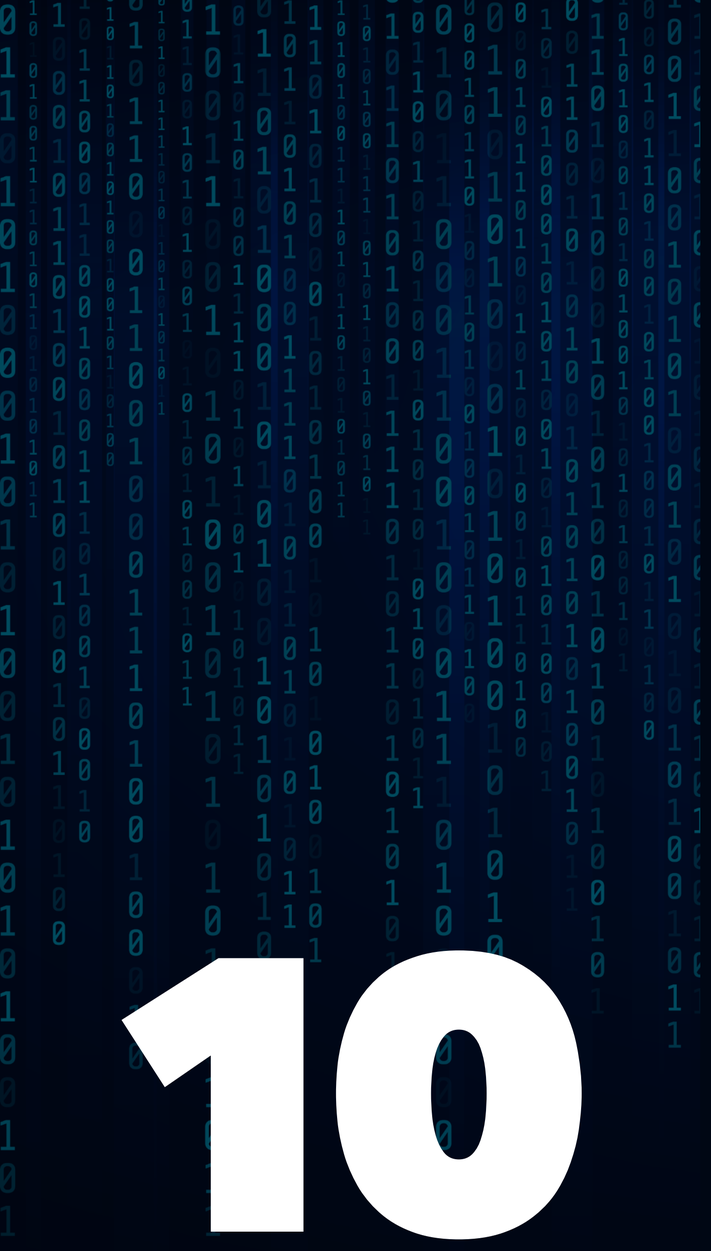
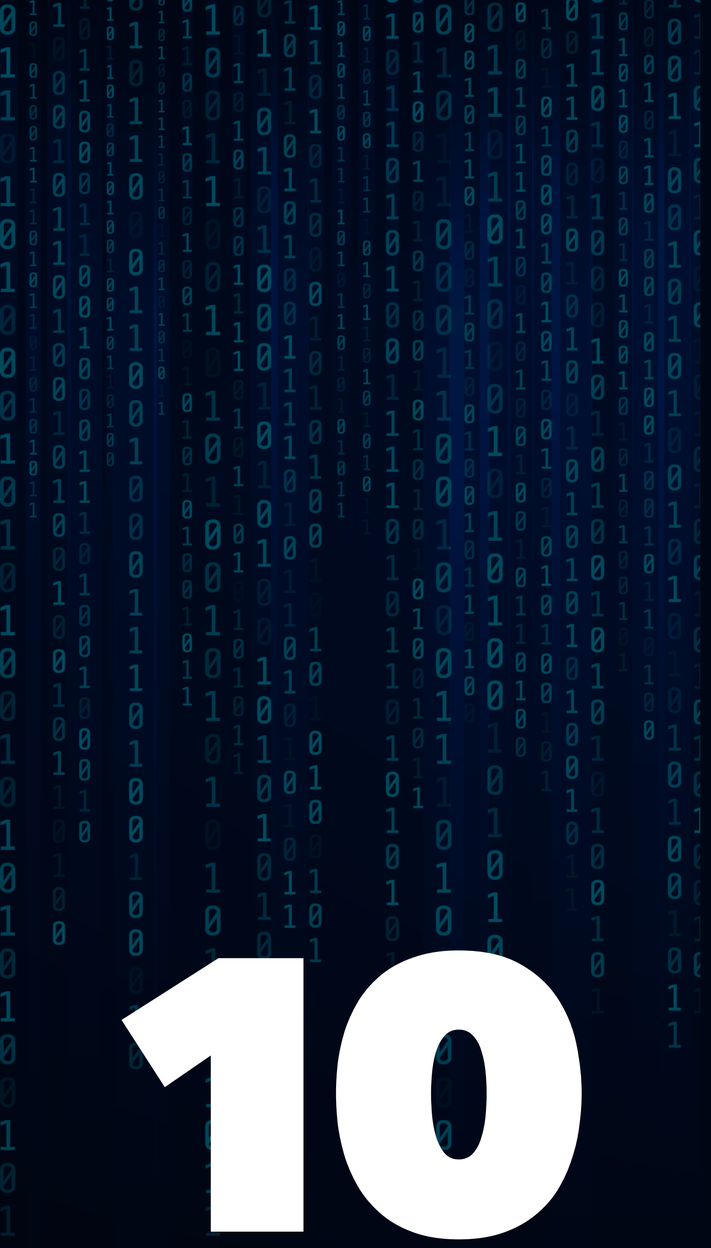


