

JavaScript

essentials for Node-RED

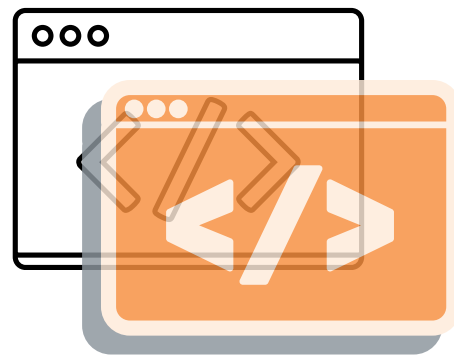




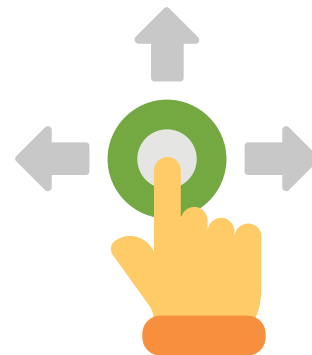
Introduction to JavaScript

JavaScript, often abbreviated as **JS**, is a versatile and widely-used programming language that powers the interactive features of **websites** and **web applications**.

Key features:



Used for both front end and back end



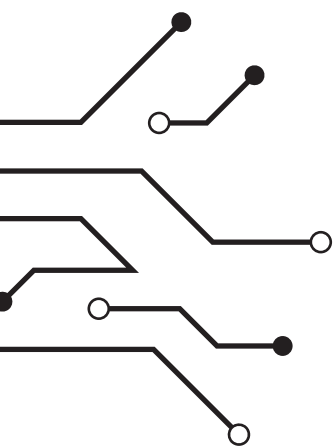
Used to create interactive element on webpage

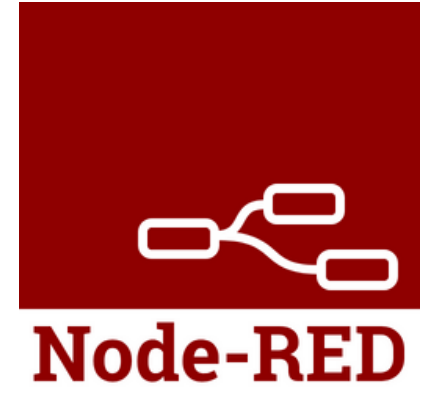


It has vast ecosystem of libraries and framework such as React, Angular and Node.js



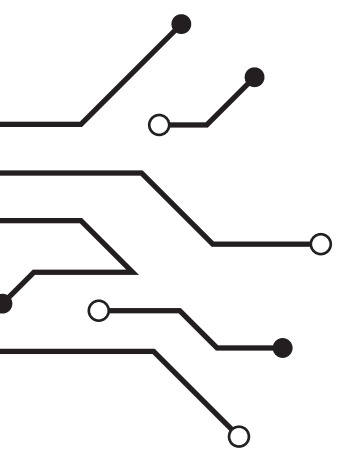
It is prototype based object oriented language allowing developers to create reusable and modular code structure

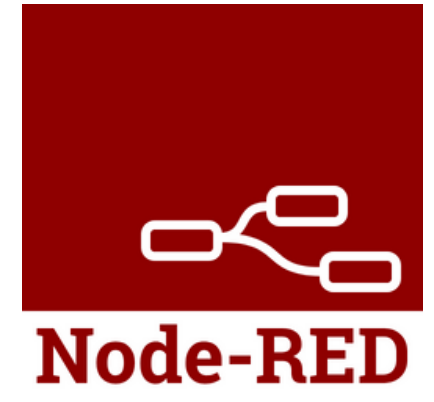




Why to learn JavaScript, If I am learning Node-RED?

- ✓ **Logic and Control:** You can use **JavaScript** functions within function nodes to **manipulate data, apply conditional logic, and create complex workflows.**
- ✓ **Debugging:** Knowledge of JavaScript helps you effectively **debug issues within your Node-RED flows.** Understanding **JavaScript error messages and debugging techniques** is invaluable when troubleshooting your applications.
- ✓ **Community Support:** JavaScript is one of the **most widely used programming languages globally.** Consequently, there's an **extensive online community and resources available for JavaScript,** making it easier to find solutions and guidance when working with Node-RED.

