**JavaScript**

**Introduction to JavaScript:**

* JavaScript is a dynamic computer programming language.
* It is light weight and most commonly used as a part of web pages, whose implementation allow client-side script to interact with the user and make dynamic pages.
* It is an interpreted programming language with object oriented capabilities.
* Client-side JavaScript is the most common form of the language. The script should be included in or referenced by a HTML document for the code to be interpreted by the browser

Limitation of JavaScript:

* Client side JavaScript does not allow the reading or writing of files. This has been kept for security purpose.
* JavaScript can’t be used for networking application because there is no such support available.
* JavaScript doesn’t have any multithreading or multipurpose capabilities.

Advantages of JavaScript:

* Less server interaction.
* Immediate feedback to the visitors.
* Increased interactivity.
* Richer interfaces.
* Lexical loading.

**Comments in JavaScript :**

* Single line comment (// is used to comment single line)
* Multi line comment (/\* \*/ is used to comment the multiple lines)

**Variables:**

* Variable is a container to store the data. “var” keyword is used to create the variable.
* JavaScript variables are identified with unique names. These unique names are called “Identifiers”.

var myName; //Defining a variable.

var myName = “pruthvi” ; //Defining and assigning a value to the variable.

Eg: var a = 1; //Integer variable

var b = ”orange”; //String variable

var c = true; //Boolean variable

**JavaScript Operators :**

JavaScript operators are symbols that are used to perform operations on operands.

Eg: var sum=10+20;   //Here, + is the arithmetic operator and = is the assignment operator.

**Types of operators used in JavaScript:**

1. **Arthematic operators:** +, - ,\* ,/ ,%
2. **Short hand operators: +=**, **-=, \*=, /=,** %**=,**
3. **Comparision operators:** ==, ===, !=, >, <, >=, <=
4. **Logical operators:** &&, ||, !
5. **Type operators:** typeof, instanceof
6. **Increment/Decrement operators:** ++, --
7. **Bitwise operators:** &, |, ~, ^, <<, >>, <<<
8. **Ternary operators:** ?

**Data types :**

JavaScript variables can hold many data types: numbers, string, objects and more

Eg: var length=16; // Number

var firstName=”pruthvi”; //String

var x={firstName : “pruthvi” , lastName : “TS”}; //object

**JavaScript Strings:** A string is a series of characters. Strings are written with quotes either single or double quotes.

Eg: var firstName=”pruthvi”; //using double quotes

var lastName=’TS’; //using single quotes

**JavaScript Numbers:** JS has onle one type of numbers. Numbers can be written with or without decimals. Internally in JS all numbers of 64 bit floating point numbers.

Eg: var x1=12.00; //written with decimals

var x2=12; //written without decimals

**JavaScript Booleans:** Booleans can only have 2 values: true or false

Eg: var x=5,v=5,z=6;

(x==y) //returns true

(x==z) //returns false

**JavaScript Arrays:** JS arrays are written with square brackets. Array items are separated by commas. Arrays are used to store the multiple values in a single variable. Array is a collection of data. Array indexes are zero-based.

Arrays can also be created using “new” keyword.

Eg: var fruits=[“mango”, “orange”, “apple”, “kiwi”];

**JavaScript Objects:** JavaScript objects are written with curly braces. Object properties are written as name : value pairs, separated by commas.

Eg: var person={ firstName : “pruthvi”, lastName : “TS”, eyeColor : ”brown” };

**Undefined:** In JavaScript, a variable without a value, has the value undefined. The typeof is also undefined.

Eg: var car; //value is undefined, type is undefined

**Empty Values:** An empty value has nothing to do with undefined. An empty string has both legal value and a type.

Eg: var car=” “; //value is “ “, the typeof is “string”

**Null:** In JavaScript null is “nothing”. It is supposed to be something that doesn’t exist.

In JavaScript, the data type of null is an object.

