

INTRO

- * use script tag in html doc for javascript
- * Another way is to link javascript using href.

→ variable
* `name = prompt("What's your name");` (Pop up will take user I/p)

`document.getElementById('welcome').innerHTML = name;`
↳ (inserts I/p in id welcome) ↓ use to insert in HTML file

→ To open JS console → Ctrl + Shift + J

→ `prompt("");` (for user I/p)

→ In console you can perform arithmetic function.

→ Type → document (in console)

↳ (It will provide html source page where you can test your code)

CONSOLE

* console.log(' '); (Prints sp only in console log)

* console.log([1,2,3]);

	[1,2,3]
0:	1
1:	2
2:	3
length:	3
proto:	Array

* console.table([1,2,3]);

	Creates table with index & element/value
--	--

* console.error("bops! something went wrong");

Pops an error in console log.

* console.clear();

↳ cleans console log

* console.warn('A Warning...');

↳ prints yellow warning in console log

* console.time('Test');

(code blocks)

console.timeEnd('Test')

Calculate; test of running of code

Test: (time)ms

⇒ JAVASCRIPT VARIABLES;

var name = 'Yash';

↳ (dynamic variable assigning)
↳ can be integer, float, char, string, boolean

* var assigning is irrespective of special cases or numbers but spaces are not allowed.

* strings must be single (' ') or double (" ") quoted

→ let keyword ; let name = 'Yash';
let name = 'Jain'; } Error

* You can't re-declare let keyword whereas re-assign it again

Overcoming → (name = 'Jain');
name variable

STRINGS

- * Const Variables; + You have to assign value while declaring whereas in let variable you can assign value to a variable afterwards.
- * Cannot re-assign the value to a variable.
- * Cannot re-declare the const variable

* `(let name = 'Yash';
console.log(typeof name);)` (Prints String)

* Data type in Javascript is defined by value not by variables

→ `variable = Symbol('My Symbol');`
`console.log(typeof variable);` (Prints symbol)

- * Few data types are -:
- 1) Integer Number (Float)
 - 2) Character
 - 3) String
 - 4) Boolean
 - 5) ~~Null~~ Null / undefined
 - 6) Symbol

* The input while using prompt is always a string

* `let variable = null;`
`console.log(typeof variable);`) Prints → object
(treats variable as an object)

* `let lang = ['HTML', 'JS', 'PHP'];`
`console.log(typeof lang)`) Prints → object
(arrays are objects in JS)

* `let person2 = {
name: 'Juan',
country: 'USA'}` } object defining

* `string = 'Hey! What's up';`
to add ('')

* To concatenate string use '+'

* If you assign a value to a variable not declared JS automatically creates it e.g. global variable → `car = 'Volvo';`
↳ (car is globally declared)

(12) `str = " I am Yash "`;
`txt = str.trim()`;
 ↳ trims the white spaces

Methods in Javascript Strings

① length;

→ (variable.length)

→ o/p is no. of char/no in strings

② concatenate;

→ (variable.concat(" ", " -- "));
 ↳ concatenates string to variable

③ uppercase;

→ (variable.toUpperCase());
 ↳ Makes whole string caps.

④ lowercase;

→ (variable.toLowerCase());
 ↳ Makes whole string small.

⑤ Index;

→ (variable.indexOf(' '));
 ↳ Tells the position of char/string in variable string.

(Starts the search from 5 index)

indexOf(' ', 5);

• `substr(start, length)`; slices up to length
 ↳ If length is not then slices till end

* If the string is not present indexOf will

⑥ Substring;

→ (variable.substring(0, 10))
 ↳ Copies (0, 10) indices to the var from variable

from the end

→ (variable.substring(learning.length - 1, learning.length))
 ↳ Copies last 8 character.

