JS →

Languages
-> programming and Scripting language

Programming → compiled and .exe
Java C++ C

Scripting language →
Interpreter
Javascript and php , python ,
Client side vs server side

Client Side → code is downloaded by the users browser and it is executed on their machine
Front end
Use Cases

Form Validation
Dynamic content Updates without refreshing the page (AJAX requests)

Browser → js engine
Chrome → V8 (c++)
Edge → Chakra

Server side scripting language

Runs on the web server → receive req → process → and then send response

→ authn db business logic

Python node.js , perl

HTML DOM
UI
Add new HTML to page , change the existing content or modify styles

Local storage
React to user interactions key press , click
Get and set the cookies
Ask questions

Client side
DOM
Fetch
Geolocation
Local storage

Server side (node js)
HTTP , filesystem , OS Modules , Database

JS compatibility table
https://caniuse.com/css-grid

https://compat-table.github.io/compat-table/es6/

Developer Console

—----------

Variables

Declaration

1. Var — ES5
2. Let —
3. Const —Block scoped

```
function example(){
    if(true)
    {
        let a = 20;
        a = 40;
        console.log(a);
    }

}

example()
```

```
function example(){
    if(true)
    {
        var a = 20;
        a = 40;

    }
    console.log(a);
```

```
}

example()
```

```
function example(){
    if(true)
    {
        const a = 20;
        console.log(a);
    }
}
example()
```

//const is similar to let → block scoped

//Interacitions in javascript

Alert , prompt , confirm

Prompt – > (title) [default]
Default → optional

Javascipt datatypes 8

String
Number
Bigint
Undefined
Null
Boolean

Object
Built in object types
Maps , arrays ,

String Coversion

Automatic

Explicit
String(false)

Strings
→ “” ‘’ ``
ESCAPE CHARACTER TO PRINT
QUOTES

```
let a = "Hello , JOHN SAID \"hEELO 0\" ";
console.log(a);
```

OR USE SINGLE QUOTES

.._ for multi line strings `let a = `This is Multi line`

```
multi line
`
console.log(a);
```

TYPE CONVERSION

String conversion
Numeric –
Boolean

Numeric conversion

```
let str2 = "123n"
console.log(Number(str2))
```

Output → NaN

Boolean Conversion

```
console.log(Boolean(1));
console.log(Boolean(0));
console.log(Boolean("hello"));
console.log(Boolean(""));
console.log(Boolean("0"));
console.log(Boolean(" "));
```

Output

```
true
false
true
false
true
true
```

Arithmetic Operations

Operands
5+ 3

Arithmetic Operations

Addition +
Subtraction
Mul
Div
Remaninder (Mod) %
Exponential **

String concatenation
→ (+)

```
console.log("6"+ 5 - 2);
```

Output 63

```
console.log("6"+ 5 * 2);
```

Output 610

Functions –.
.

1. Function declaration
2. Function Expression
3. Arrow Functions

String Methods

```
const str = "Hello,World"
console.log(str.charAt(0));
console.log(str.charCodeAt(0));
```

```
const str2 = "World";

console.log(str.concat(',',str2));

//includes (substring)

console.log(str.includes("world")); //true
console.log(str.includes("World")); //false

//endsWith

console.log(str.endsWith("World"));
console.log(str.endsWith("Hello" ,5));

//indexOf
console.log(str.indexOf("W"));
console.log(str.indexOf("l"));

//lastIndexOf
console.log(str.lastIndexOf("l"));
//padstart

console.log(str.padStart(20 , "*"))

//repeat (count)
console.log(str.repeat(3))

// replace(oldvalue , newvalue)
console.log(str.replace("l", "0"))

//replaceall

console.log(str.replaceAll("l", "0"))

//slice(start,end)
console.log(str.slice("0", "3"))

//split (seperator , limit)
console.log(str.split(""))
console.log(str.split(","))

console.log(str.toLowerCase());
```

```
let str3= "    Hello String        "
//trim() --> removes spaces from both the ends i=of the str1\
console.log(str3.trim())

console.log(str3.trimStart())

console.log(str3.trimEnd())

console.log(str.length)
```

Array Methods

```
//arrays

//creating

let arr = [1,2,3,5,"2"]
console.log(arr)
//array constructor
console.log(Array(3))
// Array.of
console.log( Array.of(3))
console.log(arr.length)
///indexOf
//lastIndexOf

//Modifying array
arr.push(3)
console.log(arr);

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


//adds the element to the begning of the array
console.log(arr.unshift(1))
console.log(arr);
```

```
//shift()
console.log(arr.shift())
console.log(arr)

let arr1 = [1,2,3,51]
let arr2 = [2,3,51 ]
 console.log(arr1.concat(arr2))

 //splice removes elements from anyposition
 let arr3 = [1,2,3,5,5]
 arr3.splice(1,2,4,8)
 console.log(arr3)

 //foreach
arr3.forEach((element) => console.log(element))

const newarr = arr3.map((x) => x*2)
console.log(newarr)


const filteredarr = arr3.filter((x) => x > 4)
console.log(filteredarr)

//reduer

let sum = arr3.reduce((acc,x) => acc + x, 0)
console.log(sum);

//some() --> tests wheter at least one element is passing the
test
let result = arr.some((x => x> 2))
console.log(result)

let result1 = arr.every((x => x> 2))
console.log(result1)

//return the first element that satisfies the condition
let found = arr3.find((x) => x >2)
console.log(found)
```

