Code </> Compile

learning made easy

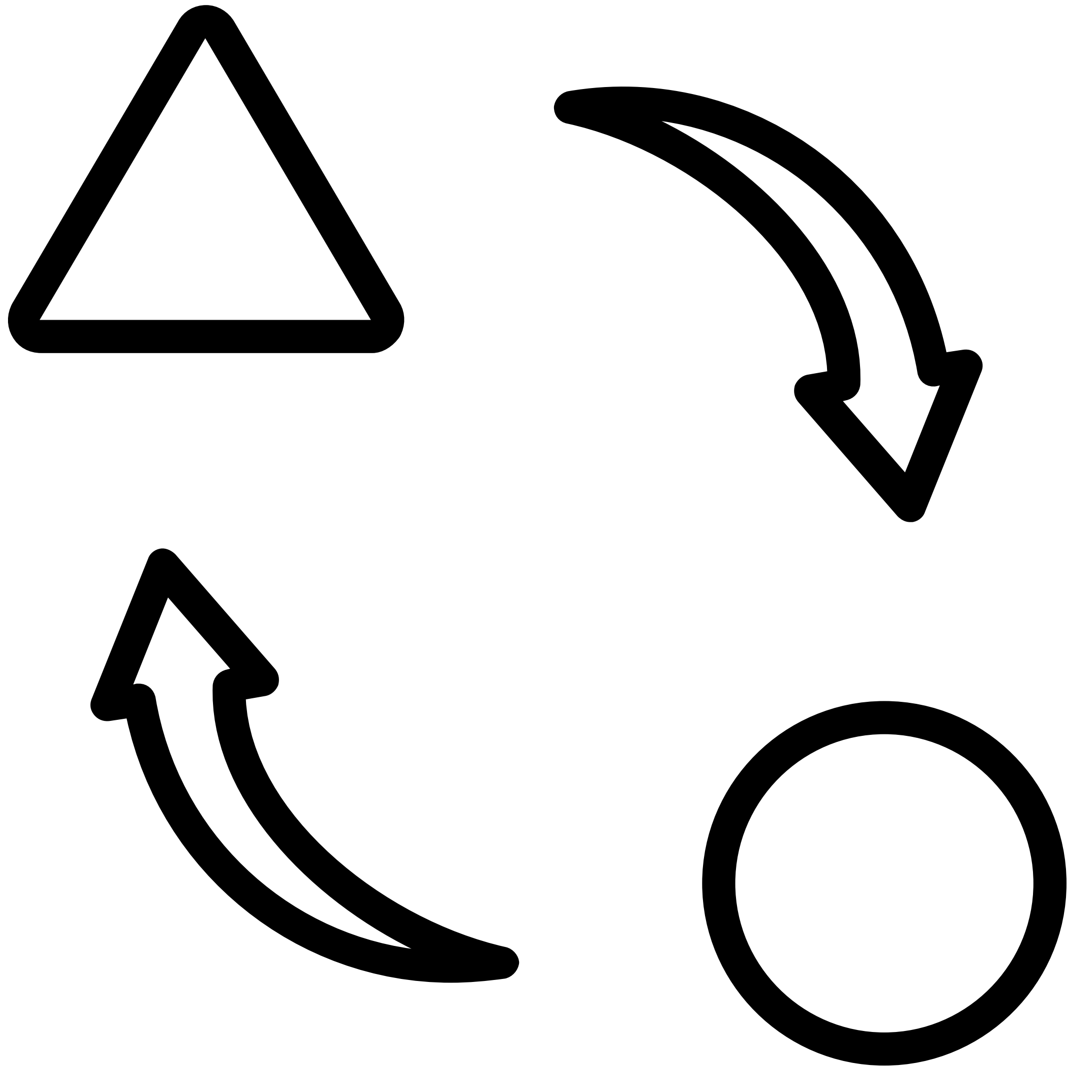
JS

Node-RED

JavaScript essentials for Node-RED

Section 6

Methods



www.codeandcompile.com

Code </> Compile
learning made easy

JS

What you will learn in this section?

In this section, we will learn about the built-in JavaScript methods which can be used to **improve the effectiveness of the code, save time and efforts**

- String methods
- Math methods
- Date methods
- Array methods
- Number methods

Advantages:

Can be used anytime in the code

Reduce the code length

Code </> Compile

learning made easy

console.log()

```
function testFunction()
{
    var value = 49;
    console.log('The course price is: ' + value)
}

testFunction();
```

```
The course price is: 49
```

Code </> Compile
learning made easy

JS

Node-RED

Parsing numbers

parseInt and **parseFloat**

Easiest way to parse values to integer and float

```
let valueString = "6.56";
let valueBinary = 0b1011;
console.log(parseInt(valueString));
console.log(parseInt(valueBinary));

console.log(parseFloat(valueString));
console.log(parseFloat(valueBinary));

return msg;
```

```
6
11
```

```
6.56
11
```

Number()

```
let valueString = "6.56";
let valueBinary = 0b1011;
console.log(Number(valueString));
console.log(Number(valueBinary));
return msg;
```

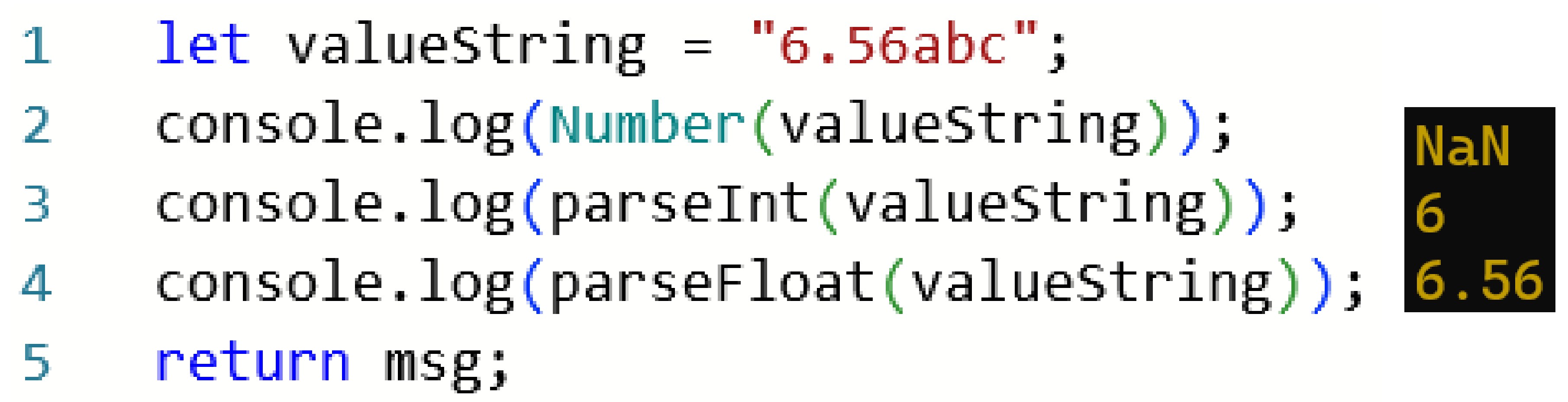
```
6.56
11
```

www.codeandcompile.com

Code </> Compile
learning made easy

Difference between Number() and parseFloat()

