Advanced Topics in Internet Application Development

Class 2 – Javascript (Cont')

Function Properties & Methods

- length
- prototype
- apply()
- call()

Function Length Property

```
function sayName(name){ alert(name); }

function sum(num1, num2){ return num1 + num2; }

function sayHi() { alert("hi"); }

alert( sayName.length );  //1
alert( sum.length );      //2
alert( sayHi.length );    //0
```

Call()

```
function sum(num1, num2) {
    return num1 + num2;
}

function callSum(num1, num2) {
    return sum.call(this, num1, num2);
}

alert(callSum(10, 10)); //20
```

Call() Demo

```
window.color = "red";
var o = { color: "blue" };

function sayColor(){
    alert(this.color);
}

sayColor();               //red
sayColor.call(this);      //red
sayColor.call(window);    //red
sayColor.call(o);         //blue
```

Apply()

```
function sum(num1, num2) {
    return num1 + num2;
}

function callSum1(num1, num2) {
    return sum.apply(this, arguments);
}

function callSum2(num1, num2) {
    return sum.apply(this, [num1, num2]);
}

alert(callSum1(10, 10)); //20
alert(callSum2(10, 10)); //20
```

Function Declarations vs. Function Expressions

- Function declarations are read and available in an execution context **before** any code is executed.
- Function expressions **aren’t complete** until the execution reaches that line of code.

```
alert( sum(10, 10) );
function sum(num1, num2) {
    return num1 + num2;
}
```

function declaration

```
alert(sum(10, 10));
[redacted](num1, num2) {
    return num1 + num2;
};
```

function expression,
unexpected identifier error.

Reference Types

COLMAN CS 2015 B

DR. SHAY HOROVITZ

Agenda

- Working with objects.
- Arrays.
- JavaScript date types.
- Primitives & primitive wrappers.

Reference Types

- A reference value (object) is an instance of a specific reference type.

- ECMAScript provides a number of native reference types, such as Object.

```
var person = new Object();

person.name = "Nimrod";
person.age = 29;
```

Reference Types

The Object Type

- There are **two** ways to explicitly create an **instance** of Object.
  - **new** operator
  - Object literal

```
var person = {};

//same as
var person = new Object();
```

```
var person = {
    name : "Nicholas",
    age : 29
};
```

Array

Array Type

```
//create an array with three items
var colors = new Array(3); // Equal to Array(3);
//create an array with one item, the string “Greg”
var names  = new Array("Greg");


//creates an array with three strings
var colors = ["red", "blue", "green"];
//creates an empty array
var names  = [];


//creates an array with three strings
var colors = ["red", "blue", "green"];
colors[colors.length] = "black"; //add a color (position 3)
colors[colors.length] = "brown"; //add a color (position 4)
```

Stack Methods

- An array object can act just like a stack (**LIFO**)

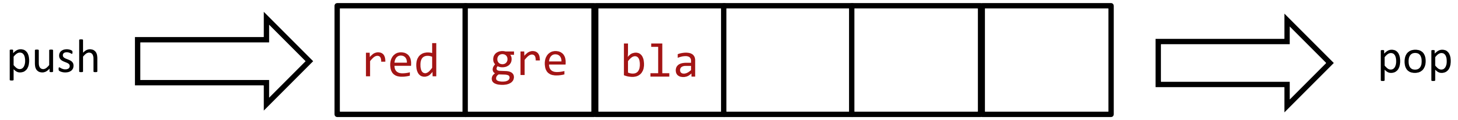
```
var colors = new Array();

var count = colors.push("red", "green");
alert(count); //2

count = colors.push("black");
alert(count); //3

var item = colors.pop(); //get the last item
alert(item); //”black”

alert(colors.length); //2
```