Write a function that takes a string as input and returns the string reversed.
Write a function that checks if a given string is a palindrome (reads the same forwards and backwards).
Write a function to count how many times a particular character appears in a string.
Write a function that replaces all occurrences of a substring with a new string.
Write a function that removes all duplicate characters from a string.
Write a function that converts a string to title case (each word starts with a capital letter).
Write a function to find the first non-repeated character in a string.
Write a function that converts a string into an array of words.
Write a function that removes whitespaces from the beginning and end of a string.
Write a function that finds the length of the longest word in a string.
Write a function that returns the index of the last occurrence of a given character in a string.
Write a function that extracts a substring from a string based on given start and end indices.
Write a function to pad a string with a specific character until it reaches a specified length.

```
// 1. Reverse a String
// Question: Write a function that takes a string as input and
returns the string reversed.

// Answer:
function reverseString(str) {
  return str.split('').reverse().join('');
}

console.log(reverseString("hello"));  // Output: "olleh"
// 2. Check if a String is a Palindrome
// Question: Write a function that checks if a given string is a
palindrome (reads the same forwards and backwards).

function isPalindrome(str) {
    const cleanedStr = str.split(' ').join('').toLowerCase();
    const reversedStr = cleanedStr.split('').reverse().join('');
    return cleanedStr === reversedStr;
  }

  console.log(isPalindrome("A man a plan a canal Panama")); //
true
  console.log(isPalindrome("hello")); // false
```

```
// 3. Count Occurrences of a Character
// Question: Write a function to count how many times a
particular character appears in a string.


function countChar(str, char) {
  return str.split(char).length - 1;
}

console.log(countChar("hello world", "l"));  // Output: 3
// 4. Replace All Occurrences of a Substring
// Question: Write a function that replaces all occurrences of a
substring with a new string.


function replaceAll(str, target, replacement) {
  return str.split(target).join(replacement);
}

console.log(replaceAll("hello world", "l", "x"));  // Output:
hexxo worxd
// 5. Remove Duplicate Characters from a String
// Question: Write a function that removes all duplicate
characters from a string.

// Answer:
function removeDuplicates(str) {
  let uniqueStr = '';
  for (let i = 0; i < str.length; i++) {
    if (!uniqueStr.includes(str[i])) {
      uniqueStr += str[i];
    }
  }
  return uniqueStr;
}
console.log(removeDuplicates("aabbcc"));  // Output: "abc"

// 6. Convert a String to Title Case
// Question: Write a function that converts a string to title
case (each word starts with a capital letter).

function toTitleCase(str) {
```

```
  return str.split(' ')
              .map(word => word.charAt(0).toUpperCase() +
word.slice(1).toLowerCase())
              .join(' ');
}

console.log(toTitleCase("hello world from javascript"));  //
Output: "Hello World From Javascript"

// 7. Find the First Non-Repeated Character
// Question: Write a function to find the first non-repeated
character in a string.

function firstNonRepeated(str) {
  for (let i = 0; i < str.length; i++) {
    if (str.indexOf(str[i]) === str.lastIndexOf(str[i])) {
      return str[i];
    }
  }
  return null;
}

console.log(firstNonRepeated("aabbccdeffg"));  // Output: "d"
console.log(firstNonRepeated("aabbcc"));       // Output: null


// 8. Convert a String to an Array of Words
// Question: Write a function that converts a string into an
array of words.

function stringToArray(str) {
    // Split the string by spaces
    const words = str.split(' ');

    const result = words.filter(word => word.length > 0);

    return result;
  }

  console.log(stringToArray("hello world from javascript"));  //
Output: ["hello", "world", "from", "javascript"]
  console.log(stringToArray("  hello   world  "));           //
Output: ["hello", "world"]
```

```
  console.log(stringToArray("this is   a test"));             //
Output: ["this", "is", "a", "test"]

console.log(stringToArray("hello world from javascript"));  //
Output: ["hello", "world", "from", "javascript"]

// 9. Trim Whitespaces from a String
// Question: Write a function that removes whitespaces from the
beginning and end of a string.

function trimString(str) {
  return str.trim();
}

console.log(trimString("   Hello World!   "));  // Output: "Hello
World!"

// 10. Find the Length of the Longest Word
// Question: Write a function that finds the length of the
longest word in a string.


function longestWordLength(str) {
  const words = str.split(' ');
  const longest = words.reduce((longest, current) =>
current.length > longest.length ? current : longest, "");
  return longest.length;
}

console.log(longestWordLength("hello world from javascript"));
// Output: 10

// 11. Find the Index of the Last Occurrence of a Character
// Question: Write a function that returns the index of the last
occurrence of a given character in a string.

function lastIndexOfChar(str, char) {
  return str.lastIndexOf(char);
}

console.log(lastIndexOfChar("hello world", "l"));  // Output: 9

// 12. Extract a Substring
```

```
// Question: Write a function that extracts a substring from a
string based on given start and end indices.

function extractSubstring(str, start, end) {
  return str.substring(start, end);
}

console.log(extractSubstring("hello world", 0, 5));  // Output:
"hello"

// 13. Convert String to Number
// Question: Write a function that converts a string to a number
(parseInt or parseFloat).


function stringToNumber(str) {
  return Number(str);  // You can also use parseInt() or
parseFloat() depending on the type of number
}

console.log(stringToNumber("42"));      // Output: 42
console.log(stringToNumber("3.14"));    // Output: 3.14

// 14. Pad a String
// Question: Write a function to pad a string with a specific
character until it reaches a specified length.


function padString(str, length, char = ' ') {
  return str.padStart(length, char);  // You can use padEnd for
padding at the end
}

console.log(padString("hello", 10, "*"));  // Output:
"*****hello"



function reverse(str) {
    return str.reverse();
  }
  console.log(reverseString("hello"));
```

Create and Access Array Elements

Create an array fruits with the elements ['apple', 'banana', 'cherry']. Log the second element of the array.

Add Elements to an Array

Add the element 'orange' to the end of the fruits array. Log the updated array.

Remove Elements from an Array

Remove the first element from the fruits array and log the updated array.

Find the Length of an Array

Find and log the length of the fruits array.

Check if an Element Exists in an Array

Check if the element 'banana' exists in the fruits array. Log a message indicating whether the element is present or not.

Array Iteration Using forEach

Use the forEach() method to log each fruit in the fruits array.

Filter Elements in an Array

Use the filter() method to create a new array that only contains fruits that have more than 5 characters. Log the new array.

Map Over an Array

Use the map() method to create a new array where each fruit in the fruits array is converted to uppercase. Log the new array.

Find an Element in an Array

Use the find() method to find the first fruit in the fruits array that has more than 5 characters. Log the result.

Reduce an Array

Use the reduce() method to concatenate all the fruits in the fruits array into a single string with spaces in between. Log the result.

Sort an Array

Sort the fruits array alphabetically and log the sorted array.

Combine Two Arrays

Combine the fruits array with another array ['grape', 'melon'] and log the new array.

Check if an Array Contains an Element

Use the includes() method to check if the fruits array contains the element 'cherry'. Log the result.
Find the Index of an Element

Use the indexOf() method to find the index of the element 'banana' in the fruits array. Log the result.
Splice Elements in an Array

Use the splice() method to remove the second element from the fruits array and add the element 'pear' in its place. Log the updated array.
Create an Array of Objects

Create an array of objects representing books with properties: title, author, and year. Log the array.
Access Nested Array Elements

Given the array const library = [['Book1', 'Author1'], ['Book2', 'Author2']], access and log the author of the second book.
Flatten an Array

Flatten a nested array const nestedArray = [[1, 2], [3, 4], [5, 6]] into a single array and log the result.
Remove Duplicate Elements from an Array

Remove duplicate elements from an array const numbers = [1, 2, 3, 2, 1, 4, 5] and log the result.
Check if an Array is Empty