By utilizing parseFloat, a trimmed string that begins with one or more numeric characters and is followed by alphanumeric characters can be converted into a Number, whereas using Number may not yield the desired outcome.



```
let valueString = "6.56abc";
console.log(Number(valueString));
console.log(parseInt(valueString));
console.log(parseFloat(valueString));
return msg;
```

```
NaN
6
6.56
```

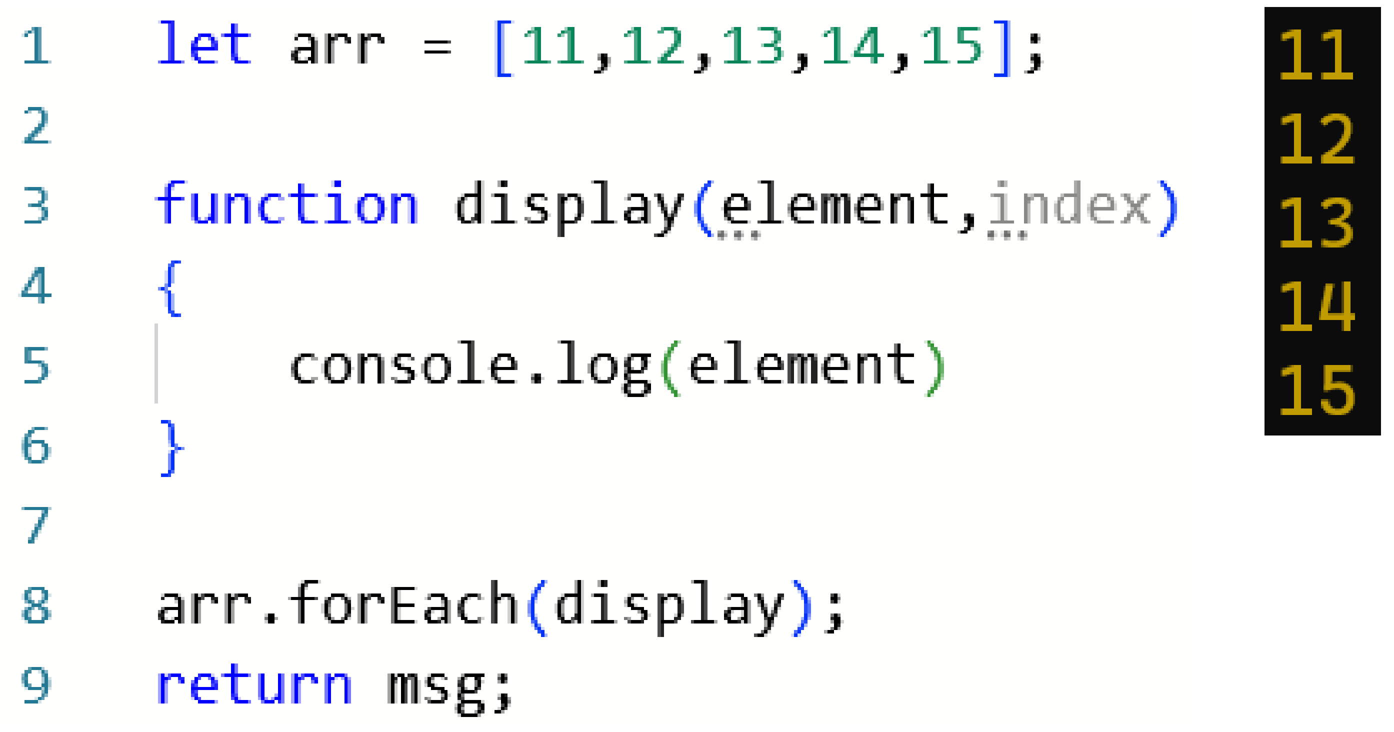
Code </> Compile
learning made easy

JS Node-RED

Array methods

forEach()

A built-in function that can be used for **executing function on each element of array.** This can be used when we want to manipulate the array.

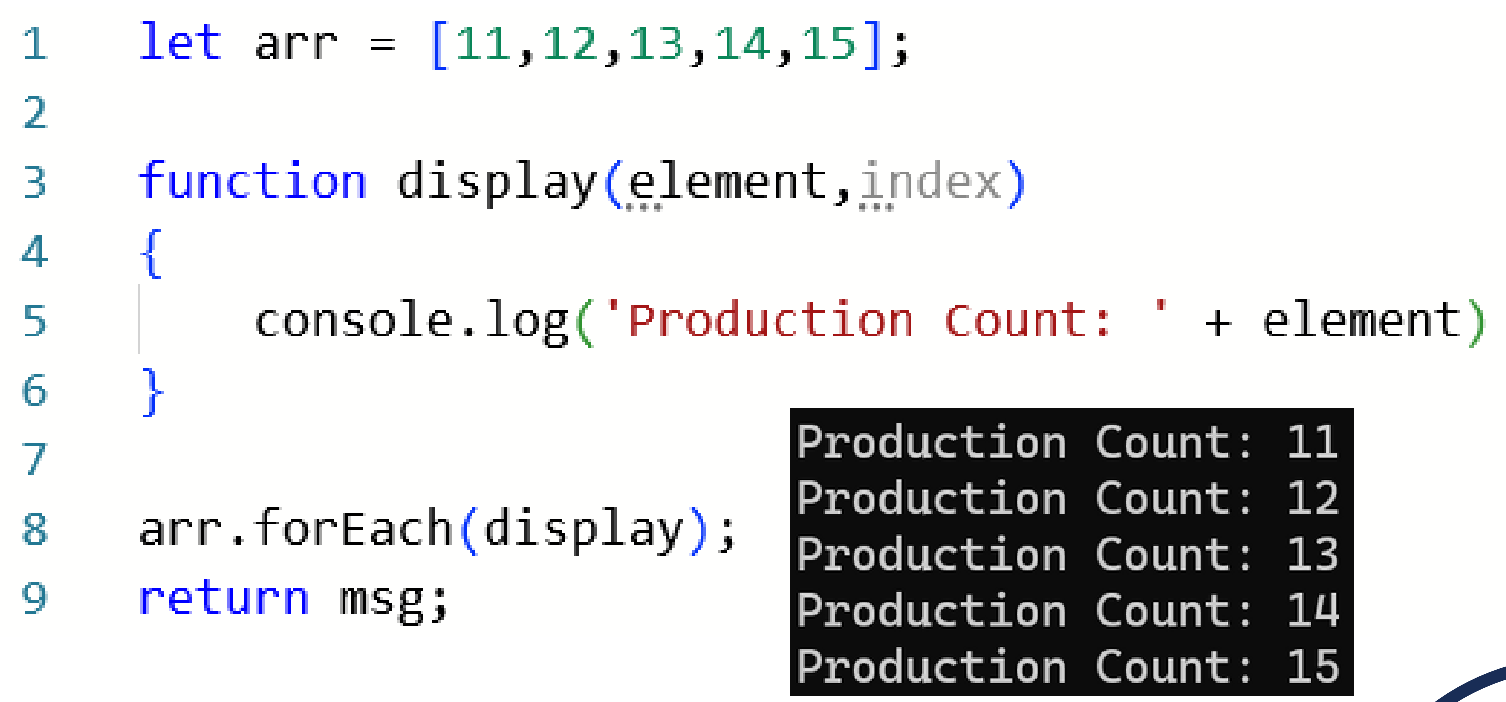


```
let arr = [11,12,13,14,15];

function display(element,index)
{
    console.log(element)
}

arr.forEach(display);
return msg;
```

```
11
12
13
14
15
```



```
let arr = [11,12,13,14,15];

function display(element,index)
{
    console.log('Production Count: ' + element)
}

arr.forEach(display);
return msg;
```

```
Production Count: 11
Production Count: 12
Production Count: 13
Production Count: 14
Production Count: 15
```