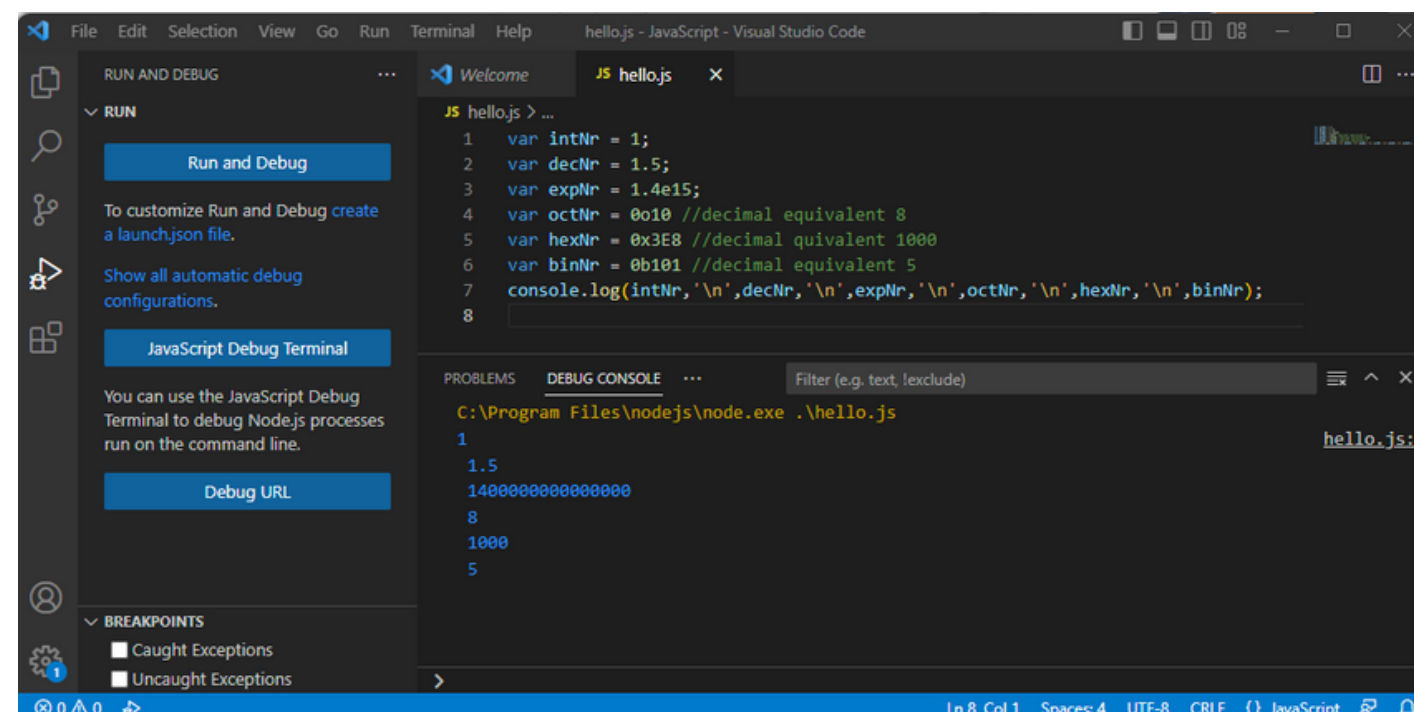




Programming Environment

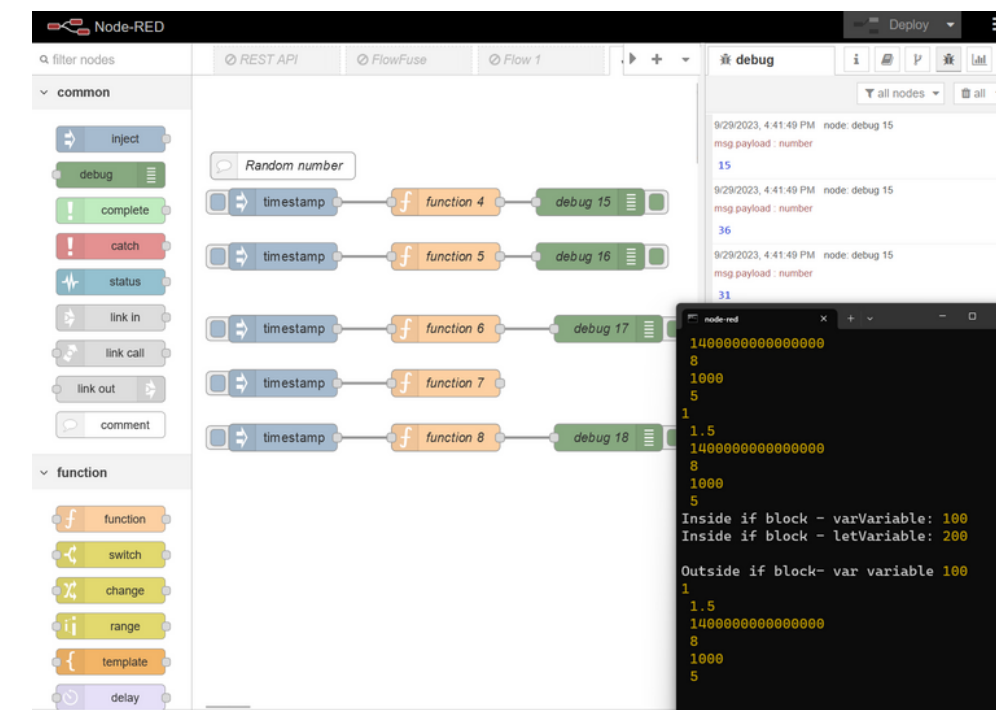
Visual Studio Code (VSCode)

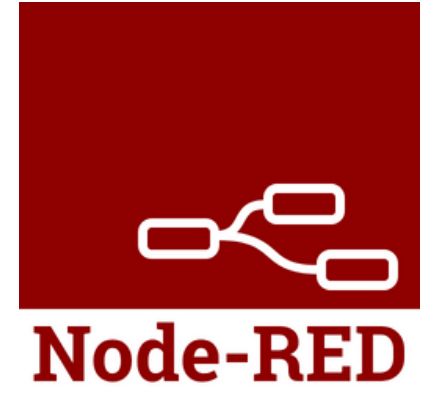
Visual Studio Code is a lightweight, feature-rich code editor developed by Microsoft. It offers powerful extensions, debugging capabilities, and a user-friendly interface,



Node-RED

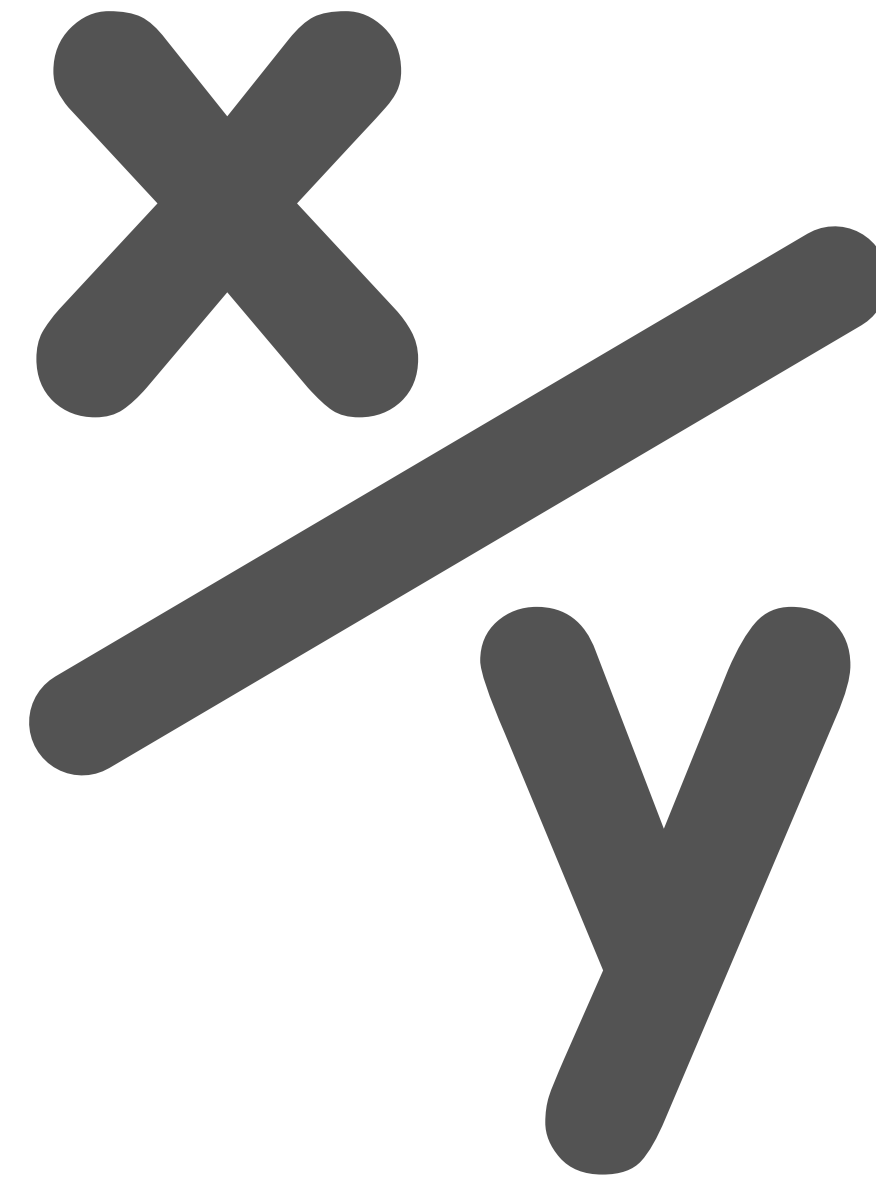
Node-RED is an open-source flow-based development tool for visual programming, ideal for IoT applications and automation tasks.