Write a function to check if an array is empty and log the result for the array const arr = [].

```
// Create and Access Array Elements
const fruits = ['apple', 'banana', 'cherry'];
console.log(fruits[1]);


// Add Elements to an Array

const fruits = ['apple', 'banana', 'cherry'];
fruits.push('orange');
console.log(fruits); // Output: ['apple', 'banana', 'cherry',
'orange']

// Remove Elements from an Array
```

```
const fruits = ['apple', 'banana', 'cherry'];
fruits.shift();
console.log(fruits); // Output: ['banana', 'cherry']


// Find the Length of an Array


const fruits = ['apple', 'banana', 'cherry'];
console.log(fruits.length); // Output: 3

// Check if an Element Exists in an Array


const fruits = ['apple', 'banana', 'cherry'];
if (fruits.includes('banana')) {
  console.log('Banana is in the array');
} else {
  console.log('Banana is not in the array');
}
// Output: "Banana is in the array"

// Array Iteration Using forEach
const fruits = ['apple', 'banana', 'cherry'];
fruits.forEach(fruit => console.log(fruit));
// Output:
// apple
// banana
// cherry

// Filter Elements in an Array
const fruits = ['apple', 'banana', 'cherry', 'grape', 'melon'];
const longFruits = fruits.filter(fruit => fruit.length > 5);
console.log(longFruits); // Output: ['banana', 'cherry', 'grape',
'melon']


// Map Over an Array
const fruits = ['apple', 'banana', 'cherry'];
const uppercaseFruits = fruits.map(fruit => fruit.toUpperCase());
console.log(uppercaseFruits); // Output: ['APPLE', 'BANANA',
'CHERRY']
```

```
// Find an Element in an Array

const fruits = ['apple', 'banana', 'cherry', 'grape'];
const longFruit = fruits.find(fruit => fruit.length > 5);
console.log(longFruit); // Output: "banana"


// Reduce an Array
const fruits = ['apple', 'banana', 'cherry'];
const concatenatedFruits = fruits.reduce((acc, fruit) => acc + '
' + fruit);
console.log(concatenatedFruits); // Output: "apple banana cherry"

// Sort an Array
const fruits = ['banana', 'apple', 'cherry'];
fruits.sort();
console.log(fruits); // Output: ['apple', 'banana', 'cherry']


// Combine Two Arrays
const fruits = ['apple', 'banana', 'cherry'];
const moreFruits = ['grape', 'melon'];
const combinedFruits = [...fruits, ...moreFruits];
console.log(combinedFruits); // Output: ['apple', 'banana',
'cherry', 'grape', 'melon']

// Check if an Array Contains an Element
const fruits = ['apple', 'banana', 'cherry'];
console.log(fruits.includes('cherry')); // Output: true

// Find the Index of an Element
const fruits = ['apple', 'banana', 'cherry'];
console.log(fruits.indexOf('banana')); // Output: 1


// Splice Elements in an Array
const fruits = ['apple', 'banana', 'cherry'];
fruits.splice(1, 1, 'pear'); // Removes 1 element at index 1 and
adds 'pear'
console.log(fruits); // Output: ['apple', 'pear', 'cherry']

// Create an Array of Objects
```

```
const books = [
  { title: 'book1', author: 'Author', year: 2001 },
  { title: 'book2', author: 'Author2', year: 2001 }
];
console.log(books);
// Output:
// [
//   { title: '1984', author: 'George Orwell', year: 1949 },
//   { title: 'Brave New World', author: 'Aldous Huxley', year:
1932 }
// ]


// Access Nested Array Elements

javascript
Copy
Edit
const library = [['Book1', 'Author1'], ['Book2', 'Author2']];
console.log(library[1][1]); // Output: "Author2"

// Flatten an Array
const nestedArray = [[1, 2], [3, 4], [5, 6]];
const flattenedArray = nestedArray.flat();
console.log(flattenedArray); // Output: [1, 2, 3, 4, 5, 6]


// Remove Duplicate Elements from an Array

const numbers = [1, 2, 3, 2, 1, 4, 5];
const uniqueNumbers = [...new Set(numbers)];
console.log(uniqueNumbers); // Output: [1, 2, 3, 4, 5]

// Check if an Array is Empty

const arr = [];
console.log(arr.length === 0); // Output: true
```

Objects in javascript

```
// Object in the js ---> key value pairs
//property(key)is string and the value can be string , number ,
function , array

//object has been created
const person = new Object();

//Add Properties
person.firstname = "John";
person.lastname = "Doe"
person.age = 30;

const person2 = {
    firstname: "Anna" ,
    lastname: "Green"
}

console.log(person)
console.log(person2)
console.log(person.firstname)


// Object.keys --> Returns the array of objects propert

console.log(Object.keys(person))

//Object.values
console.log(Object.values(person))

//return key value pair --> Object.entries

console.log(Object.entries(person))

//Object freeze
// Object.freeze(person) ==> cannot modify
person.age = 26
console.log(person)

//Object.seal ==> can modify but cannot add or delete the
properties
```

```
// Object.seal(person)
person.age = 40;
console.log(person)

person.city = "NYC"
console.log(person)

//Object has own property
// returns true

console.log(person.hasOwnProperty("firstname"))

// Object.is --> compares two values and returns true if they are
the same values

console.log(Object.is(NaN,NaN))
console.log(Object.is(+0,-0))

const entries = [["name","Alice"], ["age" , 25]];
console.log(entries)
//Object.fromEntries--> convert a list of key value pair to
object

const personobj = Object.fromEntries(entries);
console.log(personobj)
personobj.city = "nyc"
console.log(personobj)

//object.getOwnPropetry  returns array of all ownproperties name
keys
```

Spread Operator

```
const arr1 = [1,2,3]
const arr2 = [...arr1 , 8,9]
console.log(arr1)
console.log(arr2)

const arr3 = [3,4,5]
const arr4 = [...arr1 , ...arr3]
console.log(arr4)

const arr5 = [1, ...arr1 , 2]
console.log(arr5)

const myvehivle ={
brand: "ford" ,
model: "mustang" ,
color: "red"
}

const updatedvehile = {
    color: "black"
}

const myupdatedvehile = {...myvehivle , ...updatedvehile}
console.log(myupdatedvehile)
console.log(myvehivle)

const originalobj ={
    a:1,
    b:{
       c: 2},

    d:4
}

const shallowcopy = {...originalobj}

shallowcopy.b.c = 5;
console.log(originalobj)
console.log(shallowcopy)

//diff is because of primitive value(string , number) and
reference Types(object,array)
```

Create an Object to Represent a Person

Create an object person that represents a person with properties: firstName, lastName, and age. Then, log the full name of the person (combine firstName and lastName).

Add and Access Properties Dynamically

Add a new property email to the person object dynamically, and then log it.

Modify Object Properties

Modify the age property of the person object to 35. Then, log the updated object.

Check if Property Exists in an Object

Check if the age property exists in the person object. If it exists, log the age; otherwise, log a message that the property does not exist.

Delete a Property from an Object

Delete the age property from the person object and then log the object.

Object Method Example

Create an object car with properties make, model, and year. Add a method getCarInfo() that returns the car's information as a string, and log the result.

Object Destructuring

Using object destructuring, extract the properties firstName and lastName from the person object and log them.

Object Spread Operator

Create a new object updatedPerson by copying all properties from person, but change the age to 40. Log both person and updatedPerson.