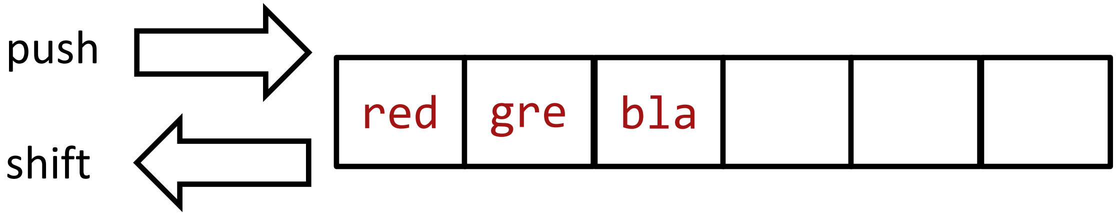


Queue Methods (FIFO)

```
var colors = new Array();
var count = colors.push("red", "green");
alert(count);             //2

count = colors.push("black");
alert(count);             //3

var item = colors.shift();
alert(item);              //”red”
alert(colors.length);     //2
```



push

shift

red gre bla

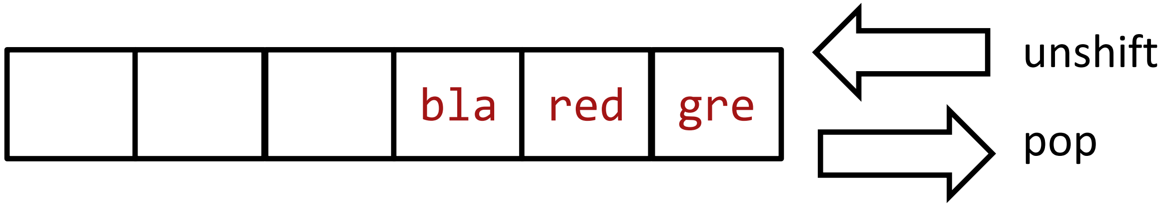
unshift() and pop()

- It adds any number of items to the front of an array and returns the new array length.

```
var colors = new Array();
var count = colors.unshift("red", "green");
alert(count);                 //2

count = colors.unshift("black");
alert(count);                 //3

var item = colors.pop();
alert(item);                  //”green”
alert(colors.length);         //2
```



Array Methods

- Reordering:
  - reverse()
  - sort()

- Manipulation:
  - concat()
  - slice()
  - splice()

- Location:
  - `indexOf()`
  - `lastIndexOf()`

- Iterative:
  - `every()`
  - `filter()`
  - `forEach()`
  - `map()`
  - `some()`
  - `reduce()`
  - `reduceRight()`

Date

Date Type

- When the Date constructor is used without any arguments, the created object is assigned the current date and time.

```
var now = new Date();
var someDate = new Date("May 25, 2004");
```

The Regexp Type

- ECMAScript supports regular expressions through the **RegExp** type. Regular expressions are easy to create using syntax similar to Perl, as shown here:
  - var expression = /pattern/flags;

Wrapper Types

Primitive Wrapper Types

- Every time a primitive value is **read**, an object of the corresponding primitive wrapper type is created behind the scenes, allowing access to any **number of methods** for manipulating the data.
  - Boolean, Number & String

```
var s1 = "some text";
var s2 = s1.substring(2);
```

Automatically converted to:

```
var s1 = new String("some text");
var s2 = s1.substring(2);
s1 = null;
```

Primitive Wrapper Lifetime

- A reference type using the **new** operator, it stays in memory until it **goes out of scope**.
- Primitive wrapper objects exist for **only one line of code before they are destroyed**.
- Calling **typeof** on an instance of a primitive wrapper type returns “**object**”.

```
var s1 = "some text";
s1.color = "red";
alert(s1.color); //undefined
```

The Boolean Wrapper

- All primitive wrapper objects convert to the Boolean value true.

```
var falseObject = new Boolean(false);
var result      = falseObject && true;
alert(result); //true

var falseValue = false;
result         = falseValue && true;
alert(result); //false
```

Singleton Built-in Object

Global Object

- The Global object “catchall” properties and methods that don’t otherwise have an owning object.
- All variables and functions defined globally become properties of the Global object.

Windows Object

- Web browsers implement it such that the **window** is the **Global object’s delegate**.
- All variables and functions declared in the global scope become properties on window.

```
var color = "red";
function sayColor(){ alert(window.color); }
window.sayColor(); //red
```

```
var global = function () {
    return this;
}();
```

The Math Object

- ECMAScript provides the Math object as a common location for mathematical formulas and information.

- The Math object execute faster than if you were to write the computations in JavaScript directly.

