DAY 1 → Javascript

Introduction to JavaScript

JavaScript is a widely-used programming language that is primarily used for building interactive web applications. It provides the necessary tools and functionalities to add dynamic behavior to web pages. In this topic, we will cover the basics of JavaScript, including variables, data types, operators, and control flow.

1. Basics of JavaScript:
- JavaScript is a client-side scripting language that runs in the browser.
- It is used to enhance the functionality and interactivity of web pages.
- JavaScript code is embedded directly within HTML files or referenced externally.

2. Variables:
- Variables are used to store data values that can be accessed and manipulated throughout the program.
- To declare a variable, use the "var", "let", or "const" keyword followed by the variable name.
- Example:

```
var age = 25;
let name = "John";
const PI = 3.14;
```

3. Data Types:
   - JavaScript has several data types, including:
     - Numbers: integers and floating-point numbers.
     - Strings: sequences of characters.
     - Booleans: true or false values.
     - Arrays: ordered collections of values.
     - Objects: key-value pairs of properties.
   - Example:
     ```
     var age = 25; // Number
     var name = "John"; // String
     var isStudent = true; // Boolean
     var numbers = [1, 2, 3]; // Array
     var person = { name: "John", age: 25 }; // Object
     ```

4. Operators:
   - JavaScript supports various operators for performing operations on variables and values.
   - Arithmetic operators: +, -, *, /, % (modulo), etc.
   - Comparison operators: ==, ===, !=, !==, <, >, <=, >=, etc.
   - Logical operators: && (AND), || (OR), ! (NOT), etc.
   - Example:
     ```
     var x = 5;
     var y = 3;
     var sum = x + y; // Addition
     var isGreater = x > y; // Comparison
     var isTrue = !false; // Logical negation
```

```

5. Control Flow:
   - Control flow allows you to dictate the order in which statements are executed.
   - Conditional statements: if, else if, else.
   - Loops: for, while, do-while.
   - Example:
    ```
    var age = 18;
    if (age >= 18) {
      console.log("You are an adult.");
    } else {
      console.log("You are a minor.");
    }
    ```

Hands-on Project: Writing JavaScript code to solve basic programming problems.

Problem Statements:

1. Write a JavaScript program to find the maximum of two numbers.
   Answer:
   ```javascript
   function findMax(a, b) {
     return (a > b) ? a : b;
   }
   ```
   ```

2. Write a JavaScript program to check if a number is even or odd.

Answer:

```javascript
function checkEvenOdd(num) {
  return (num % 2 === 0) ? "Even" : "Odd";
}
```

3. Write a JavaScript program to concatenate two strings.

Answer:

```javascript
function concatenateStrings(str1, str2) {
  return str1 + str2;
}
```

4. Write a JavaScript program to calculate the factorial of a number.

Answer:

```javascript
function factorial(num) {
  if (num === 0 || num === 1) {
    return

1;
  }
  return num * factorial(num - 1);
}
```

```

5. Write a JavaScript program to check if a string is a palindrome.
   Answer:
   ```javascript
   function isPalindrome(str) {
     var reverseStr = str.split('').reverse().join('');
     return (str === reverseStr) ? true : false;
   }
   ```

6. Write a JavaScript program to find the sum of all elements in an array.
   Answer:
   ```javascript
   function findArraySum(arr) {
     var sum = 0;
     for (var i = 0; i < arr.length; i++) {
       sum += arr[i];
     }
     return sum;
   }
   ```

7. Write a JavaScript program to remove duplicates from an array.
   Answer:
   ```javascript
   function removeDuplicates(arr) {
```
```

```
  return [...new Set(arr)];
}
```

8. Write a JavaScript program to reverse a string.
   Answer:
   ```javascript
   function reverseString(str) {
     return str.split('').reverse().join('');
   }
   ```

9. Write a JavaScript program to find the largest element in an array.
   Answer:
   ```javascript
   function findLargestElement(arr) {
     return Math.max(...arr);
   }
   ```

10. Write a JavaScript program to calculate the average of numbers in an array.
    Answer:
    ```javascript
    function calculateAverage(arr) {
      var sum = 0;
      for (var i = 0; i < arr.length; i++) {
        sum += arr[i];
      }
```

```
  return sum / arr.length;
}
```

Let's discuss primitive and non-primitive datatypes in JavaScript, as well as the difference between the `==` and `===` comparison operators.

Primitive Datatypes:
In JavaScript, there are six primitive datatypes:

1. Number: Represents numeric values, e.g., `10`, `3.14`, etc.
2. String: Represents textual data enclosed in single or double quotes, e.g., `"Hello"`, `'World'`, etc.
3. Boolean: Represents either `true` or `false`.
4. Undefined: Represents a variable that has been declared but has not been assigned a value.
5. Null: Represents the intentional absence of any object value.
6. Symbol (added in ECMAScript 6): Represents a unique and immutable value that may be used as an identifier for object properties.

