**28-10-2024**

01.

console.log("2"+"2"-0); // 0 Number

02.

const a = {};

const b = { name: "ragu1" };

const c = { name: "ragu2" };

a[b] = { name: "ragu3" };

a[c] = { name: "ragu4" };

**Explanation**

A initially empty object then a key value is object then values name value{ "[object Object]": { "name": "ragu4" } }

console.log(a[b]);

console.log(a);

------------------------------------------------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------------------------------------------------

**22-10-2024**

01.

const one = false || {} || null

const two = null || "" || false

const three = [] || 0 || true

**Explanation**

false null values are falsy value - {}

All falsy value go take last one - false

[] and true first value check first values is true so - []

console.log(one, two, three)

------------------------------------------------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------------------------------------------------

**11-12-2024**

01. arr concept

const arr = [1, 2, 3][1, 2, 3]

console.log(arr) // undefined because a[0] = 1 but you provide a[1, 2, 3]

02. obj concept

const obj1 = { 1: "A1" };

const obj = { 1: 1, 2: 1, 1: 2, 1: 3, [1]: "5", [obj1]: "A2", obj1: "A3" };

console.log(obj);

{ '1': '5', '2': 1, '[object Object]': 'A2', obj1: 'A3' }

obj take last key value 1: 3 then [1] also 1 why array value convert string

then check primtive value 1 smae [obj1] but side here non primitive value

[] to "" then obj1 is object finally '[object Object]'

03. code explain

{

    function fn(){

        console.log("hello")

    }

}

fn()

Explaination

{} -- block-scoped

fn() -- hoisting

Calling the Function: When you call fn() outside the block,

JavaScript recognizes it due to hoisting and executes the function,

03

"use strict";

{

    function fn() {

        console.log("hello");

    }

}

fn(); // Error: fn is not defined

**explain**

Non-strict mode: The function is accessible outside the block due to hoisting.

Strict mode: The function is restricted to the block scope.

05.

"use strict"

var x = 100

var y = 200

z =200 // use strict z is not defined

console.log(x, y, z);

06.

function fn(){

    return 100

}

var fn

console.log(fn)

Function declarations are hoisted with their bodies.

Variable declarations are hoisted but without initialization.

Since function declarations take precedence, fn retains the function value.

------------------------------------------------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------------------------------------------------

**13/11/2024**

12.

function test() {

    // console.log(test.abc)

}

test()

test.abc = 400

test.abc =600

test()

const a= 10

const b = new Number(10)

const c = 10

console.log(a === b, b === c)

false false

b = Number {10}

------------------------------------------------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------------------------------------------------

**18-11-2021**

function ab(a, b) {

  "use strict";

  let a = 100;

  b = 201;

  return arguments[0] + arguments[1];

}

console.log(ab(10, 30));

let a = 100; b = 201

SyntaxError: Identifier 'a' has already been declared

var abc = 100

var xy = 200

console.log(window.abc + window.xy)

ReferenceError: window is not defined

function name(params, ...b) {

  console.log(b[0])

}

name(1, 6, 7, 9);

1. console.log(!"he")

Explanation: The ! operator is a logical NOT operator. It converts its operand to a boolean and then inverts it.

"he" is a non-empty string, so it's truthy (in JavaScript, non-empty strings are treated as true in a boolean context).

Applying the ! operator to a truthy value results in false.

Output: false.

2. console.log(+false)

Explanation: The + operator is a unary plus operator. It attempts to convert its operand to a number.

false is a boolean. When converted to a number, it becomes 0.

Output: 0.

3. console.log(NaN == NaN)

Explanation: NaN (Not-a-Number) is a special value in JavaScript used to represent invalid numbers.

According to the ECMAScript specification, NaN is not equal to any value, including itself.

Output: false.

4. console.log(NaN === NaN)

Explanation: === is the strict equality operator. Like ==, it checks equality, but without type coercion.

Since NaN is not equal to any value, including itself, NaN === NaN is also false.

Output: false.

------------------------------------------------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------------------------------------------------

**27/12/2024**

function sum(a= 100, b =200){

  console.log(a+b)

}

// sum(null, 20) -->200 a values is null null is value a is null passed (null, 200) (null+200) = 200

function sum(a= 100, b =200){

  console.log(a+b)

}

// sum(undefined, 20) -->300 a values is undefined so that a take 100 + b

console.log(undefined + 2); -->NaN

const arr = [0, 1, 2, 3, 4, 5];

arr.forEach((element) => {

  console.log(element);

  if (element === 3) {

    arr.length = 0; // [0, 1, 2, 3] when reached 3 we have to arr.length = 0 so that arr looped is stop

  }

});

Array lenth zero logic

01

const arr = [1, 2];

arr.length = 0;

console.log(arr);

02

arr = [];

console.log(arr, "1");

03

arr.splice(0, arr.length);

arr.splice(0);

console.log(arr, "2");

------------------------------------------------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------------------------------------------------

**28/11/2024**

Object

const arr = [1, 2, 3, 4, 5, 6, 7, 8]

delete arr --> false

delete arr[1] -->true

arr-->(8)[1, empty, 3, 4, 5, 6, 7, 8]

const Arrobj = [{name:"ragu"}, 1]

delete obj[0].name -- >true

Arrobj(2)-->[{}, 1]

//Obj

const obj ={name:"ragu"}

delete obj --> false

delete obj.name --> true

obj-->{}

Falsy Values:

false

0 (zero)

'' or "" (empty string)

null

undefined

NaN (Not-a-Number)

truthy

Non-empty strings ('Hello', "0")

Non-zero numbers (1, -1)

Arrays ([])

Objects ({})

true

 || first truthy values

 && first falsely values

 ?? check null and undefined not check all falsely value

deep explanation

Logical Operators

1. || (Logical OR)

Evaluates operands left to right.