Scope & Memory

COLMAN CS 2015 B

DR. SHAY HOROVITZ

Agenda

- Primitive & reference values in variables
- Execution context
- Closures
- Function variable
- Recursion with functions
- Garbage collection

Memory Management

- **Value Types Use Stack Memory**
  - Allocation and deallocation are automatic and safe.
    - Undefined, Null, Boolean, Number, and String.
  - Copying Values

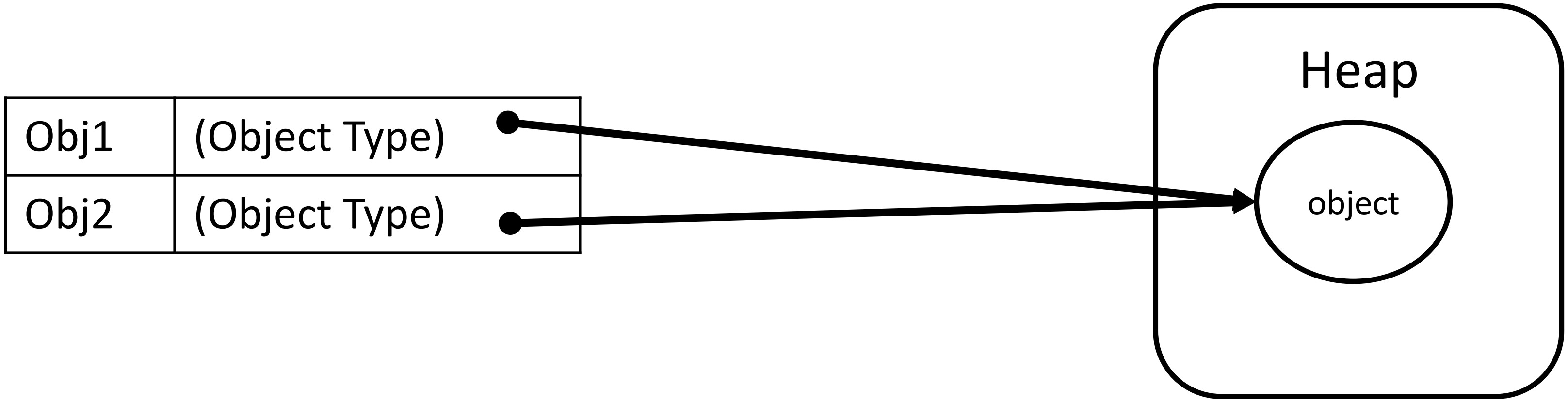
- **Reference Types and Use Heap Memory**
  - Freed by garbage collection.
  - JavaScript does not permit direct access of memory locations.
  - Arguments are passed by value.

Dynamic Properties

- With reference values, at any time you can:
  - add, change, or delete properties and methods.

```
// Reference Type
var person = new Object();
person.name = "Nicholas";
alert(person.name); //"Nicholas"
```

```
// Primitive Type
var name = "Nicholas";
name.age = 27;
alert(name.age); //undefined
```



Obj1	(Object Type)
Obj2	(Object Type)

