Telerik Academy

Scopes and Closures

Things start to get serious

JavaScript OOP

Telerik Software Academy

http://academy.telerik.com

Telerik Academy

Table of Contents

- Scopes
  - Block and function scope
  - References availability
  - Resolving references through the scope chain
  - Alternatives in ES6
- Closures
  - What is a Closure?
  - How to deal with closures?
  - Simple modules

Telerik Academy

Scope

Scope

- Scope is a place where variables are defined and can be accessed
- JavaScript has only two types of scopes
  - Global scope and function scope
    - Global scope is the same for the whole web page
    - Function scope is different for every function
  - Everything outside of a function scope is inside of the global scope

```
if(true){
    var sum = 1+2;
}
console.log(sum);
```

Scope

- Scope is a place where variables are defined and can be accessed
- JavaScript has only two types of scopes
  - Global scope and function scope
    - Global scope is the same for the whole web page
    - Function scope is different for every function
  - Everything outside of a function scope is inside of the global scope

```
if(true){
    var sum = 1+2;
}
console.log(sum);
```

The scope of the if is the global scope. sum is accessible from everywhere

Global Scope

- The global scope is the scope of the web page
  - Or the Node.js app
- Objects belong to the global scope if:
  - They are define outside of a function scope
  - They are defined without var
    - Fixable with `'use strict'`

```
function arrJoin(arr, separator) {
  separator = separator || "";
  arr = arr || [];
  arrString = "";
  for (var i = 0; i < arr.length; i += 1) {
    arrString += arr[i];
    if (i < arr.length - 1) arrString += separator;
  }
  return arrString;
}
```

Global Scope

- The global scope is the scope of the web page
  - Or the Node.js app
- Objects belong to the global scope if:
  - They are define outside of a function scope
  - They are defined without var
    - Fixable with `'use strict'`

```
function arrJoin(arr, separator) {
  separator = separator || "";
  arr = arr || [];
  arrString = "";
  for (var i = 0; i < arr.length; i += 1) {
    arrString += arr[i];
    if (i < arr.length - 1) arrString += separator;
  }
  return arrString;
}
```

arr, separator and i belong to the scope of printArr

Global Scope

- The global scope is the scope of the web page
  - Or the Node.js app
- Objects belong to the global scope if:
  - They are define outside of a function scope
  - They are defined without var
    - Fixable with `'use strict'`

```
function arrJoin(arr, separator) {
  separator = separator || "";
  arr = arr || [];
  arrString = "";
  for (var i = 0; i < arr.length; i += 1) {
    arrString += arr[i];
    if (i < arr.length - 1) arrString += separator;
  }
  return arrString;
}
```

arr, separator and i belong to the scope of printArr

arrString and arrJoin belong to the global scope

Global Scope (2)

- The global scope is one of the very worst parts of JavaScript
  - Every object pollutes the global scope, making itself more visible
  - If two objects with the same identifier appear, the first one will be overridden

Telerik Academy

Global Scope

Live Demo

Function Scope

- JavaScript does not have a block scope like other programming languages (C#, Java, C++)
  - { and } does not create a scope!
- Yet, JavaScript has a function scope
  - Function expressions create scope
  - Function declarations create scope

```
if(true)var result = 5;
console.log(result);//logs 5
```

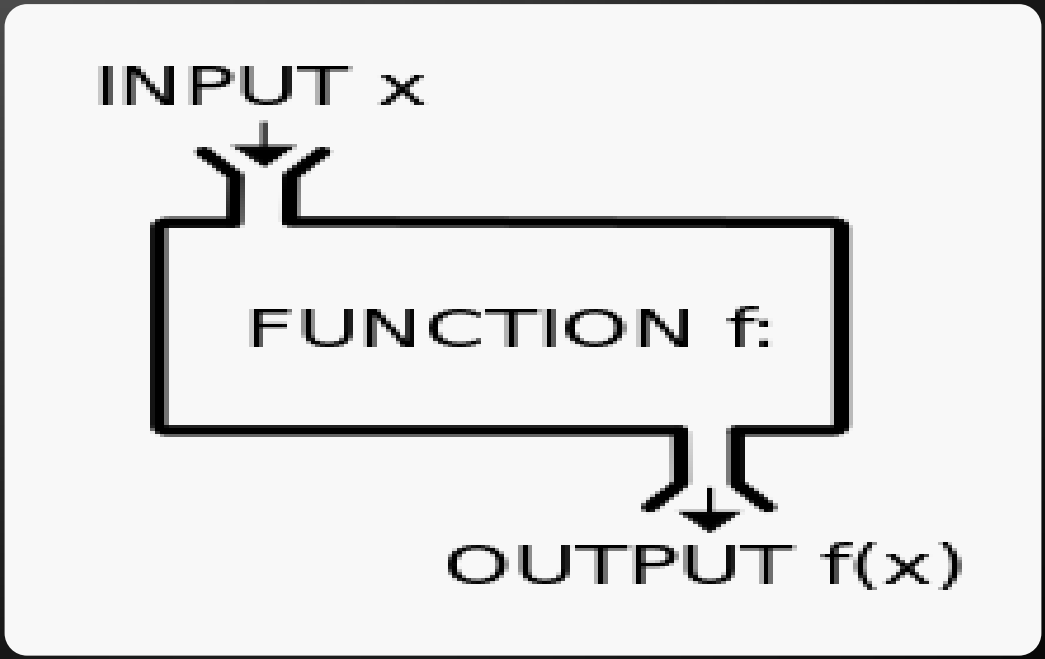
```
if(true) (function(){ var result = 5;})();
console.log(result);//ReferenceError
```

```
function logResult(){ var result = 5; }
if(true) logResult();
console.log(result); //ReferenceError
```

