1. Alert

# Syntax:

alert(“Hello world!”);

**OR**

window.alert(“Hello world!”);

Alternatively you can also use variables instead of messages or combine both string & variables using a + sign.

* Remember:

You can choose to use single quotes or double quotes at your own discretion but usually go with double quotes until dealing with HTML code which also include “double quotes” with itself, because in that case you go with ‘single quotes’.

1. Variables for Strings

JavaScript does not concern itself with the nature of the data stored in the string hence you do not need to specify the type of data (string, numeric, boolean).

It has 2 parts

# Declaration:

var myVariable;

# Initiation:

myVariable = “Hassan Imran”;

You can also accomplish this task in a single statement

var myVariable = “Hassan Imran”;

Once you have declared the variable you do not need to declare it again you can just refer it and change its value.

1. Variable for Numbers

var myNum = 30;

now you can add subtract values from it easily.

var yourNum = myNum + 20;

**Note:**

Any value enclosed in quotation marks will be considered as string.

1. Legal & Illegal Variable Names

* Cannot contain spaces
* Can only contain letters, numbers, dollar and underscore sign (provided that it cannot start with a number)
* It is case sensitive (So is the whole javaScript!)

1. Math Operators

* +
* -
* \*
* /
* % (Modulus Operator: Gives us the remainder obtained after the division;)

1. More Math Operators

* ++ (It is the increment operator if added before a variable (++a) it increases the value of the variable by 1 and then executes the statement. If added afterwards (a++) it first executes the statement and then increases the variable’s value by 1)
* -- (Decrement operator, works same as increment operator, No surprises there)

1. Math : The Order

* JavaScript Follows the BODMAS rules
* Do everyone a favor and always use parenthesis when even a slight chance of error is there.

1. Concatenating Text Strings

Simply use the + sign to join variables (strings or numeric) with actual string texts.

# For Example:

alert(“Thank you “ + userName + “, for your feedback!”);

**OR**

alert(message + userName);

* Remember if a number is in a string format (enclosed in “”) or for reason the JavaScript believes it to be, then

alert(“2” + “2”);

will tend to give you 22 instead of 4. (JavaScript takes the one plus one equals 11 metaphor VERY seriously).

1. Prompts

Prompt is similar to the alert pop-up-box with one exception, it takes an answer back.

var spec = prompt("Your species?", "human");

Now this statement does 2 things:

1. It requires a variable to store the feedback from the prompt
2. It autofills the prompt box with the answer human itself.

Obviously we can also use variables to separately store the question and place-holder answer for the prompt if we want to.

If the user gives us a value then well and good (The value is stored as string).

Although if the user presses **OK** without any value our variable stores it as a empty string **“”**.

If the user presses **CANCEL** the value stored in the variable would be **null.**

window.prompt(“”);

is also a legal command. Basically the word window here kind of represents a library as in C language. Different Libraries know how to execute different commands.

* In order to change lines or give spaces remember the C Language syntax “\n” or “\t” is required for line change or tab.

1. ***if*** Statements

if (some condition) {

lines of code to execute

}

else {

different lines of code to execute

}

* Remember JavaScript has two comparison operators ( == ) and

( === ) or ( != ) and ( !== ). The shorter ones ( == ) and ( != ) only compare the value and not the data type but the triple ones ( ===) and ( !== ) compare the data and data type too.

For Example:

12345 == “12345” //will return true

12345 === “12345” //will return false

* Once again it is obvious that for checking conditions you might want to put to sample and the comparison data in different variables and simply just compare variables.