We do not control the flow of the loop so we **never get stuck!**

www.codeandcompile.com

Code </> Compile
learning made easy

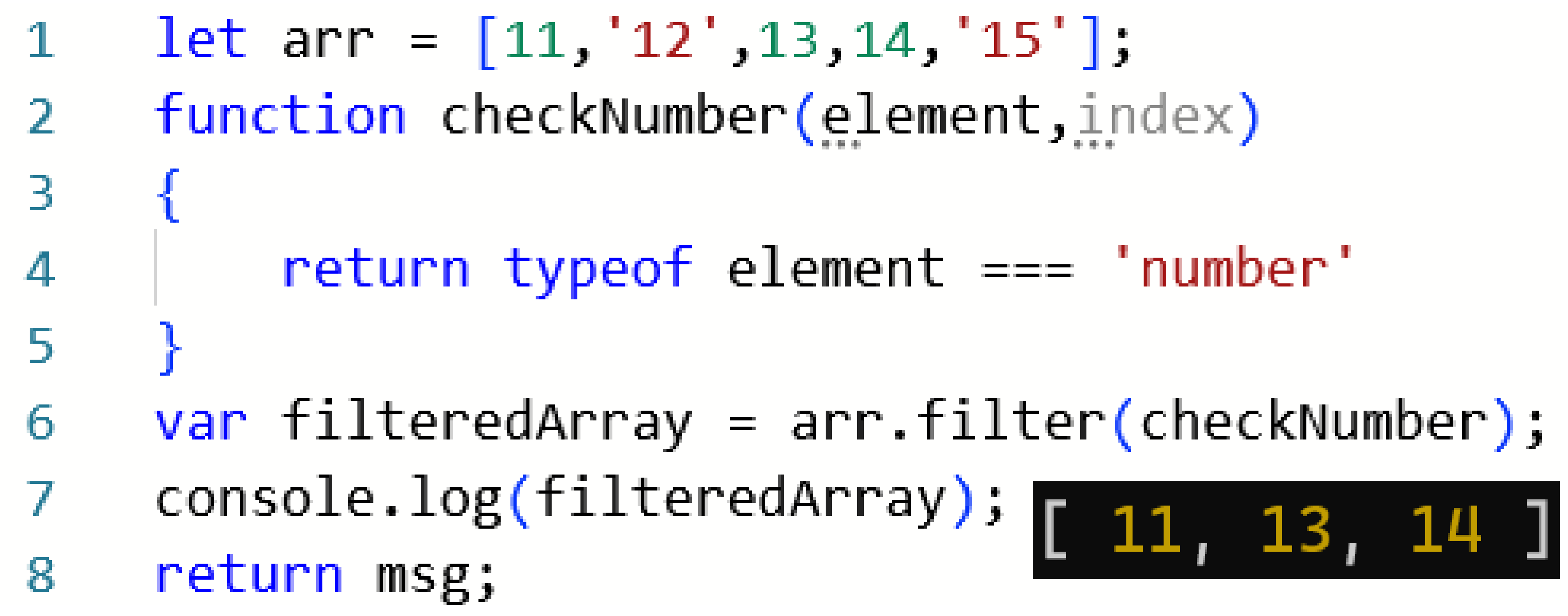
JS

Node-RED

Array methods

filter()

It takes the function as an argument and this **function should return a Boolean value.** If the Boolean value is True, the element will end up in the filtered array else not.



```
let arr = [11,'12',13,14,'15'];
function checkNumber(element,index)
{
    return typeof element === 'number'
}
var filteredArray = arr.filter(checkNumber);
console.log(filteredArray);
return msg;
```

```
[ 11, 13, 14 ]
```

www.codeandcompile.com

Code </> Compile
learning made easy

JS Node-RED

Array methods

map()

This function is used when you want to change all the values of the array.

Incrementing each element of the array with +1

```
let arr1 = [11,12,13,14,15];
let arr2 = arr1.map(x => x + 1);
console.log(arr2);
return msg;
```

```
[ 12, 13, 14, 15, 16 ]
```

Adding string to the array element

```
let arr1 = [11,12,13,14,15];
let arr2 = arr1.map(x => 'LineA: ' + x);
console.log(arr2);
return msg;
```

```
[ 'LineA: 11', 'LineA: 12', 'LineA: 13',
  'LineA: 14', 'LineA: 15' ]
```

www.codeandcompile.com

Code </> Compile
learning made easy

JS

Node-RED

Array methods

map()

This function is used when you want to change all the values of the array.

Mapping negative elements to positive

```
let arr = [11,-12,-13,14,15];
function checkPositive(element,index)
{
    if (element < 0)
    {
        return (element * -1)
    }
    else
    return (element)
}
var filteredArray = arr.map(checkPositive);
console.log(filteredArray);
return msg;
```

```
[ 11, 12, 13, 14, 15 ]
```

Code </> Compile
learning made easy

JS Node-RED

Exercises

Exercise 1:
Utilize the array method **forEach()** to transform the elements of the given array into integers and store them in a new array.

```
var arr1 = [11.34,22.3,10.23,10.45,43.1];
```

Result

```
msg.payload : array[5]
 ▸[ 11, 22, 10, 10, 43 ]
```

Exercise 2:
Utilize the array method **filter()** to separate the elements of the given array into two arrays: one for positive values and another for negative values.

```
var arr1 = [-11.34,22.3,-10.23,-10.45,43.1];
```

Result

```
msg.payload : array[2]
 ▸[ 22.3, 43.1 ]
2/9/2024, 11:31:37 AM   node: debug 119
msg.payload : array[3]
 ▸[ -11.34, -10.23, -10.45 ]
```

Code </> Compile
learning made easy

JS Node-RED

Exercises

Exercise 3:

Given an array of temperatures in Fahrenheit, use .map() to create a new array where the temperatures are converted to Celsius. The formula for conversion is (F - 32) * 5/9.

```
var arr1 = [100,98.4,101,94.67];
```

Result

```
▸ [ 37.78, 36.89, 38.33, 34.82 ]
```

Code </> Compile
learning made easy

String methods

String methods are used to manipulate the string. The following are some of the most commonly used string functions.