**JavaScript Control Statements :** Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this.

In JavaScript we have the following conditional statements:

* Use**if**to specify a block of code to be executed, if a specified condition is true
* Use **else** to specify a block of code to be executed, if the same condition is false
* Use **else if** to specify a new condition to test, if the first condition is false
* Use **switch** to specify many alternative blocks of code to be executed

**The if Statement :** Use the **if** statement to specify a block of JavaScript code to be executed if a condition is true.

Syntax:

if (condition) {  
block of code to be executed if the condition is true  
}

**The else Statement :** Use the **else** statement to specify a block of code to be executed if the condition is false.

Syntax:

if (condition) {  
    block of code to be executed if the condition is true  
} else {   
block of code to be executed if the condition is false}

**The else if Statement :** Use the **else if** statement to specify a new condition if the first condition is false.

Syntax:

if (condition1) {  
    block of code to be executed if condition1 is true} else if (condition2) {  
block of code to be executed if the condition1 is false and condition2 is true  
} else {  
block of code to be executed if the condition1 is false and condition2 is false  
}

**The JavaScript Switch Statement :**Use the switch statement to select one of many blocks of code to be executed.

Syntax:

switch(expression) {  
    case n:  
code block        break;  
    case n:  
        code block        break;  
    default:  
        code block  
}

This is how it works:

* 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.

**JavaScript Loops:** Loops are handy, if you want to run the same code over and over again, each time with a different value.

JavaScript supports different kinds of loops:

* **for**- loops through a block of code a number of times
* **for/in**- loops through the properties of an object
* **while**- loops through a block of code while a specified condition is true
* **do/while** - also loops through a block of code while a specified condition is true

**The For Loop :**

The for loop is often the tool you will use when you want to create a loop.

syntax: for (initialization; condition;increment/decrement) {  
    code block to be executed  
}

Example: for (i = 0; i < 5; i++) {  
    text += "The number is " + i + "<br>";  
}

**The For/In Loop**

The JavaScript for/in statement loops through the properties of an object:

Example: var person = {fname:"John", lname:"Doe", age:25};   
var text = "";  
var x;  
for (x in person) {  
    text += person[x];  
}

**The While Loop**

The while loop loops through a block of code as long as a specified condition is true.

Syntax: while (condition) {  
code block to be executed  
}

Example: while (i < 10) {  
    text += "The number is " + i;  
    i++;  
}

**The Do/While Loop :**

The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax: do {  
code block to be executed  
}  
while (condition);

Example: do {  
    text += "The number is " + i;  
    i++;  
}  
while (i < 10);

**Functions :**

Functions are reusable code in the program. It allow programmer to divide big programs into smaller pieces. JavaScript supports modular code functions. Function can be called when an event occurs.

* **BuiltIn Functions** **:**JavaScript has some builtin functions like alert(); document.queryselector(); document.write();

These function have global methods. All scripts and functions on a web page can access the global methods.

* **User Defined Functions :** These functions are defined by the user. If we create any function means we must call that function to invoke.

Example:

function calArea(w,h) {

var area=w\*h;

document.write(area);

}

calArea(5,5); //it will invoke the function calArea() then return 25 as the area.

**Math Object :**

It is an another variable which provides properties and methods for mathematical functions and constants. Unlike other objects math is not a constructor. All properties and methods of math are “static”. It is a global object so new keyword can’t be used on math.