⑦ Slice;

→ (variable.slice(0, 10));

↳ copies substring upto 10 in

from end

→ (variable.slice(-6));


↳ prints last 6

if str is not then

⑧ Split;

→ (variable.split(' '));

↳ Splits the spaces from variable and returns an array object

 Sahitya Bhawan
 ex → search(' ');

Eg: name = "Hello This is Yash";
output = name.split(' ');
console.log(output); ↳ split spaces
 ↳ (array of 4 words)

① replace;

↓
(As case sensitive) → (variable.replace('Present text', 'Add Text'))
→ Add Text will ~~also~~ replace Present text
to make case insensitive [/Yash/i] searches for Yash i irrespective of case. [(Yash/g) → replaces Yash (all) from the string.]

6) Includes;

- variable includes (' ');
- checks if word/string is present in variable returns boolean (T/F)

12) Repeat;

→ variable-repeat(4);
→ Repeats variable string 4 times

(13) lastIndexOf; `str.lastIndexOf(" ");`
(searches from end to beginning) → returns the position of the element where it appears last time in string.

Numbers

* Math function;

- 1) $\text{Math.PI} \rightarrow 3.14 \dots$
- 2) $\text{Math.round}(2.4) \rightarrow$ round to nearest integer
- 3) $\text{Math.ceil}(2.2) \rightarrow 3$
- 4) $\text{Math.floor}(2.9) \rightarrow 2$
- 5) $\text{Math.sqrt}(4) \rightarrow 2$
- 6) $\text{Math.abs}(-3) \rightarrow 3$
- 7) $\text{Math.pow}(2, 8) \rightarrow 2^8$
- 8) $\text{Math.min}(10, 8, 2, 1, 7, 0); \rightarrow 0$
- 9) $\text{Math.max}(10, 8, 2, 1, 7, 0); \rightarrow 10$
- 10) $\text{Math.cos}(\text{radians});$ (degree $\times \pi/180$)

* Arithmetic operations follow BODMAS

* score++ (Post increment)
++score (Pre increment)

↓
Increase &
then evaluate

Evaluate & then increase

COMPARISON OPERATORS

* >, <, >=, <=, ==, != (not equal)

* Returns boolean type after performing comparison

A-Z → 65-91
a-z → 97-123
→ '2' > 'a' (false)
→ 2 > 0 (true)

* Strict Comparison operator; (Also check typeof)

→ 2 == '2' (true)
→ 2 === '2' (false)
↓ ↓
Number String

→ 2 = '2' (error)
↳ (Assignment operator)

→ null == undefined (true) (both are not assigned any value)
null === undefined (false)
↓ ↓
(typeof → object) undefined

Convert Strings

→ Number(variable)

↳ Converts variable to a Number
(Only if string contains no)

→ parseInt(variable)

variable = 'nine';
Number(variable); → (NaN)

* Number(true/false) → 0/1

* parseInt(variable, base);

* parseFloat(variable, base);

* (Number(2.8712379).toFixed(2))
↳ prints to 2 decimal places
(2.8712)

Convert Numbers

* let n = 12345;

x = String(n); (converts to string)

1) * To convert to string

→ String(variable)

*

let date = new Date();

x = String(date); (x holds date value in string format)

2) * (boolean).toString();
(Object)

→ true.toString();

Template Strings

M-1

const product1 = 'Pizza';

price1 = 30;

product2 = 'Hamburger';

price2 = 40;

let html = '' +

'Item: ' + product1 + '' +

'Price: ' + price1 + '' +

'Item: ' + product2 + '' +

'Price: ' + price2 + '' +

'';

→ append

let app = document.querySelector('#app');

app.innerHTML = html;

M-2

html = '

Item: \${product1}

Price: \${price1}

Item: \${product2}

Price: \${price2}

';

let app = document.querySelector('#app');

app.innerHTML = html;

ARRAYS

① `let num = [10, 20, 30, 40];` (Array declaration)
 Indices \rightarrow (0, 1, 2, 3)
_{in array} \Rightarrow `num[0], num[1], num[2], num[3]`

② `let day = new Array('Sun', 'Mon', 'Tue');`
 (0, 1, 2)

* Array can hold dynamic data types.

