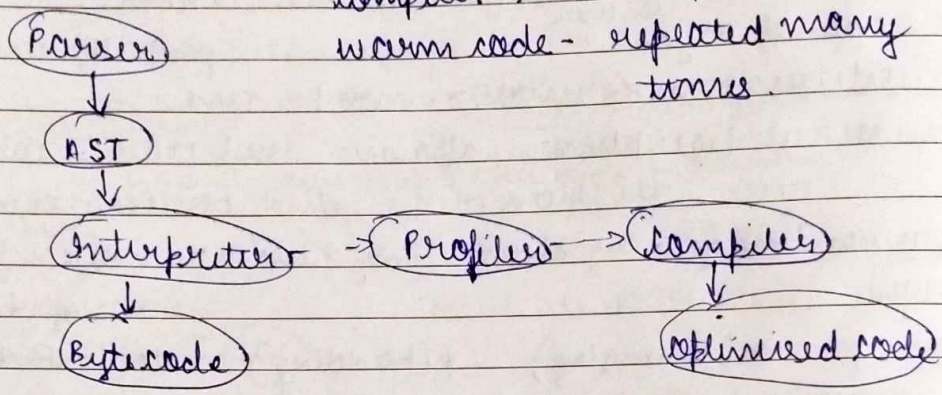
Page No.
Date

08/10/22 JavaScript for beginners :-

* Nouns - HTML, verb - JS. OOP.
* Mozilla - spider monkey (engine).
* Google chrome - weird engine (c++ language).
* JS - re eight engine.

JIT compiler - divides code into tokens.
just in time
compiler > interpreter
warm code - repeated many times.



* History :- JS - Brendan Eich - 1995 (build in 10 days)
* Tools - (i) Text editor - VS code, sublime text, Atom, notepad etc.,
(ii) Browser - All browsers come with an inbuilt JS engine.
* ECMA : European computer Manufacturers association - standardization of JS. new changes are reviewed.
* 3 ways to write JS :- console.log("Hello");
print statement
(i) console (Inspect - Ctrl + Shift + i) - console.log
(ii) Script tag
(iii) External js file.

(ii) <script tag> :-
<script> console.log("Hello world!"); </script> - incorporated in html file itself.
but is not effective because of separation of concerns
Ex: <script type = "text/javascript"> ___ </script>

(iii) external file: most effective way. .js file
Reusability

script tag can be incorporated in head / body. If we want to render the js file before the html page is loaded we can write it in head section. If we want it to be loaded after the page is loaded we can include in the body section.

* comments :- (i) Single line :- // .... } increases the
(ii) Multi line :- /* .... readable of the
.... */ code

* JS properties - (i) Dynamically type - doesn't need to be explicitly specify the data type of var.
(ii) weakly typed - type coercion is not allowed in js.
→ when we are comparing 2 values of different types the one type will force other to change it type as well so that comparison can be made possible.
=== - can stop coercion.

* variables :- name of memory location where data is stored
Syntax :- var varname = val;
→ can begin with a letter, $ or _ (underscore).
Ex: $address1, apple, _name. (valid).
*rno#, @email (invalid).

* conditional statements - if else

```
if (cond1) {
}
else if (cond2) {
}
else {
}
```

Ex:
```
var a=5, b=6;
if (a+b<11)
    console.log("less than 11");
else if (a+b>11)
    console.log("more than 11");
else
    console.log("equal");
```

* Switch - replace multiple if else if statement.

Syntax:
```
switch (expression) {
    case val: ...;
        break; -> break is optional.
    :
    default: ...; -> optional.
}
```

Ex:
```
var day = "sun";
switch (day) {
case "mon": console.log("Today is monday");
        break;
case "tues": console.log("Today is tuesday");
        break;
case "wed": console.log("Today is wednesday");
        break;
case "thurs": console.log("Today is Thursday");
        break;
case "fri": console.log("Today is Friday");
        break;
case "sat": console.log("Today is saturday");
        break;
case "sun": console.log("Today is sunday");
        break;
}
```

* Loops :- To do a repeated work until a specific condn remains true.

(i) for - used when we know the specified no of times the code should execute.

