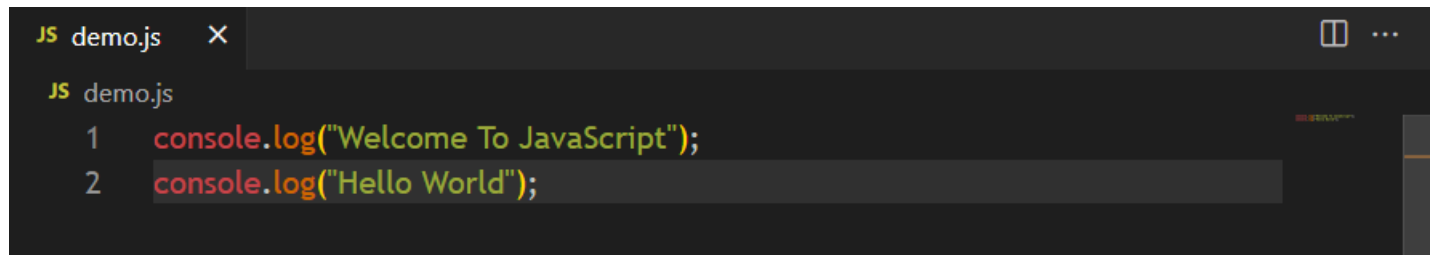
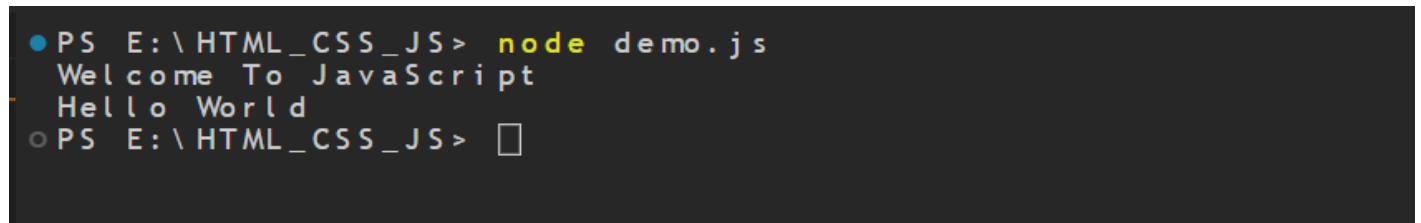


First Program:

Hello world program. in this program you need to print hello world in console. For print the hello world in JavaScript just you need to write “console.log(“hello world”);” For execution in any terminal or command prompt(cmd) the syntax is “node filename.js”.



```
JS demo.js X
JS demo.js
1 console.log("Welcome To JavaScript");
2 console.log("Hello World");
```

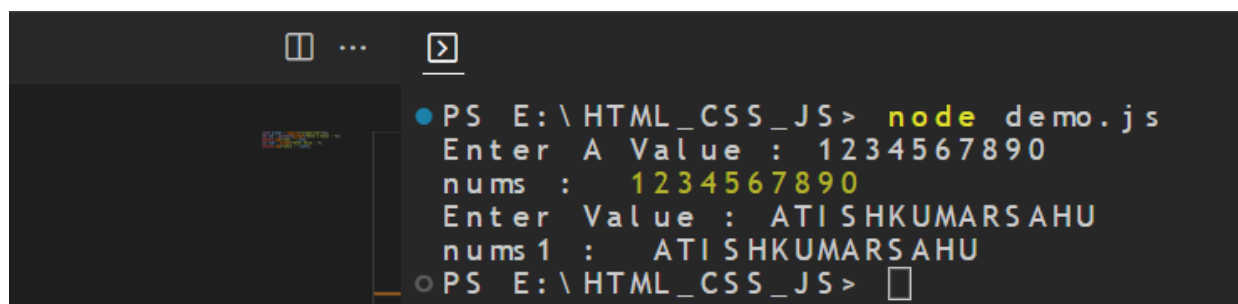


```
● PS E:\HTML_CSS_JS> node demo.js
Welcome To JavaScript
Hello World
○ PS E:\HTML_CSS_JS> □
```

User Input In JavaScript:



```
JS demo.js X
JS demo.js > ...
1 const prompt = require("prompt-sync")();
2 let num = parseInt(prompt("Enter A Value : "));
3 console.log("nums : ",num);
4 let num1 = prompt("Enter Value : ");
5 console.log("nums1 : ",num1);
```



```
● PS E:\HTML_CSS_JS> node demo.js
Enter A Value : 1234567890
nums : 1234567890
Enter Value : ATISHKUMARSAHU
nums1 : ATISHKUMARSAHU
○ PS E:\HTML_CSS_JS> □
```

Variables:

A JavaScript variable is simply a name of storage location. There are two types of variables in JavaScript. Variable name must starts with a letter (A to Z or a to z), underscore(_) or dollar (\$) sign. After first letter we can use digits (0 to 9) for example: num1, n2nu. JavaScript variables are case Sensitive.

