DATE / /

Javascript

It is a verb of a webpage and defines all the action to be performed on a webpage

Just in Time Compiler:-
Code divided to tokens
warm code - repeated more than once

Brendan Eich - JS engine, Spider Monkey which is still used by Mozilla

Tools:- ① text editor and browser

ECMA - Association to handle JS

Ways:
- Console - Ctrl+Shift+I
- Script tag
- external JS file

Code:
```
// print statement
console.log("Hello")
console.log("This is my JS Code")
```

O/p:- Hello
This is my JS code

DATE / /

use of script tag

```
<html>
<head>
  <title> Home </title>
</head>
<body>
  <h1> My first JS Code. </h1>
<script>
  console.log("Hello");
</script>
</body>
</html>
```

External file:

```
<script type="text/javascript"
src="home.js"></script>
```

Comments: // or /* ---- */

JS Properties - dynamically typed language - no need to define datatype
+ Only var and const is used for variable declaration
* It is a weakly typed language

DATE / /

Variable - name of memory location where data is stored.

Syntax:- var name = Val;

ex:- var num = 10;
var name = "Faizan";

Naming Convention - letter, $ or _ (first letter)

Conditional statement:-

Syt if(Cond){
}
else if (cond){
}
else if (cond){
}
else{
}

ex:-

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

DATE / /

Switch: - replace multiple if else statement

syn:
```
Switch (expression)
{
    case val: ...;
              break;
    case val: ...;
              break;
    case val: ...;
              val break;
    default: ...;
}
```

eg:-
```
var day = "sun";
Switch (day)
{
    case "sun": console.log("today")
                break;
    case "Tue": console.log("tues")
                break;
    case "sun": console.log("Yes")
                break;
    default: console.log("No")
             break;
}
```

DATE / /

Loops :-

Syn :- for (initialization; termination; updation) {

}

```
for (var i=0; i<5; i++)
    console.log("current val of i" : +i)
```

for each :-

```
fruits : ['a', 'b', 'c']
fruits.forEach (item =>
console.log(item));
```

for of :-

```
fruits = ['a', 'b', 'c']
for (item of fruits) {
    console.log(item);
}
```

for in :-

```
for (item in fruits) {
    console.log(item);
}
```

while loop :-

```
Syn: while (expression) {
}
```

DATE __/__/__

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

do while :-

```
do {
} while (cond);
```

Operators :-

1. Unary Operators :- a++, ++a, a--, --a
2. Arithmetic operators :- +, -, *, /, %, **
3. Shift operators :- a<<b or a>>b
   left - increment, right - decrement
4. Assignment and ternary (? :)

Object.is( ) // === - stops type conversion

```
var a = 2
console.log((a == 2) ? console.log("ok") : console.log("not ok"));
O/P :- ok
```

DATE / /

Hoisting: JS engine allocates memory to the func and var during the Creation phase, before execution.

ex:
```
console.log(a);
var a = 10;
console.log(a);
```
o/p: undefined
10

What actually happens is internally a will be set as undefined hence the same gets printed

⇒ But this should throw an error hence to overcome this instead of var we use let and CONST → this throws an error

Diff b/w let and const

ex:- for let
```
let weather = "sunny";
function change(){
  let weather = "rainy";
  console.log("change:" + weather);
}
change();
console.log("change:" + weather);
```

DATE ___/___/___

o/p: rainy // inside brace get reassigned
sunny // outside brace value is printed

But:-

```
Const weather = "sunny";
function change(){
    weather = "rainy";
    Consol.log(weather);
}
```

o/p:- Throw an error because once a variable defined cannot be reassigned

Data type - Primitive and non primitive

```
var age = 20
Var s3 = 'Alex is ${age}';
Consol.log(s3);
```

Primitive : undefined, Null

Symbol: Creat unique property of objects

Non Primitive Data Type-
(i) objects - key value pair -> {}

DATE ___/___/___

ex:- var f = { 'a': 'x', 'w': 'g', 'm': 'y' };
f['a']; => 'x'

Create objects:-
(i) object literal
(ii) New keyword

(i) Object literal - where {} to create a object in JS

```
let animal = {
    name: "Cat",
    color: "black",
    eat: function() {
        eat() {
            console.log(`${this.name} is eat`);
        }
    }
}
animal.eat()
animal["color"];
```

