JavaScript

- JavaScript is one of the most popular programming languages and is used to add interactivity to webpages, process data, as well as create various applications (mobile apps, desktop apps, games, and more).
- In HTML, JavaScript code must be inserted between <script> and </script> tags.
- JavaScript can be placed in the HTML page's <body> and <head> sections.

- The script, which is placed in the head section, will be executed before the <body> is rendered. If you want to get elements in the body, it's a good idea to place your script at the end of the body tag. **Ref: first.html**

```
<!DOCTYPE html>
<html>
   <head>
   </head>
   <body>
      <h1>Welcome to JavaScript</h1>
      <script>
         document.write("Hello World!");
      </script>
   </body>
</html>
```

- The document.write() function writes a string into our HTML document. This function can be used to write text, HTML, or both.
- The document.write() method should be used only for testing. We shall discuss other output mechanisms in the due course.
- Just like in HTML, we can use HTML tags to format text in JavaScript.
- For example, we can output the text as a heading.

Ref:first02.html

```
<!DOCTYPE html>
<html>
   <head>
   </head>
   <body>
   <h1>Welcome to JavaScript</h1>
   <script>
   document.write("<h2>Hello World!</h2>");
      </script>
   </body>
</html>
```

- You can output almost everything in the webpage using JavaScript.

- You can place any number of scripts in an HTML document.
- There is also a <noscript> tag. Its content will be shown if the client's browser doesn't support JS scripts.
- It's a good idea to place scripts at the bottom of the <body> element. This can improve page load, because HTML display is not blocked by scripts loading.
- JavaScript is the default HTML scripting language.

Ref:first03.html

```
<!DOCTYPE html>
<html>
   <head>
   </head>
   <body>
   <h1>Welcome to JavaScript</h1>
   <script>
   alert(“This is an alert box”);
      </script>
   </body>
</html>
```

- You can output almost everything in the webpage using JavaScript.

- External JavaScript
- Scripts can also be placed in external files.
- External scripts are useful and practical when the same code is used in a number of different web pages.
- JavaScript files have the file extension .js.
- Having JS scripts in separate files makes your code much readable and clearer.
-

Ref:first04.html

```
<!DOCTYPE html>
<html>
  <head>
    <script src=”js/demo.js”></script>
  </head>
  <body>
    <h1>Welcome to JavaScript</h1>
  </body>
</html>
```

demo.js

```
alert(“This is an alert box”);
```

- External scripts cannot contain <script> tags.

- Placing a JavaScript in an external file has the following advantages:
- It separates HTML and code.
- It makes HTML and JavaScript easier to read and maintain.
- Cached JavaScript files can speed up page loads.
- Code after a double slash //, or between /* and */, is treated as a comment.
- Comments are used to describe and explain what the code is doing.

- Use the var keyword to declare a variable:
- var x = 10;
- JavaScript is case sensitive. For example, the variables lastName and lastname, are two different variables.
- A variable can be declared without a value.
- A variable declared without a value will have the value undefined.
- Every written "instruction" is called a statement. JavaScript statements are separated by semicolons.

- Naming rules:
- The first character must be a letter, an underscore (_), or a dollar sign ($). Subsequent characters may be letters, digits, underscores, or dollar signs.
- Numbers are not allowed as the first character.
- Variable names cannot include a mathematical or logical operator in the name. For instance, 2*something or this+that;
- JavaScript names must not contain spaces.
- Hyphens are not allowed in JavaScript. It is reserved for subtractions.

- You must not use any special symbols, like my#num, num%, etc.
- You should use any of the JavaScript reserved words.

Reserved Words in JavaScript

abstract	else	instanceof	switch
boolean	enum	int	synchronized
break	export	interface	this
byte	extends	long	throw
case	false	native	throws
catch	final	new	transient
char	finally	null	true
class	float	package	try
const	for	private	typeof
continue	function	protected	var
debugger	goto	public	void
default	if	return	volatile
delete	implements	short	while
do	import	static	with
double	in	super	

- JavaScript variables can hold many data types, such as numbers, strings, arrays, and more.
- Unlike many other programming languages, JavaScript does not define different types of numbers, like integers, short, long, floating-point, etc.
- JavaScript numbers can be written with or without decimals.
- var num = 42; // A number without decimals
- This variable can be easily changed to other types by assigning to it any other data type value, like:
num = 'some random string'.

