__/__/__

CSS

↳ Types of CSS
↳ Types of selectors

CSS Properties

Syntax

h1 { color: blue ; font-size : 12px ; }

↑ Selector ↑ property ↑ value

① CSS background

(i) background-color : green
or rgb(200, 20, 10, 5) → opacity

Opacity :→ means transparency.

0.0 – 1.0
↓ ↳ full visible
full transparent.

(ii) background-image : url ("paper.gif");

② CSS Borders

(i) border-style : dotted
dashed
solid
double
groove
ridge
inset
outset
none
hidden

(ii) border-width : 10px
in
pt
cm
em
thick
medium
thin

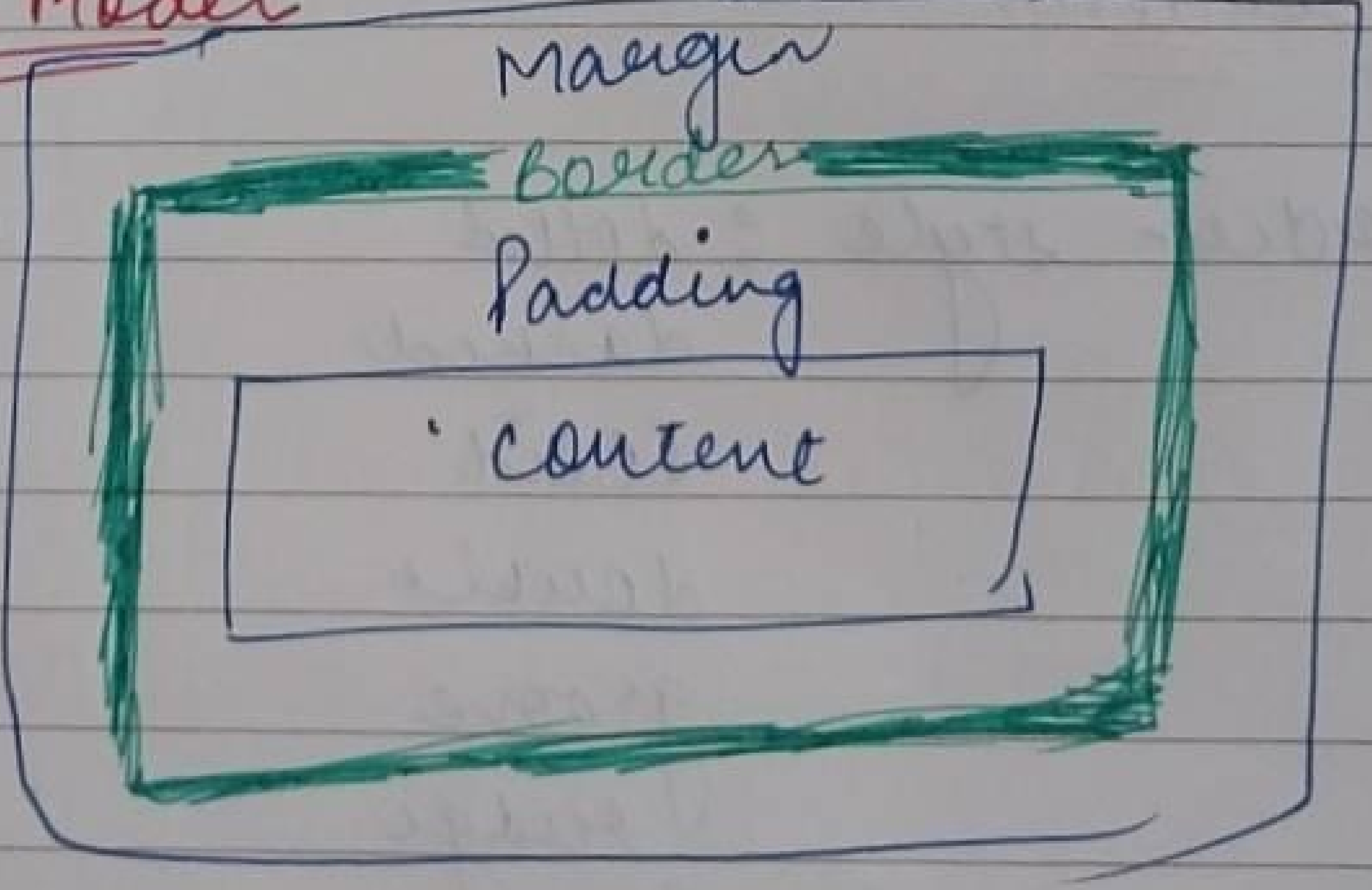
(iii) border-color : "red
#ff0000
rgb(255,0,0)

(iv) for Individual sides

border-top-style : dotted ;
border-right-style : solid ;
border-bottom-style : dotted ;
border-left-style : solid .