- Combining string
- String to array
- Array to string
- Index and positions
- Creating substring
- Replacing part of the string
- Uppercase and Lowercase

Code </> Compile
learning made easy

JS

Node-RED

Combining String

concat

```
let str1 = "Hello ";
let str2 = "Code and Compile";
console.log(str1.concat(str2));
return msg;
```

```
Hello Code and Compile
```

```
let str1 = 123;
let str2 = "Code and Compile";
console.log(str1 + str2)
console.log(str1.concat(str2));
return msg;
```

```
123Code and Compile
9 Feb 13:15:14 - [error] [function:concat] TypeError: str
1.concat is not a function
```

Code </> Compile
learning made easy

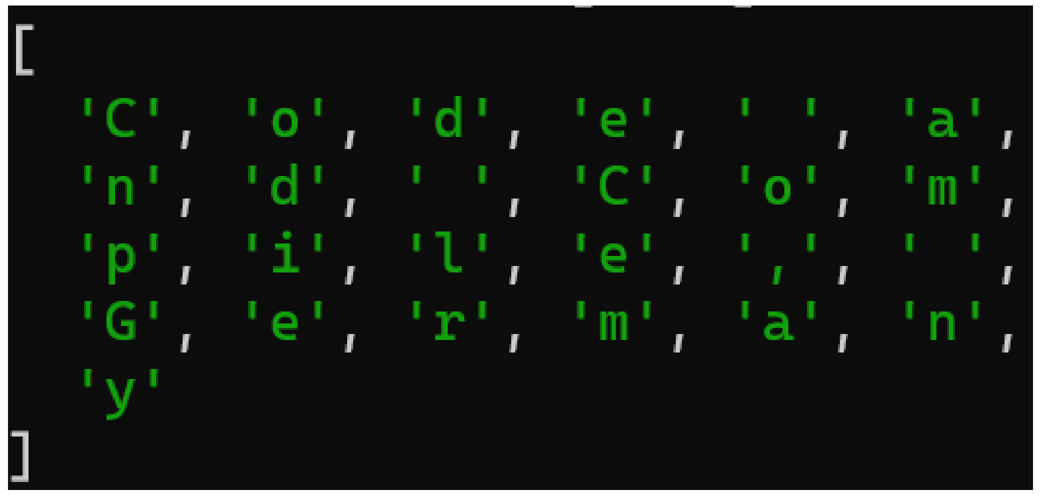
JS

Node-RED

String to Array

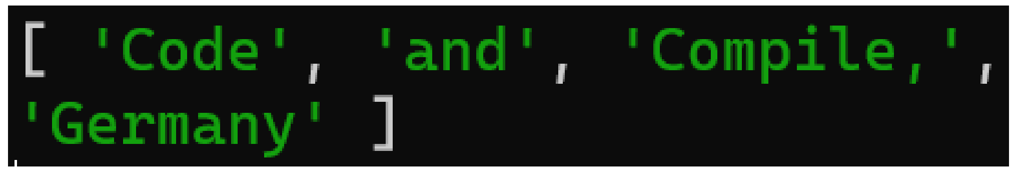
split

```
let str1 = "Code and Compile, Germany";
console.log(str1.split(""));
return msg;
```



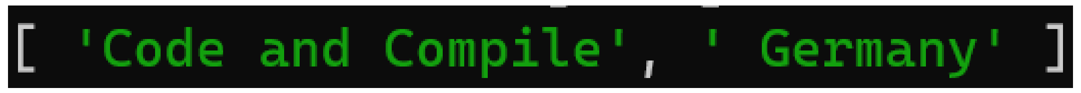
```
[
  'C', 'o', 'd', 'e', ' ', 'a',
  'n', 'd', ' ', 'C', 'o', 'm',
  'p', 'i', 'l', 'e', ',', ' ',
  'G', 'e', 'r', 'm', 'a', 'n',
  'y'
]
```

```
let str1 = "Code and Compile, Germany";
console.log(str1.split(" "));
return msg;
```



```
[ 'Code', 'and', 'Compile,',
'Germany' ]
```

```
let str1 = "Code and Compile, Germany";
console.log(str1.split(","));
return msg;
```



```
[ 'Code and Compile', ' Germany' ]
```

www.codeandcompile.com

Code </> Compile

learning made easy

JS

Node-RED

String to Array

join

```javascript
let arr1 = ["h","e","l","l","o"];
console.log(arr1.join());
return msg;
```

```
h,e,l,l,o
```

```javascript
let arr1 = ["h","e","l","l","o"];
console.log(arr1.join(''));
return msg;
```

```
hello
```

```javascript
let arr1 = ["h","e","l","l","o"];
console.log(arr1.join('-'));
return msg;
```

```
h-e-l-l-o
```

```javascript
let arr1 = [1,2,3,4,5];
console.log(arr1.join('-'));
return msg;
```

```
1-2-3-4-5
```

Code </> Compile
learning made easy

JS

Node-RED

index and positions

indexOf

```
let str1 = "Hello Code and Compile!";
console.log(str1.indexOf('C'));
return msg;
```

```
6
```

```
let str1 = "Hello Code and Compile!";
console.log(str1.indexOf('Code'));
return msg;
```

```
6
```

```
let str1 = "Hello Code and Compile!";
console.log(str1.indexOf('G'));
return msg;
```

```
-1
```

lastIndexOf

```
let str1 = "Hello Code and Compile!";
console.log(str1.lastIndexOf('o'));
return msg;
```

```
16
```

charAt

```
let str1 = "Hello Code and Compile!";
console.log(str1.charAt(6));
return msg;
```

```
C
```

If the value is out of the range, it will return an empty string

Code </> Compile
learning made easy

JS

Node-RED

Substring

It requires two parameters: the starting index and the ending index. If you omit the ending index, it will continue until the end of the string. The ending index is not inclusive.

```
let str1 = "Hello Code and Compile!"
console.log(str1.slice(6));
return msg;
```

```
Code and Compile!
```

```
let str1 = "Hello Code and Compile!";
console.log(str1.slice(6,11));
return msg;
```

```
Code
```

Code </> Compile
learning made easy

Replacing

This is used to replace part of the string

```
let str1 = "Hello Code and Compile!";
console.log(str1.replace("Code and Compile", "Rajvir"));
return msg;
```

```
Hello Rajvir!
```

```
let str1 = "H3llo Cod3 and Compil3!";
console.log(str1.replaceAll("3", "e"));
return msg;
```

```
Hello Code and Compile!
```

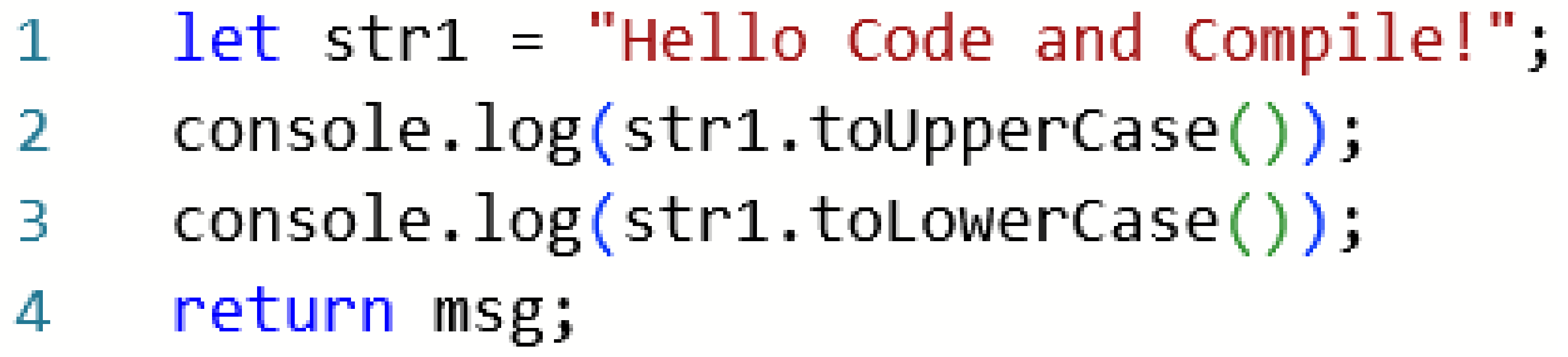
Code </> Compile
learning made easy

JS

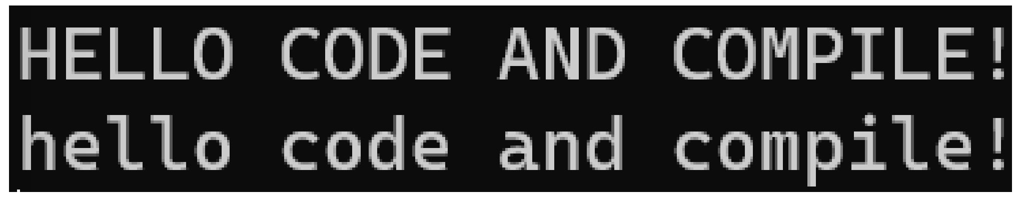
Node-RED

Uppercase and lowercase

These methods are utilized to capitalize and decapitalize the string.



```
let str1 = "Hello Code and Compile!";
console.log(str1.toUpperCase());
console.log(str1.toLowerCase());
return msg;
```