**Math Object Methods:** Following are the math methods used in JavaScript.

|  |  |
| --- | --- |
| **Method** | **Description** |
| abs(x) | Returns the absolute value of x |
| acos(x) | Returns the arccosine of x, in radians |
| asin(x) | Returns the arcsine of x, in radians |
| atan(x) | Returns the arctangent of x as a numeric value between -PI/2 and PI/2 radians |
| atan2(y, x) | Returns the arctangent of the quotient of its arguments |
| ceil(x) | Returns the value of x rounded up to its nearest integer |
| cos(x) | Returns the cosine of x (x is in radians) |
| exp(x) | Returns the value of Ex |
| floor(x) | Returns the value of x rounded down to its nearest integer |
| log(x) | Returns the natural logarithm (base E) of x |
| max(x, y, z, ..., n) | Returns the number with the highest value |
| min(x, y, z, ..., n) | Returns the number with the lowest value |
| pow(x, y) | Returns the value of x to the power of y |
| random() | Returns a random number between 0 and 1 |
| round(x) | Returns the value of x rounded to its nearest integer |
| sin(x) | Returns the sine of x (x is in radians) |
| sqrt(x) | Returns the square root of x |
| tan(x) | Returns the tangent of an angle |

Example: var num=5.7;

var result=Math.round(num); //result=6;

var result=Math.ceil(num); //result=6;

var result=Math.floor(num); //result=5;

**JavaScript Date Object :**

Date object has an data type of int JavaScript object. Date object has sepearte methods for set and get.

**JavaScript get Date methods:**

|  |  |
| --- | --- |
| **Method** | **Description** |
| getFullYear() | Get the **year** as a four digit number (yyyy) |
| getMonth() | Get the **month** as a number (0-11) |
| getDate() | Get the **day** as a number (1-31) |
| getHours() | Get the **hour** (0-23) |
| getMinutes() | Get the **minute** (0-59) |
| getSeconds() | Get the **second** (0-59) |
| getMilliseconds() | Get the **millisecond** (0-999) |
| getTime() | Get the time (milliseconds since January 1, 1970) |
| getDay() | Get the weekday as a number (0-6) |
|  | |

**JavaScript set Date methods:**

|  |  |
| --- | --- |
| **Method** | **Description** |
| setDate() | Set the day as a number (1-31) |
| setFullYear() | Set the year (optionally month and day) |
| setHours() | Set the hour (0-23) |
| setMilliseconds() | Set the milliseconds (0-999) |
| setMinutes() | Set the minutes (0-59) |
| setMonth() | Set the month (0-11) |
| setSeconds() | Set the seconds (0-59) |
| setTime() | Set the time (milliseconds since January 1, 197 |
|  | |

Example:

var date = new date(); //creates a date

var getYear=date.getFullYear(); //returns the full year of the date

**Error handling in JavaScript :**

* **Use try/catch/finally to handle runtime errors in JavaScript:** These runtime errors are called Exceptions. An Exception can occur for a variety of reasons.

Eg: Referencing a variable or a method that is not defined can cause an exception.

* **The JavaScript statements that can possible cause exception should wrapped inside a try block:** When a specific line in the try block causes an exception, the control is immediately transfer to the catch block skipping the rest of the code in try block.

try{

document.write(sayHello());

document.write(“this line will not be executed”);

}

catch(e) {

document.write(“Description=” +e.description);

document.write(“Message=” +e.message);

document.write(“stack=” +e.stack);

}

document.write(“This line will be executed”);

In this example we are not defining any function by name sayHello() so it throw an exception. So it will display the error description, message and stack of that exception.

* **try block should be followed by a catch block or finally block:** Whatever we specified in the finally block that will be executed compulsorily. If an exception present in any line of code also finally block will be executed.

finally{

document.write(“This line is guaranteed to execute”);

}

* **Syntax errors and exceptions in JavaScript:** try/catch/finally block can catch exceptions but not syntax errors.

try{

alert(“Hello ; //syntax error

}

Catch(e)

{

document.write(“Hai”);

}

* **JavaScript throw statement:** use the throw statement to raise a customized exception.

**Accessing Forms in JavaScript :**

HTML form validation can be done by JavaScript.

Example:

If a form field (fname) is empty, this function alerts a message and returns false to prevent the form from being submitted.

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

**Data Validation :** Data validation is the process of ensuring that user input is clean, correct, and useful.

Typical validation tasks are:

* Has the user filled in all required fields?
* Has the user entered a valid date?
* Has the user entered text in a numeric field?

