MERN Stack Training

Tasks

JavaScript Language - An Introduction to JavaScript, Code structure:

1. An Introduction to JavaScript:

Task 1: alert("Hello, World!");

Task 2:

```
let name = "Amar";
console.log("String example:");
console.log("Name:", name);
console.log("Type:", typeof name);
let age = 20;
console.log("\nNumber example:");
console.log("Age:", age);
console.log("Type:", typeof age);
let isStudent = true;
console.log("\nBoolean example:");
console.log("Is Student:", isStudent);
console.log("Type:", typeof isStudent);
```

```
String example:
Name: Amar
Type: string

Number example:
Age: 20
Type: number

Boolean example:
Is Student: true
Type: boolean
```

Task3:

```
let addResult = 10 + 5;
console.log("Addition: 10 + 5 =", addResult);
let subtractResult = 10 - 5;
console.log("Subtraction: 10 - 5 =",
subtractResult);
let multiplyResult = 10 * 5;
console.log("Multiplication: 10 * 5 =",
multiplyResult);
```

```
Addition: 10 + 5 = 15
Subtraction: 10 - 5 = 5
Multiplication: 10 * 5 = 50
```

Task4:

```
let s = "Good" + "Afternoon";
console.log(s);
```

```
PROBLEMS    OUTPUT

GoodAfternoon
```

Task5:

```
let name ="Amar";
let age=20;
let student=true;
console.log("type name-", typeof name);
console.log("type age-", typeof age);
console.log("type student-",typeof student);
```

```
type name- string
type age- number
type student- boolean
```

2. Code structure:

Task 6:

```
// This is a single-line comment

/*
This is a multi-line comment.
It can span multiple lines.
```

Task 7:

```
//with semicolon
Let a=10;
Let b=10;
//Without Semicolon
Let a=10
```

Task8:

```
for (let i = 1; i <= 2; i++) {
    console.log("Outer loop: " + i);
```

```
  Inner loop: 1
  Inner loop: 2
Outer loop: 2
  Inner loop: 1
  Inner loop: 2
```

```
    for (let j = 1; j <= 2; j++) {
        console.log("  Inner loop: " + j);
    }
}
```

Task 9:

```
Amar 20
```

```
let a ="Amar", age=20;
console.log(a,age);
```

Task 10:

Script at the Top:

- When the <script> tag is at the top, the JavaScript runs **before** the page content is displayed. This can make the page load slower because the browser waits for the script to finish first.

Script at the Bottom:

- When the <script> tag is at the bottom, the JavaScript runs **after** the page content is displayed. This helps the page load faster because the browser shows the content first and runs the script afterward.

The modern mode, “use strict”, Variables:

1. The modern mode, “use strict”:

Task 11:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE
Amar
```

```
a ="Amar";
console.log(a);
```

Task 12:

```
"use strict";
a ="Amar";
console.log(a);
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   GITLENS   AZURE   DEVDB   Filter (e.g. text, !exclude, \escape)
> Uncaught ReferenceError ReferenceError: a is not defined
    at <anonymous> (c:\Users\Amar\Downloads\New folder (5)\index.html:13:3)
```

Task 13:

```
"use strict";
    let a = "amar";
    console.log(a);
    delete a;
```

```
PROBLEMS 1   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   GITLENS   AZURE   DEVDB   Filter (e.g. text, !exclude, \escape)
> Uncaught SyntaxError SyntaxError: Delete of an unqualified identifier in strict mode.
    at (program) (c:\Users\Amar\Downloads\New folder (5)\index.html:17:16)
```

Task 14:

```
PROBLEMS   OUTPUT   DEBUG CONSOLE
amar
```

Without "use strict":

```
a = "amar";
    console.log(a);
```

With "use strict":

```
"use strict";
    a = "amar";
    console.log(a);
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   GITLENS   AZURE   DEVDB   Filter (e.g. text, !exclude, \escape)
> Uncaught ReferenceError ReferenceError: a is not defined
    at <anonymous> (c:\Users\Amar\Downloads\New folder (5)\index.html:13:11)
```

Task 15:

```
"use strict";
        var eval = "test";
        console.log(eval);
```

2. Variables:

Task 16:

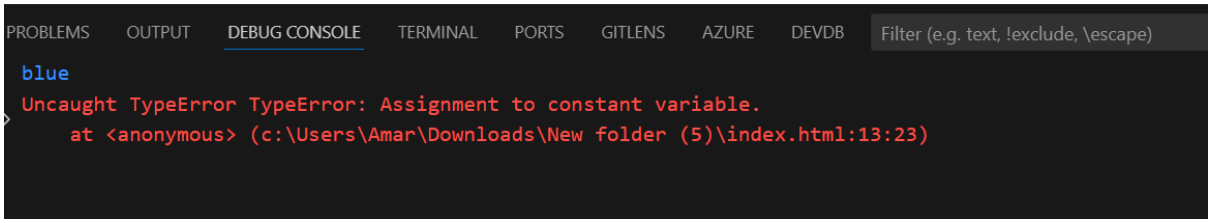
```
let age = 25;
console.log(age);
age = 20;
console.log(age);
const birthYear = 2004;
console.log(birthYear);
var city = "Cbe";
console.log(city);
city = "Cbe";
console.log(city);
```

- **let**:
- Use when the variable's value needs to be reassigned or updated. It is block-scoped, which makes it suitable for use inside loops or conditionals.
- **const**:
- Use when the variable's value should remain constant throughout its scope. It is also block-scoped, but it cannot be reassigned after initialization.
- **var**:
- Avoid using in modern JavaScript. Use var only if you need

function-scoped variables or are working with legacy code, as it has issues with hoisting and scoping compared to let and const.

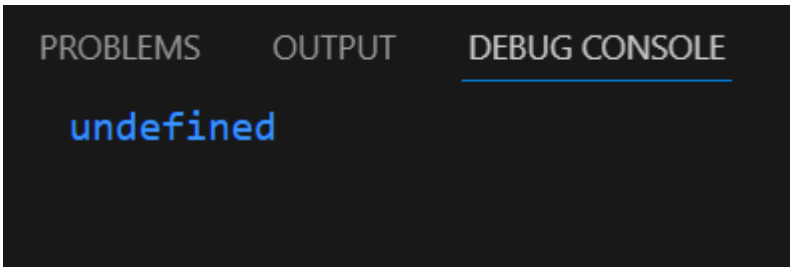
Task 17:

```
const favoriteColor = "blue";
console.log(favoriteColor);
favoriteColor = "green";
```



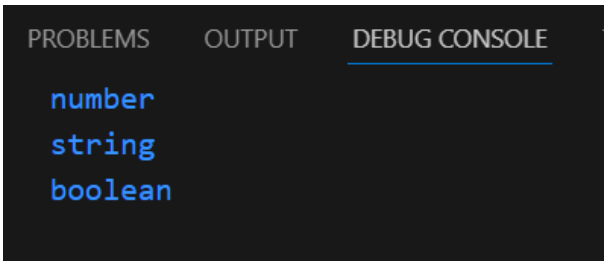
Task 18:

```
let num;
console.log(num);
```



Task 19:

```
let numberValue = 42;
let stringValue = "Hello";
let booleanValue = true;
console.log(typeof numberValue);
console.log(typeof stringValue);
console.log(typeof booleanValue);
```



Task 20:

```
let oldName = "Amar";
    let newName = oldName;
    console.log(newName);
    oldName = "Amarnaath";
    console.log(oldName);
    console.log(newName);
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE
Amar
Amarnaath
Amar
```

Data types, Basic operators, maths:

1. Data types:

Task 21:

```
let name = "Amar";
let age = 20;
let isstudent = true;
let favoriteColor = null;
let hobby;
console.log("Name:", name);
console.log("Age:", age);
console.log("Is Student:", isstudent);
console.log("Favorite Color:", favoriteColor);
console.log("Hobby:", hobby);
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE
Name: Amar
Age: 20
Is Student: true
Favorite Color: null
Hobby: undefined
```

Task 22:

```
let name = "Amar";
let age = 20;
```

```
let isstudent = true;
let favoriteColor = null;
let hobby;
console.log("Type of name:", typeof name);
console.log("Type of age:", typeof age);
console.log("Type of isStudent:", typeof isStudent);
console.log("Type of favoriteColor:", typeof favoriteColor);
console.log("Type of hobby:", typeof hobby);
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL
Type of name: string
Type of age: number
Type of isStudent: undefined
Type of favoriteColor: object
Type of hobby: undefined
```

Task 23:

```
let mySymbol = Symbol("example");
console.log("Type of mySymbol is:", typeof mySymbol);
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL
Type of mySymbol is: symbol
```

Task 24:

```
let varl = null;
console.log("Type of varl:", typeof varl);
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE
Type of varl: object
```

Task 25:

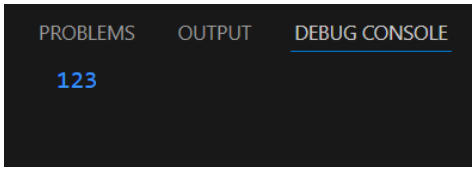
- **var**: Has **function scope**. If declared inside a function, it is accessible throughout the function, even before the declaration (due to hoisting). If declared outside a function, it is a global variable.
- **let**: Has **block scope**. It is only accessible within the block ({}) where it is declared, such as inside loops or conditionals, and is not hoisted.

2. Basic operators, maths:

Task 26:

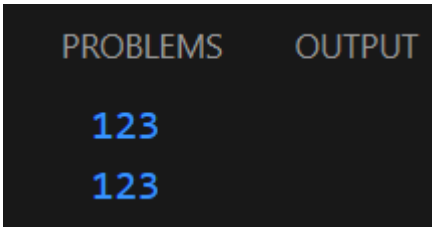
Implicit Conversion

```
let str = "123";
    let num = str * 1;
    console.log(num);
```



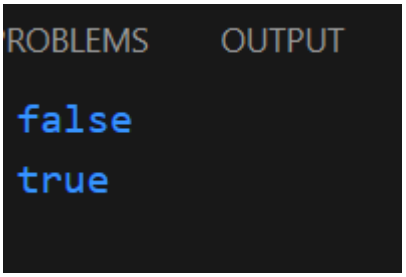
Explicit Conversion:

```
let str = "123";
    let num1 = Number(str);
    console.log(num1);
    let num2 = parseInt(str);
    console.log(num2);
```



Task 27:

```
let bool = false;
let boolToStr = String(bool);
console.log(boolToStr);
let str = "hello Hiiii";
```



```
let strToBool = Boolean(str);
console.log(strToBool);
```

Task 28:

```
let addition = a + b;
let subtraction = a - b;
let multiplication = a * b;
let division = a / b;
let modulus = a % b;
console.log(addition);
console.log(subtraction);
console.log(multiplication);
console.log(division);
console.log(modulus);
```

```
PROBLEMS    OUTPUT

15
5
50
2
0
```

Task 29:

```
let num = 10;
num++;
console.log("incrementing: " + num);
num--;
console.log("decrementing: " + num);
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE

incrementing: 11
decrementing: 10
```

Task 30:

```
let a = 1;
    let b = 10;
    let c = 2
    let result = a + b - c;
    console.log("Result:" + result);
```

```
PROBLEMS    OUTPUT

Result:9
```

Comparisons, Conditional branching: if, ‘?’:

Task 31:

```
let num1 = 10;
    let num2 = 5;
    console.log(num1 > num2);
    console.log(num1 < num2);
    console.log(num1 >= num2);
    console.log(num1 <= num2);
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE

true
false
true
false
```

Task 32:

```
let a = 10;
    let b = 5;
    console.log(a===b);
```

```
PROBLEMS    OUTPUT

false
```

Task 33:

```
let str = "apple";
    let str1 = "mango";
    console.log(str===str1);
    console.log(str > str1);
    console.log(str < str1);
```

```
PROBLEMS    OUTPUT

false
false
true
```

Task 34:

```
    let str = "apple";
    let str1 = "mango";
    console.log(str != str1);
    console.log(str!==str1);
```

```
PROBLEMS    OUTPUT

true
true
```

Task 35:

```
let str = null;
let str1 = undefined;
console.log(str == str1);
console.log(str===str1);
```

```
PROBLEMS    OUTPUT
true
false
```

2. Conditional branching: if, '?':

Task 36:

```
let a=2;
if(a%2==0){
 console.log("even");
}
else{
 console.log("odd");
}
```

```
PROBLEMS    OUTPUT
even
```

Task 37:

```
let a = 2;
 if (a > 0) {
    console.log("positive");
 } else {
    if (a < 0) {
       console.log("negative");
    } else {
       console.log("zero");
    }
 }
```

```
PROBLEMS    OUTPUT
positive
```

Task 38:

```
let a = 2;
a > 0 ? console.log("positive") : console.log("negative");
```

```
PROBLEMS    OUTPUT
positive
```

Task 39:

```
let person = {
    name: "Amar",
    age: 20,
};
let name = person?.age;
console.log(name);
```

```
PROBLEMS    OUTPUT
20
```

Task 40:

```
let number = -5;
let result = (number > 0) ? "Positive" : "Negative";
console.log(result);
```

```
PROBLEMS    OUTPUT
Negative
```

Logical operators, Functions:

1. Logical operators:

Task 41:

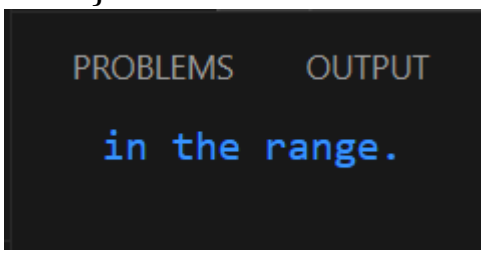
```
let a = true;
let b = false;
let c = true;
```

```
PROBLEMS    OUTPUT
true
true
false
true
true
```

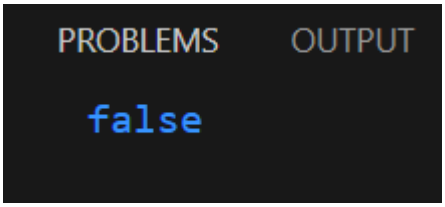
```
let d = false;
console.log(a && c)
console.log(a || c);
console.log(!a);
console.log((a || b) && c);
console.log(!(a && b));
```

Task 42:

```
let number = 15;
let lowerBound = 10;
let upperBound = 20;
if (number >= lowerBound && number <= upperBound) {
    console.log("in the range.");
} else {
    console.log("out of the range.");
}
```



Task 43:




```
let isRaining = true;
let isNotRaining = !isRaining;
console.log(isNotRaining);
```

Task 44:

```
let a = false;
let b = true;
if (a && b) {
    console.log("This won't run because a is false.");
} else {
    console.log("Short-circuiting happened. The expression didn't
```

```
evaluate b.");
}
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    GITLENS    AZURE
Short-circuiting happened. The expression didn't evaluate b.
```

Task 45:

```
let a = "Hello";
        let b = 0;
        let result = a && b;
        console.log(result);
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE
0
```

2. Functions:

Task 46:

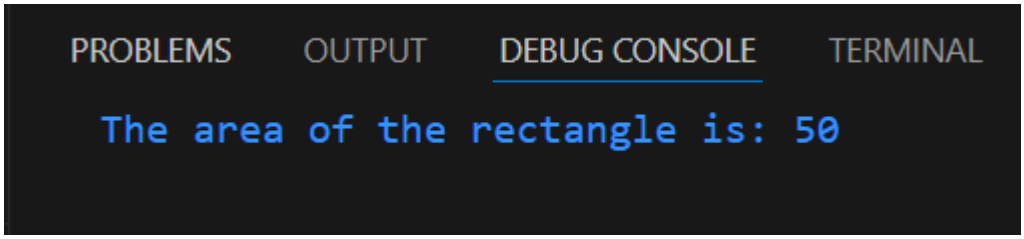
```
function addNumbers(num1, num2) {
    return num1 + num2;
}
let result = addNumbers(5, 10);
console.log("The sum is: " + result);
```

```
PROBLEMS    OUTPUT
The sum is: 15
```

Task 47:

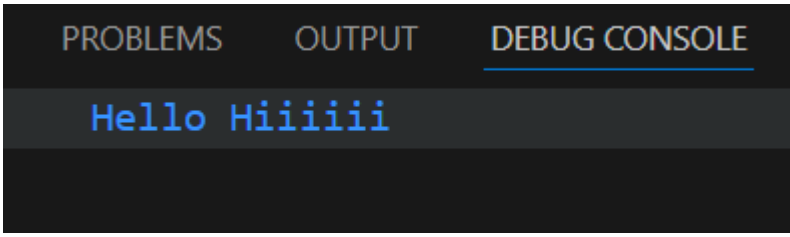
```
function calculateRectangleArea(length, width) {
    return length * width;
```

```
}
let length = 5;
let width = 10;
let area = calculateRectangleArea(length, width);
console.log("The area of the rectangle is: " + area);
```



Task 48:

```
function mern() {
   console.log("Hello Hiiiiii");
}
mern();
```



Task 49:

```
function displayMessage() {
   console.log("This function does not return a value.");
}
let result = displayMessage();
console.log("The default return value is: " + result);
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL
This function does not return a value.
The default return value is: undefined
```

Task 50:

```
function mern(name = "name", message = "Welcome") {
    console.log(`Hello, ${name}! ${message}`);
}

mern("Amar", "Good to see uu");
mern("Amar");
mern();
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE
Hello, Amar! Good to see uu
Hello, Amar! Welcome
Hello, name! Welcome
```

3. Arrow Functions:

Task 51:

```
function mern(name = "name", message = "Welcome") {
```

```
    console.log(`Hello, ${name}! ${message}`);
}

mern("Amar", "Good to see uu");
mern("Amar");
mern();
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE
Hello, Amar! Good to see uu
Hello, Amar! Welcome
Hello, name! Welcome
```

Task 52:

```
const add = (a, b) => a + b;
console.log(add(5, 3));
console.log(add(10, 15));
console.log(add(-5, 7));
console.log(add(0, 0));
```

```
PROBLEMS    OUTPUT
8
25
2
0
```

Task 53:

```
const isEven = num => num % 2 === 0;
```

```
console.log(isEven(4));
console.log(isEven(7));
console.log(isEven(0));
console.log(isEven(-2));
console.log(isEven(-3));
```

```
PROBLEMS    OUTPUT

  true
  false
  true
  true
  false
```

Task 54:

```
const maxValue = (a, b) => {
   return a > b ? a : b;
};
console.log(maxValue(10, 5));
console.log(maxValue(7, 20));
console.log(maxValue(-3, -7));
console.log(maxValue(15, 15));
console.log(maxValue(0, 1));
```

```
PROBLEMS    OUTPUT

  10
  20
  -3
  15
  1
```

Task 55:

```
const myObject = {
   value: 10,
   multiplyTraditional: function(number) {
      console.log("Inside traditional function, this.value:", this.value);
      return this.value * number;
```

```
    },
    multiplyArrow: (number) => {
        console.log("Inside arrow function, this.value:", this.value);
        return this.value * number;
    }
};
console.log(myObject.multiplyTraditional(2));
console.log(myObject.multiplyArrow(2));
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

Inside traditional function, this.value: 10
20
Inside arrow function, this.value: undefined
NaN
```