Variable :—

A Variable is simply the name of a storage location.

Eg:— age = 23; name = "Tony Stark"

↓ ↓

This is variable

Data Types in JS :—

Primitive types [or] Basic.

- Number
- Boolean
- String
- Undefined
- Null
- Bigint
- Symbol.

Note:— typeof defines the type of variable Eg:— typeof 5.9 ⇒ Num

Eg:—

follow → yes or no (Boolean), text (string)

Count (number)

Numbers in JS :—

- Positive (14) & Negative (-4)
- Integers (45, -50)
- floating numbers — with decimal (4.6, -8.9)

Operations in JS :—

```
a = 20
b = 10
// addition          // subtraction       // Multiplication
sum = a+b            diff = a-b           prod = a * b
                     ↓ variable
// division          // modulo
div = a/b            rem = a % b
```

- Modulo (remainder operator)
  12 % 5 = 2
- Exponentiation (power operator)
  2 ** 3 = 8

Note ⇒

- Modulo operator in JS used to detect a number is even or odd.

⇒ odd % 2 = something → odd.

⇒ number % 2 = 0 → even.

Note ⇒

- 2**3 is same as 2^3

NaN in JS :—

The NaN global property is a value representing Not-A-Number.

Eg:— 0/0 = NaN ⇒ typeof → Number

NaN - 1, NaN * 1, NaN + NaN always gets NaN.

Operator Precedence :—

This is the general order of solving an expression.

1. ()
2. **
3. *, /, %
4. +, -

Let Keyword :—

Syntax of declaring variables

Eg:— Let age = 23;
age = age + 1;

Let cgpa;
cgpa = 8.9;

```
Let num1 = 1;
Let num2 = 2;
Let sum = num1 + num2;
```

Const Keyword :-

Values of constants can't be changed with re-assignment & they can't be re-declared

Eg:-

```
Const year = 2025;
year = 2026; // Error
year = year + 1; // Error
```

```
Const pi = 3.14;
Const g = 9.8;
```

Var Keyword :-

Old Syntax of creating variables

Eg:-

```
var age = 23;
var cgpa = 8.9;
```

```
var num1 = 1;
var num2 = 2;
var sum = num1 + num2;
```

PRACTICE Qs

1. What is the value of age after this code runs?

```
let age = 23;
age + 2;        // 23
```

2. What is the value of avg after this code runs?

```
let hindi = 80;
let eng = 90;
let math = 100;
let avg = (hindi + eng + math) / 3;    // 90
```

Assignment Operators :-

```
age = age + 1
age += 1
```

```
age = age * 1
age *= 1
```

```
age = age - 1
age -= 1
```

Unary Operators :—

```
age = age + 1          age = age - 1
age += 1               age -= 1

age++  // increment    age--  // decrement
```

- Pre-increment [change, then use]

```
Let age = 10;
Let newAge = ++age;
```

- Post-increment [Use then c

```
Let age = 10;
Let newAge = age++;
```

PRACTICE QS

1. What is the value of each variable in each line of code?

```
Let num = 5;              // num = 5
Let newNum = num++;       // newNum = 5 , num = 6
newNum = ++num;           // newNum = 7 , num = 7.
```

13) Identifier rules :—

All Javascript variables must be identified with unique names (identifiers).

- Names can contain letters, digits, underscores, and dollar signs. (no space)
- Names must begin with a letter
- Names can also begin with $ and _
- Names are case sensitive [y and Y are different variables]
- Reserved words [like JS keywords] cannot be used as names.

camelCase :—

Way of writing identifiers

- camelCase [JS naming convention]
- snake_case
- PascalCase

14) Boolean in JS :—

Boolean represents a truth value → true [or] false / yes [or] no

Eg:— Let age = 23;
Let isAdult = true;

Let age = 13;
Let isAdult = false;

15) What is TypeScript ?

Statsc Typed, where JS is dynamic typed

• Designed by Microsoft.

16) PRACTICE Qs

1) Find the errors in the following code?

(a) Let 1age = 5; // 1age
Let 2age = 10;

(b) Let marks = 75;
Let isPass = True; // true

(c) Let isPass = 'true'; // true & 'true' => string

17) String in JS :—

Strings are text [or] sequence of characters

Eg:— Let name = "Tony stark";
Let role = 'Ironman';
Let char = 'a';
Let num = '23';
Let empty = " ";

18) String Indices :—

Let name = "TONY STARK"; [index = position]

T	O	N	Y		S	T	A	R	K
0	1	2	3	4	5	6	7	8	9

name[0] → 'T'

name[1] → 'O' ...

→ This is 0 based indexing.

Eg:- let name = "TONY STARK";
name[0] = T

- name.length => 10 [length of the string.
- type of name.length // number.

Concatenation :-
Adding strings together
"tony" + " " + "stark" = "tony stark";
"tony" + 1 = tony1;

19) null & undefined in JS :-

- undefined :-

A Variable that has not been assigned a value is of type undefined.

Eg:- let a; // undefined.

- null :-

The null value represents the intentional absence of any object value.

To be explicitly assigned.

Eg:- let a = null; // null.

PRACTICE Qs :-

1) Declare your name as a string and print its length in JS
name = "Konda Vinay Kumar"; -> // 17

2) Declare your first name as a string & print its first character.
name = "Vinay"; -> name[0] -> // V

3) Declare your first name as a string & print its last character.
name = "Vinay"; -> name[name.length - 1] -> // y.

4) What is the output of following code:
"apnacollege" + 123; => // apnacollege123.

5) What are lengths of an empty string & a string with a single space?

- Length of an empty string is `emptyString = "";` →
  `emptystring.length => // 0;`
- Length of string with single space →
  `singleSpaceString = " Vinay";`
  `singleSpaceString.length`
  `// 6` → `singleSpaceString + 1 <=>`

→ Javascript Part 02

02)

Console.log() :—

To write (log) a message on the console.

Eg:— `console.log("Apna College");`
`console.log(1234);`
`console.log(2+2);`
`console.log("Apna", "College", 123);`

02. Linking JS File :—

`<script src="app.js"></script>`

Note:— We add script tag at the last of body tag because the page should load html elements first.

03. Template Literals :—

They are used to add embedded expressions in a string.

```
let a = 5;
let b = 10;
console.log(`Your pay ${a + b} rupees`);
// console.log("Price ", a+b, "rupees");
```

04. Operators in JS :—

- Arithmetic [+, -, *, /, %, **]
- Unary [++, --]
- Assignment [=, +=, -=, *=, /=, %= etc.]
- Comparison
- Logical

Eg:
```
let a = 10;
let b = 5;
console.log(a++); // 10
console.log(++a); // 12
```

Eg. 2:—
```
let a = 10;
let b = 5;
b = a;
console.log(b); // 10
```

05. Comparison Operators :—

Comparison Operators to compare 2 values

- `>` → Greater than
- `>=` → Greater than or equal to
- `<` → Lesser than
- `<=` → Lesser than or equal to
- `==` → equal to
- `!=` → not equal to

Eg:—
```
• let age = 18;
  console.log(age > 18); // false
```
- 5 != 5 // false
- 5 != 4 // true.

- 5 == 5 // true.
- 5 == 4 // false.

- 5 == '5' // true

Note:— == operator doesnot look numbers type only compares values.

Eg:—
```
let n = 5;
let str = '5';
n == str
// Output : True.
```

- 4 != str
// True

IMP :-

==	===
• Compares value, not type	• compares type & value
"123" == 123	"123" === 123
true	false
1 == '1'	1 === '1'
true	false
0 == ''	0 === ''
true	false
0 == false	0 === false
true	false
null == undefined	null === undefined
true	false.

06. Comparison for non-numbers :-

'a' > 'A'

true

'a' > 'b'

false

'b' < 'c'

true

'B' < 'c'

true

'*' < '&'

false

'a' → 61, 'b' → 62....

'A' → 41, 'B' → 42...

07. Conditional statements :-

- if-else
- nested if-else
- switch

08. if statement :

```
// some code before if

if (some condition) {
    // Do something
}

// some code after if.
```

Eg:-

```
• console.log("before if statement");
  let age = 23;
  if (age >= 18) {
      console.log("You can Vote!");
  }
console.log("after if statement");
```

Note:-

- If the condition in if statement is false, it will not execute any code inside the curly brackets.

Eg:-

```
• let firstName = "Vinay";
  if (firstName == "Vinay") {
      console.log(`Welcome ${firstName}`);
  }
```

09. PRACTICE Qs :-

01. Create a traffic light system that shows what to do based on color.

```
→ let color = "green";
  if (color == "red") {
      console.log("stop");
  }
  if (color == "yellow") {
      console.log("Go slow!");
  }
  if (color == "Green") {
      console.log("Go");
  }
```

10. Else if statement :-

• else if statement :—

```
if (condition) {
    // code
}
else if (condition 2) {
    // code
}
else if (condition 3) {
    //code
}
```

Note:—

• The condition in the if statement is true then the program stops there i.e., it will not execute chalks else if state.

Eg:—

```
• let marks = 75;
  if (marks >= 80) {
     console.log ("A+");
  } else if (marks >=60) {
     console.log ("A");
  } else if (marks >= 33) {
     console.log ("B");
  } else if (marks < 33) {
     console.log ("F");
  }
```

Eg:—

```
• let age = 14;
  if (age >= 18) {
     console.log ("You can vote");
  }
  else if (age < 18) {
     console.log ("You cannot vote");
  }
```

II. Else statement :—

```
• if (condition 1) {
    // Do something
  }
  else {
    // Do something else
  }
```

Eg:-

```
let age = 17;
if (age >= 18) {
  console.log("You can vote");
} else {
  console.log("You cannot vote");
```

Note:-
else statement will execute automatically if if & else if's are false.

