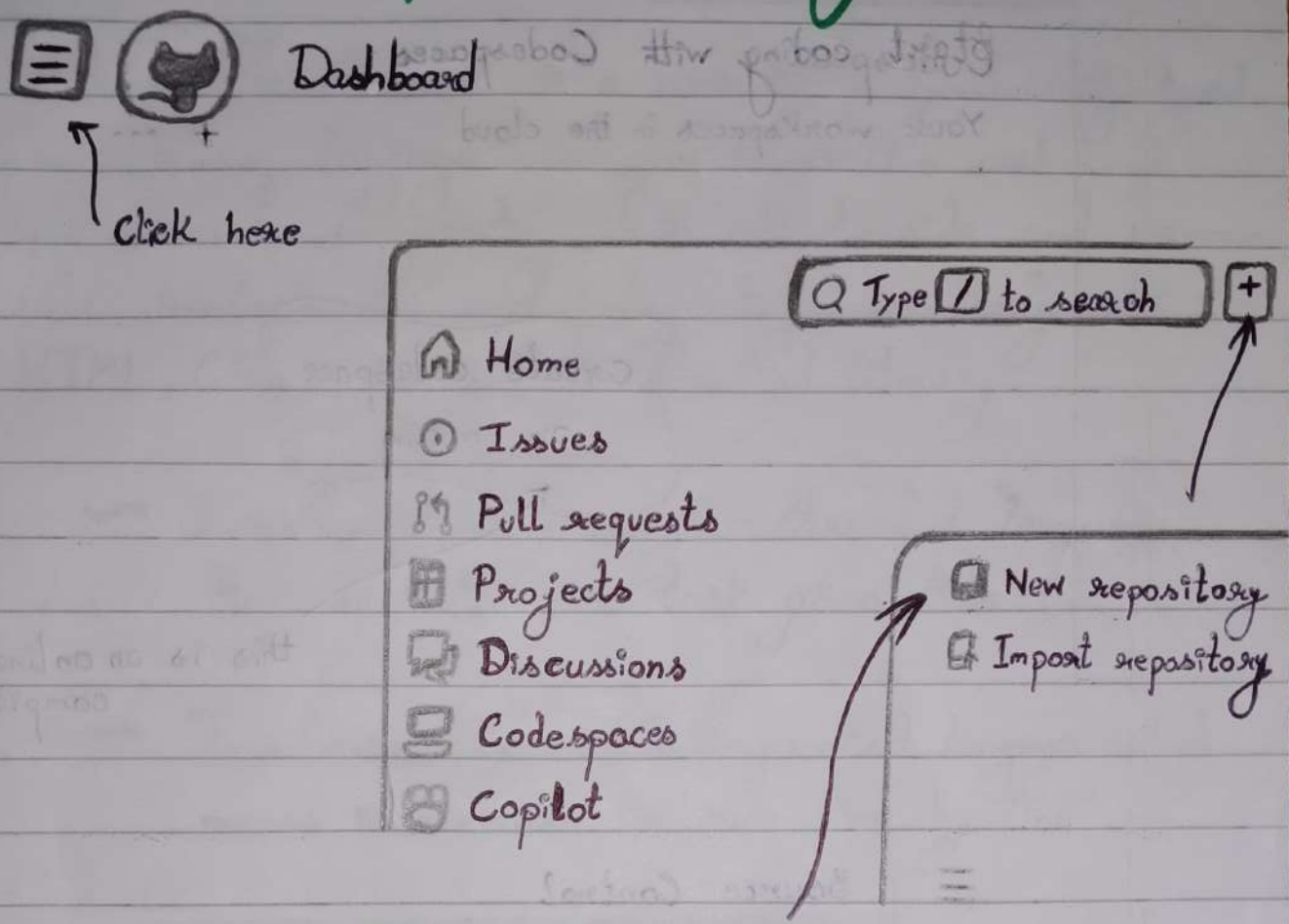
Online compiler – using Github



Fill in blocks :

