

|  |
| --- |
| JAVA SCRIPT **HEADING**   1. Introduction 2. Variables 3. Data types 4. Interaction method 5. Basic operator 6. Arthematic operator 7. Comparision operator 8. Assignment operator 9. Conditional branching 10. Logical operator 11. Loops(Loops denotes to executing set of statements in repeat until some condition meets)  [Skill Safari] |

DATE : 27/03/2023

**Java script made upon it’s working over the statement.**

Introduction :

JavaScript was initially created to “make web pages alive”.

It is using for add functionality in an web page also it is a interpreted programming language , makes a web sides more interactive and dynamic.

Java script is a client-side language,which means it is executed on the user's computer rather than on the server. This allows for fast and responsive user experiences, as the code can be executed immediately without waiting for the server to respond. JavaScript can also be used on the server-side using platforms like Node.js, which allows developers to use the same language on both the front-end and back-end of a web application.

One of the unique features of JavaScript is its event-driven programming model, which allows code to be triggered by user interactions such as clicking a button, hovering over an element, or scrolling a page. This makes it an ideal language for creating interactive and dynamic web content.

JavaScript is a powerful and versatile programming language that is essential for modern web development.

INTERPRETED;

JavaScript is an interpreted programming language, which means that the code is executed directly without the need for compilation. When a JavaScript program is run, the code is parsed and executed by the browser or runtime environment, line by line, as it is encountered.

This is different from compiled programming languages, such as C++ or Java, which require a compilation step before the code can be executed. In a compiled language, the source code is first compiled into machine code, which is then executed by the computer.

One advantage of using an interpreted language like JavaScript is that it is easy to learn and use, ~~as there is no need to write complex build scripts or set up a compilation environment~~. This makes it particularly well-suited for web development, where fast iteration cycles and a rapid development pace are often necessary.

Another advantage of interpreted languages is that they are typically more flexible and dynamic than compiled languages, as they allow code to be executed on the fly, without the need for a pre-defined set of data structures or functions. This makes it easier to write code that can adapt to changing requirements or user input.

Negative

However, one potential disadvantage of using an interpreted language like JavaScript is that it can be slower to execute than a compiled language. This is because the interpreter has to read and interpret the code each time it is run, rather than executing pre-compiled machine code. However, modern JavaScript engines have become increasingly optimized, and JavaScript performance has improved significantly in recent years.

Variables :

A **value** is a piece of data that can be assigned to a variable or used in an expression

An **argument**, on the other hand, is a value that is passed to a function when it is called. In JavaScript, a function can take zero or more arguments, which can be of any data type.

In JavaScript, you can print the value of a variable to the console using the console.log() method. This method takes one or more arguments, which can be variables, strings, or other data types, and prints them to the console.

Two types ; 1. let

1. Const

Syntax : variable variable name = value

**Exiting value changing method;**

**Varname = value**

Function Scoping:

When a variable is declared inside a function using the var keyword, it is function-scoped. This means that the variable is accessible only within the function in which it is declared, as well as any nested functions. If the variable is declared outside of any function, it becomes a global variable, which can be accessed from anywhere in the code.

**Block Scoping:**

**When a variable is declared inside a block using the let or const keyword, it is block-scoped. This means that the variable is accessible only within the block in which it is declared, as well as any nested blocks.**

JavaScript variables are used to store data values that can be used and manipulated by a program. Variables in JavaScript are declared using the var, let, or const keyword.

var is the older way of declaring variables in JavaScript, while let and const were introduced in the ECMAScript 6 (ES6) standard. let and const have some advantages over var, such as block scoping and immutability (in the case of const), which can make code easier to reason about and less prone to errors.

JavaScript also supports variable hoisting, which means that variables declared using var are moved to the top of their scope at runtime. This can sometimes lead to unexpected behavior if a variable is used before it is declared, so it is generally recommended to use let or const instead of var to avoid this issue.

**Variable Syntax** ; variable variable name = value

( values are must be data types primitive or non-primitive)

**Data types** ;

In JavaScript, data types are used to represent different kinds of values that can be used and manipulated by a program. There are six primitive data types in JavaScript:

1. Number: represents numeric values, such as 10 or 3.14.

In JavaScript, the number data type is used to represent numeric values, including integers and floating-point numbers (numbers with decimal places).

The number data type in JavaScript uses the 64-bit floating-point format, also known as the "double-precision" format. This means that a number value can represent numbers with up to 15 digits of precision, including fractional numbers. However, due to the limitations of floating-point arithmetic, some operations on number values can lead to rounding errors or imprecise results.

1. String: represents textual data, such as "Hello, world!".
2. Boolean: represents true or false values.

4.Null: represents a value that is intentionally empty or

non-existent.

5.Undefined: represents a value that has not been assigned a value yet.

6.Symbol: represents a unique identifier.

In addition to these primitive data types, JavaScript also has two complex data types:

1. Object: represents a collection of properties and values, such as {name: "John", age: 30}.
2. Array: represents an ordered collection of values, such as [1, 2, 3].
3. Function:

JavaScript is a dynamically typed language, which means that the data type of a variable is determined dynamically at runtime, based on the type of value that is assigned to it. This allows for greater flexibility and ease of use, as variables can change their data type during the execution of a program.

