What is Javascript

JavaScript is a scripting language that enables you to create dynamically updating content, control multimedia, animate images, and pretty much everything else. (Okay, not everything, but it is amazing what you can achieve with a few lines of JavaScript code.)

In HTML, JavaScript code is inserted between `<script>` and `</script>` tags.

Scripts can also be placed in external files:

```
<script src="yourScript.js"></script>
```

Note: Placing scripts at the bottom of the <body> element improves the display speed, because script interpretation slows down the display.

External JavaScript Advantages

Placing scripts in external files has some advantages:

- It separates HTML and code
- It makes HTML and JavaScript easier to read and maintain
- Cached JavaScript files can speed up page loads

Introduction to Variables in JavaScript

Welcome to the world of JavaScript, where variables are an essential part of programming. Variables can be thought of as containers that hold values which can be used and manipulated throughout your code.

In JavaScript, variables are used to store data of different types, such as numbers, strings, and objects. They allow us to write dynamic and interactive programs that respond to user input and other events.

Declaring Variables in JavaScript

In JavaScript, variables are declared using the var, let, or const keywords.

The var keyword is used to declare a variable with global or function scope. The let and const keywords were introduced in ES6 and are used to declare variables with block scope. Block scope means that the variable is only accessible within the block of code where it was declared.

Assigning Values to Variables in JavaScript

In JavaScript, variables can be assigned different types of values such as numbers, strings, booleans, and objects.

To assign a value to a variable, we use the assignment operator (=). For example, if we want to assign the value 10 to a variable called 'x', we would write: x = 10.

Using Variables in JavaScript

Variables are a fundamental concept in programming that allow us to store and manipulate data. In JavaScript, variables can be declared using the var, let, or const keywords.

Once a variable is declared, we can assign values to it using the assignment operator (=). For example, we can declare a variable called 'age' and assign it a value of 25 like this: 'var age = 25;'. We can then use this variable throughout our program wherever we need to reference the age of the user.

Scope of Variables in JavaScript

In JavaScript, the scope of a variable refers to the part of the code where the variable can be accessed. There are two types of scope in JavaScript: global scope and local scope.

Global scope refers to variables that are accessible from anywhere in the code, while local scope refers to variables that are only accessible within a specific function or block of code. When a variable is declared outside of any function or block of code, it has global scope. When a variable is declared inside a function or block of code, it has local scope.

Data types

JavaScript has several primitive data types, such as numbers, strings, booleans, undefined, null, symbol and bigint. In addition, there are non-primitive data types like objects, arrays, and functions. Each of these data types has its own unique properties and methods that can be used to manipulate them. By the end of this presentation, you'll have a solid understanding of all the major data types in JavaScript.

Primitive Data Types

In JavaScript, primitive data types are the basic building blocks of any program. They are simple, immutable values that do not have any properties or methods associated with them.

The 7 primitive data types in JavaScript are: numbers, strings, booleans, undefined, null, symbol and bigint . Booleans can only have two values: true or false. Null represents a deliberate non-value, while undefined represents an unintentional non-value. Numbers represent numeric values, including integers and floating-point numbers. Strings represent textual data and can be enclosed in single or double quotes.

Numbers

The number data type in JavaScript is used to represent both integer and floating-point numbers. It has several properties and methods that can be used to manipulate numeric values. One important property of the number data type is NaN, which stands for 'Not a Number'. This value is returned when a mathematical operation cannot be performed, such as dividing by zero or taking the square root of a negative number.

Some useful methods of the number data type include toFixed(), which rounds a number to a specified number of decimal places, and toString(), which converts a number to a string. Additionally, the Math object in JavaScript provides many useful functions for working with numbers, such as Math.floor() and Math.random().

Strings

In JavaScript, a string is a sequence of characters that can be used to represent text. Strings are enclosed in single or double quotation marks and can contain any combination of letters, numbers, symbols, and spaces.

One important property of strings in JavaScript is their length, which can be accessed using the .length method. Additionally, there are many built-in methods that can be used to manipulate and work with strings, such as .toUpperCase(), .toLowerCase(), and .replace(). These methods can be incredibly useful when working with user input or manipulating data.

Booleans

Booleans are a fundamental data type in JavaScript, representing a binary value of either true or false. They are commonly used in conditional statements, which determine the flow of program execution based on whether a condition is met or not. For example, if a user is logged in, display their profile information; otherwise, prompt them to log in.

It's important to note that while booleans can be explicitly set to true or false, they can also be implicitly coerced from other data types. For example, an empty string or the number 0 will be coerced to false, while any non-empty string or non-zero number will be coerced to true. This can lead to unexpected behavior if not accounted for in your code.

Undefined and Null

In JavaScript, undefined and null are two distinct data types that represent absence of a value. Undefined is used when a variable has been declared but has not been assigned a value, while null is used to explicitly indicate the absence of a value.

It's important to note that undefined and null are not interchangeable. While both represent absence of a value, they have different meanings and use cases. For example, if a function returns null, it means that the function has completed successfully but there is no value to return.