Owner* / Repository name*

Urgekol

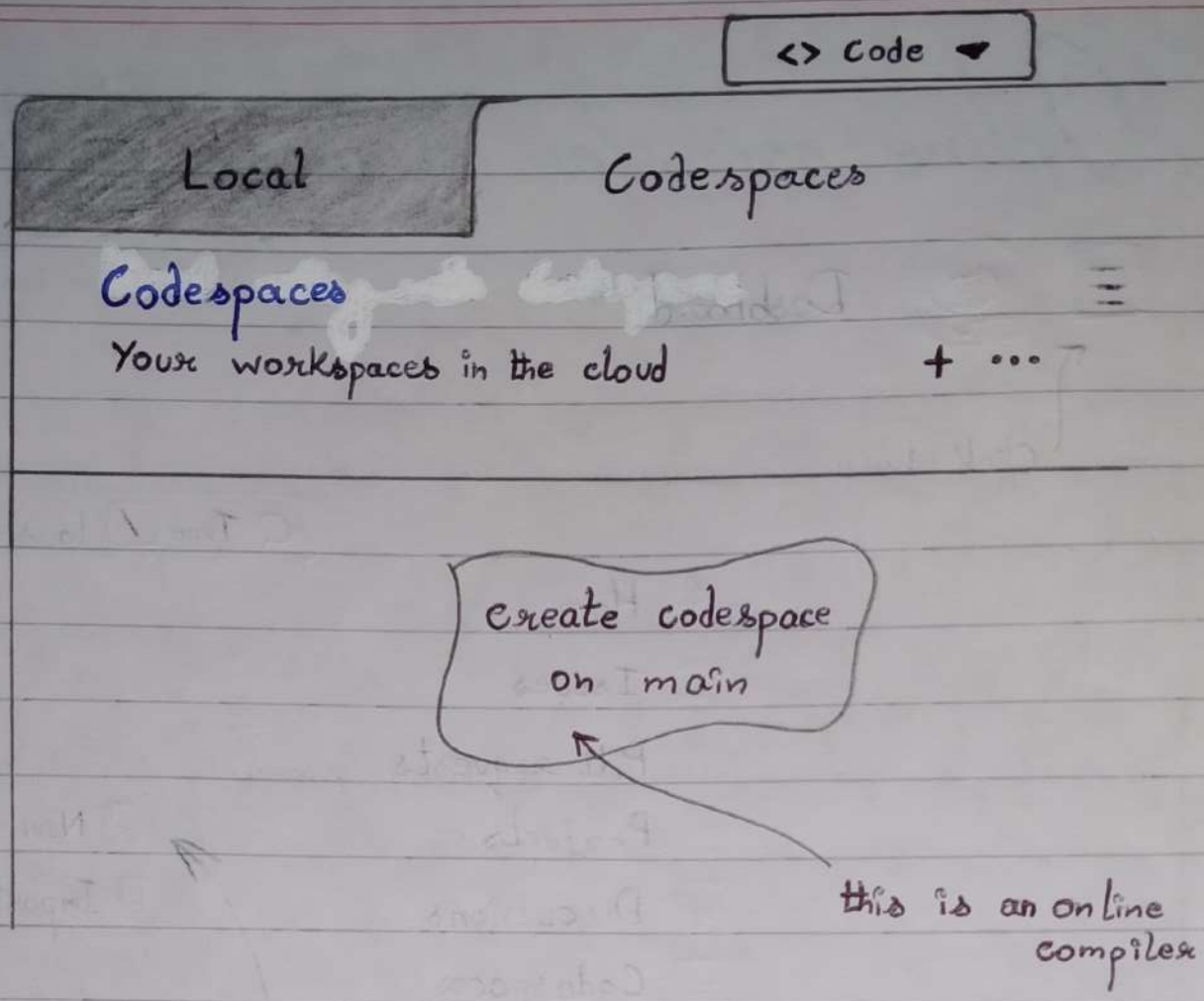
Description

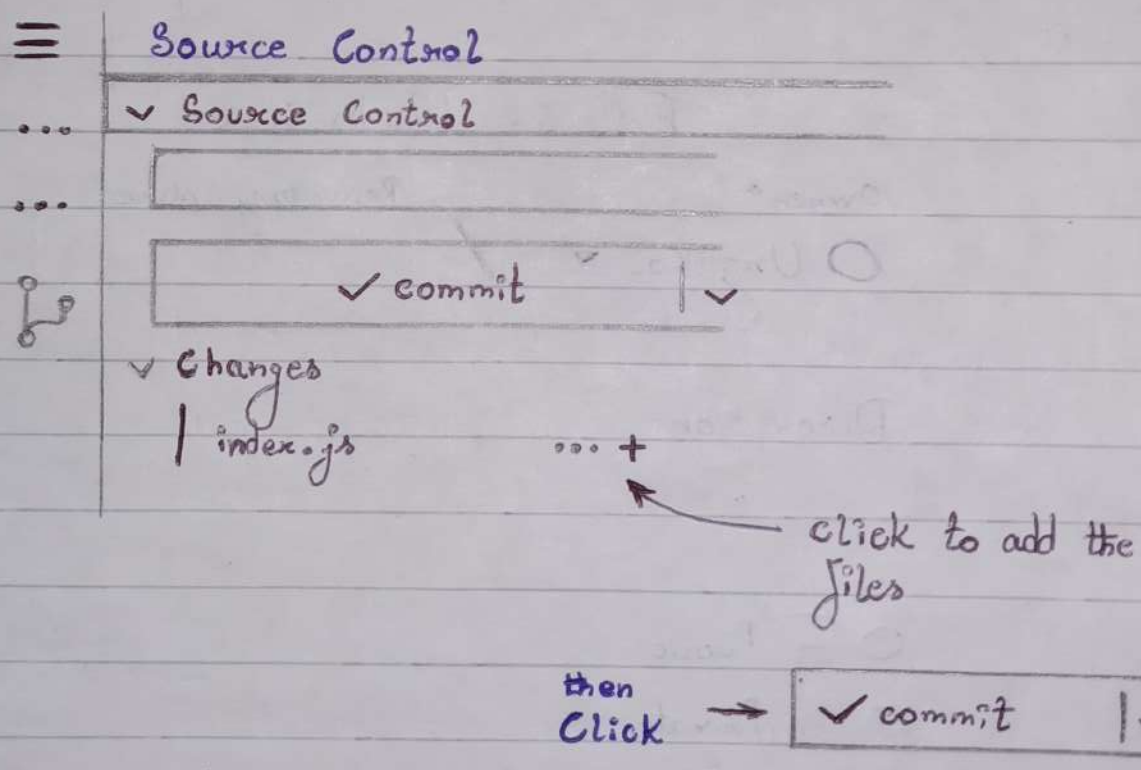
Public

Private

Add a Readme file

Create Repository


<> Code
Local
Codespaces
Codespaces
Your workspaces in the cloud
+ ...
create codespace on main
this is an online compiler


Source Control
Source Control
commit
Changes
index.js
... +
click to add the files
then Click
commit

General JavaScript

JavaScript is a versatile, dynamically typed programming language used for interactive web applications, supporting both client-side and server-side development, and integrating seamlessly with HTML, CSS, and a rich standard library.

- JavaScript is a single-threaded language that executes one task at a time
- JavaScript is an Interpreted language which means it executes the code line by line
- The data type of the variable is decided at run-time in JavaScript that's why it is called Dynamically typed.

Features of JavaScript

- Client-Side Scripting

  JavaScript runs on the user's browser, so has a faster response time without needing to communicate with the server

- Versatile

  JavaScript can be used for a wide range of tasks, from simple calculations to complex server-side applications

- Event-Driven

  JavaScript can response to user actions (clicks, keystrokes) in real-time

- Asynchronous

  JavaScript can handle tasks like fetching data from servers without freezing the user interface

- Rich Ecosystem

  There are numerous libraries and frameworks built on JavaScript, such as React, Angular, and Vue.js, which make

development faster and efficient

Execution Context

The Execution Context in JavaScript is the environment where the code is executed.

It consists of variable declarations, function declaration, and the scope chain.

