upGrad

#RahoAmbitious

Full Stack Software Development

upGrad

Course: JavaScript and Server-Side Communication

Lecture On: JavaScript Scope, Closure and Hoisting

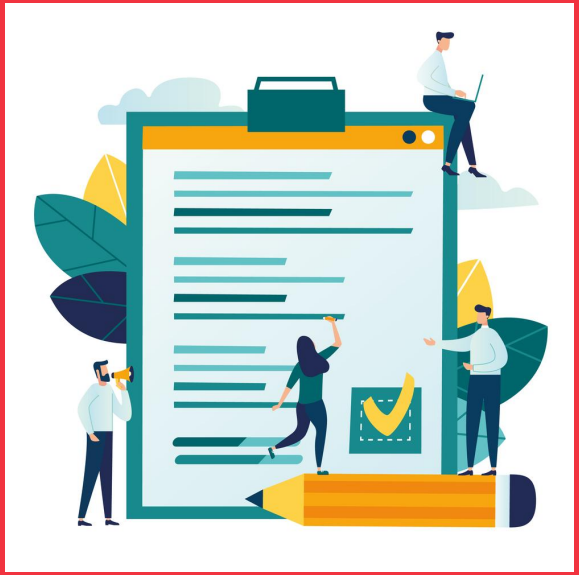
Instructor: Siddhesh Prabhugaonkar

In the previous class, we discussed…

- Arrays and array methods of JS
- Objects and functions in JS

Today's Agenda

- Lifetime or scope of a code block
- Closures of JavaScript
- Hoisting in JavaScript



JavaScript Scope

JavaScript Scope

upGrad

- JavaScript (JS) scope defines the accessibility of the variables.
- The two types of scopes in JavaScript are **local scope** and **global scope**.
- Variables declared within a JS function become local to that function. They can be accessed and used only inside that function; they cannot be accessed outside. Local variables are created when a function starts and are deleted when a function is completed.
- Variables that are declared outside a function become global variables. These variables will be accessible by all the functions and scripts in the web page.

```
 //a cannot be accessed here

function functionName () {
   var a = 50;
   //a can be accessed here
}

//a cannot be accessed here
```

Local Scope

```
var a = 50;

function functionName () {
    //a can be accessed here
}

//a can be accessed here
```

Global Scope

Points to Remember

1. The scope chain works from inwards to outwards. This means that a variable is looked for inside the local scope first. If the variable is found in the local scope, then it is considered declared. In case the local scope does not contain the variable that is searched for, the control goes to the global scope to look for this variable. If the variable is found in the global scope, then it is considered. If it is not found in the global scope either, then it means that the variable is not defined anywhere, and thus, you get an error stating that the given variable is ‘not defined’.

```
var x = 10;
function foo() {
    console.log(x);
}
foo(); //10
```

Points to Remember

2. If one function is called inside another function, but both the functions are defined separately in the global scope, then a variable defined in one function cannot be referenced inside another function. This is because in such a case, both the functions have different local scopes, as they are defined separately in the global scope.

```
function func1() {
    var i = 20;
    func2();
}
undefined
function func2() {
    console.log(i);
}
undefined
func1()
/*
Uncaught ReferenceError: i is not defined
    at func2 (<anonymous>:2:17)
    at func1 (<anonymous>:3:5)
    at <anonymous>:1:1
*/
```

Lexical Scope

- Lexical scope refers to the fact that when one function, which is referred to as the parent function, contains another function, which is referred to as the child function, the child function can access everything defined inside the parent function.

- In the example code given on the right side:
  - *grandparent()* function can access the variable - *g_age*
  - *parent()* function can access the variables - *g_age*, *p_age*
  - *child()* function can access the variables - *g_age*, *p_age*, *c_age*

- Lexical scope is also known as the static scope. Remember that in lexical scope, only a child function can access its parents' resources, but a parent function cannot access its children's resources.

```
function grandparent(){
    var g_age = 75;
    parent();

    function parent() {
        var p_age = 50;
        child();

        function child(){
            var c_age = 25;
            console.log(c_age, p_age, g_age);
        }
    }
}

grandparent(); // 25 50 75
```

Poll 1 (15 Sec)

What will be the output of the code snippet on the console given in the adjoining screenshot?

