HyperionDev

Full-Stack Web Development

HyperionDev

JavaScript Fundamentals

Full Stack Web Development Session Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(Fundamental British Values: Mutual Respect and Tolerance)**
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: **Questions**

Full Stack Web Development Session Housekeeping cont.

- For all **non-academic questions**, please submit a query: **www.hyperiondev.com/support**
- Report a **safeguarding** incident: **www.hyperiondev.com/safeguardreporting**
- We would love your **feedback** on lectures: **Feedback on Lectures**

Objectives

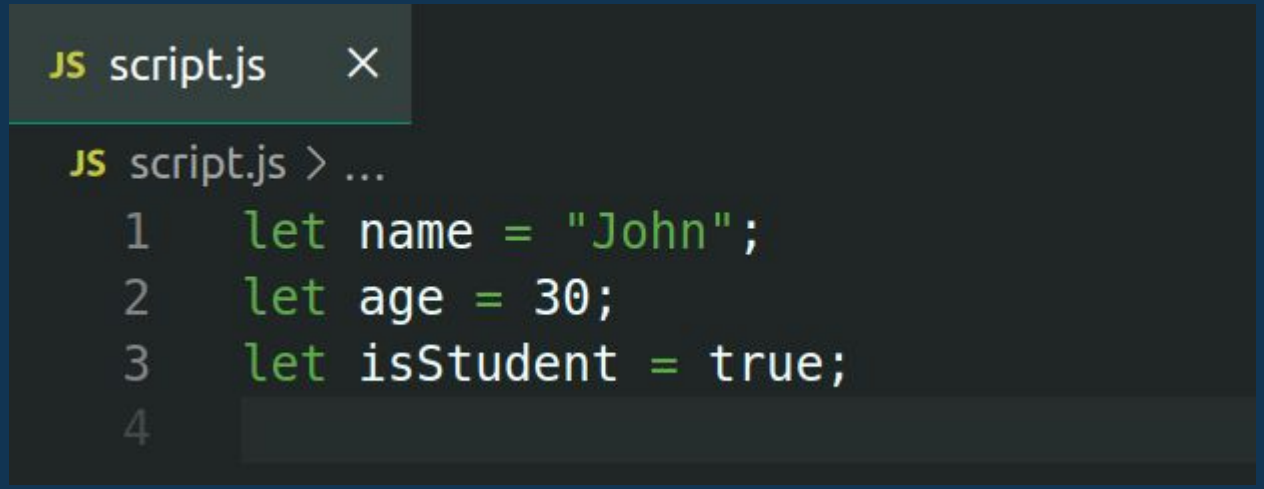
- Understand the importance of JavaScript in modern web development.
- Understand and use variables and different data types in JavaScript.
- Implement iterations using loops.
- Apply control flow statements (if-else).

What is the Relevance of JavaScript?

- JavaScript is one of the core technologies of the web, alongside HTML and CSS.
- It allows for the creation of dynamic content that can respond to user actions, such as clicks and form submissions.
- JavaScript enhances user interaction by providing real-time feedback without needing to reload the entire page.
- Examples include form validation, interactive maps, and dynamic loading of content.
- JavaScript has frameworks and libraries like React, Angular, and Vue.js that have revolutionized front-end development.

Variables and Data Types

- Variables are named storage for data that can be changed.
- Data Types:
  - Primitive: String, Number, Boolean, Null, Undefined, Symbol
  - Non-Primitive: Object, Array
-



let, const, and var

- In JavaScript, variables can be declared using var, let, and const.
- Choosing the right type of variable declaration is crucial for writing clean and maintainable code.
- **var**:
  - Variables declared with var are scoped to the function in which they are declared.
  - Variables declared with var can be redeclared within the same scope without causing an error.

let

- Variables declared with let are scoped to the block, statement, or expression in which they are used.
- This includes loops, if statements, and other block-level constructs.
- Variables declared with let cannot be redeclared within the same block scope, which helps prevent bugs.

const

- Block-Scoped:
  - Like let, variables declared with const are also block-scoped.
- Cannot Be Redeclared:
  - Variables declared with const cannot be redeclared within the same block scope.
- Must Be Initialized:
  - const requires an initial value at the time of declaration. This value cannot be changed (immutable reference).

