x = null - nothing
x = undefined - we know something by but don't know what it is

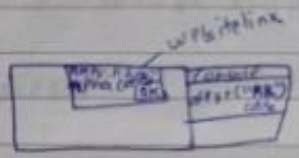
Date : / /

(first.js) JAVASCRIPT — programming language that is use to give instruction to the computer

Input(code) → (computer) → output

Console in browser | VScode (visual studio)

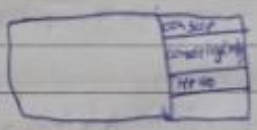
website link



on website ← alert("Apna college");

on console (Ctrl+Shift+I) or F12 ← console.log is used to log (print) a message to the console

console.log("Hello JS");



Connecting JS to HTML

<HTML>
<Head> </head>
<body>
<script> </script> </body>
</HTML>

before </body>

HTML —(directly connected)→ Browser
JS — (indirectly connect) → HTML (connect)

<script src="first.js"> </script>

name of .js file

We don't use data type (int, float etc) because it is dynamically typed

Variable :- Container for data

In console :- direct variable is pointed
In JS file :- (console.log (use) to print variable)

(1) Input :- age = 24 (declaration)
output :- 24, (console.log age)
Input :- age (after declaration)
Output :- 24,

(2) Input :- 2+3.8
output :- 5.8

(3) Input :- name = "Tony"
Output :- Tony
Input :- name = "Stark"
output :- Tony Stark

(4) Input x = null
output null
output y = undefined
output undefined

(5) Input :- X = true; Y = false
output true false

Ex- "123" + 4   "abcd" + 123
output => 1234   output abcd123

Date: ___/___/___

Variable rule

- Case sensitive a ≠ A
- only letter, digit, underscore(_), $ is allowed. (space not allowed)
- first character letter, underscore and $
- Reserved words cannot be variable names. (ex- console) but Console (capital, not reserved)

Tip:- We should use camel case while defining variable
fullName ✓
(small) (capital)

Keyword before variable

don't use it • Var : global scope variable, can be re-declared, can be updated
• let : block scope variable, can be updated
• const : block scope variable,

Ex- let a;
a = 10;
console.log(a) // 10

Ex- let a;
console.log(a) // undefined

ex - let name = "tony stark";
name = "Robert";
name = "Abhisheks";
console.log(name); // Abhisheks

} we are updating it, we can't re-declare it
let name = "tony stark"
let name = "Robert"
let name = "Abhishek"
(within a block { })

Ex- var age = 24;
var age = 59;
var age = 86;
} we are re-declaring and updating it

console.log(age); // 86

Ex- const a;
console.log(a)
// error because missing initializer.

Ex - const Password = 1234; const PI = 3.14; (initializer)
Password = 564; (wrong we can't update it or redeclare it)

let – update can be update
const – can't
const obj – key can be update

Date : __/__/__

Data-types in JS (7)

typeof (input: variable | output: datatype)

Sr. No.	Primitive	typeof variable
1	Number	'number'
2	String	'string'
3	Boolean	'boolean'
4	Undefined	'undefined'
5	Null	'object' but considered primitive data type
6	BigInt	'bigint'
7	Symbol	

Non-primitive
1. Object (Arrays, functions)
collection of values

update value

```
const student {
  age: 20
  name: Kumar
}
student["age"] = student["age"]+1
```

student["age"] || 20

student["age"]+1 || 21

Ex- Input: let age = 24;
Input: typeof age
output: 'number'

Ex- let num = BigInt("123");
output: 123n
output: 'bigint'

Ex - let x = Symbol("Hello");
output: Symbol(Hello)
output: 'symbol'

update value

Key Accessing 2 Way

Key : Value
e.g. age : 24
name : "Rahul"

Accessing key in object

1. Input :- student["fullName"]
output :- 'Rahul Kumar'

2. Input :- student.fullName
output :- 'Rahul Kumar'

Object.
generally const is used but we can use let

Ex -
```
const student = {
  fullName: "Rahul Kumar",   ← comma
  age: 20,                   ← comma
  cgpa: 8.2,                 ← comma
}
```