12. PRACTICE Qs :-

01. Create a system to calculate popcorn prices based on the size customer asked for :

if size is 'XL', price is ₹. 250
if size is 'L', price is ₹. 200
if size is 'M', price is ₹. 100
if size is 'S', price is ₹. 50.

```
let comPrice = "M";
if (comPrice == "XL") {
  console.log("The price is 250");
} else if (comPrice == "L") {
  console.log("The price is 200");
} else if (comPrice == "M") {
  console.log(comPrice == "M") ("The Price is 100");
} else if (comPrice == "S") {
  console.log("The price is 50");
}
else {
  console.log("Please, select a size XL, M, L or, S");
}
```

13. Nested if-else :-

Nesting is writing if-else inside if-else statements. It can have many levels.

```
if marks >= 33
    if marks >= 80
        print "O"
    else
        print "A"
else
    print "Better luck next time!"
```

Eg:-

```
let marks = 45;
if (marks >= 33) {
    console.log("Pass");
    if (marks >= 80) {
        console.log("Grade : O");
    } else {
        console.log("Grade : A");
    }
} else {
    console.log("Better luck next time!");
}
```

14. Logical Operators :-

Logical Operators to combine expressions.

&& Logical AND (exp1) && (exp2)

|| Logical OR , ! Logical NOT.

-> true && true
// true

-> true && false
// false

-> false && true
// false

-> false && false
// false.

— AND —

Eg:-

- (5 > 3) && (3 > 1) // true
- (5 > 3) && (3 < 1) // false
- let marks = 75;
  if (marks >= 33 && marks >= 80) {
  console.log ("Pass");
  console.log ("A+");
  }

— OR —

- let marks = 75;
  if (marks >= 33 ~~&&~~ || marks >= 80) {
  console.log ("Pass");
  console.log ("A+");
  }

— NOT —

- !true // false
- let marks = 75;
  if (!(marks < 33)) {
  console.log ("Pass");
  console.log ("A+");
  }

Eg:- let marks = 75;
if ((marks > 33 && marks <= 80) || !false) {
console.log ("Pass");
}

Note:- If we use multiple Logical statements, it will start executing from left → right.

15. PRACTICE Qs :-

Q1. A "good string" is a string that starts with the letter 'a' and has a length > 3. Write a program to find if a string is good or not.

→ let goodString = "Automotive";
if ((goodString[0] == = "a") && (goodString.length > 3)) {
console.log (" This is a good string");
}
else {
console.log (" It's a bad string");
}

02. Predict the output of the following code :
let num = 12;
if ((num % 3 === 0) && ((num + 1 == = 15) || (num - 1 == = 11)
)) {
console.log ("safe");
} else {
console.log ("unsafe");
}
// Output : Safe.

16. Truthy :—
Everything in JS is true [or] false (in boolean context)
This doesn't mean their value itself is false [or] true, but they are treated as false [or] true if taken in boolean context.

Falsy values :—
false, 0, -0, on (BigInt value), " " (empty string), null, undefined, NaN.

Truthy values :—
Everything else.

Eg:—
if (true) {
console.log ("It has true value");
} else {
console.log ("It has false value");
}
// It has true value.

17. Switch statement :—

- Used when we have some fixed values that we need to compare to.

```
let color = "red";
switch (color) {
    case "red" :
        console.log("stop");
        break;
    case "yellow" :
        console.log("slow down");
        break;
    case "green" :
        console.log("Go!");
        break;
    default :
        console.log("Broken Light");
}.
```

18. PRACTICE QS :—

01. Use switch statement to print the day of the week using a number variable (day) with value 1 to 7.
1 = Monday, 2 = Tuesday & so on.

```
let day = 4;
switch (day) {
    case 1 :
        console.log("Monday");
        break;
    case 2:
        console.log("Tuesday");
        break;
    case 3:
        console.log("Wednesday");
        break;
    case 4:
        console.log("Thursday");
        break;
    default :
        console.log("End!");
}
```

19. Alerts :—

Alert & Prompt :—

- Alert displays an alert message on the page.
  - alert ("something is wrong");

- Prompt displays a dialog box that asks user for some input.
  - prompt ("please enter your roll no".);
  - console.log ("This is a simple log");
  - console.error ("This is an error msg");
  - console.warn ("This is a warning msg");

- Javascript Part 03 :—

01. String methods :—

Methods - actions that can be performed on objects.

Format :

stringName.method()

02. Trim method :—

let msg = " Hello ";

str.trim()

Trim whitespaces from both ends of string & returns a new one.

- let msg = " Hello ";
  msg.trim(); // 'Hello'
  msg; // " Hello ";

Output : "Hello", but value of msg remains same.

03. Strings are Immutable :— In JS :—

No changes can be made to strings.
Whenever we do try to make a change, a new string is created and old one remains same.

Eg:- • let msg = "apna ";
msg.trim(); // 'apna';
let str = msg.trim();
Console.log(str); // 'apna'.

• Note => Console.log(msg); // 'apna ';

04. ToUpperCase and ToLowerCase :-

String methods :-

• let str = "Random string";
str.toUpperCase() "RANDOM STRING".
str.toLowerCase() "random string".

05. Methods with Arguments :-

Argument is some value that we pass to the method.

Format:
stringName.method(arg).

• Index of :-
Returns the first index of occurence of some value in string. [0-n] gives -1 if not found.

Eg:-
let str = "ILoveCoding";
str.indexOf("Love") // 1.
str.indexOf("J") // -1 (not found).
str.indexOf("O") // 2 (only 1 index).

06. Method chaining :-

Using one method after another. Order of execution will be left to right.

- str.toUpperCase().trim()

07. slice Method :—

Returns a part of the original string as a new string.

- let str = "ILoveCoding";
  str.slice(5) // coding [or] str.slice(start, end)
  str.slice(1, 4) // love.
  str.slice(-num) // str.slice(start)
  [or]
  str.slice(length - num)

08. Replace and Repeat Method :—

Searches a value in the string & returns a new string with the value replaced.

- let str = "ILoveCoding";
- str.~~slice~~replace("love", "do") // Idocoding.
- str.~~slice~~replace("o", "x") // Ilxvecoding.

Repeat :—

- Returns a string with the number of copies of a string.
- let str = "Mango";
  str.repeat(3) // MangoMangoMango.

09. PRACTICE Qs :—

01. for the given string :—
  let str = " help! ";
  Trim it & convert it to uppercase.

- str.trim().toUpperCase().

02. for the string -> let name = "ApnaCollege", predict the output for following :—

-> name.slice(4,9) // Colle

-> name.indexOf("na") // 2

-> name.replace("Apna", "Our") // OurCollege

03. Separate the "College" part in above string & replace 'l' with 't' in it.

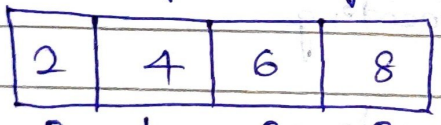
- let str = "College";
  str.replace("l", "t"); // Cotlege.

10. Array (Data Structure) :—

-> Linear collection of things.

- let students = ["Vinay", "Konda"];

11. Visualizing Array :—



-> let nums = [2,4,6,8];
nums.length; // 4.
typeof (nums); // object.

12. Creating Arrays :—

let marks = [99,85,93,76,62];
let names = ["adam", "bob", "catlyn"];
let info = ["aman", 25, 6.1] // mixed array.

// empty array
let newArr = [];

- names[0]; // adam
- names[0][0]; // a

13. Arrays are Mutable :-

let fruits = ["mango", "apple", "lychis"];

fruits[0] = "banana";

fruits; // banana, apple, lychis.

→ fruits[10] = "papaya";

→ Console.log(fruits); // banana, pineapple, lychis, empty x7, papaya

14. Array Methods :-

- push : add to end
- pop : delete from end & returns it
- Unshift: add to start
- shift : delete from start & returns it

- let cars = ["audi", "bmw", "benz", "maruti"];
  cars.push("toyota");
  cars.pop(); // toyota
  cars.unshift("toyota");
  cars.shift();

15. PRACTICE Qs :-

Q1. For the given start state of an array, change it to final form using methods

start: ["january", "july", "march", "august"];

final: ['july', 'june', 'march', 'august'];

- Let months = ['january', 'july', 'march', 'august'];
  months.shift();
  months.shift();
  months.unshift('june');
  months.unshift('july');

16. IndexOf and Includes Method :—

IndexOf : returns index of something.

Eg:— let primary = ['red', 'yellow', 'blue'];
primary.indexOf('yellow'); // 1
primary.indexOf('green'); // -1
primary.indexOf('Yellow'); // -1

Includes : search for a value.

Eg:— primary.includes("red"); // true.
primary.includes("green"); // false.

17. Concatenation and Reverse :—

→ concat : merge 2 arrays

- Let primary = ['red', 'yellow', 'blue'];
- Let secondary = ['orange', 'green', 'violet'];

Eg:— primary.concat(secondary);
// red, yellow, blue, orange, green, violet.

→ reverse : reverse an array

primary.reverse(); // blue, yellow, red.

18. Slice in Arrays :—

Slice : copies a portion of an array.

- let colors = ['red', 'yellow', 'blue', 'orange', 'pink', 'white'];

slice (start, end)

```
colors.slice();
// red, yellow, blue, orange, pink, white
colors.slice(2);
// blue, orange, pink, white.
colors.slice(2,3);
// blue.
colors.slice(-2);
// pink, white
```

19. Splice in Arrays :—

Splice : removes / replaces / add elements in place.

Splice (start, deleteCount, item0 .... itemN)

- let colors = ['red', 'yellow', 'blue', 'orange', 'pink', 'white'];

```
colors.splice(4);
// pink, white
colors;
// red, yellow, blue, orange.
colors.splice(0,1);
// red
colors;
// yellow, blue, orange.
colors.splice(0, 1, "black", "grey");
// yellow.
colors;
// black, grey, blue, orange.
```

20. Sort in Arrays :—

sort : sorts an array.

- let days = ['monday', 'sunday', 'coedhesday', 'tuesday'];
  days.sort();
  // monday, sunday, tuesday, coedhesday.

- let squares = [25, 16, 4, 49, 36, 9]
  squares.sort();
  // 16, 25, 36, 4, 49, 9.

Note :— for numbers this will not coork properly.

21. PRACTICE Qs :—

01. For the given start state of an array, change it to final form using splice.

start : ['january', 'july', 'march', 'august'];

final : ['july', 'june', 'march', 'august'];

- let months = ['january', 'july', 'march', 'august'];
  months.splice(0, 2, "july", "june");

02. Return the index of the "javascript" from the given array, if it is coas reversed.

['c', 'c++', 'html', 'javascript', 'python', 'java', 'c#', 'sql']

- lang.reverse().indexOf('javascript');
  // 4.