Syntax:- for(initialization; termination; updation){
}

Ex:- for(val i=0; i<5; i++)
console.log("current value of i:"+i);

(ii) for each :- let fruits = ['apple', 'peach', 'orange']
fruits.forEach(item => console.log(item));

(iii) for of :- for arrays & strings
for(items in fruits){
console.log(item);
}

(iv) for in :- gives the index of the item

(v) while :- used when we know the specified expression depending on whose value the code should execute.
while(expression){
}

Ex: var i=0
while(i<5){ console.log("current value of i:"+i);
i++; }

(vi) do while :- at least code must execute once irrespective of the condition dr=10
dr++; 10 11 ++ first assigns & increment.

* Operators :- (i) Unary :-
→ Increment
→ Decrement — postfix (Ex: a++) , prefix (Ex: ++a)

(ii) Arithmetic :-
add, sub, multiply, divide, modulus & exponent.

(iii) Shift :- bit wise shift is performed.
left & right shift (<< , >>)
[Left shift - multiply
Right shift - divide]

(iv) Relational :- <, >, <=, >=, ==, !=
== → allow type coercion. === doesn't allow coercion.

NAN == NAN ⇒ false
NAN === NAN ⇒ false
+0 == -0 ⇒ true
+0 === -0 ⇒ true
over come object is (+0, -0) false

* type coercion:- === can stop coercion.
* Bitwise :- & - and, | - or, ^ - xor ~ - not
xor - which returns 1 if both bits are different else 0.

0	0	0
0	1	1
1	0	1
1	1	0

XOR
bitwise xor - ^.

* Logical :- checks for the validity of condition & specifies the course of action to be taken.
&& - logical and || - OR ! - not.

* assignment & ternary operators :-
= > assignment operator
ternary ? :

* Hoisting :- JS engine allocates memory to the fun and var during the creation phase before execution.
fun is full hoisted, but for var, only left part is hoisted.
stop's hoisting
let, const, (sum) will not get hoisted as hoisting takes place when the 1st word is var / fun.

```
var foodthought = function () {
  console.log("original choice" + favouriteFood);
  var favouriteFood = "sushi";
  console.log("new choice" + favouriteFood);
}
foodthought();
```

o/p original choice undefined
new choice sushi

is not an error like other programming language.

let & const :- It enables block level scope → let
const is used to declare the variables whose value is not going to change once defined

let within the block, is its scope.

* Data Types :- dynamically typed

Primitive	Non primitive
→ Number (int, float)	→ Object (key value pairs) its type.
→ String	var fruits = {
→ Boolean	'apple': 'red',
→ null & undefined	'watermelon': 'green',
	'mango': 'yellow'
	};

undefined : var a;
a.

fruits ['apple']; → "red".

var age = 20;
var a3 = 'Hello' + age;
console.log(a3);

typeof() - returns its type.

(1) → true
(-1) → true
(3.5) → true
("0") → true

(0) → false
(null) → false
(undefined) → false
(NaN) → false
("") → false

Null : var b = null;
b
null.

→ Symbol - create unique properties of objects.

let sym1 = Symbol('foo');
let sym2 = Symbol('foo');
sym1 = = = sym2.
false.

→ objects can be created in 2 ways:-
(i) object literal (ii) new keyword.

var obj = { name: "faizan", roll no: 10, sing:
sing: function {
console.log ('' + this.name + sings');
}
};
console.log (obj.name);
obj.sing();
console.log (obj["roll no"]);

JS object creation - new keyword.

let animal = new Object();

Object comparison :- JSON.stringify(cat) === JSON.stringify(mouse) → true

→ Arrays :- It is used to store ordered data together. []

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

console.log(arr[2]); //3.

var arr1 = [2, 34, 5, 6, , 47];

console.log(arr1[4]); //undefined

let arr = new Array(23, 'cat', new Object());

typeof(arr) ⇒ "object".

→ Array methods :- Push and pop arr.push(3); → at the end.

→ unshift & shift :- add and delete elements from the front respectively.

unshift - add.
shift - delete

→ splice :- to add new elements in the array at the specified position → inclusive.

arr.splice(1, 3, "9999"); → adding

1 and 3rd index are included.

arr.splice(1,3) → remove elements.