Phases of Execution Context

- Creation Phase
  - The JS engine allocates memory for variables and functions
  - Variables are initialised with undefined, and functions are stored in memory
- Execution Phase
  - The code is executed line by line
  - Variables are assigned actual values
  - Functions are invoked

Types of Execution Context

- Global Execution Context (GEC)
  - Created when the script starts.
  - Represents the global scope (window in browsers, global in Node.js)

- Function Execution Context (FEC)
  - Created every time a function is called
  - Has its own variable environment and scope

eval()

eval() is a function property of the global object.

Syntax

eval (expression)

The argument of the eval() function is a string. It will evaluate the source string as a script body, which means both statements and expressions are allowed.

It returns the completion value of the code.

Examples:

- ```
  let x = 10;
  let y = 20;
  let result = eval("x + y");
  console.log (result);
  ```

  Output : 30

- ```
  eval("var a = 5 ; var b = 10;");
  console.log (a+b);
  ```

  Output : 15

- ```
  eval ("function greet()
  {
  return `Hello World`;
  }");
  console.log (greet());
  ```

  Output : Hello World
  ```

Hoisting

Hoisting is JavaScript's default behaviour of moving variable and function declarations to the top of their scope during the execution phase

Examples :

- ```
  console.log (x);
  let x = 5;
  console.log (x);
  ```

  Output :
  ReferenceError : Cannot access 'x' before initialization

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

  Output :
  Undefined
  5

- ```
  console.log (x);
  const x = 5;
  console.log (x);
  ```

  Output :
  ReferenceError : Cannot access 'x' before initialization

ReferenceError

= Temporal Dead Zone (TDZ)
  ```

Temporal Dead Zone (TDZ)

The Temporal Dead Zone (TDZ) refers to the time between when a variable is hoisted in the execution context and when it is actually initialized (assigned a value).

During this period, the variable exists but cannot be accessed because it's in an "uninitialized" state. If we try to access it before initialization, a **ReferenceError** will be thrown.

Variables

The types of variable are:

- var
- let
- const

var	let
• Function - scoped	• Block - scoped
• Hoisted with undefined	• Hoisted but in TDZ (not initialised)

- Re-declaration is allowed within the same scope

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

var x = 10;
console.log (x);
```

Output:
5
10

- Re-declaration is not allowed in the same scope

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

let x = 10;
console.log (x);
```

Output:
5
SyntaxError: Indentifier 'x' has already been declared

-

```
function test()
{
    var x = 10;
    if (true)
    {
        var x = 20;
        console.log (x);
    }
    console.log (x);
}
test();
```

-

```
function test ()
{
    let x = 10;
    if (true)
    {
        let x = 20;
        console.log (x);
    }
    console.log (x);
}
test ();
```

Output:
20
20

Output:
20
10

Data Types in JS

Primitive

String, Boolean, Undefined, Number, Null

Non-Primitive

Objects, Arrays, Functions

Type conversion

- Converting to a number....

Type coercion — automatic or implicit conversion

Type conversion — explicit conversion

```
console.log (Number ("123"));

cosole.log (parseInt ("123"));

console.log (parseFloat ("123.45"));
```

```
console.log (Number (true));  // or console.log (+true);

console.log (Number (false));
```

```
console.log (Number ("123abc"));

console.log (parseInt ("123abc"));

console.log (parseFloat ("3.14abc"));
```

- Converting to a String ....

```
console.log (String (42));

console.log ((123.45).toString ());
```

```
console.log (String (false));

console.log ((true).toString ());
```

- Converting to a Boolean

```
console.log(Boolean(1));

console.log(Boolean(0));

console.log(Boolean("Hello"));

console.log(Boolean(""));

console.log(Boolean(null));

console.log(Boolean(undefined));

console.log(Boolean([]));

console.log(Boolean({}));
```