22. Array References :—

[1] === [1]
// false

[1] == [1]
// false

• **Note**:— references : Address in memory.

23. Constant Arrays :—

```
const arr = [1,2,3];
arr.push(4);   // 1,2,3,4
arr.pop();     // 1,2,3,4
```

Note :— You can't change array to new Array.

24. Nested Arrays :— [or] Multi dimensional Arrays

array of arrays

```
let nums = [ [2,4], [3,6], [4,8] ];
             0      1      2
nums[0];    // 2,4
nums[0][0]; // 2
```

	0,0	0,1	
	2	4	
1,0	3	6	1,1
2,0	4	8	2,1

rows = 3 —>
cols = 2 —>

25. PRACTICE QS :—

01. Create a nested Array to show the following tick-tac-toe game state.

X		O
	X	
O		X

2D array

'x'	null	'o'
null	'x'	null
'o'	null	'x'

```
let game = [ ['x', null, 'o'], [null, 'x', null], ['o', null, 'x'] ];

game; //
game[0][1] = 0;
```

Javascript Part 04 :—

01. for Loops:

Used to iterate a piece of code.

```
for(initialisation; condition; updation) {
    // do something
}
```

02. Dry Run:

```
for (let i=1; i<=5; i++) {
    console.log(i);
}
```

① ← ② ↓ ③ ↑ ④

Start; ending; update

Note:— ① only runs once & variables declared inside for loop only runs inside it.

03. Print odd Numbers:

- Print all odd numbers (1 to 15)

```
for (let i=1; i<=15; i=i+2) {
    console.log(i);
}
```

start; end; update

1 — start
3 +2
5 +2
7 +2
9
11
13
15 — end

04. Print Even Numbers:

- Print all even numbers (2 to 10)

```
for (let i=2; i<=10; i=i+2) {
    console.log(i);
}
```

2 — start
4 +2
6 +2
8
10 — end

05. Infinite Loops:

- `for(let i=1; i>=0; i++) {`

`}`

```
• for (let i=1;  i<=5;  i--) {
  }

• for (let i=1;   ;  i++) {
  }
```

• ending condition is missing [or] true then it is an infinite Loop.

06. Print Multiplication Table :

• Print the multiplication table for 5

```
→ for( let i=5;  i<=50;  i = i+5); {
    console.log(i);
  }
```

Eg:-

```
• let  n = prompt("Write your number");
    for( let i=n;  i<=n*10;  i=i+n) {
      console.log(i);
    }
    n=parseInt(n);
```

07. Nested for loop :

```
• for( let i=1;  i<=3;  i++) {
    for( let j=1;  j<=3;  j++) {
      console.log(j);
    }
  }
```

output :

1
2
3
1
2
3
1
2
3

08. While Loops :

- **while loop**

```
while(condition) {
  // do something
}
```

```
let i = 1;
while (i<=5) {
  console.log(i);
  i++;
}
```

09. favourite Movie : (Activity)

```
const favMovie = "Dictator";
let guess = prompt("Guess my fav movie");
while ((guess != favMovie) && (guess != "quit")) {
    guess = prompt("wrong guess, please try again");
}
if (guess == favMovie) {
    console.log("congrats");
}
else {
    console.log("You quit");
}
```

10. Break keyword :

```
let i=1;
while(i<=5) {
    console.log(i);
    i++;
}
```

```
if(i==3) {
    break;
}
```

- This stops the execution of the program when the condition is true.

11. Loops with Arrays :

- let fruits = ["mango", "apple", "banana", "litchi", "orange"];

```
for (let i=0; i<fruits.length; i++) {
    console.log (i, fruits[i]);
}
```

output:

```
0   mango
1   apple
2   banana
3   litchi
4   orange
```

12. Loops with Nested Arrays :—

Nested Loops with Nested Arrays

- let heroes = [
  ["ironman", "spiderman", "thor"],
  ["superman", "coonder coorman", "flash"]];

```
for (let i=0; i< heroes.length; i++) {
    console.log (`List # ${i}`);
    for (let j=0; j< heroes[i].length; j++) {
        console.log(heroes[i][j]);
    }
}
```

13. for of Loops:

- for (element of collection) {
  // do something
  }

- let fruits = ["mango", "apple", "banana", "litchi", "orange"];
```
for (fruit of fruits) {
    console.log(fruit);
}
```

output :
```
mango
apple
banana
litchi
orange
```

- for (char of "mango") {
```
    console.log(char);
}
```

output :
```
m
a
n
g
o
```

14. Nested for of loop :

- let heroes = [
```
    ["ironman", "spiderman", "thor"],
    ["superman", "coonder coonman", "flash"]];
for (list of heroes) {
    for (hero of list) {
        console.log(hero);
    }
}
```

Javascript Part 05

01. Object Literals :

Used to store keyed collection & complex entities.

property => (Key, value) pair

objects are a Collection of properties.

02. Creating Object Literals :

- let delhi = {
  latitude : " 28.7041° N ",
  longitude : " 77.1025° E"
  };

- const item = {
  price : 100.99,
  discount : 50,
  colors : ["red", "blue"]
  };

- const student = {
  name : " Shradha",
  age : 23,
  marks : 94.4,
  city : "HYD"
  };

03. Creating a post :

Thread / Twitter post

- Create an object literal for the properties of thread/twitter post which includes -
  - username
  - content
  - likes
  - reposts
  - tags

→ • const post = {
username : "@vinaykumar",
content : "This is my #first post",
likes : 150,
reposts : 5,
tags : ["@abhi", "@codeDevi"]
};

04. Get Values :

• let student = {
name : "shradha",
marks : 94.4
};
→ student["name"]
→ student.name

05. Conversion in Get Values :

JS automatically converts objects keys to strings. Even if we made the number as a key, the number will be converted to string.

Eg:- const obj = {
1 : "a",
2 : "b",
true : "c",
null : "d",
undefined : "e"
};

06. Add and Update Values :

• Change the city to "Mumbai"
• Add a new property, gender : "female"
• Change the marks to "A"

```
const student = {
    name : "Vinay",
    age : 24,
    marks : 94.8%,
    city : "Delhi"
};
```

```
-> student.city = "Mumbai";
-> student.gender = "female";
-> student.marks = "A";
```

- To delete an object:

```
-> delete student.marks;
```

07. Nested Objects :-

Storing information of multiple students

```
const classInfo = {
    vinay : {
        grade : "A+",
        city : "NZB"
    },
    person1 : {
        grade : "B",
        city : "KMR"
    },
    person2 : {
        grade : "f",
        city : "USA"
    }
};
```

08. Array of objects :—

Storing information of multiple students

```
const classInfo = [
    {
        name : "Vinay",
        grade : "A+",
        city : "NZB"
    },
    {
        name : "Person1",
        grade : "A+",
        city : "HYD"
    },
    {
        name : "Person2",
        grade : "B",
        city : "Mangalore"
    }
];
```

→ Adding values :

- `classInfo[0].gender = "Male";`

09. Math Object :

Properties	Methods
Math.PI	Math.abs(b)
Math.E	Math.pow(a,b)
	Math.floor(b)
	Math.ceil(b)
	Math.random()

- `Math.abs(-12)` // 12

→ gives absolute value [or] gives neutral number.

- Math.pow(2,4); // 16 [or] $2^4 = 16$
- Math.floor(5.5) // 5
  → gives less than [or] equal [or] round off value. ie
  → Nearest Smallest integer value
- Math.ceil(5.5) // 6
  → Largest integer value
- Math.random()
  → gives value from 0 to 1 but exclusive of 1.

10. Random Integers :

From 1 to 10

Step-1 :

```
let num = Math.random(); // 0.467
```

Step-2 :

```
num = num * 10; // 4.67
```

Step-3 :

```
num = Math.floor(num); // 4
```

Step-4 :

```
num = num + 1; // 5
```

[or]

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

11. PRACTICE QS :—

01. Generate a random number between 1 and 100
- → Math.floor(Math.random() * 100) + 1;

02. Generate a random number between 1 and 5.
- → Math.floor(Math.random() * 5) + 1;

03. Generate a random number between 21 and 25.
- Math.floor(Math.random() * 5) + 25;

12. Guessing Game :

- User enters a max number and then tries to guess a random generated number between 1 to max.

```
const max = prompt ("Enter the max number");
const random = Math.floor(Math.random() * max) +1;
let guess = prompt ("Guess the number");

while (true) {
    if (guess == "quit") {
        console.log("User quit");
        break;
    }
    if (guess == random) {
        console.log("You are right!, Congrats!!!");
        break;
    } // else {
    //guess = prompt ("Your guess was wrong");
    }
}
    else if (guess < random) {
        guess = prompt ("hint: Your guess was too small!");
    }
    else {
        guess = prompt ("hint: Your guess was too large");
    }
}
```

Javascript Part 06 :

01. What are functions :

- Functions in JS

→ Function Definition (telling JS)

```
function funcName() {
    // do something
}
```

- ```
function hello() {
    console.log("hello");
}
```

→ Function Calling (Using the function)

```
functionName();
```

- ```
hello();
```

Eg:-

- ```
function print1to5() {
    for(let i=1; i<=5; i++) {
        console.log(i);
    }
}
print1to5();
```

- ```
function isAdult() {
    let age = 18;
    if(age >= 18) {
        console.log("Adult");
    } else {
        console.log("Not adult");
    }
}
isAdult();
```

02. Practice Qs 1 :

- Create a function that prints a poem.

```
-> function printPoem() {
       console.log(" Twinkle Twinkle, little star ");
       console.log(" How I wonder what you are");
   }
   printPoem();
```

03. Practice Qs 2 :

- Create a function to roll a dice & always display the value of the dice (1 to 6)

```
-> function rollDice() {
   let random = Math.floor(Math.random() * 6) + 1;
       console.log(random);
   }
   rollDice();
```

04. Functions with Arguments :

Values we pass to the function

- function funcName(arg1, arg2....) {
  // do something.
  }

Eg:->

```
- function printName(name) {
      console.log(name);
  }
  printName("Vinay");
  printName("Konda");
```

- function printInfo(name, age) {
  console.log(`${name}'s age is ${age}`);
  }
  printInfo("Vinay", 24);

Note:-
If we pass a value first it will occupy first & it will save order wise rest are undefined.
Eg:-
- printInfo(24); // 14's age is undefined.