→ slice :- It can help to create new array from the existing array.

let arr1 = arr.slice(1);

In slice both the indexes are not included/i.e, inclusive.

1st arg inclusive last arg exclusive.

→ Array printing :- for, for each, for in, for of.

for(let i=0; i<arr.length; i++)

console.log(arr[i] + " ");

* Func :- code snippet that generally perform

same operation.
→ helps in modularisation of code
→ enhances readability, reusability.
→ fun can be created using 2 ways

using fun keyword

```
ex: fun happy(){
    console.log("I'm grateful");
}
happy(); → can be called before declared.
```

Fun expression.

```
ex:
var faith = function(){
    console.log("hope");
}
faith();
```

Fun are hoisted completely.

* IIFE - Immediately invoked function expression.
⇒ It is not hoisted by JS engine.
→ NO explicit calling of IIFE is required.

```
ex: Syntax: (function(){
    })();
(function(a,b){
    console.log("answer is:"+(a+b));})(10,5);
o/p: answer is: 15.
```

→ methods of fun: call, Bind & Apply.
call & apply can be used to borrow methods of another object.

```
let animal = {
    name: "dog";
    eat(a, b){
        console.log(this.name + " is eating" + a + " " + b);
    };
};
let human = {
    name: "Ravi";
};
animal.eat(5, 'bones');
animal.eat.call(human, 10, 'chips');
                ↑ object
animal.eat.apply(human, 10, 'chips')
```

error

animal.eat.apply(human, [10, 'chips']);

pass as an array.

Page No.
Date

→ bind - returns the fun definition that can be used later.

let human_eat = animal.eat.bind(human);
human_eat(5, 'apples');
// Ravi is eating 5 apples

→ current fun

→ This keyword :- this is a object whose property is the function.

→ Arrow function :- to make lexically bound, we can use arrow functions.

var anotherFunc = () => {
  console.log('b', this);
};

→ .

→ Higher order fun :- returns fun as argument/ taking fun as argument.

Ex :- let Interval
fun printIt() {
  console.log("hi");
}
setInterval(printIt, 1000);
clearInterval(1)

function canVote(age){
  if (age >= 18)
    return true;
  else
    return false;
};

=>

return age >= 18.

or

function canDrive(age){
  return age >= 16;
};
function canMarry(age){
  return age >= 21;
};

optimal code for the above code

functions a.

```
function age_req (x_age){
    return function(age){
        return age >= x_age;
    };
};
let can_vote = age_req(18);
can_vote(27);
let can_Drive = age_req(16);
can_Drive(27);
let can_Marry = age_req(21);
can_Marry(27);
```

* OOPs - object oriented programming.
  4 pillars - encapsulation, abstraction, inheritance, polymorphism.

1. encapsulation - wrapping of data and functions

```
class student {
    constructor(rollno, name){
        this.rollno = rollno;
        this.name = name;
    }
    attendance(){
        console.log(this.name + "is present.");
    }
}
```

→ Object :- physical entity of class, it is a class's instance
new keyword - allocate memory in Heap

→ constructor - spl fnⁿ used to initialise the object
Default constructor is implicitly passed by the compiler.

* Abstraction: Hiding of unnecessary information
interface & abstract classes are there in java but
they are not in JS but can be

implemented by prototype.

→ Prototypical inheritance: easy inheritance and all
are object to access properties of some other object
__proto__ → is used to get and set prototype.
prototypeOf().

ram.__proto__ ... __proto__
refers to array refers to array
prototype object prototype.

ram.__proto__.__proto__.__proto__
undefined // because there is nothing after object

→ let father = {
name: 'jhon'};
let ram: object.create(father);
console.log(father.isPrototypeOf(ram)); => o/p: true

child.__proto__ = parent;

var parent = {
name: "father",
sing() {
console.log("singing");
},
eat: function {
console.log("eating")
},
drink: () => {
console.log("drinking")
}
};

var child = {
name: "ram",
eat() {
console.log("eating")
}
};
child.__proto__ =
parent
for (let p in child) {
console.log(p +
" " + child.ram
ownProperty(p));

* Inheritance -
→ In JS it can have only 1 constructor.
→ extends keyword is used to inherit from another class.
→ constructor is not inherited in classical inheritance but can be invoked by child class using super keyword.
→ super keyword is used to invoke parent class constructor and method.

```
ex. class Fruits {
      constructor() {
        this.no_of = 2
        console.log("parent cons");
      }
    }
    class Apple extends Fruits {
      constructor() { super(); }
      console.log("child cons");
    }
```

let obj = new Apple();

o/p: parent cons
child cons

Always 1st parent class constructor will be called and then the child class cons will be called.