(v) border-radius : 5px
(for rounded border)

CSS Box Model



Scanned by TapScanner

__/__/__

example

```
div {
        background-color - light grey;
        width : 300px;
        border : 15px solid green;
        padding : 50px;
        margin : 20px;
    }

<body>
      <div  ≡ ≡ -   </div>
</body>
```

Width and height

Width and height →

```
    Width : 100px;
    height : 100px
```

* Width and height specifies width and height of the content Area.

Total width calculation

```
div {
       width : 320px;
       padding : 10px;
       border : 5px
       margin : 0
    }
```

→ left (10px) + right (10px)

Total width = 320px + 20px + 10px + 0

↳ left border + right border

= 350 px

↳ Margin
↳ Padding
↳ Outline → line drawn outside border.

CSS- Text

(i) color: blue
(ii) text- align : left
right
center
Justify

(ii) text-decoration-line: overline
linethrough
underline

(iii) text-decoration-color
(iv) text-decoration-style: solid
double
dotted
dashed
wavy

(V) text-shadow : 2px 2px ~~red~~ 5px red

horizontal Vertical blur color

CSS-font

(i) font family : "Times New Roman"
"Arial"

(ii) font-weight : normal
bold

Scanned by TapScanner

(iii), font-size: 40px
30px

40px | 16 = 2.5em

30px | 16 = 1.875em

css- links

There are 4 pseudoclasses for links

1. link →
2. visited →
3. hover →
4. active →

```
a: link {

}
```

css- position

1. position : static { According to normal flow }
2. position : relative { relative to its normal position }
3. position : fixed { always stays in the same place even if page is scrolled. }
4. position: absolute { positioned relative to the nearest position ancestor }
5. position : sticky { b/w relative and fixed }

first scrolls and then fixed to one point.

Dropdown

Navigation → Horizontal
→ Vertical

You can create a Navigation Bar using lists

to remove bullets
ul{
list-style-type: none }

- Home
- News
- contact

```
<ul>
<li> <a class="active" href = "default.asp"> Home </a></li>
<li> < ______ > News </a></li>
<li> < ______ > contact </a></li>
```