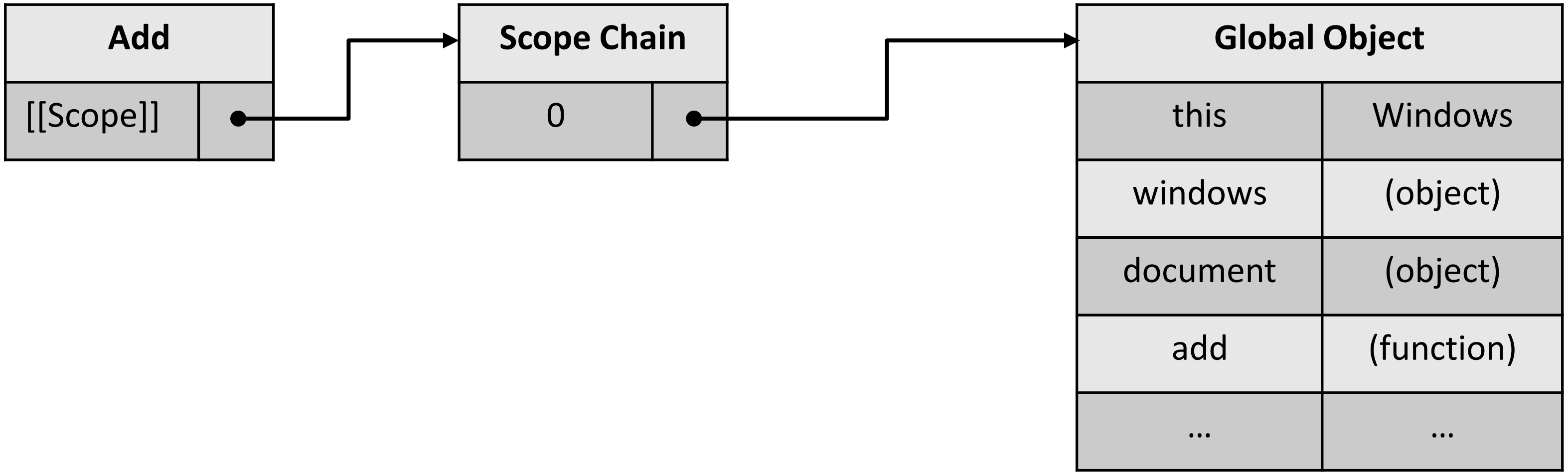
Execution Context & Scope

When Function Created

```
function add(num1, num2) {
    var sum = num1 + num2;
    return sum;
}

add.length === 2;
Object.getPrototypeOf(add) === Function.prototype;
```

When the add() function is **created**, its scope chain is populated with a single variable object