Section 1

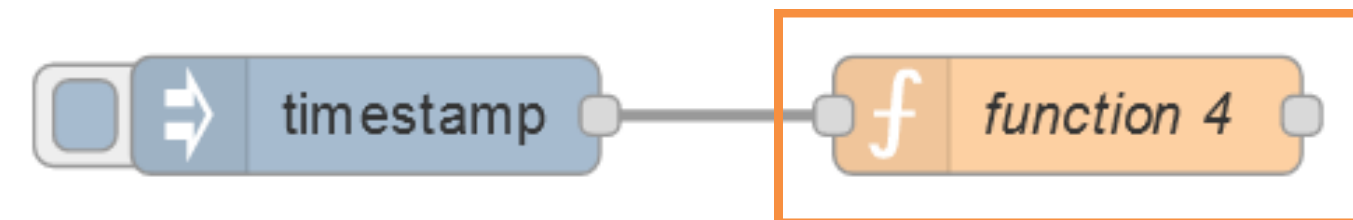
Variables and Operators






Console log

```
1 console.log("Hello Log!");  
2 console.log(4 + 10);  
3 console.info("Hello Info!");  
4 console.error("Hello Error!");
```

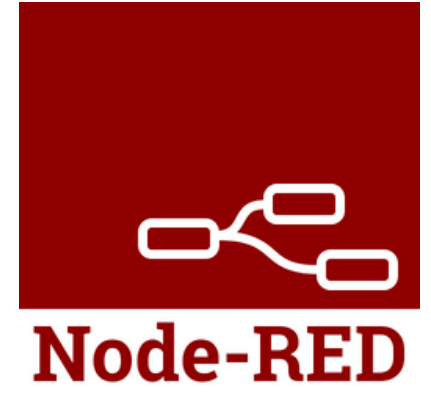


Terminal Log



```
Hello Log!  
Hello Info!  
Hello Error!  
14
```





Indentation and white space

Comparison between code **without space**, **with space**, **with space and indent**

Without space

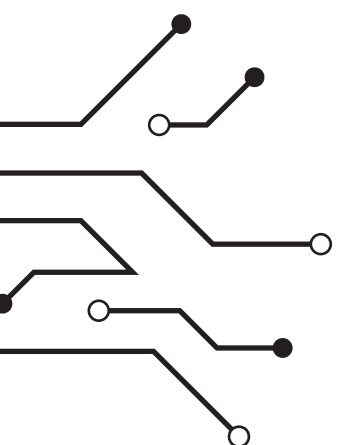
```
1 let scared = true; if (scared) { console.log("Don't worry"); } else { console.log("You're brave") }
```

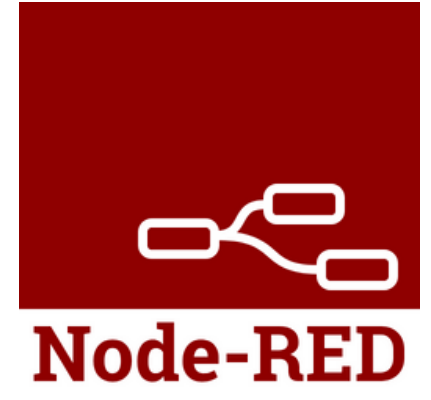
With space

```
1 let scared = true;
2 if (scared)
3 {
4 console.log("Don't worry");}
5 else
6 {
7 console.log("You're brave");
8 }
```

With space and indent

```
1 let scared = true;
2 if (scared)
3     {console.log("Don't worry");}
4 else
5     {console.log("You're brave");}
```





Semi-colon and comments

After every statement you should **insert a semicolon (;)**

```
1 let value = true;
```

For code block like if and else statement **semicolon is not required**

```
1 let value = true;
2 if (value)
3 |   {   }
4 else
5 |   {   }
```

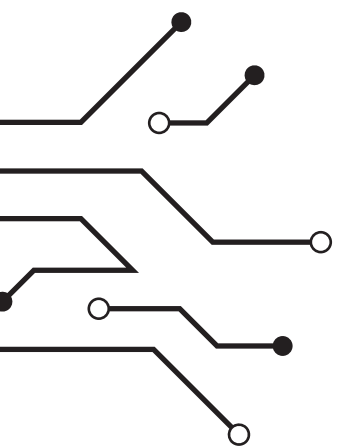
Use comments to make your script understandable

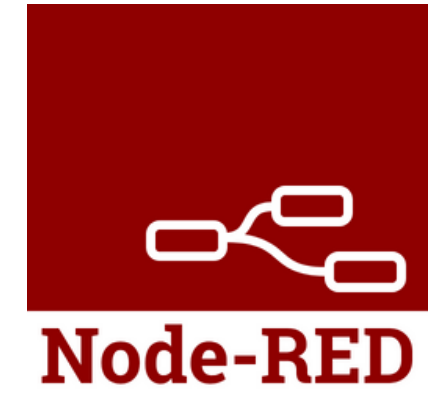
Single line comment

```
1 //Program version 1.0
2 //Edited on: 25/9/2023
3 //Edited by: Rajvir Singh
```

Multi line comment

```
1 /*Program version: 1.0
2 Edited on: 25/9/2023
3 Edited by: Rajvir Singh
4 */
```