Operators in JavaScript

- JavaScript has a variety of operators that can be used to perform different types of operations.
  - Arithmetic Operators
  - Comparison Operators
  - Logical Operators

script.js

```
var x = 5 + 2; // x is 7
var y = x * 3; // y is 21
var z = y / 4; // z is 5.25
var k = y % x; // modulus is 0
```

script.js

```
var x = 5;
var y = 2;

console.log(x == y); // false
console.log(x != y); // true
console.log(x > y); // true
console.log(x < y); // false
console.log(x >= y); // true
console.log(x <= y); // false
```

script.js

```
var x = true;
var y = false;

console.log(x && y); // false
console.log(x || y); // true
console.log(!x); // false
```

const

- Block-Scoped:
  - Like let, variables declared with const are also block-scoped.
- Cannot Be Redeclared:
  - Variables declared with const cannot be redeclared within the same block scope.
- Must Be Initialized:
  - const requires an initial value at the time of declaration. This value cannot be changed (immutable reference).

Control Flow in JavaScript

- Control flow determines the order in which statements are executed in a program.
- This allows for decision-making, making your programs dynamic and responsive to different inputs.

If-else statements

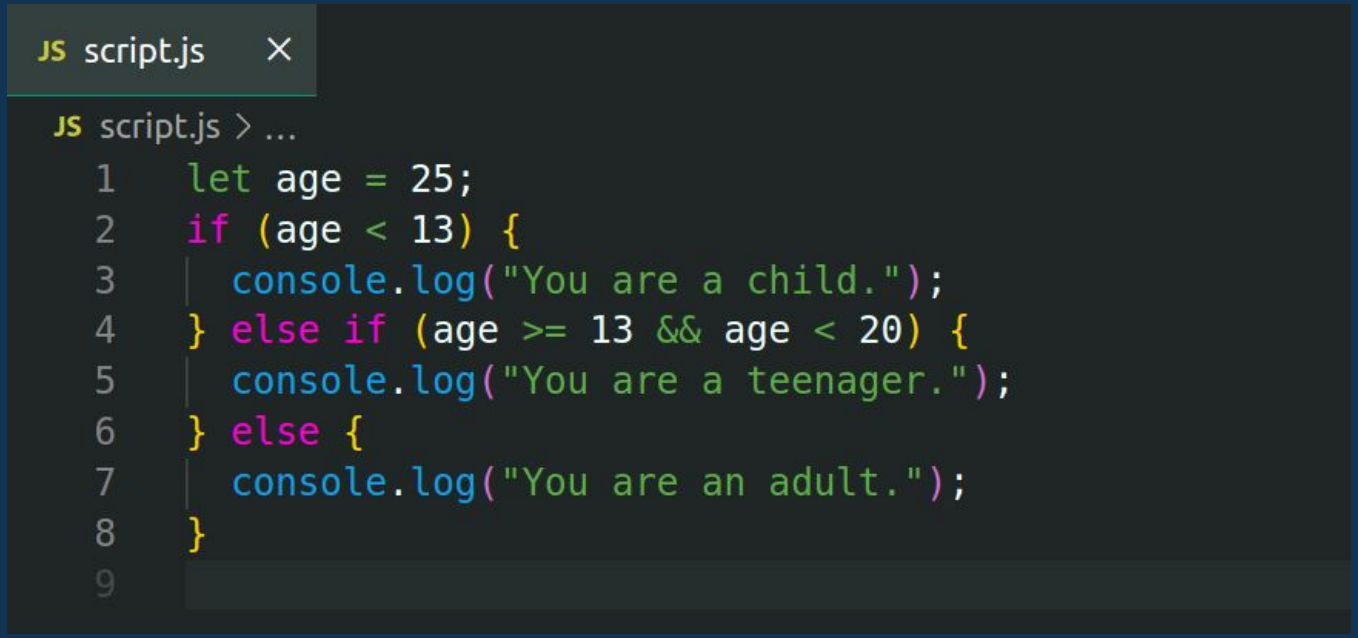
- The purpose of these statements is to execute certain blocks of code based on specific conditions.
- Syntax:

JS script.js

```
JS script.js
1   if (condition) {
2     // code to be executed if condition is true
3   } else {
4     // code to be executed if condition is false
5   }
6
```

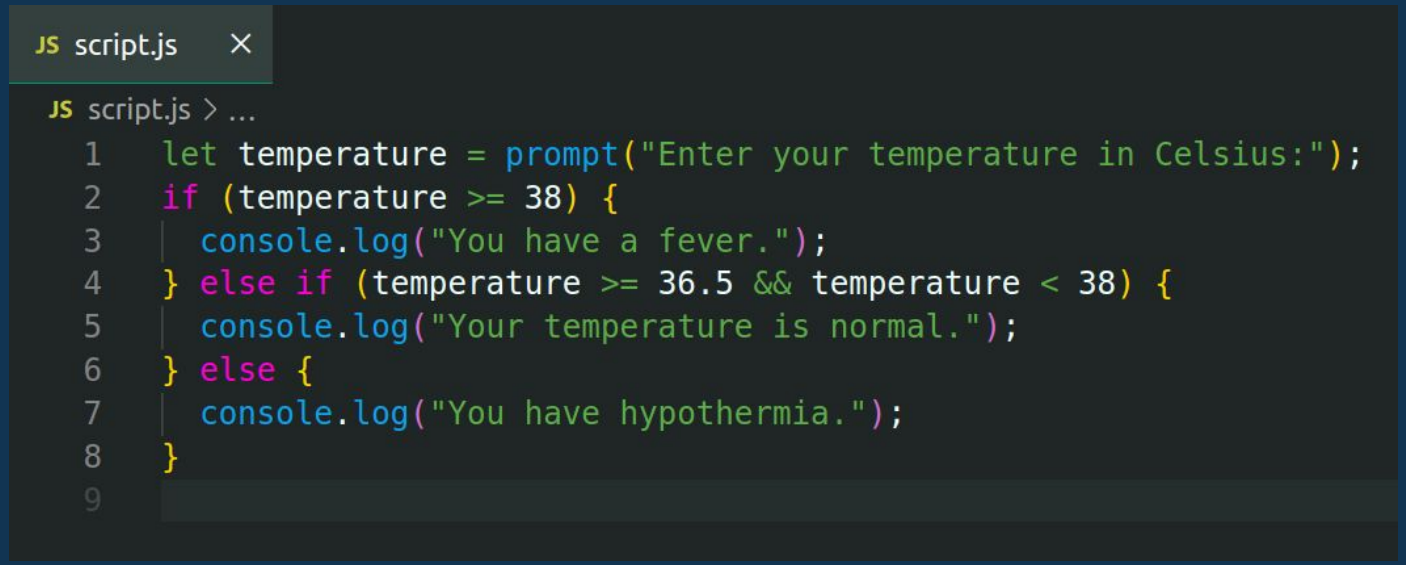
Nested if-else Statements

- **The purpose of nested if-else statements is to test multiple conditions.**



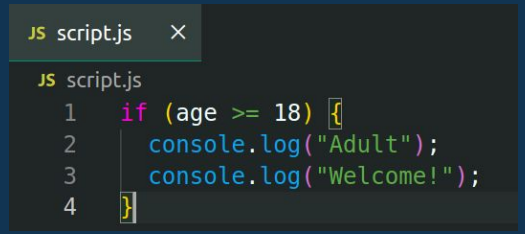
Practical Application

- **Task**: Write a program that takes a user's temperature input and prints out if they have a fever, normal temperature, or hypothermia.
- Example code:



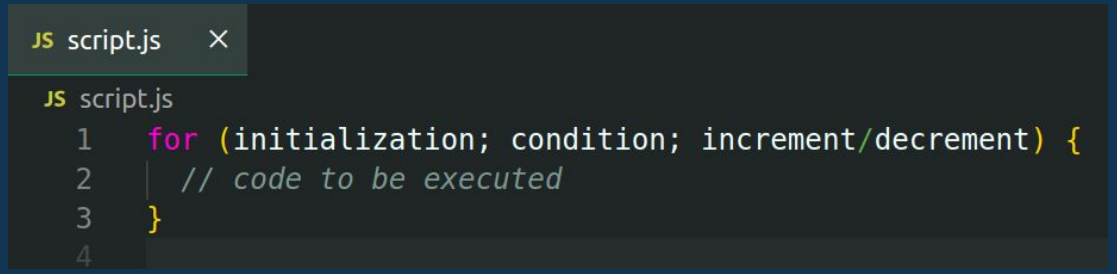
Common Mistakes to Avoid

- **Mistake**: Using a single equals sign (=) instead of double equals (==) or triple equals (===) for comparison.
  - ***Example***: if (age = 18) will assign 18 to age instead of comparing.
  - ***Correct***: if (age == 18) or if (age === 18)
- **Mistake**: Forgetting to use curly braces for blocks of code with multiple statements.
  - ***Example***: if (age >= 18) console.log("Adult"); console.log("Welcome!"); (Second console.log always executes)
  - ***Correct:***



Iterations and Loops
for loop

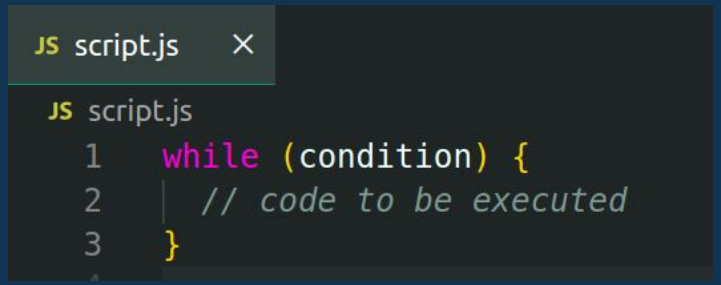
- The `for` loop repeats a block of code a specified number of times.
- Syntax:



- **Initialization**: This is executed once before the loop starts. It is used to initialize variables.
- **Condition**: Before each iteration, the condition is evaluated. If the condition is true, the loop continues. If false, the loop stops.
- **Increment/Decrement**: This is executed after each iteration of the loop, typically used to update the loop counter.

while loop

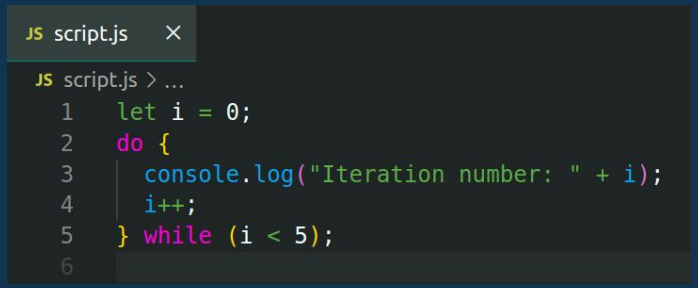
- The `while` loop repeats a block of code as long as a specified condition is true.
- Syntax:



- **Condition**: Before each iteration, the condition is evaluated. If the condition is true, the loop continues. If false, the loop stops.

do-while loop

- The `do-while` loop is similar to the while loop, but it guarantees that the block of code is executed at least once before the condition is tested.
- Syntax:



- The block of code inside the `do` is executed once initially, then the condition is evaluated. If the condition is true, the loop continues. If false, the loop stops.

Code-Along Activity

- **Task**: Write a program that takes user input and prints the multiplication table.

Hyperiondev

Questions and Answers

HyperionDev

Thank You for attending!