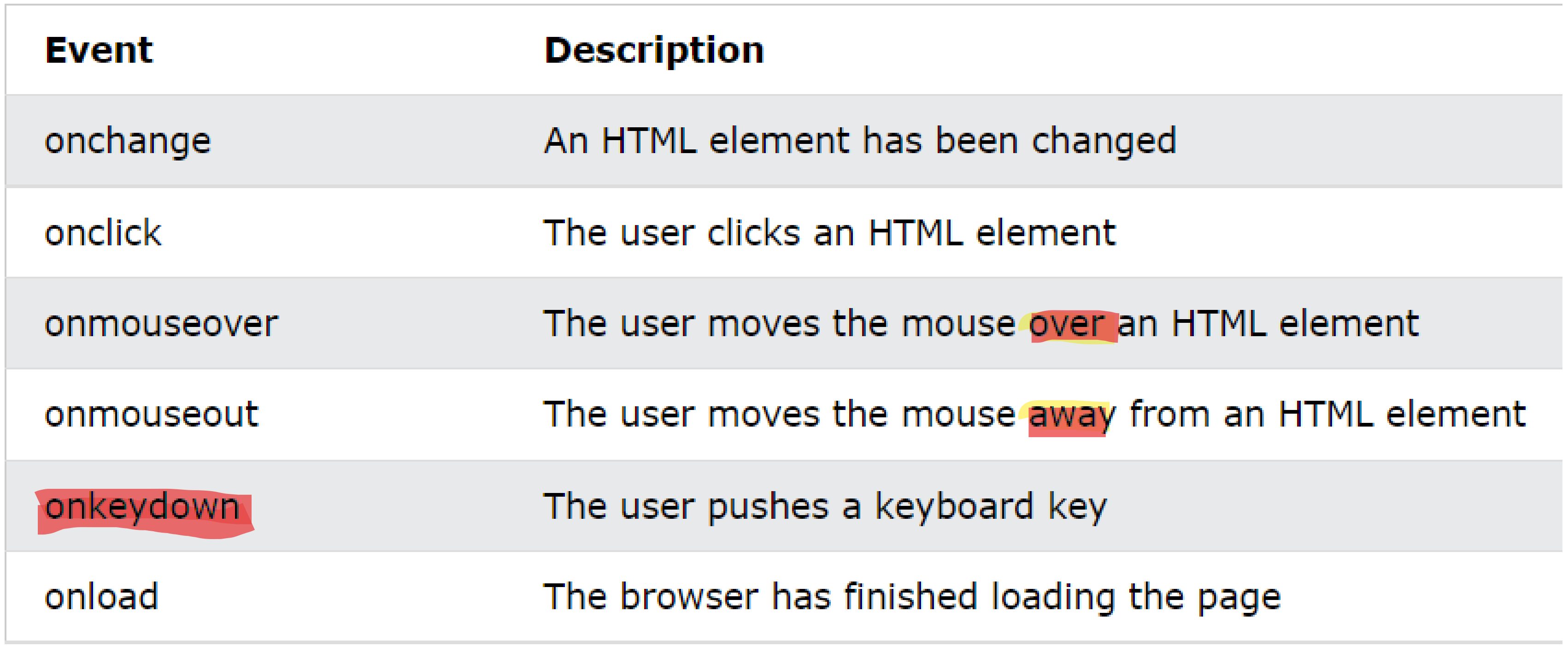
The function sayHello() which

Will write the “Hello world”

statement

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

Common HTML Events



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

JavaScript Strings

- JavaScript strings are for storing and manipulating text.
- A JavaScript string is zero or more characters written inside quotes.
- You can use single or double quotes
- You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

```
let answer2 = "He is called 'Johnny'";
let answer3 = 'He is called "Johnny"';
```

- To find the length of a string, use the built-in length property:

```
let text = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
let length = text.length;
```

JavaScript String Methods

- There are 3 methods for extracting a part of a string:
  - slice(start, end)
  - substring(start, end)
  - substr(start, length)

```
let str = "Apple, Banana, Kiwi";
let part = str.slice(7, 13);
let part = str.slice(7);
```

- substr() is similar to slice(), the difference is that the second parameter specifies the length of the extracted part.

```
let str = "Apple, Banana, Kiwi";
let part = str.substr(7, 6);
```

JavaScript String Methods

- The replace() method replaces a specified value with another value in a string:

```
let text = "Please visit Microsoft!";
let newText =
text.replace("Microsoft", "W3Schools");
```

- The replace() method replaces only the first match:
- The replace() method is case sensitive. Writing MICROSOFT (with upper-case) will not work
- To replace case insensitive, use a regular expression with an /i flag (insensitive):

```
let text = "Please visit Microsoft!";
let newText =
text.replace(/MICROSOFT/i, "W3Schools");
```

- To replace all matches, use a regular expression with a /g flag (global match):

JavaScript String Methods

- A string is converted to upper case with toUpperCase():
- A string is converted to lower case with toLowerCase():

```
let text1 = "Hello World!";
let text2 = text1.toUpperCase();
```

- concat() joins two or more strings:
- The concat() method can be used instead of the plus operator. These two lines do the same:

```
text = "Hello" + " " + "World!";
text = "Hello".concat(" ", "World!");
```

- The trim() method removes whitespace from both sides of a string:

```
let text = "      Hello World!      ";
let text.trim()    // Returns "Hello
World!"
```

JavaScript String Search

- JavaScript methods for searching strings:
  - String indexOf()
  - String lastIndexOf()
  - String search()
  - String match()
  - String includes()

```
let str = "Please locate where 'locate' occurs!";
str.indexOf("locate");
```

- The indexOf() method returns the index of (the position of) the first occurrence of a specified text in a string:
- The lastIndexOf() method returns the index of the last occurrence of a specified text in a string.
- Both indexOf(), and lastIndexOf() return -1 if the text is not found:
- Both methods accept a second parameter as the starting position for the search.

JavaScript String Search

- The search() method searches a string for a specified value and returns the position of the match
- The two methods, indexOf() and search(), are **Not** equal?
  - The search() method cannot take a second start position argument.
  - The indexOf() method cannot take powerful search values (regular expressions).
- The match() method searches a string for a match against a regular expression, and returns the matches, as an Array object.

```
let text = "The rain in SPAIN stays mainly in the plain";
text.match(/ain/g);
```

- The includes() method returns true if a string contains a specified value.
- The startsWith() method returns true if a string begins with a specified value, otherwise false.
- The endsWith() method returns true if a string ends with a specified value, otherwise false.