Loop Through Object Properties

Write a function that loops through the properties of the person object and logs both the key and the value.

Nested Object Access

Access the city property from the address object inside person and log it.

Object Method Returning Another Object

Write a method createEmployee() inside an object that returns another object with name and position properties.

Object Freezing

Freeze the person object to prevent adding new properties, and attempt to add a new property. Check if it is added.

Object Merging

Merge two objects person and address into one new object fullInfo. Log the merged object.
Object Property Short-Hand

Refactor the following code using shorthand property names:

const firstName = 'John';
const lastName = 'Doe';
const person = { firstName: firstName, lastName: lastName };

```
// 1. Create an Object to Represent a Person
// Question: Create an object person that represents a person
with properties: firstName, lastName, and age. Then, log the full
name of the person (combine firstName and lastName).
const person = {
  firstName: 'John',
  lastName: 'Doe',
  age: 30,
};

console.log(person.firstName + ' ' + person.lastName); // Output:
"John Doe"

// 2. Add and Access Properties Dynamically
// Question: Add a new property email to the person object
dynamically, and then log it.

const person = {
  firstName: 'John',
  lastName: 'Doe',
  age: 30,
};

person.email = 'john.doe@example.com';
console.log(person.email); // Output: "john.doe@example.com"


// 3. Modify Object Properties
// Question: Modify the age property of the person object to 35.
Then, log the updated object.
```

```
const person = {
  firstName: 'John',
  lastName: 'Doe',
  age: 30,
};

person.age = 35;
console.log(person); // Output: { firstName: 'John', lastName:
'Doe', age: 35 }

// 4. Check if Property Exists in an Object
// Question: Check if the age property exists in the person
object. If it exists, log the age, otherwise log a message that
the property does not exist.

const person = {
  firstName: 'John',
  lastName: 'Doe',
};

if ('age' in person) {
  console.log(person.age);
} else {
  console.log('Age property does not exist');
}
// Output: "Age property does not exist"

// 5. Delete a Property from an Object
// Question: Delete the age property from the person object and
then log the object.

const person = {
  firstName: 'John',
  lastName: 'Doe',
  age: 30,
};

delete person.age;
console.log(person); // Output: { firstName: 'John', lastName:
'Doe' }
```

```
// 6. Object Method Example

const car = {
  make: 'Toyota',
  model: 'Corolla',
  year: 2021,
  getCarInfo: function() {
    return `${this.year} ${this.make} ${this.model}`;
  },
};

console.log(car.getCarInfo()); // Output: "2021 Toyota Corolla"


// 7. Object Destructuring
// Question: Using object destructuring, extract the properties
firstName and lastName from the person object and log them.

const person = {
  firstName: 'John',
  lastName: 'Doe',
  age: 30,
};

const { firstName, lastName } = person;
console.log(firstName, lastName); // Output: "John Doe"
// 8. Object Spread Operator
// Question: Create a new object updatedPerson by copying all
properties from person, but change the age to 40. Log both person
and updatedPerson.

const person = {
  firstName: 'John',
  lastName: 'Doe',
  age: 30,
};

const updatedPerson = { ...person, age: 40 };

console.log(person);         // Output: { firstName: 'John',
lastName: 'Doe', age: 30 }
console.log(updatedPerson); // Output: { firstName: 'John',
lastName: 'Doe', age: 40 }
```

```
// 9. Loop Through Object Properties
// Question: Write a function that loops through the properties
of the person object and logs both the key and the value.
const person = {
  firstName: 'John',
  lastName: 'Doe',
  age: 30,
};

for (const key in person) {
  if (person.hasOwnProperty(key)) {
    console.log(key + ': ' + person[key]);
  }
}


// 10. Nested Object Access
// Question: Access the city property from the address object
inside person and log it.

const person = {
  firstName: 'John',
  lastName: 'Doe',
  age: 30,
  address: {
    street: 'street1',
    city: 'NYC',
    zip: '10001',
  },
};

console.log(person.address.city);

// 11. Object Method Returning Another Object
// Question: Write a method createEmployee() inside an object
that returns another object with name and position properties.

const company = {
  createEmployee: function(name, position) {
    return {
      name: name,
      position: position,
```

```
    };
  },
};

const employee = company.createEmployee('Anna', 'Dev');
console.log(employee); // Output: { name: 'Anna', position: 'Dev'
}

// 12. Object Freezing
// Question: Freeze the person object to prevent adding new
properties, and attempt to add a new property. Check if it is
added.

const person = {
  firstName: 'John',
  lastName: 'Doe',
  age: 30,
};

Object.freeze(person);
person.email = 'john.doe@abc.com'; // This won't work because the
object is frozen

console.log(person.email); // Output: undefined

// 13. Object Merging
// Question: Merge two objects person and address into one new
object fullInfo. Log the merged object.

const person = {
  firstName: 'John',
  lastName: 'Doe',
  age: 30,
};

const address = {
  street: 'street1',
  city: 'NYC',
  zip: '10001',
};

const fullInfo = { ...person, ...address };
console.log(fullInfo);
```

```
// 14. Object Property Short-Hand
// Question: Refactor the following code using shorthand property
names:

const firstName = 'John';
const lastName = 'Doe';

const person = { firstName: firstName, lastName: lastName };
console.log(person);

const firstName = 'John';
const lastName = 'Doe';

const person = { firstName, lastName }; // shorthand property
console.log(person);
// Output: { firstName: 'John', lastName: 'Doe' }
```

```
// let person ={
//     firstname: "John",
//     lastname: "Doe",
//     getFullname() {
//         return `${this.firstname}${this.lastname}`
//     }
// ,
//     setFullName(name){
//         let[first,last] = name.split(" ");
//             this.firstname =first;
//             this.lastname = last;

//     }

// }

// console.log(person.getFullname());
// person.setFullName("Jane Smith");
// console.log(person.getFullname());
```

1. Normal function

```
// let person ={
//     firstname: "John",
//     lastname: "Doe",
//     getFullname() {
//         return `${this.firstname}${this.lastname}`
//     }
// ,
//     setFullName(name){
//         let[first,last] = name.split(" ");
//             this.firstname =first;
//             this.lastname = last;

//     }

// }

// console.log(person.getFullname());
// person.setFullName("Jane Smith");
// console.log(person.getFullname());
```

2. Getter and setter

```
let person ={
    firstname: "John",
    lastname: "Doe",

    get fullName(){
        return `${this.firstname} ${this.lastname}`
    }
,
    set fullName(name) {
        let[first,last]  = name.split(" ");
        this.firstname =  first;
        this.lastname = last;
    }

}



console.log(person.fullName)
```

```
person.fullName = "Jane Smith"
console.log(person.fullName)
```

Sets

```
//set is a collection of unique values .
//value can be primitive or object
// maintains the insertion order

const set = new Set();

//methids of set

//add

set.add(1)
set.add(2)
set.add(1)

//delete method

set.delete(2)
console.log(set)

//has

console.log(set.has(2))
console.log(set.has(1))


//clear

console.log(set.clear());
console.log(set)


//size

console.log(set.size)

set.add(1)
```

```
set.add(2)
set.add(4)
//Iteration method


console.log(set.values())
console.log(set.keys())

//spread operator
console.log([...set])
```

Exercises on Set

1. **Remove Duplicates from an Array**
   Write a function that removes all duplicates from an array using a Set.
2. **Intersection of Two Arrays**
   Find the common elements between two arrays using a Set.
3. **Unique Characters in a String**
   Write a function that checks if a string contains all unique characters.
4. **Count Unique Elements**
   Write a function that counts the number of unique elements in an array using a Set.