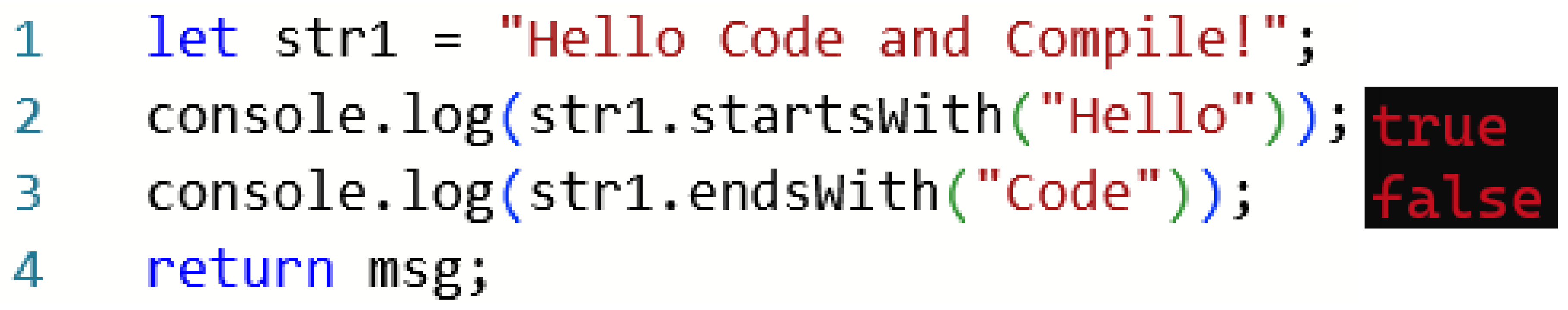


```
HELLO CODE AND COMPILE!
hello code and compile!
```

Code </> Compile
learning made easy

Start and end of the string

These methods are employed to confirm whether a string begins or ends with a particular letter or sequence of letters.



```javascript
let str1 = "Hello Code and Compile!";
console.log(str1.startsWith("Hello"));
console.log(str1.endsWith("Code"));
return msg;
```

```
true
false
```

Code </> Compile
learning made easy

Exercises

Exercise 1:
Utilize various string methods to get the following result:

```
let str1 = ["H3ll0","C0de", "&","C0mpil3"];
```

Result
```
Hello Code and Compile ,Germany
```

Exercise 2:
Utilize various string methods to capitalize only first letter of the each word

```
var str1 = "Hello Code and compile";
var str2 = "hello cODE and cOMpile";
```

Result
```
Hello Code And Compile
```

Code </> Compile
learning made easy

JS Node-RED

Number methods

There are some built-in number methods that we can use directly.

Specifying number of decimals

This allows us to determine the number of decimal places we can specify.

```
var value = 12.16734563;
console.log(value.toFixed(3)); 12.167
```

"12.167" ⚠ It converts the output to string format

Specifying precision

This allows us to get the number with the defined precision.

```
var value = 12.16734563;
console.log(value.toPrecision(3)); 12.2
```

"12.2" ⚠ It converts the output to string format

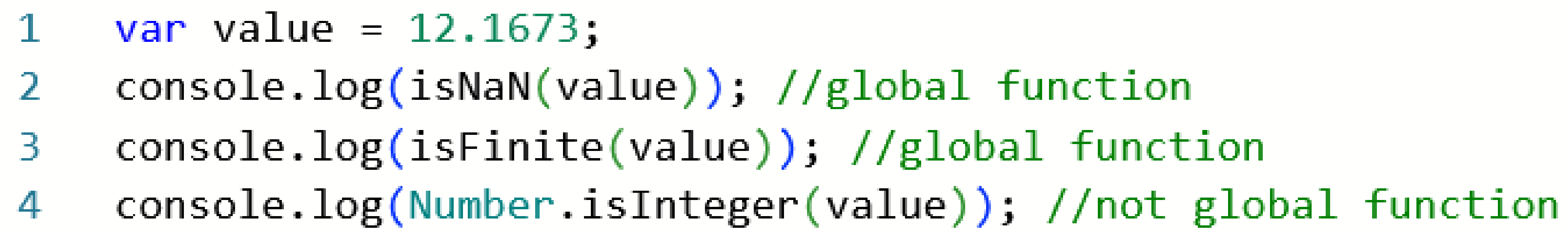
Code </> Compile
learning made easy

JS

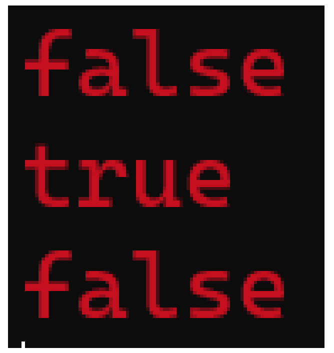
Node-RED

Number methods

To check property of the number



```
var value = 12.1673;
console.log(isNaN(value)); //global function
console.log(isFinite(value)); //global function
console.log(Number.isInteger(value)); //not global function
```



```
false
true
false
```

www.codeandcompile.com

Code </> Compile
learning made easy

JS Node-RED

Math methods

There are some built-in number to perform calculations and operations on numbers

Find highest and lowest number

```
let numbers = [100, 31, 24, 15, 64];
console.log(Math.max(...numbers));
console.log(Math.min(...numbers));
```

```
100
15
```

The spread operator in JavaScript, denoted by three dots (...), is a convenient way to **expand elements of an iterable (such as an array) or an object's properties in places where multiple elements or variables are expected**

www.codeandcompile.com

Code </> Compile
learning made easy

JS Node-RED

Math methods

Find highest and lowest number for large arrays

For very large arrays, using the spread operator with Math.max() or Math.min() might result in a "**Maximum call stack size exceeded**" error because it essentially **tries to pass a large number of arguments to a function**. An alternative approach for large arrays is to use the **reduce()** method:

```
var arr = []
for (let index = 0; index < 1000000; index++) {
    arr.push(Math.random()*100)
}
let max = arr.reduce((a, b) => Math.max(a, b));
let min = arr.reduce((a, b) => Math.min(a, b));
console.log(max); 99.99975156036567
console.log(min); 0.000010017592599886882
```