Output: { fullName: 'Rahul Kumar', age: 20, cgpa: 8.2 }.
output: 'object'

function name() { // code return x; } — it reads return

function Name (Para1, Para2) => { // code return x; } 3 — without return

Date : __/__/__

Comments in JS

// Kya haal chal : single line comment
/* Hello ji */ : Multiple line comment

Operators in Js

Arithmetic operators :- +, -, *, /, %, **

a ** b = a^b (powers)

operands: a + b (operator)

Ex- 2**3 = 2^3 = 8

Unary operator a++, a--
++a, --a

Assignment operators =, +=, -=, *=, /=, %=, **=

comparison operators >, >=, <, <=

Value: == (equal to), != (not equal)

Strict value and type matching: !==, ===

* === mostly used than ==

Ex- console.log(5==5) // true
console.log(5!=5) // false

Ex- a=5 b="5"
console.log(a==b) // true

a=5 b="5"
console.log(a===b) // false

logical operators &&, ||, !

change true into false and via-versa

output: // true // false

a=6, b=5;
!(a>b) // false
(a>b) // true

Ex- let obj = { item: "pen", price: 10 };
console.log ("cost of ", obj.item, "is", obj.price);
output : cost of pen is 10

Conditional Statements. if(condition) { }

Ex- let color;

if (mode === "dark-mode") {
color = "black";
}

switch
switch(condition) {
case 1: ;
break;
default: ;
}

If-else Statement

if(condition) { }
else { }

else if statement

if(condition) { }
else if(condition) { }
else { }

Tenary operator. condition ? true output : false output

Ex - let result = 25>=18 ? "adult" : "not adult";
console.log(result);

Ex - let result = 25>=18 ? console.log("adult") : console.log("not adult");
Ex - Console.log(25>=18 ? "adult" : "false");

prompt("Hello"); It's input is string not number

Popup and can take input as well

let x = prompt("Hello");
Console.log(x); // Hello

Date : __/__/__

loops in JS

for loop

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

i is available in for loop only

King	console
write code i++	5 King

while loop

```
while (condition) {
    // code
}
```

do while

```
do {
    // code
} while (condition);
```

for-of loop

for string, array

```
for (let val of sts var) {
    // code
    console.log(val)
}
```

ex-
```
let string = "Apna";
for (let i of string) {
    console.log(i);
}
```
// A
// P
// n
// a

for-in loop

for key, array

use in object

```
for (let key in obj var) {
    // code
    console.log(key)
}
```

ex-
```
let student = {
    name: "Rahul"
    age : 20;
    [gpa: 10]
}
for (let s in student) {
    console.log(i);
}
```
// name
// age
// gpa

```
console.log(student["key"]);
```
// Rahul
// 20

methods of string

Str.toUpperCase( )
Str.toLowerCase( )
Str.trim( )

abhishek
abhishek
abhishek

ABHISHEK
abhishek
abhishek

Str.charAt(index)
Str.concat(str2)
Str.slice(start, end)
Str.replace(searchval, newval)

Date: exclusive

Here new string is created and old string is unchanged

new string comes with changes.

String. (It is immutable like Java)

let str = "Apna college" or let str = 'Apna college' or let str = `Apna

Property of string :-

Python ⇒ str.length ;// 11
Index value ⇒ Str[0]; // A
Str[6]; // C

let str = "Apna college";
Apna college
0 1 2 3 4 5 6 7 8 9 10
str[6];

King - Abhishek ⇒ 13
King is Abhishek ⇒ 14
King is Abhishek ⇒ 14

Template literals (` `)

let obj = {
item: "pen",
Price: 10,
};

let output = `the cost of ${obj.item} is ${obj.price} rupees`;

number ⇒ string

console.log("the cost of", obj.item, "is", obj.price, "rupees");
console.log(output);

number

String interpolation

`String txt ${expression} string text`

tip: common for programmers to forget methods of string just google it

Ex - `This is a template literal ${1+2+3}`;
output This is is template literal 6

string type

Escape character (\n)

new line

console.log("King\nAbhishek");
// King
// Abhishek

```
let str = "I LOVE JS";
str[0] = S ✗
str = str.replace("I", "S"); ✓
console.log(str);
```

array - may data under same name
object: - may data with different name

Date: / /

Arrays (collection of items) → It is mutable

```
0 1 2 3 4 ← Index
let marks = [96,75,48,83,66);
let marks = ["rahul", "Bobby", "Mukesh"];

let info = ["rahul", 86, "Dahi"]; can be done but not preffer
           string  number  string
```

