Lesson 1

Client-side scripting - for interfactivity

- querySelector only returns first match (single value)

example:

querySelector(#p)

RESULT: <p id = "p"> hello </p>

querySelectorAll returns all matches

- querySelectorAll returns all matches

example:

querySelectorAll ('p')

RESULT: [p#p]

- document.body.matches('html > \*')

RESULT: true

EXPLANATION: html > \* : all child nodes of HTML

- p.textContent = 'hi'

RESULT: "hi"

- p.nodeValue

RESULT: #null (always!)

- p.nodeName

RESULT: #text

- p.nodeType

(numeric - result: 3, constant in DOM)

<a href = ' '> link </a>

- p.lang = 'en-US'

RESULT: "en-US"

- p.setAttribute

- p.getAttribute

- p.style = 'color: blue'

RESULT: "color: blue"

- p.style.color = '#0ff'

RESULT: '#0ff'

- p.style.bottomBorderColor

move UP : parentNode

move DOWN : childNode

move SIDEWARDS : nextSibling // previousSibling

whitespaces are read as text

comments are read as comments

- document.body.childNodes - return everything (i.e. element, comment, whitepace)

- document.body.children - only returns elements

- document.body.parentNode - always returns the parent

- document.body.parentElement - can return null

- p.innerHTML

insert \ / remove

>CHILDREN <

replace / \ append - add to the end

- h1 = document.createElement('h1')

RESULT: "<h1><h1>"

- document.body.appendChild(h1)

RESULT: "<h1></h1>"

- t = document.createTextNode('hi')

- h1.appendChild(+)

- t2.document.createTextNode('hello')

- h1.insertBefore(t2, +)

- var t3 = document.createTextNode('pamalit')

- h1.replaceChild(t3, t)

- h1.removeChild(t2)

- h1.cloneNode

- h2 = h1.cloneNode(true)

- var b = document.body.cloneNode(true)

RESULT: undefined

- b

RESULT: <body>..</body> ~entire body

- document.importNode (similar to clone)

- document.adoptNode (transfer nodes, original disappears)

Lesson 2  
programmatic constructs (javaScript)  
- DOM is NOT a part of javaScript  
functionalities:  
Javascript is an impeerative programming language  
-> dynamically type language  
> var a = 100  
> typeOF a  
RESULT: "number"  
-> data types are not explicitly defined  
ECMA Script 5  
variables:  
a. var - variables with a 'var' keyword becomes a global variable  
b. let - variable only exists within the block (similar since i is only accessible in the for-loop where it was instantiated)  
 - veriable is NOT associated w/ windows -> for (int i){}  
c. const - new values can NOT be changed  
 - doesn't introduct a global property of the variable  
e.g. a = 100  
 RESULT: 100  
 window.a  
 RESULT: 100  
(1)  
var a = 10;  
function f(){  
 var a = 20;  
console.log(a);  
 {  
 var a = 30;  
 console.log(a);  
 }  
 console.log(a);  
}  
RESULT:  
f(): 10  
 20  
 30

(2)

let a = 10;  
function f(){  
 let a = 20;  
 console.log(a);  
 {  
 let a = 30;  
 console.log(a);  
 }  
console.log(a);  
}

RESULT

f(): 20  
30  
20

(3)

const a = 10;

function f(){

const a = 20;

console.log(a);

{

const a = 30;

console.log(a);

}

console.log(a);

}

RESULT:

f(): 20

30

20

Lesson 3   
data types:  
A. SIMPLE/PRMITIVE  
B. REFERENCE/OBJECT  
A.  
- boolean - for controlling conditional statements  
EXAMPLE  
b = false  
typeof b (RESULT: "boolean")  
'falsy' - undefined is false (values not defined)  
 - example is value '0' (everything else is true)  
'truey' - example is String 'hello' (null values are false)  
 - empty values are false  
 - example: while (1){} <- infinite loop  
- numbers - can be in decimal, binary (prefix 0b), octal (prefix 0),  
 hexadecimal (prefix 0x), scientific notation, floating point  
 - alwars floating point division  
- strings - single quote or double quotes can be used  
 EXAMPLE: "cam't", 'can't', 'abc", "def', 'can\'t' -> RESULT: "can't" template literal:  
 e.g. 'abc  
 def  
 ghi' -> back pick(?)  
 RESULT: "abc  
 def  
 ghi"  
 string interpolation:  
 e.g. 'price: ${a}'  
 RESULT:  
 price: 100  
- undefined is different from null

Lesson 4  
B.  
- standard/core objects (structure types)  
Array, Boolean, Date, Error, Function, JSON, Math, Number, Object, RegExp, String, Set, WeakMap, Weakest  
\* in js, everything is an object (?)  
'hello'.length >> 5  
'hello'.toUpperCase >> "HELLO"  
'hello.substr(2) >> "llo"  
 |-> starting character  
s = "hello"  
String.prototype.reverse = function{  
 var rev = ' '  
 for (let i = this.length - 1; i >= 0; i--){  
 rev += this.charAt(i);  
 }  
 return rev  
}  
'hello'.reverse()  
RESULT: "olleh"  
Math.Pi >> 3.1415...  
Math.floor(1.2) >> 1 (smallest # smaller than 1.2)  
Math.ceil(1.2) >> 2 (smallest # larger than 1.2)  
Math.random() >> 0.316549

d = new Date()  
d.getHous()  
d.getMonth() - zero based (Jan = 0)  
d.setYear()  
d.getTime() - milliseconds from January 1, 1970