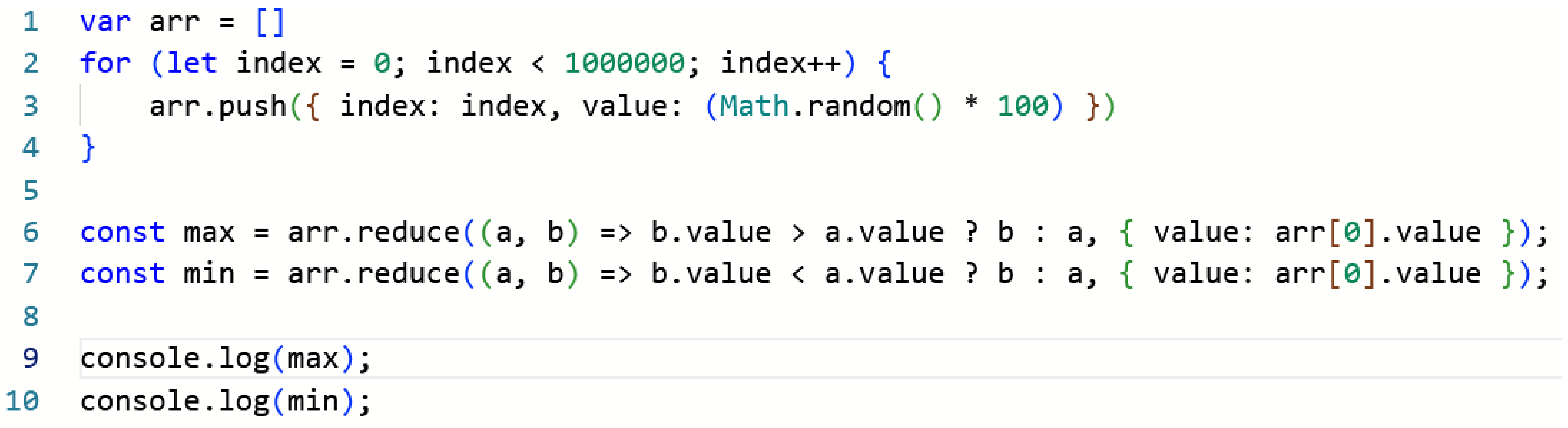
The reduce method is used to **reduce the array to a single value**

Code </> Compile
learning made easy

JS Node-RED

Math methods

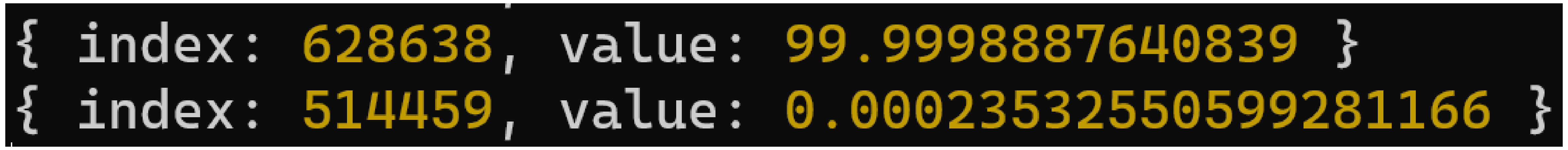
Find highest and lowest number for large arrays with objects



```
var arr = []
for (let index = 0; index < 1000000; index++) {
    arr.push({ index: index, value: (Math.random() * 100) })
}

const max = arr.reduce((a, b) => b.value > a.value ? b : a, { value: arr[0].value });
const min = arr.reduce((a, b) => b.value < a.value ? b : a, { value: arr[0].value });

console.log(max);
console.log(min);
```



```
{ index: 628638, value: 99.9998887640839 }
{ index: 514459, value: 0.00023532550599281166 }
```

www.codeandcompile.com

Code </> Compile
learning made easy

Math methods

There are some built-in number to perform calculations and operations on numbers

Square root and raising the power

```
let number = 64;
console.log(Math.sqrt(number));
console.log(Math.pow(number,2));
```

```
8
4096
```

JS

Node-RED

Code </> Compile
learning made easy

JS Node-RED

Math methods

Math.round, Math.ceil, Math.floor and Math.trunc

These methods are used to convert decimals to integers

- **Math.round:** Rounds a number to the nearest integer. If the fractional part is 0.5 or higher, it rounds up; otherwise, it rounds down. Example: **Math.round(1.5)** gives 2, and **Math.round(1.4)** gives 1.
- **Math.ceil**: Rounds a number up to the next largest integer. It always rounds up, regardless of the fractional part. **Math.ceil(1.1)** gives 2
- **Math.floor**: Rounds a number down to the nearest integer. It always rounds down, no matter what the fractional part is. **Math.floor(1.9)** gives 1
- **Math.trunc**: Removes the decimal part of the number, truncating it to an integer without rounding. It just cuts off the digits after the decimal point. **Math.trunc(1.9)** gives 1, and Math.trunc(-1.9) gives -1

Code </> Compile

learning made easy

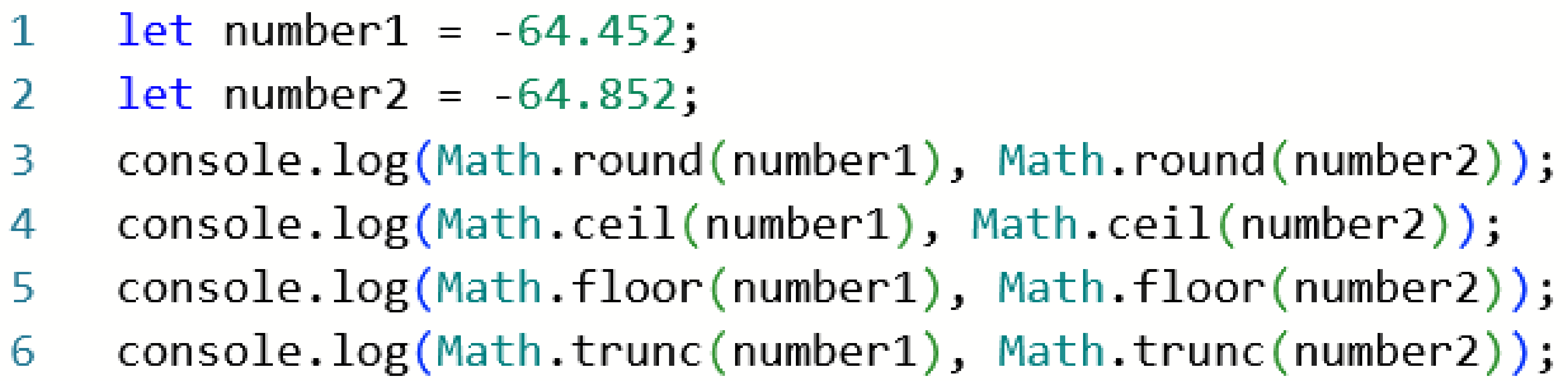
JS

Node-RED

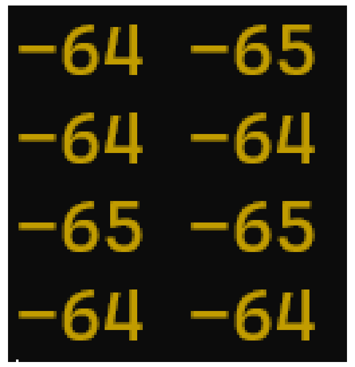
Math methods

Math.round, Math.ceil, Math.floor and Math.trunc

These methods are used to convert decimals to integers



```
let number1 = -64.452;
let number2 = -64.852;
console.log(Math.round(number1), Math.round(number2));
console.log(Math.ceil(number1), Math.ceil(number2));
console.log(Math.floor(number1), Math.floor(number2));
console.log(Math.trunc(number1), Math.trunc(number2));
```



```
-64 -65
-64 -64
-65 -65
-64 -64
```