Array Methods;

1) `(array.length)`

2) `((Array.isArray(array-name));)` Prints boolean
 T/F

3) You can simply overwrite array indices.
`array[1] = 'New';`

4) `(array.indexOf('New'));`

\rightarrow Returns the index if 'New' is present

\rightarrow Returns -1 if not present

5) `array.push('array');`

\hookrightarrow adds 'array' at last index

6) `array.unshift('array0');`

\hookrightarrow adds 'array0' at first index

(Removes null)

7) `array.splice(3, null, 'array1');`

\downarrow Inserts \downarrow Replaces \downarrow adds 'array'
 at 3rd index null

8) `array.pop();`

\hookrightarrow removes last index element

9) `array.shift();`

\hookrightarrow removes first index element

10) `array.splice(3, 2);`

\hookrightarrow Removes 2 elements starting from index 3.

11) array.reverse();

↳ Reverses array element

12) array.concat(array 2)

↳ concatenates 2 array.

13) array.sort();

↳ Sorts the array in ascending order

sort().

14) array.reverse();

↳ Reverse array element after sorting

15) if an array consist of multi numbers having same values .sort() won't work.

{ let num = [1, 1, 8, 3, 3, 0, 10];

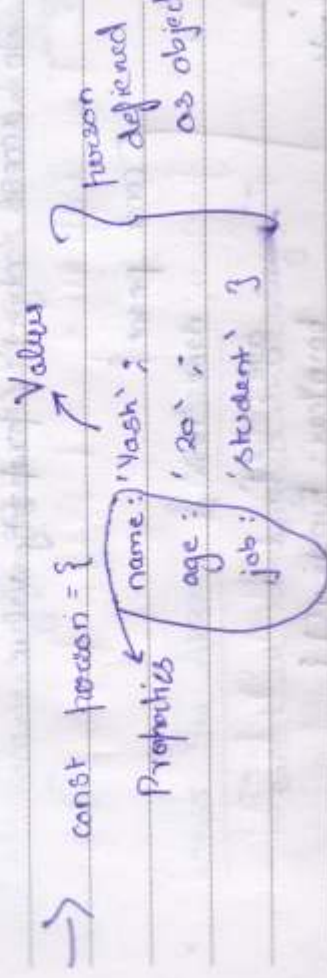
num.sort(function(number1, number2){

return number1 - number2; })

↳ sorting can multiple similar element array

Javascript Objects

- Similar to an array properties;



→ Accessing objects;

pe → object.property

→ object['property']

Example; (person.name) → prints Yash

* object properties can hold array.

eg. → const person {

name: ['Yash', 'Pri', 'Ria'];

* object can be nested;

const person {

address: {

city: 'Mumbai'

state: 'Maharashtra'

person

object property can also hold function declaration

To access object's property value

```
const person = {
  name: 'Yash',
  age: '20',
  bornYear: function() {
    return new Date().getFullYear() - this.age;
  }
}

console.log(person.bornYear());
```

↓
this keyword is used to fetch object property value

Array of objects;

```
let cars = [
  { model: 'Mustang', engine: '6.3',
    year: 2015 },
  { model: 'Ferrari', engine: '6.9',
    year: 2016 },
  { model: 'Challenger', engine: '6.4',
    year: 2017 }
]
```

(cars is an array containing objects)

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

→ prints rows of objects

```
console.log(cars[i].model);
```

→ prints model on each line

CONST IN ARRAYS & OBJECTS;

↳ cannot be re-assigned in strings

```
const new = [1, 2, 3];
```

new[0] = "const can be re-assign in arrays";
 (can also perform array methods)
 (can access them individually only)

```
const car = {
```

```
  name: 'Mustang',
```

```
  engine: '6.4',
```

```
  carName: 'Challenger',
```

```
  car = {
```

```
    name: 'Ferrari',
```

```
    engine: '6.9' }
```

Can assign individually

(X)

* FUNCTIONS

→ function helloJS(); (function Declaration)

→ Func. Definition;

```
function helloJS() {  
  console.log('Hello');  
}
```

→ Func. Calling;

```
helloJS();
```

* Can pass arguments to func.

```
function sum(a, b) {  
  return (a+b);  
}
```

Prints (17) console.log(sum(8, 9));

→ (takes value to func. call)

→ (Passes argument to func.)

