Function Hoisting

Hoisting is a JavaScript technique which moves variables and function declarations to the top of their scope before code execution begins. Within a scope no matter where functions or variables are declared, they're moved to the top of their scope.

One of the advantages of JavaScript putting function declarations into memory before it executes any code segment is that it allows you to use a function before you declare it in your code. For example:

```
function catName(name) {
  console.log("My cat's name is " + name);
}

catName("Tiger");

/*
The result of the code above is: "My cat's name is Tiger"
*/
```

The above code snippet is how you would expect to write the code for it to work. Now, let's see what happens when we call the function before we write it:

```
catName("Chloe");

function catName(name) {
  console.log("My cat's name is " + name);
}
/*
The result of the code above is: "My cat's name is Chloe"
*/
```

Even though we call the function in our code first, before the function is written, the code still works. This is because of how context execution works in JavaScript.

Hoisting works well with other data types and variables. The variables can be initialized and used before they are declared.

Only declarations are hoisted

JavaScript only hoists declarations, not initializations. If a variable is declared and initialized after using it, the value will be undefined. For example:

```
console.log(num); // Returns undefined, as only declaration was hoisted, no initialization has happened at this stage

var num; // Declaration

num = 6; // Initialization
```

The example below only has initialization. No hoisting happens so trying to read the variable results in ReferenceError exception.

```
console.log(num); // Throws ReferenceError exception
num = 6; // Initialization
```

Initializations using let and const are also not hoisted.

```
// Example with let:
a = 1; // initialization.
let a; // Throws ReferenceError: Cannot access 'a' before initialization

// Example with const:
a = 1; // initialization.
const a; // Throws SyntaxError: Missing initializer in const declaration
```

Below are more examples demonstrating hoisting.

```
// Example 1
// Only y is hoisted

x = 1; // Initialize x, and if not already declared, declare it - but no hoisting as there is no var in the statement.
console.log(x + " " + y); // '1 undefined'
// This prints value of y as undefined as JavaScript only hoists declarations
var y = 2; // Declare and Initialize y

// Example 2
// No hoisting, but since initialization also causes declaration (if not already declared), variables are available.

a = 'Cran'; // Initialize a
b = 'berry'; // Initialize b

console.log(a + "" + b); // 'Cranberry'
```

Function within Function

Here the task is to create nested functions, JavaScript support nested functions. In the examples given below the output returning is combination of the output from the outer as well as inner function(nested function).
Approach:

- Write one function inside another function.
- Make a call to the inner function in the return statement of the outer function.
- Call it **fun(a)(b)** where a is parameter to outer and b is to the inner function.
- Finally return the combined output from the nested function.

Example

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>
      Nested functions in JavaScript.
    </title>
  </head>
  <body id = "body" style = "text-align:center;">
    <h1 style = "color:green;" >
      Function in Function
    </h1>
    <p id = "BUTTON_UP" style =
      "font-size: 15px; font-weight: bold;">
    </p>
    <button onclick = "BUTTON_Fun()">
    click here
    </button>
    <p id = "BUTTON_DOWN"
      style = "font-size: 24px;
           font-weight: bold;
           color: green;">
    </p>
 <script>
       var up = document.getElementById('BUTTON_UP');
       var down =
document.getElementById('BUTTON_DOWN');
       up.innerHTML =
         "Click on the button to call nested function.";
       function fun1(a) {
       function fun2(b) {
         return a + b;
       }
       return fun2;
       }
       function BUTTON_Fun() {
         down.innerHTML =
            fun1("A Online Computer Science Portal: ")
            (" Function in Function");
       }
     </script>
   </body>
</html>
```

Function Expressions

The function keyword can be used to define a function inside an expression.

You can also define functions using the Function constructor and a function declaration.

```
const getRectArea = function(width, height)
{
 return width * height;
};
console.log(getRectArea(3, 4));
```

```
<!DOCTYPE HTML>
<html>
   <head>
      <title>
         Function Expression
      </title>
   </head>
   <body id = "body" style = "text-align:center;">
      <h1 style = "color:green;" >
         Area of the rectangle
      </h1>
      <p id = "BUTTON_UP" style =
         "font-size: 15px; font-weight: bold;">
      </p>
      <button onclick = "BUTTON_Fun()">
      click here
      </button>
      <p id = "BUTTON_DOWN"
        style = "font-size: 24px;
              font-weight: bold;
              color: green;">
      </p>
```

```
 <script>
          var up = document.getElementById('BUTTON_UP');
          var down =
document.getElementById('BUTTON_DOWN');
          up.innerHTML =
            "Click on the button to call function expression.";

const getRectArea = function(width, height) {
  return width * height;
};
         function BUTTON_Fun() {
             down.innerHTML =
                  getRectArea(3, 4);

          }
       </script>
    </body>
</html>
```

Passing function as arguments

```
function functionName(parameter1, parameter2, parameter3)
{
  // code to be executed
}
```

Function parameters are the names listed in the function definition.

Function arguments are the real values passed to (and received by) the function.

Parameter Rules

JavaScript function definitions do not specify data types for parameters.

JavaScript functions do not perform type checking on the passed arguments.

JavaScript functions do not check the number of arguments received.

```
x = sumAll(1, 123, 500, 115, 44, 88);

function sumAll() {
  var i;
  var sum = 0;
  for (i = 0; i < arguments.length; i++) {
    sum += arguments[i];
  }
  return sum;
}
```