- JavaScript numbers are always stored as double precision floating point numbers. **Ref: first05.html**

```
<script>
   var num = 10.345;
   document.write(num); document.write(“<br>”);
   num = 20;
   document.write(num); document.write(“<br>”);
   num = “Hello world!”;
   document.write(num); document.write(“<br>”);
</script>
```

- **Strings**
- JavaScript strings are used for storing and manipulating text.
- A string can be any text that appears within quotes. You can use single or double quotes.
- var name = 'John';
- var text = "My name is John Smith";
- You can use quotes inside a string, as long as they don't match the quotes surrounding the string.
- var text = "My name is 'John' ";
- You can get double quotes inside of double quotes using the escape character like this: \" or \' inside of single quotes.

Code	Outputs
\'	single quote
\"	double quote
\\	backslash
\n	new line
\r	carriage return
\t	tab
\b	backspace
\f	form feed

- **Booleans**
- In JavaScript Boolean, you can have one of two values, either true or false.
- These are useful when you need a data type that can only have one of two values, such as Yes/No, On/Off, True/False.
- Example:
- var isActive = true;
- var isHoliday = false;
- The Boolean value of 0 (zero), null, undefined, empty string is false.
- Everything with a "real" value is true.

- **Arithmetic Operators:**
- `+  -  *  /  %  ++  --`
- `var x = 10 + 5;`
- `document.write(x);  // Outputs 15`
- You can get the result of a string expression using the **eval**() function, which takes a string expression argument like eval("10 * 20 + 8") and returns the result. If the argument is empty, it returns undefined.

- **Multiplication**: **Ref: first06.html**
- var x = 10 * 5; document.write(x); // Outputs 50
- x = ‘10’ * 5; document.write(x); // Outputs 50
- x = ‘10’ * ‘5’; document.write(x); // Outputs 50
- x = s10 * 5; document.write(x); // Outputs NaN
- **Division:**
- var x = 10 / 0; document.write(x); // Outputs Infinity

- **Modulus:**
- Modulus (%) operator returns the division remainder.
- `var myVariable = 26 % 6;` //myVariable equals 2
- In JavaScript, the modulus operator is used not only on integers, but also on floating point numbers.
- **Assignment Operators:**
- `= += -= *= /= %=`
- You can use multiple assignment operators in one line, such as x -= y += 9.

- **Comparison Operators:**

Operator	Description	Example
==	Equal to	5 == 10 false
===	Identical (equal and of same type)	5 === 10 false
!=	Not equal to	5 != 10 true
!==	Not Identical	10 !== 10 false
>	Greater than	10 > 5 true
>=	Greater than or equal to	10 >= 5 true
<	Less than	10 < 5 false
<=	Less than or equal to	10 <= 5 false

- **Comparison Operators** (contd.)
- When using operators, be sure that the arguments are of the same data type; numbers should be compared with numbers, strings with strings, and so on.
- **Logical Operators:**
- && || !
- Return value: true or false
- Conditional (Ternary) Operator
- Syntax: variable = (condition) ? value1: value2

- Concatenation operator (+) is an important String operator.

```
var mystring1 = "Hello ";

var mystring2 = "Universe.";

document.write(mystring1 + mystring2);
```

- The if Statement

```
if (condition) {
   statements
}
```

if else statement:

```
if (expression) {
   // executed if condition is true
}
else {
   // executed if condition is false
}
```

- if else if statement

```
var course = 1;
if (course == 1) {
   document.write("<h1>HTML Tutorial</h1>");
} else if (course == 2) {
   document.write("<h1>CSS Tutorial</h1>");
} else {
   document.write("<h1>JavaScript Tutorial</h1>");
}
```

- **Switch Statement**: Syntax:

```
switch (expression) {
  case n1:
     statements
     break;
  case n2:
     statements
     break;
  default:
     statements
}
```

- The switch expression is evaluated once. The value of the expression is compared with the values of each case. If there is a match, the associated block of code is executed.

- **for** statement. Syntax:

  for (statement 1; statement 2; statement 3) {

  code block to be executed

  }

- Statement 1 is executed before the loop (the code block) starts.
- Statement 2 defines the condition for running the loop (the code block).
- Statement 3 is executed each time after the loop (the code block) has been executed. **Ref: for.html**
- You can have multiple nested for loops.