Lesson 5  
ARRAYS - dunamically sized  
2 wars: array object constructor  
 var emptyArray = new Array();  
 array literal syntax  
 var alsoEmptyArray = [];  
 - storage is NOT allocated statically  
e.g. var arrayWithLengthFive = new array(5);  
 var arrayWithOneElementWithValueFive = [5];  
e.g. var mixedElement = [10, true, 'hi', new Date()];  
 var matrix = new Array(){  
 new Array = (1, 2, 3, 4)  
 new Array = (5, 6, 7, 8)  
 new Array = (9, 10, 11 12)  
 }  
 try: table(matrix)  
e.g. var multiDimArray {  
 ['apple', 'banana', 'orange', [ 4, 5, 6, [7, 8, 9], true, false, false], []  
}  
e.g array destructuring:  
 var array = [1, 2, 3, 4, 5]; -\ value of a: 1 - value of b: 2  
 var [a b, c, d, e] = array; -/ value of c: 3  
 value of d: 4  
 value of e: 5  
 var [m, n, ...others] = array; - m: value 1, n = value 2, others = array  
 var [, x, y] = array;  
 \-> skip 1st element  
 \-> skip 3rd and 4th element  
e.g.array indeces can be non-contiguous  
 var array = [1, 2, 3, 4, 5]  
 array [10] = 10;  
 for (let i in array){ <- if this is an array, it will step into indices one at a time  
 console.log(i, array[i]);  
 }  
e.g. function (a, b){  
 a = 10;  
 b = 20;  
 [a, b] = [b, a]  
 }  
e.g.var array = [1, 2, 3];  
 array.length = 10; (7 undefined slots available)  
 array[array.length] = 4;  
 var newLength = array.push(5);  
 var firstElement = array.shift (0); remove and adjust  
 array.reverse(); >> 5 4 undefined 3 2  
 var elementsRemoved = array.splice (2, 3, 'a', 'b', 'c')  
 >> 5 4 'a' 'b' 'c'  
 array.lengfth = 2  
 RESULT: 5, 4  
 new length = 12  
e.g.accessor methods (arrays are not modified) -> but returns a value  
 - indexof  
 - find  
e.g.iterators - to step through elements one by one and do something  
 |  
 -> argument passed in a function - RETURN value  
 - index  
 - array  
array.forEach(function(element, index, array){  
 console.log(element);  
 }  
 array.every  
 array.something  
 array.filter

a = [1, 2, 3, 4, 5, 6, 7, 8]  
 a.map (v = > v \* 2)  
 |-> same as  
 function (v) {  
 return v\*2  
 }  
- without iterators,  
 res = []  
 for (i = 0; i < a.length; i++){  
 + = a[i] \* 2  
 res.push(t);  
 }  
 return res  
- to take only even numbers  
 a.filter (v = > v % 2 == 0)  
 RESULT: [2, 4, 6, 8]  
- odd  
 a.filter(v => v % 1 == 0)  
 RESULT: [1, 3, 5, 7]  
- indeces which are divisble by 3  
 a.filter ((v, i) => i % 3 == 0)  
 RESULT: [1, 4, 7]  
- indeces divisible by 3 and preceeding element is 3  
 a.filter((v,i,a) => i % 3 == 0 && a[i-1] == 3)  
 RESULT: [4]  
- calculate sum of elements  
 a.reduce((acc, v) => acc += v)  
 RESULT: 36  
- accumulator starts from some value  
 a.reduce ((acc, v) => acc += v, 10)  
 RESULT: 46 |-> accumulator starts from 10  
- every - check every element  
 example: a.every (v => v > 0)  
 RESULT: true  
- some  
 a.some(v=> v % 2 == 0)  
 RESULT: true  
- findIndex  
- forEach  
- keys  
- entries  
- map  
- reduce  
- reduceRight  
- values