→ may / may not be mutable.

* classical vs prototypal inheritance :
  - immutable - can't be changed at runtime
  - multiple inheritance is a difficult task.
  - objects can be easily inherited from multiple objects as only extension of object is required.

4. Polymorphism :- many forms.

overloading overriding

→ JS doesn't support classical method overloading
→ It will allow to have same fun name but it will consider only the last implementation of that function.

→

* Method overloading :- JS doesn't allow this but takes the latest implementation of the function. (compile time polymorphism).

* Method overriding :- This is runtime polymorphism

```
class parent {
    live() {
    console.log("Lives in Iran");
    }
}
class child extends parent {
    live() {
    console.log("Lives in India");
    }
}
```

```
let obj = new child();
obj.live();
```

o/p :- Lives in India.

* Exceptional handling :- abnormal cond" that may happen at runtime and disturbs the normal flow of the program.

Keywords :- try, catch, throw, finally.

→ Exception handling ensures normal execution of our program.

1. try - code that might throw an error.

```
try {
    console.log(a);
}
catch(error) { console.log("we got an error -->"+error)
}
```

→ Don't keep any other statement that might throw an exception.

→ From ES10 the parameter in catch block is Optional.

→ try{}{}
catch{}{}

→ throw Keyword :- we can define our custom message.

```
function a(age) {
  try {
    if (age < 16)
      console.
```

```
    else {
      console.log("You can vote");
    }
  a(14);
// Error: You are under age.
```

```
function a(age) {
  if (age < 16) {
    try {
      throw new Error("You are under age");
    }
    catch(error) {
      console.log(error);
    }
```

* Finally:- It will anyhow get executed. even if catch is not included.

* DOM - Document Object Model.
  → when browser parses the HTML code, it converts it into document that will contain objects corresponding to each element in HTML doc known as nodes.
  → DOM provides different functionalities which can be used to access and manipulate the webpage.
  → 2 main objects { document - entry point of the doc. (console.dir(document))
    window

document = console.log(document.domain | URL | body | head | title | all) all[3];
  ↳ all elements in html page.
  ↳ global object;

* window :- current tab / browser
  ↳ new window
  ↳ alert(), prompt()
  window.setInterval();
  window.open("", "", width = 100, height = 100);
  window.print()
  window.open("https://google.com");

var age = prompt("Please enter your age:");

if (age > 20)
alert("Great you are a valid user/visitor");
else
alert("Try again");

This page says
please enter age

* Element, text and Attribute nodes & DOM contains the element is a tree-like structure where the HTML node forms the root of the tree.
→ All text is also converted and they form a text node.
→ All the attributes create attribute nodes.
window → object → ele → text → attribute

→ Accessing different DOM elements.
(i) document.getElementById()
(ii) document.getElementsByClassName()
(iii) document.getElementsByTagName()
(iv) document.querySelector().
(v) document.querySelectorAll().

<h2 id = "header">
<p>
var head = getElementBy Id(header).
console.dir(head).

* getElementsByClassName() - Elements ✗.
var subHeading = document.getElementByClassName('sub-heading')
Ex: <ul> class Hobbies
<li class = "hobby">cricket
<li class = "hobby"> chess
<li class = "hobby"> carrom

var list_of_hobbies = getElementsByClassName("hobby");
console.log(list_of_hobbies);

* By Tag Name :- <ui> <head> <h1>.
var subHeading = document.getElementsBy TagName("h2");

* querySelector():- "#" for ID, "." for class,
"li" - nth - of - type, h2 ul etc.
It just returns the first matched element.

var a = document.querySelector(".hobby");

* manipulating style :- DOM object style property is one way to manipulate an HTML element's style
eg var mainHeading = document.getElementById('main-heading');
mainHeading.style.color = 'blue';

* manipulating text and content :-
Style syntax :- mainHeading.textContent = "GoodBye";
This method can be used to alter the text but only works in plain text, no modifiers like I will be preserved
1. D. textContent - displays only the content.
1. D. innerHTML - displays content and inner style elements as well.

