Code Compile

learning made easy

JavaScript essentials for Node-RED

Section 5

Functions

Code Compile
learning made easy

What you will learn in this section?

In this section, we will learn about the importance of using functions in Node-RED which is essentially reduce the amount of coding.

- Basic functions
- Parameters and arguments
- Arrow functions
- Variable scope
- Nested functions

Advantages:

Code organization and reusability

Helps to write low maintenance code

Code Compile
learning made easy

Writing functions

You can write function in the function node like shown below. This is the simplest form of writing a function.

```
1  function testFunction()
2  {
3      //content of the function
4  }
```

Calling the function

The above function can be called like this

```
6  testFunction();
```

```
1  function testFunction()
2  {
3      console.log('JavaScript essentials for Node-RED');
4  }
5
6  testFunction();
```

```
JavaScript essentials for Node-RED
```

Example: Variables in functions

```
1  function testFunction()
2  {
3      var value = 49;
4      console.log('The course price is: ' + value);
5  }
6
7  testFunction();
```

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

Code Compile
learning made easy

Assigning function to a variable

You can also assign a function to the variable. It's an alternate way of calling functions

```
1  var coursePrice = function()
2  {
3      var value = 49;
4      console.log('The course price is: ' + value);
5  }
6
7  coursePrice(); The course price is: 49
```

Naming the function

- Keep it short and descriptive
- You can use camelCase which makes it easier to read
- **randomNumber** is better than myfunc.

Code Compile
learning made easy

Exercise

Write a function (**randomNumber**) to create random numbers in an array of 10 elements and display the output in debug and console

```
[
  0.42339619782165694,
  0.5599265355462526,
  0.18019745363189954,
  0.299775830369019,
  0.1448563670456875,
  0.8045404722747431,
  0.9830817073968321,
  0.5822890565279779,
  0.4226523649578986,
  0.5977595300149618
]
```

msg.payload : array[10]

▼ array[10]

0: 0.42339619782165694
1: 0.5599265355462526
2: 0.18019745363189954
3: 0.299775830369019
4: 0.1448563670456875
5: 0.8045404722747431
6: 0.9830817073968321
7: 0.5822890565279779
8: 0.4226523649578986
9: 0.5977595300149618

Code Compile
learning made easy

Parameters and Arguments

Parameters are the variables that are **defined inside the parenthesis of the function** and **Arguments** are the values that are **passed to the function**

Function **without parameters** and calling it **without arguments**

```
1   function coursePrice()
2   {
3       console.log('The course JavaScript costs 49€');
4   }
5   coursePrice();
```

```
The course JavaScript costs 49€
```

Function **with parameters** and calling it **with arguments**

```
1   function coursePrice(name, value)
2   {
3       console.log(`The course ${name} costs ${value}€`);
4   }
5   coursePrice('JavaScript',49);
6   coursePrice('Node-RED made Easy',49);
```

```
The course JavaScript costs 49€
The course Node-RED made Easy costs 49€
```

Function has different results based on the arguments that are passed to its parameters

Code Compile
learning made easy

Parameter validation (typeof)

We do parameter validation to check it the argument passed are in the correct datatype

```
1  function coursePrice(name, value)
2  {
3      if ((typeof name === 'string') && (typeof value === 'number'))
4      {console.log(`The course ${name} costs ${value}€`);}
5      else
6      {console.log("The input is not valid!")}
7  }
8  coursePrice('JavaScript',49);
9  coursePrice(49,'JavaScript');
```

Code Compile
learning made easy

Parameter validation (no or incomplete arguments)

```
1   function coursePrice(name, value)
2   {
3       {console.log(`The course ${name} costs ${value}€`);}
4   }
5   coursePrice();
```

```
The course undefined costs undefined€
```

```
1   function coursePrice(name, value)
2   {
3       {console.log(`The course ${name} costs ${value}€`);}
4   }
5   coursePrice('JavaScript');
6   coursePrice(49);
```

```
The course JavaScript costs undefined€
The course 49 costs undefined€
```

Code Compile
learning made easy

Default parameters

Default parameters are used to assign values to the function parameters when no arguments are passed to that parameter

```
1   function coursePrice(name = 'JavaScript', value = 49)
2   {
3       {console.log(`The course ${name} costs ${value}€`);}
4   }
5   coursePrice();
```

```
The course JavaScript costs 49€
```

Code Compile
learning made easy

Project 1

Task 1

Write a function to define an array of random elements (between 0 ~ 100). The number of elements and their precision should be passed by an argument

```
randomNumber(10,2);
```

msg.payload : array[10]

[47.97, 75.51, 34.37, 38.64, 10.47, 19.69, 10.64, 45.21, 19.81, 95.03]

```
randomNumber(5,4);
```

msg.payload : array[5]

[22.1199, 58.5081, 55.1456, 12.6288, 7.7982]

Task 2

If no arguments are passed the output should be like the following

```
randomNumber();
```