Lesson 6  
function - abstraction for a code block (alaways has return value)  
e.g.function sayHello(parameter) { \  
 console.log('hello'}; <- body > block  
 } /  
 to invoke: sayHello()  
e.g.function saySomething (something){  
 console.log(something);  
 }  
 to use: saySomething (something){  
 console.log(something};  
 }  
 to use: saySomething('hi') >> 'hi'  
 saySomething(100) >> 100  
e.g.function greater(v1, v2){  
 if (v1 > v2){  
 return v1;  
 } else if (v2 > v1){  
 return v2;  
 }  
 }  
 to invoke: greater(10, 20) >> 20  
 greater('a', 'b') >> "b"  
 greater('a', 'A') >> "a"  
 greater(true, false) >> true  
 greater([], {}) >>  
e.g.functions can eturn any type of value  
 function fn(arg){  
 switch(arg){  
 case 0: return true;  
 case 1: return 100;  
 case 2: return 'hello';  
 case 3: return [1, 2, 3];  
 case 4: return {name: 'Ako', age: 10}  
 case 5: return function (num){ \  
 return num \* 2 > anonymous function  
 } /  
 }  
 }  
e.g.function expression (if the function will only be used once)  
 (function (a, b) {return a+b;})(10, 20);  
 RESULT: 30  
 \*To avoid name conflicts  
e.g.var add = new Function ('a, b', 'return a+b')  
 var sum = add (10, 20);  
 functions are executable code,  
 they can be assigned to variables  
e.g. animous function syntax  
 var subtract = function(a, b){  
 return a-b;  
 }  
 var difference = subtract (10, 20);  
e.g.arrow syntax  
 var multiply = (a, b) => {return a \* b;}  
 var product = multiply(10, 20)  
 if functions only have one argument, () can be omitted  
 var triple = n => 3 \* n;  
 var thrice = triple(5);  
e.g.functions can be recursive  
 function factorial(b){  
 if (n < 0){  
 throw "invalid argument.";  
 }else if (n==0){  
 return 1;  
 }else{  
 return n\*factorial (n-1);  
 }  
 }  
 }  
e.g.functions can be nested  
 function outer(){  
 //some code...  
 function inner(){  
 //inner code...  
 }  
 //some more code...  
 }  
e.g.execution context object - 'this', 'arguments' -> can be used if arguments have no names  
 -> like an array  
 function args(){  
 console.log(this);  
 console.log(arguments.length);  
 for (let i = 0; i < arguments.length; i++){  
 total += arguments;  
 }  
 return total;  
 }  
 possible invokations: sum(1, 2)  
 sum(1, 2, 3, 4, 5, 6, 7, 8, 9)  
e.g.function arguments can have default values  
 function fn (a=10, b=20, c, d=40, e){  
 console.log ('a = ${a}');  
 console.log ('b = ${b}');  
 console.log ('c = ${c}');  
 console.log ('d = ${d}');  
 console.log ('e = ${e}');  
 }  
e.g.functions can have rest parameters  
 function fn(a, b... others){  
 console.log('a = ${a}');  
 console.log('b = ${b}');  
 console.log('others = ', others);  
 }  
 try: fn(1, 2, 3, 4, 5)  
 RESULT: a = 1  
 b = 2  
 others = 3, 4, 5

Lesson 7  
Mutator methods  
- fill  
- push - add elements and increase length \  
 > STACK  
- pop - remove last element and return value /  
- shift - remove first element and shift \  
 > QUEUE  
- unshift - insert an element at the first /  
 position and shift  
- sort - sort elements of the array  
 a.sort() - normal sort  
 a.sort (function (e1, e2)) - return -1, 0, 1  
e.g.var fruits = ['apple', banna, watermelon, kiwi]  
 fruit.sort() – alphabetical  
 fruits.sort((v1, v2) => Math.sign(v1.length - v2.length));  
- splice - add/remove element from an array  
e.g.fruits.splice(2, 1) -> starting from index 2, remove 1 element  
 fruits.splice(2, 1, 1, 2, 3, 4) <- add after removing 1 element startng from index 2  
- reverse  
- peel

Accessor methods  
- concat  
 e.g. a.concat(b)  
- split - will produce an array  
- join  
 e.g.  
 var str = 'saint louis university';  
 var rts = str.split('').reverse().join('');  
 RESULT: reverse of slu

Lesson 8  
OBJECTS (Prototype Object Orientation)  
var emptyObj = new Object();  
var alsoEmptyObject = {};  
-> RESULT: Object {}

e.g.  
var student = {}  
student.idno = '2155912'  
student['name'] = 'Juan Dela Cruz';  
Student.toString = function() {  
 return '$ {this.idno}: ${this.name}  
}  
> Student.toString()  
< 2155912 :

JSON - converts objects into a string representation  
 JSON.stringify(Student)  
 RESULT: "{"id no": "2155912", "name" : "Juan Dela Cruz"}"  
 localStorage.setItem('2155912', JSON.stringify(student))  
 o = JSON.parse(student)  
 > o  
 < Object: "{idno: 2155912, name: Juan Dela Cruz}"

- typeOf  
 - instanceOf   
constructor function - use 'new' keyword  
function Peson (name, age){  
 this.name = name;  
 this.age = age;  
 this.speak = function(){  
 console.log('Hello! I'm ${this.name}.');  
 }  
}  
var p = new Person ('Britney', '21');  
p instanceOf Person  
 RESULT: true  
p1 = {"name" : "ikaw", age: 100}  
p1 instance of Person  
 RESULT: false  
p1 instanceOf Object  
 RESULT: true  
p2 = Person ('askdkj', 1000)  
 RESULT: undefined