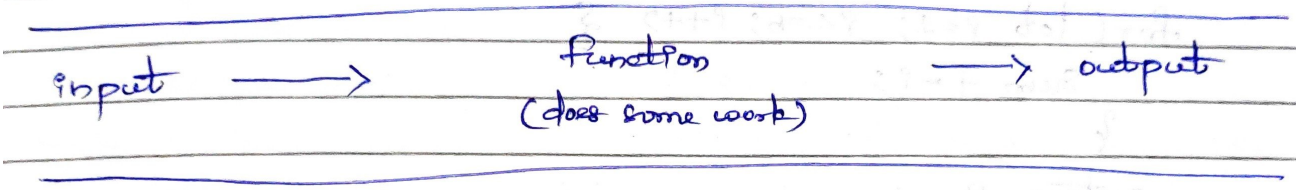
- function sum(a,b) {
  console.log(a+b);
  }
  sum(1,5); // 6

05. Practice Qs 3 :
- Create a function that gives us the average of 3 numbers.
  function calcAvg(a,b,c) {
  let avg = (a+b+c)/3;
  console.log(avg);
  }
  calcAvg(2,4,6); // 4.

06. Practice Qs 4 :
- Create a function that prints the multiplication table of a number.
  function table(n) {
  for(let i=n; i<=n*10; i=i+n) {
  console.log(i);
  }
  }
  table(5);

07. Return Keyword :

- **Return :—**
return keyword is used to return some value from the function



- function funcName(arg1, arg2, ...) {
    // do something
    return val;
  }

Eg:—

```
function sum(a,b) {
    return a+b;
}
let s = sum(3,4);   // 7.
console.log(s);
```

Note:— if we code/ print after return statement then that will not be executed.

- function isAdult(age) {
    if(age >= 18) {
      return "adult";
    } else {
      return "not adult";
    }
  }

08. Practice Qs 5 :-

- Create a function that returns the sum of numbers from 1 to n.

→
```
function getSum(n) {
    let sum = 0;
    for(let i=1; i<=n; i++) {
        sum += i;
    }
    return sum;
}
getSum(3);      // 6.
```

09. Practice Qs 6 :

- Create a function that returns the concatenation of all strings in an array.

→
```
let str = ["hi", "hello", "Bye", "!"];
function concat(str) {
    let result;
    for(let i=0; i<=str.length; i++) {
        result = result + str[i];
    }
    return result;          // hihelloBye!
}
```

10. What is Scope :

- Scope determines the accessibility of variables, objects and functions from different parts of the code.
  - function
  - Block
  - Lexical
  - Global

- **function scope:**
Variables defined inside a function are not accessible (visible) from outside the function.

```
- function calSum(a, b) {
        let sum = a+b;  // function Scope
  } calsum(1,2);
  console.log(sum);     // error: —
```

Note:— You can declare variables with same name also but outside the function.

Eg:—

```
- let sum = 54;  // Global Scope
  function name(){
    // code
  }
  name();
```

- **Global Scope** : It has access everywhere.

11. **Block Scope** :—
Variables declared inside a {} block cannot be accessed from outside the block.

Eg:—
```
{
    let a=25;
}
console.log(a);  // error.
```

Note:— Block scope doesn't work on 'var'.

12. **Lexical Scope** :— (nested function)
A Variable defined outside a function can be accessible inside another function defined after the variable declaration.
The Opposite is NOT true.

- function outerfunc () {
  let x=5;
  let y=6;
  function innerfunc () { // function scope.
    console.log(x);
  }
  → innerfunc();
}
outerfunc(); // 5

- function outerfunc() {
  function innerfunc() {
    console.log(x);
    console.log(y);
  }
  let x= 5;
  let y=6;
  innerfunc();
}
// hoisting

13. Practice Qs 7 :

- What will be the output ?

```
let great = "hello";          // Global Scope
function changeGreat() {
    let great = "namaste";    // function Scope
    console.log(great);
    function innerGreat() {
        console.log(great);   // Lexical Scope.
    }
}
console.log(great);
changeGreat();                // hello
                              // namaste.
```

14. function Expressions :

- const variable = function (arg1, arg2....) {
    // do [or] return something
  }

- const sum = function (a, b) {
    return a+b;
  }
  sum(2,3);

→ A function stored in a variable is known as FE/ function expression. And this is name less.

15. Higher Order functions :

- A function that does one [or] both.
  - takes one [or] multiple functions as arguments.
  - returns a function.

Takes one [or] multiple functions as arguments.

- function multipleGreet (func, n) {
    for (let i=1; i<=n; i++) {
      func();
    }
  }
  let greet = function() {
    console.log("hello");
  }
  multipleGreet(greet, 2);

Eg:-

```
let greet = function() {
    console.log("hello");
}
greet();
greet();
```

-> THE ABOVE PROGRAM CAN BE DONE IN ANOTHER WAY - FOR THAT WE USE HIGHER ORDER FUNC's <-

```
function multipleGreet(func, count) {
    for(let i=1; i<=count; i++) {
        func();
    }
}
let greet = function() {
    console.log("hello");
}
multipleGreet(greet, 2);
```

***

16. Higher Order functions (Returns) :

Returns a function.

```
function oddEvenTest(request) {
    if(request == "odd") {
        return function(n) {
            console.log(!(n%2==0));
        }
    } else if(request == "even") {
        return function(n) {
            console.log(n%2==0);
        }
    } else {
        console.log("wrong request");
    }
}
```

- for odd & Even numbers :—

```
let odd = function (n) {
    console.log ( !(n%2 ==0));
}

let even = function (n) {
    console.log (n%2 ==0) ;
}
```

[or]

```
function oddOrEvenFactory (request) {
    if (request == "odd") {
        let odd = function (n) {
            console.log (!(n%2 ==0));
        }
        return odd;
    } else if (request == "even") {
        let even = function (n) {
            console.log (n%2 ==0);
        }
        return even ;
    } else {
        console.log ("wrong request");
    }
}
let request = "odd" ; // even.
```

17. Methods :

Actions that can be performed on an object.

```
const calculator = {
    add : function (a, b) {
        return a+b;
    },
```

```
    sub : function(a,b) {
        return a-b;
    },
    mul : function (a,b) {
        return a*b;
    }
};
calculator.add(1,2); //3.
```

- Methods (Shorthand)

```
const calculator = {
    add(a,b) {
        return a+b;
    },
    sub(a,b) {
        return a-b;
    }
};
```

Javascript Part 07

01. This Keyword :

- "This" Keyword refers to an object that is executing the current piece of code.

```
const student = {
    name : "Vinay",
    age  : 24,
    eng  : 95,
    math : 93,
    phy  : 97,
    getAvg() {
        let avg = (eng + math + phy) / 3;
        console.log(avg);
    }
}                                   //error
```

→ In the above the method "getAvg()" cannot access math, eng, phy directly.
→ In this case, we have to use "this" keyword.

```
getAvg() {
    let avg = (this.eng + this.math + this.phy)/3;
    console.log(avg);
}
```

02. try & Catch :

- The try statement allows you to define a block of code to be tested for errors while it is being executed.

- The Catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

- try {
    console.log(a);
  } catch {
    console.log("variable a doesn't exist");
  }

Eg:-

```
console.log("hello");
console.log(a);                    // hello
console.log("hello");
```

- Here it stops executing after the error.

NOW

```
console.log("hello");
try {
    console.log(a);
} catch(e) {
    console.log("error");
}
console.log("hello");
```

// hello
error
hello.

03. Arrow functions: (Miscellaneous Topics)

- const func = (arg1, arg2....) => { func defn }
- const sum = (a,b) => {
    console.log(a+b);
  };

Eg:-

```
const cube = (n) => {
    return n*n*n;
};
console.log(cube(3));   // 27.
```

```
const pow = (a,b) => {
    return a**b;
};
pow(2,3);  // 16.
```

04. Implicit Return in Arrow functions :

```
const func = (arg1, arg2...) => { value }
```

```
const mul = (a,b) => (
        a*b
);
```

→ In this we remove "return" keyword and we will use "braces" inside of curly braces.

05. Set Timeout function :

```
setTimeout(function, timeout)
                ↳ call back.
```

```
→ console.log("Hi there");

setTimeout( () => {
        console.log("Apna College");
}, 4000);

console.log("Welcome to");
```

Note:— the setTimeout func executes only after certain period of time.

06. Set Interval function :—

- setInterval (function, timeout)

```
setInterval( () => {
    console.log("Apna College");
}, 2000);
```

- clearInterval (id)

Note:— This executes the statement after certain time and repeats the same again

- ```
let id = setInterval(() => {
    console.log("Vinay");
}, 2000);
clearInterval(id);
```

- It stops executing when we write clearInterval(id);

\*\*\*

## 07. This with Arrow function :

| Arrow func | Normal Function |
|---|---|
| 1) Lexical scope<br>parent scope → call | 1) Scope → this → calling object |

Eg:—

- ```
const student = {
    name : "Vinay",
    marks : 95,
    prop : this,
};                                  // window obj because of "this"
```

```
const student = {
    name : "Vinay",
    marks : 100,
    prop : this,        // global scope
    getName : function () {
        return this.name;  -> // Vinay
    },
};
```

```
const student = {
    name : "Vinay",
    marks : 95,
    prop : this,        // global scope
    getName : function () {
        console.log(this);
        return this.name;
    },
    getMarks : () => {
        console.log(this);   // parent's scope -> window
        return this.marks;
    },
};
```

```
    getInfo1 : function () {
        setTimeout( () => {
            console.log(this);  // student
        }, 2000);
    },
    getInfo2 : function () {
        setTimeout(function () {
            console.log(this);      // window.
        }, 2000);
```

08. PRACTICE Qs :

- Write an arrow function that returns the square of a number 'n'.

```
const square = (n) => {
    return n*n;
};
console.log(square(7));     //49
```