Print	console.log(marks) // (4) [96,75,48,83,66]
Length	console.log(marks.length) // 4
typeof	console.log(typeof marks) // object (key = index)
Index	If we want to access more than index it will return undefined.
Index	arr[0] // 96
for-of loop Print	for(let i of marks){ console.log(i) } // 96,75,48,83,66

key : value
0 : 96
1 : 75
2 : 48
3 : 83
4 : 66

Arrays Methods

- unshift(): add to start
- Push(): add to end
- Pop(): delete from end & return
- shift(): delete from start & return
- toString(): convert arrays to string

```
let arr = [1,2,3];
arr.push(4,5,6);
console.log(arr) // 1,2,3,4,5,6
```

```
let arr = [1,2,3];
arr.pop();
console.log(arr) // 1,2
```

```
let arr = [1,2,3];
console.log(arr.toString()); // "1,2,3" ← string
console.log(arr); // 1 2 3 ← number
```

- arr.concat(arr2); // [1,2,3,4,5,6]
- arr.concat(arr2, arr3); // [1,2,3,4,5,6,7,8,9]
- arr.slice(1,3); // 2 3 including start idx
- arr.splice(start idx, delcount, newEle2, ele3);

1 2 3 4 5 6 7
0 1 2 3 4 5 6

splice(2,2,101,102)

function can be used as value
let x = fun; function fun() {
}

Date: __/__/__

<script src="demo.js"></script>

it is preferred when we how does mall text

~~JAVA~~

Arrow functions (compact way of writing a function)

Parameter
```
const sum = (a,b) => {
    return a+b; //console.log(a+b)
}
```

output
```
console.log(sum) // (a,b) => {      // returning value
                    return a+b;
                }
```

output
```
sum
~~console.log~~(3,4) // 7
```

Compare
```
const Pain = () => {
    console.log("Hello");
}
```

output
```
Pain() // 'Hello'
```

forEach

```
let arr = [34,56,85,78];
arr.forEach((element, index, array) => {
  console.log(element, index, array);
});
```

// 34 0 [34,56,85,78]
// 56 1 [34,56,85,78]
// 85 2 [34,56,85,78]
// 78 3 [34,56,85,78]

arr.map

```
let arr = [34,56,85,78];
let arr = arr.map((elem) => {
    return (element)
});
console.log(x);
```

// 34
// 56
// 85
// 78

arr.filter

```
let arr = [1,2,3,4,5,6,7,8,9];
let x = arr.filter((element) => {
    return (element).1.2==0
});
console.log(x);
```

// [2 4 6 8]

arr.reduce

```
let arr = [1,2,3,4,5,6,7,8,9];
let x = arr.reduce((e1,e2) => {
    return e1+e2;
});
console.log(x);
```

// 45 (1+2=3+3=6+4=10+5=15+6=21+7=28+8=36+9=45)

DOM (When Page is loaded, DOM is created by browser)

Date : ___/___/___

(Document object model)

- console.log(window) // window & window.window.self: window
- console.log(document) // #document (http://index.html) <html><body></body></html>
- console.log(document.head) // <head>--</head>
- console.log(document.body) // <body>---</body>

Window object - represent an open window in a browser. It is browser's object (not JS's) & is automatically created by browser.
It is a global object with lots of properties & methods.

DOM Manipulation

Selection

```
let x = document.getElementsByTagName("Div");
// 0: div.e   <div class="one"></div>
// 1: div.0   <div class="0"> Two</div>

let x = document.getElementsByClassName("kya");
// 0: div.kya   <div class="kya"> one</div>

let x = document.getElementById("yp");
// <div id="2"> Two</div>

let x = document.querySelector("div")   .kya  #yp  first
let x = document.querySelectorAll("div")   .kya  #yp  all
```

DOM Manipulation

Properties (for get, set(update))

- tagName - return tag
- innerText - text content
- textContent - hidden text content and visible text content
- innerHTML - return plain text with \n like in html

Ex - `x.tagName` `x[0].tagName`

- `x` : query selector
- `x[0]` : getElementsByTagName

```
let divss = document.querySelectorAll(".box");
let count=10;
for(let i of divss){
  i.innerText=" value is " + count;
  console.log(i);
  count++;
}
```

Date : __/__/__

DOM Manipulation
Attributes

"class","a"

- getAttribute("attr") (connecting JS)
- setAttribute(attr, newAttr value)

DOM manipulation
Style

- x.style.background="red"
- x.classList.add("dark"); (change class)

DOM Manipulation
Insert Element

Inside	x.append(el)	(at end)	first we create then we insert element
	x.prepend(el)	(at start)	
outside	x.before(el)	(at start)	
	x.after(el)	(at end)	

let el = document.createElement("div")

DOM Manipulation
Delete Element

x.remove()

removeChild(element)

Events

- The change in the state of an object is known as Event
- These are fired to notify code of "interesting changes" that may affect code execution.