Exercises on Map

1. **Count Frequency of Elements**
   Write a function that counts the frequency of each element in an array using a Map.
2. **Group by Property**
   Given an array of objects, group the objects by a specified property using a Map.

Input 1: Group people by age

const people = [

{ name: "Alice", age: 25 },

{ name: "Bob", age: 25 },

```
  { name: "Charlie", age: 30 }
];
console.log(groupBy(people, "age"));
```

Output 1:

```
Map(2) {
  25 => [ { name: "Alice", age: 25 }, { name: "Bob", age: 25 } ],
  30 => [ { name: "Charlie", age: 30 } ]
}
```

3. **Remove Duplicate Keys**
   Write a function that removes duplicate keys from an array of key-value pairs using a Map.
4. **Convert Map to Object**
   Write a function to convert a Map into a plain JavaScript object.
5. **Reverse a Map**
   Write a function to reverse the keys and values of a Map.

Input sample: [

['a', 1],

['b', 2],

['c', 3]

]);

Output: Map { 1 => 'a', 2 => 'c' }

Counting the frequency of each ele

```
Function counFrequency(array){
Const countMap = new Map();
array.forEach(item =>{
countMap.set(item, countMap.get(item) || 0 ) +1 );


}
```

1. for Loop

1. Write a for loop to generate a Fibonacci sequence of a given length n.
2. Write a for loop to find all prime numbers between 1 and 50.

---

2. while Loop

1. Use a while loop to print all numbers in reverse order from 100 to 1.
2. Write a while loop to repeatedly divide a number by 2 until it becomes less than 1. Count the number of divisions.

---

3. do...while Loop

1. Use a do...while loop to prompt the user to enter a number greater than 10. Continue prompting until the condition is met.

Hint **-: num = parseInt(prompt("Enter a number greater than 10:"), 10);**

2. Write a do...while loop to find the sum of digits of a number.

Solution:

```
let number = 54321; // The input number
```

```
let sum = 0;     // Variable to store the sum of the digits


do {
        sum += number % 10;           // Add the last digit of the number to the sum
        number = Math.floor(number / 10); // Remove the last digit by dividing by 10 and taking the floor
} while (number > 0);              // Repeat until there are no digits left


console.log(sum); // Outputs: 15
```

Step-by-Step Explanation:

1. **Initialization**:
    - Declare a variable number and assign it the value 54321 (the input number).
    - Initialize another variable sum to 0. This will be used to store the cumulative sum of the digits as they are processed.
2. let number = 54321;
3. let sum = 0;
4. **Start the do...while loop**:
    - A do...while loop ensures the block of code inside it runs **at least once**, even if the condition is false initially.
5. do {
6. // Code to process each digit of the number
7. } while (number > 0);
8. **Extract the last digit**:
    - Use the modulo operator (%) to extract the last digit of number. For example, 54321 % 10 gives 1 (the last digit).
    - Add this digit to the sum variable.
9. sum += number % 10; // Add the last digit to the sum
10. **Remove the last digit**:
    - Use integer division to remove the last digit from number. Divide number by 10 and use Math.floor() to discard the decimal part.

Example: Math.floor(54321 / 10) gives 5432.

11. number = Math.floor(number / 10); // Remove the last digit
12. **Repeat the process**:
    - The loop repeats the steps of extracting and removing the last digit until number becomes 0.
13. **Terminate the loop**:

- The condition number > 0 is checked at the end of each iteration. Once number becomes 0, the loop terminates.

14. **Output the result**:
    - After the loop ends, the value of sum contains the total sum of the digits. Print the result using console.log().
15. console.log(sum); // Outputs: 15

Example Walkthrough:

Initial Values:

- number = 54321
- sum = 0

Iteration	number % 10 (Last Digit)	sum After Addition	Math.floor(number / 10) (Updated number)
1	1	0 + 1 = 1	5432
2	2	1 + 2 = 3	543
3	3	3 + 3 = 6	54
4	4	6 + 4 = 10	5
5	5	10 + 5 = 15	0

Final Output:

- sum = 15

---

4. for...of Loop

1. Use a for...of loop to filter out all odd numbers from an array.
2. Write a for...of loop to concatenate all strings in an array into a single string.

---

5. for...in Loop

1. Write a for...in loop to calculate the total of all numeric values in an object.

Step-by-Step Hints:

Hint 1: Define the object

Create an object with a mix of numeric and non-numeric values. For example:

const obj = { a: 10, b: 20, c: "hello", d: 30 };

This object contains four key-value pairs, where keys a, b, and d have numeric values, and key c has a string value.

Hint 2: Initialize a total sum variable

Before starting the loop, declare a variable total and set its value to 0.

let total = 0;

This variable will hold the cumulative sum of numeric values.

Hint 3: Use a for...in loop

Iterate through all the keys in the object using a for...in loop.

```
for (const key in obj) {
    // Code inside the loop
}
```

The loop will cycle through each key in the object (a, b, c, d).

Hint 4: Check if the value is a number

Inside the loop, use typeof to determine if the value associated with the current key is a number.

```
if (typeof obj[key] === "number") {
    // Code to handle numeric values
}
```

This ensures that non-numeric values (like strings) are ignored.

Hint 5: Add numeric values to the total

If the value is a number, add it to the total variable.

```
if (typeof obj[key] === "number") {
        total += obj[key];
}
```

For example:

- On the first iteration, total = 10 (value of a).
- On the second iteration, total = 30 (value of b added).
- The string value at c is skipped.
- On the fourth iteration, total = 60 (value of d added).

Hint 6: Output the final total

After the loop ends, log the final value of total to the console.

```
console.log(total); // Outputs: 60
```

The total sum of the numeric values in the object is 60.

Question:

2. Write a for...in loop to create a string of all the keys in an object, separated by commas.

Sol:

Step 1: Understand the Problem

We have an object with key-value pairs. The goal is to extract all the keys, combine them into a single string, and separate them with commas.

