JAVASCRIPT – BY ANGELA YU

SPECIAL THANKS TO NINAD DHULAP

JavaScript is a scripting language like acting script which actors like HTML and CSS elements to perform specific actions.

Java vs JavaScript

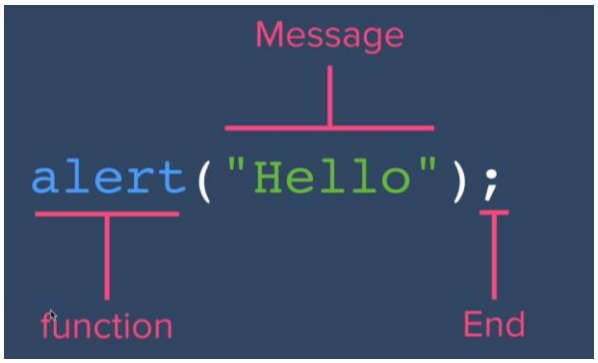
JavaScript is a interpreted programming language while java is compiled programming language. (Carpet- car & pet)

Compiler vs interpreter. 1→ 2→

In olden days interpreted languages tend to be seen as almost like toy languages. They aren't powerful and very slow, and they had to execute all the instructions line by line whereas compiled languages were seen as the more serious languages and you end up with very fast running programs.

Interpreted languages: JS, Py, Ruby

Compiled languages: Java, C/C++, Swift



“...” quotation marks says that inside sentence is a string. Coding quotation and MS-Word quotation are different so if you copy paste any code then be alert.

Data Types:

```
Var myname = “Ninad”;
```

Var says that we are creating a new variable, creating a new data container, and that container has the name myname. And then we set that variable to be equal to the value Ninad.

Var has to make a box/container and the value is stored in that box.

You can modify able to modify (as variables can vary)

1) String 2)number 3)Boolean

Variables:

You can vary the data that you keep inside it.

You can modify it.

Camel casing = var myName; (next word’s 1st letter is capital)

Functions:

- Alert (...) is used to show in pop up. (window.alert())
- Typeof (...) shows data type.
- Prompt (...) is used to get input from user.

Ex.

index.js ×

```
var tweet = prompt ("compose your tweet: ");
var tweet_len = tweet.length;
alert ("You've written " + tweet_len + " chars " + (140 - tweet_len) + " char remaining.");
```

In this you see an alert after writing of post how many letters remaining & used.

- Slice (x, y) allows you to slice and dice your strings to

  separate them into individual characters.

Ex.

```
var name = "Angela";
name.slice(0,1);
```

it sliced it and give ‘A’ only ‘A’ is at position 0.

```
var name = "Angela";
name.slice(5,6);
```

it gives last letter ‘a’.

```
var tweet = prompt("Compose your tweet:");
var tweetUnder140 = tweet.slice(0,140);
alert(tweetUnder140);
```

or

```
alert(prompt("Compose your tweet:").slice(0,140));
```

- Var.length gives length of string
- Var.toUpperCase () changes all char to the upper case.
- Var.toLowerCase () changes all char to lower case.

Ex-

```
var name = prompt ("what is your name? --> ");
var first_let = name.slice(0,1);
var cap = first_let.toUpperCase();
var rest_ofname = (name.slice(1,name.length)).toLowerCase();
alert (cap + rest_ofname );
```

Output: Ninad

- Console.log (...)- it will log the string that's contained inside here inside the console.

Difference of alert & console.log:

Alert can be seen by users, but console.log is only for developers to debug your code.

- Math.floor (…) – round a number downwards to its nearest integer.
- Math.round (…) – similar to math.floor but more mathematically correct.
- Math.pow (base, exponent) to give power.
- Math.random – generates random number. It is 16 decimal place number. (0 – 0.99……9) and never reaches 1.
  For dice – n = n * 6, n = Math.random(n) + 1;

Simple love Calulator -

```
var name1 = prompt ("name1");
var name2 = prompt ("name2");

var n = Math.random();
n = n * 100;
n = Math.floor(n) + 1;

if (n > 70) {
    alert ("Your love score is " + n + " % " + " you love each  other like kanye loves kanye");
}
else if ((n > 30) & (n < 70)) {
    alert ("Your love score is " + n + " % " + " you love each  average");
}
else {
     alert ("Your love score is " + n + " % " + "in my opinion just break it up :(");
}
```