(ii) New keyword: Create object via Constructor
→ Dot or quote can be used to assign properties of the objects

```
ex:- let animal = new Object();
     animal.name = "Cat";
     animial["color"] = "black";
```

DATE __/__/____

animal.eat = function() {
  console.log(`${this.name} is eating`);
};
animal.eat();
// - Cat is eating

Object Comparision
Object.is(), == and === check for referential equality
→ let animal = new Object();
let cat = animal;
let dog = animal;
cat == dog // same object reference
→ true
let horse = new Object();
cat == horse
→ false
cat == = horse
→ false
Object.is(cat, horse);
→ false

To compare the content! - JSON.stringify
ex JSON.stringify(cat) == = JSON.stringify(horse)
→ true

DATE __/__/____

Arrays: Store ordered data together ([])
→ can have element of different types
ex:- var a = [1, true, 'h'] // Heterogeneous

We can create Array using new keyword
→ let arr = new Array(23, 'Cat', new Object())
>> arr
>> (3) [23, "Cat", {...}]
no of elements

Method - push and pop

a = [1,2,3] a.push(4) => [1,2,3,4]
a.pop() => 4

Inserting and deleting element from first position

let a = [1, 2];
a.unshift(3);
o/p:- (3) [3,1,2]

let a.shift();
>> a
>> (2) [1, 2]

splice - add new element in array from specified to and from index

DATE __/__/__

```
let arr=[1,2,3,4,5];
arr.splice(1,3,"Hello");
arr
>> (3)[1,"Hello",5]
```

Slice - Creates new array from existing array

```
let a1=arr.slice(1);
a1
>> (4)[2,3,4,5]
```

```
let a=[1,2,3,4,5]
let a1=a.slice(1,3);
a1
>> (2)[2,3]
```

(3 will not be included)

Array iterators - for, forEach, for in, for of

```
for(let i=0; i<arr.length; i++)
    console.log(arr[i]+" ");
```

```
forEach()
arr.forEach(item=> console.log
    (item+" "));
```

```
for of
for (item of arr)
    console.log(item+" ");
```

DATE / /

Function - snippet performing operation
Created → using function keyword
Function expression

① Using Function Keyword
→ function happy(){
Console.log("I am grateful");
}
// Calling
happy()
o/p :- I am grateful.
→ o/p is def at Compile time

Function expression
ex1 - var faith = function(){
Console.log("hope");
}
faith()
→ o/p is def at runtime

IIFE
(function(){
})();
ex1 - (function(a,b){
co.lo("A:" + (a+b));
})(0,5);

DATE / /

Methods of function : call(), apply(), Bind()

① Call & Apply - borrow method from other function

```
function print() {
    console.log("Hello");
}
print();   // Hello
print.call();  // Hello
print.apply(); // Hello
```

ex1

```
let animal = {
    name: 'animal',
    eat(a,b) { console.log(this.name + " eating " + b + " " + a) }
};
let human = {
    name: 'human'
};
animal.eat('apple', 5);
animal.eat.call(human, 'Orange', 2);
animal.eat.apply(human, ['orange', 2]);
```

used later

// same prog as previous

let human_eat = animal.eat.bind(human)

human_eat(5, 'apple');

This keyword - is a object whose property is function

ex1- obj.func(this)

```
function a() {
    Console.log(this);
}
a();
```

this refers to object that is calling the function

Arrow function - lexically scoped

```
Const obj = {
  name: 'S',
  sing() {
    Console.log('a', this);
    Var anotherfunc = function() {
      Console.log('b', this);
    };
    anotherfunc();
  }
}
obj.sing();
```

DATE __/__/__

o/p :- a { name : "sam", sing : f } } bind
b { window { ——— } }

ex1 with arrow

```
const obj = {
    name : 'Sam',
    sing(){
      console.log('a', this);
      var anotherfun = () => {
        console.log("b", this);
      }
      anotherfun();
    }
}
obj.sing();
```

// arrow always bind

o/p :- a { hi : 5, 5 : f }
b { ——— }

Higher order function - take or return function as argument

ex2

```
function print(){
    console.log("hi");
}
setInterval(print, 1000);
clearInterval - to stop
```

DATE / /

Return fn as arg:

```
function canVote(age)
{
    if(age>=18)
        return true;
    else
        return false;
};
function canDrive(age)
{
    return age>=16;
}
function canMarry(age){
  return age>=21; }
let j
Console.log(canDrive(17));
Console.log(canMarry(22));
Console.log(canVote(19));

function age_req(r_age)
{
    return function(age)
        return age>=r_age
}
```