Primitive datatypes are immutable, meaning their values cannot be changed once they are assigned. They are passed by value when assigned to variables or passed as function arguments.

Non-Primitive Datatypes:

In JavaScript, non-primitive datatypes are also known as reference types. They include:

1. Object: Represents a collection of key-value pairs enclosed in curly braces, e.g., `{}` or `{ name: "John", age: 25 }`.
2. Array: Represents an ordered collection of values enclosed in square brackets, e.g., `[]` or `[1, 2, 3]`.
3. Function: Represents a reusable block of code that performs a specific task.

Non-primitive datatypes are mutable, meaning their values can be changed even after assignment. They are passed by reference when assigned to variables or passed as function arguments.

Difference between `==` and `===`:
In JavaScript, the `==` and `===` operators are used for comparison. Here's the difference between them:

1. `==` (Equality Operator): The `==` operator compares the values of two operands after performing type coercion if necessary. It converts the operands to a common type and then checks for equality.

Example: `1 == '1'` returns `true` because the operands are converted to the same type (`1` and `1`) before comparison.

2. `===` (Strict Equality Operator): The `===` operator, also known as the strict equality operator, compares both the

values and the types of the operands. It returns `true` only if both values are of the same type and have the same value.

Example: `1 === '1'` returns `false` because the operands have different types (`number` and `string`).

In general, it is recommended to use the `===` operator (strict equality) because it performs a more precise comparison by considering both the values and the types. The `==` operator (equality) may lead to unexpected results due to type coercion.

Understanding the distinction between primitive and non-primitive datatypes, as well as the differences between `==` and `===`, will help you write more reliable and predictable JavaScript code.

````
1. Question: What will be the output of the following code?
   ```javascript
   console.log(foo);
   var foo = "Hello";
   ```
   Answer: The output will be `undefined`. In JavaScript, variable declarations using `var` are hoisted to the top of their scope, but the initializations are not. So, `var foo` is hoisted and declared but remains undefined until the line `foo = "Hello"` is executed.

2. Question: What will be the output of the following code?
   ```javascript
   console.log(bar);
   let bar = "World";
   ```
````

Answer: ReferenceError: `bar` is not defined. Unlike `var`, variables declared with `let` and `const` are not hoisted to the top of their scope. Therefore, the `console.log` statement tries to access an undefined variable, resulting in an error.

3. Question: What will be the output of the following code?

```javascript
let x = "outside";
function foo() {
  console.log(x);
  let x = "inside";
}
foo();
```

Answer: ReferenceError: `x` is not defined. In this case, `let x` is hoisted to the top of the `foo` function scope, but it is not initialized before the `console.log` statement. Hence, it throws an error.

4. Question: What will be the output of the following code?

```javascript
var x = 1;
if (function f() {}) {
  x += typeof f;
}
console.log(x);
```

Answer: The output will be `"1undefined"`. Although the function `f` is defined within the `if` statement, it does not have a name in the outer scope. Therefore, `typeof f` returns `"undefined"`, which is concatenated with the existing value of `x`.

5. Question: What will be the output of the following code?

```javascript
var x = 1;
if (true) {
  var x = 2;
  console.log(x);
}
console.log(x);
```

Answer: The output will be `2` and `2`. The `var x` declaration inside the `if` statement has function scope, so it modifies the value of the outer `x` variable.

6. Question: What will be the output of the following code?

```javascript
var x = 1;
function foo() {
  var x = 2;
  console.log(x);
}
foo();
console.log(x);
```

Answer: The output will be `2` and `1`. The `var x` declaration inside the `foo` function creates a new variable with function scope that shadows the outer `x` variable.

7. Question: What will be the output of the following code?

```javascript
var x = 1;
function bar() {
  x = 2;
  console.log(x);
}
bar();
console.log(x);
```

Answer: The output will be `2` and `2`. In this case, the `x` variable is accessed and modified directly within the `bar` function, so it affects the value of the outer `x` variable.

8. Question: What will be the output of the following code?

```javascript
var x = 1;
function baz() {
  console.log(x);
}
(function () {
  var x = 2;
```

```
    baz();
  })();
  ```

  Answer: The output will be `1`. The `x` variable inside the IIFE (Immediately Invoked Function Expression) has a separate scope from the outer `x` variable, so the `baz` function accesses the outer variable and prints its value.

9. Question: What will be the output of the following code?
  ```javascript
  console.log(foo);
  let foo = "Hello";
  ```
  Answer: ReferenceError: `foo` is not defined. Similar to `let`, variables declared with `const` are also not hoisted to the top of their scope. Therefore, trying to access `foo` before its declaration results in an error.

10. Question: What will be the output of the following code?
    ```javascript
    function foo() {
      console.log(bar);
      const bar = "World";
    }
    foo();
    ```
    Answer: ReferenceError: `bar` is not defined. In this case, the variable `bar` is declared with `const` inside the `foo` function, but it is not initialized before the `console.log` statement. Hence, it throws an error.