Functions:

Creating functions – function get_Milk () {...}

Calling function – get_Milk ();

Comparators:

===	Is equal to
!==	Is not equal to
>	Is greater than
<	Is lesser than
>=	Is greater or equal to
<=	Is lesser or equal to

Difference between '===' & '==':

'==' check only value in variable

'===' checks var value and data type also

Ex. Var a = 1 (number); var b = "1"(string);

If(a == b) → ans is yes;

If(a === b) → ans is no;

JavaScript Arrays & Collections:

- Var.length to get length
- Var[_] to show value at position '_'.
- Var.includes(_) it finds value '_' if found it gives Boolean value.

```
var my_guests = ["Ketan", "Kaushal", "Shruti", "Vaibhav", "Akanksha"];
console.log (my_guests.length);
console.log (my_guests[2]);

if (my_guests.includes(prompt ("Type your name: "))) {
    alert ("Welcome!");
}
else{
    alert ("sorry! but get lost");
}
```

- Var.push(_) it attach '_' value at the end of array.

- Var.pop (_) it removes '_' value from the end.

FizzBuzz Problem:

```
var output = [];
var i = 1;

function fizzBuzz() {
    if ((i % 3 === 0) && (i % 5 === 0)) {
        output.push("FizzBuzz ");
    } else if (i % 3 === 0) {
        output.push("Fizz ");
    } else if (i % 5 === 0) {
        output.push("Buzz ");
    } else {
        output.push(i);
    }
    console.log(output);
    i++;
}
```

99 bottles of beer:

```
var numOfBottles = 99;
var original = numOfBottles;
var bottleWord;
while (numOfBottles >= 0) {

    if (numOfBottles === 1) {
        bottleWord = "bottle";
    } else {
        bottleWord = "bottles";
    }

    if (numOfBottles === 0) {
        console.log("No more bottles of beer on the wall, no more bottles of beer");
        console.log("Go to the store and buy sum more, " + original + " bottles of beer.");
        break;
    }
    console.log(numOfBottles + " " + bottleWord + " of the beer on the wall" + numOfBottles + " "
                + bottleWord + " of beer");
    numOfBottles--;
    console.log("Take one down, pass it arround, " + numOfBottles + " " + bottleWord
                + " of beer on the wall.");
}
```

For loop:

```
For(var i=0; i<n; i++) {...}
```

Fibonacci Series:

```
function fibonacciGenerator (n) {
    var output = [];

    if (n === 1) {
        output = [0];
    }
    else if (n === 2) {
        output = [0,1];
    }
    else {
        output = [0, 1];
        for(var i = 2; i < n; i++) {
            output.push(output[output.length-1]+ output[output.length - 2]);
        }
    }
    return output;
}
```

DOM: Document Object Model

JavaScript work similar to CSS (in case of declaring it - inline/internal/external CSS).

Onload = "…" means when the body of our web site gets loaded up, then whatever JavaScript we place inside these two quotation marks will get carried out.

```
<body onload="alert('Hello');">
  <h1>Hello</h1>
</body>
```

This is inline JS. Avoid this, because it is difficult to debug.

Internal JS:

```
<body>
  <script type="text/javascript">
    alert ("Hello");
  </script>
  <h1>Hello</h1>
</body>
```

External JS:

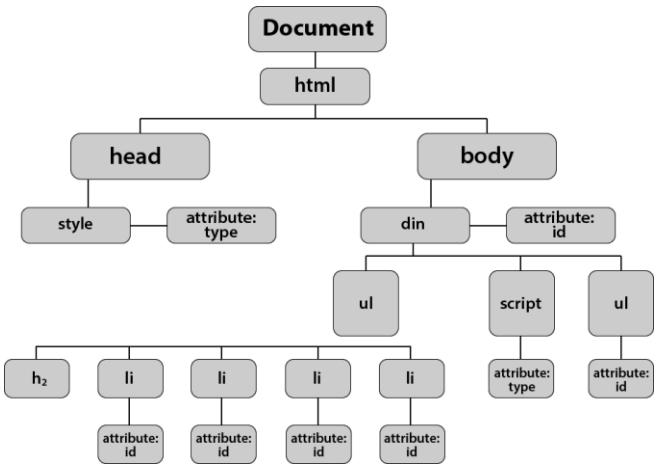
always put the script right at the end just before the closing body tag. benefit of this is, if you have a lot of JavaScript code, which usually is a little bit more time consuming to run, then at least all of the content of your web site will have been loaded up for the users to see, and the parts that they can't perceive, like the script code, happens afterwards, so your website looks like it loads faster

```
<body>
  <script src="index.js" charset="utf-8"></script>
  <h1>Hello</h1>
</body>
```

Dom –

It basically catalogs the web page into individual objects that we can select and manipulate.