1. Comparison Operators

* == (equals to – does not compare data type)
* === (equals to – compare data type)
* != (not equals to – does not compare data type)
* !== (not equals to – compares data type)

The (!) exclamation mark is referred to as the NOT operator

* a > b (a is greater than b)
* a < b (a is less than b)
* a <= b (a is less than or equal to b)
* a >= b (a is greater than or equal to b)

1. ***if***… ***else*** and ***else if*** Statements

If you want to test for more than one condition with ***if*** statement then you use else if statement.

if (1 condition) {

lines of code to execute

}

else if (2 condition){

different lines of code to execute

}

else {

different lines of code to execute

}

1. Testing Set of Conditions

# Logical AND

&&

# Logical OR

||

* When using more than one condition in a single if statement be sure to use parenthesis to avoid unnecessary errors.

1. ***if*** Statements Nested

Basically if you just use an ***if*** statement within another ***if*** or ***else if*** it makes it a nested if statement.

1. Arrays

Many Places you have to use the single quotes after the innerHTML etc tags because when you go on writing the HTML code & you have to introduce an attribute then using double quote closes the one opened in the start making the attibute value JavaScript in nature. Hence the single quotes

* Array numbering starts ALWAYS from zero
* Try to use plural names to define arrays, so that it is easy to identify later on
* A single Array may contain different types of data

var countries = [“Pakistan”, “Australia”, “USA”, 23 ]

Now when you refer to countries[1] in your code it gives the value Australia.

Notice here that you may choose to declare an empty array and later on declare its value or initiate as we should say.

var names[];

names[1] = “You”;

names[2] = “Me”;

* If an array element is empty it will return “Undefined” as the value
* **Note:**

If we declare a text-string without any quotes at all JS recognizes it as an Identifier or a keyword. This property actually helps us a lot.

For Example:

Instead of writing

document.getElementById(“anything”).innerHTML

Again and again we can simply store the document.getElementById in a variable and refer to that variable again and again.

var x = document.getElementById;

1. Adding/Removing elements from arrays

Once an array has been declared, we can add its element anywhere in the whole code by simply referring to its name and position.

names[1] = “You”;

names[2] = “Me”;

Even if we leave out empty elements for example names[0] is empty in the above array, we can declare it later on.

names[0] = “Reader”;

Although if you refer to an empty element it gives you undefined.

* Using arrayName.pop(); we can remove the last element of any array.
* Assuming that a limited array has been defined and you want to add more element to the end of it, so you can use

arrayName.push(“his name”, “her name”);

1. Array: Removing, inserting & extracting elements

* **Shift:**

Use arrayName.shift(); to remove the first element of the array

* **Unshift:**

Use arrayName.unshift(“this”, “that”); to add these 2 elements at the start of the array.

* **Splice:**

We use splice to add and/or remove elements from any position in an array.

arrayName.splice(2, 2, “apple”, “orange”, “banana”);

The 1st number indicates the position/number at which you start adding or removing.

The 2nd number indicates how many element should be removed from the number and onwards.

And the rest are the elements.

In the above example we

1. Move in at number 2 of the array
2. Delete 2 elements 2 and 3
3. And then add the element apple, orange and banana at 2,3 and 4 positions respectively.

To add without removing:

arrayName.splice(2, **0**, “apple”, “orange”, “banana”);

To remove without adding:

arrayName.splice(2, 2);

* **Slice:**

To copy consecutive elements from one array to another we use slice.

var anyThing = array1.slice(1,5);

The 1st number indicates the element from which to start copying and the 2nd number denotes the element ***after*** the last element to be copied. In effect this copies 1,2,3,4.

You can also use this technique to reduce the array to only the elements required.

Extra Array functions

1. arrayName.toString(); Converts array to comma separated string
2. arrayName.Join(“\*”); Will join array elements with \* in b\w
3. array1.concat(array2,array3) Joins array 1 with 2 & 3 and returns a new one, Can also pass an actual array [] instead of name in it.
4. Array1.sort() arranges in alphabetical order
5. Array1.reverse() Reverses it. Can be used for descending order

But according to sort 25 is > 100 why b\c 2 > 1. So we use

1. Array1.sort( function(a,b) {return a-b} ) this means that sorting will be done on the basis of number differences.. (High to low b-a) (Low to high a-b)
2. Array1.sort( function(a,b) {return 0.5 – Math.random() } )

Here it should be noted that the parameters of the prototype sort() is basically a compareFunction that should be introduced by the programmer.

