**Javascript**

* Interpreted programming language, used to change the behaviour of a web page (execution at runtime)
* Very easy to implement
* Cross platform
* Javascript statements are enclosed within **<script></script>** tag
* <script></script> tags can be place anywhere in the webpage, but it is highly recommended that you place it at the end of the page.

**Statements**

* Instructions to be executed
* Statements are separated using semicolons ; and are optional
* Ignores white spaces
* document.write ("Hello Marlabs");

**Comments**

* // single line comment
* /\* multiple line comments \*/

**Variables**

* Javascript provides 3 primitive data types
  + Numbers
  + String
  + Boolean
* Variables are containers for storing values
* Declare a variable using **var** keyword
  + var username; // declared a variable with no value (undefined)
  + username = “Marlabs”; // assigned a value to the variable
  + var username = “Marlabs”; // single step
* Declare multiple variable using a single statement
  + var name = “Arun”, company = “Marlabs”, age=”30”;
* Variable names should always start with either a letter or an underscore character and should not start with a numeral
* Should never use javascript reserved keywords as variable names

**Operators**

* Javascript supports following operators
  + Arithmetic
  + Comparison
  + Logical
  + Assignment
  + Conditional
* **Arithmetic Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| + | Addition | var n = 4 + 5; |
| **-** | Subtraction | var n = 4 - 5; |
| **\*** | Multiplication | var n = 4 \* 5; |
| **/** | Division | var n = 4 / 2; |
| **%** | Modulus | var n = 4 % 2; |
| **++** | Increment | n++; n = n +1; |
| **--** | Decrement | n--; n = n-1; |

* **Assignment Operators**

|  |  |
| --- | --- |
| **Operator** | **Example** |
| **=** | a = b |
| **+=** | a += b; a = a + b; |
| **-=** | a -= b; a = a - b; |
| **\*=** | a \*= b; a = a \* b; |
| **/=** | a /= b; a = a / b; |
| **%=** | a %= b; a = a % b; |

* **Logical Operators**

|  |  |
| --- | --- |
| **Operator** | **Example** |
| **&&** | ((n=0) && (i > 0)) |
| **||** | ((n=0) || (i > 0)) |
| **!** | **!**((n=0) && (i > 0)) |

* **Comparison Operators**

|  |  |
| --- | --- |
| **Operator** | **Example** |
| **==** | Check for equal value |
| **===** | Check for equal value and type |
| **!=** | Not equal value |
| **!==** | Not equal value and type |
| **>** | Greater than |
| **<** | Less than |
| **>=** | Greater than or equal to |
| **<=** | Less than or equal to |
| **?** | Conditional operator |

**Functions**

* A block of javascript code designed to perform a specific task.
* Make your code reusable
* Javascript has number of builtin functions
* **Definition**
  + Use function keyword to define a function
  + Can have parameters
  + function myfirstfunction() {  
     document.write(“Hello Marlabs”);  
    }
* Calling function
  + myfirstfunction();
  + This will call the function defined above and will write “Hello Marlabs” to document body
* Parameters
  + function showdetails(name, age) {  
     document.write(“hello ”+name+”, your age is ”+age);  
    }
  + showdetails(“Arungopan”, 30);
* Javascript functions can also return values using **return** keyword
* Function declaration inside HTML elements
  + <p onclick="(function(){alert('helloo');})()">Click here.</p>

**Variable Scope**

* **Local scope**
  + Variables declared inside a function becomes local to that function and can be only be accessed inside that function
  + function userdetails() {  
     var name = “Arungopan”; // local variable  
    }
  + Function parameters will also act as local varibales
  + deleted when the function call gets completed
* Global scope
  + var name = “Arungopan”; // global variable  
    function userdetails() {  
     document.write(name);  
    }
  + deleted when you close the page
* Closure
  + A closure is an inner function that has access to the outer function variables, scope and function parameters.
  + Closure makes it possible to have private variables.
  + Three main scopes:
    - access to its own local scope
    - access to the outer function’s variables, parameters
    - access to the global variables
  + function show\_name(firstname, lastname) {  
     var name\_str1 = "Displaying name without prefix firstname = "+firstname+" lastname = "+lastname;  
     document.getElementById('demo').innerHTML = name\_str1;  
     function show\_fullname(prefix\_txt) {  
     return prefix\_txt + firstname + " "+lastname;  
     }  
      
     return show\_fullname;   
    }  
      
    fullname\_fn = show\_name("arun", "gopan");  
    document.getElementById('demo2').innerHTML = fullname\_fn("My fullname is :");  
      