* manipulating attributes : can be modified by using getAttribute() and setAttribute().
var link = document.querySelector('a');
link.getAttribute('href');
link.setAttribute('href', 'https://www.yahoo.com');
1st change to 2nd

* attributes and properties :-
→ attributes are converted into properties when DOM object is created.
→ not every attribute has 1 to 1 mapping to properties eg:- id is mapped to 1 and changes are synced.

var x = document.getElementsByTagName("*");
var len = x.length;
for(var i=0; i<leng; i++){
console.log(x[i].tagName);
}
Prints all elements present.

* DOM events - click, change, mouseover & mouseout.

-> To add a listener, use the addEventListener method through this syntax :- element.addEventListener('type', function to call);

```
var button = document.querySelector("button");
Ex <h1 onclick = "change(this)">click on this text </h1>.
  function change(id){
    id.innerHTML = "clicked";
  }
```

If we want to change the color of each individual <li>, we can use a for loop :-

```
var lis = document.querySelectorAll('li');
for(var i=0; i<lis.length; i++){
  lis[i].addEventListener('click', function(){
    this.style.color = 'pink';
  });
}
```

* Closure :- when fun^n gets called it is popped out of the stack but still all the arguments are stored in closure which are linked in sub functions. can be done by nesting functions

closure

```
function parent(){
  console.log("hello");
  function child(){
    console.log("Bye");
  }
  return child;
}
```

```
function parent(){
  var x = 10;
  function child(){
    console.log(x+10);
  }
  return child;
}
```

not self contained

* Advanced function :- currying - transformation of fun^n that translates a fun^n from callable as f(a,b,c) into callable as f(a)(b)(c).

-> It doesn't call a fun^n. It just transforms it.

let multiplyBy7 = curriedMultiply(7);
multiplyBy7(2)
O/P: 14

let curriedMultiply (a) = (b) => a * b;

Closure

ex: let add = function(x){
return function(y){
console.log(x+y);
}
}
let addby5 = add(5);
addby5(2);
addby5(5);
addby5(10);

let an
using bind:
let mul = function(x,y){
console.log(x*y);
}
let mulBy10 = mul.bind(this, 10);
mulBy10(9);
mulBy10(8);
mulBy10(9);

using arrow fun: let mul = (x)=>(y) => console.log(x*y);
let mul10 = mul(10);
mul10(8);
mul10(9);
mul10(10);

* composition: combining multiple simple functions to build a more complicated one. The result of each fun is passed to the next one. ex in math like f(g(x)).
ex: const add = (a,b) => a+b;
const mult = (a,b) => a*b;
add(2, mult(3,5))

* promise: It is an object that may produce a single value sometime in future.
3 state: Fulfilled, Rejected, Pending.
-> Before promise callbacks were used.
ex: complicated nested fun stack.
ex: i. addEventListener("click", submitForm);

ex: ->

Ex:- const promise = new Promise ((resolve, reject) => {
if (true)
resolve ("It worked!");
else
reject ("It broked!");
})
promise.then (result => result + "!");
O/P:- Promise {resolved => "It worked!"}

* Advanced Array Function - map :-
-> To create a copy of the array with some manipulation.
let newarray1 = arr.map (num => num * 2)
newarray1.
(4) [2, 4, 6, 8]
Pure function :- output/result remains same, how many ever times the function is called.
map :- no for, no new array, no side effect and also purity of function is maintained.
-> filter :- used to filter the array as per condition.
Ex:- let arr = [10, 2, 30, 4];
let filarr = arr.filter (num => num > 9);

-> reduce :- can be used to combine filter and map.
let arr = [1, 2, 3, 4];
let reducearr = arr.reduce ((acc, num) => acc + num, 5);
reducearr
O/P:- 15.

Ping Pong Game & Color Picker Game.