JavaScript Arrays

- An array is a special variable, which can hold more than one value:

```
const cars = ["Saab", "Volvo", "BMW"];
```

- You can also create an array, and then provide the elements:

```
const cars = [];
cars[0]= "Saab";
cars[1]= "Volvo";
cars[2]= "BMW";
```

- The following example also creates an Array, and assigns values to it:

```
const cars = new Array("Saab", "Volvo", "BMW");
```

- You access an array element by referring to the index number:

```
const cars = ["Saab", "Volvo", "BMW"];
let car = cars[0];
```

JavaScript Arrays

- To change an Array Element
- This statement changes the value of the first element in cars:

```
const cars = ["Saab", "Volvo", "BMW"];
cars[0] = "Opel";
```

- With JavaScript, the full array can be accessed by referring to the array name:

```
const cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
```

- JavaScript variables can be objects. Arrays are special kinds of objects.
  - Because of this, you can have variables of different types in the same Array.
  - You can have objects in an Array. You can have functions in an Array. You can have arrays in an Array:

```
myArray[0] = Date.now;
myArray[1] = myFunction;
myArray[2] = myCars;
```

Array Properties and Methods

- The length property of an array returns the length of an array (the number of array elements).

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let length = fruits.length;
```

- To access the last array element you can write:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits[fruits.length - 1];
```

- One way to loop through an array, is using a for loop:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fLen = fruits.length;
text = "<ul>";
for (let i = 0; i < fLen; i++) {
  text += "<li>" + fruits[i] + "</li>";}
text += "</ul>";
```

JavaScript Array Methods

- The JavaScript method toString() converts an array to a string of (comma separated) array values.

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.toString();
```

- The join() method also joins all array elements into a string.
- It behaves just like toString(), but in addition you can specify the separator:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.join(" * ");
```

Popping and Pushing

- The pop() method removes the last element from an array:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.pop();
```

- The pop() method returns the value that was "popped out":

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits.pop();
```

- The push() method adds a new element to an array (at the end):

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Kiwi");
```

- The push() method returns the new array length:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let length = fruits.push("Kiwi");
```

Popping and Pushing

- Shifting is equivalent to popping, working on the first element instead of the last.
- The shift() method removes the first array element and "shifts" all other elements to a lower index.

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.shift();
```

- The unshift() method adds a new element to an array (at the beginning), and keeping older elements:

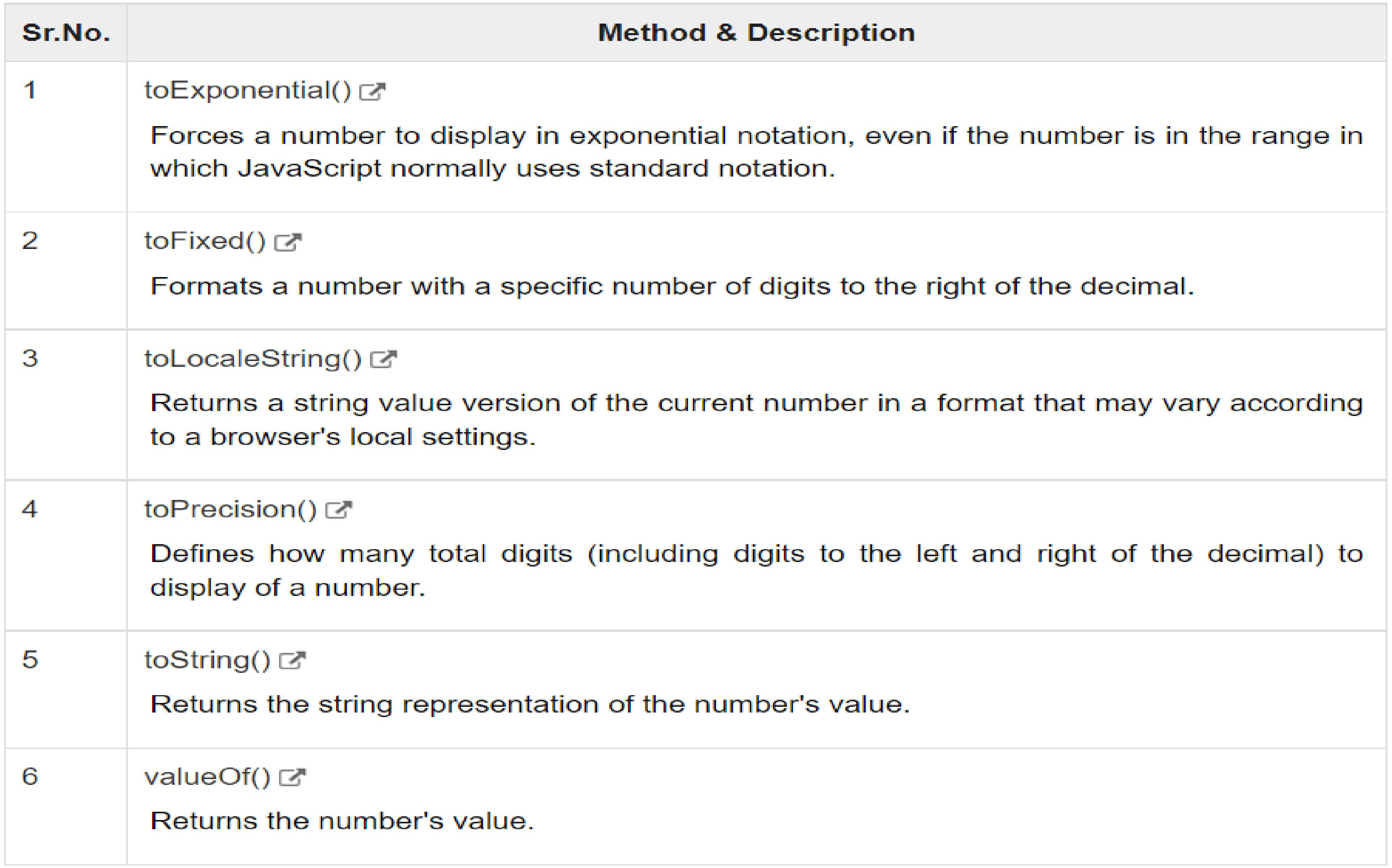
```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");
```