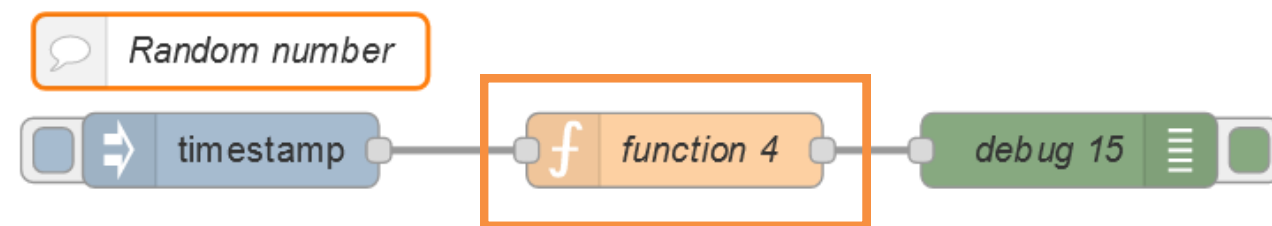




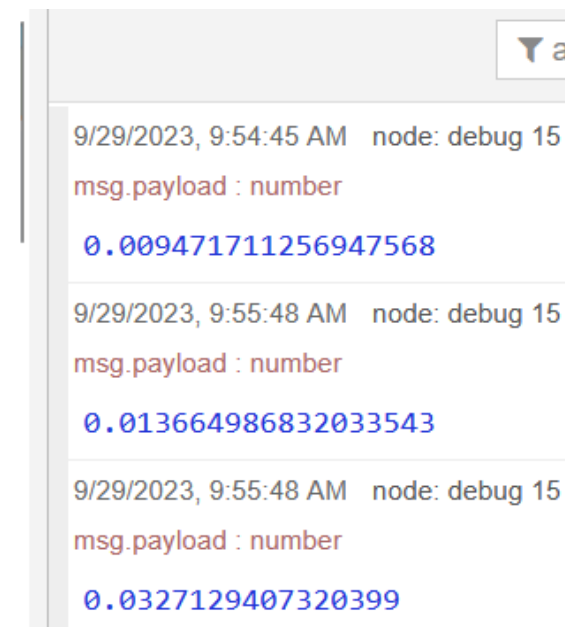
Random numbers

Math.random();

This will give a number between 0 and 1



```
msg.payload = Math.random();  
return msg;
```



```
9/29/2023, 9:54:45 AM node: debug 15  
msg.payload : number  
0.009471711256947568  
  
9/29/2023, 9:55:48 AM node: debug 15  
msg.payload : number  
0.013664986832033543  
  
9/29/2023, 9:55:48 AM node: debug 15  
msg.payload : number  
0.0327129407320399
```

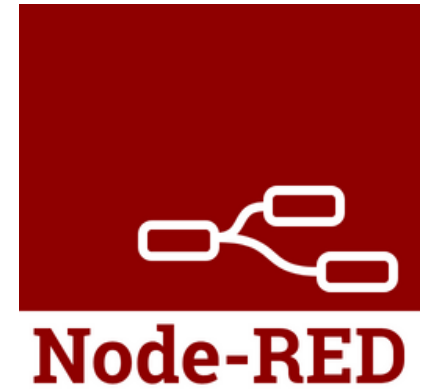
Multiply with 100 to get random number between 0 and 100

```
msg.payload = Math.random()*100;  
return msg;
```

```
9/29/2023, 9:58:43 AM node: debug 15  
msg.payload : number  
66.78993241030578
```

```
9/29/2023, 9:58:43 AM node: debug 15  
msg.payload : number  
14.34436881035186
```





Round up value

Math.floor();

The **Math.floor()** static method always rounds down and returns the largest integer less than or equal to a given number.



```
msg.payload = Math.floor(Math.random()*100);  
return msg;
```

9/29/2023, 10:01:11 AM node: debug 15

msg.payload : number

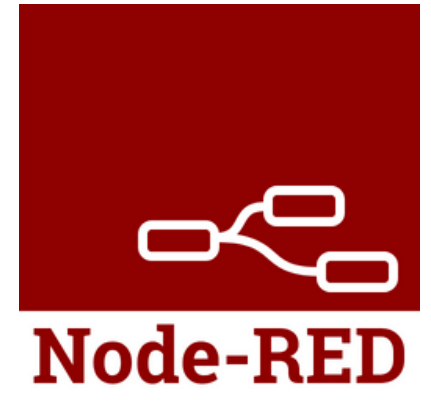
45

9/29/2023, 10:01:12 AM node: debug 15

msg.payload : number

94





Declaring variables- let, var and const

- **let** has block scope. which means **you can use the variable in the specific block of code** in which they were defined.
- **var** has global scope which means **you can use the variable in the entire script**
- **const** is used for the values which are **assigned only once**

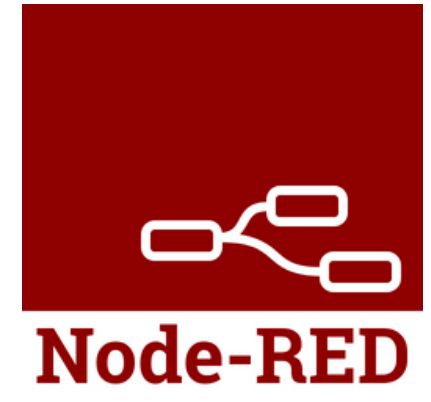
```
if (true) {  
    var varVariable = 100; // Global-scoped variable (var)  
    let letVariable = 200; // Block-scoped variable (let)  
  
    console.log("Inside if block - varVariable:", varVariable);  
    console.log("Inside if block - letVariable:", letVariable);  
}  
  
console.log("\nOutside if block- var variable",varVariable); //0  
// Uncommenting the line below would result in an error  
//console.log("Outside if block- let variable",letVariable);
```

```
let nr1 = 12;  
var nr2 = 8;  
const PI = 3.14159
```

Redeclaring the const variable will give you an error!

```
const Pi = 3.14;  
let Pi = 4.5;
```





Naming variables

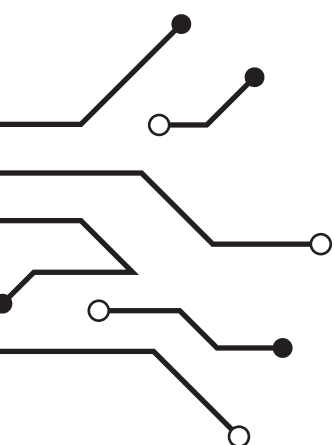
Variable names should be:

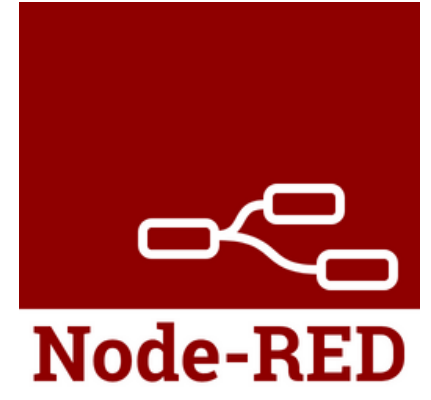
- **Descriptive**
'let iTankLevel' is better than using 'let x'
- **Cannot contain spaces** but using underscore is a nice alternative
'let iTank_Level'

Typical Naming Style: camel case:

This naming style use lower case first and upper case for every new word.

Example: **ageOfBuyer**





String: Double quote or Single quote?