- The while loop: Syntax:

  while (condition) {

  code block

  }

- Example.

```
var i=0;
while (i<=10) {
   document.write(i + "<br />");
   i++;
}
```

- **The do...while loop:**
- Syntax:

```
do {
    code block
} while (condition);
```

- Note the **semicolon** used at the end of the do...while loop.
- **Ref: doWhile.html**

- Break:
- The break statement "jumps out" of a loop and continues executing the code after the loop.

```
for (i = 0; i <= 10; i++) {
   if (i == 5) {
     break;
   }
   document.write(i + "<br>");
}
```

- **<u>Continue:</u>** **Ref: continue.html**

  The continue statement breaks only one iteration in the loop, and continues with the next iteration.

```
for (i = 0; i <= 10; i++) {
   if (i == 5) {
      continue;
   }
   document.write(i + "<br>");
}
```

- **<u>JavaScript Functions:</u>**
- A JavaScript function is a block of code designed to perform a particular task.
- The main advantage of using functions is Code reuse.
- Defining a function:

```
function name() {
  //code to be executed
}
```

- **Ref: myFunction.html**

- Functions can take parameters.
- You can define a single function, and pass different parameter values (arguments) to it.

  Ref: sayHello.html

```
function sayHello(name) {
   alert("Hi, " + name);
}
sayHello("Hari");
sayHello("Saroj");
sayHello("Krishna");
```

- You can define multiple parameters for a function by comma-separating them.

  function myFunc(x, y) {

    // some code

  }

Ref: sayHello2.html

```
function sayHello2(name, age) {

  document.write( name + " is " + age + " years old.");

}
sayHello2("John", 20)
```

- When calling the function, provide the arguments in the same order in which you defined them.

- If you pass more arguments than are defined, they will be assigned to an array called arguments. They can be used like this: arguments[0], arguments[1], etc.
- If a function is called with missing arguments (fewer than declared), the missing values are set to undefined, which indicates that a variable has not been assigned a value.

- A function can have an optional return statement. It is used to return a value from the function.
- This statement is useful when making calculations that require a result.
- When JavaScript reaches a return statement, the function stops executing.

```
function myFunction(a, b) {
   return a * b;
}
var x = myFunction(5, 6);       // x gets value 30
```

- If you do not return anything from a function, it will return undefined.

- JavaScript offers three types of popup boxes, the Alert, Prompt, and Confirm boxes.
- An alert box is used when you want to ensure that information gets through to the user.
- When an alert box pops up, the user must click OK to proceed.
- The alert function takes a single parameter, which is the text displayed in the popup box.

- Example:
- alert("Do you really want to leave this page?");
- To display line breaks within a popup box, use a backslash followed by the character n.
- alert("Hello\nHow are you?");
- Be careful when using alert boxes, as the user can continue using the page only after clicking OK.

- A **prompt box** is often used to have the user input a value before entering a page.
- When a prompt box pops up, the user will have to click either OK or Cancel to proceed after entering the input value.
- If the user clicks OK, the box returns the input value. If the user clicks Cancel, the box returns null.
- The prompt() method takes two parameters.
- - The first is the label, which you want to display in the text box.
- - The second is a default string to display in the text box (optional).

- Example: **Ref: prompt.html**
- var user = prompt("Please enter your name");
- alert(user);
- When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value. Do not overuse this method, because it prevents the user from accessing other parts of the page until the box is closed.

- A **confirm box** is often used to have the user verify or accept something.
- When a confirm box pops up, the user must click either **OK** or **Cancel** to proceed.
- If the user clicks **OK**, the box returns **true**. If the user clicks **Cancel**, the box returns **false**.
- Do not overuse this method, because it also prevents the user from accessing other parts of the page until the box is closed.

Example: **Ref: confirm.html**

```
var result = confirm("Do you really want to leave this page?");

if (result == true) {

  alert("Thanks for visiting");

}

else {

  alert("Thanks for staying with us");

}
```

JavaScript Objects

- JavaScript variables are containers for data values. Objects are variables too, but they can contain many values.
- Think of an object as a list of values that are written as name:value pairs, with the names and the values separated by colons.
- Example:

```
var person = {
 name: "Ashok", age: 24,
 favColor: "green", height: 183
};
```

- These values are called properties. JavaScript objects are containers for named values.