DATE / /

OOPS :- 4 main pillars are
(i) encapsulation
(ii) Abstraction
(iii) Inheritance
(iv) Polymorphism

(i) Encapsulation - Wrapping object, function, property inside a Container
(ii) Abstraction - hiding unnecessary information that is not required by the user
(iii) Inheritance - reusability of the property from other class
(iv) Polymorphism - Act differently on different input

Encapsulation -

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

DATE ___/___/___

→ allocating memory

var adam = new Student (23, 'Adam');
→ constructor
adam.attendance()
o/p :- Adam is present

★ [class + Constructor = Encapsulation]

Objects - physical entity of class. It is a class instance. new keyword used to create object. when we create the the object constructor is called

- var name = {
  fname = ('Rai', lname = 'pawez')
  }; // literals // directly created object

with new:
var adam = new Student();
object

Constructor - used to allocate memory and initialize object
Syntax - Constructor() { }
new call → Constructor

ex:- class Box {
  Constructor (l) {
    this.l = l;
  }

DATE __/__/__

```
display(){
    console.log(this); // method
        }
}
let obj = new Box(20);
  obj.display();
Box {l: 20}
```

↳ object

Abstraction:- To implement abstract and interface classes we use prototype

Prototypical Inheritance: allow one object to access properties of some other object

* `__proto__` is a property that is used to get and set prototype → reference work on every object — refers to prototype of object

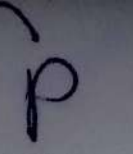
ex- `son.__proto__ = father` // son can access father properties and methods

- `prototypeOf()` is a method provided by Object which is available to every thing or everything in obj in JS

ex `father.prototypeOf(son)` // true

↳ inherited from father

DATE __/__/__



Object.create() -

```
let father = {n: 'f'}
let son = Object.create(father);
father.isPrototypeOf(son); // true

let son1 = {};
father.isPrototypeOf(son1); // false
```

hasOwnProperty() - To check if property is inherited or is its own

```
var par = {
    n: 'p',
    sing: () => console.log(`${name} singing ---`),
    food: () => console.log("This is food").
}

var child = { name: 'child',
food: () => console.log("This is my food").
}

child.__proto__ = parent;
for (let prop in child) { console.log(prop + "---" + child.hasOwnProperty(prop)); });
```

Inheritance - takes place on class level
- extends keyword - inherit from another

DATE __/__/__

class

- constructor is not inherited in class inheritance but can be invoked by child class using super keyword

eg

```
class Fruit {
    constructor(name, color) {
        this.name = name;
        this.color = color;
    }
    growth() {
        console.log(this.color + " fruit growing....");
    }
}

class Apple extends fruit {
    constructor(name, color, type) {
        super(name, color);
        this.type = type;
    }
    eat() {
        console.log(this.name + " eating...");
    }
}

let TastyApple = new Apple('TastyApple', 'Red', 'big');
```

Parent

DATE __/__/____

TastyApple.grow();
TastyApple.eat();
o/p:- Red fruit growing....
TastyApple eating.....

[Super.print()] → allows method of parent

Polymorphism:- poly + morphism (many + forms)
divided [overloading
overriding

Overloading - JS doesnot support classical method overloading
- It will allow you to have same function name but it will consider only the last implementation of the function
- Ex:- class Father {
constructor(name){
this.name = name;
}
display(){ console.log("I am" + this.name); }
display(Vehicle){ console.log("I am driving" + vehicle); }
Var alex = new Father('alex');
alex.display('bike')

The main difference in method overloading in JS and other language is that only last method will be considered.

alex.display() // last implementation

"I am showing undefined"

=> Method overriding - happens at runtime

-> ex1. class Father {
display() { console.log("I am father"); }
addres() { console.log("I live in India"); }
}
class child extends Father {
// child class has its own implementation of display
// it has overridden the display method of Father class
display() { console.log("I am child!"); }

var ram = new child();
ram.addres() // I live in India
ram.display() // I am child!

// Method must have same signature, ie if that method do not have arg then it should also donot contain it.