Single and double quotes can be used like below:

```
1 var singleString = 'Hi there!';  
2 var doubleString = "How are you?";
```

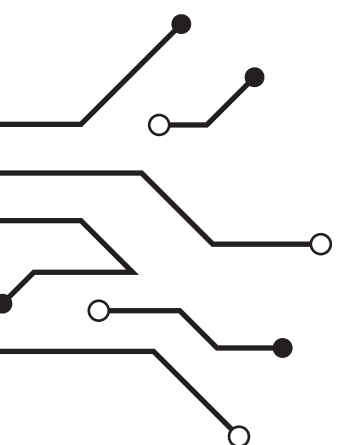
The main thing to understand is that you can use **single quote as literal characters in double quoted strings**, and vice-versa.

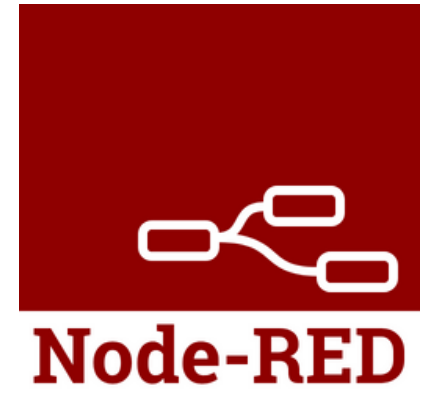
```
4 console.log('Let's learn JavaScript');  
5 console.log("Let's learn JavaScript");  
7 console.log("Do you want to learn JavaScript? \"Yes\"");  
8 console.log('Do you want to learn JavaScript? "Yes"');
```

Using backticks ``

```
1 var language = 'JavaScript';  
2 msg.payload = `Let's learn ${language}`; "Let's learn JavaScript"  
3 return msg;
```

msg.payload : string[22]





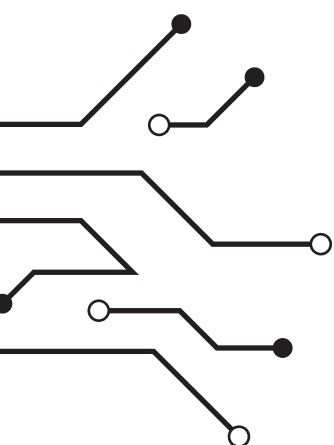
Escape character

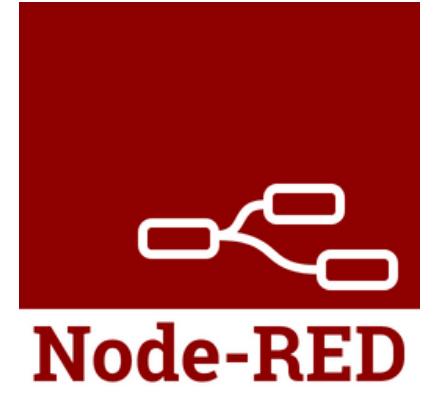
These are used to have **double quotes, single quotes or backticks** in our script.

```
1 var str1 = "Hello, what's your name?Is it \"Mike\"?";  
2 var str2 = 'Hello, what\'s your name?Is it "Mike"?';  
3 console.log(str1) Hello, what's your name?Is it "Mike"?  
4 console.log(str2) Hello, what's your name?Is it "Mike"?
```

So the backslash tells the compiler that the next character is a literal character and not end of the string.

```
1 var str1 = 'New \nline';  
2 var str2 = 'I\'m a backslash --> \\!';  
3 console.log(str1); New  
4 console.log(str2); line  
I'm a backslash --> \!
```

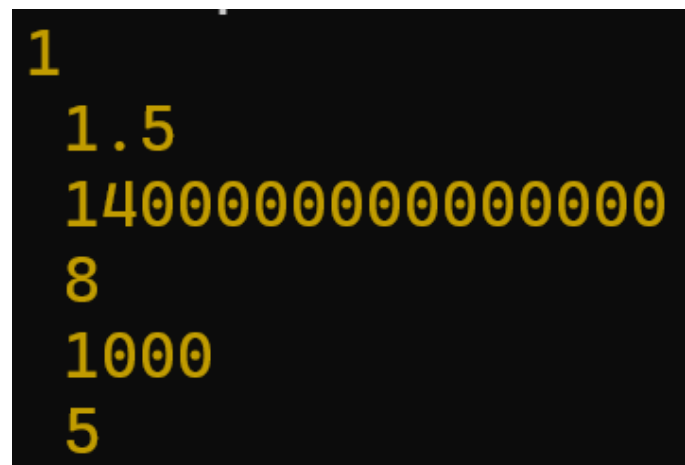




Number

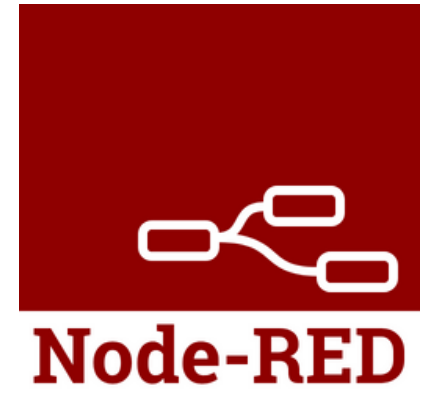
The following are various types of number representations:.

```
1  var intNr = 1;
2  var decNr = 1.5;
3  var expNr = 1.4e15;
4  var octNr = 0o10 //decimal equivalent 8
5  var hexNr = 0x3E8 //decimal equivalent 1000
6  var binNr = 0b101 //decimal equivalent 5
7  console.log(intNr, '\n', decNr, '\n', expNr, '\n', octNr, '\n', hexNr, '\n', binNr);
```

A screenshot of a terminal window with a black background and yellow text. It shows the output of the JavaScript code: the integer 1, the decimal 1.5, the scientific notation 1400000000000000, the octal 8, the hexadecimal 1000, and the binary 5, each on a new line.

```
1
1.5
1400000000000000
8
1000
5
```





Boolean

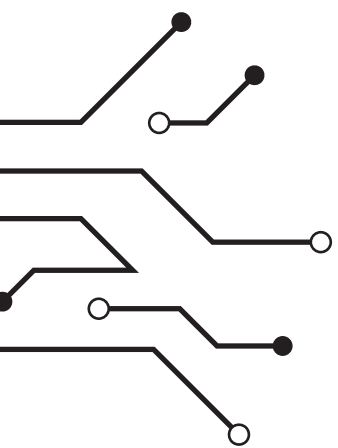
It can hold two values: **true** or **false**.