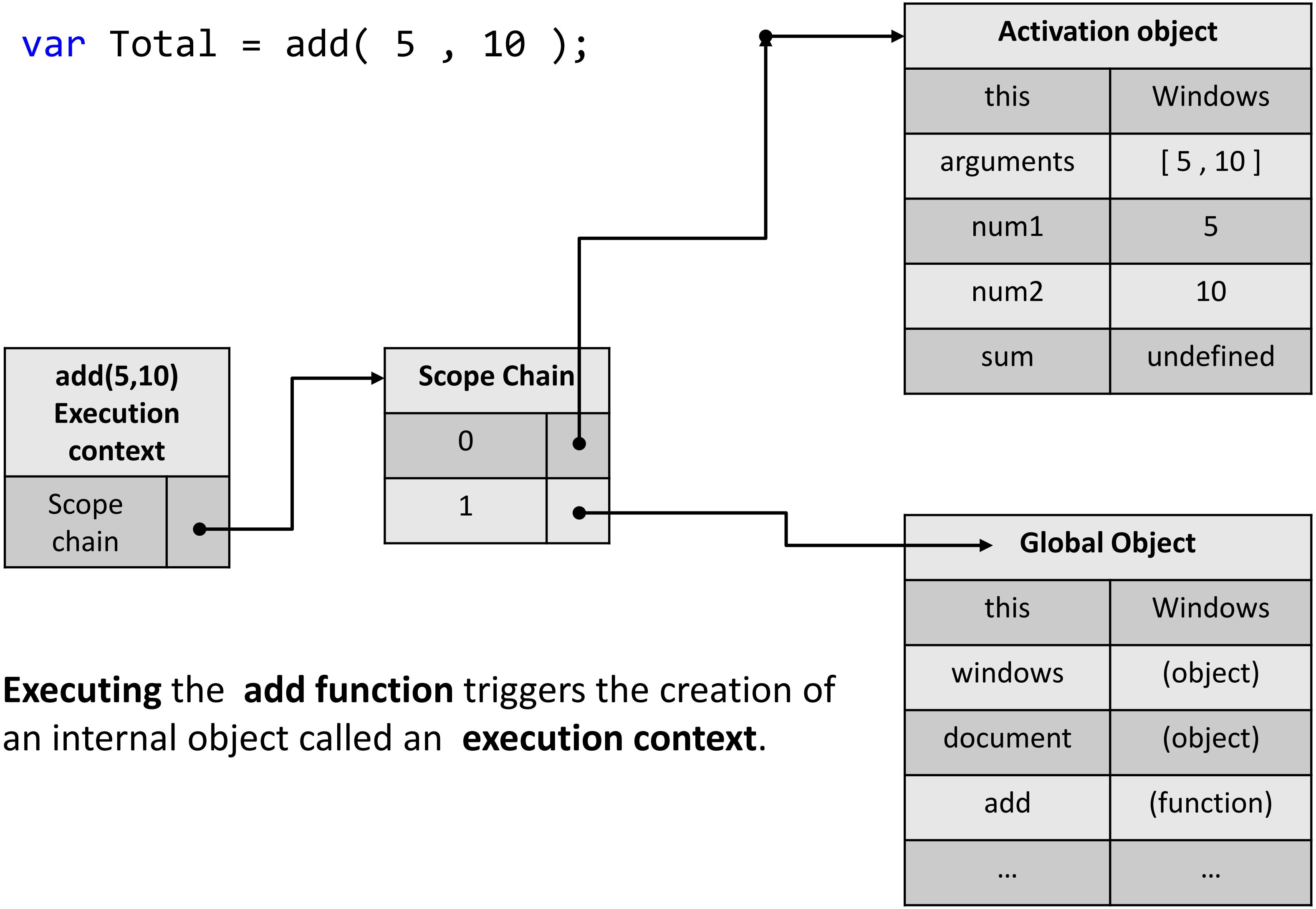
Add	
[[Scope]]	●

Scope Chain	
0	●

Global Object	
this	Windows
windows	(object)
document	(object)
add	(function)
…	…

When Function Executing

```
var Total = add( 5 , 10 );
```



add(5,10) Execution context	
Scope chain	●

Scope Chain	
0	●
1	●

Activation object	
this	Windows
arguments	[5 , 10]
num1	5
num2	10
sum	undefined

Global Object	
this	Windows
windows	(object)
document	(object)
add	(function)
…	…

Executing the **add function** triggers the creation of an internal object called an **execution context**.

When Function Executing

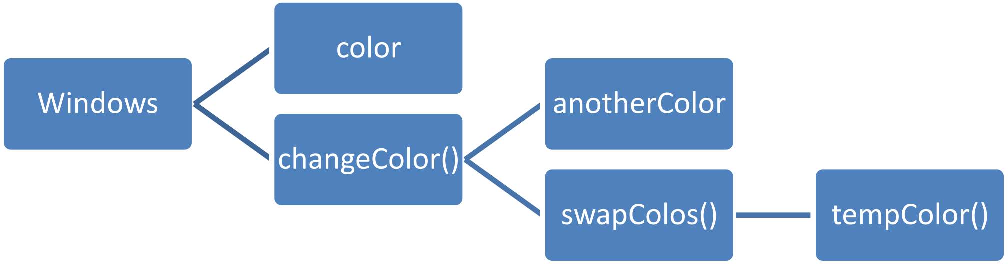
- Each execution context is **unique**.
  - **multiple calls** to the same function result in **multiple execution contexts** being created.

- The execution context is **destroyed** once the function has been **completely executed**.

The activation object

- Acts as the variable object for this execution and contains entries for:
  - local variables
  - named arguments
  - Arguments collection
  - this
- Pushed to the front of the scope chain.
- When the execution context is **destroyed**, so is the activation object.

Scope Chain



```
var color = "blue";

function changeColor() {

    var anotherColor = "red";

    function swapColors(){
        var tempColor = anotherColor;
        anotherColor = color;
        color = tempColor;
        // color, anotherColor, and tempColor
        // are all accessible here.
    }
    // color and anotherColor are accessible here,
    // but not tempColor.
    swapColors();
}
//only color is accessible here
changeColor();
```

Variable Declaration

- Using var automatically added to the most **immediate context** available.

- If a variable is initialized without first being declared, it gets added to the **global context** automatically.