- The length property provides an easy way to append a new element to an array:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[fruits.length] = "Kiwi";
```

Numbers

- Number Methods



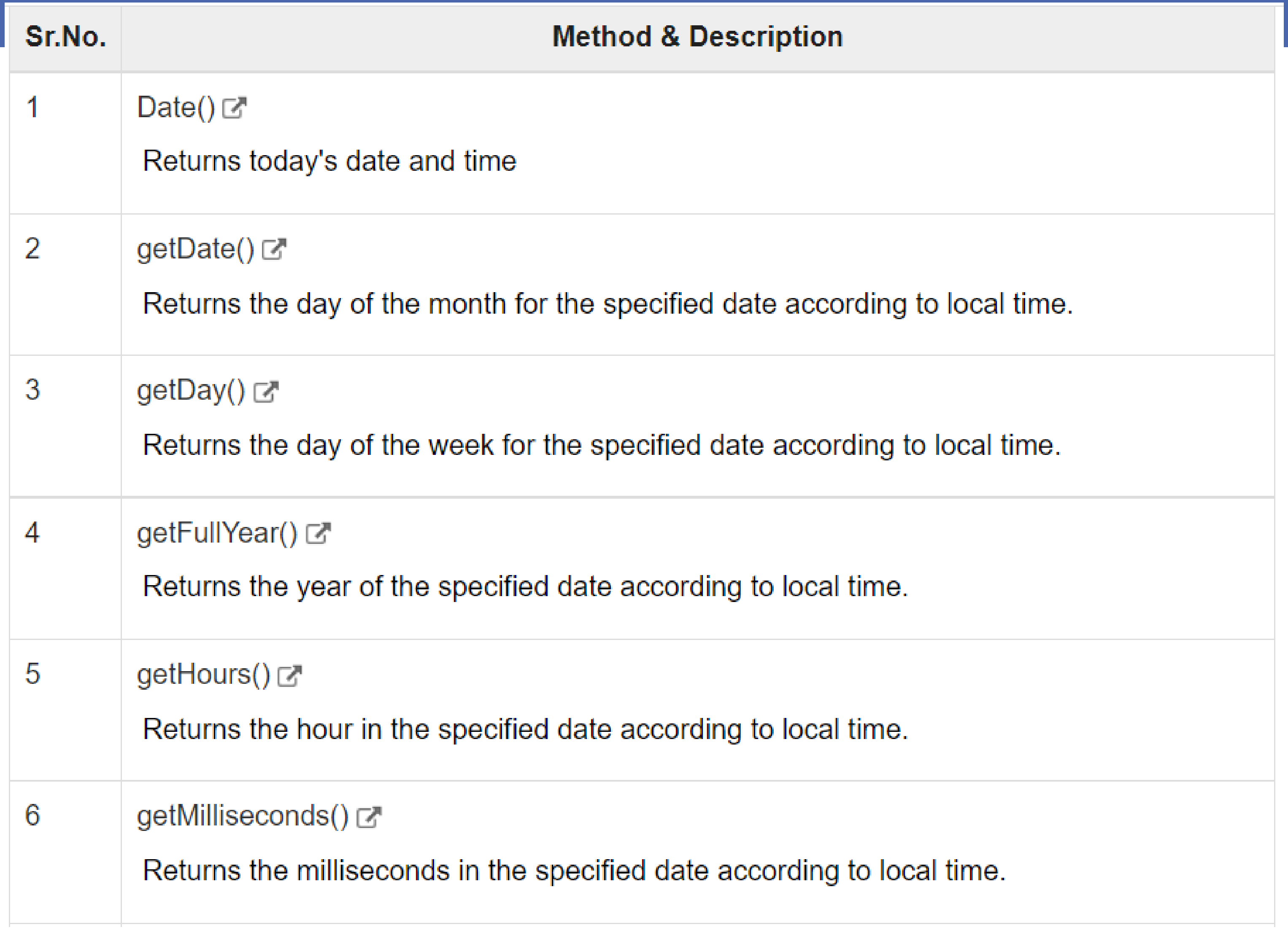
Sr.No.	Method & Description
1	toExponential() Forces a number to display in exponential notation, even if the number is in the range in which JavaScript normally uses standard notation.
2	toFixed() Formats a number with a specific number of digits to the right of the decimal.
3	toLocaleString() Returns a string value version of the current number in a format that may vary according to a browser's local settings.
4	toPrecision() Defines how many total digits (including digits to the left and right of the decimal) to display of a number.
5	toString() Returns the string representation of the number's value.
6	valueOf() Returns the number's value.

Date

- The Date object is a datatype built into the JavaScript language. Date objects are created with the new Date( ).
- Once a Date object is created, a number of methods allow you to operate on it. Most methods simply allow you to get and set the year, month, day, hour, minute, second, and millisecond fields of the object

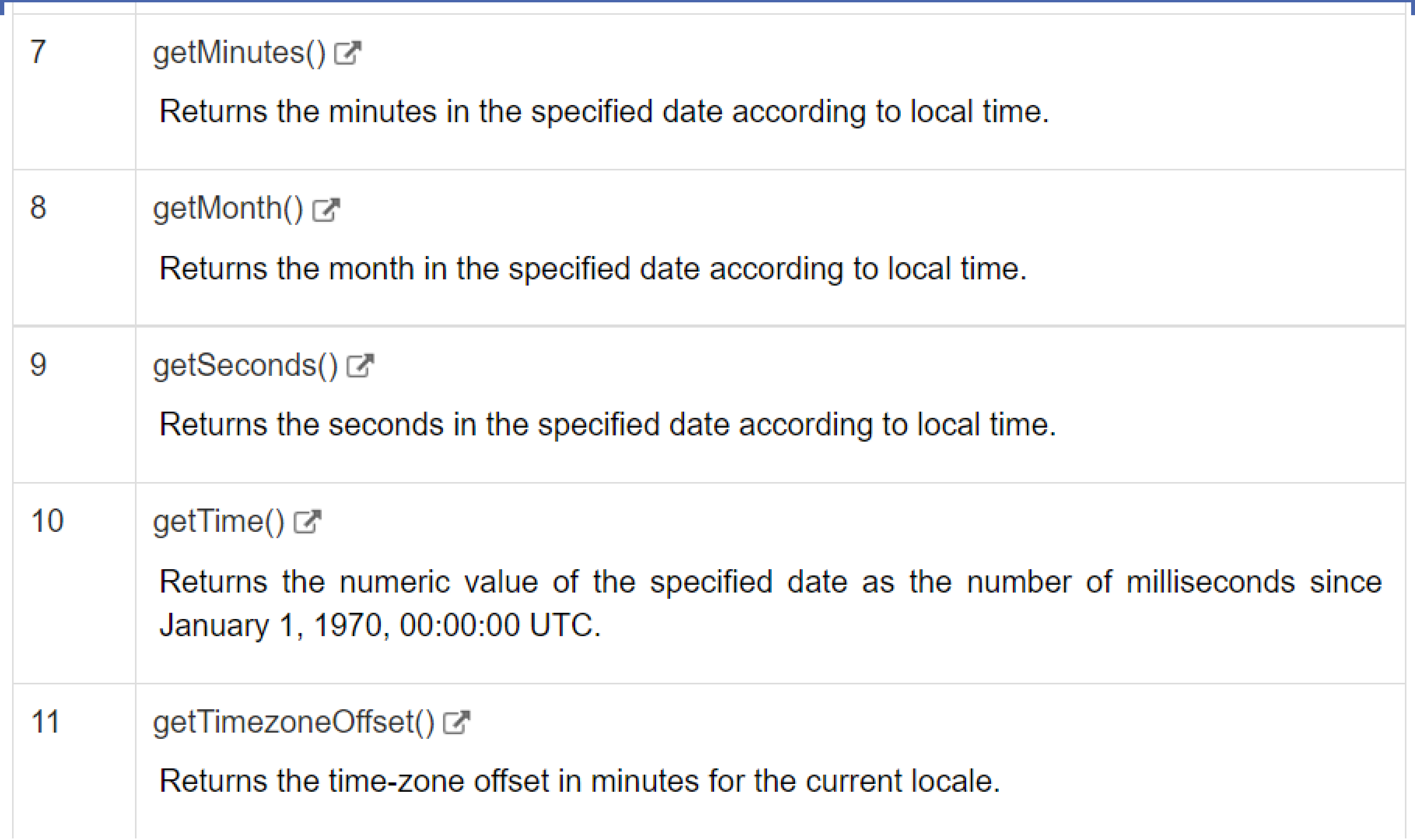
```
const d = new Date();
```

Date



Sr.No.	Method & Description
1	Date() Returns today's date and time
2	getDate() Returns the day of the month for the specified date according to local time.
