**Javascript**

* Developed circa 1995 by Brendan Eich at Netscape Communications as the scripting language for the Netscape Navigator Browser
* Formerly called Mocha, then LiveScript, then JavaScript
* Standardized by ECMA International as ECMAScript

**Latest version:** JavaScipt 1.8.1, ECMAScript 5 [ECMA-262 5th Edition]

**Common version:** JavaScript 1.5, Jscript 5.5, ECMAScript v3 [ECMA-262 3rd Edition]

**Document Manipulation**

-embedded script

-externally link script

Async – middle

Nonasync defer – at the end

**Call to Function**

document.getElementByID – returns 1 element only

text.innerHTML

h1.outerHTML

h1.textContent

h1.title

h1[“lang”] h1.lang

h1.setAttribute(“title”, “value”)

h1.getAttribute(“lang”)

Attributes that are not part of basic sets

h1.innerText = “hello”;

h1.style = “color:blue”;

h1.style.color = “red”;

h1.style.fontsize = ‘2em’;

h1.documnet.getEmentID(x);

h1.className = ‘abc’

Changing Style:

Stylesheet/change the styleID

document.getElementsByID - node list

- returns a set

document.getElementsByTagName

document.getElementsByClassName

document.getElementsByTagNameNS

document.getElementsByName

On a subsection of a document (everything)

Thoughout the h1 / paragraph

querySelector(‘p’)- returns only the 1st one

querySelectorAll(‘p’) – returns all matching elements

querySelectorAll(‘p:first-child’)

querySelectorAll(‘p:first-of-type’)

querySelectorAll(‘body h1 p’)

querySelectorAll(‘h1 + p’)

For document or for specific elements (matches the selection)

p.document.querySelector(‘p’)

p.document.querySelector(‘h1’)

p.document.querySelector(‘body’)

p.document.querySelector(‘body > p’)

document.head.childNodes

[text, meta, text, title, text, comment, text, style, text]

document.head.children

[meta, title, style]

Possible to Traverse the Entire

document.body.children.length

document.body.children

document.body.children[0]

document.body.childElementCount

document.body.firstChild

document.body.firstElementChild

document.body.lastChild

document.body.lastElementChild

document.body.firstChild.nextSibling

document.body.firstChild.nextElementSibling

document.body.firstChild.previousSibling

document.body.firstChild.previousElementSibling

document.body.parentNode

document.body.parentElement

wondow.screen.width

window.navigator.vendor

P.NodeType

p.NodeName

p.NodeValue

p.firstChild.NodeType

p.lang = ‘en-US’

p[‘lang’] = ‘en-US’

p.setAttribute(‘title’, ‘ksdfg’)

p.style.color = ‘blue’

p.style.backgroundColor = ‘blue’

Dynamically Manipulate

p.innerHTML = ‘hello <em> there </em> handsome’

p.childNodes

-text, em , text

var h = document.createElement (‘h1’)

h

<h1></h1>

document.body.appendChild(h)

<h1></h1>

var t = document.createTextNodes (‘hi’)

t

“hi”

var t1 = document.createTextNodes (‘singit’)

t1

“singit”

h.”hi”appendChild(t)

h.insertBefore(t1, t)

var t2 = document.createTextNodes (‘palit’)

t2

“palit”

h.replaceChild(repl, ref)

h.replaceChild(t2,t1)

h.replaceChild(t1)

h.insertAdjacentElement

h.append(multiple)

h.cloneNode

var h2 = h.cloneNode ()

h2

<h1></h1>

var h2 = h.cloneNode (true)

h2

<h1>”palit”</h1>

document.importNode (te, true) – want a copy and added

document.adoptNode (te) – taking/removing it then added

document.createDocumentFragment()

df

df.appendChild(document.createTextNode(“hi”))

Event Handler

1. onclick = ‘alert’

-not applicable, freezes, prevention

-status bar

2. p.addEventListener

-elements that you are handed

3 Elements:

A. p.addEventListener (‘click’, f1)

-html: function f1(){

Console.log(handler);

}

B. p.removeEventListener (‘click, f1’)

C. p.onclick = ‘’

“”

Standard Objects

Functions - abstraction for code blocks

* All functions return any type of value

function sayHello(){

console.log(‘hello!’);

}

sayHello()

hello!

Accepts Argument (any type)

function saySomething(something){

console.log(something);

}

function greater(v1, v2){

if (v1 > v2) {

return v1;

}else if (v1 < v2) {

return v2;

}

}

greater (100, 50)

‘a’ > ‘A’

‘a’

Functions that can return any type of value

function fn(args){

Switch(args){

Case 1: return true;

Case 2: return 100;

Case 3: return ‘hello’;

Case 4: return [1, 2, 3];

Case 5: return {name:’ako’, age:10};

Case 6: return function(name){return n\*2};

}

}

Ex. fn(2, 6)

200

Function Expressions

(function (a, b) {return a + b;}) (10, 20);

var add = new Function (‘a, b’, ‘return a+b’);

sum = add (10, 20);

ex: sum

30

var subtract = function(a, b){

return a – b;

}

var diff = subtract (10, 20);

Arrow Syntax

var multiply = (a, b) => {return a \* b;}

var product = multiply (10, 20);

var divide = (a, b) => a/b;

var quotient = divide (10, 20);

var triple = n => 3\*n triple(100) = 300

var thrice = triple(5); thrice = 15

var zero = () => 0; zero

() => 0

zero()

0

Functions can be recursive

function factorial(n){

if (n < 0){

throw “Invalid argument”;

}else if (n == 0){

return 1;

}else{

Return n \* factorial (n-1);

}

}

Ex: 3!