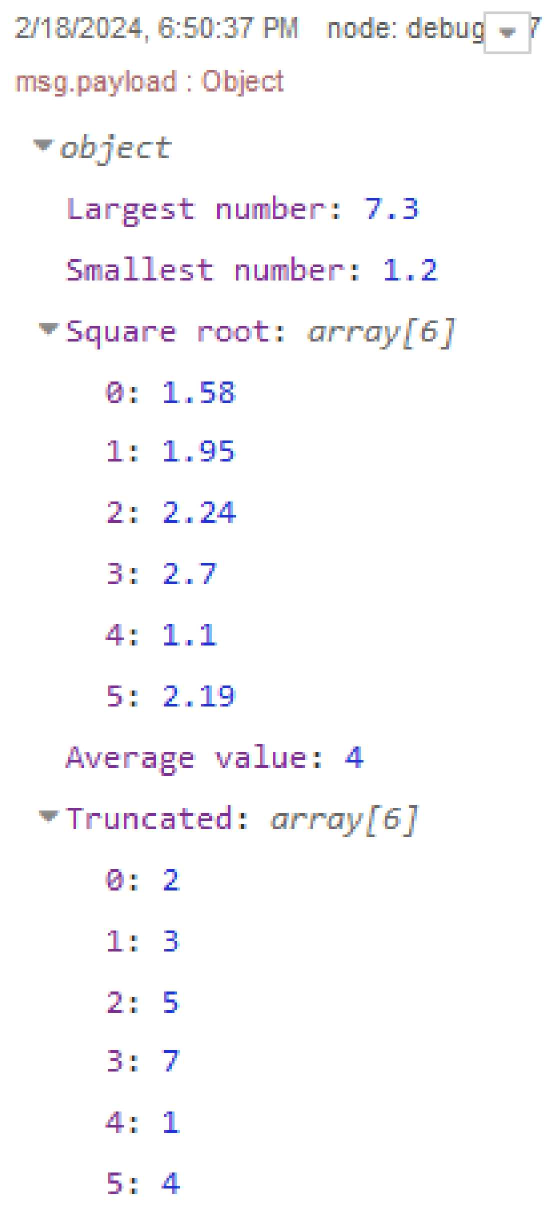
Exercise

Analyzing and Processing Number Data

Given an array of positive numbers, perform various mathematical operations to analyze and process the data.

```javascript
let numbers = [2.5, 3.8, 5, 7.3, 1.2, 4.8];
```

- Determine the largest and smallest numbers in the array.
- Create a new array containing the square roots of each number, rounded to max. two decimal places.
- Calculate the average (mean) of the numbers, then round it to the nearest whole number.
- Create a new array where each number's decimal points are truncated.



```
2/18/2024, 6:50:37 PM   node: debug 7
msg.payload : Object
▾object
  Largest number: 7.3
  Smallest number: 1.2
 ▾Square root: array[6]
    0: 1.58
    1: 1.95
    2: 2.24
    3: 2.7
    4: 1.1
    5: 2.19
  Average value: 4
 ▾Truncated: array[6]
    0: 2
    1: 3
    2: 5
    3: 7
    4: 1
    5: 4
```

Code </> Compile
learning made easy

JS

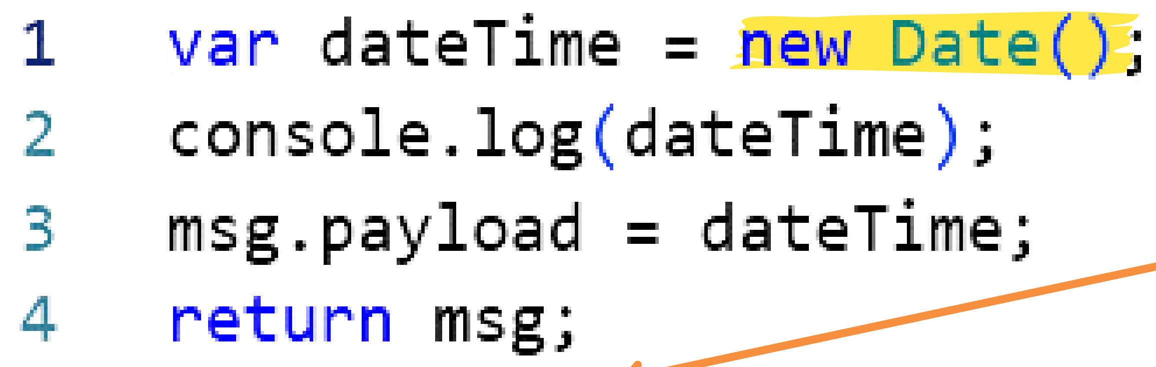
Node-RED

Date methods

In JavaScript, there are built in Date objects which has a lot of built-in functions to implement on the date.

Creating dates

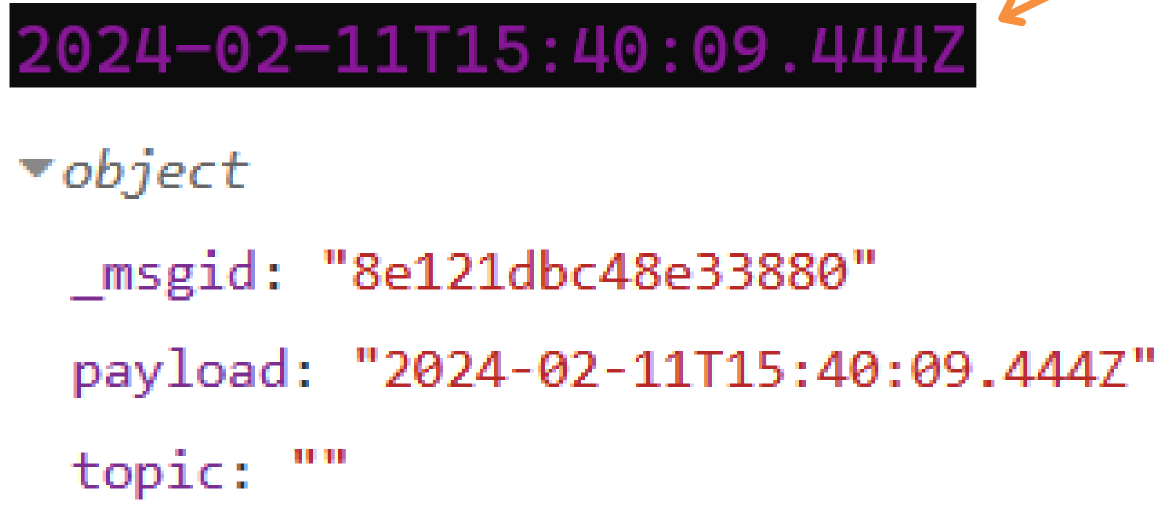
new Date() returns a date object (UTC) with the current date and time.



```
var dateTime = new Date();
console.log(dateTime);
msg.payload = dateTime;
return msg;
```

Date and time is separated with a capital T.