3	getDay() Returns the day of the week for the specified date according to local time.
4	getFullYear() Returns the year of the specified date according to local time.
5	getHours() Returns the hour in the specified date according to local time.
6	getMilliseconds() Returns the milliseconds in the specified date according to local time.

Date



7	getMinutes() Returns the minutes in the specified date according to local time.
8	getMonth() Returns the month in the specified date according to local time.
9	getSeconds() Returns the seconds in the specified date according to local time.
10	getTime() Returns the numeric value of the specified date as the number of milliseconds since January 1, 1970, 00:00:00 UTC.
11	getTimezoneOffset() Returns the time-zone offset in minutes for the current locale.

Date

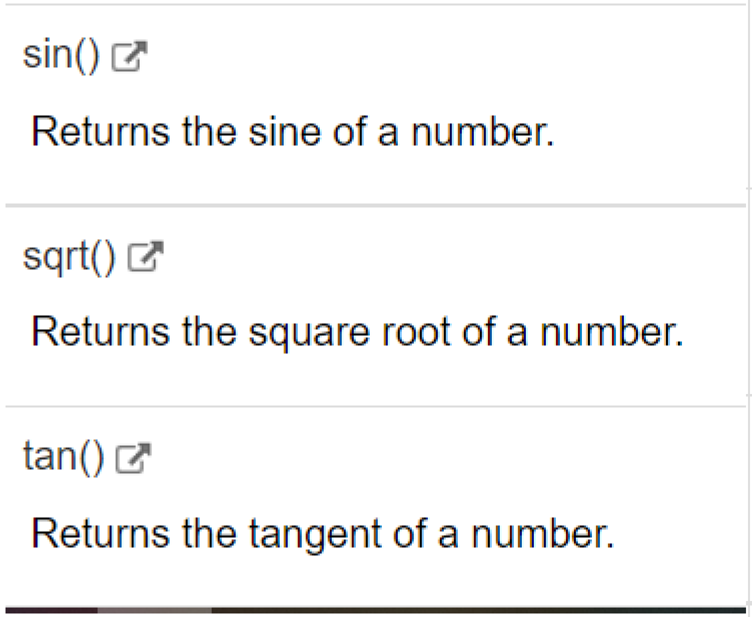
```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript getTime()</h2>
<p>The internal clock in JavaScript counts from midnight
January 1, 1970.</p>
<p>The getTime() function returns the number of milliseconds
since then:</p>
<p id="demo"></p>
<script>
const d = new Date();
document.getElementById("demo").innerHTML = d.getTime();
</script>
</body>
</html>
```

Math

- The math object provides you properties and methods for mathematical constants and functions.
- Thus, you refer to the constant pi as Math.PI and you call the sine function as Math.sin(x), where x is the method's argument.