[Or]

```
const square = n => (n * n);
console.log(square(7));     //49.
```

- Write a function that prints "Hello World" 5 times at intervals of 2s each.

```
let id = setInterval(() => {
    console.log("Hello World");
}, 2000);

setTimeout(() => {
    clearInterval(id);
}, 12000);
```

Javascript Part 08 :

01. Array Methods :

- forEach
- map
- filter
- some
- every
- reduce

→ forEach :

```
arr.forEach(some func defn or name);
```

Eg:-

```
let arr = [1, 2, 3, 4, 5];
function print(el) {
    console.log(el);
}
arr.forEach(print);
```

[or]

```
arr.forEach(function(el) {
    console.log(el);
});
```

[or]

```
arr.forEach( (el) => {
    console.log(el);
});
```

Eg 2:->

```
let arr = [
    {
        name : "Vinay",
        marks : 100,
    },
    {
```

```
    name : "person1",
    marks : 95,
  },
    {
      name : "person2",
      marks : 99,
    },
  ];
  arr.forEach((student) => {
        console.log(student.marks);
  });
```

02. Map and filter :

- **Map :**

  let newArr = arr.map(some func defn or name);

- let nums = [1, 2, 3, 4];
  let double = nums.map(function (el) {
        return el * 2;
  });

Eg:-

```
let students = [
    {
      name : "Vinay",
      marks : 95,
    },
    {
      name : "phone",
      marks : 98,
    },
    {
      name : "Android",
      marks : 100,
    },
```

```
];
let gpa = students.map( (el) => {
    return el.marks / 10;
});
gpa;    // 9.5, 9.8, 10.0.
```

- **filter :**

- let newArr = arr.filter(some func defn [or] name);

- ```
  let nums = [2, 4, 1, 5, 6, 2, 7, 8, 9];
  let even = nums.filter( (num) => (num % 2 == 0) );
  ```

03. **Every and Some :** (AND)

Returns true if every element of an array gives true for some function. Else returns false.

- arr.every(some func defn [or] name);

Eg:-

```
[1, 2, 3, 4].every( (el) => (el % 2 == 0) );
// false
[2, 4].every( (el) => (el % 2 == 0) );
// true.
```

- **Some :** (OR)

→ Returns true if some elements of array give true for some function. Else returns false.

- arr.some(some func defn [or] name);

Eg:-

```
[2, 2, 3, 4].some( (el) => (el % 2 == 0) );
// true.
```
```

OR

- [1, 3].some((el) => (el % 2 == 0));
  // false

04. Reduce Method :
Reduces the array to a single value

- arr.reduce(reducer func with 2 variables for (accumulator, element));

[1, 2, 3, 4].reduce((res, el) => (res + el));
// 10.

Explanation of above code :—

[1, 2, 3, 4]
↑ ↑ ↑ ↑

let finalVal = nums.reduce((res, el) => res + el);

(res, el)

(0, 1) => 1
(1, 2) => 3
(3, 3) => 6
(6, 4) => 10

[or]

- let nums = [1, 2, 3, 4];
  let finalVal = nums.reduce((res, el) => {
  console.log(res);
  return res + el;
  });
  console.log(finalVal);

05. Maximum in Array :—

Finding Maximum in an array.

```
let nums = [2,3,4,5,3,4,7,8,1,2];

let result = nums.reduce ((max, el) => {
    if (el > max) {
        return el;
    } else {
        return max;
    }
});                                    // 8
```

06. PRACTICE QS :

01. Check if all numbers in our array are multiples of 10 or not ?

```
let nums = [10,20,30,40];
let answer = nums.every ((el) => {
    el % 10 == 0;
});
console.log(answer);                   // true
```

Note :— if any nums in an array is not multiple of 10. Eg :— [10, 20, 30, 40, 69] // false.

02. Create a function to find the minimum number in an array.

```
let nums = [10, 20, 30, 40, 5];
let min = nums.reduce ((min, el) => {
    if (min < el) {
        return min;
    } else {
        return el;
    }
```

```
console.log(min);   // 5.
```

[or]

We can write as a function

```
function getMin(nums) {
    let min = nums.reduce((min, el) => {
        if (min < el) {
            return min;
        } else {
            return el;
        }
    });
    return min;
}
let nums = [10, 20, 30, 40, 5];
getMin(nums);   // 5
```

Eg:- `getMin([10, 2, 3, 1, 20]);` // 1.

07. Default Parameters :

- Giving a default value to the arguments.

```
function func(a, b = 2) {
    // do something
}
```

```
function sum(a, b = 3) {
    return a + b;
}
sum(2);   // 5.
```

Note:-

```
function sum(a=2, b) {
    return a+b;
}
sum(1,3);   // 4
sum(1);     // a=1, b= undefined / NaN
```

08. Spread :

- Expands an iterable into multiple values.

```
function func(...arr) {
    // do something
}
```

Eg:-

01. `console.log(..."apnacollege");`
    // a p n a c o l l e g e

02. `let arr = [1,2,3,4,5];`
    `Math.min(...arr);` // 1
    `console.log(...arr);` // 1 2 3 4 5

Note:- Instead of "Math.min(arr[0], arr[1], ...)" code used spread "Math.min(...arr);

09. Spread Array Literals :

- `let arr = [1,2,3,4,5];`
  `let newArr = [...arr];`
  `newArr;` // [1,2,3,4,5];

- `let chars = [..."hello"];`
  `chars;` // ['h','e','l','l','o'];

- let odd = [1, 3, 5, 7, 9];
  let even = [2, 4, 6, 8, 10];
  let nums = [...odd, ...even];
  // 1, 3, 5, 7, 9, 2, 4, 6, 8, 10.

10. Spread Object Literals:

- let data = {
    email: "roshan@gmail.com",
    password: "abcd",
  };
  let dataCopy = {...data, id: 123};

11. Rest: (Opposite of spread)

- Allows a function to take an indefinite number of arguments and bundle them in an array.

- function sum(...args) {
    return args.reduce((add, el) => add + el);
  }

Eg-01:—

- function sum(...args) {
    for(let i=0; i<args.length; i++) {
      console.log("You gave us:", args[i]);
    }
  }
  sum(1, 2); // 1
  2
  sum(1) // 1.

arguments

Eg - 02 :

```
function min (...args) {
    return args.reduce ((min, el) => {
        if (min > el) {
            return el;
        } else {
            return min;
        }
    });
}
min(1, 10, -2, 9); // -2
```

12. Destructuring :

- Storing values of array into multiple variables.

```
let names = ["tonyp", "bruce", "steve", "peter"];
let [winner, runnerUp] = names;
console.log(winner, runnerUp);
// tonyp, bruce
```

```
let names = ["Vinay", "Konda", "abc", "xyz", "pyq"];
let [winner, runnerup, ...others] = names;
console.log(others); // abc, xyz, pyq
```

Note :— Remaining elements are stored in "others" because we used rest concept.

13. Destructuring Objects :

```
const student = {
```

```
name : "Vinay",
class : 9,
age : 24,
subjects : ["Hindi", "English", "Math", "Science"],
username : "Vinay@123",
password : 1234,
};

const { username : user , password : pass } = student;

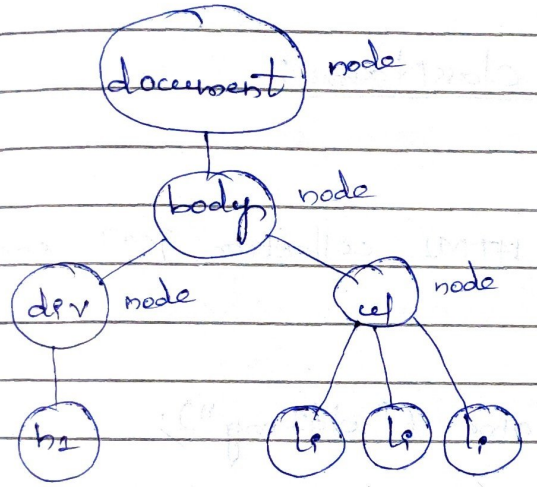
console.log(user) ; // Vinay@123.
```

Javascript Part 08

01. Introduction :—

- DOM (Document Object Model) :
- The DOM represents a document with a logical tree.
- It allows us to manipulate / change webpage content (HTML elements).

02. What is DOM :—




```
<body>
  <div>
    <h1> Todo </h1>
  </div>
  <ul>
    <li> Eat </li>
    <li> Code </li>
    <li> sleep </li>
  </ul>
</body>
```

03. Download Starter Code :

1) Select (prog)
2) changes / Manipulation.

05. Selecting Elements :

- getElementById
  - Returns the element as an object [or] null (if not found).

Eg:- `document.getElementById("id");`

⇒ `let progObj = document.getElementById("id");`

⇒ `console.dir(progObj);`

• By above step we get objects inside it.

imgObj.src = "assets/ png";

• This will change the image.

Note:- • We use "getElementById" to access a particular id for the manipulation.

06. Selecting Elements by className:

• getElementsByClassName:
Returns the elements as an HTML collection [or] empty collection (if not found).

• document.getElementsByClassName("oldImg");

Eg:- Changing the Images to one single image.

• let smallImages = document.getElementsByClassName("oldImg");

for (let i=0; i<smallImages.length; i++) {
smallImages[i].src = "assets/spiderman-img.png";
console.log(`Value of image no. ${i} is changed.`);
}

07. Selecting Elements by TagName:

• getElementsByTagName:
Returns the elements as an HTML Collection [or] empty collection (if not found).

. To change text :—

• document.getElementsByTagName("p").innerText = "abcs";

08. Query Selectors :

Allows us to use any CSS Selectors.

• document.querySelector('p'); // Selects first p element.

• document.querySelector('#myId');
// selects first element with Id = myId.

• document.querySelector('.myclass');
// selects first element with class = myclass.

• document.querySelectorAll("p"); // Selects all p elements.

09. Setting Content on Objects :

• Using properties and Methods :

• innerText
- Shows the visible text contained in a node.

• textContent
- Shows all the full text (even hidden text)

• innerHTML
- Shows the full markup.

*
*

* Manipulation of properties :—

- para.innerText = "abc";
  // abc

- para.innerText = "Hi, I am Peter
  // Hi, I am Peter
  // This does not change 'peter' to bold text.

- para.innerHTML = "Hi, I am Peter
  // Hi, I am Peter
  // This changes 'peter' to bold text.

Eg:-

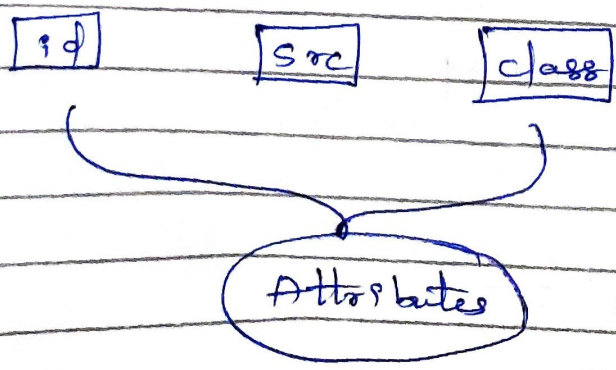
- let heading = document.querySelector('h1');
  // <h1> Spider Man </h1>

heading.innerHTML = `<u> ${heading.innerText} </u>`;

10. Manipulating Attributes :-

- obj.getAttribute(attr)
- obj.setAttribute(attr, val)

Eg:- let img = document.querySelector('img');
img.getAttribute('id');
// mainImg
img.setAttribute('id', 'SpiderManImg');



11. Manipulating Style (With style Attribute):

- style Property
- obj.style

Eg:-
```
• let heading = document.querySelector('h2');
  heading.style.color = 'purple';
  // changes the heading color to purple.
```

Eg:- Changing color of hyperlinks [or] 'a' attributes