In addition, *JavaScript also has the ability to coerce values from one data type to another.* *For example, a string can be converted to a number* using the parseInt() function, and a number can be converted to a string using the to String() method.

Understanding data types is important in JavaScript programming, as it affects how data is stored, processed, and displayed by a program.

**Primitive data type ;**

Primitive values are stored directly in the memory location assigned to the variable, and they are immutable, **which means that their values cannot be changed once they are assigned.**

**Non-primitive (or reference) data type:**

Non-primitive values are stored in memory locations called references, and they are mutable, **which means that their values can be changed after they are created.**

It's important to note that primitive data types are passed by value, while non-primitive data types are passed by reference. This means that when a primitive data type is passed as an argument to a function, a copy of its value is passed, and any changes made inside the function do not affect the original value.

On the other hand, when a non-primitive data type is passed as an argument, a reference to its memory location is passed, and any changes made inside the function will affect the original value.

**Number :**

1. Whole num
2. Desimal num
3. Infinity
4. -Infinity

**Type of :**

**It** is using to identify value type

Let varname= value

Console.log(type of variable name);

**String:**

1. Double quotes = “value”
2. Single quotes = ‘value’
3. Back ticks = `value $

(We can use string+value over using by $ we can use multiple string name and value)

Ex ; console.log (First Name ${“value”})

**NUll :**

A variable hasn’t a value

Ex ; let name = null;

Console.log (name);

So it’s output is as null

**Undefinied** :

Undifined means a values aren’t define variable

Ex ; let age;

Console.log (age)

It’s output also **undifined,** because I ain’t assign a value to variable .

**Boolean :**

**true or false**

**yes or no**

**let isAdmin = true;**

**console.log(isAdmin);**

**console.log(typeof isAdmin);**

**Interaction methods :**

Prompt :

Syntax name(parameters , parameters) method function

**Type conversation :**

String to **(number) ex ; let age = 20**

Console.log (`”age”`, number (20)); console.log( Number(20));

**Number to string** ; varname = use double quotation to number

**Boolean conversation :**

Falsy Values: A value is considered falsy if it converts to false when coerced to a boolean. The following values are falsy in JavaScript:

1. false
2. 0
3. -0
4. null
5. undefined
6. NaN
7. ""

Truthy Values: A value is considered truthy if it converts to true when coerced to a boolean. Any value that is not falsy is considered truthy. Some common truthy values include:

1. true
2. 1
3. "hello" (non-empty string)
4. [] (empty array)
5. {} (empty object)
6. function() {} (empty function)

Ex; Boolean conversion

Boolean(value) => boolean

true, false

1, 0

let varname = 0;

console.log(Boolean(varname));

Expression :

A= b - This unary value call expression

**Basic operaters** :

Arthematic operators:

1. Addition +

console.log(10 + 20);

1. Subraction -

console.log(10 - 20);

1. Multiplication \*

console.log(10 \* 20);

1. Division /

console.log(10 / 20); // quotient

1. Remainder %

console.log(17 % 2); // remainder

1. Exponentiation \*\*

console.log(2 \*\* 3); // 2 \* 2 \* 2 => 8

console.log(2 \*\* 4); // 2 \* 2 \* 2 \* 2 => 16

**Assignment operators** :

1. Equal to = assign a value to a variable
2. += The plus equals operator adds a value to a variable

{ let x = 10;

x += 5; equivalent to x = x + 5;

console.log(x); // output: 15}

1. Increment (++) or (--)
2. Pre - increment ‘
3. Post - increment

**Comparison operators** => Boolean => true or false

1. Greaterthan => ">"

console.log(10 > 20); // false

1. Lessthan => "<"
2. Equalsto => "=="
3. Greaterthan or equals to => ">="
4. Lessthan or equalsto => "<="
5. Not equalsto => "!="
6. Strict equality operator => "==="

The strict equal to operator checks if two values are equal and of the same type, and returns true if they are.

1. Strict Not equals<!==> The strict not equal to operator checks if two values are not equal or of different types, and returns true if they are not equal.

**Conditional branching :**

**Syntax:**

if (condition) {

code to be executed if the condition is true

}

The if statement can also be extended to include an else statement, which is executed when the condition in the if statement is false. The syntax of the **if-else** statement is as follows:

if (condition) {

code to be executed if the condition is true

} else {

code to be executed if the condition is false

}

The if statement can also be extended to include **else if** statements, which allow for multiple conditions to be tested. Here is an example of an if-else if-else statement:

let num = 0;

if (num > 0) {

console.log("The number is positive");

} else if (num < 0) {

console.log("The number is negative");

} else {

console.log("The number is zero");

}

**Logical operator:**

1. | | (OR): Returns true if at least one operand is true, otherwise returns false.
2. && (AND): Returns true if both operands are true, otherwise returns false.
3. ! (NOT): Returns the opposite boolean value of its operand.
4. Nullish coalescing ??: it is evaluate left to right , and a ?? b a is not nullish because a is not null or idendyfied.

**Loops: 1.**while , 2. do……while , 3. for(;;)

1. Initialization => Starting of loop

2. Condition => Ending of the loop

3. Increment/Decrement =>Pattern

**Loops denotes to executing set of statements in repeat until some condition meets**

Syntax:

Loop initialization;

while (condition) { ==>closure

loop body

increment/decrement

}