* If argument isn't passed it is processed as undefined

```
function hello (fname = 'Yash', lname = '') {  
  console.log('Hello {fname} {lname}');  
  hello();  
}
```

Prints → Hello Yash

(In this method lname will not be underlined)

```
function sum(a = '9', b = '8') {  
  return (a+b);  
}
```

(takes when args are undefined)

```
sum(2);  
sum(, 2);  
sum(1, 2);
```

prints → 10
11
3

Func. EXPRESSION

```

* const sum = function (n1, n2) {
  return (n1 + n2);
}
console.log (sum (3, 8)); // Prints 11

```

* Functions ^{that} are invoked immediately (IIFE's),
 ↓
 Immediately-Invoked function expression

```

* (function() {
  console.log ('Hello, IIFE's!');
})();

```

Declaring IIFE

3) () ; → for passing arguments
 These aren't need to be called

METHODS

```

* Property method: const Player = {
  play: function () {
    console.log ('Playing song');
  },
  pause: function () {
    console.log ('Pause...');
  }
}

```

* Methods can be declared outside objects
 → Player.remove = function (id) {
 console.log ('id', id); removed from playlist
 }
 add to player object

Date Methods

★ `const today = new Date();` → Prints → Present date in default method
`console.log(today);`

★ `const date = new Date('May 15 2020');`
`console.log(date);` → (will be printed)
 ('5-15-2020')

Methods in Date

- 1) `getMonth()` → fetches month
- 2) `getDate()` → fetches date
- 3) `getDay()` → fetches day of week + 7
 (Sun - 17)
- 4) `getFullYear()` → fetches year
- 5) `getHours()` → fetches hours

6) `getMinutes()` → fetches min.

`let today = new Date();`

`(today.setFullYear(2000);`

`output = today.getFullYear());` → (prints 2)

★ We can use set method to all the also
 get Date method for pre-setting data

CONDITION OPERATOR

```
if (condition) {
```

```
code execute; }
```

→ (OR not/undefined)

```
if (false) { → if (0) {
```

```
... }
```

Code won't work

```
if (cond) {
```

```
... }
```

```
else {
```

```
... }
```

Code

format

(only one execute)

```
if (cond) {
```

```
... }
```

```
else if (cond) {
```

```
... }
```

```
else {
```

```
... }
```

Code

format

→ (perform AND operation on both cond.)

```
* if (cond1 && cond2) {
```

```
... }
```

operator → (|| (OR), !(NOT))

Ternary operator

```
let rent = 2000;
```

```
(rent == 2000 ? 'Rent is OK' : 'Rent is high')
```

executes if

↑ true

condition?

(Statement? cond1: cond2)

↓

executes if

false

SWITCH CASE

→ $\left(\begin{array}{l} \text{Compare strictly the value} \\ \text{of 'a' \& definition type} \end{array} \right)$

```
case 'a': console.log('In case a');
```

```
break;
```

```
case 'b':
```

```
console.log('In case b');
```

```
break;
```

```
default:
```

```
console.log('No matching case');
```

```
break;
```

```
}
```

More cleaner in use for multiple cases.

```
switch (1) {
```

```
case 1:
```

```
console.log('Hello');
```

```
case 3:
```

```
console.log('Bye');
```

```
default: console.log('Error');
```

Multiple cases having same constant

FOR LOOP

→ for (Initialize, Condition, Increment) {

```
Code - - - }
```

→ for (let i = 1; i < 11; i++) { prints } → 1-10
console.log('Hi');

→ break → take action out of loop

→ continue → take action to the beginning of loop

→ while / do-while

• while (condition) {

```
code - - - } while structure
```

• do {

```
code - - -
```

```
increment; while (condition);
```

code will work at least once irrespective of condition

ForEach

* array.forEach(function(value, index) {
 console.log(`\${index}: \${value}`);
 })