Horizontal Navigation Bar

(1) li {
display: inline ;
}

in inline property next element will come in same line

(2) li{
float: left
}

li a {
display: ~~block~~ inline;
color: white;
padding: 10px ; }

example in VSC

.active
{ background-color : ~~#~~green }

CSS- dropdowns

Example 1   Hover over me
↑ text will be displayed

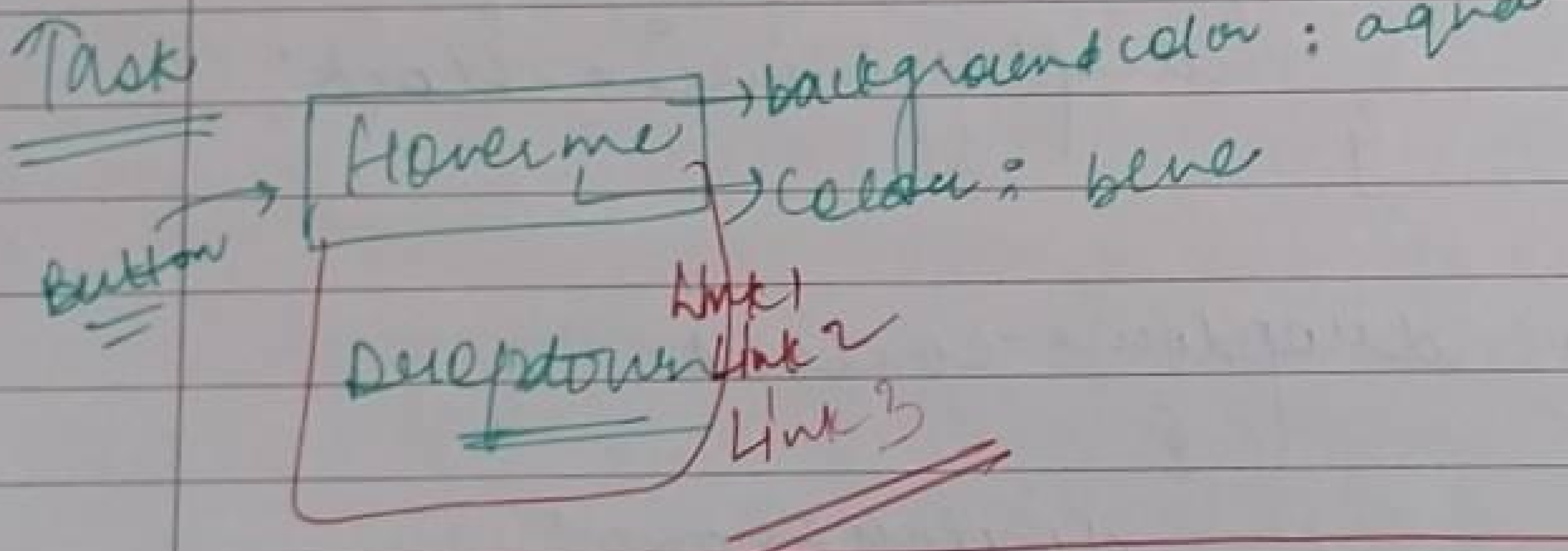
```
<html>
 <head>
 <style>
      .dropdown
2      { position: relative;
         display: inline-block;
      }

3   .dropdown-content
         {
          display: none;
          position: absolute.
          background-color: #f9f9f9;
          min-width: 160px;
          padding: 12px; 16px;
         }
        → hover over this element to show-
4   .dropdown:hover  .dropdown-content
          {
            display: block;
             color: red;
          }
<|style>
</head>
```

```
<body>

(1)   <h2> Hoverable Dropdown </h2>
      <p> Move the mouse over me </p>

      <div class = "dropdown">
        <p> Mouse Over me </p>
         <div class = "dropdown-content">
         <p> Hello World </p>
       </div>
     </div>
    </body>
    </html>
```



Color coding in Hexadecimal (0-255)

RGB
#L_ __ __
Red Green Blue

Decimal to Hexadecimal
$10 \quad\quad 16$

(1) $(255)_{10} \longrightarrow (FF)_{16}$

$16 \,|\, 255$
15 → (15)

(15) (15) F
F

__/__/__

0	$A \rightarrow 10$
1	$B \rightarrow 11$
2	$C \rightarrow 12$
3	$D \rightarrow 13$
4	$E \rightarrow 14$
5	$F \rightarrow 15$
6	
7	
8	
9	
a	

② $(89)_{10} \rightarrow (59)_{16}$

$$\begin{array}{r|l} 16 & 89 \\ \hline & 5 \quad 9 \end{array}$$

③ $(161)_{10} \rightarrow (\quad)_{16}$

$$\begin{array}{r|l} 16 & 161 \\ \hline & \underbrace{10}_{} \quad 9 \end{array}$$

10 1
↓
$(A1)_F$

Hexadecimal to decimal

$(A1)_{16} \rightarrow (\quad)_{10}$

$10 * 16^1 + 1 * 16^0$

$160 + 1$

$= 161$

Block level elements
(takes full line)

<div>
<h1> — <h6>
<p>
<form>

Inline
(doesnot start with
new line)

<a>
<img>

$$\begin{tikzcd} & \text{client side} \\ \text{Javascript} \arrow[ur] \arrow[r] & \text{server-side} \end{tikzcd}$$

Javascript

Javascript was Initially created to "make page live".

Why Javascript is so Important?
JS can be used as client-side and server-side & scripting languages

Server-side and client side Scripting

↳ ~~Nodejs~~
↳ client side Scripting means the browser processes the code instead of a Web-server.
↳ client side scripts are commonly used when we want to validate data before sending it to web browser.
↳ client side Javascript Includes validating input, animation, applying styles, some events (mouse click -)

Server side Scripting enables backend access to database, file system and servers
↳ Server side Javascript is a code running over a server local resources.
example Nodejs
↳ is a open-source and it is nothing but Javascript framework.