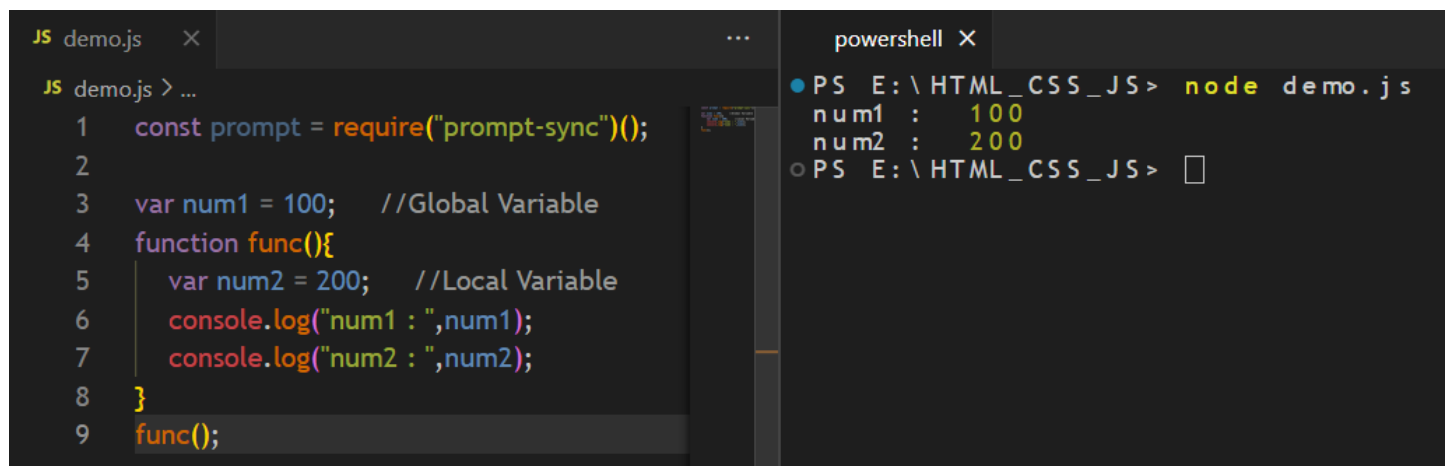
Type Of Variables:

Local Variable:

A JavaScript local variable is declared inside block or function. it is accessible within the function or block only.

Global Variable:

A global variable is accessible from any function. A variable declared outside the function or declared with window object is known as global variable.



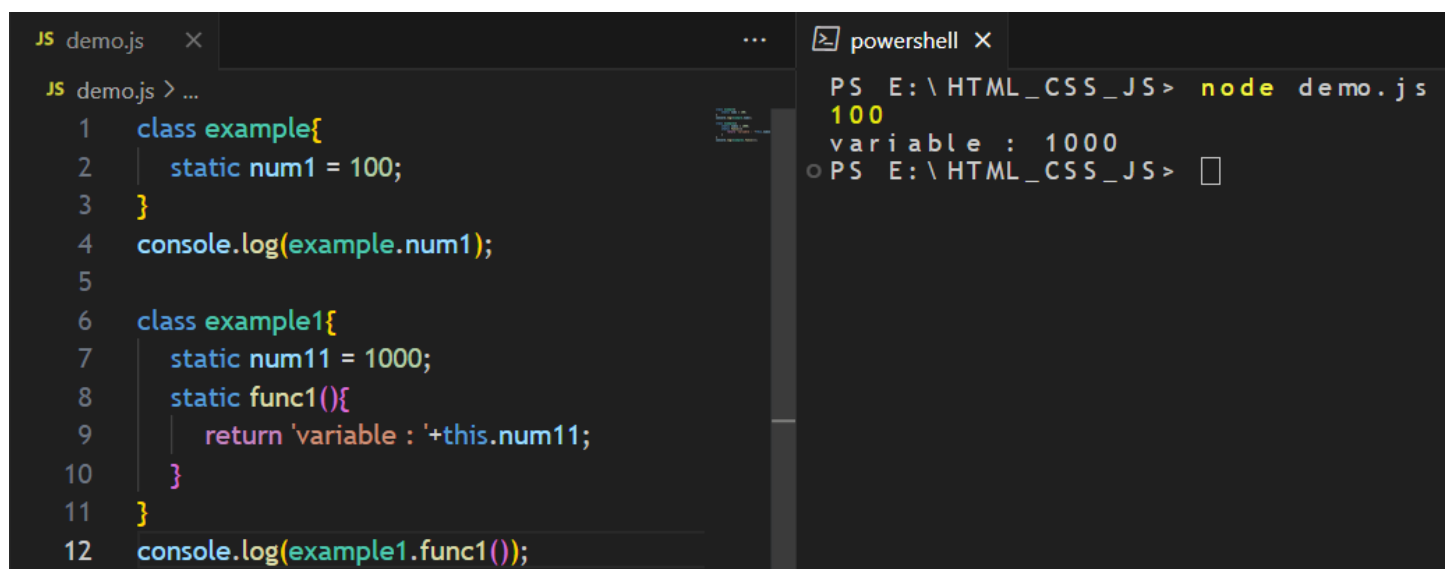
The screenshot shows a code editor with a file named 'demo.js' and a PowerShell terminal window. The JavaScript code defines a global variable 'num1' and a function 'func' that logs 'num1' and a local variable 'num2'. The PowerShell terminal shows the output of running 'node demo.js', which is 'num1 : 100' and 'num2 : 200'.

```
JS demo.js > ...
1 const prompt = require("prompt-sync")();
2
3 var num1 = 100; //Global Variable
4 function func(){
5     var num2 = 200; //Local Variable
6     console.log("num1 : ",num1);
7     console.log("num2 : ",num2);
8 }
9 func();
```

```
powershell X
PS E:\HTML_CSS_JS> node demo.js
num1 : 100
num2 : 200
PS E:\HTML_CSS_JS>
```

Static Variable:

In the context of JavaScript, a static variable is often emulated using a closure – a function that "closes over" its lexical scope, allowing variables to persist even after the function has finished executing.