    </script>
* **Anonymous functions**
  + Functions which doesn't have a name.
  + Created using function operator
  + Can be passed as parameter to another function
  + var abc = function() {  
     alert('helloo');  
    }  
    abc();
  + Passing anonymous function as parameter
  + var abc = function(text) {  
     text = (text) ? text : 'Helloo';  
     alert(text);  
    };  
      
    function sayHi(func) {  
     func();  
    }  
      
    sayHi(abc);
* **Self invoking function**
  + Gets called automatically, without being called.
  + (function() {  
     alert(“helloo”);  
    })();
* **Closure with self invoking functions**
  + var inc\_counter = (function() {  
     var inc\_cntr = 0;  
     return function() { return inc\_cntr += 1;}  
    })();  
    inc\_counter();
  + In the example you can see that inner function can access outer function variable.
  + A closure is a special kind of object that combines two things: a function, and the environment in which that function was created. The environment consists of any local variables that were in-scope at the time that the closure was created. In this case, myFunc is a closure that incorporates both the

**JavaScript Hoisting**

* Hoisting is JavaScript's default behavior of moving declarations to the top
* All variable and function declarations are hoisted to the top of their scope
* JavaScript Initializations are Not Hoisted

**Events**

* Javascript supports events listed below
  + onchange // element value changes
  + onclick // element is clicked
  + onmouseover // when you move mouse over an element
  + onmouseout // when you move mouse away from an element
  + onload /// when browser finish loading page
  + onkeydown
  + onkeypress
  + onkeyup
  + onblur
  + onchange
* <button onclick="showname(‘Arungopan’)">Show name</button>  
  function showname(name) {  
   alert(name);  
  }

**Conditional Statements**

* **If-else if**
* **Switch**

**Loops**

* For loop
  + for (i = 0; i < 10; i++) { alert(i); }
* While loop
  + while (i < 10) {  
     alert(i);  
     i++;  
    }
* **Break** and **continue** statements
  + Break : used to exit from a loop
  + Continue : used to continue to the next iteration

**Objects**

* As we learned already that variables are used to hold values, just a single value.
* Objects are also variables which can hold multiple values.
* The code below creates a variable user and assigns a value to it.
  + var user = “Arungopan”;
* Imagine in the example above if i want to store more information about the user.
  + var user = {“name”:”Arun", “age”:30, “phone”:”9446614757”};
  + Objects store data as key value pair.
* So objects are containers for named value.

**Object Definition**

* The most simplest way to create is show below.
  + var user = {“name”:”Arun", “age”:30, “phone”:”9446614757”};
* Using **new keyword**
  + var user = new Object(); // creates an empty object
  + User.name = “Arungopan”; //assigns value
* Using **object constructor**
  + function user(name, age, phone) {  
     this.name = name;  
     this.age = age;  
     this.phone = phone;  
    }  
    var user1 = new user(“Arun”, 30, “9446614757”);
  + this refers to an new empty object, which is created using the new keyword
  + If you try to call the user() without the new keyword. It returns undefined, because the user() is not returning anything, previously it was the new keywords which was returning you the object.

**Objects are addressed by reference not by value**

* var user = {“name”: “Arun”, “age”:30};
* var user1 = user;
* In the above example **user1 is not a copy of user**, it is the user itself.
* Any changes to user1 will reflect user.

**Object Properties**

* var name = user.name;
* var name = user[“name”];
* Using for loop
  + for (key in user) {  
     alert(key + “ ”+ user[key]);  
    }
* Delete properties from an object
  + delete user.name; // remove name property from user object

**Object Methods**

* Object properties can hold values, other objects and functions
* Object method is a property containing function definition.
* var user = {  
   “firstname” : “Arun”,  
   “lastname” : “gopan",  
   “fullname” : function () { return this.firstname + “ “+ this.lastname}  
  }  
  user.fullname();