__/__/____

↳ Node.JS is an open source, cross-platform JavaScript runtime environment and library for running web applications outside the client's browser.

Frontend Framework

↳ React (React Native & React JS)

↳ AngularJS

↳ VueJS

Backend Framework

↳ nodeJS

Javascript display possibilities

① Using InnerHTML

eg.

```
<body>
<h1> This is a first script </h1>
<p id="demo"> </p>

<Script>
   document.getElementById("demo").
                        innerHTML= "Hello";
</Script>
</body>
</html>
```

② Using document.write

```
<Script>
   document.write("Hello");
</Script>
```

Scanned by TapScanner

__/__/__

③ Using window. alert ( )

```
<html>
<body>
  <script>
    Window. alert (5+6);
  </script>

</body>
</html>
```

④ Using console. log ( )

```
<script>
  Console. log( "Hello")
  </script>
 </body>
  <--
```

Scope :→ determines the accessibility of variables.

① Global scope :→

↳ A variable declared outside a function becomes "Global".

↳ All the scripts and functions on a web page can access it.

e.g var a=10;

Scanned by TapScanner

② Local scope

Function scope | ~~Global~~ Block scope

(Variables declared within a Javascript function, becomes local to the function. Local variable are created when a function starts and deleted when function ends.

→ In ES6, this concept is introduced. Variables declared inside { } cannot be accessed from outside the block.

```
{
    let x = 2;
    const y = 2
}
```

eg
```
function myFunction(param)
{
        Local variable.
    var carName = "volvo"

}
```

Let, Var and const

① let

↳ Variables defined with "let" cannot be redeclared

↳ Variables defined with let have Block scope eg-

1
```
{ let x=2

}
// x cannot be used here.
```

2
```
let x = 10
    //x is 10 here

{ let x= 2;
    //x is 2 here
}

// x is 10 here
```

3
```
{ var x=2
  let x=3    X redeclaring is not allowed.
                 in same block.
}
```

4
```
let x= 3
{
 let x=2   // Allowed.
}
```

_/_/_

② Var

↳ Variables donot have block scope.
↳ You can redeclare variables

①
```
{
  var x = 2
  }
// x = 2 can be used here
```

②
```
var x = 10 ;
{ var x = 2 ;

}
// x is 2 here.
```

③
```
{
 var x = 10 ;
  var x = 2 ;

  }
 // is allowed.
```

③ Const

↳ Const variables cannot be reassigned
↳ Variables defined with "const" have block scope.

e.g.
```
const PI = 3.14
      PI = PI + 10 ;   // error
```

↳ must be assigned a value when they are declared.

const PI = 3.141 ;

JS Hoisting

JS's default behaviour of moving only declarations to the top of the current scope.
In other words, a variable/function can be used before it has been declared

e.g.-

```
x = 5;
console.log(x)
var x;
```

```
x = 5
console.log(x+y)
var y = 7;
```

error ~~[illegible]~~.

(because only declaration are hoisted not initialization)

__/__/____

Functions

JS function is a block of code designed to perform a particular task.
It's defined with 'function' keyword, followed by a name, followed by paranthesis ().

```
function name( parameter1, parameter2)
  {

  }
```

↳ Accessing a function without () will return the function defenition instead of function result.

e.g.-
```
<body>
<p id="abc" >  </p>
<Script >
   function myfunction (p1, p2)
  {
   return p1 * p2;
  }
document.getElementById("abc")
        .innerHTML = myfunction
                          (4, 3);
</script>
```

__/__/__

* Result of function will store in one variable.

```
<body>
<p id="abc"> </p>

<script>
  var x = myfunction (4, 3)
  document.getE ________          = x

  function myFunction(a, b)
     {
       return a*b;
     }

  </script>
</body>
```

* Without () will return a function definition

```
<body>
<p id="abc"> </p>

<script>
      function myfunction (a, b) {
        return a-b

               }

  document.getElementById.innerHTML
                        = myfunction
```

_/_/_

O/P

```
function myFunction (a, b)
{
  return a - b
}
```

* Function as an expression

```
<body>
<p id = "abc" > </p>
<script>
const x = function (a, b) { return
                                a*b };
```

$\overset{\text{no function name}}{}$

```
document.getElementById ("abc").innerHTML
                       = x(4, 3);
```

* That variable can be used as function name.