Object Properties

- You can access object properties in two ways.
- objectName.propertyName
  //or
  objectName['propertyName']
- This example demonstrates how to access the age of our person object. **Ref: objProperty.html**

```
var person = {
    name: "John", age: 31, favColor: "green", height: 183
};
var x = person.age;
var y = person['age'];
document.writeln(x);  Document.write(y);
```

- JavaScript's built-in **length** property is used to count the number of characters in a property or string.

  ```
  var course = {name: "Java", lessons: 41};

  document.write(course.name.length);    //Outputs 4
  ```

- Objects are one of the core concepts in JavaScript.

Object Methods

- An object method is a property that contains a function definition.
- Use the following syntax to access an object method.

```
objectName.methodName()
```

- As you already know, document.write() outputs data. The write() function is actually a method of the document object.

```
document.write("This is some text");
```

- Methods are functions that are stored as object properties.

Creating your own objects

- The standard way to create an "object type" is to use an *object constructor* function.

```
function person(name, age, color) {
  this.name = name;
  this.age = age;
  this.favColor = color;
}
```

- The **this** keyword refers to the current object.

- **Creating Objects**
- var p1 = new person("Hari", 42, "green");
- var p2 = new person("Vimal", 21, "red");
- document.write(p1.age); // Outputs 42
- document.write(p2.name); // Outputs "Amy"
- p1 and p2 are now objects of the person type. Their properties are assigned to the corresponding values.

Object Initialization

- Use the object literal or initializer syntax to create single objects.
- var John = {name: "John", age: 25};
- var James = {name: "James", age: 21};
- Objects consist of properties, which are used to describe an object. Values of object properties can either contain primitive data types or other objects.

Adding Methods

- Methods are functions that are stored as object properties.
- Use the following syntax to create an object method:

  methodName = function() { code lines }
- Access an object method using the following syntax:

  objectName.methodName()
- Defining methods is done inside the constructor function.

Ref: createObject.html

```
function person(name, age) {
  this.name = name;
  this.age = age;
  this.changeName = function (name) {
    this.name = name;
  }
}
var p = new person("David", 21);
p.changeName("John");
//Now p.name equals to "John"
```

- **Adding Methods (contd.)**
- You can also define the function outside of the constructor function and associate it with the object.

```
function person(name, age) {
  this.name= name;
  this.age = age;
  this.yearOfBirth = bornYear;
}
function bornYear() {
  return 2016 - this.age;
}
```

```
var p = new person("A", 22);

document.write(p.yearOfBirth());   // Outputs 1994
```

- Note: Call the method by the property name you specified in the constructor function, rather than the function name.
- **Ref: ObjMethod.html**

JavaScript Arrays

- Arrays store multiple values in a single variable.
- To store three course names, you need three variables.

```
var course1 ="HTML";

var course2 ="CSS";

var course3 ="JS";
```

- But what if you had 500 courses? The solution is an array.

```
var courses = new Array("HTML", "CSS", "JS");
```

- This syntax declares an array named courses, which stores three values, or elements.

- Accessing array elements

```
var course = courses[0]; // HTML

courses[1] = "C++"; //Changes the second element
```

- Attempting to access an index outside of the array, returns the value undefined.

```
var courses = new Array("HTML", "CSS", "JS");

document.write(courses[10]);

//Outputs "undefined"
```

- **Ref: array01.html**

- You can also declare an array, tell it the number of elements it will store, and add the elements later.

```
var courses = new Array(3);

courses[0] = "HTML";

courses[1] = "CSS";

courses[2] = "JS";
```

- An array is a special type of object.
- An array uses numbers to access its elements, and an object uses names to access its members.

- JavaScript arrays are dynamic, so you can declare an array and not pass any arguments with the Array() constructor. You can then add the elements dynamically.

```
var courses = new Array();

courses[0] = "HTML";

courses[1] = "CSS";

courses[2] = "JS";

courses[3] = "C++";
```

- You can add as many elements as you need to.

- **Array Literal**
- For greater simplicity, readability, and execution speed, you can also declare arrays using the array literal syntax.

```
var courses = ["HTML", "CSS", "JS"];
```

- This results in the same array as the one created with the `new Array()` syntax.
- You can access and modify the elements of the array using the index number, as you did before.
- The array literal syntax is the recommended way to declare arrays.