If no compareFunction is introduced then the arrays are sorted on the basis of their Unicode value.

It returns the sorted array.

1. ***for*** loops

**Syntax:**

for (var i = 0;i<=10;i++){

Some code;

}

Now here there are 3 parts in the ***for*** statement:

1. **Initiation:**

We declared a counter variable i and set its value to zero.

1. **Limitation:**

We set the limit upto which the loop will repeat.

1. **Increment/decrement:**

We set how much the value of the loop counter i should change after each run.

1. ***for***loops: Flags, Booleans, array length, and loopus interruptus

* **Flags:**

Now suppose if a loop runs and we achieve the task we were hoping to, everything’s fine. But what happens if we fail to achieve our goal through the for loop?

Here we use flags, for example a Boolean variable having a default value of false. But if the goal is achieved then it changes to true.

var taskComplete = false;

here we do not use quotes because true and false are actually keywords and not text strings.

* **Break:**

Now assuming that we achieved our task on the second loop, now our for loop still has to run 8 more useless loops even though we have achieved our goal.

To stop this we use break. It breaks the loop and jumps out of it. Usually we place it within a conditional statement to make sure the goal is achieved.

* **Length of loop:**

We can determine the length of an unrestricted array by adding the keyword .length at the end of it.

var numElements = names.length;

Note that the length of an array will be counted from 1.

1. ***for*** loops nested

We can use a for loop within another for loop.

The note-worthy thing in this concept is that if loop2 is inside loop1, then each time loop1 runs, loop2 runs completely upto its termination and so on.

1. Changing Case

Now assume you are compare a person’s quiz answer to the actual answer.

The actual answer is ‘united states’, but this person writes ‘United States’.

JavaScript falsifies this answer.

Now either you include every possible typecase scenario of the answer into the answer array,

Or

You convert the user’s answer to lowercase.

Obviously the latter is easy.

answerToCheck = answerToCheck.toLowerCase();

or toUpperCase();

This prototype is only for strings

**Note:**

We can use string.split(‘ ’) for splitting a string into an array with elements being divided on the basis of a space ‘ ’.

We can array.toString(‘ ’) it back to life :P

1. Strings: Measuring length & Extracting Parts

Now suppose you want a word to be an initial cap as in (Jack) instead of jack or JACK.

This cannot be done by the toUpperCase or toLowerCase techniques.

For such a situation we divide our string into 2 parts:

1. The initial letter which has to be capitalized.
2. The whole string other than that.

To copy the letter we use the slice method, in the exact same way as we did in arrays, except in the case of a string the index number refers to the letter of the string.

So a string is basically an array with letters as the elements..

We can copy the first letter by

var firstChar = userName.slice(0,1);

This copies the first character ( index 0 to 1 ) to a different variable.

What about the rest of the characters??

Var otherChar = username.slice(1)

Since end limit not specified therefore it copies the whole remaining string

Now transform the firstChar into capital and otherChar into lowercase and concatenate the both and bam you’re done.

For practical problems see this chapter in book.

1. Strings: Finding Segments

Now you want to search for a specific phrase in a paragraph and censor it or replace it.. What do you do?

Let’s say you are a Harry Potter fan and you want to censor the name of Lord Voldemort and replace it with He who should not be named..

1. Run a counter that looks for lord Voldemort.
2. Check to see if from 0 to 14 to see if lord Voldemort exists
3. Then slice the para from 0 to i and from i + 14 and concatenate He who should not be named in between..