**Object Prototypes**

* Every JavaScript function has a prototype property
* Primarily for inheritance
* Methods and properties added to function’s prototype property will be available to all instances of that function
* If you create a object using new keyword like show below
  + var user = new Object();
  + Inherit from the Object.prototype.
* Standard way of creating an object prototype is by using object **constructor function**   
  function user(firstname, lastname, age, phone) {  
   this.firstname = firstname;  
   this.lastname = lastname;  
   this.age = age;  
   this.phone = phone;  
  }  
  var user1 = new user(“Arun”, “gopan”, 30, “9446614757”);
* Adding new properties to existing prototype
  + user.prototype.company = “Marlabs”;
  + user.prototype.fullname = function() { return this.firstname + “ “+this.lastname};

**Math Object**

* Makes mathematical task easier
* Math.min(), Math.max()
  + Used to find min and max values from a set of values
  + Math.min(0, 50, 100, 150, -5);
* Math.round()
  + Rounds to the nearest integer value
  + Math.round(4.5);
* Math.ceil()
  + Rounds up to the nearest integer value
* Math.floor()
  + Rounds down to the nearest integer value

**Arrays**

* Unlike normal variables which stores single value, arrays are used to stores multiple values in a single variable.
* Creating an array
  + var users = [“Arun”, “Nanda”, “Naveen”];
  + var users = new Array(“Arun”, “Nanda”, “Naveen”);
* Accessing values in array
  + By index number
    - var name = user[0];
* Array index starts from zero
* Arrays are also special type of objects
  + Arrays uses **numbered indexes** and object uses **named indexes**
* Length of an array
  + users.length will return length of user array
* Adding new array element
  + user.push(“Bipin”);
  + User[user.length] = “Bipin”;
* Removing an element
  + user.shift(); // removes the first element from an array
  + user.pop(); // removes the last element from an array
* Deleting an array index
  + delete user[1];
  + user.splice(2, 1); // index, how many elements to delete
  + user.splice(user.indexOf(“Arun”));

**Date Object**

* Created using new Date();
  + document.write(Date());
* **Methods**
  + var date\_str = new Date();
  + **getTime()**
    - date\_str.getTime(); // return time in milliseconds
  + **getFullYear()**
    - date\_str.getFullYear(); // return current year
  + **getDay()**
    - date\_str.getDay(); // return current day
  + **Methods to set dates**
    - setFullYear()
      * var date\_ste = new Date();  
        d.setFullYear(2018);
    - setDate()
      * var date\_ste = new Date();  
        d.setDate(10);
  + **Adding 2 days to the current day**
    - date\_str.setDate(date\_str.getDate() + 2);
    - document.write(date\_str);
  + **Adding 2 years to the current year**
    - date\_str.setFullYear(date\_str.getFullYear() + 2);
    - document.write(date\_str);

**Javascript Strings**

* var username = “arungopan”;
* Can use single / double quotes to enclose the string.
* **String length**
  + var username\_len = username.length; // return string length
* **Escape quotes**
  + Javascript allows to use single / double quotes as part of the string.
  + var username = “arungopan”gopakumar”; // invalid
  + In Order to store the above value properly we need to escape the string.
    - var username = “arungopan\”gopakumar”;
* **indexOf**
  + Will return position of a substring in a string
  + var test\_str = “Hi my name is Arun”;
  + var pos = test\_str.indexOf(“Arun”);
* **Match**
  + Used to find an occurrence of a substring in a string.
  + test\_str.match(“Arun”)
* Both **indexOf** and **match** functions are case sensitive
* **toLowerCase**
  + Converts a string to lowercase
  + test\_str.toLowerCase();
* **toUpperCase**
  + Converts a string to uppercase
  + test\_str.toUpperCase();
* **replace**
  + Find and replace a substring in a string
  + str.replace(‘username’, ‘Arun’);
  + You can also pass regular expression inside replace function
  + str.replace(new RegExp(',', 'g'), '');
  + Replace callback function  
    - var str = "user1 is from india and user1 works for company1";  
        
      var cntr = 0;  
      str = str.replace(/user1/g, function(match) {  
       cntr++;  
       if(cntr==2) {  
       return 'user12345'  
       } else {  
       return 'user1';  
       }  
      });  
        
      console.log(str);
* **substring**
  + Returns portion of a string
  + str.substring(0,5);