* self invoking functions

↳ is invoked automatically, without being called.

↳ You have to add paranthesis around the function to indicate that it is a function expression.

```
<body>
   <p id="abc"></p>
   <script>
      (function()
         {
         document.getElementById("abc")
               .innerHTML = "Hello, I called
                              myself";
         })();

      </script>
   </body>
   </html>
```

* Arrow Functions

↳ helps you to write short code

```
<script>
const x = (x, y) => { return x * y };
document. ________ = x(5 * 5)
```

__/__/__

* We can also omit return keyword and the curly brackets if function is a single statement.

⇒ e.g.

```
const x = (x,y) => x*y
```

* Random number generator

↳ using Math.random() method

↳ By default it returns random number from 0 (inclusive) to 1 (exclusive)

e.g.-

```
<script>
document.elementById("abc").innerHTML
                   = Math.random();
</script>
```

e.g.- Return a random number b/w 0 to

```
let x = Math.random() * 10
```

Random whole no. b/w 1 to 10

```
Math.floor((Math.random() * 10
              + 1);
```

Recursion

Task ⇒

```
power (2, 5)
    2* power(2, 4);
    2* 2* power(2, 3);
    2* 2* 2* power(2, 2);
    2* 2* 2* 2* power (2, 1)
    2* 2* 2* 2* 2
```

Solⁿ

```
function power(number, exponent)
  {
   if (exponent === 1)
     {
       return number;
     }
   else
     {
       return number *power(number,
                             exponent-1);
     }

console.log (power (2, 5))
```

_/_/_

(2) factorial program using arrow function

```
const factorial = (n) =>
    {
    if(n===0||n===1)
     {
       return 1;
     }
     else
     {
       return n * factorial(n-1);
     }
  }
console.log(factorial(5))
```

__/__/__

Javascript Array

An array is a special variable, which can hold more than one value

⇒ const cars = ["saab", "volvo", "BMW"]

second method to create an array with "new" keyword ⇒ const cars = new Array("saab", "volvo", "BMW")

Accessing array

Accessing first element → cars [0] ⟶ index number of first element.

Accessing last array element → const a = ["1", "2", "3", "4"]
let b = a[a.length - 1];

Looping Array Elements

using for

```
<script>
const name = ["Amit", "Aakash", "Gaurav"];
let nlen = name. length;
```

___/___/___

```
~~let text = "<ul>";~~
        let text = "<ul>";
for ( let i = 0 ; i < nLen ; i++ )
  {
   text += "[illegible]" + fruits[i] + "<br>";
  }

~~text += "</ul>"~~

document.getElementById ("demo").innerHTML
                                  = text ;

</script>
```

↳ Array. forEach() function

forEach() method calls a function (callback function) once for every array element.

Passed as an argument to another function

```
<script>
      let text = " ";
const  a = ["1", "2", "3", "4"]

    a . forEach (myFunction) ;
       text += "~~[illegible]~~" ;
    document. getElementById ("demo"). inner
                                  = text ;

    function   myFunction(value)
      {
        text += " value + "<br>";
      }

   </script>
```

Scanned by TapScanner

__/__/____

Array Methods

① .tostring() converts an array to a string of array values (comma separated)

eg - <script>

```
const a = ["1", "2", "3"];
document. ______________ = fruits.toString()
</script>
```

O/P

```
1, 2, 3
```

② join() → same as .tostring() but you can specify a separator.

eg -

```
~~const~~ a = ["1", "2", "3"]
document. ______________ = fruits.join("*");
```

③ pop() → removes last array element

```
<script>

const a = ["1", "2", "3"]
```

_/_/__

```
a.pop();
console.log(a)                    O/P   1,2
```

④ push() → adds a new element to an array.

```
const a = ["1", "2", "3",]
    a.push("4");
```

⑤ shift() → removes first array element and shifts all other elements to lower Index.

e.g.
```
const a = ______
 a.shift();
                                  O/P    2,3
```

⑥ unshift() → adds a new element to an array and "unshifts" older elements.

```
const  a = ["1", "2", "3", "4"]

     a. unshift ("5");
```

⑦ Splice() → method can be used to add new items to an array.