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

if (text.slice(i, i + 12) === "World War II") {

text = text.slice(0, i) + "He who should not be named" + text.slice(i + 12);

}

}

JS has made this even easier

Using indexOf we can find the index of the first character of the required phrase. In case the phrase doesn’t exist it returns the value of -1.

var charIndex = text.indexOf(“JavaScript”);

if (charIndex !== -1){

text = text.slice(0,charIndex) + “JS” + text.slice(charIndex+10);

}

We can determine the last character of the phrase too by lastIndexOf

var charIndex = text.indexOf(“JavaScript”);

if (charIndex !== -1){

text = text.slice(0,charIndex) + “JS” + text.slice(text.lastIndexOf(“JavaScript”));

}

1. Strings: Finding a character at a location

Now we only know one way how to extract a character at a single location and that is by slice method.

The more efficient way to do that is using charAt method

var firstChar = myString.charAt(0);

var lastChar = myString.charAt(myString.length-1);

Why -1? Because length starts from 1 and index from 0.

This method is efficient in detecting and letter or symbol that you are looking for..

1. Strings: Replacing Characters

Now we have learnt 2 different ways of replacing characters or phrases.

1. Using a loop and slice(i,i=12) and concatenate method 🡺 (Dark Ages)
2. Using a loop and and indexOf and concatenate method 🡺 (Middle Ages)

Now the easiest is replace method..

:D

text = text.replace(“Lord Voldemort”, “He who should not be named”);

Now this only replaces the first instance of the phrase encountered. For replacing all instances we have to use global replace.. in that we only change the quotes of the to-be-replaced-phrase with / (back slash) and /g . This is known as Global replace.

It is a regExp thingy.

We can use “gi” instead of a g at the end to make it case insensitive

text = text.replace(/Lord Voldemort/g, “He who should not be named”);

1. Rounding off

All math functions begin with “Math.” .

To round off we use Math.round(myVar)

To round up we use Math.ceil

To round down we use Math.floor

1. Random numbers

For generating a random number we use

Math.random();

This gives a 16 digit decimal random number. This is a float value.

Can convert it to integer by multiply it by 1E16 or 1 followed by 17 zeros.

Now since the random number is always a fraction of 1, we can convert it into a dice roll by simply multiplying by 6 and rounding off.

Math.random()\*6;

But there is a problem with that..

It includes 0 and 6. Whereas 0 is not in a dice.

So what if we round up? I mean that should do the job right?

Well not quite so, because Math.random can return 0 as a value and when 0 is rounded up it gives you…. Well a zero :P

Hence we round down after adding a 1 to the value

Why add 1, well to avoid the zero b/c 0+1 = 1 which when rounded down is still 1, and why round down b/c it may return 5.99 which increments to 6.99 which on round down will return a six.

For this we use

Math.floor(Math.random()\*6 + 1);

1. Converting Strings 🡺Integers, Decimals

We can simply use numbers stored as strings in mathematical expressions and JS does the math. You can add, subtract, multiply & divide any string number by any “numeric” number or any other string number keeping in mind that if you add 2 string numbers or a string number with another one it concatenates.

To overcome that just add a plus (+) sign before the first string number OR use

parseInt(anyStringNumber);

The only issue with the above technique is that it rounds down the numeric string in case of it being a float.

For that use

parseFloat(anyStringFloatNumber);

Now keep in mind that in case you subtract, multiply or divide a numeric string with an actual string it will give you an NaN (Not a number) output.

1. Converting String to Numbers and Vice versa

Now instead of differentiating between float and integer conversion of any number string to number, you can simply use

Number(myString) (With a capital N)

And in case you have a long number and want your user to see it in quotes you can simply convert it into a string

myNumString = myNum.toString();

1. Controlling float length

To set the no. of decimal places in a float number you can use

myFloat = myLongFloat.toFixed(2);

If you leave the toFixed parenthesis empty it makes it an integer.

Although there is a catch to toFixed(), in some browsers it rounds down at 5.. so use slice and replace the 5 by 6 in it, make it a number again and send it to toFixed

On a side note you can use toInt() as well if you want to convert it onto an integer.