**HTML DOM (Document Object Model)**

* HTML DOM defines a standard for accessing html document.
* Defines elements as objects
* Methods to access elements
* Events for all elements
* Can set / get **property** to all elements using **methods**
* **Finding elements**
  + **document.getElementById()**
  + <div id=”my\_div” class=”div\_cls”>helloo</div>  
    var div\_elem = document.getElementById(‘my\_div’);
  + **document.getElementsByTagName()**
  + var div\_elem = document.getElementsByTagName(‘div’);
  + **getElementsByClassName()**
  + var div\_elem = document.getElementsByClassName(‘div\_cls’);
* **Changing Elements**
  + **Content**
  + document.getElementById(‘my\_div’).innerHTML = “Hello Marlabs”;
  + **Attributes**
  + document.getElementById(‘my\_div’).className = “new\_div\_cls”;
  + **Style**
  + document.getElementById("‘my\_div’").style.color = "blue";
  + document.getElementById("‘my\_div’").style.background = "red";
  + document.getElementById("‘my\_div’").style.fontSize = "large";
* **Events**
  + **Onclick**
  + <div onclick=”test\_fn(this)” id=”click\_event”>hello</div>  
    <script>  
     test\_fn(this\_obj) {  
     this\_obj.innerHTML = “Hello Marlabs”;  
     }  
    </script>
  + The above example can also be written as
  + document.getElementById(‘click\_event’).onclick = test\_fn;
  + Here we won't get the this object of the element, which we have create
  + We can use all the events which discussed earlier via HTML DOM
* **EventListeners**
  + Javascript allows you add / remove event listeners to elements
  + **addEventListener()**
  + document.getElementById("‘my\_div’").addEventListener(“click”, test\_fn)
  + Above example attach an event listeners to fire when a user clicks on the element
  + You can also write the function definition along it.
  + document.getElementById("‘my\_div’").addEventListener(“click”, function () { alert(“hello Marlabs”); });
  + **removeEventListener()**
  + document.getElementById("‘my\_div’").removeEventListener(“click”, test\_fn)
* **Page load events**
  + Checks whether the DOM elements has been loaded.  
    window.addEventListener("DOMContentLoaded", function () {  
     document.getElementById('abc').innerHTML = 'Helo Marlabs!!!';  
     });
  + Checks all of the HTML is loaded, and any subresources like images are loaded  
     window.onload = function() {  
     document.getElementById('abc').innerHTML = 'Helo Marlabs!!!';  
     };

**addEventListener, handleEvent and passing objects**

* addEventListener can take an object as a second argument that will look for a method called handleEvent and call it  
  + var logger = {  
     x: 0,  
     handleEvent: function(){  
     this.x++;  
     console.log(this.x);  
     }  
    };  
      
    document.getElementById('button').addEventListener('click', logger);

**Javascript Popup Boxes**

* **Alert**
  + Used to show a response message or an alert message.
  + alert("Hello Marlabs");
* **Confirm Box**
  + Used when you need to show a confirm message to user.
  + var res = confirm("Are you sure you want to continue");  
    if(res==true) {  
     alert(“Okay”);  
    } else {  
     alert(“Cancel”);  
    }
* **Prompt Box**
  + Used when you want the user to enter a value before proceeding
  + var res = prompt(“Please enter your username”);  
    if(res!=””) {  
     alert(“Okay”);  
    }

**Javascript Window location**

* **Redirect** 
  + window.location="<http://www.marlabs.com>";
* Window.location.href will return the current url

**Typeof**

* Used to find variable type
  + typeof "Arungopan"

**isArray()**

* Array.isArray(num), to check whether the variable is an array.

**setTimeout()**

The setTimeout() method calls a function or evaluates an expression after a specified number of milliseconds.

setTimeout(function(){ alert("Hello"); }, 3000);

**clearTimeout()**

The clearTimeout() method clears a timer set with the setTimeout() method.

clearTimeout(function\_name);

timeout\_var = setTimeout(function(){ alert("Hello"); }, 3000);

window.clearTimeout(timeout\_var);

**setInterval()**

The setInterval() method calls a function or evaluates an expression at specified intervals (in milliseconds).

setInterval(function(){ alert("Hello"); }, 3000);

**clearInterval()**

The clearInterval() method clears a timer set with the setInterval() method.