It turns each of these elements and their associated data into a tree structure with a whole bunch of objects that you can select and manipulate.



Objects:

Object inside DOM can have properties and methods.

(It is similar to Classes in Java and C++)

Properties – something about object.

Methods – What things object can do.

Ex:- car.color, car.brake(), ……

Set Property:

Ex:- car.numberOfDoors = 0;

Call Method:

Ex:- car.brake();

document.firstElementChild.lastElementChild.firstElementChild

it changes Hello to GOOD BYE

```
> scrpt1 = document.firstElementChild.lastElementChild.lastElementChild;
<   <h1>Hello</h1>
> scrpt1.innerHTML = "GOOD BYE";
< 'GOOD BYE'
```

get property: object.property;

set property: object.property = value;

Method is a function that object can do something.

```
> document.lastElementChild.lastElementChild.lastElementChild.lastElementChild.innerHTML="Ninad";
< 'Ninad'
> document.lastElementChild.lastElementChild.lastElementChild.lastElementChild;
<  ▼<li class="list">
     ::marker
     "Ninad"
   </li>
```

- document returns entire HTML document.
- document.firstElementChild is HTML tag.
- ….innerHTML changes content of element (Ex:- <h1>……<\h1>)
- ….style.color = "…" it changes color.
- document.querySelector("element") it selects element.
- ….click() to click on any element like checkbox.

- document.getElementsByTagName("element") It is same as querySelector. But it returns all elements in an array list. (use .length to get length of array)

If you want to select particular element then,

document.getElementsByTagName("element")[2]

Search elements by ...

- Tag name -

it gives how many li tags are present in file. to combine element + class. It gives array.

```
document.getElementsByTagName("li")[2].innerHTML = "SAMSUNG";
'SAMSUNG'
```

- Class – getElementsByClass it is plural therefore returns all elements. (in an array)

```
> document.getElementsByClassName("btn")[0].style.color = "yellow";
< 'yellow'
```

- ID – getElementByID it is single therefore returns single element. (cause ID is single).

```
> document.getElementById("title").innerHTML = "Anneonghaseyo!";
< 'Anneonghaseyo!'
```

- Query selectors –

Selectors are (in CSS) tags before {…}.

Ex. Body {…}, .container {…}, #title {…}

```
> document.querySelector("h1");
<    <h1 id="title">Anneonghaseyo!</h1>
> document.querySelector(".btn");
<    <button class="btn" style=":active color:red;">Click Me</button>
> document.querySelector("#title");
<    <h1 id="title">Anneonghaseyo!</h1>
```

when you're combining selectors, so things that occur in the same element then there are no spaces. In query selectors we can able to put tags, Id as well as class in it.

```
> document.getElementsByTagName("li")[2].style.color="purple";
< 'purple'
```

```
> document.getElementsByTagName("li").length;
> document.querySelector("li.item");
```

```
> document.querySelector("li a");
<    <a href="https://www.google.com">Google</a>
> document.querySelector("li.list");
<  ▶ <li class="list">…</li>
```

When one selector matches all then only first element display. To get all use this ⬇

- Query selector All –

```
> document.querySelectorAll("#li .list");
< ▶ NodeList(3) [li.list, li.list, li.list]
> document.querySelectorAll("#li .list")[2];
<  ▶ <li class="list">…</li>
```

It returns all elements in form of array. To access individual use …[_].