Length property

- JavaScript arrays have useful built-in properties and methods.
- An array's length property returns the number of it's elements.

```
var courses = ["HTML", "CSS", "JS"];

document.write(courses.length);  //Outputs 3
```

Combining Arrays

- JavaScript's concat() method allows you to join arrays and create an entirely new array.

```
var c1 = ["HTML", "CSS"];

var c2 = ["JS", "C++"];

var courses = c1.concat(c2);
```

- The courses array that results contains 4 elements (HTML, CSS, JS, C++).
- The concat operation does not affect the c1 and c2 arrays. It returns the resulting concatenation as a new array.

JavaScript does not support Associative Arrays
(i.e. arrays with named indexes)

- However, you still can use the named array syntax, which will produce an object.

```
var person = []; //empty array

person["name"] = "Hari";  person["age"] = 46;

document.write(person["age"]); //Outputs "46"
```

- Now, person is treated as an object, instead of being an array.
- The named indexes "name" and "age" become properties of the person object.
- As the person array is treated as an object, the standard array methods and properties will produce incorrect results. For example, person.length will return 0.

- Remember that JavaScript does not support arrays with named indexes.
- In JavaScript, arrays always use numbered indexes.
- It is better to use an object when you want the index to be a string (text).
- Use an array when you want the index to be a number.
- If you use a named index, JavaScript will redefine the array to a standard object.

- The Math object allows you to perform mathematical tasks, and include several properties:

E	Euler’s constant
LN2	Natural log of the value 2
LN10	Natural log of the value 10
LOG2E	The base 2 log of Euler’s constant
LOG10E	The base 10 log of Euler’s constant
PI	Returs the constant PI

```
document.write(Math.PI);

//Outputs 3.141592653589793
```

- Math has no constructor. There's no need to create a Math object first.
- The Math object contains a number of methods that are used for calculations:

```
abs(x), acos(x), asin(x), atan(x), ceil(x), cos(x),
sin(x), exp(x), floor(x), random(), round(x),
sqrt(x), ...
```

- ```
  var number = Math.sqrt(4);
  document.write(number);    // outputs 2
  ```
  ```

- To get a random number between 1-10:
  `Math.ceil(Math.random()*10)` **`//Ref:random.html`**
- Let us create a program that will ask the user to input a number and alert its square root:

```
var n = prompt(“Enter a number”,””);
var answer = Math.sqrt(n);
alert(“The square root of “ + n + “ is “ + answer);
```

- **Ref:sqrt.html**

setInterval()

- The **setInterval**() method calls a function or evaluates an expression at specified intervals (in milliseconds).
- It will continue calling the function until **clearInterval**() is called or the window is closed.

```
function myAlert() {
   alert("Hi");
}
setInterval(myAlert, 3000);
```

- This will call the myAlert function every 3 seconds (1000 ms = 1 second).
- Write the name of the function without parentheses when passing it into the setInterval method. **Ref: setInterval.html**

Date object

- The Date object enables us to work with dates.
- A date consists of a year, a month, a day, an hour, a minute, a second, and milliseconds.
- Using new Date(), create a new date object with the current date and time.

```
var d = new Date();