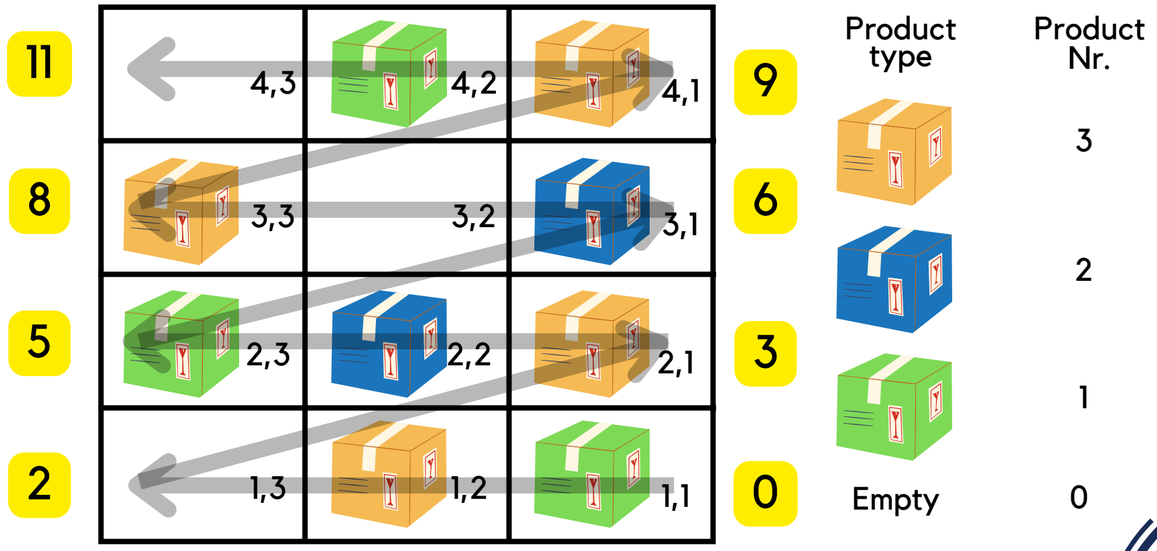
msg.payload : array[10]

[47.3, 49.9, 21.3, 79.8, 5.7, 36, 89.3, 50.9, 52.2, 71.6]

Code Compile
learning made easy

Project 2

- Call a function to initialize a single dimension array of 12 elements with a **random number** (0~3) using FOR loop such that every number represents the product type.
- Call a function to calculate total number of ‘type’ (argument) boxes or empty location.
- Call a function to store the box (argument) in the nearest empty location. Throw an error if there is no empty location
- Call a function to retrieve the box from location ‘pos1’ (argument) and store it into the empty location ‘pos2’ (argument). If location is not empty throw an error.



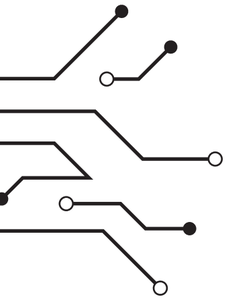
Code Compile
learning made easy

Arrow functions =>

Another way of writing functions

```
1  var addNumbers = (x,y) => {msg.payload = x + y;}
2  addNumbers(3,5);
3  console.log(msg.payload);
4  return msg;
```

Body of the function



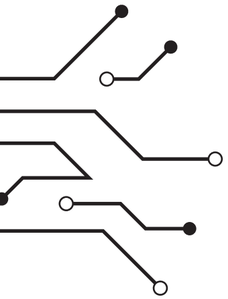
Code </> Compile
learning made easy

Arrow functions =>

Defining random values in the array using arrow function

```
1   var arr = [];
2   var randomNumbers = (x,y) =>
3   {
4       for (let index = 0; index < x; index++) {
5           arr.push(Math.floor(Math.random() * y));
6       }
7   }
8   randomNumbers(10,3);
9   console.log(arr);
10  msg.payload = arr;
11  return msg;
```

```
[
  2, 1, 2, 0, 2,
  0, 0, 2, 2, 2
]
```



Code Compile
learning made easy

Returning function values

We can also let the function **returns a value** when they are called. In this case, we have to **mention the return value**. The return value **can be stored in a variable**.

```
1  function addNumbers(x,y)
2  {
3      return x + y;
4  }
5  msg.payload = addNumbers(3,4);
6  return msg;
```

```
msg.payload : number
7
```

```
1  var arr = [];
2  function randomNumbers(x,y)
3  {
4      for (let index = 0; index < x; index++) {
5          arr.push(Math.floor(Math.random() * y));
6      }
7      return arr;
8  }
9
10 msg.payload = randomNumbers(10,3);
11 return msg;
```

```
msg.payload : array[10]
▸ [ 2, 0, 1, 0, 2, 1, 2, 0, 2, 1 ]
```

Code Compile
learning made easy

Variable scope in function

Variables that are defined inside the function cannot be accessed outside the function

```
1  function addNumbers(x,y)
2  {
3      let z = x + y;
4      console.log(z);
5  }
6  addNumbers(3,4);
7  console.log(z);
8  return msg;
```

```
1  function addNumbers(x,y)
2  {
3      var z = x + y;
4      console.log(z);
5  }
6  addNumbers(3,4);
7  console.log(z);
8  return msg;
```

```
7
4 Feb 21:20:38 - [error] [function:function 132] ReferenceError: z is not defined (line 7, col 13)
```