Note that HTML is for structure, CSS for style & JS for behaviour, but we can change styles by JS also using .styles

Use JS to change behaviour of style not change entire style using JS (it's work of css, code it separately in CSS) How? ⬇

Create and remove CSS class using JS –

```
> document.querySelector("button").classList.add("invisible");
< undefined
> document.querySelector("button").classList.remove("invisible");
< undefined
```

Toggle: if the class invisible is already applied then remove it, and if it's not applied then apply it.

```
> document.querySelector("button").classList.toggle("invisible");
< true
> document.querySelector("button").classList.toggle("invisible");
< false
> document.querySelector("button").classList.toggle("invisible");
< true
```

InnerHTML VS textContent

InnerHTML gives you inner content with HTML tags

Ex – h1-strong-hello-\strong-h1 then it gives Hello

```
> document.querySelector("h1").innerHTML = "<em>Annyeoung</em>";
< '<em>Annyeoung</em>'
```

textContent just give you a text. Ex – Hello

```
> document.querySelector("h1").innerHTML;
< '<strong>Hello</strong>'
> document.querySelector("h1").textContent;
< 'Hello'
> document.querySelector("h1").innerHTML = "<em>Tech Giants</em>";
< '<em>Tech Giants</em>'
> document.querySelector("h1").textContent;
< 'Tech Giants'
> document.querySelector("h1").innerHTML;
< '<em>Tech Giants</em>'
```

Manupulating HTML elements Attribute:

```
> document.querySelector("a");
<   <a href="https://www.google.com">Google</a>
> document.querySelector("a").attributes;
< ▶ NamedNodeMap {0: href, href: href, length: 1}
> document.querySelector("a").getAttribute("href");
< 'https://www.google.com'
> document.querySelector("a").setAttribute("href", "https://www.bing.com");
< undefined
```

Set Attribute:

Requires two parameter (attribute, new_value).

Advance JS and DOM Manipulation

Event Listeners:

The addEventListener() method of the EventTarget interface sets up a function that will be called whenever the specified event is delivered to the target. MDN Docs→

```
document.querySelector("button").addEventListener("click",handleclick);

function handleclick() {
  alert ("just cliked");
}
```

Here you aren't using handleClick() (() without this our function waits for call) if we add this then it directly run this line without any click.to avoid this we use anonymous function.

Anonymous function:

click on h1 tag code is run.

```
var numOfButtons = document.querySelectorAll(".drum").length;
for (var i = 0; i < numOfButtons; i++) {
  document.querySelectorAll(".drum")[i].addEventListener("click", function() {
    alert("I got Clicked!");
  })
}
```

Example: Wikipidea (JavaScript)

Here $0 is shortcut of H1 tag speicified in JavaScript Wikipedia's code.

$0.addEventListener (input1, input2)

Here input1 is specifying what event it should listen to.

Input 2 specifying what it should do once that event gets detected.

Higher order functions:

These types of functions which can *take other functions as inputs* are called higher order functions.

Why we input a function in Event Listeners?

```
function add(num1, num2) {
    return num1 + num2;
}
function mult(num1, num2) {
    return num1 * num2;
}

function calc (num1,num2,operator) {
    return operator (num1, num2);
}
```

Ex-event Listeners

```
> function add(num1, num2) {
      return num1 + num2;
  };
< undefined
> function mult(num1, num2) {
      return num1 * num2;
  };

  function calc (num1,num2,operator) {
      return operator (num1, num2);
  };

< undefined
> calc (5,6,add);
< 11
> calc(5,6,mult);
< 30
```

Whenever you pass a function as an argument, remember not to use parenthesis i.e. ().

Ex.

```
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}

function myCalculator(num1, num2, myCallback) {
  let sum = num1 + num2;
  myCallback(sum);
}

myCalculator(5, 5, myDisplayer);
```

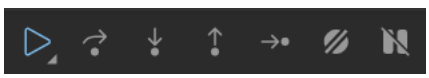
Right: myCalculator(5, 5, myDisplayer);

Wrong: ~~myCalculator(5, 5, myDisplayer());~~

Debugger:

To enter in debug mode write debugger; in console [shift+enter] and write your function and enter.

It display what happening in each step.

Controls: 

Project - Drum Kit Related:

How to add Audio in JavaScript ?

`const audioElement = new Audio('Sound-src');` Learn More→

How to add background image using CSS?

`background-image: url("img/src")` Learn More→

Event listener for keyboard:

```
document.addEventListener("keydown", function(event) {
  console.log(event);
});
```

Here, event is used to get all info about what event happened. This function is not call by us but it is call by the object that experienced the click. (you can use any word instead of 'event' it does exactly same thing)

Ex-

```
KeyboardEvent {isTrusted: true, key: 'b', code: 'KeyB', location: 0, ctrlKey: false, …}
  isTrusted: true
  altKey: false
  bubbles: true
  cancelBubble: false
  cancelable: true
  charCode: 0
  code: "KeyB"
  composed: true
  ctrlKey: false
  currentTarget: null
  defaultPrevented: false
  detail: 0
```

This:

This is basically the identity of the button that triggered the event listener.