```
• let links = document.querySelectorAll(".box a");
  for (let i=0; i< links.length; i++) {
      links[i].style.color = "green";
  }
```

[or]

```
for (link of links) {
    link.style.color = "purple";
}
```

Note:- Style property only used for inline styles

12. ClassList Property:

using classList

→ obj.classList

- classList.add() to add new classes
- classList.remove() to remove classes
- classList.contains() to check if class exists
- classList.toggle() to toggle between add & remove.

Eg:-
```
• heading.classList.add("abc");
```

13. Navigation on Page :

- Parent Element
- children
- previous Element Sibling / next Element Sibling.

Eg :- • let h4 = document.querySelector('h4');
h4.parentElement ;
// ...

Note :- • box.childElementCount; // 2
• It shows the no. of children of a parent.

Eg 02 :-
• let img = document.querySelector('img');
img.previousElementSibling;
// <h1> Spider Man </h1>
img.previousElementSibling.style.color = "green";

14. Adding Elements on Page :-

• document.createElement('p') // creates new element
• appendChild(element) // Adds elements to the end
• append(element) // makes changes to existing elements.
• prepend(element) // adds elements to the start
• insertAdjacent(where, element)

Eg :-
• let newP = document.createElement('p');
console.dir(newP); //
newP.innerText = "Hi, I am a new P";
let body = document.querySelector('body');
body.appendChild(newP);
// Hi, I am a new P

Note :- append means end. "This will create new <P> in the end"

14. insertAdjacentElement (position, element)
- position
  - beforebegin // Before the target Element itself.
  - afterbegin // Just inside target Element, before its first child.
  - beforeend // Just inside the target Element, after its last child.
  - afterend // After the target Element itself.

15. Removing Elements from Page :
- removeChild (element).
- remove (element)

Eg:-
- let body = document.querySelector('body');
  body.removeChild('btn');
  // <button> New Button </button>

[or]

- btn.remove();
- p.remove();

16. Practice Qs :
Q1. Add the following elements to the container using only javascript and the DOM methods.
i) a <p> with red text that says "Hey I'm red!"
ii) an <h3> with blue text that says "I'm a blue h3!"
iii) a with a black border and pink background color with the following elements inside of it.
- another <h1> that says "I'm in a div".
- a <p> that says "ME TOO!".

i) • let para1 = document.createElement("p");
para1.innerText = "Hey I'm red";
document.querySelector("body").append(para1);
para1.classList.add("red");

CSS :—

```
.red {
  color: red;
}
.blue {
  color: blue;
}
```

} for i) & ii)

ii)

• let h3 = document.createElement("h3");
h3.innerText = "I'm a blue h3";
document.querySelector("body").append(h3);
h3.classList.add("blue");

iii)

• let div = document.createElement("div");
let h1 = document.createElement("h1");
let para2 = document.createElement("p");

h1.innerText = "I'm in a div";
para2.innerText = "ME TOO!";

div.append(h1);
div.append(para2);
div.classList.add("box");

for iii)

CSS

```
.box {
  border: 1px solid black;
  bg-c : pink;
}
```

document.querySelector("body").append(div);

Note:— • We can use 'prepend' instead of 'append'.

Javascript Part 20

01. DOM Events :

- Events are signals that something has occurred. [user inputs/ actions]

Inline Events :— [In index.html]

Eg:— <button onclick = "console.log("clicked")"> click me </ button>

02. Mouse and Pointer Events :

- onclick (when an element is clicked)
- onmouseenter (when mouse enters an element).

01) onclick :—

- index.html

<button> click me </button>

- app.js

```
let btn = document.querySelector("button");
btn.onclick = function() {
    console.log("The button was clicked");
};
```

[or]

- You can create a function and assign it to onclick

```
function sayHi() {
    console.log("Hi!");
};
btn.onclick = sayHi;
```

Eg:—

- When we have multiple buttons =>

```
let btns = document.querySelectorAll("button");
for (btn of btns) {
    btn.onclick = sayHi;
}
function sayHi() {
```

```
    console.log("Hi!!");
};
```

Note:— We don't use () parantheses when assigning a function name to onclick.

02) onmouseenter :—

- When a mouse / cursor hovers over a button you can trigger an event.

Eg:—

```
btn.onmouseenter = function() {
    console.log("Hover");
};
```

03. Event Listeners :—

- addEventListener
- element.addEventListener(event, callback)

```
btn.addEventListener("click", function() {
    console.log("button clicked");
});
```

Note:— In onclick it cannot take more than one function, so for this we use "addEventListener".

Eg:—

```
let btns = document.querySelectorAll("button");
for (btn of btns) {
    // btn.onclick = sayHello;
    // btn.onclick = sayName;
```

```
btn.addEventListener("click", sayHello);
btn.addEventListener("click", sayName);
}

function sayHello() {
    alert("Hello!");
};

function sayName() {
    alert("Apna College");
};
```

04. Activity :

- html :—

```
<h3> Generate a random colour </h3>
<button> Generate a Color </button>
<div> This is your new Color </div>
```

- js :—

```
let btn = document.querySelector("button");
btn.addEventListener("click", function() {
    let h3 = document.querySelector("h3");
    let randomColor = getRandomColor();
    h3.innerText = randomColor;

    let div = document.querySelector("div");
    div.style.backgroundColor = randomColor;
});

function getRandomColor() {
    let red = Math.floor(Math.random() * 255);
    let green = Math.floor(Math.random() * 255);
    let blue = Math.floor(Math.random() * 255);
```

```
    let color = `rgb(${red}, ${green}, ${blue})`;
    return color;
}
```

05. Event Listeners for Elements :—

- `<p> This is a paragraph </p>`
- `<div class="box"> </div>`

```
let p = document.querySelector("p");
p.addEventListener("click", function() {
    console.log("Parab was clicked");
});

let box = document.querySelector(".box");
box.addEventListener("mouseenter", function() {
    console.log("Mouse inside div");
});
```

06. This in Event Listeners :—

- When 'this' is used in a callback of event handler of something, it refers to that something.

Eg :—

```
let btn = document.querySelector("button");
let p = document.querySelector("p");
let h1 = document.querySelector("h1");
let h3 = document.querySelector("h3");

function changeColor() {
    console.dir(this.innerText);
    this.style.backgroundColor = "blue";
}
```

```
btn.addEventListener("click", changeColor);
p.addEventListener("click", changeColor);
h2.addEventListener("click", changeColor);
h3.addEventListener("click", changeColor);
```

07. Keyboard Events :—

- In keyboard events we will have an default event called as "event". [or] "e"

```
let btn = document.querySelector("button");
btn.addEventListener('click', function(event) {
    console.log(event);
    console.log("Button is clicked");
});
```

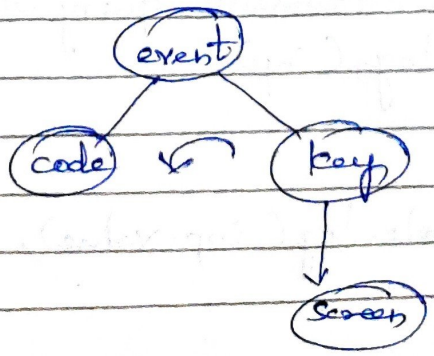
→ (1)

- Main events :—
- keydown — When a key is pressed on the keyboard.
- keyup — fired when a key is released.
- keypress —

Eg:->

```
<input placeholder = "Search" />
let inp = document.querySelector("input");
inp.addEventListener("keydown", function() {
    console.log("key is pressed");
});
```

- When an event is trigged



Eg-02:

```
let inp = document.querySelector("input");
inp.addEventListener("keydown", function(event) {
    console.log(event.key);
    console.log(event.code);
    console.log("key was pressed");
});
```

08. Form Events :

```
<form action="/action">
    <input placeholder="Username" /> <br> <button> Register </button>
</form>
```

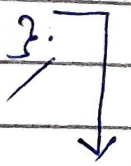
```
let form = document.querySelector("form");
form.addEventListener("submit", function(event) {
    event.preventDefault();
    alert("form submitted");
});
```

Note :-

- We use "preventDefault" to stop the default activity when submitting a form.

09. Extracting form Data :

```
let form = document.querySelector("form");
form.addEventListener("submit", function(e) {
    e.preventDefault();
    let inp = document.querySelector("input");
    console.log(inp);
});
```



```
console.log(inp.value);
```

```
<form action = "/action">
    <input type = "text" id = "user">
    <input type = "password" id = "pass">
    <button> submit </button>
</form>
```