//d stores the current date and time
```

- The other ways to initialize dates allow for the creation of new date objects from the specified date and time.
- new Date(milliseconds)
- new Date(dateString)
- new Date(year, month, day, hours, minutes, seconds, milliseconds)
- JavaScript dates are calculated in milliseconds from 01 January, 1970 00:00:00 Universal Time (UTC). One day contains 86,400,000 millisecond.

- For example:
- //Fri Jan 02 1970 00:00:00

  var d1 = new Date(86400000);
- //Fri Jan 02 2015 10:42:00

  var d2 = new Date("January 2, 2015 10:42:00");
- //Sat Jun 11 1988 11:42:00

  var d3 = new Date(88,5,11,11,42,0,0);
- JavaScript counts months from 0 to 11. January is 0, and December is 11.
- Date objects are static, rather than dynamic. The computer time is ticking, but date objects don't change, once created.

- Date Methods

Method	Description
getFullYear()	gets the year
getMonth()	gets the month
getDate()	gets the day of the month
getDay()	gets the day of the week
getHours()	gets the hour
getMinutes()	gets the minutes
getSeconds()	gets the seconds
getMilliseconds()	gets the milliseconds

- Let’s create a program that prints the current time to the browser once every second. **Ref: time.html**

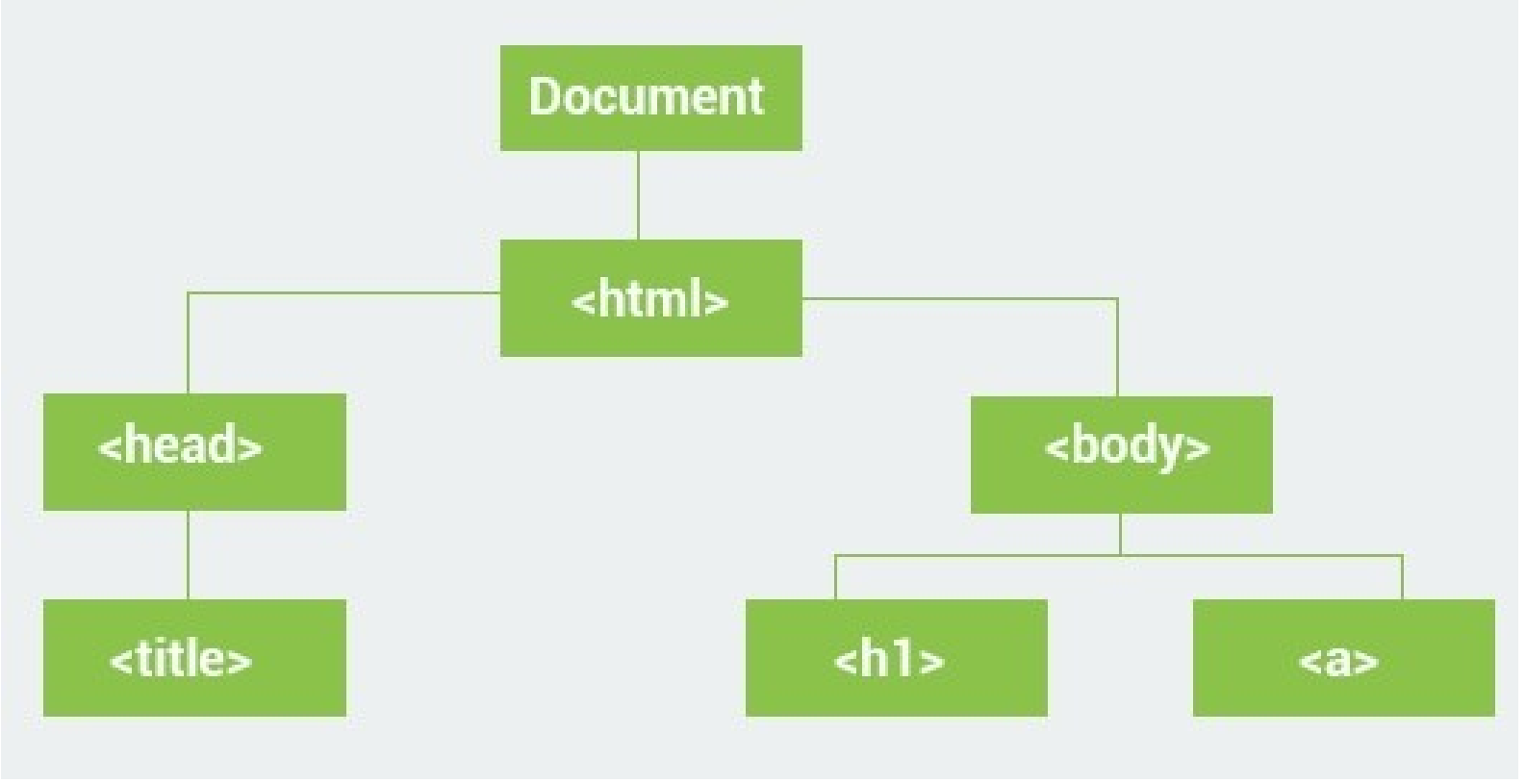
```
<p id=”demo></p>
<script>
   function printTime(){
      var d = new Date();
      var hours = d.getHours();
      var mins = d.getMinutes();
      var secs = d.getSeconds();
      document.getElementById("demo").innerHTML=
         hours+":"+mins+":"+secs;
   }
   setInterval(printTime,1000);
</script>
```

- The innerHTML property sets or returns the HTML content of an element.