3\* (2!)

2\*1!

1\*0!

1

factorial (-1)

error

factorial (1)

value

Functions can be nested

function outer(){

//code

function inner(){

//code

}

//code

}

‘this,’ ‘arguments,’ and ‘apply’….

-refers to current object

function args(){

console.log(this);

console.log(arguments.length);

for(let i=0; I < arguments.length; i++){

console.log(arguments[i]);

}

}

args();

args(1,2,3);

args.apply(‘hi’, [1, 2, 3, 4]);

Ex:

args()

sum()

0

sum(300)

300

Function arguments can have default values

function fn (a = 10, b = 20, c, d = 40, e){

(‘a = ${a}’);

(‘b = ${b}’);

(‘c = ${c}’);

(‘d = ${d}’);

(‘e = ${e}’);

}

Functions rest parameters

function fn (a, b, …. others){

(‘a = $ {a}’);

(‘b = $ {b}’);

(‘others =’ , others);

}

fn()

a = undefined

b = undefined

others = []

fn(1)

a = 1

b = undefined

others = []

Arrays

var emptyArray = new Array(); //constructor

var alsoEmptyArray = []; //literal

length – writable

Can be mix

[10, true, ‘hello’, new date()]

Var matrix = new Array(

new Array(1, 2, 3)

);

matrix[0]

(1, 2, 3)

Var multiDimArray = [

[ ];

[ [ ] ];

[];

]

table(multiDimArray)

Array Destructuring

var array = [1, 2, 3, 4, 5];

var [a, b, c, d, e] = array;

var [m, n, … others] = array;

var [ , x, , y] = array; //skip

Array indexes can be non-contiguous

var array = [1, 2, 3, 4, 5]

array [10] = 10;

for (let I in array){

console.log (I, array [i]);

}

Multicore Methods – modify

.push - add 1 element

.shift – remove 1st element

.reverse – changes in place

.splice – remove elements I array then insert

Iterator:

Element, index, array

array.forEach(function(element, index, array)){

console.log(element);

}

var array = [1, 2, 3]

array.length = 10;

array[array.length] = 4

var newlength = array.push(5)

firstElement = array.shift(0)

array.reverse();

elementsRemoved.splice(2, 3, ‘a’, ‘b’, ‘c’);

array.length = 2

Methods:

1. Mutators – modify arrays

Array.prototype.fill()

Array.prototype.pop()

2. Accessor – does not modify arrays

Array.prototype.concat()

Array.prototype.indexOf()

Array.prototype.join()

3. Iterators – execute something for each elements in array

Array.prototype.filter()

Array.prototype.map()

Array.prototype.every()

Array.prototype.reduce()

a.filter(function(v, index, are){true})

a.filter (v = > v% v ==0)

a.filter(v,i = > i % 3 == 0)

[1, 4, 7]

a.filter ((v, I, a) = > i% 3 ==0 && a[i - 1] ==3)

[3]

a.filter (v => true)

a[1, 2, 3, 4, 5, 6, 7]

a.map(v => v\*2)

[2, 4, 6, 8, 10, 12, 14]

var allPositive = array.every(e => e > 0)

return true;

reduce –> (function accumulator())

a.reduce(a, v) = > a += v

28

a.reduce (a, v) => a += v, 10

38

Var str = ‘saint louis unviersity’;

Var ots = str.split(‘’).reverse().join(‘’);

<’reverse SLU’>

Objects

Object constructor = new object [];

Literal syntax = new object({});

Literal syntax = new object(true);

Literal syntax = new object(10);

Literal syntax = new object(‘hi’);

Objects Properties & Mthods

var students = {};

student.idno = ‘2153118’;

student[‘name’] = ‘Juan Dela Cruz’;

student.toString = function(){

return ‘ $ {this.idno}: ${this.name}’;

}

JSON

var student = {

}

var address {

}

var subjects: [

{

}

]

Ex. student.address.cityOfProvince

student.subject[0].course.no

JSON.stringify(student)

localStorage.setItem(‘stu’, JSON.stringift(stu))

var o = JSON.parse(‘{“nagan”: “siak”}’)

o.nagan

“siak”

localStorage.getItem(‘stu’)

Ex. Circle

Object <- radius = 5

getArea(){

}

circle.area

89……

Instance of Objects

-student

-circle

Constructor Functions

Function Person (name, age){

this.name = name;

this.age = age;

this.speak = function{

}

constructor (‘Hello! I’m ${this.name}’);

}

var p = new Person (‘Britney’, 21);

p.speak();

console.log(p instanceOfperson);

p = new person (‘sdfg’, 12)

p instanceOfPerson

true