10

INM120

Introduction to Adaptive Web Design



10

JAVESCRIPT REVIEW

Let's take a moment to revisit what we've learned about JavaScript before diving into more exciting topics.

10

JAVASCRIPT OPERATORS:

JavaScript operators are symbols used to perform operations on variables and values. They are the building blocks of expressions, allowing you to manipulate and combine values. Here are some common types of JavaScript operators:

ASSIGNMENT OPERATORS:

Assignment operators in JavaScript are used to assign values to variables. They allow you to take a value or the result of an expression and store it in a variable.

Basic Assignment (=):

The equal sign (=) is used to assign a value to a variable.

Compound Assignment Operators:

JavaScript also provides compound assignment operators, which combine an arithmetic operation with the assignment. Compound assignment operators include **+=** (addition and assign), **-=** (subtract and assign), ***=** (multiply and assign), **/=** (divide and assign), and **%=** (modulus and assign).

10

JAVASCRIPT OPERATORS:

ARITHMETIC OPERATORS:

Arithmetic operators in programming are used to perform mathematical calculations on numeric values. Here's a detailed explanation of each arithmetic operator:

Addition (+):
The addition operator is used to add two numbers.

Subtraction (-):
The subtraction operator is used to subtract the right operand from the left operand.

Multiplication (*):
The multiplication operator is used to multiply two numbers.

Division (/):
The division operator is used to divide the left operand by the right operand.

Modulus (%):
The modulus operator gives the remainder of the division of the left operand by the right operand.

JAVASCRIPT OPERATORS:

NOTE: Concatenation refers to the process of **combining** two or more **strings** into a single string. In programming, this operation is commonly used to join text or string values. In JavaScript, for example, the concatenation operator is the plus sign (+).

```
let firstName = "John";
let lastName = "Doe";
let fullName = firstName + " " + lastName;

console.log(fullName);  // Outputs: John Doe
```

10

10

JAVASCRIPT OPERATORS:

COMPARISION OPERATORS:

Comparison operators in programming are used to **compare values** and **return a Boolean result** (true or false). They are commonly used in **conditional statements** and **loops**.

Equality (== and ===):
== checks for equality, but it performs type coercion, meaning it converts the operands to the same type before making the comparison.
=== checks for strict equality without type coercion. It returns true only if **both** the **value** and the **type** are the same.

Inequality (!= and !==):
!= checks for inequality with type coercion.
!== checks for strict inequality without type coercion.

Greater Than (>):
The greater-than operator checks if the left operand is greater than the right operand.

Less Than (<):
The less-than operator checks if the left operand is less than the right operand.

Greater Than or Equal To (>=):
The greater-than-or-equal-to operator checks if the left operand is greater than or equal to the right operand.

10

JAVASCRIPT OPERATORS:

COMPARISION OPERATORS:

Less Than or Equal To (<=):
The less-than-or-equal-to operator checks if the left operand is less than or equal to the right operand.

LOGICAL OPERATORS:

Logical operations, also known as logical operators, are used in programming to perform logical comparisons between values or expressions. These operators return a Boolean result (either true or false).

Logical AND (&&):
The logical **AND** operator returns true if **both** of the operands are true; otherwise, it returns false.

Logical OR (||):
The logical **OR** operator returns true if at **least one** of the operands is true; it returns false only if both operands are false.

Logical NOT (!):
The logical NOT operator negates the value of its operand. If the operand is true, ! makes it false; if the operand is false, ! makes it true.

IF STATEMENT

An if statement is a fundamental control flow statement in programming that allows you to **conditionally execute a block of code**.