Most often, the purpose of data validation is to ensure correct user input.

Validation can be defined by many different methods, and deployed in many different ways.

**Server side validation** is performed by a web server, after input has been sent to the server.

**Client side validation** is performed by a web browser, before input is sent to a web server.

**DOM (Document Object Model) :**

* It is an application programming interface that has logical structure of well performed html, xml documents.
* When an HTML document is loaded into the web browser it becomes document object.
* Document object represents the html document that is displayed in the window.
* It has many properties which refers to other objects which allow access to and modification of the document content.
* The HTML DOM is a standard for how to get, change, add, or delete HTML elements.
* HTML DOM methods are **actions** you can perform.
* HTML DOM properties are **values** that you can set or change.

**The DOM Programming Interface:**

The HTML DOM can be accessed with JavaScript (and with other programming languages).

In the DOM, all HTML elements are defined as **objects**.

The programming interface is the properties and methods of each object.

A **property** is a value that you can get or set (like changing the content of an HTML element).

A **method** is an action you can do (like add or deleting an HTML element).

**The getElementById Method:**

The most common way to access an HTML element is to use the id of the element.

**The innerHTML Property:**

The easiest way to get the content of an element is by using the **innerHTML** property.

The innerHTML property is useful for getting or replacing the content of HTML elements.

**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:**

|  |  |
| --- | --- |
| **Method** | **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.setAttribute*(*attribute, value*)* | Change the attribute value of an HTML element |
| element.style.property *=* new style | Change the style of an HTML element |

**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(element) | 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 |

# DOM Elements:

Often, with JavaScript, you can manipulate the HTML elements.

To do so, you have to find the elements first. There are a couple of ways to do this:

* Finding HTML elements by id
* Finding HTML elements by tag name
* Finding HTML elements by class name
* Finding HTML elements by CSS selectors
* Finding HTML elements by HTML object collections

**Finding HTML Element by Id:**

The easiest way to find an HTML element in the DOM, is by using the element id.

This example finds the element with id="intro":

Example: var myElement = document.getElementById("intro");

If the element is found, the method will return the element as an object (in myElement).

If the element is not found, myElement will contain null.

**Finding HTML Elements by Tag Name:**

This example finds all <p> elements:

Example:

var x = document.getElementsByTagName("p");

**Finding HTML Elements by Class Name:**

If you want to find all HTML elements with the same class name, use getElementsByClassName().

This example returns a list of all elements with class="intro".

Example:

var x = document.getElementsByClassName("intro");

**Finding HTML Elements by CSS Selectors:**

If you want to find all HTML elements that matches a specified CSS selector (id, class names, types, attributes, values of attributes, etc), use the querySelectorAll() method.

This example returns a list of all <p> elements with class="intro".

Example:

var x = document.querySelectorAll("p.intro");

**Finding HTML Elements by HTML Object Collections:**

This example finds the form element with id="frm1", in the forms collection, and displays all element values.

Example: var x = document.forms["frm1"];  
var text = "";  
var i;  
for (i = 0; i < x.length; i++) {  
    text += x.elements[i].value + "<br>";  
}  
document.getElementById("demo").innerHTML = text;

**How to create and change Element:**

* Creation of element.

Example:

var createPTag=document.createElement(“P”);

creratePTag.innerHTML=”I just created a new P tag with JS”;

document.getElementById(“main container”).appendChild(createPTag);

Here we created new P tag and we placed that new P tag in main container.

* We change the text of the tag by using tag Id

Example:

var p1=document.getElementById(“p1”);

P1.innerHTML=”I changed the text by using javascript”;

Here we changed the content of p1with the help of javascript innerHTML.

* We can also create the text element.

Example:

var createTextElement=document.createTextNode(“This is another text node”);

document.getElementById(“main container”).appendChild(createTextElement);

Here we created new text element and placed it in the main container.