- In our case, we are changing the HTML content of paragraph tag with id=”demo”. This overwrites the content every second, instead of printing it repeatedly to the screen.

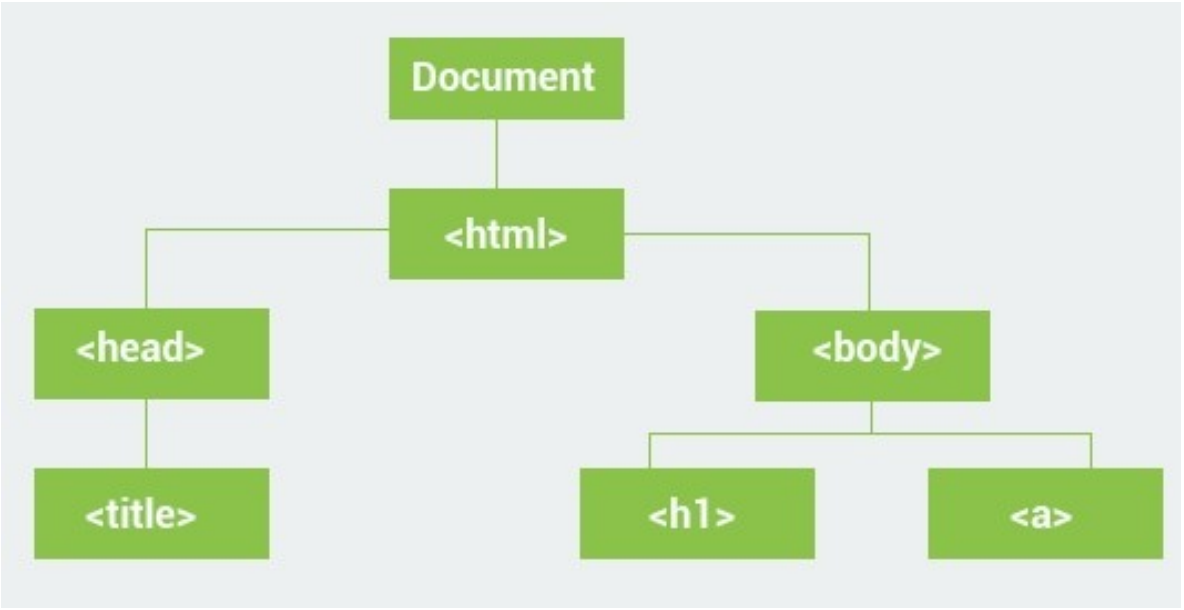
What is DOM

- When you open any webpage in a browser, the HTML of the page is loaded and rendered visually on the screen.
- To accomplish that, the browser builds the Document Object Model of that page, which is an object oriented model of its logical structure.
- The DOM of an HTML document can be represented as a nested set of boxes.
- JavaScript can be used to manipulate the DOM of a page dynamically to add, delete and modify elements.

DOM Tree



- The DOM represents a document as a tree structure.
- HTML elements become interrelated nodes in the tree.
- All those nodes in the tree have some kind of relations among each other.
- Nodes can have child nodes. Nodes on the same tree level are called siblings.



- <html> has two children (<head>, <body>);
- <head> has one child (<title>) and one parent (<html>);
- <title> has one parent (<head>) and no children;
- <body> has two children (<h1> and <a>) and one parent (<html>);
- It is important to understand the relationships between elements in an HTML document in order to be able to manipulate them with JavaScript.

The document Object

- There is a predefined **document** object in JavaScript, which can be used to access all elements on the DOM.
- In other words, the document object is the owner (or root) of all objects in your webpage.
- So, if you want to access objects in an HTML page, you always start with accessing the document object

- For example:

```
document.body.innerHTML = "Some text";
```

- As **body** is an element of the DOM, we can access it using the document object and change the content of the innerHTML property.
- The innerHTML property can be used on almost all HTML elements to change their content.

Selecting Elements

- All HTML elements are objects. And as we know every object has properties and methods.
- The document object has methods that allow you to select the desired HTML element.
- These three methods are the most commonly used for selecting HTML elements:
- //finds element by id
  document.getElementById(id)
- //finds elements by class name
  document.getElementsByClassName(name)
- //finds elements by tag name
  document.getElementsByTagName(name)

- In the example below, the getElementById method is used to select the element with id="demo" and change its content:
- var elem = document.getElementById("demo");
- elem.innerHTML = "Hello World!";
- The example above assumes that the HTML contains an element with id="demo", for example .

Selecting Elements

- The getElementsByClassName() method returns a collection of all elements in the document with the specified class name.
- For example, if our HTML page contained three elements with class="demo", the following code would return all those elements as an array:

```
var arr =  document.getElementsByClassName("demo");

//accessing the second element

arr[1].innerHTML = "Hi";
```

- **Ref: dom1.html**

- Similarly, the getElementsByTagName method returns all of the elements of the specified tag name as an array.
- The following example gets all paragraph elements of the page and changes their content:

```
<p>hi</p>
<p>hello</p>
<p>hi</p>
<script>
var arr = document.getElementsByTagName("p");
for (var x = 0; x < arr.length; x++) {
  arr[x].innerHTML = "Hi there";
}
</script>
```

Ref: dom2.html

- The script will result in the following HTML:

```
<p>Hi there</p>
<p>Hi there</p>
<p>Hi there</p>
```

- We used the length property of the array to loop through all the selected elements in the above example.

Working with DOM

- Each element in the DOM has a set of properties and methods that provide information about their relationships in the DOM:
- element.childNodes returns an array of an element's child nodes.
- element.firstChild returns the first child node of an element.
- element.lastChild returns the last child node of an element.
- element.hasChildNodes returns true if an element has any child nodes, otherwise false.

Working with DOM

- element.nextSibling returns the next node at the same tree level.
- element.previousSibling returns the previous node at the same tree level.
- element.parentNode returns the parent node of an element.
- We can, for example, select all child nodes of an element and change their content:

Ref:dom3.html

```
<html>
  <body>
    <div id ="demo">
      <p>some text</p>
      <p>some other text</p>
    </div>
    <script>
     var a = document.getElementById("demo");
     var arr = a.childNodes;
     for(var x=0;x<arr.length;x++) {
       arr[x].innerHTML = "new text";
     }
    </script>
  </body>
</html>
```

Changing Attributes

- Once you have selected the element(s) you want to work with, you can change their attributes.
- As we have seen in the previous lessons, we can change the text content of an element using the innerHTML property.
- Similarly, we can change the attributes of elements.
- For example, we can change the src attribute of an image:

```
<img id="myimg" src="orange.png" alt="">
<script>
   var el = document.getElementById("myimg");
   el.src = "apple.png";
</script>
```

Ref:dom4.html

- We can change the href attribute of a link:

```
<a href="http://www.example.com">Some link</a>
<script>
   var el = document.getElementsByTagName("a");
   el[0].href = "https://www.goole.co.in";
</script>
```

- Practically all attributes of an element can be changed using JavaScript.

Changing Style

- The style of HTML elements can also be changed using JavaScript.
- All style attributes can be accessed using the style object of the element.
- For example: **Ref: dom5.html**

```
<div id="demo" style="width:200px">some text</div>
<script>
   var x = document.getElementById("demo");
   x.style.color = "#6600FF";
   x.style.width = "100px";
</script>
```

- The code above changes the text color and width of the div element.
- All CSS properties can be set and modified using JavaScript. Just remember, that you cannot use dashes (-) in the property names: these are replaced with camelCase versions, where the compound words begin with a capital letter.
- For example: the background-color property should be referred to as backgroundColor.

Creating Elements

- Use the following methods to create new nodes:
- `element.cloneNode()` clones an element and returns the resulting node.
- `document.createElement(element)` creates a new element node.
- `document.createTextNode`(text) creates a new text node.
- For example:
- ```
  var node =
    document.createTextNode("Some new text");
  ```
  ```

- This will create a new text node, but it will not appear in the document until you append it to an existing element with one of the following methods:
- `element.appendChild(newNode)` adds a new child node to an element as the last child node.
- `element.insertBefore(node1, node2)` inserts node1 as a child before node2.

```
<div id ="demo">some content</div>
<script>
  //creating a new paragraph
  var p = document.createElement("p");
  var node = document.createTextNode("Some new text");
  //adding the text to the paragraph
  p.appendChild(node);

  var div = document.getElementById("demo");
  //adding the paragraph to the div
  div.appendChild(p);
</script>
```

Ref:dom6.html

This creates a new paragraph and adds it to the existing div element on the page.