```
var pi_val = Math.PI;
var sine_val = Math.sin(30);
```

Math



sin() ↗

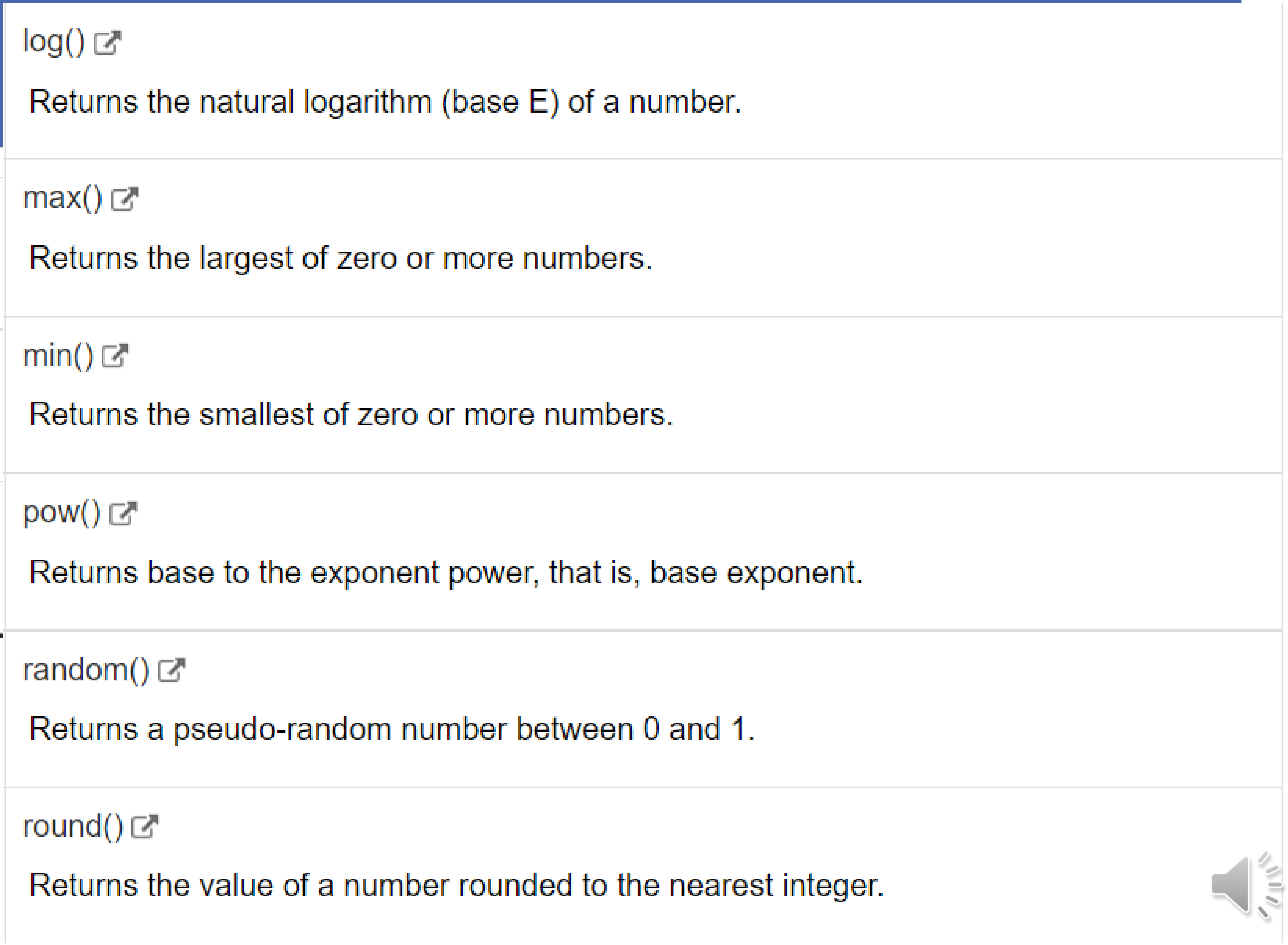
Returns the sine of a number.

sqrt() ↗

Returns the square root of a number.

tan() ↗

Returns the tangent of a number.



log() ↗

Returns the natural logarithm (base E) of a number.

max() ↗

Returns the largest of zero or more numbers.

min() ↗

Returns the smallest of zero or more numbers.

pow() ↗

Returns base to the exponent power, that is, base exponent.

random() ↗

Returns a pseudo-random number between 0 and 1.

round() ↗

Returns the value of a number rounded to the nearest integer.

Regular Expressions

- In JavaScript, regular expressions are often used with the two string methods: search() and replace().

Modifier	Description
i	Perform case-insensitive matching
g	Perform a global match (find all matches rather than stopping after the first match)
m	Perform multiline matching

Expression	Description
[abc]	Find any of the characters between the brackets
[0-9]	Find any of the digits between the brackets
(x|y)	Find any of the alternatives separated with |

Regular Expressions

- In JavaScript, regular expressions are often used with the two string methods: search() and replace().