1. Current Date & Time

var myDate = new Date();

This creates a date with the following format:

1. Week day ( 0 for sun & 6 for sat )
2. Month (0 for Jan & 11 for dec)
3. Date (1-31)
4. Year
5. Hour (24-hour format)
6. Minutes
7. Seconds
8. GMT offset
9. Timezone

Now remember myDate is basically an object so you cannot use commands like charAt, slice, indexOf etc on it. You can although convert it into a string.

var actualDate = myDate.toString();

var actualDate = myDate.toTimeString();

var actualDate = myDate.toDateString();

similarly there are toLocaleDateString() which displays the date as the actual format that humans use and local time string in am pm.

or simply get the required information by the following methods

var myDay = myDate.getDay();

1. Extracting parts of Date & Time

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Gets | Range | Example |
| getDay() | Day of the week | 0-6 | 0 for sunday |
| getMonth() | Month | 0-11 | 0 for jan |
| getDate() | Day of month | 1-31 |  |
| getFullYear() | Year |  | 2018 |
| getHours() | Hour | 0-23 | 0 is midnight  12 is noon  23 is 11 pm |
| getMinutes() | Minute | 0-59 |  |
| getSeconds() | Second | 0-59 |  |
| getMilliseconds() | Millisecond | 0-999 |  |
| getTime() | Milliseconds since midnight jan 1, 1970 | | |

1. Specifying a date & time

Now say you want to calculate the days remaining in your graduation, well how do you do that?

Simple get the current date, specify the future date (& or time) and simply calculate the difference of them both (milliseconds) and convert it.

var msDiff = new Date("June 30, 2035").getTime() - new Date().getTime();

var daysTillGrad = Mathfloor(msDiff / (1000 \* 60 \* 60 \* 24));

And this is how you specify the future or past date and time

var d = new Date("July 21, 1983 13:25:00");

1. Changing elements of date & time

|  |  |  |
| --- | --- | --- |
| Method | Example (d is var for date) | Result |
| setFullYear() | d.setFullYear(2010); | Year changed to 2010 |
| setMonth() | d.setMonth(2); | Month is 2 (feb) |
| setDate() | d.setDate(14); | 14th |
| setHours() | d.setHours(21); | 9 pm |
| setMinutes() | d.setMinutes() |  |
| setSeconds() |  |  |
| setMilliseconds() |  |  |

So on..

Now each command / line changes 1 element in the date object and the others remain intact

1. Functions

Think of functions as your servants or workers. You can order it to bring you something (like a value), or simply do any other work.

The main objective of functions is that to not repeatedly code for the same task over and over again.

Another objective to use functions is to secure data and not leave out any global variable (especially necessary when communicating with server side). Global and local variables are later discussed in chapter 38.

The syntax is:

function NameOfFunction(parameters) {

code…

return a;

}

Now the function name is also the same as legal variable names.

Any type of code can be written in the function.

Now the parameters are separated by a comma for multiple parameters.

We can return “a” as a variable which can contain any value. We can directly return a mathematical line too such as a + b.

# Void functions:

Functions which do not return any value are called void function.

Functions should technically be declared at the top before any other JS code so that they would be ready when called.

But now even if you declare it later JS saves the functions first in its memory.

Simply call the function by its name and parenthesis (in case of parameters, should also be written).

Eg:

myTimeFunction()

OR

myAdd(a,b)

1. Giving data to functions

Once you have declared a function with parameters, when you later call that function you can simply give either values (eg: 1,2,4.56 etc), or you can also pass variables as parameters.

1. Passing Data Back from Functions

You can pass back data from functions with the keyword return..

Now remember once JS has read and executed the return statement it considers the function ended. Even if it contains more code after the return statement it still doesn’t read or execute those statements.

1. Local vs Global variable

There are 2 variable scopes. Global and local.

Variables with global scope are global variables and those with local scope are local variables.