Exception Handling <- abnormal condition
-> try, catch, throw, finally

try and catch block: try block contain the code which might throw an error
If statement in try throws an exception, it gets caught in a catch block

ex1
```
try {
    console.log(q);
}
catch (error){
    console.log("we got an error");
}
```
o/p -> we got an error

ReferenceError: a is not defined

Throw keyword:- There may be scenarios where JS engine will not throw the exception but application wants to consider it as an exception
→ So throw keyword can be used to throw explicitly an user defined exception.

ex1
```
function isEligible(age){
    if (age < 18)
        try{
            throw new Error("you are not");
        }
        catch(error){
            console.log(error);
        }
    else
        console.log("you are");
}
isEligibleToVote(13)
```

Finally block:

```
try{
    console.log(4);
}
catch(error){
    console.log("we got an error -->"+error);
}
finally{
    console.log("Prog ended");
}
```

o/p:- we got an error "error"
Prog ended

DOM → Document Object Model

* when browser parses the HTML code, it converts it into a document that will contain objects corresponding to each element in HTML doc known as node
* There are two main objects that we deal with in general - document and window
* DOM is not strictly used / tied to browser as there are other tools like nodejs that can parse HTML

Document :-

```
<!DOCTYPE html>
<html>
<head>
<title> Document </title>
```

<script type="text/javascript" src="f.js"></script>
<head>
<body></body>
</html>

console.log(document) => #document (object provides element of html page

console.dir(document) => #document → contains list of item that can be used ie, properties

//to know the domain name
console.log(document.domain); => Domain Name
console.log(document.URL); => Location of file

To change the title
document.title = 12345; → change the title

(or)
document.all[2].textContent = "tt";

Window - refers to current tab or browser
* alert(), prompt() etc.
* It is a global object.
=> window.setInterval();

Resize the window => window.open("", "", width = 100, height = 100);

window.print(); → prints the content of current window

Open other website:-
window.open("https://google.com");

In that editor:-
var age = prompt("Please enter age:"); → user can send it to browser to check a condition
if (age > 20)
alert("Great you are a valid user");
else
alert("Try again");

Element, Text and Attribute nodes:-
→ different method for accessing different DOM element
① document.getElementById() - takes in an ID name and return the one element that matches that ID
* ID is unique and can be associated with one element in entire document
Ex: ID = "main-head"
var mainHeading = document.getElById("main-heading");

② document.getElementsByClassName()-
 Hobbies
 Cricket (<li class="hobby">)
 Rest
 Sp.
 M

To get / access this :-

var head = document.getElementsByClassName

console.log(head);

o/p : properties of that particular element (0: 1

in list)

③ document.getElementsByTagName() - u/s

tag

ex1 var tag = document.getElementsByTa

('p');

o/p : collection of paragraph → para

0 : p → properties of the para

1 : p

2 : p

2] want to access the para :-

tag[0].textContent = "Hello there";

④ document.querySelector() - uses CSS-sty

selector

ex1 '#' for ID, '.' for class, li : nth-of-ty

, h2 ul etc.

→ returns the first matched element

ex1 var para = document.querySelector(

provide the first class that it matches

but what if we want all the elements

with class = Lobby" then

document.querySelectorAll(".hobby");
('li a.special') → li inside (a) and is then it '.special' class.

Manipulating style:

ex1 var I_D = document.getElementById("heading");
⇒ I_D.style.color = 'red';
⇒ I_D.align = "center";
⇒ I_D.style.background = 'blue';

Manipulating text and Content:

① I_D.textContent = 'Header'; } gives same
② I_D.innerHTML = "Head"; } but diff is ↓

ex1 <h2 id="heading"> this is Header </h2>

text content ⇒ but it ignores them
innerHTML ⇒ considers all the inner tags like span

I_D.innerHTML = '<h4> Header tag </h4>'; → is considered but it cannot be done in textContent

Manipulating Attributes:- getAttribute(), setAttribute()

var link = document.querySelector('a');
link.getAttribute('href');
link.setAttribute('href', 'https://www.yahoo.com/');

DOM Events :: click, change, mouseover, mouseout

① To add a listener
element.addEventListener('type', function())
call)

ex1 var button = document.querySelector('button');
button.addEventListener('click', function(){
console.log('The');
});

<body>
<h1 onclick="change(this)">click</h1>
</body>
<script type="text/js" src="first-js"></script>

first-js

change(id){
id.innerHTML = "Clicked";
}

Advanced Array Function :- map — New Arr, No new array, No push, no side effect and also parts of function is maintained. map always return a value.

let newarr = arr.map(num => num**2)
newarr
>> (4) [2, 4, 6, 8]