```
<script>
let text = "\nIs th\nis it?";
let result = text.match(/^is/m);
document.getElementById("demo").innerHTM
L = "Result is: " + result;
</script>
```

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Regular Expressions</h2>
<p>Do a global search for the numbers 1 to 4 in a string:</p>
<p id="demo"></p>
<script>
let text = "123456789";
let result = text.match(/[1-4]/g);
document.getElementById("demo").innerHTML = result;
</script>
</body>
</html>
```

Regular Expressions

Metacharacter	Description
\d	Find a digit
\s	Find a whitespace character
\b	Find a match at the beginning of a word like this: \bWORD, or at the end of a word like this: WORD\b
\uxxxx	Find the Unicode character specified by the hexadecimal number xxxx

Quantifier	Description
n+	Matches any string that contains at least one *n*
n*	Matches any string that contains zero or more occurrences of *n*
n?	Matches any string that contains zero or one occurrences of *n*

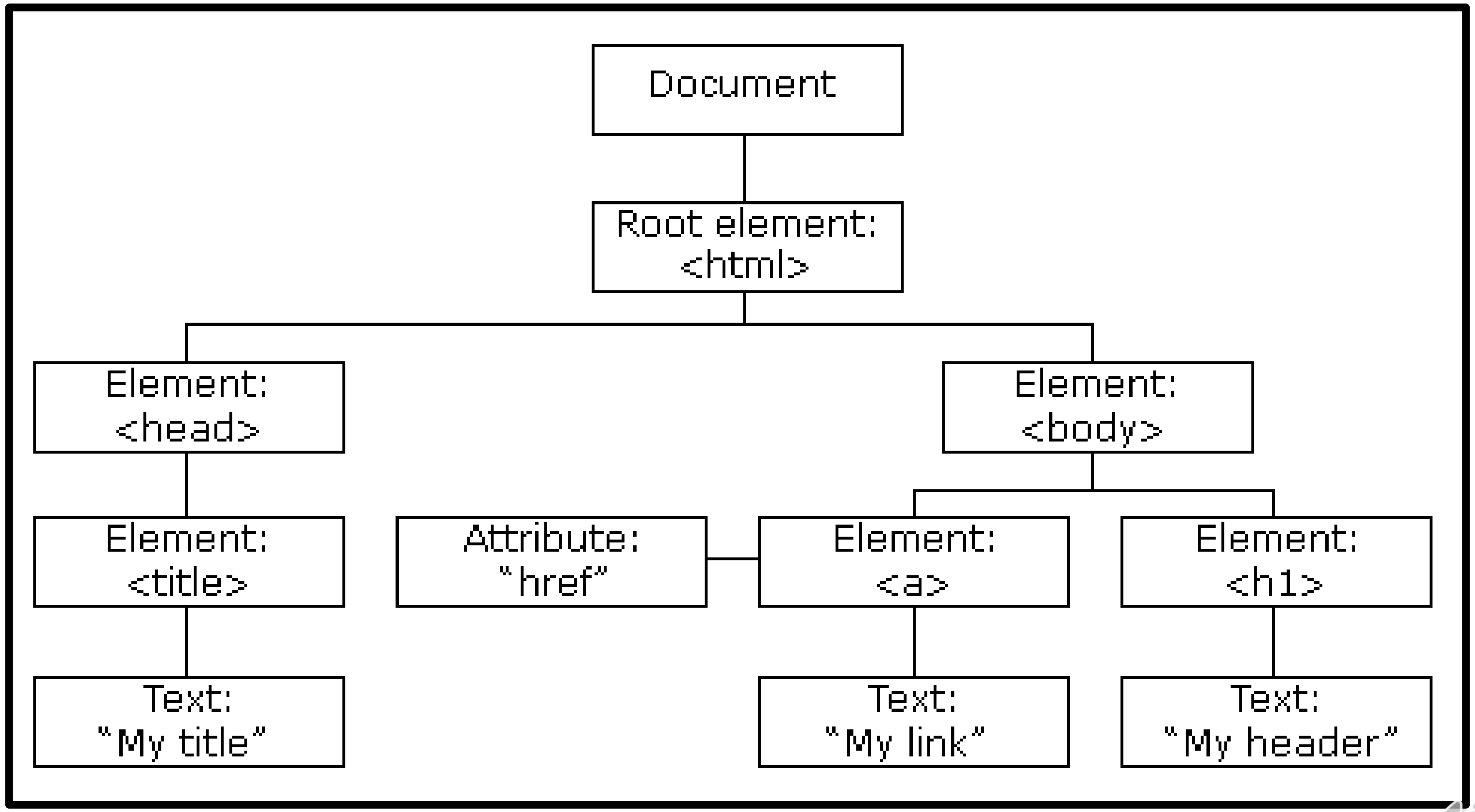
Regular Expressions

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Regular Expressions</h2>
<p>Search for the characters "LO" in the
<b>beginning</b> of a word in the phrase:</p>
<p>"HELLO, LOOK AT YOU!"</p>
<p>Found in position: <span id="demo"></span></p>
<script>
let text = "HELLO, LOOK AT YOU!";
let result = text.search(/\bLO/);
document.getElementById("demo").innerHTML =
result;
</script>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Regular Expressions</h2>
<p>Do a global search for an "l", followed by zero or more
"o" characters:</p>
<p id="demo"></p>
<script>
let text = "Hellooo World! Hello W3Schools!";
let result = text.match(/lo*/g);
document.getElementById("demo").innerHTML = result;
</script>
</body>
</html>
```

JavaScript HTML DOM

- 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:



JavaScript - HTML DOM Methods

- HTML DOM methods are actions you can perform (like add or delete).
- HTML DOM properties are values (of HTML Elements) that you can set or change.
- The HTML DOM can be accessed with JavaScript (and other programming languages).
- The following example changes the content (the innerHTML) of the <p> element with id="demo":

```
<html>
<body>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>
</body>
</html>
```

- In the example above, getElementById is a method, while innerHTML is a property.

JavaScript - HTML DOM Methods

- The most common way to access an HTML element is to use the id of the element.
- In the example above the getElementById method used id="demo" to find the element.
- The easiest way to get the content of an element is by using the innerHTML property.

```
<html>
<body>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>
</body>
</html>
```

JavaScript - HTML DOM Methods

- Finding HTML Elements

Method	Description
document.getElementById(*id*)	Find an element by element id
document.getElementsByTagName(*name*)	Find elements by tag name
document.getElementsByClassName(*name*)	Find elements by class name

- Changing HTML Elements

Property	Description
element.innerHTML = *new html content*	Change the inner HTML of an element
element.attribute = *new value*	Change the attribute value of an HTML element
element.style.*property* = *new style*	Change the style of an HTML element
Method	**Description**
element.setAttribute*(attribute, value)*	Change the attribute value of an HTML element

JavaScript - HTML DOM Methods

➢ Adding and Deleting Elements

Method	Description
document.createElement(*element*)	Create an HTML element
document.removeChild(*element*)	Remove an HTML element
document.appendChild(*element*)	Add an HTML element
document.replaceChild(*new, old*)	Replace an HTML element
document.write(*text*)	Write into the HTML output stream

➢ Adding Events Handlers

Method	Description
document.getElementById(*id*).onclick = function(){*code*}	Adding event handler code to an onclick event

JavaScript - HTML DOM Methods

```
<!DOCTYPE html>
<html>
<body>
<img id="myImage" src="smil
ey.gif">
<script>
document.getElementById("my
Image").src = "landscape.jp
g";
</script>

</body>
</html>
```

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

<p>Bla bla bla</p>
<script>
document.write(Date());
</script>

<p>Bla bla bla</p>

</body>
</html>
```

```
<html>
<body>

<p id="p2">Hello
World!</p>

<script>
document.getElementById("p
2").style.color = "blue";
</script>

</body>
</html>
```

JavaScript - if...else Statement

- JavaScript supports conditional statements which are used to perform different actions based on different conditions. Here we will explain the if..else statement.

```
<html>
<body>
 <script type = "text/javascript"> var age =
15;
if( age > 18 )
{ document.write("<b>Qualifies for
driving</b>"); }
else { document.write("<b>Does not qualify for
driving</b>"); }
</script>
<p>Set the variable to different value and then
try...</p>
</body>
 </html>
```

```
<script type = "text/javascript">
var book = "maths";
if( book == "history" )
 { document.write("<b>History Book</b>");
}
else if( book == "maths" )
 { document.write("<b>Maths Book</b>"); }
else if( book == "economics" )
{ document.write("<b>Economics
Book</b>"); }
else { document.write("<b>Unknown
Book</b>"); }
 </script>
```

JavaScript - Switch Case

- Starting with JavaScript 1.2, you can use a switch statement which handles exactly this situation, and it does so more efficiently than repeated if...else if statements.

```
<script type = "text/javascript">
 var grade = 'A’;
document.write("Entering switch block<br />");
switch (grade) { case 'A': document.write("Good
job<br />"); break;
case 'B': document.write("Pretty good<br />"); break;
case 'C': document.write("Passed<br />"); break;
case 'D': document.write("Not so good<br />"); break;
case 'F': document.write("Failed<br />"); break;
default: document.write("Unknown grade<br />") }
document.write("Exiting switch block");
</script>
```

JavaScript - While Loops

- The most basic loop in JavaScript is the while loop. The purpose of a while loop is to execute a statement or code block repeatedly as long as an expression is true. Once the expression becomes false, the loop terminates.

```
<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>
```

The do...while Loop

- The do...while loop is similar to the while loop except that the condition check happens at the end of the loop. This means that the loop will always be executed at least once, even if the condition is false.

```
<html>
<body>
<script type = "text/javascript">
var count = 0; document.write("Starting Loop" + "<br />");
do { document.write("Current Count : " + count + "<br />");
count++; }
while (count < 5);
document.write ("Loop stopped!");
</script>
<p>Set the variable to different value and then try...</p>
 </body>
</html>
```

JavaScript - For Loop

- The 'for' loop is the most compact form of looping. It includes the following three important parts –
  - The loop initialization
  - The test statement
  - The iteration statement
- You can put all the three parts in a single line separated by semicolons.

```
<html>
<body>
<script type = "text/javascript">
var count;
document.write("Starting Loop" + "<br />");
for(count = 0; count < 10; count++)
 { document.write("Current Count : " + count );
document.write("<br `/>"); }
 document.write("Loop stopped!");
</script>
 <p>Set the variable to different value and then
try...</p>
</body>
 </html>
```

JavaScript for...in loop

- The for...in loop is used to loop through an object's properties.

```
<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>
```

JavaScript - Loop Control

- JavaScript provides full control to handle loops and switch statements.
- The break statement, is used to exit a loop early, breaking out of the enclosing curly braces.

```
<html>
<body>
<script type = "text/javascript">
var x = 1;
document.write("Entering the loop<br /> ");
 while (x < 20) { if (x == 5)
 { break; // breaks out of loop completely }
 x = x + 1;
document.write( x + "<br />"); }
document.write("Exiting the loop!<br /> ");
 </script>
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

JavaScript - Loop Control

- The continue statement tells the interpreter to immediately start the next iteration of the loop and skip the remaining code block.
- When a continue statement is encountered, the program flow moves to the loop check expression immediately and if the condition remains true, then it starts the next iteration, otherwise the control comes out of the loop.