Non-Primitive(Reference) Data Types

Non-primitive data types in JavaScript are objects, arrays, and functions. Unlike primitive data types, which are basic building blocks of a program, non-primitive data types are more complex and can be manipulated in various ways.

Objects, for example, are collections of properties and methods that can be used to represent real-world entities such as people, cars, or buildings. Arrays, on the other hand, are collections of values that can be accessed using index numbers. Functions are reusable blocks of code that can be called multiple times with different arguments.

Objects

Objects are a fundamental data type in JavaScript that allow you to store and manipulate collections of data. They are made up of key-value pairs, where each key is a string and each value can be any data type, including other objects.

One of the most powerful features of objects in JavaScript is their ability to have methods, which are functions that are associated with an object. These methods can be used to perform actions on the data stored in the object, making it a versatile and flexible data type.

Arrays

Arrays are an essential part of programming in JavaScript. They allow us to store and manipulate collections of data, such as lists of numbers or strings. In JavaScript, arrays are a type of object that have a special length property and a set of methods that make working with them much easier.

One of the most powerful features of arrays is their ability to be indexed. This means that each item in the array can be accessed by its position, or index, within the array. For example, if we had an array of numbers called myArray, we could access the first item in the array by using myArray[0]. This makes it easy to loop through an array and perform operations on each item.

Functions

Functions are an essential part of JavaScript programming. In fact, functions in JavaScript are considered as first-class citizens, meaning they can be treated like any other data type.

Functions can be defined using the function keyword, followed by a name (optional), and a set of parentheses that may contain parameters. The body of the function is enclosed in curly braces and contains the code that will be executed when the function is called. Functions can also return values using the return keyword.

Operators

avaScript operators are symbols that are used to perform operations on operands. For example:

var sum=10+20;

Here, + is the arithmetic operator and = is the assignment operator.

There are following types of operators in JavaScript.

1. Arithmetic Operators
2. Comparison (Relational) Operators
3. Bitwise Operators
4. Logical Operators
5. Assignment Operators
6. Special Operators

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on the operands. The following operators are known as JavaScript arithmetic operators.

`+` Addition,

`-` Subtraction

`*` Multiplication

`/` Division

`%` Modulus (Remainder)

`++` Increment

`--` Decrement

JavaScript Comparison Operators

he JavaScript comparison operator compares the two operands. The comparison operators are as follows:

Operator	Description	Example
==	Is equal to	10==20 = false
===	Identical (equal and of same type)	10==20 = false
!=	Not equal to	10!=20 = true
!==	Not Identical	20!==20 = false
>	Greater than	20>10 = true
>=	Greater than or equal to	20>=10 = true
<	Less than	20<10 = false
<=	Less than or equal to	20<=10 = false

JavaScript Bitwise Operators

he JavaScript comparison operator compares the two operands. The comparison operators are as follows:

Operator	Description	Example
&	Bitwise AND	(10==20 & 20==33) = false
|	Bitwise OR	(10==20 | 20==33) = false
^	Bitwise XOR	(10==20 ^ 20==33) = false
~	Bitwise NOT	(~10) = -10
<<	Bitwise Left Shift	(10<<2) = 40
>>	Bitwise Right Shift	(10>>2) = 2
>>>	Bitwise Right Shift with Zero	(10>>>2) = 2

JavaScript Bitwise Operators

The bitwise operators perform bitwise operations on operands.

Operator	Description	Example
&	Bitwise AND	(10==20 & 20==33) = false
|	Bitwise OR	(10==20 | 20==33) = false
^	Bitwise XOR	(10==20 ^ 20==33) = false
~	Bitwise NOT	(~10) = -10
<<	Bitwise Left Shift	(10<<2) = 40
>>	Bitwise Right Shift	(10>>2) = 2
>>>	Bitwise Right Shift with Zero	(10>>>2) = 2

Logical Operators

The following operators are known as JavaScript logical operators.

Operator	Description	Example
&&	Logical AND	(10==20 && 20==33) = false
||	Logical OR	(10==20 || 20==33) = false
!	Logical Not	!(10==20) = true

Assignment Operators

Operator	Description	Example
=	Assign	10+10 = 20
+=	Add and assign	var a=10; a+=20; Now a = 30
-=	Subtract and assign	var a=20; a-=10; Now a = 10
*=	Multiply and assign	var a=10; a*=20; Now a = 200
/=	Divide and assign	var a=10; a/=2; Now a = 5
%=	Modulus and assign	var a=10; a%=2; Now a = 0

Special Operators

Operator	Description
(?:)	Conditional Operator returns value based on the condition. It is like if-else.
,	Comma Operator allows multiple expressions to be evaluated as single statement.
delete	Delete Operator deletes a property from the object.
in	In Operator checks if object has the given property
instanceof	checks if the object is an instance of given type
new	creates an instance (object)
typeof	checks the type of object.
void	it discards the expression's return value.
yield	checks what is returned in a generator by the generator's iterator.