Telerik Academy

Function Scope

Live Demo



Telerik Academy

Resolving References through the Scope Chain

- JavaScript resolves the object references due to the simple rule "Closer is better":
  - if a function `outer()` declares object x, and its nested function `inner()` declares object x:
    - outer() holds a reference to the outer x
    - inner() holds a reference to the inner x

```
function outer(){
  var x = 'OUTER';
  function inner(){
    var x = 'INNER';
    return x;
  }
  inner();
  return { x: x, f: inner };
}
```

Telerik Academy

Resolving References through the Scope Chain

Live Demo

ECMAScript 6 Way of Working with Scopes

- ECMAScript 6 introduces a new way to handle scopes:
  - The key word 'let'
- let is much like var
  - Creates a variable

```
if(false){
  var x = 5;
  let y = 6;
}
console.log(x); //prints undefined
console.log(y); //throws error
```

- But, let creates a block scope
- Yet, still not supported
  - Can be used with Babel.js or Traceur

Telerik Academy

Block scope with let

Live Demo

Telerik Academy

Closures

Closures

- Closures are a special kind of structure
  - They combine a function and the context of this function

```
function outer(x){
  function inner(y){
    return x + " " + y;
  }
  return inner;
}
```

Closures

- Closures are a special kind of structure
  - They combine a function and the context of this function

```
function outer(x){
  function inner(y){
    return x + " " + y;
  }
  return inner;
}
```

inner() forms a closure.
It holds a reference to x

Closures

- Closures are a special kind of structure
  - They combine a function and the context of this function

```
function outer(x){
  function inner(y){
    return x + " " + y;
  }
  return inner;
}
```

inner() forms a closure.
It holds a reference to x

In the context of f1,
x has value 5

```
var f1 = outer(5);
console.log(f1(7)); //outputs 5 7
```

Closures

- Closures are a special kind of structure
  - They combine a function and the context of this function

```
function outer(x){
  function inner(y){
    return x + " " + y;
  }
  return inner;
}
```

inner() forms a closure.
It holds a reference to x

In the context of f1,
x has value 5

```
var f1 = outer(5);
console.log(f1(7)); //outputs 5 7
```

In the context of f2,
x has value "Peter"

```
var f2 = outer("Peter");
console.log(f2("Petrov"));  //outputs Peter Petrov
```

Telerik Academy

Simple Closures

Live Demo

Closures Usage

- Closures can be used for data hiding
  - Make objects invisible to the outside
    - Make them private

```
var school = (function() {
    var students = [];
    var teachers = [];

    function addStudent(name, grade) {...}
    function addTeacher(name, speciality) {...}
    function getTeachers(speciality) {...}
    function getStudents(grade) {...}

    return {
        addStudent: addStudent,
        addTeacher: addTeacher,
        getTeachers: getTeachers,
        getStudents: getStudents
    };
})();
```

Closures Usage

- Closures can be used for data hiding
  - Make objects invisible to the outside
    - Make them private

```
var school = (function() {
    var students = [];
    var teachers = [];

    function addStudent(name, grade) {...}
    function addTeacher(name, speciality) {...}
    function getTeachers(speciality) {...}
    function getStudents(grade) {...}

    return {
        addStudent: addStudent,
        addTeacher: addTeacher,
        getTeachers: getTeachers,
        getStudents: getStudents
    };
})();
```

This is actually called a Module

Telerik Academy

Closures

Live Demo

Telerik Academy

Scopes and Closures

Questions?

http://academy.telerik.com