# Global Variable:

The one declared in the main body of your code (not inside a function). Can be accessed from anywhere in the code.

# Local Variable:

Declared inside a function. Can only be accessed from within the function.

Although if you declare a variable in your code without var it automatically becomes a global variable even if within a function.

The only exception to this is your parameters of the function when declaring the function.

Now you can change the value of a global variable from within a function but it is good practice to return that value explicitly.

1. Switch Statements

See the syntax and you can understand it quite clearly

switch(myNum) {

case “1” :

alert(“You came first”);

break;

case “2”:

alert(“You can second”);

break;

default:

alert(“You failed”);

}

Here myNum is any variable having a certain value for which we are checking. The default is added in case no condition matches.

1. Switch Statements: Understanding them

switch(myNum) {

case “1” :

alert(“You came first”);

break;

case “2”:

alert(“You can second”);

break;

default:

alert(“You failed”);

}

Now here we have added a break statement after each case which is to optimize our code and not to allow it to waste time and memory resources to check all other statements too even after a true match has been found.

If you do not use a default case no worries. But the coders, even if they do not use a default case, and a case block is their last in the switch statement, they tend to add a break at the end because if they later add more statements to the switch case they do not have to remember to add another break before.

1. While loops

While loop is almost same as the for loop except for the arrangement of the conditions (Initiation; limitation; increment/decrement).

Here is a typical while loop:

var j=0;

while(j<=10) {

Some awesome code;

j++;

}

1. Do While loops

It is the same as the while loop except for the arrangement and addition of a do statement.

var j=0;

do{

again awesome code;

} while(j<=10);

Now it is possible that you design such a statement (in for or while loop) that never runs a loop because the statement is contradicting. But in the case of do while loop the statement is run at least one single (the first) time because it does not check the validity and limitation of the condition until the do statement runs.

1. Placing Scripts

Can place anywhere but is preferred to be

* + - 1. In an external file
      2. At the end of the page (so the images and html and css of the page don’t get stuck while loading the heavy gun JS)

# Internal JS

Using <script> </script> anywhere in the html tags of the webpage

# External JS

Using <script src=””></script> anywhere

1. Commenting

Single line comment, Just add // before the line

Multi-line comment, (like css) add /\* before and \*/ after

1. Events:link

# Events:

All the user actions such as moving the cursor, typing a key or entering, changing from one filed to another, selecting an option, these are called **Events.**

# Event Handler

The JS code that responds to the event is called an event handler.

Now there are different type of JS event handlings

* + - 1. Inline-handing
      2. Handling by Scripting

For now we’ll use inline handling.

We simply use the keywords onClick=”anyFunction()” with the HTML opening tag and bam it is a functional event.

We can also define and code the whole function within the quotes too.

Since you cannot nest double quotes within double quotes or single within single hence we use singles within doubles and all the more reason not to define functions inline.

Therefore we only refer to the functions even if we use inline event handlers.

* 1. Events: button, Mouse, Fields

onClick=”callThisFunction();”

Now if, in the case of an anchor tag, we do not want a click to refresh the page i.e. we do not want a hash # in the href. What happens due to that is the page refreshes unnecessarily and a # comes in the url.

The point was we do not want that. We just want a link which does not have any effect on html but it does run a JS function in the back. How do we do that?

We use something like:

<a href=”JavaScript:void(0);” onClick=”aFunction();”></a>

Now you can apply this to an image, a button, a link any element but the point is it will work.

Similarly for mouse we have

onMouseover=””

onMouseout=””

And for fields we have

onFocus=””;

onBlur=””

Now there are a few things that we can do with these inline event handlers which are mentioned in the upcoming chapters

49. Reading field values

Now first off we have to revise how we used to submit forms.

<form onSubmit=”thisThat();”>

Some awesome fields

<input type=”submit” value=”submit”>

</form>

Now to get the value of a specific field we have,

var userMail = document.getElementById(“email”).value;

and if we don’t use the value at the end of it we get the JS version of the element.