1. 20
2. 40
3. undefined
4. Uncaught ReferenceError: y is not defined

```
var x = 20;
function foo() {
   console.log(x);
}
x = 40;
foo();
```

Poll 1 (Answer)

What will be the output of the code snippet on the console given in the adjoining screenshot?

1. 20
2. **40**
3. undefined
4. Uncaught ReferenceError: y is not defined

```
var x = 20;
function foo() {
   console.log(x);
}
x = 40;
foo();
```

Poll 2 (15 Sec)

What will be the output of the code snippet given in the adjoining screenshot?

1. 390
2. 750
3. Undefined
4. Uncaught ReferenceError: x is not defined

```
function foo() {
   var x = 390;
   bar();
   x = 750;
}
function bar() {
   console.log(x);
}
foo();
```

Poll 2 (Answer)

What will be the output of the code snippet on the console given in the adjoining screenshot?

1. 390
2. 750
3. Undefined
4. **Uncaught ReferenceError: x is not defined**

```
function foo() {
   var x = 390;
   bar();
   x = 750;
}
function bar() {
   console.log(x);
}
foo();
```

JavaScript Closure

JavaScript Closure

upGrad

- JavaScript (JS) closure is a function that has access to the parent scope even after the parent function has closed.
- For instance, in the code given below, the variable *add* is assigned the value of a function. Notice that the function is anonymous as well as an IIFE (Immediately-Invoked Function Expression) and hence the final result after the function is invoked is returned back to the variable *add*.
- Inside the anonymous function, **var counter=0;** is invoked only once. It sets the counter to zero and then returns a function expression.
- Every time the counter increases, it updates the current value.
- Hence, at the end of three add() methods, the counter value becomes 3.
- Notice that the last value of the variable *counter* is accessible even after the function ends. This is called closure. The variable counter is accessible even outside the function and its value is preserved.

```
var add = (function() {
   var counter = 0;
   return function() {
      counter += 1;
      return counter;
   }
})();

add(); //1
add(); //2
add(); //3
```

Poll 3 (15 Sec)

What will be the output of the code snippet in the adjoining screenshot?

1. employee0
   employee1
   employee2
   employee3
   employee4

2. Uncaught
   prefix is not defined

```
function createEmployeeID() {
      var prefix = "employee";
      var generateID = function() {
            for (var i = 0; i < 5; i++) {
                  console.log(prefix + i);
            }
      }
      return generateID;
}
var print = createEmployeeID();
print();
```

Poll 3 (Answer)

What will be the output of the code snippet in the adjoining screenshot?

1. **employee0**
   employee1
   employee2
   employee3
   employee4

2. Uncaught ReferenceError: prefix is not defined

```
function createEmployeeID() {
      var prefix = "employee";
      var generateID = function() {
            for (var i = 0; i < 5; i++) {
                  console.log(prefix + i);
            }
      }
      return generateID;
}
var print = createEmployeeID();
print();
```

JavaScript Hoisting

JavaScript Hoisting

upGrad

- Hoisting is the default behaviour of JS to move all the declarations to the top.
- For instance, in the example given below, even if variable declaration is done after initialisation, the variable still gets the value. This is because by default, all the declarations are moved to the top of current scope (the current function scope or the current script scope).

```
number = 1;
console.log(number); // 1
var number;
```

- However, you need to understand that only the declaration is moved to the top of the current scope but not the initialisation.
- For instance, in the code given below, the declaration of the variable *number2* is moved to the top but not the value. Hence, the value of the variable *number2* is *undefined*.

```
number1 = 1;
console.log(number1, number2); //number1 undefined
var number2 = 2;
```

JavaScript Hoisting

upGrad

```
function add(x, y) {
    console.log(x+y);
 }
 add(1, 6); //7
```

Here, it is straightforward that the output will be 7.

```
add(1, 6);
function add(x, y) {
   console.log(x+y);
}
```

Notice that here, you are calling a function before declaring and defining it.

Again, the output will be 7 because similar to variable declarations, JavaScript hoists function declarations at the top of the code as well. Thus, you can call any function that has been declared later.

JavaScript Hoisting

upGrad

```
var add;
add(1, 6);
```

Notice that add is not declared as a function anywhere.