No Block-Level Scopes

- JavaScript’s lack of block-level scopes is a common source of confusion.

```
if (true) {
    var color = "blue";
}
alert(color); //"blue"
```

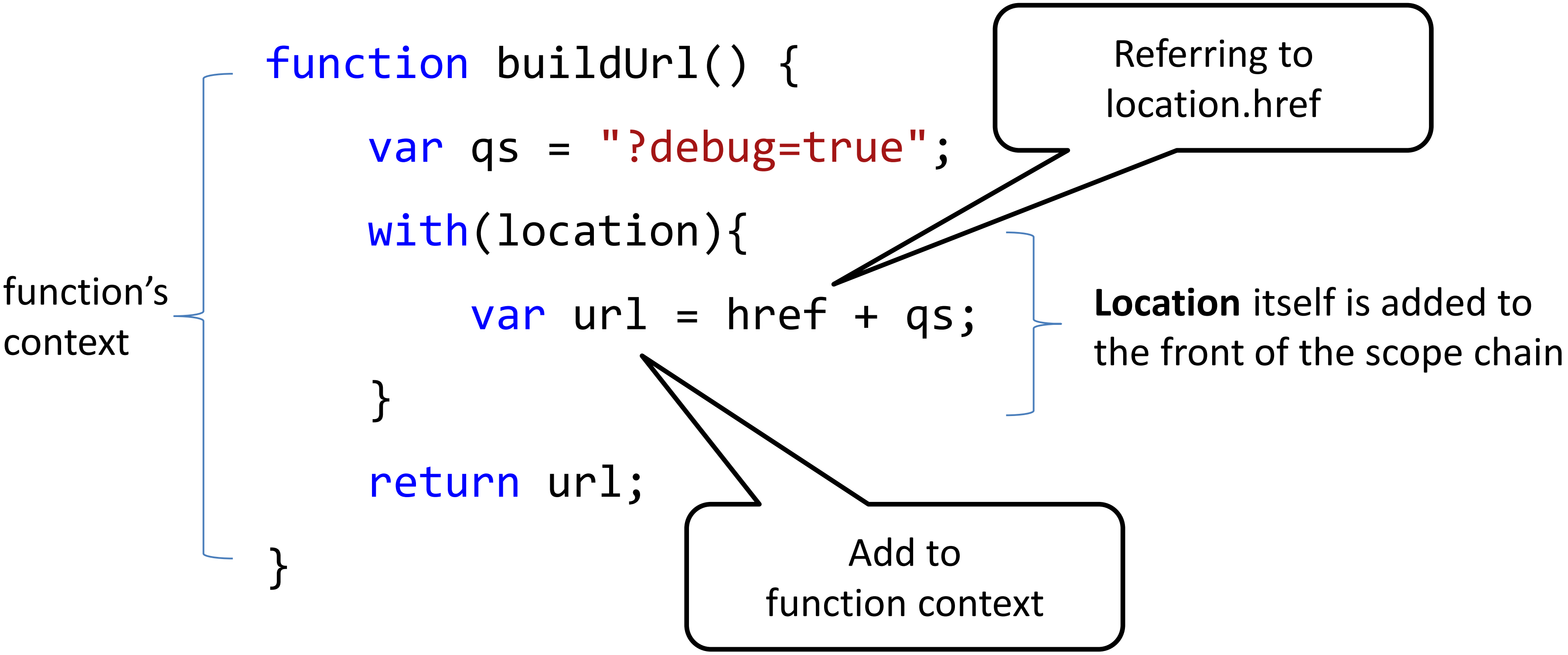
No Block-Level Scopes

```
function outputNumbers(count) {
    for (var i = 0; i < count; i++) {
        alert(i);
    }
    alert(i); //count
}
```

```
// Solutions
function outputNumbers(count) {
    (function () {
        for (var i = 0; i < count; i++) {
            alert(i);
        }
    })();
    alert(i); //causes an error
}
```

Scope Chain Augmentation

- The catch block & a with statement create **scope chain.**



```
function buildUrl() {
    var qs = "?debug=true";
    with(location){
        var url = href + qs;
    }
    return url;
}
```