Store setInterval function in a variable and then pass it to clearInterval()

interval\_var = setInterval(function() { }, 1000);

window.clearInterval(interval\_var);

clearInterval(function\_name);

**getAttribute()**

The getAttribute() method returns the value of the attribute with the specified name, of an element.

document.getElementById(‘test’).getAttribute("class");

**setAttribute()**

The setAttribute() method adds the specified attribute to an element, and gives it the specified value.

document.getElementById(‘test’).setAttribute("class", “test\_cls”);

**createElement()**

var para\_elem = document.createElement("p");

**createTextNode**

var para\_elem = document.createElement("p");

var textNode = document.createTextNode("Hello Marlabs");

para\_elem.appendChild(node);

**appendChild**

var element = document.getElementById("div1");

element.appendChild(para);

**Style**

document.getElementById("myH1").style.color = "red";

**querySelector()**

The querySelector() method returns the first element that matches a specified CSS selector(s) in the document.

document.querySelector(".example").style.backgroundColor = "red";

document.querySelector("div#jjj > p");

document.querySelector("p");

document.querySelector("p.test\_cls");

document.querySelector("#div\_id").innerHTML = "Hello Marlabs";

document.querySelector("a[age='30']").style.border = "10px solid red";

**querySelectorAll()**

The querySelectorAll() method returns all elements in the document that matches a specified selector

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

for(i=0; i<x.length; i++) {

x[i].style.backgroundColor = "red";

}

**insertAdjacentHTML(position, text)**

position is the position relative to the element, and must be one of the following strings:

'beforebegin' : Before the element itself.

'afterbegin' : Just inside the element, before its first child.

'beforeend' : Just inside the element, after its last child.

'afterend' : After the element itself.

**JSON**

JSON is a format for storing and transporting data.

JSON is often used when data is sent from a server to a web page.

JSON stands for JavaScript Object Notation

var json\_obj = {

"user\_info": {

"name":"Arungopan",

"email":"arungopan@marlabs.com",

"location":"India"

},

"prof\_info": {

"name":"Marlabs",

"join\_date":"March 2016"

},

"interest": {

"movie":"Intern",

"game":"Cricket",

"bikes":"Bullet"

}

};

JSON.stringify(); // converts objects in JSON strings

JSON.parse(); // converts to object

**Mixins**

Mixins are ways in which we can reuse function.

Refer example below.

function user(name) {

this.name = name;

}

function employee(name) {

this.name = name;

}

var interest = {

addInterest : function(interest) {

return this.name+"'s interest include "+interest;

}

};

var org = {

showOrg : function(org) {

return this.name+" works for "+org;

}

};

function extend(protoObj, mixin) {

for(key in mixin) {

protoObj[key] = mixin[key];

}

return protoObj;

}

//extend(employee.prototype, interest);

**Mixins using jQuery**

jQuery.extend(employee.prototype, interest, org);

var emp1 = new employee('Arun');

document.write(emp1.addInterest(['Movies', 'Bikes']))+"<br />";

document.write(emp1.showOrg("Marlabs"))+"<br />";

**Mixins using class expressions**

let mixin1 = (superclass) => class extends superclass {  
 fn() {  
 document.write('Fn1() called!!! <br />');  
 super.fn();  
 }  
}  
  
let mixin2 = (superclass) => class extends superclass {  
 fn() {  
 document.write('Fn2() called!!! <br />');  
 super.fn();  
 }  
}  
  
class S {  
 fn() {  
 document.write('superFn() called!!! <br />');  
 super.fn();  
 }  
}  
  
class C extends mixin1(mixin2(S)) {  
 fn() {  
 document.write('Fn() called!!! <br />');  
 super.fn();  
 }  
}  
  
new C().fn();

**Debounce in JavaScript**

* The goal behind each implementation is to reduce overhead by preventing a function from being called several times in succession
* <http://underscorejs.org/#debounce>
* function debounce(func, wait, immediate) {  
   var timeout;  
   return function () {  
   var context = this, args = arguments;  
   var later = function () {  
   timeout = null;  
   if (!immediate) func.apply(context, args);  
   };  
   var callNow = immediate && !timeout;  
   clearTimeout(timeout);  
   timeout = setTimeout(later, wait);  
   if (callNow) func.apply(context, args);  
   };  
   };  
   function calculateLayout() {  
   console.log('Arun here!!!');  
   }  
   var lazyLayout = debounce(calculateLayout, 300, true);  
   jQuery('#divElem').scroll(lazyLayout);
* Instead of using lazyLayout if calculateLayout is used directly, the number of times the function gets called will be too high, which is a serious performance degrade for your application.

**Functional programming in JavaScript**

<https://medium.com/javascript-scene/master-the-javascript-interview-what-is-functional-programming-7f218c68b3a0>