Thus, the output will be as follows:
Uncaught TypeError: add is not a function

The variable *add* contains the default value *undefined*, and so, the value *undefined* cannot be considered as a function.

```
add(1, 6);

var add = function(x, y) {
   console.log(x + y);
};
```

Here, the output will be as follows:
Uncaught TypeError: add is not a function

This happens because *add* is a variable here, and you are assigning a function to this variable. Recall that only the declaration is moved to the top of the current scope and not the initialisation. So, the variable *add* is declared but it is not assigned the value as a function. So, the variable add is undefined. Now, you cannot consider an undefined variable as a function. If you do, you get the same result as above.

Poll 4 (15 Sec)

Which of the following options displays the correct output of the code snippet given below?

1. Uncaught ReferenceError: x is not defined
2. 0
3. undefined
4. 10

```
console.log(x);
var x;
x = 10;
```

Poll 4 (Answer)

Which of the following options displays the correct output of the code snippet given below?

1. Uncaught ReferenceError: x is not defined
2. 0
3. **undefined**
4. 10

```
console.log(x);
var x;
x = 10;
```

Hands-On Exercise (3 mins)

upGrad

Write a constructor **Book** which accepts three parameters as input: the name of the book, the author of the book and the year in which the book was published.

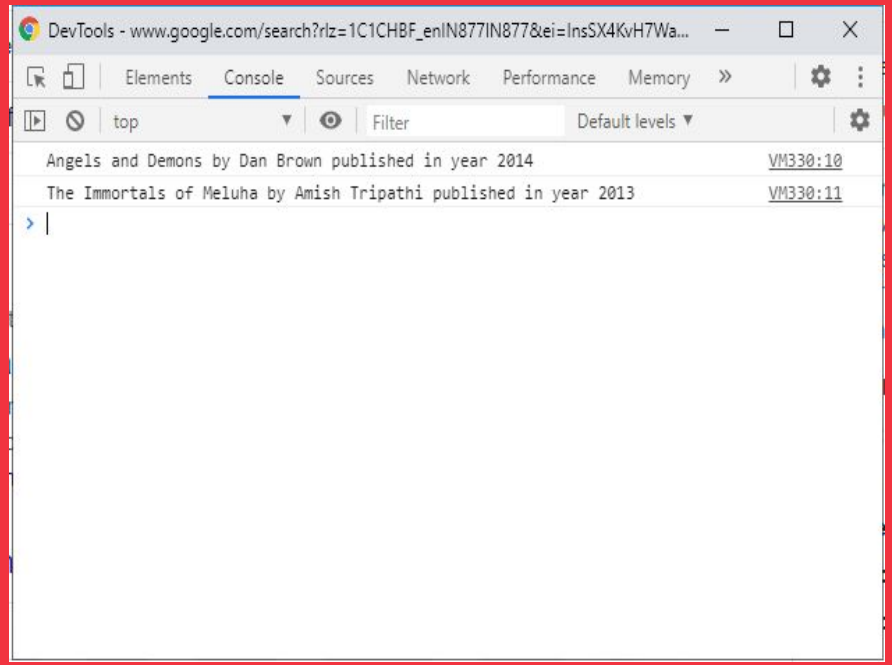
When the Book constructor is called by new keyword, it should assign the the received parameters to the name, author and year properties of the current instance.

The book details are given as:

- Angels and Demons, Dan Brown, 2000
- The Immortals of Meluha, Amish Tripathi, 2010

You should then be able to access the details of a particular book using dot notation.

The output should be as shown in the image alongside.



The stub code is provided here.

The solution is provided here.

Tasks to complete after today's session

MCQs
Coding Questions

Key Takeaways

- The functions in JS are blocks of code that execute a particular task, such as adding two numbers. These functions may or may not return a value.
- A variable declared inside a function cannot be accessed outside it. This phenomenon is called local scope. On the other hand, a variable declared outside the function can be accessed by all the functions in the web page; hence, it has a global scope.
- By default, JS moves all the declarations to the top. This is called hoisting.

In the next class, we will discuss...

- bind(), call() and apply() methods
- DOM Manipulation using JavaScript

upGrad

#RahoAmbitious

Thank you!