```
<html>
<body>
<script type = "text/javascript">
var x = 1;
document.write("Entering the loop<br /> ");
 while (x < 10)
{ x = x + 1;
if (x == 5) { continue; // skip rest of the
loop body }
document.write( x + "<br />"); }
document.write("Exiting the loop!<br /> ");
</script>
<p>Set the variable to different value and
then try...</p>
</body>
</html>
```

JavaScript - Loop Control

- Starting from JavaScript 1.2, a label can be used with break and continue to control the flow more precisely.
- A label is simply an identifier followed by a colon (:) that is applied to a statement or a block of code.
- Note – Line breaks are not allowed between the ‘continue’ or ‘break’ statement and its label name. Also, there should not be any other statement in between a label name and associated loop.

JavaScript - Loop Control

```
<html>
<body>
<script type = "text/javascript">
document.write("Entering the loop!<br /> ");
outerloop: // This is the label name
for (var i = 0; i < 5; i++)
{ document.write("Outerloop: " + i + "<br />");
innerloop:
for (var j = 0; j < 5; j++)
{ if (j > 3 ) break ; // Quit the innermost loop
if (i == 2) break innerloop; // Do the same thing
if (i == 4) break outerloop; // Quit the outer loop
document.write("Innerloop: " + j + " <br />"); } }
document.write("Exiting the loop!<br /> ");
</script>
</body>
</html>
```

JavaScript - Page Redirection

- You might have encountered a situation where you clicked a URL to reach a page X but internally you were directed to another page Y. It happens due to page redirection.
- There could be various reasons why you would like to redirect a user from the original page.
  - You did not like the name of your domain and you are moving to a new one.
  - You have built-up various pages based on browser versions or their names or may be based on different countries.
  - The Search Engines may have already indexed your pages. But while moving to another domain, you would not like to lose your visitors coming through search engines.

JavaScript - Page Redirection

- It is quite simple to do a page redirect using JavaScript at client side. To redirect your site visitors to a new page, you just need to add a line in your head section as follows.

```
<html>
<head>
<script type = "text/javascript">
function Redirect()
{ window.location = "https://www.tutorialspoint.com"; }
</script>
</head>
<body>
<p>Click the following button, you will be redirected to home page.</p>
<form>
<input type = "button" value = "Redirect Me" onclick = "Redirect();" />
</form>
</body>
</html>
```

JavaScript - Page Redirection

```
<html>
<head>
<script type = "text/javascript">
function Redirect()
{ window.location =
"https://www.tutorialspoint.com"; }
document.write("You will be redirected
to main page in 10 sec.");
setTimeout('Redirect()', 10000);
</script>
</head>
<body>
</body>
</html>
```

```
<html>
<head>
<script type = "text/javascript">
var browsername = navigator.appName; I
f( browsername == "Netscape" )
{ window.location =
"http://www.location.com/ns.htm"; }
else if ( browsername =="Microsoft Internet
Explorer")
{ window.location =
"http://www.location.com/ie.htm"; }
else { window.location =
"http://www.location.com/other.htm"; }
</script>
</head>
<body>
</body>
</html>
```

JavaScript - Dialog Boxes

- JavaScript supports three important types of dialog boxes. These dialog boxes can be used to raise and alert, or to get confirmation on any input or to have a kind of input from the users.

```
<html>
<head>
<script type = "text/javascript">
function Warn()
{ alert ("This is a warning message!");
document.write ("This is a warning message!"); }
</script>
 </head>
<body>
 <p>Click the following button to see the result: </p>
<form> <input type = "button" value = "Click Me" onclick = "Warn();" />
</form>
</body>
</html>
```

JavaScript - Dialog Boxes

➢ Confirmation Dialog Box

```
<html>
 <head>
<script type = "text/javascript">
function getConfirmation()
{ var retVal = confirm("Do you want to continue ?");
if( retVal == true )
{ document.write ("User wants to continue!"); return true; }
else
{ document.write ("User does not want to continue!"); return false; } }
</script>
</head>
<body>
<p>Click the following button to see the result: </p>
<form> <input type = "button" value = "Click Me" onclick = "getConfirmation();" />
</form>
</body> </html>
```

JavaScript - Dialog Boxes

- Prompt Dialog Box

```
<html>
<head>
<script type = "text/javascript">
function getValue()
{ var retVal = prompt("Enter your name : ", "your name here");
document.write("You have entered : " + retVal); }
</script>
</head>
<body>
<p>Click the following button to see the result: </p>
<form> <input type = "button" value = "Click Me" onclick = "getValue();" />
</form>
</body>
</html>
```

JavaScript Use Strict

- The "use strict" directive was new in ECMAScript version 5.
- It is not a statement, but a literal expression, ignored by earlier versions of JavaScript.
- All modern browsers support "use strict" except Internet Explorer 9 and lower.
- It helps you to write cleaner code, like preventing you from using undeclared variables.
- As an example, in normal JavaScript, mistyping a variable name creates a new global variable. In strict mode, this will throw an error, making it impossible to accidentally create a global variable.

JavaScript Use Strict

- Strict mode is declared by adding "use strict"; to the beginning of a script or a function.
- Declared at the beginning of a script, it has global scope (all code in the script will execute in strict mode):

```
"use strict";
x = 3.14;       // This will cause an error
because x is not declared
```

Not Allowed in Strict Mode

- Using a variable, without declaring it, is not allowed:
- Using an object, without declaring it, is not allowed:
- Deleting a variable (or object) is not allowed.
- Deleting a function is not allowed.
- Duplicating a parameter name is not allowed:
- Octal numeric literals are not allowed:
- Octal escape characters are not allowed:
- Writing to a read-only property is not allowed:
- Writing to a get-only property is not allowed:

JavaScript Classes

- Use the keyword class to create a class.
- Always add a method named constructor():

```
class Car {
  constructor(name, year)
{
    this.name = name;
    this.year = year;
  }
}
```

- A JavaScript class is not an object.
- It is a template for JavaScript objects.

JavaScript Classes

- When you have a class, you can use the class to create objects:

```
let myCar1 = new Car("Ford", 2014);
let myCar2 = new Car("Audi", 2019);
```

- Class methods are created with the same syntax as object methods.
- Use the keyword class to create a class.
- Always add a constructor() method.
- Then add any number of methods.

JavaScript Classes

- Create a Class method named "age", that returns the Car age:

```
class Car {
  constructor(name, year) {
    this.name = name;
    this.year = year;
  }
  age() {
    let date = new Date();
    return date.getFullYear() - this.year;
  }
}

let myCar = new Car("Ford", 2014);
document.getElementById("demo").innerHTML =
"My car is " + myCar.age() + " years old.";
```

JavaScript Class Inheritance

- To create a class inheritance use the extends keyword.
- A class created with a class inheritance inherits all the methods from another class.
- Create a class named "Model“ which will inherit the methods from the "Car" class:

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  present() {
    return 'I have a ' + this.carname;
  }}
class Model extends Car {
  constructor(brand, mod) {
    super(brand);
    this.model = mod;
  }
  show() {
    return this.present() + ', it is a ' + this.model;
  }}
let myCar = new Model("Ford", "Mustang");
document.getElementById("demo").innerHTML = myCar.show();
```