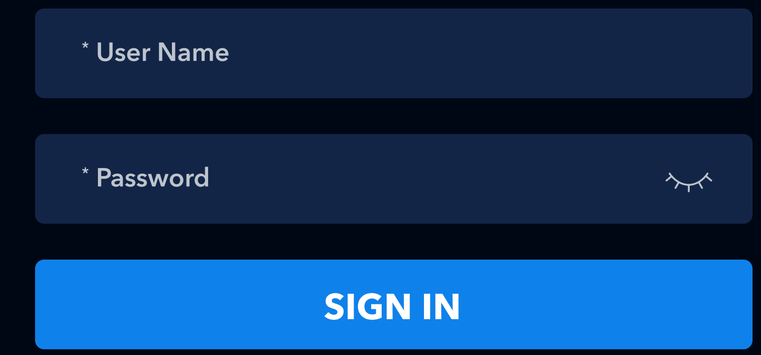
Data on the Server:
let userName = “Kosar”
let password = “WeeWoo123”

Input Value Typed By User:
User Name: Kosar
Password: weewoo123

On press of submit button:

```
if (userName == “Kosar” && password == “WeeWoo123”) {
  // Allow Access to Account
} else {
  // Deny Access
}
```

Result will be **Deny Access**



10

10

FOREACH ( )

forEach() is a method available for arrays in JavaScript. It is used to iterate over each element in the array and execute a provided function once for each element. The purpose of forEach() is to perform an action on each item in the array, without the need for an explicit loop.

Example a Marketing email:

Subject: **Exciting News: Upcoming Sale at the Lego Store!**

Dear **Lego Enthusiast**,

An exciting sale is on the horizon at the Lego Store.
Stay tuned for more details and be prepared to unleash your creativity with amazing discounts on your favourite sets.

Happy building!

Best regards,
Lego

At Lego, To make emails more personal, instead of using 'Lego Enthusiast,' they'll use your name and they won't change names of recipients one by one. Instead why not use for each?

10

FOREACH ( )

On Lego Server:

```
let Recipient = [“Kosar”, “James”, “Simul”];

Recipient.forEach( function(recipientName) {

  console.log(`

        Dear ${recipientName},
        An exciting sale is on the horizon at the Lego Store.
        Stay tuned for more details and be prepared to unleash your
       creativity with amazing discounts on your favourite sets.

        Happy building!

        Best regards,
       Lego

  `) })
```

This message will be logged 3 time in the console, each time changing the **recipientName** to **Kosar**, then to **James**, and finally to **Simul**

10

Concatenation VS Template Literal:

```
const name = "John";
const age = 25;

// Concatenation using the + operator
const messageConcatenation = "Hello, my name is " + name + " and I am " + age + " years old.";
console.log(messageConcatenation);

// Template literal with embedded variables
const messageTemplateLiteral = `Hello, my name is ${name} and I am ${age} years old.`;
console.log(messageTemplateLiteral);
```

In the template literal example, the **variables name** and **age** are **enclosed within ${ } inside backticks**.

This makes the code more **concise** and **readable** compared to the concatenation method. The resulting output in both cases will be the same:

Hello, my name is John and I am 25 years old.

10

(FAT) ARROW FUNCTION

A fat arrow function, also known as an arrow function, is a concise way to write function expressions in JavaScript. It was introduced in ECMAScript 6 (ES6) and provides a shorter syntax compared to traditional function expressions. The syntax uses the => (fat arrow) to define functions.

```
const greetUsers = function (userName) {
  return `Hello ${userName}.`;
};
const greetUsersArrow = (userName) => `Hello ${userName}.`;
```

10

KEYWORD “THIS”:

In JavaScript, the this keyword is a special variable that is implicitly defined in every function. It refers to the object on which the function is invoked, and its value is determined by how the function is called. The value of this can be different depending on the context in which the function is executed.

Global Context:
In a function that is not a method of an object, and if the function is invoked in the global context, this refers to the global object, which is window in a browser environment.

```
function globalContextFunction() {
    console.log(this); // refers to the global object (e.g., window in a browser)
}
globalContextFunction();
```

Method Invocation:
When a function is a method of an object, this refers to the object on which the method is called.

```
const myObject = {
    property: "Hello",
    method: function() {
        console.log(this.property); // refers to myObject
    }
};
myObject.method();
```

KEYWORD “THIS”:

Arrow Functions:
Arrow functions do not have their own this context. Instead, they inherit this from the enclosing lexical scope.

```
const arrowFunction = () => {
    console.log(this); // refers to the value of this in the enclosing scope
};
arrowFunction();
```

10

10

HOISTING:

Hoisting in JavaScript means that **variable** and **function declarations** are **moved to the top** of their containing scope during the code execution.

This allows you to use them before they are actually written in the code. However, only the declarations are moved, not the assignments.

For example, with variables:

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

And with functions:

```
sayHello(); // "Hello, world!"
function sayHello() {
  console.log("Hello, world!");
}
```

Just keep in mind that while hoisting can be convenient, it's a good practice to **declare variables at the beginning of their scope** and **define functions before using them** to make the code more readable.