**Note:**

In the HTML DOM, Element object represents an HTML Element. Element object can have child nodes of type element nodes, text nodes or comment nodes.

We can find the HTML elements by Id, tag name, class name, css selectors, HTML object collections.

We can create the Element of any type either p, a or list etc. We can also change the element by using innerHTML.

**JavaScript Events:**

HTML events are things that happen to HTML Elements. When JavaScript is used in HTML pages, JavaScript can “react” on these events.

A HTML event can be something the browser does or something a user does.

JavaScript lets you execute code when events are detected.

Example:

An onclick attribute (with code), is added to a button element:

<button onclick=”document.getElementById(‘demo’).innerHTML=Date()”> The time is? </button>

 The code changes the content of its own element (using **this**.innerHTML):

<button onclick=”this.innerHTML=Date()”> The time is? </button>

**The common events used in JavaScript are:**

|  |  |
| --- | --- |
| **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 |

**Constraint validation using DOM methods:**

There are 2 methods which are used in constraint validation:

1. checkValidity() : Returns true if input element has valid data.
2. setCustomValidity() : Sets the validation message property of input element.

Constraints validation properties:

|  |  |
| --- | --- |
| **Property** | **Description** |
| validity | Contains boolean properties related to the validity of an input element. |
| validationMessage | Contains the message a browser will display when the validity is false. |
| willValidate | Indicates if an input element will be validated. |
|  |  |

Validity Properties:

|  |  |
| --- | --- |
| **Property** | **Description** |
| customError | Set to true, if a custom validity message is set. |
| patternMismatch | Set to true, if an element's value does not match its pattern attribute. |
| rangeOverflow | Set to true, if an element's value is greater than its max attribute. |
| rangeUnderflow | Set to true, if an element's value is less than its min attribute. |
| stepMismatch | Set to true, if an element's value is invalid per its step attribute. |
| tooLong | Set to true, if an element's value exceeds its maxLength attribute. |
| typeMismatch | Set to true, if an element's value is invalid per its type attribute. |
| valueMissing | Set to true, if an element (with a required attribute) has no value. |
| valid | Set to true, if an element's value is valid. |

**DOM Nodes :**

According to the W3C HTML DOM standard, everything in an HTML document is a node:

* The entire document is a document node
* Every HTML element is an element node
* The text inside HTML elements are text nodes
* Every HTML attribute is an attribute node (deprecated)
* All comments are comment nodes



With the HTML DOM, all nodes in the node tree can be accessed by JavaScript.

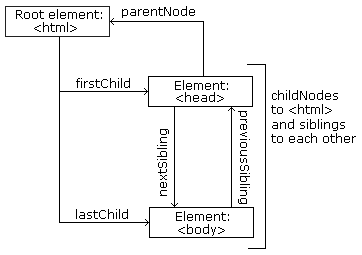
New nodes can be created, and all nodes can be modified or deleted.

**Node Relationships:**

The nodes in the node tree have a hierarchical relationship to each other.

The terms parent, child, and sibling are used to describe the relationships.

* In a node tree, the top node is called the root (or root node)
* Every node has exactly one parent, except the root (which has no parent)
* A node can have a number of children
* Siblings (brothers or sisters) are nodes with the same parent



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

**Navigating Between Nodes:**

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

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

**InnerHTML:**

We use the innerHTML property to retrieve the content of an HTML element.

However, learning the other methods above is useful for understanding the tree structure and the navigation of the DOM.

**DOM Root Nodes:**

There are two special properties that allow access to the full document:

* document.body - The body of the document
* document.documentElement - The full document

**The nodeName Property:**

The nodeName property specifies the name of a node.

* nodeName is read-only
* nodeName of an element node is the same as the tag name
* nodeName of an attribute node is the attribute name
* nodeName of a text node is always #text
* nodeName of the document node is always #document

**The nodeValue Property:**

The nodeValue property specifies the value of a node.