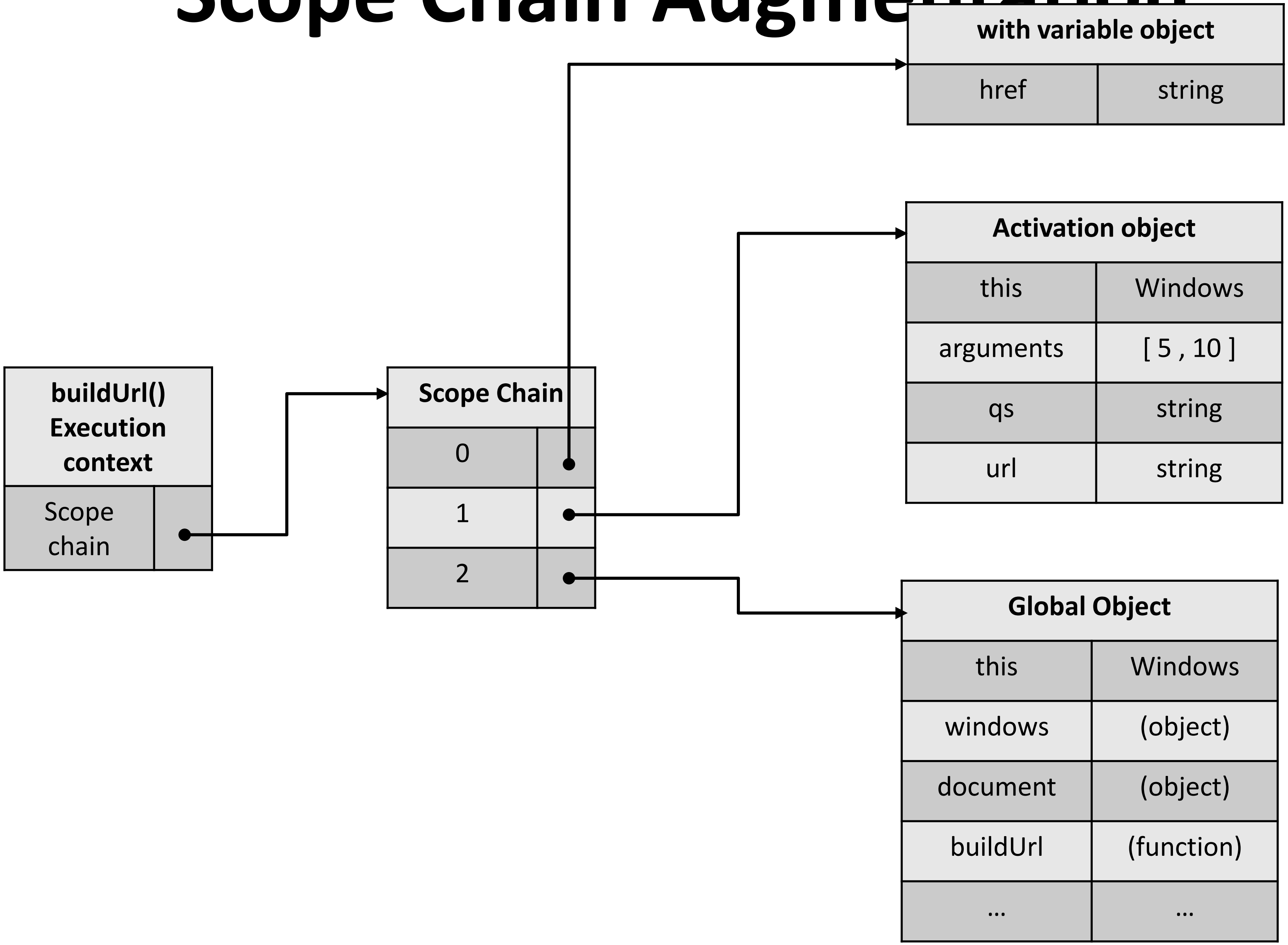
function's context

Referring to location.href

Location itself is added to the front of the scope chain

Add to function context

Scope Chain Augmentation



buildUrl() Execution context	
Scope chain	●

Scope Chain	
0	●
1	●
2	●

with variable object	
href	string

Activation object	
this	Windows
arguments	[5 , 10]
qs	string
url	string

Global Object	
this	Windows
windows	(object)
document	(object)
buildUrl	(function)
…	…

Dynamic Scopes

- A dynamic scope is one that **exists only through execution of code** and therefore cannot be determined simply by **static analysis.**

  - **with** statement.

  - **catch** clause of a try-catch statement.

  - a function containing **eval()**.

Dynamic Scopes

```
function execute(code) {
        eval(code);

        function subroutine() {
            return window;
        }

        var w = subroutine();   //what value is w?
    };


execute("var window = {};")
```