__/__/____

e.g. - const fruits = ["Apple", "Mango", "Kiwi"]
fruits.splice(2, 0, "Lemon", "Orange"]

2 → position where new elements should be added.

0 → How many elements should be removed.

O/P [- - - - , 'Lemon', 'Orange']

(8) slice() → slice out a piece of an array into new array.

const fruits = ["Apple", "Mango", "lemon", "~~Mango~~ Cherry"]

cont fruit1 = fruits.slice(1);
console.log(fruit1);

O/P Mango, lemon, Cherry

or fruits.slice(1, 2)

1 → starting

2 → ending.

Sorting an array

sort() method works well for strings not for numbers.

① (40,100,) → 40,100 ② 100 1 → 1,100 ③ 100,5 ④ 100 25 ⑤ 100,10 → 100

Ascending

```
<script>
const points = [40, 100, 1, 5, 25, 10 ]
      console.log( points)
points.sort(function (a, b) { return a-b });

      console.log(points);

<script>
```

eg-
⇒ for (40,100)   sort() method calls compare function.

sort { 40-100 = -60)
40 comes first

If result is -ve a<b (comes first)
If result is +ve a > b
If result is 0 a = b

Random Image Generator

```
<Script>

   function getRandomImage()
   {
      var randomImage = new Array();

randomImage[0] = " image link ";
 randomImage[1] = " ______ ";
             [2]      __
             [3]      __
             [4]      __
             [5]
__/__/__

var number = Math.floor (Math.random()
                 * randomImage.
                        lengeh);

return document.getElementById("result").innerHTML
              += "<img src = " ' + "random
                    Image[number]+ '" " />",
                                        )
    }
</script>
<body>
<h2> Random image Generator </h2>
<button   onClick = "getRandomImage ( )" >
          Generage Image </button>
<span  id = "result"  align = "center" > </span>
  </body>
</html>
```

__/__/__

JS Objects

↳ Objects are variables too but can contain many values

↳ values are written as name:value pairs

(1) eg.
```
const person = {
    firstName : "abc",
    lastName : "xyz",
    age : 50,
    eyeColor : "blue"
    };
```

Accessing object properties.

Objectname.propertyName

eg- person.lastName;

(2) method() can also be add as a property

```
const person = {
    ⋮
fullName : function()
    { return this.firstName + " " +
        this.lastNa
```

__/__/____

Here, "this" keyword refers to person object

Object constructors

Sometimes we need a "blueprint" for creating many objects of the same type

e.g.

```
function person(first, last, age, eye)
{
    this.firstName = first;
    this.lastName = last;
     this.age = age;
    this.eyecolor = eye;
}
```

```
const myFather = new Person("John", "Doe",
                             "50", "blue");

const myMother = new Person("Soe", "Rim",
                             48, "green");
```

// Adding property to object

✓ myFather.nationality = "Hindu";

// cannot add new property to an object constructor

_/_/__

Person. nationality = "English"

To add a new property or method to constructor, you must add it to constructor function

e.g
```
function Person( —— )
        {
          ≡
          this. nationality = "English";

          ≡
        }
```

Javascript PopUp Boxes

① Alert Box :->

```
        window.alert("Hi, I am a aler
```

② Confirm Box :->

It will have two options either "ok" or "cancel".

```
<body>
<h2> confirm box </h2>
<button onClick = "myfunction()" >Try
                                  </button>

<p id="demo"> </p>
```

Scanned by TapScanner

__/__/__

```
<Script>
function myFunction()
        {
        var txt;
    if(confirm("Press a button"))
        {
          txt="You pressed Ok!";
    }
      else {
            txt="you pressed cancel";
        }
    document ____________ = txt;
    }

  </Script>
```

(3) Prompt Box:→
prompt box is often used if we want the user to input any value

Syntax:- window.prompt("sometext", "defaulttext");

```
let person = prompt("Please enter your
                     name", "Amit");
```

—
→
✓
✓