Returns the first truthy value found.

If no truthy value is found, it returns the last falsy value.

console.log(null || "Hello"); // "Hello"

console.log(false || 0 || undefined || "Default"); // "Default"

console.log(null || undefined || NaN); // NaN

2. && (Logical AND)

Evaluates operands left to right.

Returns the first falsy value found.

If all operands are truthy, it returns the last truthy value.

console.log("Hello" && 42); // 42

console.log(true && "World" && null && 10); // null

console.log(1 && 0 && "Something"); // 0

Interaction with null and undefined

|| treats null and undefined as falsy values:

javascript

Copy code

let value = null;

console.log(value || "Default"); // "Default"

let anotherValue = undefined;

console.log(anotherValue || "Fallback"); // "Fallback"

&& treats null and undefined as falsy values:

javascript

Copy code

let value = null;

console.log(value && "Something"); // null

let anotherValue = undefined;

console.log(anotherValue && "Something else"); // undefined

Practical Examples

Providing default values using ||:

javascript

Copy code

let username = null;

console.log(username || "Guest"); // "Guest"

Checking all conditions using &&:

javascript

Copy code

let isAuthenticated = true;

let hasPermission = true;

console.log(isAuthenticated && hasPermission && "Access granted"); // "Access granted"

Short-circuit evaluation:

With ||: Stops at the first truthy value.

With &&: Stops at the first falsy value.

03. Nullish Coalescing Operator (??)

For cases specifically involving null and undefined, JavaScript provides the ?? operator:

Returns the right-hand operand if the left-hand operand is null or undefined.

Does not treat 0, false, or '' as null or undefined.

Example:

javascript

Copy code

console.log(null ?? "Default"); // "Default"

console.log(undefined ?? "Fallback"); // "Fallback"

console.log(0 ?? "Should not replace"); // 0

let x;

let y;

x = y = 12; // Explicitly declare both variables

console.log(x, y); // Outputs: 12 12

ReferenceError. This is because variables declared using var (or not declared at all) are hoisted but not initialized until the code execution reaches the assignment.

console.log(x, y); // ReferenceError: x is not defined

"3" + 12 \* 5(360)--> 12\* 5 "3"+60 => "360"

"3" + 12 \* 5 / 2 +"1" ->

12 \* 5 / 2

60/2 -> 30

"3" + 30 + "1" -> 3301

"3" \*\* 3 square root -> 27

3+"3"\*3 ->

"3" \* 3 = 3+9 "39"

3+"3"+30 => "3330"

------------------------------------------------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------------------------------------------------

**29-11-2024**

01.

Expression  Result  Explanation

[] == []    false   Different array references.

{} === {}   false   Different object references.

{} == {}    false   Different object references, even with type coercion.

[] === []   false   Strict equality requires the same reference, which these arrays don’t share.

[] == ""    true    Type coercion converts the empty array to an empty string, resulting in equal values.

02.

let person = {name :"ragu"}

const mem = [person]

person = null

console.log(mem)

Primitive Values vs References:

In JavaScript, objects (like { name: "ragu" }) are stored as references in memory.

When you assign person to mem, it does not copy the object itself. Instead, mem stores a reference to the same object in memory.

Reassigning person:

When you do person = null, you are only updating the variable person to point to null.

The original object ({ name: "ragu" }) in memory is still referenced by mem.

Summary:

person and mem initially share a reference to the same object.

Reassigning person to null does not affect the reference stored in mem.

The object { name: "ragu" } remains in memory as long as mem references it.

03.

debounce

function handleOnchange(params) {

    console.log("1 minutes delay", params);

  }

  function debouce(callback, delay) {

    let timerId;

    clearTimeout(timerId)

    return function (...arg) {

      //[...arg] why because when user type any character one by one like this(r)(a)(g)(u) we need marge [ragu]

      timerId = setTimeout(() => {

        callback(...arg); // then array spead opreater to [ragu] --> ragu then pass agrugment

      }, delay);

    };

  }

  function nameFn(params) {

  // we want call handleOnchange with 8 seconds

    let val = debouce(handleOnchange, 1000);

    val(params);

  }

  nameFn("ragu");->nameFn("r")nameFn("a")nameFn("g")nameFn("u")

04

"use strict"; x =10;

console.log(x);

x is not declared using var, let, or const, it will throw a ReferenceError

05

let xyz = 1 + abc;

abc = 100

console.log(xyz, abc)

ReferenceError

------------------------------------------------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------------------------------------------------

**04-12-2024**

01.

var a = 20;

console.log(a); // Outputs: 20

function a() {

    console.log("20");

}

a(); // Throws: Uncaught TypeError: a is not a function

Functions are hoisted first, but var declarations overwrite them during runtime.

So, after var a = 20, a becomes a number, not a function. Hence, calling a() throws an error.

02.

console.log(Array(1, 2))

(2) [1, 2]

console.log(Array(1))

[empty]

Key Differences:

Multiple arguments: They are treated as the actual elements of the array.

Single numeric argument: It specifies the length of the array (empty slots).

03.

console.log(Number.isNaN(NaN)); // true

console.log(Number.isNaN("NaN")); // false (string is not coerced)

console.log(Number.isNaN(undefined)); // false

The method Number.isNaN() checks if a given value is exactly the special NaN value

console.log(Object.is(NaN, NaN)); // true

console.log(Object.is(+0, -0));   // false

console.log(Object.is(5, 5));     // true

Object.is same value comparison