Step 2: Create the Object

Define the object with some key-value pairs.

```
const obj = { a: 1, b: 2, c: 3 };
```

Here, the object obj has the following key-value pairs:

- a: 1
- b: 2
- c: 3

Step 3: Declare a Variable to Store Keys

Initialize an empty string to store the keys during the loop.

```
let keys = "";
```

Step 4: Use a for...in Loop to Iterate Through Keys

Use the for...in loop to iterate over each key in the object.

```
for (const key in obj) {
        keys += key + ", ";
}
```

Explanation:

- key: Represents the current property (or key) being accessed in the object.
- keys += key + ", ": Appends the current key followed by a comma and a space.

Step 5: Remove the Trailing Comma

After the loop completes, the keys string will have an extra comma and space at the end. Use the slice() method to remove the last two characters.

```
console.log(keys.slice(0, -2));
```

Final Solution

```
const obj = { a: 1, b: 2, c: 3 };

let keys = "";

for (const key in obj) {
        keys += key + ", ";
}

console.log(keys.slice(0, -2));
```

6. Array.prototype.forEach()

1. Use `forEach` to calculate the product of all numbers in an array.
2. Write a program using `forEach` to count the occurrences of each character in a string.

---

7. Array.prototype.map()

1. Use `map` to create a new array where each element is the square of the original array.
2. Write a program using `map` to capitalize the first letter of each word in an array.

---

8. Array.prototype.filter()

1. Use `filter` to create a new array containing only the numbers greater than 10 from the original array.
2. Write a program using `filter` to extract all words that start with the letter "a" from an array.

Hint : use word.startsWith("a")) method

---

9. Array.prototype.reduce()

1. Use `reduce` to find the maximum number in an array.

```
const max = numbers.reduce((a, b) => (  //logic  ))
```

2. Write a program using `reduce` to concatenate all elements of an array into a single string.

```
const sentence = words.reduce((a, b) => a + " " + b);
```

---

10. Recursion

1. Write a recursive function to calculate the sum of all numbers from 1 to `n`.

```
let number = 54321;

let sum = 0;
```

```
do {
        sum += number % 10;
console.log(sum)
        number = Math.floor(number / 10);
} while (number > 0);
                console.log(sum); \
```

2. Write a recursive function to reverse a string.

Solution:

```
function reverseString(str) {
	if (str.length === 0) return "";
	return str[str.length - 1] + reverseString(str.slice(0, -1));
}
console.log(reverseString("hello")); // Outputs: "olleh"
```

Reverse the number

```
function reverseNumber(num) {
  let reversedNum = 0;

  // Handle negative numbers
  let isNegative = num < 0;
  num = Math.abs(num);  // Work with the absolute value for reversal

  while (num > 0) {
	let digit = num % 10;  // Get the last digit
	reversedNum = reversedNum * 10 + digit;
	num = Math.floor(num / 10);
  }
```

```
  return isNegative ? -reversedNum : reversedNum;  // Restore the negative sign if needed
}

let num = 12345;
let reversed = reverseNumber(num);
console.log(reversed);  // Outputs: 54321

let negativeNum = -12345;
let reversedNegative = reverseNumber(negativeNum);
console.log(reversedNegative);
```

Lexical Scoping and closures

```
// // lexical scooping ---> functions scope s determined by its
position in the source code . Functions are executed in the scope in
which they were defined not where they were called

// function outer() {

//     let outervar = " I am outer var";
//     function inner(){
//        console.log(outervar)
//     }
//    inner()

// }

// outer();


// function createCounter(){
//     let count = 0;
//     return {
//         increment(){
//             count++
//             console.log(count);
//         },
//         decrement(){
```

```
//             count --;
//             count--;
//             console.log(count)
//         }
//     };
// }


// const counter1 = createCounter();
// counter1.increment();
// counter1.increment();


// const coounter2 = createCounter();
// coounter2.increment()
```

Hoisting in Javascript

Hoisting Summary Table

Type	Hoisted?	Behavior Before Initialization
var	Yes	undefined
let	Yes (TDZ applies)	ReferenceError
const	Yes (TDZ applies)	ReferenceError
Function Declarations	Yes	Can be called before declaration
Function Expressions	No	undefined if assigned to var, error otherwise
Arrow Functions	No	undefined if assigned to var, error otherwise

```
// greet();
// function greet() {
//     console.log("Hello")
// }

//arrow function --> no
// const greet = () =>
// {
//     console.log("Greet")
// }
```

```
//

// const greet = function() {
//     console.log("hello")
// }



// variable

// console.log(x);
// var x = 1;

console.log(x)    //Reference error
let x =2
//let and const -->  Accessing the variable before declaration throe=ws
reference error
```

Function calling and function bind

Bind –. Returns a new function , does not call immediately

```
function greet(name) {
    console.log(`This is ${this.name}`);
}

const person = {name: "Anna"}

const new1 = greet.bind(person)
console.log(new1)
new1();
```

Function call

Doesnot return new function , immediately calls

```
function greet(name) {
    console.log(`This is ${this.name}`);
}

const person = {name: "Anna"}


greet.call(person)
```

This c , func bind and call

Function Type	Method	Behavior	Example
Regular Function	`.bind()`	Creates a new function with `this` permanently bound to the specified object.	`function greet() { console.log(this.name); } const bound = greet.bind({ name: 'Alice' }); bound(); // "Alice"`
	`.call()`	Immediately invokes the function with `this` explicitly set to the provided object.	`function greet() { console.log(this.name); } greet.call({ name: 'Bob' }); // "Bob"`
	`.apply()`	Same as `.call()`, but takes an array of arguments instead of a comma-separated list.	`function greet(msg) { console.log(`${msg}, ${this.name}`); } greet.apply({ name: 'Charlie' }, ['Hello']); // "Hello, Charlie"`
Function Expression	`.bind()`	Works the same way as with regular functions since function expressions are still regular functions.	`const greet = function() { console.log(this.name); }; const bound = greet.bind({ name: 'Eve' }); bound(); // "Eve"`

	`.call()`	Works the same way as with regular functions.	`const greet = function(msg) { console.log(`${msg}, ${this.name}`); }; greet.call({ name: 'Dan' }, 'Hi'); // "Hi, Dan"`
	`.apply()`	Works the same way as with regular functions.	`const greet = function(msg) { console.log(`${msg}, ${this.name}`); }; greet.apply({ name: 'Fay' }, ['Hey']); // "Hey, Fay"`
Arrow Function	`.bind()`	Has no effect, because arrow functions inherit `this` from their lexical scope (not dynamically bound).	`const greet = () => console.log(this.name); const bound = greet.bind({ name: 'Grace' }); bound(); // undefined (or global context)`
	`.call()`	Cannot change `this`, as it is inherited from the lexical scope.	`const greet = () => console.log(this.name); greet.call({ name: 'Hank' }); // undefined (or global context)`
	`.apply()`	Cannot change `this`, same as `.call()`.	`const greet = () => console.log(this.name); greet.apply({ name: 'Ivy' }); // undefined (or global context)`

```
//arrow functions do not have their own this . they inherit from
surrounding lexical scope . If u need to use this , use regular
functions



peron={
    name: "John",
    age: 21 ,
    greet: function ()
    {
        console.log(this.name);
    }
}
```

```
//arrow function inside the other function
function outer(){
    const self = this;
    self.name = "john;"
    const greet =() =>
    {
        console.log(this.name);
    };
    greet();
}

outer();


const p1 = {
    name: "john",
    greet: function() {
        const innergreet = () =>
        {
            console.log(this.name);
        };
        innergreet();
    }
}

p1.greet();
```

```
//exception handelling -- >

//commonn Javascript Errors
```