50. Setting Field Value

Suppose that your user enters a value in one field and now you want to automatically enter the value of another field based on that field. Like a zip code automatically enters the town name or city name.

document.getElementById(“anyThing”).value = theCity;

51. Setting para text

Suppose you have an incomplete paragraph with click for more at the end.

Now that click for more has a click for more at the end of it.

Remember that this anchor tag does not have a # in the href but a JavaScript:void(0); in it.

Just get the element and change the innerHTML.

If you do not want to simply replace the whole text use use += after the innerHTML.

52. Playing with images & text

Remember since we manipulate the styles and stuff of elements hence it is absolutely necessary for us to give all the elements (to be changed) unique IDs.

Now to put a certain class on the text or the image to change its properties like zoom in the picture and zoom out etc so for that we have to add on a class to the element.

So to add a class to the element and remove all others we use:

document.getElementById(“pic”).className = “zoom”;

Now if we want to add the zoom class and not remove the others we use a += instead of an = and a space before the class we write like “ zoom”.

53. Swapping Images

To change an image on click or on hovering mouse or stuff we simply use the onMouseover and onMouseout on the element.

<img src=”img1.jpg” id=”pic” onMouseover=”src=’img2.jpg’”>

If we do not want do this inline we can use a function which then later on changes the image.

<img src=”img1.jpg” id=”pic” onMouseover=”change(Id,’img Src’)”;

change(a,b){

document.getElementById(a).src = b;

}

54. getting Elements by id in style

Simply put the document.getElementById(“pic”) in a var and use it elsewhere in the whole function, simply by

a.src = “img2.jpg”;

similarly we can also extract the href from an anchor tag by

var b = a.href;

55. Setting Styles

Now suppose you don’t want to change the whole class of the element instead just change a parameter a value of a certain property.

For that we use

document.getElementById(“this”).style.fontSize = “2em”;

Or

document.getElementById(“that”).cssFloat = “left”;

document.getElementById(“that”).visibility = “left”;

document.getElementById(“that”).margin = “0 10px”;

In case you just want to read the value

Like a.style.margin or .cssFloat or something, it does not tell you the value of the style in an external file or a internal style. It only gives you the properties of the inine styling.

Although if you do want to get the styles of an elements which are not written inline then you use document.getComputedStyle(“id”).margin or whichever property.

Now you should get the whole list of how certain css properties are referred-to by JS.

56. Targeting all elements by tag names

.getElementsByTagName(“p”);

Now that is the syntax.

But in actual practice we assign the getElementsByTagName to a variable (Which is an array in actual).

Like

var para = document.getElementsByTagName(“p”);

now para is an array with the first para being para[0], second being para[1] and so on.

Now here you can use a loop to apply a certain style to each paragraph.

(Note it doesn’t allow to apply it directly without loop, I tried it!)

57. Targeting tag names within an ID

Now if you only want to target the tag names in a specific id you can do the following:

Firstly we target the id:

var a = document.getElementById(“fish”);

var b = a.getElementsByTagName(“p”);

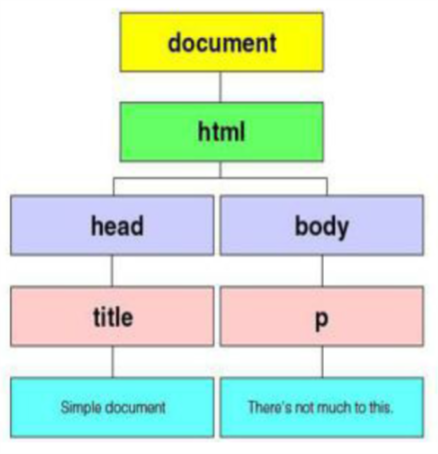
Apply styles

and so on.

58. The DOM

The approaches to targeting elements that we learnt from last 2 chapters, we know that we can target elements by their number and tag names and can target them even if they are within an id. Now for an even more refined work we use the DOM approach.