Math Properties

- Math.E (Euler's number)

$$e = \lim_{n \to \infty} \left(1 + \frac{1}{n}\right)^n$$

- Math.PI (π)
- Math.SQRT2 ($\sqrt{2}$)
- Math.SQRT1_2 ($\frac{1}{\sqrt{2}}$)
- Math.LN2 ($\ln(2)$)
- Math.LN10 ($\ln(10)$)
- Math.LOG2E ($\log_2 e$) $= \frac{1}{\ln(2)}$
- Math.LOG10E ($\log_{10} e$)

$$= \frac{1}{\log(10)}$$

Date

The Data object represents dates and times in JavaScript. It allows you to create and manipulate dates, perform operations like getting the current date and time, formatting dates, and extracting specific components such as the year, month, day, hour, minute, and second.

- ```
  let currentDate = new Date();

  let currentYear = currentDate.getFullYear();

  let currentMonth = currentDate.getMonth();

  let currentDay = currentDate.getDate();
  ```

Comparision Operators

- Equality Operator (==)

```
5 == 5          // true          "123" == 123
'5' == 5        // true          // true
"5" == 5        // true          "55" == '5'
"123" == "123"  // true          // true
```
```

NaN == NaN // false

0 == false // true

0 == null // false

Infinity == NaN // false

- Strictly equality Operator (===)

5 === 5 // true

'5' === '5' // true

'5' === 5 // false

"55" === 55 // false

— same data type and same data

Logical Operators

&& || ! ?? { Nullish Coalescing

The nullish coalescing operator returns the right-hand operad when the left-hand operand is either null or undefined.

Otherwise, it returns the left-hand operand.

```
let username = null;
let defaultName = "Guest";

console.log (userName ?? defaultName);

// Guest
```

```
let userName = "Kartik";
let defaultName = "Guest";

console.log (userName ?? defaultName);

// Kartik
```

Special Operator

- in - operator → true / false

```
let lang = ["HTML", "CSS", "JavaScript"]
            0       1       2

console.log (1 in lang); // true
console.log (3 in lang); // false
console.log ("HTML" in lang); // false
```

{ index present or not

```
const Data = {
        name : "Rahul"
        age : 21
        city : "Noida"
};

console.log ("name" in Data); // true
console.log ("address" in Data); // false
```

— property exist in the object or not

- instanceof - operator — tests if an object is an instance of a particular class or constructor

```
let lang = ["HTML",
            "CSS",
            "JavaScript"];

console.log (lang instanceof Array);
console.log (lang instanceof Object);
console.log (lang instanceof String);
console.log (lang instanceof Number);
```

Number Object

- let num = Number ("10");
  ↳ 10

- let num = Number.MAX_VALUE;
  ↳ 1.7976931348623157e+308

- let num = Number.MIN_VALUE;
  ↳ 5e-324

- let num = Number.POSITIVE.INFINITY;
  ↳ Infinity

- let num = Number.NEGATIVE.INFINITY;
  ↳ -Infinity

Modulation:

(person.js)

```
export const name = "Jesse";
export const age = 40;
```

(person.js)

```
const name = "Jesse";
const age = 30;

export {name, age};
```

```
import {name, age} from
"./person.js";

console.log(name);
console.log(age);
```

or,

```
import data from "./person.js";

console.log (data.name);
console.log (data.age);
```

- **default** { allows only one export

(message.js)

```
const message = () => {

    const name = "Jesse";
    const age = 30;

    return name + " " + age;
}

export default message;
```

```
import message from
"./message.js";

console.log(message());
```

ES6 Modules	Common JS
Uses import and export (supports both named and default exports)	Uses require() to import and module.exports (or exports) to export

`export const add = (a, b) => {`
`  return a + b;`
`};`
	`const add = (a, b) => {`
	`  return a + b;`
	`}`

`export default function multiply (a, b) {`
`  return a * b;`
`}`
	`const multiply = (a, b) => {`
	`  return a * b;`
	`}`
	`module.exports = {add, multiply};`

`import multiply, {add} from "./math.js";`	`const math = require("./math.js");`
`console.log (add (2, 3));` `console.log (multiply(2, 3));`	`console.log (math.add(2, 3));` `console.log (math.multiply(2, 3));`

(package.json)