- Pointer Events (mouseovers)
- Mouse Events (click, double click etc)
- Keyboard events (keypress, keyup, keydown)
- Form events (submit etc)
- Drient event & Many more

(1) Inline event handling (not good way)

Ex - this is a box

event as attribute

(2) Event handling in JS (when both written JS is executed)

```
node.event=()=>{
  // code
}
```

Ex - let btn = document.querySelector("#id")

```
btn.onclick=()=>{
  console.log("btn was clicked");
}
```

Date : __/__/__

Event Object :- It is a special object that has details about the event. All event handlers have access to the Event object's properties and methods.

```
node.event=(e)=>{
  // code.
}
```

Ex- let btn = document.querySelector("#btn");

```
btn.onclick=(evt)=>{
  console.log(evt); // PointerEvent {- - - - - -}
  console.log(evt.type); // click
  console.log(evt.target); // <button id="btn" onclick="">btn</button>
  console.log(evt.clientX, evt.clientY); // 130 19
};
```

Event Listeners (It is used when we want to add more than one event, function)

1. node.addEventListener(event, callback); to add
2. node.removeEventListener(event, callback); to remove

Ex- let btn = document.querySelector("#btn");

```
btn.addEventListener("click", ()=>{
  console.log("button was clicked");
});
btn.addEventListener("click", ()=>{
  console.log("Hello");
});

// button was clicked
// Hello
```

Date : __/__/__

LinkedIn
Profile Picture
Name, bio
Headline - Unique

23 24 25 26 27 28

~~Study~~ it contain Prototype that contain some object or function like Push()

Object creation - One way
(entity may properties and method)

```
const student = {
    name : "Rahul",
    marks : 94.4,
    printMarks : function() {
        console.log("marks = ", this.marks); //student.marks
    },
}
```

F1 → (with fn)

```
    Print marks() {
        console.log("marks = ", this.marks);
    }
}
```

F2 → (Print)

User defined Prototypes.
xyz.__proto__ = employee;

* If object's prototype have same method, objects method will be used.

* Object Creation - (Two^nd way) Using class

```
class Class Rahul {
    constructor() {....}
    function() {...}
    function (Para) {...}
}

Object: let myobj = new Rahul();
```

this Keyword.

```
SetPrice(x) {
    this.x = x;
}
```

Classes in JS
constructor() method is:
• automatically invoked by new
• initializes object

let obj = new Rahul()

Class Rahul {
without — constructor() { console.log("Hello") }
with Param — constructor(branch) { console.log("Yash") }
this.branch = branch
}

let x = new Rahul(); // Hello
let y = new Rahul("Yash"); // Yash

Super Keyword

Class Person {
constructor { this.species = "Hello"; } → ①
}
class One extends Person {
constructor (branch) {
super(); // ①
this.branch = branch → ②
}
}

function hello() { console.log("hello"); }
console.log(1);
setTimeout(hello, 4000); in 4s hello function run
console.log(2);

asynchronized — it will wait and output after allow other code to execute

output:-
// 1
// 2
// Hello

Inheritance in JS

class Parent { }
class Child extends Parent { }
Properties & method are passing to child

* if child and parent have same method child will works = method overriding

Ex - class Parent {
hello() { ... }
}
class Child extends Parent { }

let obj = new Child();
obj.hello(); // function

Callback Hell:- Nested callbacks forming of doom

```
function getData(dataId, getNextData) {
    setTimeout(() => { console.log("Data", dataId);
        if (getNextData) {
            getNextData();
        }
    }, 2000);
}
```

// callback hell (Difficult to understand for other)
getData(1, () => { getData(2, () => { getData(3); });

* to avoid this Promises is used