* DOM stands for Document Object Model
* It is an organization chart automatically created when page loads (like a map)
* It has the locations of all things such as the tags, text, images, links, lists, tables etc.
* Hence you can edit or change anything on an html page by telling JS where it is on the map
* Here is an example o how a DOM exists
  + Document (1st Level)
  + HTML (2nd level)
  + Head (3rd level)
  + Title (4th level)
  + Title text (5th level)
  + Body (3rd level)
* And so on.



* Nodes:
  + Here each box is a node
  + There are 3 fundamental nodes:
    1. Document
    2. Element (HTML, Head, Body, p tags)
    3. The text in them (Strings and other data)

59. The DOM: Parents and Children

* Each node is enclosed, packed within another node (with the exception of the document)
* The inner node is the child of the node enclosing it (outer node)
* All the elements and/or text within a node are siblings of same level

60. The DOM: Finding Children

Now you can target a certain element by the 3 methods that you learnt previously i.e.

1. By Id
2. By tag names
3. By tag names within an Id

Or you can use the DOM childNode method as:

var a = document.childNodes[0].childNode[1].childNode[1].innerHTML;

var b = a.innerHTML;

1. Go to document
2. Go to its 1st child (HTML)
3. Go to its 2nd child (body)
4. Go to its 2nd child (the 2nd sibling div within body)

Now this could also be easy if we targeted any id and gotten its childNodes.

61. The DOM: Junk Artifact & nodeType

In an ideal world the way we referred to the nodes in the last chapter would work perfectly. But that doesn’t happen here. The browsers reads the enter and whitespace for indentation as text.

# Solution 1:

Write your html markup as this

<html

><head

></head

><body

></body

></html>

This seems very desperate but does the job without any interference from JS.

# Solution 2:

Make your code compact by minifiers out there like

<http://www.willpeavy.com/minifier/>

But do keep the original file too for future editing.

# Solution 3 (JS):

Here we write our HTML code as we like and let JS take out the junk nodes.

Here we introduce the concept of node type.

var a = target-node.nodeType;

here target node is any node like childNodes[1] etc.

And nodeType is 1 when it is any tag and 3 when it is any text.

1. Now we get a div by its id
2. We loop and check whether we have encountered a tag or the text (junk text).
3. If the tag we want to target is not the only child we have to loop through the childNodes and check the type and then we also have to get the second tag.

And this junk is the reason why most coders prefer the getElementById method although some things (discussed later) can only be achieved by DOM methods.

62. The DOM: More ways to target elements

We can refer to nodes as

.firstChild

.lastChild

.nextSibling

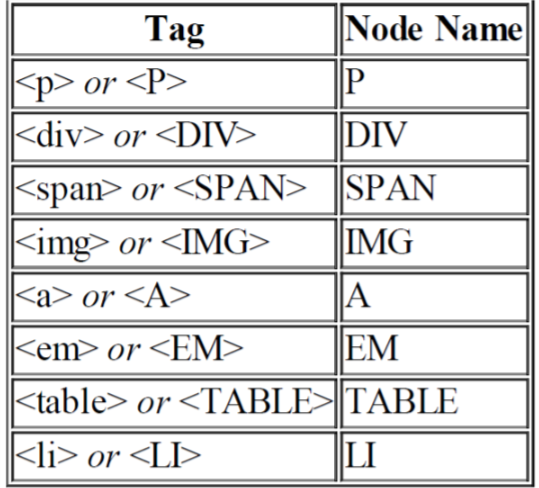
.previousSibling

.parentNode

If there are no siblings or a child JS should return null but again junk may exist so its better to use an id to target a node and using getElementById.

63. The DOM: Getting a target’s name

Earlier we have learnt how to get the nodeType. Similarly we can also get the name of the node by nodeName.



If the node is text then #text in small is assigned to the variable.

If the node is text