Code Compile
learning made easy

Variable scope in function

Variables that are defined inside the function cannot be accessed outside the function

```
1   function addNumbers(x,y)
2   {
3       return  x + y;
4   }
5   msg.payload = addNumbers(3,4);
6   console.log(addNumbers(3,4));
7   return msg;
```

msg.payload : number

7

7

```
1   var z = 0;
2   function addNumbers(x,y)
3   {
4       z = x + y;
5       console.log(z);
6   }
7   addNumbers(3,4);
8   console.log(z);
9   msg.payload = z;
10  return msg;
```

msg.payload : number

7

7

Code Compile
learning made easy

‘let’ vs ‘var’ in function scope

‘**let**’ has **block scope** where as ‘**var**’ has **function scope**

```
1  function addNumbers(x,y)
2  {
3      if (typeof x === 'number' && typeof y === 'number')
4      {
5          let z = x + y;
6          console.log(z);
7      }
8
9  }
10 addNumbers(3,4);
```

7

```
1  function addNumbers(x,y)
2  {
3      if (typeof x === 'number' && typeof y === 'number')
4      {
5          let z = x + y;
6
7      }
8      console.log(z);
9  }
10 addNumbers(3,4);
```

function : (error)

"ReferenceError: z is not defined"

www.codeandcompile.com

Code Compile
learning made easy

‘let’ vs ‘var’ in function scope

‘**let**’ has **block scope** where as ‘**var**’ has **function scope**

```
1  function addNumbers(x,y)
2  {
3      if (typeof x === 'number' && typeof y === 'number')
4      {
5          var z = x + y;
6          console.log(z);
7      }
8
9  }
10 addNumbers(3,4);
```

7

const variable also has block scope

```
1  function addNumbers(x,y)
2  {
3      if (typeof x === 'number' && typeof y === 'number')
4      {
5          var z = x + y;
6
7      }
8      console.log(z);
9  }
10 addNumbers(3,4);
```

7

Code Compile
learning made easy

Nested functions

When we have a **function inside another function, we call this phenomena nested functions**.

Just like we understood nested loops before.

The nested function can be called only inside the outer function. Outside the outer function it is out of scope

```
1   function randomNumbers(x,y)
2   {
3       var arr = []
4       for (let index = 0; index < x; index++)
5       {
6           arr.push(Math.floor(Math.random()*y));
7       }
8
9       function displayArray(array)
10      {
11          console.log(array);
12      }
13      displayArray(arr);
14
15  }
16  randomNumbers(10,11);
```

```
[
  6, 2,  2,  8, 6,
  8, 9, 10, 10, 1
]
```

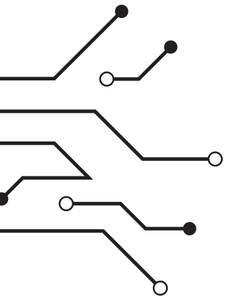
Code Compile
learning made easy

Anonymous functions

These are the **functions without name.** We usually **store these functions in the variables**. These are used for **passing a function as an argument to another function**

```
1   var anonymous = function ()
2   {
3       console.log('Anonymous for the voiceless')
4   }
5   anonymous();
```

```
Anonymous for the voiceless
```



Code Compile
learning made easy

Function Callback

A callback function is like asking a friend to do a task, and then saying, "When you're finished, give me a call back with the results."

In programming, especially in JavaScript, this concept lets you run a piece of code right after a certain task is completed.

```
1   //named function
2   function getRandomValue(callback) {
3       var random = Math.random();
4       callback(random);
5   }
6
7   //anonymous function
8   var display = function(value) {
9       console.log('The value is: ' + value);
10  }
11
12  getRandomValue(display);
```

```
The value is:0.8228417561796368
```

We will see another example in **Section 6** when we talk about **reduce function**

Code Compile
learning made easy

Project 3

Task 1: Array analyzer
Write a function that takes array number and returns an object with the count of positive numbers, negative numbers, and the average of all numbers.

Task 2: Sensor data converter
Write two functions that converts sensor data from one unit to another:

- Temperature from Celsius to Fahrenheit
- Distance from millimeters to inches

Code Compile
learning made easy

Project 3

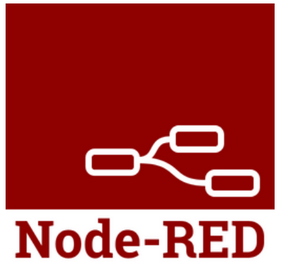
Task 3: OEE Calculator
Write several functions that calculates the OEE of a machine.
Reference: https://www.oee.com/calculating-oee/

Availability = Run Time / Planned Production Time
Run Time = Planned Production Time – Stop Time
Performance = (Ideal Cycle Time × Total Count) / Run Time
Quality = Good Count / Total Count
OEE = Availability × Performance × Quality

OEE = Availability × Performance × Quality

Can you show that on the dashboard with UI elements?

Code </> Compile



Thank you!

www.codeandcompile.com