JavaScript Class Inheritance

- The super() method refers to the parent class.
- By calling the super() method in the constructor method, we call the parent's constructor method and gets access to the parent's properties and methods.
- Unlike functions, and other JavaScript declarations, class declarations are not hoisted.
- That means that you must declare a class before you can use it:

```
//You cannot use the class yet.
//myCar = new Car("Ford")
//This would raise an error.
class Car {
  constructor(brand) {
    this.carname = brand;
  }
}
//Now you can use the class:
let myCar = new Car("Ford")
```

JavaScript - Animation

- You can use JavaScript to create a complex animation having, but not limited to, the following elements –

  - Fireworks
  - Fade Effect
  - Roll-in or Roll-out
  - Page-in or Page-out
  - Object movements

JavaScript - Animation

- JavaScript can be used to move a number of DOM elements (<img />, <div> or any other HTML element) around the page according to some sort of pattern determined by a logical equation or function.
- JavaScript provides the following functions to be frequently used in animation programs.
  - setTimeout(function, duration) − This function calls function after duration milliseconds from now.
  - setInterval(function, duration) − This function calls function after every duration milliseconds.
  - clearTimeout(setTimeout_variable) − This function calls clears any timer set by the setTimeout() functions.

JavaScript - Animation

- To demonstrate how to create HTML animations with JavaScript, we will use a simple web page:

```
<!DOCTYPE html>
<html>
<body>
<h1>My First JavaScript Animation</h1>
<div id="animation">My animation will go here</div>
</body>
</html>
```

- All animations should be relative to a container element.

```
<div id ="container">
  <div id ="animate">My animation will go
here</div>
</div>
```

JavaScript - Animation

- The container element should be created with style = "position: relative".
- The animation element should be created with style = "position: absolute".

```
#container {
  width: 400px;
  height: 400px;
  position: relative;
  background: yellow;
}
#animate {
  width: 50px;
  height: 50px;
  position: absolute;
  background: red;
}
```

JavaScript - Animation

- JavaScript animations are done by programming gradual changes in an element's style.
- The changes are called by a timer. When the timer interval is small, the animation looks continuous.
- The basic code is:

```
function myMove() {
  let id = null;
  const elem = document.getElementById("animate");
  let pos = 0;
  clearInterval(id);
  id = setInterval(frame, 5);
  function frame() {
    if (pos == 350) {
      clearInterval(id);
    } else {
      pos++;
      elem.style.top = pos + 'px';
      elem.style.left = pos + 'px';    } }}
```

JavaScript - Errors & Exceptions Handling

- There are three types of errors in programming: (a) Syntax Errors, (b) Runtime Errors, and (c) Logical Errors.
- Syntax errors, also called parsing errors, occur at compile time in traditional programming languages and at interpret time in JavaScript.
- For example, the following line causes a syntax error because it is missing a closing parenthesis.

```
<script type = "text/javascript">
   <!--
      window.print(;
   //-->
</script>
```

JavaScript - Errors & Exceptions Handling

- Runtime errors, also called exceptions, occur during execution (after compilation/interpretation).
- For example, the following line causes a runtime error because here the syntax is correct, but at runtime, it is trying to call a method that does not exist.

```
<script type = "text/javascript">
   <!--
      window.printme();
   //-->
</script>
```

JavaScript - Errors & Exceptions Handling

- Logic errors can be the most difficult type of errors to track down. These errors are not the result of a syntax or runtime error. Instead, they occur when you make a mistake in the logic that drives your script and you do not get the result you expected.
- You cannot catch those errors, because it depends on your business requirement what type of logic you want to put in your program.

The try...catch...finally Statement

- The latest versions of JavaScript added exception handling capabilities. JavaScript implements the try...catch...finally construct as well as the throw operator to handle exceptions.
- You can catch programmer-generated and runtime exceptions, but you cannot catch JavaScript syntax errors.

```
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
}
```

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

The try...catch...finally Statement

```
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 = "Error: " + err + ".";
  }
  finally {
    document.getElementById("demo").value = "";
  }}
```

JavaScript Scope

- Before ES6 (2015), JavaScript had only Global Scope and Function Scope.
- ES6 introduced two important new JavaScript keywords: let and const.
- These two keywords provide Block Scope in JavaScript.
- Variables declared inside a { } block cannot be accessed from outside the block.
- Variables declared with the var keyword can NOT have block scope and can be accessed from outside the block.

```
{
  let x = 2;
 var y = 2;
}
// x can NOT be used here
// y can be used here
```

JavaScript Scope

- JavaScript has function scope: Each function creates a new scope.
- Variables defined inside a function are not accessible (visible) from outside the function.
- Variables declared with var, let and const are quite similar when declared inside a function.
- Local variables are created when a function starts, and deleted when the function is completed.

JavaScript Scope

- If you assign a value to a variable that has not been declared, it will automatically become a GLOBAL variable.
- This code example will declare a global variable carName, even if the value is assigned inside a function.

```
myFunction();

// code here can use carName

function myFunction() {
  carName = "Volvo";
}
```

- Variables declared Globally (outside any function) have Global Scope.
- Global variables can be accessed from anywhere in a JavaScript program.

JavaScript Arrow Function

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

```
let myFunction = (a, b) => a * b;
```

- Before

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

- After

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

JavaScript Arrow Function

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

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

- If you have parameters, you pass them inside the parentheses:

```
hello = (val) => "Hello " + val;
```

- In fact, if you have only one parameter, you can skip the parentheses as well:

```
hello = val => "Hello " + val;
```

JavaScript / jQuery DOM Selectors

- jQuery was created in 2006 by John Resig. It was designed to handle Browser Incompatibilities and to simplify HTML DOM Manipulation, Event Handling, and Animations.
- For more than 10 years, jQuery has been the most popular JavaScript library in the world.
- However, after JavaScript Version 5 (2009), most of the jQuery utilities can be solved with a few lines of standard JavaScript:

JavaScript / jQuery DOM Selectors

- Finding HTML Element by Id
- Return the element with id="id01":
- Using jQuery

```
myElement = $("#id01");
```

- Using JavaScript

```
myElement =
document.getElementById("id01");
```

JavaScript / jQuery DOM Selectors

- Get Text Content
- Get the inner text of an HTML element:
- Using jQuery

```
myText = $("#02").text();
```

- Using JavaScript

```
myText =
document.getElementById("02").textContent;
```

Thank You