UTC time is defined with a capital letter Z.



```
2024-02-11T15:40:09.444Z

▾object
  _msgid: "8e121dbc48e33880"
  payload: "2024-02-11T15:40:09.444Z"
  topic: ""
```

www.codeandcompile.com

Code </> Compile
learning made easy

JS

Node-RED

Date methods

Get Methods

- **getFullYear()**: Get year as a four digit number (yyyy)
- **getMonth()**: Get month as a number (0-11)
- **getDate()**: Get day as a number (1-31)
- **getDay()**: Get weekday as a number (0-6). In JavaScript, the first day of the week (day 0) is Sunday.
- **getHours()**: Get hour (0-23)
- **getMinutes()**: Get minute (0-59)
- **getSeconds()**: Get second (0-59)
- **getMilliseconds()**: Get millisecond (0-999)
- **getTime()**: Get time (milliseconds since January 1, 1970)

Code </> Compile
learning made easy

JS

Node-RED

Date methods

Change date format

```
var dateTime = new Date();
console.log(dateTime.toLocaleString('de-DE'));
console.log(dateTime.toLocaleString('en-CA'));
console.log(dateTime.toLocaleString('sv-SE'));
```

```
11.2.2024, 17:20:31
2024-02-11, 5:20:31 p.m.
2024-02-11 17:20:31
```

```
console.log(dateTime.toLocaleDateString('sv-SE'));
console.log(dateTime.toLocaleTimeString('sv-SE'));
```

```
2024-02-11
17:24:51
```

The difference between Local time and UTC time can be up to 24 hours.

More info: https://www.w3schools.com/jsref/jsref_tolocalestring.asp

www.codeandcompile.com

Code </> Compile
learning made easy

Date methods

Parse: Returns milliseconds since January 1, 1970)

```
var dateTime = new Date();
dateTime = dateTime.toLocaleString('sv-SE');
console.log(Date.parse(dateTime)); 1707670993000
return msg;
```

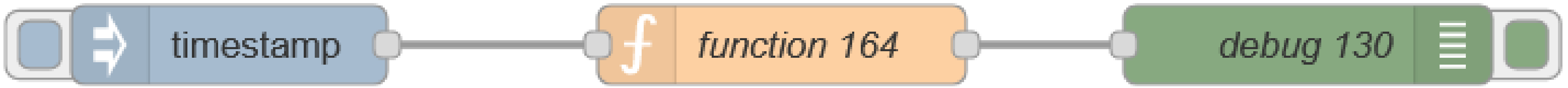
Apply methods on the result

```
var dateTime = new Date();
dateTime = dateTime.toLocaleString('sv-SE');
var milliseconds = Date.parse(dateTime);
var milliseonds_to_date = new Date(milliseconds);
console.log(milliseonds_to_date.getHours());     18
console.log(milliseonds_to_date.getMinutes());   11
console.log(milliseonds_to_date.getSeconds());   38
```

www.codeandcompile.com

Code </> Compile
learning made easy

Timestamp



```
var dateTime = new Date(msg.payload);
console.log(dateTime);
msg.payload = dateTime;
return msg;
```

msg.payload : Date

```
"Sun Feb 11 2024 18:14:53 GMT+0100 (Central
European Standard Time)"
```

Learn more: https://www.w3schools.com/js/js_dates.asp

www.codeandcompile.com

Code </> Compile
learning made easy

JS

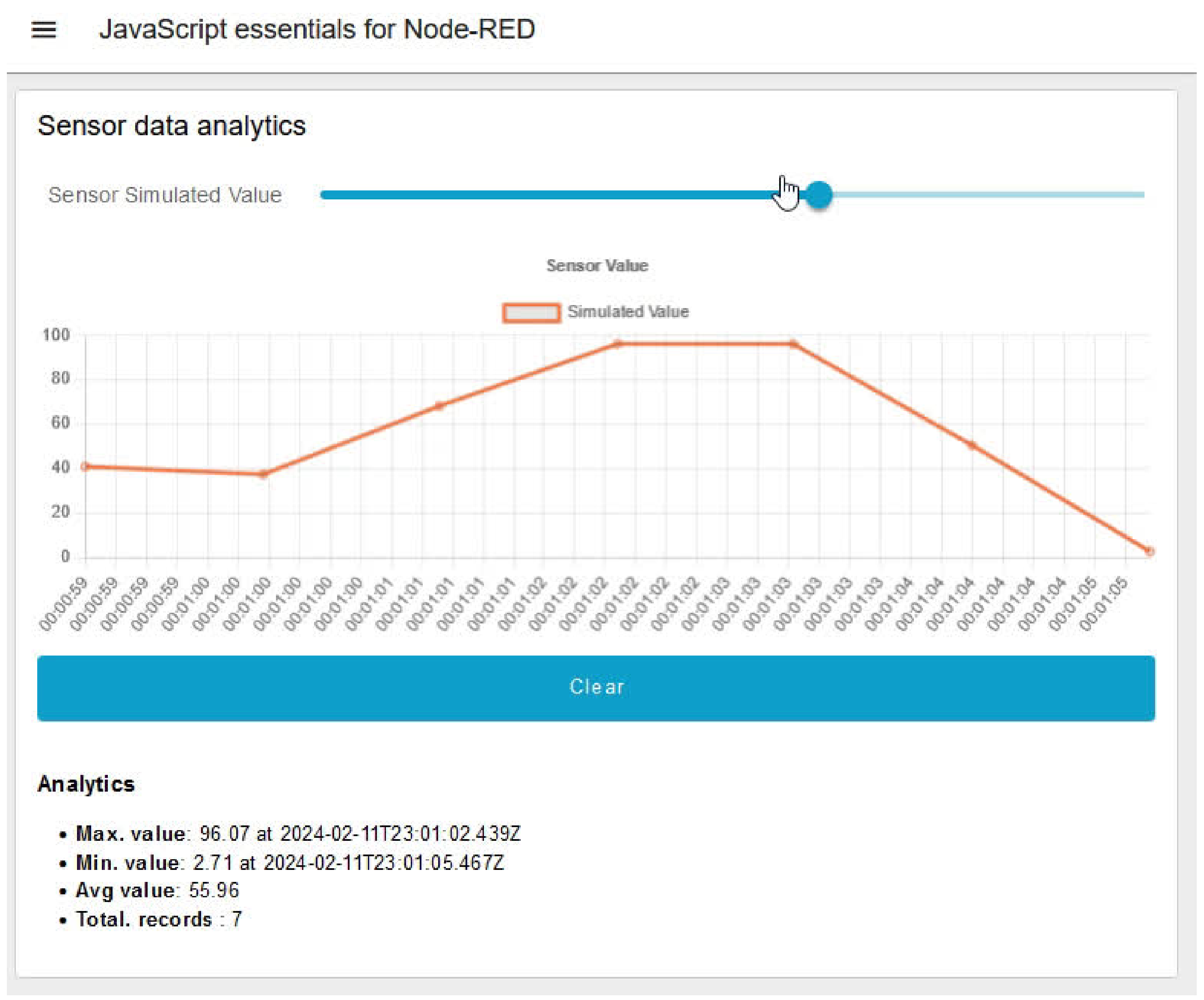
Node-RED

Exercise

Data Logging with Timestamps

- Simulates data from a sensor and logs the data when it changes with a timestamp.
- Store the result in an array of objects and show it on the chart
- Calculate the followings:
  - Maximum value with timestamp
  - Minimum value with timestamp
  - Average value
  - Total records logged

Use Math.random() to simulate sensor value between 0 and 100



Analytics

- **Max. value**: 96.07 at 2024-02-11T23:01:02.439Z
- **Min. value**: 2.71 at 2024-02-11T23:01:05.467Z
- **Avg value**: 55.96
- **Total. records** : 7

Code </> Compile

JS

Node-RED

Thank you!

www.codeandcompile.com