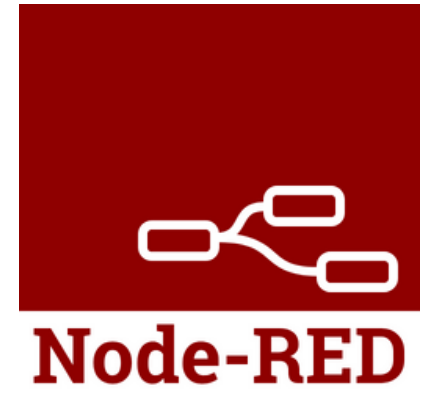
```
var bool1 = false;  
var bool2 = true;
```

Symbol

Brand new datatype that can be used when it is important that variables are not equal even if their value and type are same. These symbol datatype can be used when we use Objects.

```
var sym1 = Symbol('Hello');  
var sym2 = Symbol('Hello');  
console.log(sym1 == sym2) ; false
```





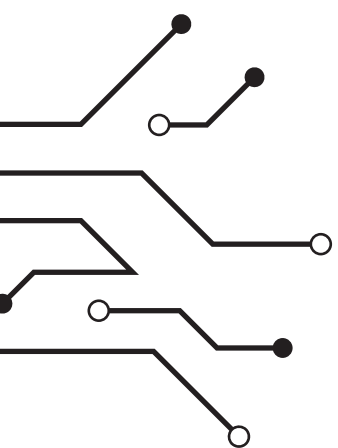
Undefined

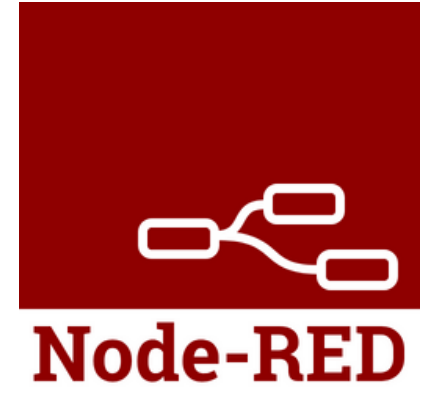
JavaScript has a **special data type** for a variable that has not been assigned a value. And this datatype is called '**undefined**'

```
var value = undefined;  
console.log(value); undefined
```

We should never do that. Because if we are checking if two variables are same or not. If one variable is undefined and we manually assign the other variable to undefined, they will be considered equal.

```
var value1 = undefined;  
var value2;  
console.log(value1 == value2); true
```





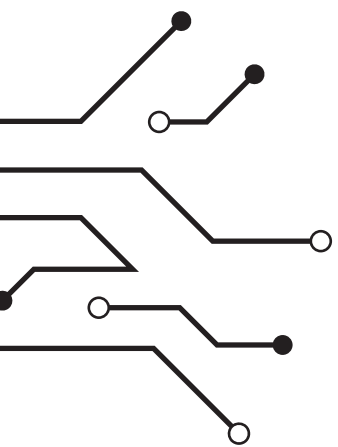
null

null is **case sensitive** and it says that the **variable is empty or has an unknown value**

```
1 var value1 = undefined;
```

It is always better to define the value null rather than undefined to a variable.

```
1 var value1 = undefined;  
2 var value2;  
3 console.log(value1 === value2); true  
4  
5 var value3 = null;  
6 console.log(value3 === value2); false
```



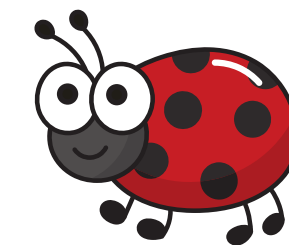
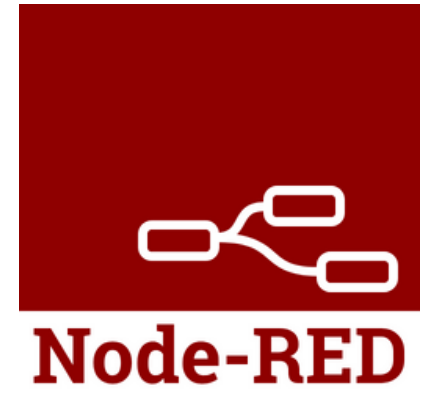
type of

This returns the data type of the variable.

```
var bool = true;  
var str = "Hello";  
var nr = 7;  
var undef = undefined;  
var unknown = null;
```

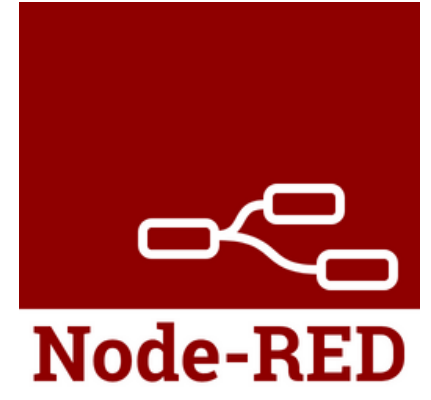
```
console.log("bool", typeof bool);  
console.log("str", typeof str);  
console.log("nr", typeof nr);  
console.log("undef", typeof undef);  
console.log("unknown", typeof unknown);
```

```
bool boolean  
str string  
nr number  
undef undefined  
unknown object
```



The result of null is object which is a **bug since forever** and now cannot be removed due to backward compatibility





Converting data type

Sometimes **JavaScript** converts the data types **automatically** but it could be **dangerous**

```
1 var nr1 = 2;  
2 var nr2 = "2";  
3 console.log(nr1 * nr2);
```

 4

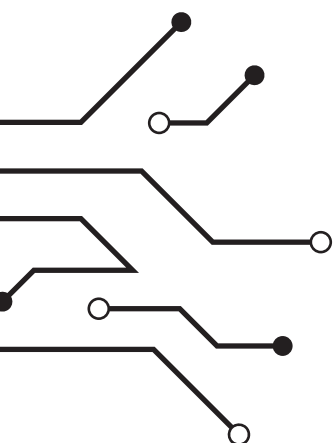
```
1 var nr1 = 2;  
2 var nr2 = "2";  
3 console.log(nr1 + nr2);
```

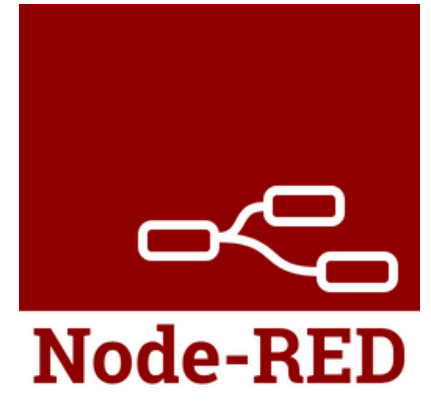
 22

Here the '+' sign has been used to concatenate the string.