The screenshot shows a code editor with a file named 'demo.js' and a PowerShell terminal window. The JavaScript code defines a class 'example' with a static variable 'num1' and a class 'example1' with a static variable 'num11' and a static function 'func1'. The PowerShell terminal shows the output of running 'node demo.js', which is '100' and 'variable : 1000'.

```
JS demo.js > ...
1 class example{
2     static num1 = 100;
3 }
4 console.log(example.num1);
5
6 class example1{
7     static num11 = 1000;
8     static func1(){
9         return 'variable : '+this.num11;
10    }
11 }
12 console.log(example1.func1());
```

```
powershell X
PS E:\HTML_CSS_JS> node demo.js
100
variable : 1000
PS E:\HTML_CSS_JS>
```

Var Keyword:

Let keyword is used to declare a variable as a global variables. user can access the value and the variable globally and changeable also. The var variable can be redeclared and you can reassigned the value of a variable is the variable is a var type. variables are hoisted on top and can be used anywhere. in a program you can redeclared the var type variable anywhere.

Let Keyword:

Let keyword is used for declare the variable in locally. If you used let keyword to declare a variable then the variable can accessible locally and it is changeable as well. You can't redeclare the variable if you use let keyword but you can reassign the variable. Let type variables have block level of scope. Before use of the variable you have to initialize the variable. User can redeclared the variable inside the block.

Const Keyword:

Const keyword is used for to declare variable locally. If you use const to declare variable then the variable will one be accessible within the block. Variable declared using const values can't be reassigned. So user should assign the value while declaring the variable.

Difference Between Var, Let & Const:

1. var function-scoped. Variable declared with var are hoisted to the top of their scope and accessible throughout the entire function, even before the point where they are declared.

Let and const block-scoped. They are confined to the block (enclosed by curly braces {}) in which they are declared. They are not hoisted to the top of the block and are only accessible after the point of declaration.

2. var Hoisted to the top of its scope. This means that you can use a variable declared with var before its actual declaration in the code.

Let & Const Also hoisted, but unlike var, if you try to use them before the declaration, you'll get a ReferenceError.

3. var and let: Can be reassigned. You can change the value of a variable declared with var or let. Const Cannot be reassigned. Once a value is assigned to a variable with const, it cannot be changed.

4. var Can be declared and initialized without a value. If you don't assign an initial value, the variable will have the value undefined.

Let and Const Require an assignment upon declaration. If you try to declare a variable with let or const without assigning a value, you'll get a SyntaxError.

5. var Has no TDZ. Variables declared with var are hoisted and can be accessed anywhere in the function, even before the actual declaration. Let & Const Have a TDZ. If you try to access them before their declaration point, you'll get a ReferenceError.

6. var declarations are added as properties on the global object (window in browsers). This can lead to unintended global variable declarations. Let & Const declarations do not create properties on the global object.

7. var allows variable redeclaration within the same scope, potentially leading to unexpected behavior. let and const do not allow redeclaration within the same scope.

8. var is function-scoped. Variables declared with var are only scoped to the function they are declared in. let and const are block-scoped. They are scoped to the nearest enclosing block, whether it's a function, loop, or conditional statement.

9. The Temporal Dead Zone is the time between entering a scope and the variable being declared, where trying to access the variable results in a ReferenceError. var is not affected by the Temporal Dead Zone. Both let and const are affected by the Temporal Dead Zone.

JavaScript Datatype:

Number:

There are two types of numbers are there integer and float type number. Example: 12, 555, 23.66, etc. BigInt is a new datatype which was introduced in ES20 to store a very large integer Value. Example: let a = BigInt(419441859849849841954);

String:

String is sequence of character which plays an important role in every programming. For single character we use single quotation(") mark for string we use double quotation("") mark and tilt (`) symbol. Example: 'A', "Lipun", `Atish`.

Boolean:

Boolean value is used in conditional statement. In boolean there are two values true and false. True is for correct statement which denoted by 1 and false is for incorrect statement which denoted by 0. Example: let a = true; let b = false;

Null:

Null value is used for if you want to give alternate value for 0 then you can use null value.

Undefined:

During the coding if you don't give any value to a variable then during the execution it will shows undefined.

Arrays:

JavaScript arrays are written with square brackets. Array items are separated by commas. In array you can store string value and number type value. Ex: Let arr = ["aa", "bb"]; let arr = [1, 2, 4, 5, 6];

Objects:

Objects represents instance through which we can access members. JavaScript objects are written with curly braces {}. Object properties are written as name: value pairs, separated by commas.