```
let form = document.querySelector("form");
form.addEventListener("submit", function (event) {
    event.preventDefault();

    let user = document.querySelector("#user");
    let pass = document.querySelector("#pass");

    console.log(user.value);
    console.log(pass.value);
});
```

[or]

Note :- Instead of accessing values using input types we can access using "form" elements

```
let form = document.querySelector("form");
form.addEventListener("submit", function (event) {
    event.preventDefault();

    console.log(form.elements[0].value);
    console.log(form.elements[1].value);
});
```

Note :- Instead 'form' in console.log we can use 'this'.

10. More Events :

01. change event :—

- The change event occurs when the value of an element has been changed (only works on <input>, <textarea> and <select> elements).

02. input event :—

- The input event fires when the value of an <input>, <select>, [or] <textarea> element has been changed.

Eg :—

- 01) let form = document.querySelector("form");
  form.addEventListener("submit", function (event) {
  event.preventDefault();
  });

let user = document.querySelector("#user");
user.addEventListener("change", function() {
console.log("input changed");
console.log("final value =", this.value);
});

- 02) let user = document.querySelector("#user");
  user.addEventListener("input", function() {
  console.log("input event");
  console.log("final value =", this.value);
  });

Activity:

```
<div class="container"> </div>
<input type="text">

let inp = document.querySelector("input");
let container = document.querySelector(".container");
inp.addEventListener("input", handleTyping);

function handleTyping() {
    container.innerText = inp.value;
};
```

Javascript Part 11 :-

01. Event Bubbling :

- ```

      <ul>
          <li> one </li>
          <li> two </li>
          <li> three </li>
      </ul>

  ```

- ```
  div {
      bg-color : pink;
      h : 200 px;
      w : 400 px;
  }
  ul {
      bg-color : blue;
  }
  li {
      bg-color : green;
  }
  ```

- ```
  let div = document.querySelector ("div");
  let ul = document.querySelector ("ul");
  let lis = document.querySelectorAll ("li");

  div.addEventListener ("click", function() {
      console.log ("div was clicked");
  })

  ul.addEventListener ("click", function() {
      console.log ("ul was clicked");
  })

  for (li of lis) {
      li.addEventListener ("click", function() {
          console.log ("li was clicked");
      });
  }
  ```

Note :- when we click on nested element it triggers its parent elements.
  ```

- To stop Event Bubbling we have to use →
  event.stopPropagation();

02. Building Todo with DOM :—

- <h1> TODO App </h1>
  <input type = "text" placeholder = "Add List" >
  <button> Add Task </button>

- ul li {
    list-style-type : none;
    margin-top : 5px;
  }
  .delBtn {
    margin-left : 5 px;
  }

- const inp = document.queryselector ("input");
  const btn = document.queryselector ("button");
  const ul = document.queryselector ("ul");
  const list = document.queryselector ("li");
  const delBtn = document.queryselectorAll ("delBtn");

// Displaying list-item

btn.addEventListener ("click", function () {
  let val = inp.value;
  const el = document.createElement ("li");
  el.innerText = val;
  ul.appendChild (el);
  inp.value = "";

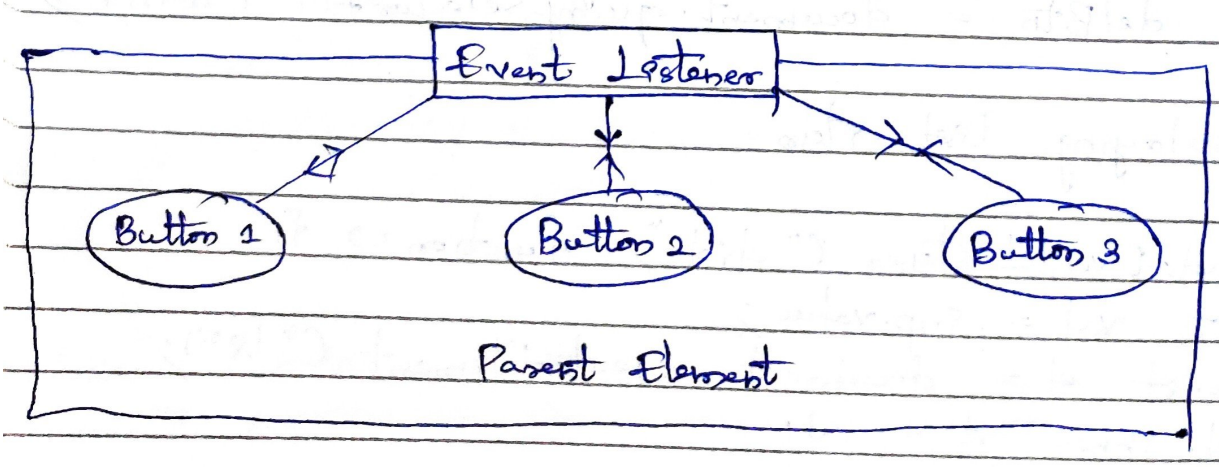
```
// creating delete Button
const renBtn = document.createElement("Button");
    renBtn.classList.add("delBtn");
renBtn.innerText = "Delete";
    el.appendchild(renBtn);
});

// functionality of delete button
// Event Delegation

ul.addEventListener("click", function(event) {
    if ((event.target.nodeName == "BUTTON")) {
        let listItem = event.target.parentElement;
        listItem.remove();
    } else {
        console.log("Error deleting a listItem");
    }
});
```

03. Event Delegation:

- A Technique in Javascript where a single event listener is attached to a parent element instead of attaching event listeners to multiple child elements.



- In this case we'll use "event.target".

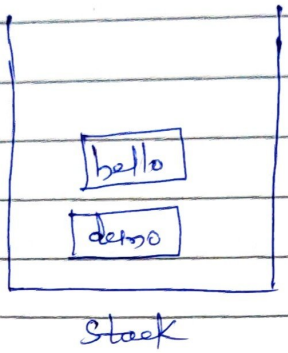
Eg:- console.dir(event.target);

Javascript Part 12

01. JS Call stack :

- The call stack works based on the last-in-first-out (LIFO) principle.

```
hello() {
}
demo() {
  hello();
}
demo();
```



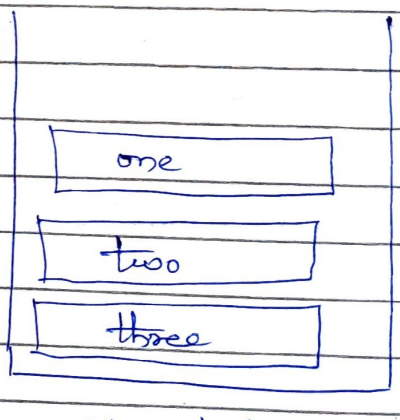
function calls

02. Visualizing the Call stack :

```
function one() {
  return 1;
}

function two() {
  return one() + one();
}

function three() {
  let ans = two() + one();
  console.log(ans);
}
three();
```



// 3

03. Breakpoints :—

- To know the call stack we use breakpoints [or] debugger.

04. JS is single threaded :

```
setTimeout (function () {
    console.log("Apna college");
}, 2000);

console.log ("hello...");
```

Note :—

- Javascript is synchronous in nature but it acts as asynchronous in "Callbacks".

05. Callback Hell :

- Callback hell, also known as the "Pyramid of doom", occurs in JS when we have multiple nested callbacks.

```
function task1() {
    setTimeout (() => {
        console.log("Task 1 complete");
    }, 2000);
}

function task2() {
    setTimeout (() => {
        console.log("Task 2 complete");
    }, 1000);
}

function task3() {
    setTimeout (() => {
        console.log ("Task 3 complete");
    }, 3000);
}
```

```
function task4() {
    setTimeout(() => {
        console.log("Task 4 is complete");
    }, 1500);
}

task1();
task2();
task3();
task4();
console.log("All tasks complete");
```

- To Execute everything in sequence we use =>

```
function task1(callback) {
    setTimeout(() => {
        console.log("Task 1 complete");
        callback();
    }, 2000);
}
function task2(callback) {
    setTimeout(() => {
        console.log("Task 2 complete");
        callback();
    }, 1000);
}
function task3(callback) {
    setTimeout(() => {
        console.log("Task 3 complete");
        callback();
    }, 3000);
}
```

```
function task4(callback) {
    setTimeout(() => {
        console.log("Task 4 complete");
        callback();
    }, 1500);
}

task1(() => {
    task2(() => {
        task3(() => {
            task4(() => { console.log("All tasks complete"); });
        })
    });
})
```

(and)

- `h1 = document.querySelector("h1");`

```
function changeColor(color, delay, nextColorChange) {
    setTimeout(() => {
        h1.style.color = color;
        if (nextColorChange) nextColorChange();
    }, delay);
}

changeColor("red", 1000, () => {
    changeColor("orange", 1000, () => {
        changeColor("green", 1000, () => {
            changeColor("yellow", 1000)
        });
    });
});
```

Note:—

- To get rid from callback hell we use 'promises' and "async & await".