We can use JavaScript built-in methods to deal with data type conversions





Converting data type

There are three conversion methods:

String():

It converts variable to type **String** and take any value including **undefined** and **null**

```
1 var nr1 = 2;  
2 msg.payload = String(nr1);  
3 return msg;
```

msg.payload : string[1]

"2"

Number():

It converts variable to type **Number**. If conversion is not possible then it will convert that to **NaN**

```
1 var age = '26';  
2 msg.payload = Number(age);  
3 return msg;
```

msg.payload : number

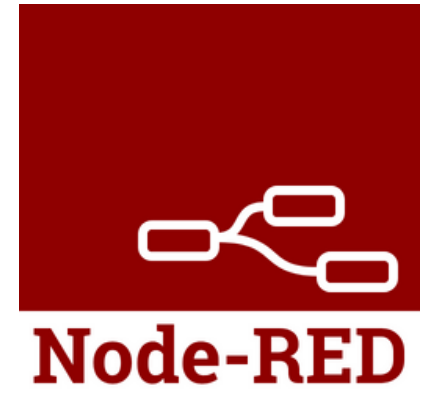
26

Boolean():

It converts the variable to **Boolean**. It will be true for everything except for null, undefined, 0(number) or empty string, and NaN

```
1 console.log(Boolean(22));  
2 console.log(Boolean('22'));  
3 console.log(Boolean(undefined));  
4 console.log(Boolean(0));  
5 console.log(Boolean(''));  
6 console.log(Boolean(NaN));
```

true
true
false
false
false
false



Operators

Operators are used to make calculations

Arithmetic

Addition `+`

Using addition for numbers

```
1 var num1 = 22;  
2 var num2 = 34;  
3 console.log(num1 + num2);
```

56

Using addition for Strings: **Concatenate**

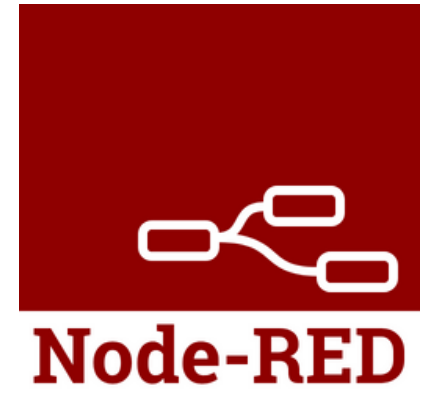
```
1 var str1 = 'Hello '  
2 var str2 = 'World';  
3 console.log(str1 + str2);
```

Hello World

Using addition for numbers and string

```
1 var num1 = 22;  
2 var num2 = '34';  
3 console.log(num1 + num2);
```

2234



Operators

Operators are used to make calculations

Arithmetic

Subtraction '-'

Using subtraction for numbers

```
1 var num1 = 22;  
2 var num2 = 34;  
3 console.log(num1 - num2); -12
```

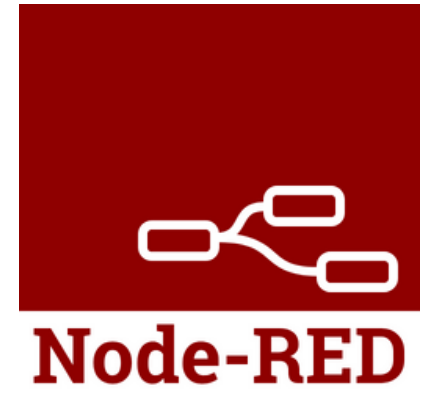
Using subtraction for Strings: **Concatenate**

```
1 var str1 = 'Hello '  
2 var str2 = 'World';  
3 console.log(str1 * str2); NaN
```

Using subtraction for numbers and strings

```
1 var num1 = 22;  
2 var str2 = 'World';  
3 console.log(num1 - str2); NaN
```





Operators

Operators are used to make calculations

Arithmetic

Multiplication `*`

Using multiplication for numbers

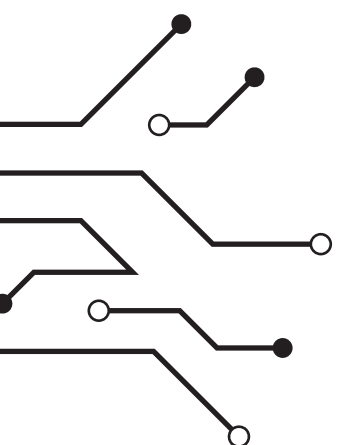
```
1 var num1 = 22;  
2 var num2 = 4;  
3 console.log(num1 * num2); 88
```

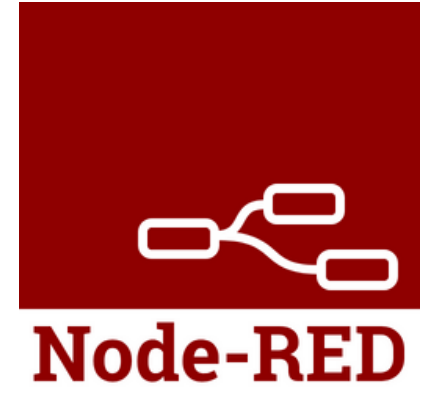
Using multiplication for Strings: **Concatenate**

```
1 var num1 = "22";  
2 var num2 = "4";  
3 console.log(num1 * num2); 88
```

Using multiplication for numbers and strings

```
1 var num1 = "22";  
2 var num2 = 4;  
3 console.log(num1 * num2); 88
```





Operators

Operators are used to make calculations

Arithmetic

Division `/`

Using division for numbers

```
1 var num1 = 22;  
2 var num2 = 4;  
3 console.log(num1 / num2);
```

5.5

Modulus `%`

It tells the remainder after division

```
1 var num1 = 22;  
2 var num2 = 4;  
3 console.log(num1 % num2);
```

2

```
1 var num1 = 20;  
2 var num2 = 4;  
3 console.log(num1 % num2);
```

0

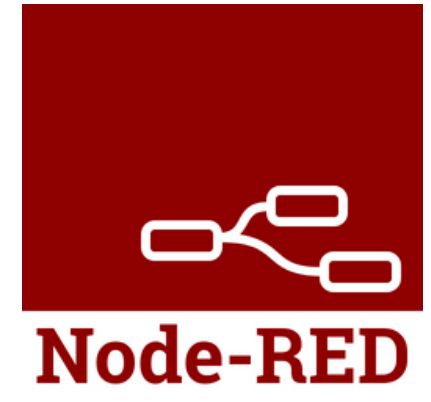
Exponentiation `**`

It means raising the base number to the power of the exponent

```
1 var num1 = 20;  
2 var num2 = 4;  
3 console.log(num1 ** num2);
```

160000





Operators

Operators are used to make calculations

Unary operators

Increment (+1)

```
1 var num1 = 20;  
2 num1++;  
3 console.log(num1); 21
```

Decrement (-1)

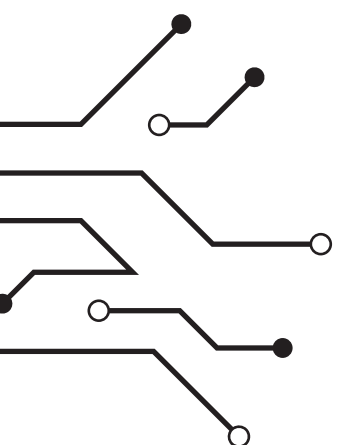
```
1 var num1 = 20;  
2 num1--;  
3 console.log(num1); 19
```