Regular Expression:

A regular expression is a pattern of characters. The pattern is used for searching and replacing characters in strings. The RegExp Object is a regular expression with added Properties and Methods. Represents regular expression.

Operators In JavaScript:

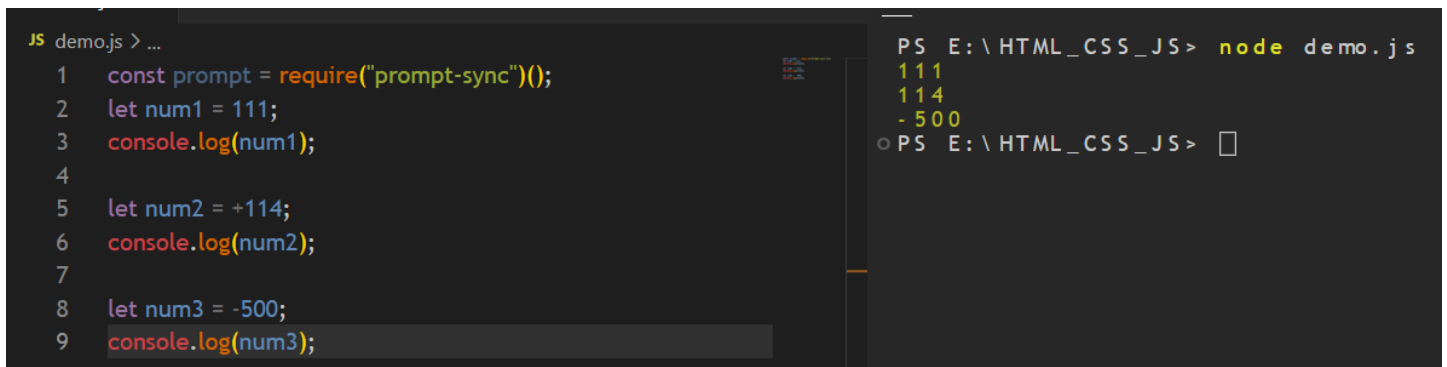
Before you know about operator first you need to know about what is operand. An Operand is the input values which are used Operator to create an resultant output. An Operators is used to perform an operation on Operand of any type.

Unary Operator:

Unary operators in JavaScript are unique operators that consider a single input and carry out all possible operations.

The Unary plus, unary minus, prefix increments, postfix increments, postfix decrements, and prefix decrements are examples of these operators. These operators are either put before or after the operand.

The unary operators are more effective in executing functions than JavaScript; they are more popular. Unary operators are flexible and versatile since they cannot be overridden.



The image shows a code editor on the left and a terminal on the right. The code editor contains the following JavaScript code:

```
JS demo.js > ...
1 const prompt = require("prompt-sync")();
2 let num1 = 111;
3 console.log(num1);
4
5 let num2 = +114;
6 console.log(num2);
7
8 let num3 = -500;
9 console.log(num3);
```

The terminal on the right shows the output of running the code:

```
PS E:\HTML_CSS_JS> node demo.js
111
114
-500
PS E:\HTML_CSS_JS> 
```

Increment Operator:

Prefix: The operator uses to inserts one value before the incremental value by one.

Example: (++value)

Postfix: The operator uses to inserts one value after the incremental value by one.

Example: (value++)

```
JS demo.js X ... >
JS demo.js > ...
1 const prompt = require("prompt-sync")();
2
3 let num1 = 200;
4 console.log(num1);
5 console.log(num1++);
6 console.log(num1);
7
8 let num2 = 400;
9 console.log(num2);
10 console.log(++num2);
11 console.log(num2);

PS E:\HTML_CSS_JS> node demo.js
200
200
201
400
401
401
PS E:\HTML_CSS_JS>
```

Decrement Operator:

The prefix operator subtracts one value from the given input value before "--value". The postfix operator subtracts one value before the incremental value by one "value--".

```
JS demo.js X ... >
JS demo.js > ...
1 const prompt = require("prompt-sync")();
2
3 let num1 = 100;
4 console.log(num1);
5 console.log(num1--);
6 console.log(num1);
7
8 let num2 = 200;
9 console.log(num2);
10 console.log(--num2);
11 console.log(num2);

PS E:\HTML_CSS_JS> node demo.js
100
100
99
200
199
199
PS E:\HTML_CSS_JS>
```

Arithmetic Operator:

In Arithmetic Operators these are the following operators are there Addition(+), Subtraction(-), Multiplication(*), Division(/), Modulo(%).

```
JS demo.js X ... >
JS demo.js > ...
1 const prompt = require("prompt-sync")();
2
3 console.log(100 + 200);
4 console.log(500 - 300);
5 console.log(600 * 400);
6 console.log(800 / 200);
7 console.log(1000 % 30);

PS E:\HTML_CSS_JS> node demo.js
300
200
240000
4
10
PS E:\HTML_CSS_JS>
```

Assignment Operator:

In Assignment Operator these are the following operators are there Assignment(=), Assignment-Addition(+=), Assignment-Subtraction(-=), Assignment-Multiplication(*=), Assignment-Division(/=), Assignment-Modulo(%=).

```
JS demo.js x
JS demo.js > ...
1  const prompt = require("prompt-sync")();
2
3  let num1 = 10;
4  console.log(num1);
5  console.log(num1 += 20);
6  console.log(num1);
7
8  let num2 = 20;
9  console.log(num2);
10 console.log(num2 -= 5);
11 console.log(num2);
13 let num3 = 40;
14 console.log(num3);
15 console.log(num3 *= 50);
16 console.log(num3);
17
18 let num4 = 500;
19 console.log(num4);
20 console.log(num4 /= 20);
21 console.log(num4);
22
23 let num5 = 600;
24 console.log(num5);
25 console.log(num5 %= 13);
26 console.log(num5);
```

Relational Operator:

In Relational Operator these are the following Operators are there Smaller Than(<), Greater Than(>), Smaller Than Equal(<=), Greater Than Equal(>=), Equal-Equal(==), Equal-Equal-Equal(===), Not Equal(!=)

```
JS demo.js x
JS demo.js > ...
1  const prompt = require("prompt-sync")();
2
3  console.log("14 > 55", (14 > 55));
4  console.log("55 >= 44", (55 >= 44));
5  console.log("66 < 22", (66 < 22));
6  console.log("12 <= 22", (12 <= 22));
7  console.log("12 == 12", (12 == 12));
8  console.log("55 != 12", (55 != 12));
9  console.log("12 === 12", ("12" === 12));

PS E:\HTML_CSS_JS> node demo.js
14 > 55 false
55 >= 44 true
66 < 22 false
12 <= 22 true
12 == 12 true
55 != 12 true
12 === 12 false
PS E:\HTML_CSS_JS>
```

Bitwise Operator:

In Bitwise Operator these are the following signs are used AND(&), OR(|), XOR(^), NOT(!). Bitwise Operator both are used for integer numbers and boolean values.

```
JS demo.js x
JS demo.js > ...
1  const prompt = require("prompt-sync")();
2  console.log("10 | 20 : ", (10 | 20));
3  console.log("10 & 20 : ", (10 & 20));
4  console.log("10 ^ 20 : ", (10 ^ 20));
5  console.log("~(10 | 20) : ", ~(10 | 20));
6  console.log("~(10 & 20) : ", ~(10 & 20));
7  console.log("~(10 ^ 20) : ", ~(10 ^ 20));
8  console.log("~10 : ", ~10);
9  console.log("~20 : ", ~20);

PS E:\HTML_CSS_JS> node demo.js
10 | 20 : 30
10 & 20 : 0
10 ^ 20 : 30
~(10 | 20) : -31
~(10 & 20) : -1
~(10 ^ 20) : -31
~10 : -11
~20 : -19
PS E:\HTML_CSS_JS>
```

```
JS demo.js x
JS demo.js > ...
1 const prompt = require("prompt-sync")();
2
3 console.log("true & true :",(true & true));
4 console.log("true & false :",(true & false));
5 console.log("false & true :",(false & true));
6 console.log("false & false :",(false & false));

PS E:\HTML_CSS_JS> node demo.js
true & true : 1
true & false : 0
false & true : 0
false & false : 0
PS E:\HTML_CSS_JS>
```

```
JS demo.js x
JS demo.js > ...
1 const prompt = require("prompt-sync")();
2
3 console.log("true | true :",(true | true));
4 console.log("true | false :",(true | false));
5 console.log("false | true :",(false | true));
6 console.log("false | false :",(false | false));

PS E:\HTML_CSS_JS> node demo.js
true | true : 1
true | false : 1
false | true : 1
false | false : 0
PS E:\HTML_CSS_JS>
```

```
JS demo.js •
JS demo.js > ...
1 const prompt = require("prompt-sync")();
2 console.log("true ^ true :",(true ^ true));
3 console.log("true ^ false :",(true ^ false));
4 console.log("false ^ true :",(false ^ true));
5 console.log("false ^ false :",(false ^ false));
6

PS E:\HTML_CSS_JS> node demo.js
true ^ true : 0
true ^ false : 1
false ^ true : 1
false ^ false : 0
PS E:\HTML_CSS_JS>
```

Conditional Or Logical Operator:

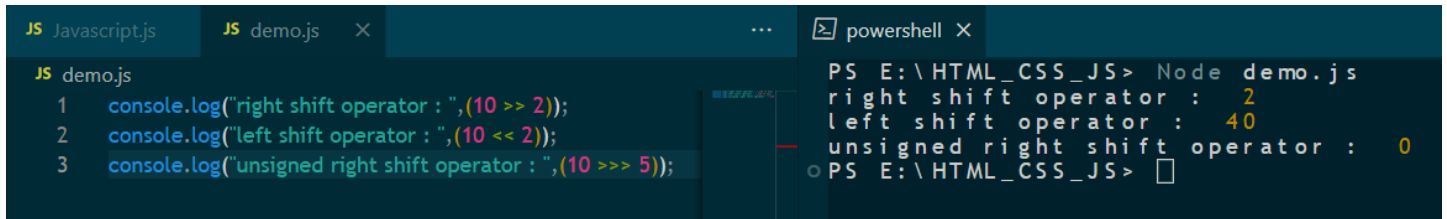
In this operator these are the following operators are used Logical AND(&&), Logical OR(||), Nullish Coalescing Assignment Operator(??).

```
JS Javascript.js JS demo.js x powershell x
JS demo.js > ...
1 console.log("5 < 10 && 6 > 1 -> "+(5 < 10 && 6 > 1));
2 console.log("5 > 10 && 6 > 1 -> "+(5 > 10 && 6 > 1));
3
4 console.log("5 == 5 || 6 == 5 "+(5 == 5 || 6 == 5));
5 console.log("7 == 5 || 6 == 5 "+(7 == 5 || 6 == 5));
6
7 let num = 10;
8 console.log("num : ",num);
9 let res = num ??= 12;
10 console.log("num : ",num);
11 console.log("res : ",res);
12
13 let num1;
14 console.log("num1 : ",num1);
15 let res1 = num1 ??= 100;
16 console.log("num1 : ",num1);
17 console.log("res1 : ",res1);

PS E:\HTML_CSS_JS> Node demo.js
5 < 10 && 6 > 1 -> true
5 > 10 && 6 > 1 -> false
5 == 5 || 6 == 5 true
7 == 5 || 6 == 5 false
num : 10
num : 10
res : 10
num1 : undefined
num1 : 100
res1 : 100
PS E:\HTML_CSS_JS>
```


Shift Operator:

Shift Operator is used for to shift the values by number of defined place value. these are the following signs are used in shift operator. Left-Shift Operator(<<), Right-Shift Operator(>>), Unsigned Right Shift Operator(>>>).

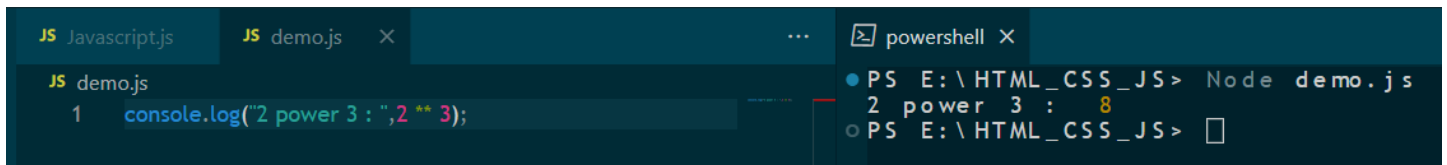


```
JS Javascript.js JS demo.js x ... powershell x
JS demo.js
1 console.log("right shift operator : ",(10 >> 2));
2 console.log("left shift operator : ",(10 << 2));
3 console.log("unsigned right shift operator : ",(10 >>> 5));

PS E:\HTML_CSS_JS> Node demo.js
right shift operator : 2
left shift operator : 40
unsigned right shift operator : 0
PS E:\HTML_CSS_JS>
```

Power Operator:

Power Operator is used for to find the defined power of any value. The required operator is power(**) which is used as a power operator.

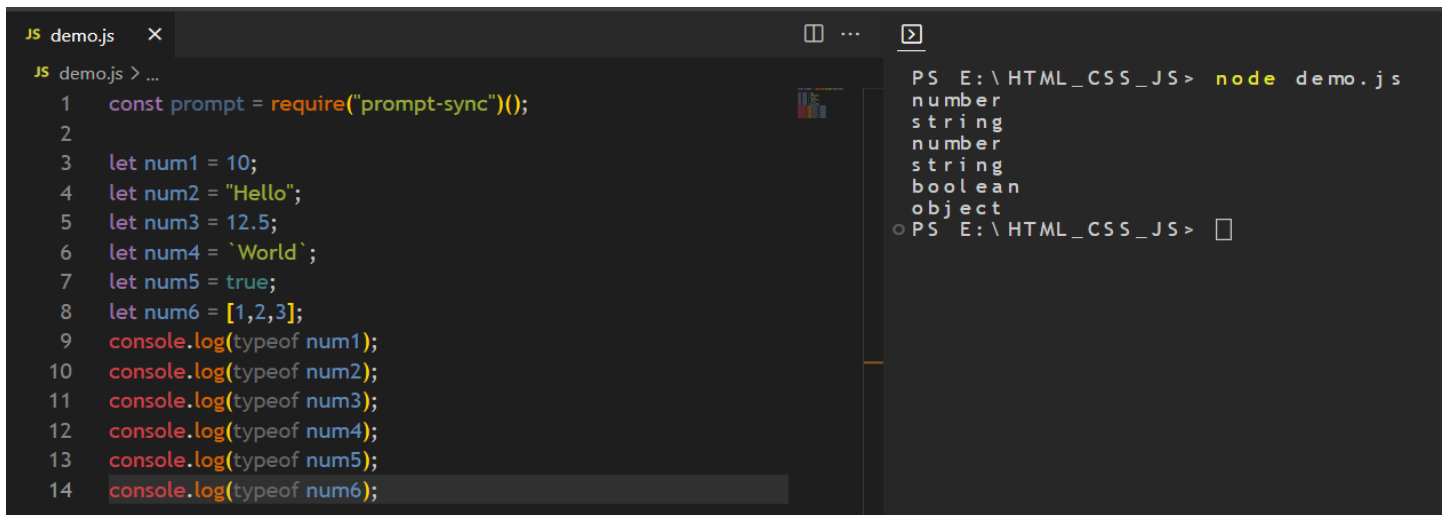


```
JS Javascript.js JS demo.js x ... powershell x
JS demo.js
1 console.log("2 power 3 : ",2 ** 3);

PS E:\HTML_CSS_JS> Node demo.js
2 power 3 : 8
PS E:\HTML_CSS_JS>
```

Typeof Operator:

Typeof operator is used for to know about the type of a particular value. we use "typeof" statement to know about the type of a particular input/value. Checks the type of object.