Syntax Error —>
Type Error : ull.toString()
Reference Error → accessing a variable which is not defined”
Range Error → number outside the valid range – > new Array(-1
URIError -

Try {
//code that may throw an error
}
Catch(error) {
//handling error}
Finally{
//code that runs regardless of error}

```
property --> name : type of error
Message → the message
Stack → call stack tree(use for debugging)
```

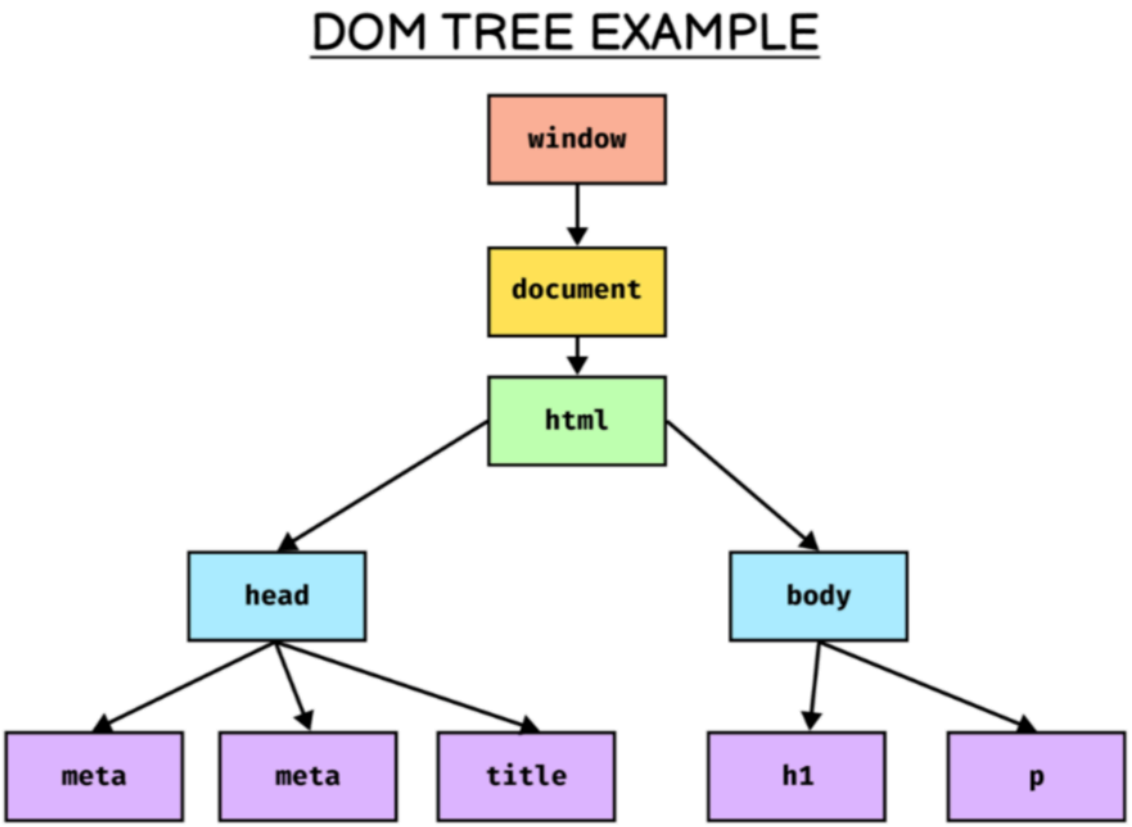
Transpilation – Babel –< handle syntax related isse
Example `//nullish coalescing (??)`

```
// height = height ??100;
```

Polyfills _> handle environment related issues ‘ → use library like core-js

```
if(!Math.truc){
    Math.truc = function(number){
        // logic
    }
}
```

DOM
Is a programming interface for web documents



Root node → html
Child Node → head and body
Leaf nodes → Text nodes or empty tags can be leaf nodes

Modify the HTML elements
Change CSS Styles
Respond to user events
Add remove or modify attributes

Basic Dom Manipulation methods

a. **Accessing the elements - > getelementById**
b. **getElementByClassName**
c. **querySelector**
d. **querySelectorAll: Select all the elements that match a css selector**

Changing the elements
Let element= document.getElementById(“myElement”)
element.innerHTML = “NewHTMLcontent;
element.textContent = “New text content;

Changing the Styles

Element.style.color = “red”

Element.style.fontSize = “red”

Creating or Appeanding the elements

document.createElement(“ul”)
Remove elements
removeChild

JS Events
Event Listeners
Quick and simple events
Event llisteners – multiple event handlers

Click : triggered when mouse button is clicked
Submit : TRIGGERED WHEN FORM IS SUBMITTED
KEYDOWN/KEYup : TRIGGERED WHEN KEY IS PRESSED OR RELEASED
MOUSEOVER / MOUSEOUT :
FOCUS/BLUR : TRIGGERED WHEN ELEMENT GAINS OR LOSES FOCUS
CHANGE : TRIGGERED WHEN THE VALUE OF INPUT FEILD CHANGES

eVENT oBJECTS

Event.target : element that triggered the event
Event.type
event.preventDefault() :
event.stopPropogation() : stops the event from propagating to other elements

Take array input from user and then display it

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Document</title>
</head>
```

```
<body>
    <p id ="array"></p>
    <script>
        let arr = new Array();
        while(arr.length < 5){
            const input = prompt(`Enter the ${arr.length}th
number`,"");
            if(input!== null && input.trim() !=""){
                arr.push(input)
            }
            else{
                if(arr.length>0)
            {
                alert("you Entered less than 5 values ")
                break;
            }
            }
        }
        const output1  = document.getElementById('array')
        array.textContent = `You entered ${arr.join(",")}`;
    </script>
</body>
</html>
```

JavaScript DOM Manipulation Assignment Questions

1. **Change Text Content**
    - **You have a paragraph element with an ID `demo`. Create a function that changes the text inside this paragraph when a button is clicked. What will you do to update the text inside the paragraph?**
2. **Add a New Item to a List**
    - **Create a function to add a new list item to an unordered list with the ID `myList`. The new item should be called "New Item". How do you append the new list item to the list?**

      Hint:

```
const newItem = document.createElement("li");

newItem.textContent = "New Item";

document.getElementById("myList").appendChild(newItem);
```

3. **Remove an Element**
    - **You have a div element with the ID `myDiv`. Write a function that removes this div element when a button is clicked.**
4. **Toggle Visibility of an Element**
    - **Write a function to toggle the visibility of a paragraph element with the ID `toggleParagraph`. The paragraph should hide or show when a button is clicked.**
    - **Display**
    - **function toggleVisibility() {**
    - **const para = document.getElementById("toggleParagraph");**
    - **if (para.style.display === "none") {**
    - **para.style.display = "block";**
    - **} else {**
    - **para.style.display = "none";**
    - **}**
    - **}**
    -
5. **Change the Background Color of a Div**
    - **Create a function that changes the background color of a div element with the ID `colorDiv` to "orange" when a button is clicked.**
6. **Greet the User Based on Input**
    - **You have an input field where a user can enter their name (ID `username`). Create a function that reads the user's name and displays a greeting message below the input field.**

```
function greetUser() {

    const name = document.getElementById("username").value;

    const greetingMessage = document.getElementById("greetingMessage");

    greetingMessage.textContent = `Hello, ${name}! Welcome to our website.`;

}
```

7. **Change InnerHTML of an Element**

- You have a div element with the ID `container`. Write a function that changes its `innerHTML` to display a new paragraph saying "This is a modified paragraph."

8. Change OuterHTML of an Element
   - You have a div element with the ID `container`. Create a function that changes the `outerHTML` to replace the div with a section containing a new heading and paragraph.
9. Toggle Between Two Classes
   - You have a button element with the ID `toggleButton` and two classes: `active` and `inactive`. Write a function that toggles between these two classes when the button is clicked.

```
button.classList.toggle("active");
        button.classList.toggle("inactive");
```

10. `Scroll Event Handler -->`

```
    <!-- Log "Scrolling..." to the console whenever the user scrolls
the page. -->
```

Hint : use ‘scroll’ event

```
taskItem.querySelector('.task-name').classList.toggle('completed');
```

Lab Manual: To-Do List with Due Dates

Objective

The goal of this lab is to create a responsive To-Do List application with due dates. It includes features such as adding tasks, marking tasks as completed, identifying overdue tasks, and clearing all tasks.

Pre-requisites

1. **Basic understanding of HTML, CSS, and .**
2. **Familiarity with Bootstrap Icons (optional).**

Steps to Create the Application

Step 1: Setting Up the HTML Structure

1. **Create an index.html file.**
2. **Add the following structure for your application:**

```
<!DOCTYPE html>
 <html lang="en">
 <head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width,
initial-scale=1.0">
      <title>To-Do List with Due Dates</title>
 </head>
 <body>
      <div class="container">
      <h1>To-Do List</h1>

      <div class="addtask">
            <input id="taskInput" type="text" placeholder="Add a new
task...">
            <input id="dueDateInput" type="datetime-local">
            <button id="addtask">Add Task</button>
      </div>

      <ul id="taskList"></ul>
      <button id="clearAllBtn">Clear All Tasks</button>
      </div>
 </body>
 </html>
```

3. **Save the file and open it in your browser. You should see a title, input fields, and a button.**

Step 2: Adding Styling

1. **Inside the <head> section, include the following:**

```
<link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap-icons/font/bootstrap-icons.css">
 <style>
      body {
      font-family: 'Poppins', sans-serif;
      background: linear-gradient(to right, #6a11cb, #2575fc);
      color: #fff;
      margin: 0;
      padding: 0;
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
      }
```

```
      .container {
      background: #fff;
      border-radius: 15px;
      padding: 30px 20px;
      max-width: 500px;
      width: 90%;
      box-shadow: 0 10px 25px rgba(0, 0, 0, 0.2);
      }

      h1 {
      text-align: center;
      font-size: 2rem;
      margin-bottom: 20px;
      color: #6a11cb;
      }

      .addtask input, .addtask button {
      padding: 12px;
      margin-bottom: 10px;
```

```
      font-size: 1rem;
      border: 2px solid #ddd;
      border-radius: 10px;
      }

      ul {
      list-style: none;
      padding: 0;
      }

      li {
      background: #f9f9f9;
      padding: 15px;
      margin-bottom: 10px;
      border-radius: 10px;
      }
 </style>
```

2. **Save and refresh your browser. Your app now has a cleaner, modern design.**

Step 3: Adding for Task Management

1. **Add the following <script> tag before the closing </body> tag:**

html

```
<script>
      const taskInput = document.getElementById('taskInput');
      const dueDateInput = document.getElementById('dueDateInput');
      const addTaskBtn = document.getElementById('addTaskBtn');
      const taskList = document.getElementById('taskList');

      // Add task functionality
      addTaskBtn.addEventListener('click', () => {
      const taskName = taskInput.value.trim();
      const dueDate = dueDateInput.value;

      if (!taskName || !dueDate) {
            alert('Please enter both a task and a due date!');
            return;
      }

      const taskItem = document.createElement('li');
      taskItem.innerHTML = `
             <span>${taskName}</span>
```

```
            <span>${new Date(dueDate).toLocaleString()}</span>
        `;
        taskList.appendChild(taskItem);
        taskInput.value = '';
        dueDateInput.value = '';
        });
</script>
```

2. **Save and refresh your browser. Test adding tasks and see them displayed.**

Step 4: Enhancing Task Functionality

1. **Add buttons to mark tasks as completed and delete tasks:**

```
const taskItem = document.createElement('li');
 taskItem.innerHTML = `
        <span>${taskName}</span>
        <span>${new Date(dueDate).toLocaleString()}</span>
        <button class="complete">Complete</button>
        <button class="delete">Delete</button>
 `;
 taskList.appendChild(taskItem);

 // Mark as complete
 taskItem.querySelector('.complete').addEventListener('click', () =>
{
        taskItem.querySelector('span').classList.toggle('completed');
 });

 // Delete task
 taskItem.querySelector('.delete').addEventListener('click', () => {
        taskItem.remove();
 });
```

Step 5: Handling Overdue Tasks

1. **Write a function to check for overdue tasks:**

```
function checkOverdue() {
      const tasks = taskList.querySelectorAll('li');
      tasks.forEach(task => {
      const dueDateText =
task.querySelector('span:nth-child(2)').textContent;
      const dueDate = new Date(dueDateText);
      const currentDate = new Date();

      if (currentDate > dueDate) {
            task.style.color = 'red';
      }
      });
 }
 setInterval(checkOverdue, 60000); // Check every minute
```

Step 6: Adding a Clear All Functionality

1. **Implement the clear all functionality:**

```
const clearAllBtn = document.getElementById('clearAllBtn');
clearAllBtn.addEventListener('click', () => {
      if (confirm('Are you sure you want to delete all tasks?')) {
      taskList.innerHTML = '';
      }
});
```

Step 7: Testing the Application

1. **Open the file in your browser.**
2. **Test all features:**
    - **Adding tasks with a due date.**
    - **Marking tasks as completed.**
    - **Deleting individual tasks.**
    - **Automatically highlighting overdue tasks.**
    - **Clearing all tasks.**