Prefix operator

```
1 var num1 = 20;  
2 console.log(++num1); 21
```

Postfix operator

```
1 var num1 = 20;  
2 console.log(num1++); 20
```



Comparison Operators

Loose equality `==` ; Strict equality `===`

The output of the comparison operator is always **Boolean**

Equal `==` and `===`

```
1 var num1 = 20;  
2 var num2 = 20;  
3 console.log(num1 == num2); true
```

```
1 var num1 = 20;  
2 var num2 = '20';  
3 console.log(num1 == num2); true
```

```
1 var num1 = 20;  
2 var num2 = '20';  
3 console.log(num1 === num2); false
```

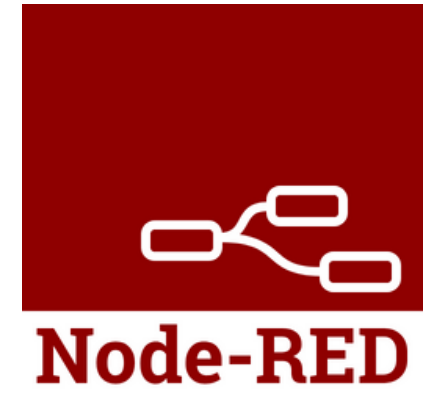
Not Equal `!=` and `!==`

```
1 var num1 = 20;  
2 var num2 = 20;  
3 console.log(num1 != num2); false
```

```
1 var num1 = 20;  
2 var num2 = '20';  
3 console.log(num1 != num2); false
```

```
1 var num1 = 20;  
2 var num2 = '20';  
3 console.log(num1 !== num2); true
```





Comparison Operators

Loose equality `==` ; Strict equality `===`

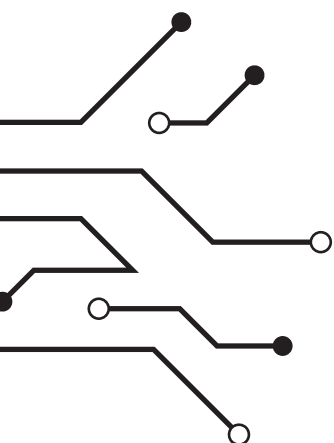
The output of the comparison operator is always **Boolean**

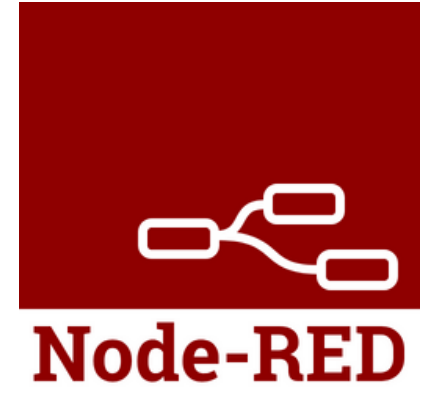
Greater than

```
1 var num1 = 20;  
2 var num2 = 20;  
3 console.log(num1 > num2); false  
4 console.log(num1 >= num2); true
```

Smaller than

```
1 var num1 = 20;  
2 var num2 = 20;  
3 console.log(num1 < num2); false  
4 console.log(num1 <= num2); true
```





Logical Operators

The output of the logical operator is always **Boolean**

AND

```
1 var num1 = 10;
2 var num2 = 20;
3 var num3 = 30
4 console.log(num1 < num2 && num2 < num3); true
5 console.log(num1 > num2 && num2 < num3); false
```

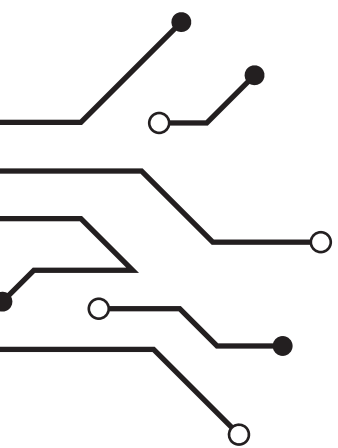
OR

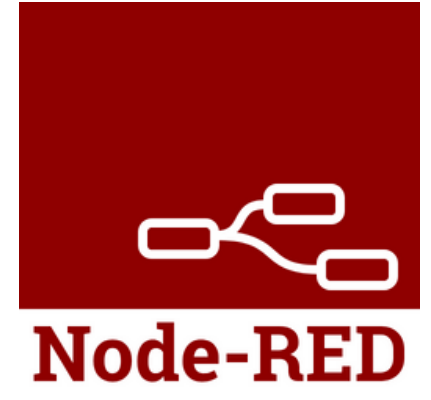
```
1 var num1 = 10;
2 var num2 = 20;
3 var num3 = 30
4 console.log(num1 < num2 || num2 < num3); true
5 console.log(num1 > num2 || num2 < num3); true
```

NOT

```
1 var bool1 = false;
2 console.log(!bool1); true

1 var num1 = 20;
2 var num2 = 30
3 console.log(!(num1 < num2)); false
```





Project 1

Write a program to convert **Celcius to Fahrenheit** and log the output in the following format:

```
The temperature 40°C is equivalent to 104°F!
```

$\text{Fahrenheit} = (\text{Celcius} * 9/5) + 32$

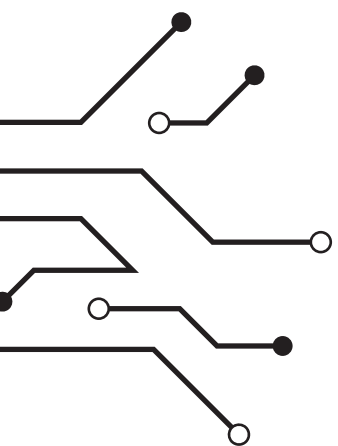
Project 2

Set value for height in inches and weight in pounds, then convert the values to centimeter and kilos and output the result

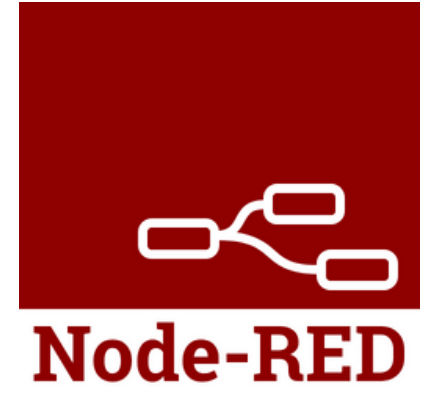
$1 \text{ inch} = 2.54 \text{ cm}$

$2.2046 \text{ pound} = 1 \text{ Kilo}$

Then calculate the BMI which is equivalent to weight (in kilos) divided by squared height (in meters). Output the BMI result to the console or debug window



Code  Compile



Thank you!

www.codeandcompile.com