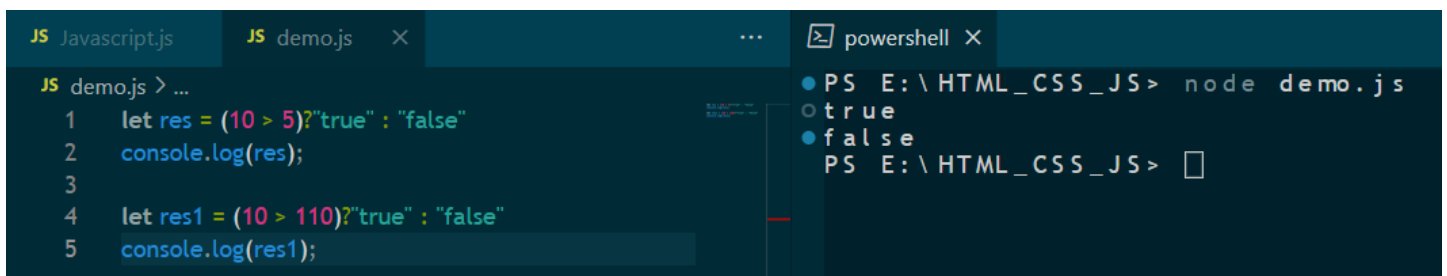


```
JS demo.js x ... powershell x
JS demo.js > ...
1 const prompt = require("prompt-sync")();
2
3 let num1 = 10;
4 let num2 = "Hello";
5 let num3 = 12.5;
6 let num4 = `World`;
7 let num5 = true;
8 let num6 = [1,2,3];
9 console.log(typeof num1);
10 console.log(typeof num2);
11 console.log(typeof num3);
12 console.log(typeof num4);
13 console.log(typeof num5);
14 console.log(typeof num6);

PS E:\HTML_CSS_JS> node demo.js
number
string
number
string
boolean
object
PS E:\HTML_CSS_JS>
```

Ternary Operator:

It is an type of conditional checking operator where you can print output by checking the condition. The writing style of Ternary Operator is [Condition ? Output1 : Output2].

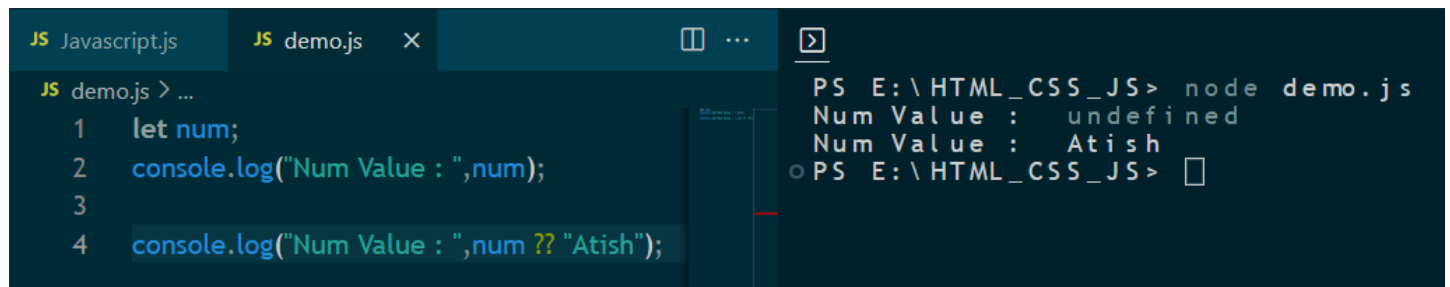


```
JS Javascript.js JS demo.js x ... powershell x
JS demo.js > ...
1 let res = (10 > 5)? "true" : "false"
2 console.log(res);
3
4 let res1 = (10 > 110)? "true" : "false"
5 console.log(res1);

PS E:\HTML_CSS_JS> node demo.js
true
false
PS E:\HTML_CSS_JS>
```

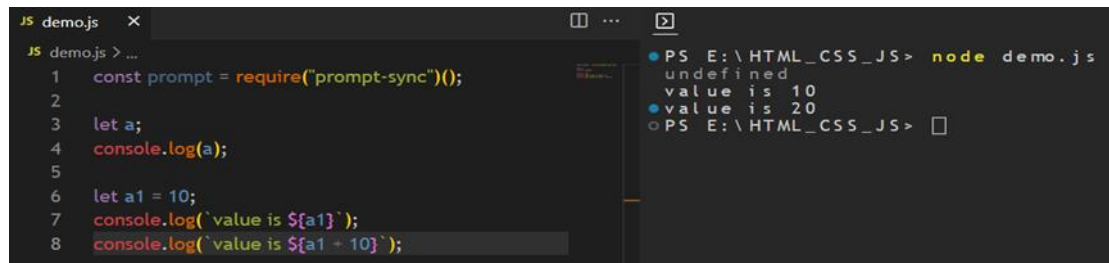
Nullish Coalescing Operator:

This operator returns the right-hand value if the left-hand value is null or undefined. If not null or undefined then it will return left-hand value.



```
JS Javascript.js JS demo.js x ... [ ]
JS demo.js > ...
1 let num;
2 console.log("Num Value : ",num);
3
4 console.log("Num Value : ",num ?? "Atish");

PS E:\HTML_CSS_JS> node demo.js
Num Value : undefined
Num Value : Atish
PS E:\HTML_CSS_JS> [ ]
```



```
JS demo.js x ... [ ]
JS demo.js > ...
1 const prompt = require("prompt-sync")();
2
3 let a;
4 console.log(a);
5
6 let a1 = 10;
7 console.log(`value is ${a1}`);
8 console.log(`value is ${a1 + 10}`);

PS E:\HTML_CSS_JS> node demo.js
undefined
value is 10
value is 20
PS E:\HTML_CSS_JS> [ ]
```

Comma Operator:

This allows multiple expression to be evaluated as single statement. A comma operator is used as a separator for multiple expressions at a place that requires a single expression. When a comma operator is placed in an expression, it executes each expression and returns the rightmost expression.

Delete Operator:

This operator deletes a property from the object. The delete operator deletes both the value of the property and the property itself. After deletion, the property cannot be used before it is added back again. The delete operator is designed to be used on object properties. It has no effect on variables or functions. The delete operator should not be used on predefined JavaScript object properties. It can crash your application.

In Operator:

This allows checks if object has the given property. The in operator is an inbuilt operator in JavaScript which is used to check whether a particular property exists in an object or not. It returns a boolean value true if the specified property is in an object, otherwise, it returns false.

Instanceof:

It checks if the object is an instance of given type. The instanceof operator in JavaScript is used to check the type of an object at run time. It returns a boolean value if true then it indicates that the object is an instance of a particular class and if false then it is not.

New Operator:

Create an instance. New keyword in JavaScript is used to create an instance of an object that has a constructor function. On calling the constructor function with 'new'.

Void:

The void keyword in JavaScript, is used to evaluate an expression which does not return any value. The void operator is an unary operator that accepts the single operand, which may be of any type. The importance of the void keyword come into role when we just need to evaluate an expression instead of returning its value. It means, by using it, we can prevent the browser from displaying the result of the execution of the expression.

Yield:

yield keyword is used to resume or pause a generator function asynchronously. A generator function is just like a normal function but the difference is that whenever the function is returning any value, it does it with the help of 'yield' keyword instead of return it. Yield can't be called from nested functions or from callbacks.

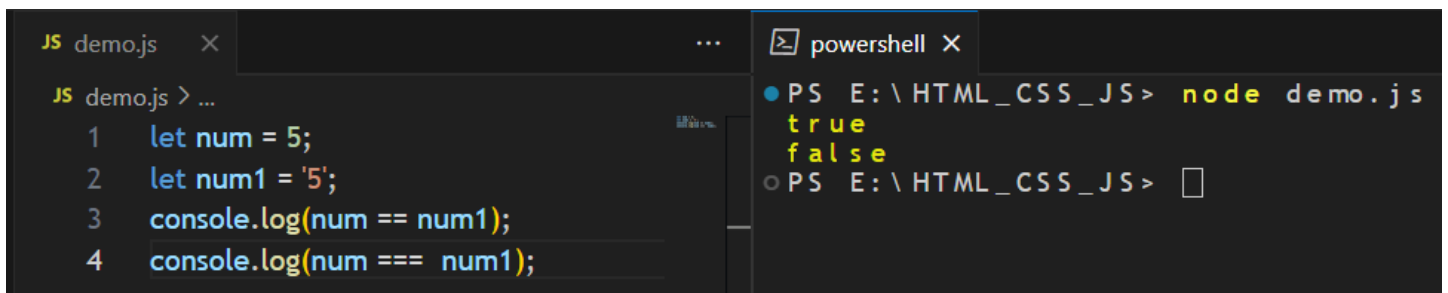
Difference between =, == & === Operator:

Assignment Operator = is used for assignment. It assigns the value on the right-hand side to the variable on the left-hand side.

The == operator performs type coercion if the operands are of different types before making the comparison. This means that it attempts to convert the operands to the same type before making the comparison. The loose equality operator (==) can lead to unexpected results due to automatic type conversions. It's often safer to use strict equality (===) to ensure that the values and types are exactly the same.

The === operator, also known as the strict equality operator, does not perform type coercion. It checks both the value and the type of the operands. Using strict equality (===) is generally recommended because it avoids unexpected type coercion. It provides a more predictable behavior, especially when comparing values of different types.

Assignment vs comparison Remember that = is an assignment operator, used to assign values to variables. == and === are comparison operators used to compare values.



```
JS demo.js x ... powershell x
JS demo.js > ...
1 let num = 5;
2 let num1 = '5';
3 console.log(num == num1);
4 console.log(num === num1);

PS E:\HTML_CSS_JS> node demo.js
true
false
PS E:\HTML_CSS_JS> 
```