06. Setting up for promises:—

- Promises:

The promise object represents the eventual completion (or failure) of an asynchronous operation and its resulting value. →

```
function saveToDb(data, success, failure) {
    let internetSpeed = Math.floor(Math.random() * 10) + 1;
    if(internetSpeed > 4) {
        success();
    } else {
        failure();
    }
}

saveToDb("Todo", () => {
    console.log("Your data saved :)");
},
() => {
    console.log("Weak connection. Data not saved");
}
);

saveToDb("Hello Db", () => {
    console.log("Success 2 : data 2 saved");
},
() => {
    console.log("failure 2: weak connection");
}
);
```

```
saveToDb ("Vinay", () => {
    console.log("Success 3");
},
() => {
    console.log("Failure 3");
});
```

07. Refactoring with promises :

- Promises → object
  Resolve & reject
  ↓ success
  ↓ failure

State → pending
failure
fullfilled

```
• function saveToDb (data) {
    return new Promise ((success, failure) => {
      let internetSpeed = Math.floor (Math.random()*10)+1;
      if (internetSpeed > 4) {
        success();
      } else {
        failure();
      }
    });
  }
```

• Note :— We will use resolve and reject instead of success & failure.

```
saveToDb("Todo").then(() => {
    console.log("Promise was resolved");
    // console.log(request);
})
.catch(() => {
    console.log("Promise was rejected");
});
```

→ ↓

08. Then() and Catch() Methods:

- Promises
  - then() & catch()

```
let request = saveToDBPromise("Todo");
request
    .then(() => {
        console.log("Promise resolved");
    })
    .catch(() => {
        console.log("Promise rejected");
    });
```

09. Promise Chaining:

Promises:

- Improved Version:

```
saveToDBPromise("Todo")
    .then(() => {
        console.log("Promise 1 resolved");
        return saveToDBPromise("Hello");
    })
    .then(() => {
        console.log("Promise 2 resolved");
    })
    .catch(() => {
        console.log("Some promise rejected");
    });
```

Note:—

- We can use multiple 'then' and only one 'catch'.

```
saveToDb("Hello").then(() => {
    console.log("Data2 saved");
});
```

[or]

```
return saveToDb("Hello");
})
.then(() => {
    console.log("Data2 saved");
})
```

10. Results and Errors in Promises:

Promises

Promises are rejected and resolved with some data (valid results [or] errors).

```
saveToDBPromise("Todo");
.then((result) => {
    console.log("result :", result);
    console.log("Promise 1 resolved");
    return saveToDBPromise("Hello");
})
.then((result) => {
    console.log("result :", result);
    console.log("Promise 2 resolved");
})
.catch((error) => {
    console.log("error:", error);
    console.log("Some promise rejected");
});
```

11. Refactoring old Code :

- **Promises :**

Let's apply promises to our callback hell.

```
h1 = document.querySelector("h1");

function changeColor(color, delay) {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            h1.style.color = color;
            resolve("color changed!");
        }, delay);
    });
}

changeColor("red", 1000)
    .then(() => {
        console.log("red is done");
        return changeColor("orange", 1000);
    })
    .then(() => {
        console.log("Orange is done");
        return changeColor("green", 1000);
    })
    .then(() => {
        console.log("green is done");
        return changeColor("blue", 1000);
    })
    .then(() => {
        console.log("blue is done");
    });
```

Javascript Part 13

01. Async functions :

- async and await keywords

- **Async keyword :**
Creates an Async function.

```
async function greet() {
    return "Hello";          // returns a promise
}
```

```
let hello = async () => { };   // returns a promise.
```

Note :- We can create errors using 'throw' keyword.

```
async function greet() {
// throw "some random error";
    return "hello";
}

greet()
.then(() => {
    console.log("Promise was resolved");
})
.catch((err) => {
    console.log("Error");
})
```

02. Await keyword :

- Pauses the execution of its surrounding async function until the promise is settled (resolved or rejected).

- async function showc() {
    await colorchange ("violet", 1000);
    await colorchange ("indigo", 1000);
    await colorchange ("green", 1000);
    await colorchange ("yellow", 1000);
    await colorchange ("orange", 1000);

    return "done";
}

Eg:-

- function getNum() {
    return new Promise ((resolve, reject) => {
        setTimeout (() => {
            let num = Math.floor(Math.random() * 10) + 1;
            console.log(num);
            resolve();
        }, 1000);
    });
}

async function demo() {
    /* await */ getNum();
    /* await */ getNum();
    /* await */ getNum();
}

demo();

- We can use the above concept for 'refactoring old code'

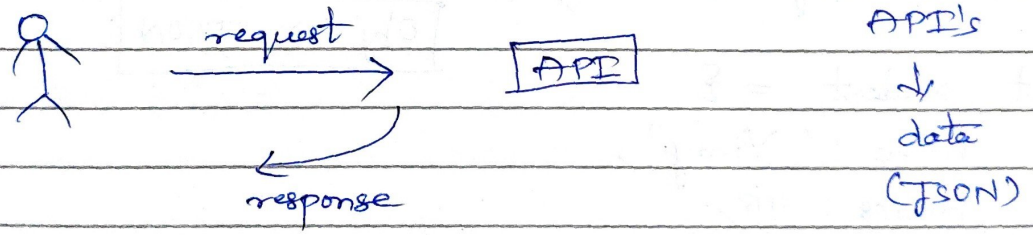
03. Handling Rejections :

- Await Keyword
- Handling Rejections with Await.
- We use try() and catch() to handle errors in await.

```
async function hitApi() {
    try {
        const response = await fetch("/url");
        const json = await response.json();
        console.log(json);
    } catch(e) {
        console.log("Error hitting api request");
    }
}
hitApi();
```

04. What is an API ?

- Application Programming Interface



05. Accessing some API's :

- Some Random API's

→ https://catfact.ninja/fact (sends random cat facts).

→ https://www.boredapi.com/api/activity (sends an activity to do when bored).

→ https://dog.ceo/api/breeds/image/random (sends cute dog pictures)

06. What is JSON :

- Javascript object Notation (www.json.org)

07. Accessing JSON Data :

- Accessing Data from JSON

- JSON.parse(data) Method

→ To parse a string data into a JS object

- JSON.stringify(json) Method

→ To parse a JS object data into JSON.

Eg:- JSON → obj

- let jsonRes = '{"fact": Something.", "length": 10}';

let validRes = JSON.parse(jsonRes);
console.log(validRes);

Obj → JSON

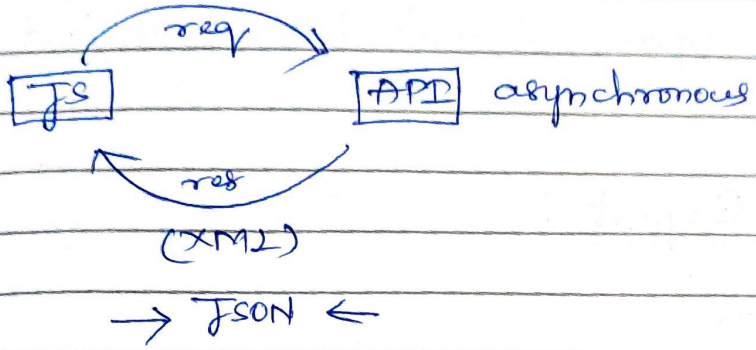
- let student = {
name : "Vinay",
marks : 95,
};
JSON.stringify(student);

08. API Testing Tools :

- Tools
- Hoppscotch
- Postman

09. What is Ajax ?

- Asynchronous Javascript and XML



10. Http Verbs :

Examples :

- GET – to get data from API
- POST – to post data to API
- DELETE – to delete data.

11. Status Codes :

Examples :

- 200 – OK
- 404 – Not found
- 400 – Bad Request
- 500 – Internal Server Error.

12. Adding Information in URL's :

- Query Strings
- https://www.google.com/search?q=harry+porter

Key ↑ Value

- ?name=Vinay&marks=95

13. Https Headers :

- Header, value
- Supplies additional information
- It gives API's response formats like JSON, text, HTML

14. Our first API Request :

- let url = "/url";

```
fetch(url)
  .then((res) => {
      return res.json();
  })
  .then((data) => {
      console.log("data 1 = ", data);
      return fetch(url);
  })
  .then((res) => {
      return res.json();
  })
  .then((data2) => {
      console.log("data 2 = ", data2);
  })
  .catch((err) => {
      console.log("ERROR -", err);
  })

// console.log("I am good");
```

15. Using fetch with Async Await :

- fetch with async / await

```
async function getfacts() {
    try {
        let res1 = await fetch(url);
        let data1 = await res1.json();
        console.log("data 1 - ", data1);

        let res2 = await fetch(url);
        let data2 = await res2.json();
        console.log("data 2 - ", data2);
    } catch (e) {
        console.log("Error :", e);
    }
}

getfacts();
```

Javascript Part 14

01. Using Axios:

- Library to make HTTP requests

```
async function getFacts() {
    try {
        let res = await axios.get(url);
        console.log(res);        // res.data.fact
    } catch(e) {
        console.log("Error", e);
    }
}
```

Note:—

- Using Axios we don't need to parse.

Eg:— Activity:

```
<h1> Get random facts </h1>
<button> New fact </button>
<div id="result"> </div>
```

```
let btn = document.querySelector("button");

btn.addEventListener("click", async () => {
    let fact = await getFacts();
    let p = document.querySelector("#result");
    p.innerText = fact;
});

let url = "/url";

async function getFacts() {
    try {
        let res = await axios.get(url);
        return res.data.fact;
```

```
} catch (e) {
    console.log("error");
    return "No fact found";
  }
}
```

02. Dog Pictures API :

```
let btn = document.querySelector("button");
let url = "/url";

btn.addEventListener("click", async () => {
    let link = await getImage();
    let img = document.querySelector("#result");
    img.setAttribute("src", link);
});

async function getImage() {
  try {
      let res = await axios.get(url);
      return res.data.message;
    } catch (e) {
      console.log("Error", e);
      return "/";
    }
}
```

03. Sending Headers with API Requests :

- Sending Headers

```
const config = { headers: { Accept: "application/json" } };
let res = await axios.get(url, config);
console.log(res.data);
```

- const url = "https://icanhazdadjoke.com/";

```
async function getJokes() {
    try {
        const config = {
            headers : { accept : "application/json" }
            };
        let res = await axios.get(url, config);
        console.log(res.data);
    } catch (err) {
        console.log(err);
    }
}
```

04. Activity using Query strings:

- Axios:
- Updating Query strings
- let url = "http://universities.hipolabs.com/search?name="
  let btn = document.querySelector("button");

```
btn.addEventListener("click", async () => {
    let country = document.querySelector("input").value;
    let colArr = await getColleges(country);
    show(colArr);
});

function show(colArr) {
    let list = document.querySelector("#list");
    list.innerText = "";
    for(col of colArr) {
        console.log(col.name);
```

```
    let li = document.createElement("li");
    li.innerText = col.name;
    list.appendChild(li);
  }
}

async function getColleges(country) {
  try {
    let res = await axios.get(url + country);
    return res.data;
  } catch (e) {
    console.log("Error", e);
    return [];
  }
}
```

- Import and Export in JS :-

// Named export
- export const fname = "Konda";
// Default export
- function name() {
    return "Vinay";
  }
  export default name;

- export.js :-

- import.js :-

```
import name from "./export.js";   // Default
import { fname } from "./export.js"  // Named.
console.log(fname);
let res = name();
console.log(res);
```

Note:- You have to set type="module" in package.json.