Print → 0: values

1: 1
 2: 0

MAP

```
const Cart = [
  { id: 1, product: 'Book' },
  { id: 2, product: 'Clothes' },
  { id: 3, product: 'laptop' }
]
```

```
const productName = Cart.map(function(productName) {
  return productName.product;
});
```

array of products

FOR IN (using Iterators)

```
let car = {
```

```
  name: 'Camero',
  engine: 6,
  year: 1879,
  make: 'chevy'
}
```

car object

```
for (let key in car) {
```

```
  console.log(`${car[key]}`);
```

values

```
  console.log(`${key}`);
```

← property of car object

```
}
```


Try / Catch

* `setTimeout (function () {
 console.log ('Completed...');
 }, 2000);`

↓
 timeout in ms
`setTimeout (function () { ... }, time)`

`try {`
 executable code;
`}`

→ (error is code not executed)

`catch (error) {`
`console.log(error);`
`}`

`finally {`
`console.log ('Execute always');`
`}`



Important for un-interrupted code running

WINDOWS OBJECT

→ `console.log` }
 → `prompt` } window object
 → `alert`

• `window.alert ('Hello');` } Both work
`alert ('Hello');`

→ `confirm ('Are you sure');`

↳ Pops a dialog box at top of w
 with 'ok' and 'cancel'
 ↓
 Proceeds else
 (if) Statement

→ `let height = window.outerHeight;` → (takes window
`width = window.outerWidth;` & width)

→ `let h = window.innerHeight;` → (takes dimension of
`w = window.innerWidth;` box)

→ `document.body.style.backgroundColor = 'red';`
 ↓
 tag style property value

* `let url = window.location;`
 \rightarrow `console.log(url);` \rightarrow [is an object containing several url info.]
 \rightarrow `console.log(url.host);` \rightarrow (href, host, port etc.)
 \rightarrow (prints host)

* Redirect using JS;

\rightarrow `window.location.href = 'http://google.com';`

* `let a = 'a', b = 'b', c = 'c';`
`for (var i = 0; i < 10; i++) {`
`console.log(a);`
`console.log(a) \rightarrow (prints \rightarrow 10)`

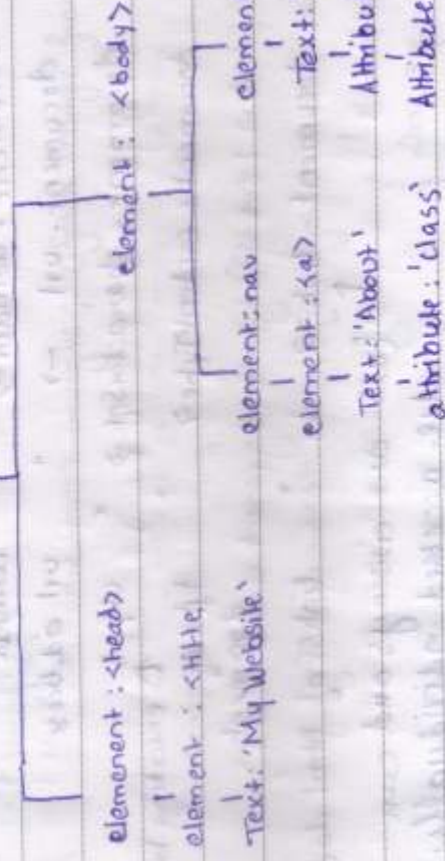
* `let a = 'a', b = 'b', c = 'c';`
`for (let i = 0; i < 10; i++) {`
`console.log(a);`
`console.log(a) \rightarrow (prints \rightarrow a)`

Document Object Model; (DOM)

(HTML file)

Document

root(html)



* Each is refer to as nodes and DOM is that how each nodes are connected.

* DOM Scripting; manipulating or changing Text of nodes in your HTML doc.

* Traversing through various nodes in document is known as DOM traversing.