* nodeValue for element nodes is undefined
* nodeValue for text nodes is the text itself
* nodeValue for attribute nodes is the attribute value
* The most important nodeType properties are:

|  |  |  |
| --- | --- | --- |
| **Node** | **Type** | **Example** |
| ELEMENT\_NODE | 1 | <h1 class="heading">Neorays</h1> |
| ATTRIBUTE\_NODE | 2 | class = "heading" (deprecated) |
| TEXT\_NODE | 3 | Neorays |
| COMMENT\_NODE | 8 | <!-- This is a comment --> |
| DOCUMENT\_NODE | 9 | The HTML document itself (the parent of <html>) |
| DOCUMENT\_TYPE\_NODE | 10 | <!Doctype html> |

**The HTMLCollection Object:**

The getElementsByTagName() method returns an **HTMLCollection** object.

An HTMLCollection object is an array-like list (collection) of HTML elements.

The length property defines the number of elements in an HTMLCollection.

**The HTML DOM NodeList Object:**

A **NodeList** object is a list (collection) of nodes extracted from a document.

A NodeList object is almost the same as an HTMLCollection object.

Some (older) browsers return a NodeList object instead of an HTMLCollection for methods like **getElementsByClassName()**.

All browsers return a NodeList object for the property **childNodes**.

Most browsers return a NodeList object for the method **querySelectorAll()**.

The length property defines the number of nodes in a node list.

**The Difference Between an HTMLCollection and a NodeList:**

An HTMLCollection (previous chapter) is a collection of HTML elements.

A NodeList is a collection of document nodes.

A NodeList and an HTML collection is very much the same thing.

Both an HTMLCollection object and a NodeList object is an array-like list (collection) of objects.

Both have a length property defining the number of items in the list (collection).

Both provide an index (0, 1, 2, 3, 4, ...) to access each item like an array.

HTMLCollection items can be accessed by their name, id, or index number.

NodeList items can only be accessed by their index number.

Only the NodeList object can contain attribute nodes and text nodes.

**JavaScript Cookies:**

**Why cookies are needed?**

Web application works on HTTP protocol which is the stateless protocol. So after the web server has processed the client request for a web page, the web server will not remember anything about the client that made the request.

Cookies let you store user information in web pages.

**What are Cookies?**

Cookies are data, stored in small text files, on your computer.

When a web server has sent a web page to a browser, the connection is shut down, and the server forgets everything about the user.

Cookies were invented to solve the problem "how to remember information about the user":

* When a user visits a web page, his name can be stored in a cookie.
* Next time the user visits the page, the cookie "remembers" his name.

Cookies are saved in name-value pairs like:

username = Pruthvi

When a browser requests a web page from a server, cookies belonging to the page is added to the request. This way the server gets the necessary data to "remember" information about users.

**Create a Cookie with JavaScript:**

JavaScript can create, read, and delete cookies with the **document.cookie** property.

With JavaScript, a cookie can be created like this:

document.cookie = "username=Pruthvi";

You can also add an expiry date (in UTC time). By default, the cookie is deleted when the browser is closed:

document.cookie = "username=Pruthvi; expires=Thu, 18 Dec 2018 12:00:00 UTC";

With a path parameter, you can tell the browser what path the cookie belongs to. By default, the cookie belongs to the current page.

document.cookie = "username=Pruthvi; expires=Thu, 18 Dec 2018 12:00:00 UTC; path=/";

**Read a Cookie with JavaScript:**

With JavaScript, cookies can be read like this:

var x = document.cookie;

**Change a Cookie with JavaScript:**

With JavaScript, you can change a cookie the same way as you create it:

document.cookie = "username=chubby; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/";

The old cookie is overwritten.

**Delete a Cookie with JavaScript:**

Deleting a cookie is very simple.

You don't have to specify a cookie value when you delete a cookie.

Just set the expires parameter to a passed date:

document.cookie = "username=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/;";