```
{
  "type": "module"
}
```

Static: The module structure is determined at compile time, and optimizations.

Dynamic: The modules are loaded synchronously at runtime using function calls

Imports are hoisted; must appear at the top of the file; generally loaded asynchronously in browsers

Loaded and evaluated when require() is called (synchronous loading)

Standardized in ES6; supported by modern browsers and Node.js (with .mjs extension or "type": "module" setting)

Traditional module system in Node.js; widely used in Node.js libraries.

Immediately Invoked Function Expression (IIFE)

```
//named IIFE
(function kitty () {
    console.log ("Hello");
})();
//unnamed IIFE
((name) => {
    console.log (`Hi, ${name}`);
})('Arun');
```

Output:
Hello
Hi, Arun

Function Declaration	Function Expression
A function is declared using the function keyword with a name	A function is defined and assigned to a variable
Function declarations are hoisted	Function expression are not hoisted

```
greet();  // ✓ Works due to hoisting
function greet()
{
    console.log("Hello");
}
```

```
greet();  // ✗ Error: Cannot access 'greet' before initialization
const greet = function() {
    console.log("Hello");
}
```

- Arrow Functions — are not hoisted

```
greet();  // ✗ Error: Cannot access 'greet' before initialization

const greet = () => console.log("Hello");
```

Array

— creating an Empty array

```
let a = [ ];
console.log (a);
```

— creating an array and initializing with values

```
let b = [10, 20, 30];
console.log (b);
```

— same

```
let b = new Array (10, 20, 30);
console.log (b);
```

— to find length

```
let b = [10, 20, 30];
console.log (b.length);
```

— to increase the length

```
b.length = 5;
console.log (b);
```

— creating an array with empty slots, then filling it

```
let arr = new Array (5).fill ("123");
console.log (arr);
```

— add an element at the end

```
arr.push (50);
console.log (arr);
```

— remove the last element

```
arr.pop ();
console.log (arr);
```

— add an element at the beginning

```
arr.unshift (350);
console.log (arr);
```

— for-each

```
arr.forEach ((x) => console.log (x));
```

call back

iterator object (Array, String, Map)

```
for (let i in arr)
{
    console.log (i)
}
```

— will print just the index numbers

```
for ( let i of arr)
{
    console.log (i);
}
```

— finding the first element > 20

```
let arr = [10, 20, 30];
console.log (arr.find (x => x>20));
```

— reversing the array

```
console.log (arr.reverse());
```

— getting a portion of an array (index 1 to 2)

```
console.log (arr.slice (1, 2));
```

— concatenating arrays

```
let brr = ["a", "b", "c"];
let newArr = arr.concat (brr);
console.log (newArr);
```

— joining elements into a string

```
console.log (brr.join ('-'));
```

```
const coding =["js", "ruby", "java", "python"
                                "cpp"];

function printMe(item)
{
    console.log(item);
}

arr.forEach(printMe);
```

call back
- just function name

item = each element of arr

Output: js
ruby
java
python
cpp

```
arr.forEach((item, index, arr) => {
    console.log(item, index, arr)
});
```

call back

Output:

```
js 0 [`js`, `ruby`, `java`, `python`, `cpp`]
ruby 1 [`js`, `ruby`, `java`, `python`, `cpp`]
java 2 [`js`, `ruby`, `java`, `python`, `cpp`]
python 3 [`js`, `ruby`, `java`, `python`, `cpp`]
cpp 4 [`js`, `ruby`, `java`, `python`, `cpp`]
```

```
▫ const myCoding = [
    {
        languageName : "JavaScript",
        File : "js"
    },
    {
        languageName : "Java",
        File : "java"
    },
    {
        languageName : "Python",
        File : "python"
    },
    {
        languageName : "C++",
        File : "cpp"
    }
];
```

```
myCoding.forEach((item) => {
    console.log(item.languageName);
});
```

Output: JavaScript
Java
Python
C++

DOM

The DOM is a programming interface for HTML and XML documents.

It represents the page so that programs (like JS) can change the document structure, style, and content.

Key points

- it turns every element, attribute, and text into a node that can be accessed and manipulated.
- think of it as a tree structure where each branch is an element on the webpage.

in any website

function and other utilities

- console.log (window.document)

  or,

  console.dir (document)

- console.log (document) // document is an object

<!DOCTYPE html> // mentions the document type is HTML