Ex- Drum Project

we've attached a listener to our button and when the user clicks on the button the listener gets triggered, and the function that was linked to the event listener gets called.

```
for (var i = 0; i < numOfButtons; i++) {
  document.querySelectorAll(".drum")[i].addEventListener("click", function() {

  })
```

In this code we uses for loop to add event listener to all buttons. So we don’t have to write many lines.

```
for (var i = 0; i < numOfButtons; i++) {
  document.querySelectorAll(".drum")[i].addEventListener("click", function() {
    console.log(this.innerHTML);
  })
```

This code returns which button is pressed (Ex: w, s, a,..)

Objects in JavaScript:

Properties	Bell Boy1	Bell Boy2	Bell Boy3	Bell Boy4
Name:				
Age:				
Has work Permit:				
Languages				

Objects are like Exel sheet.

Watch this for good Explanation: Click → lec.166

How to create Object?

Here Object is housekeeper1

```
var houseKeeper1 = {
    yearsOfExperience: 12,
    name: "Jane",
    cleningRepertoire: ["bathroom", "bedroom", "lobby"]
}
```

How to access:

Housekeeper1.name, ……

Disadvantage is that you've to use same line of code again and again for all housekeepers

So we use Constructor to create different objects.

Constructor in JavaScript:

First letter has to be capitalized

How to create?

```
function HouseKeeper (yearsOfExperiance, name, cleaningRepertoire){
    this.yearsOfExperience = yearsOfExperiance;
    this.name = name;
    this.cleaningRepertoire = cleaningRepertoire;
}
```

Initialization:

```
> var housekeeper1 = new HouseKeeper(12, "Jane", ["bathroom", "lobby", "bedroom"]);
< undefined
> housekeeper1.name;
< 'Jane'
```

We initialized and gave details of housekeeper1 in line no.1.

Objects, their Methods and the Dot Notation

Methods: if we want our object also have an associate function (for behavior) we can write function in constructor.

How to create?

```
function HouseKeeper (yearsOfExperience, name, cleaningRepertoire){
    this.yearsOfExperience = yearsOfExperience;
    this.name = name;
    this.cleaningRepertoire = cleaningRepertoire;
    this.clean = function () {
        alert("Cleaning in progress");
    }
}
```

How to access?

```
> var houseKeeper1 = new HouseKeeper(12, "Anna", ["Keeping"]);
< undefined
> houseKeeper1.clean();
< undefined
```

AddEventListener (background code/behind the scene with event):

```
> function anotherAddEventListener (typeOfEvent, callBack) {
      //Detect Event Code
  var eventThatHappened = {
      eventType: "Keypress",
      key: "p",
      durationOfKeypress: 2
  }

  if (eventThatHappened.eventType === typeOfEvent) {
      callBack(eventThatHappened);
  }
  }
< undefined
> anotherAddEventListener("Keypress", function(event) {
      console.log(event);
  });
    ▶ {eventType: 'Keypress', key: 'p', durationOfKeypress: 2}
```

Here, callback function returns value of eventThatHappened.

Window timeout function():

```
1 var bellBoy = {
2     name: "timm",
3     age: 19,
4     hasWorkPermit: true,
5     languages: ["french, english"]
6 }
```

```
> bellBoy
< ▶ {name: 'timm', age: 19, hasWorkPermit: true, languages: Array(1)}
> bellBoy.name
< 'timm'
```

```
var size = document.querySelectorAll(".drum").length;
for (var i = 0; i < size; i++) {
    document.querySelectorAll(".drum")[i].addEventListener("click", function() {
    this.style.color = "white"
  });
}
```

Constructor function:

what's special about constructor functions is that the names have to be capitalized

```
function HouseKeeper (yearsOfExperience, name, cleaningRepertoire) {
    this.yearsOfExperience = yearsOfExperience;
    this.name = name;
    this.cleaningRepertoire = cleaningRepertoire;
}
```

```
> var houseKeeper1 = new HouseKeeper (4, "Tom", ["lobby","bathroom"]);
< undefined
> console.log(houseKeeper1.name);
  Tom
```