JavaScript 1: Language Fundamentals

Chapter 8

Randy Connolly and Ricardo Hoar

Fundamentals of Web Development

Chapter 8

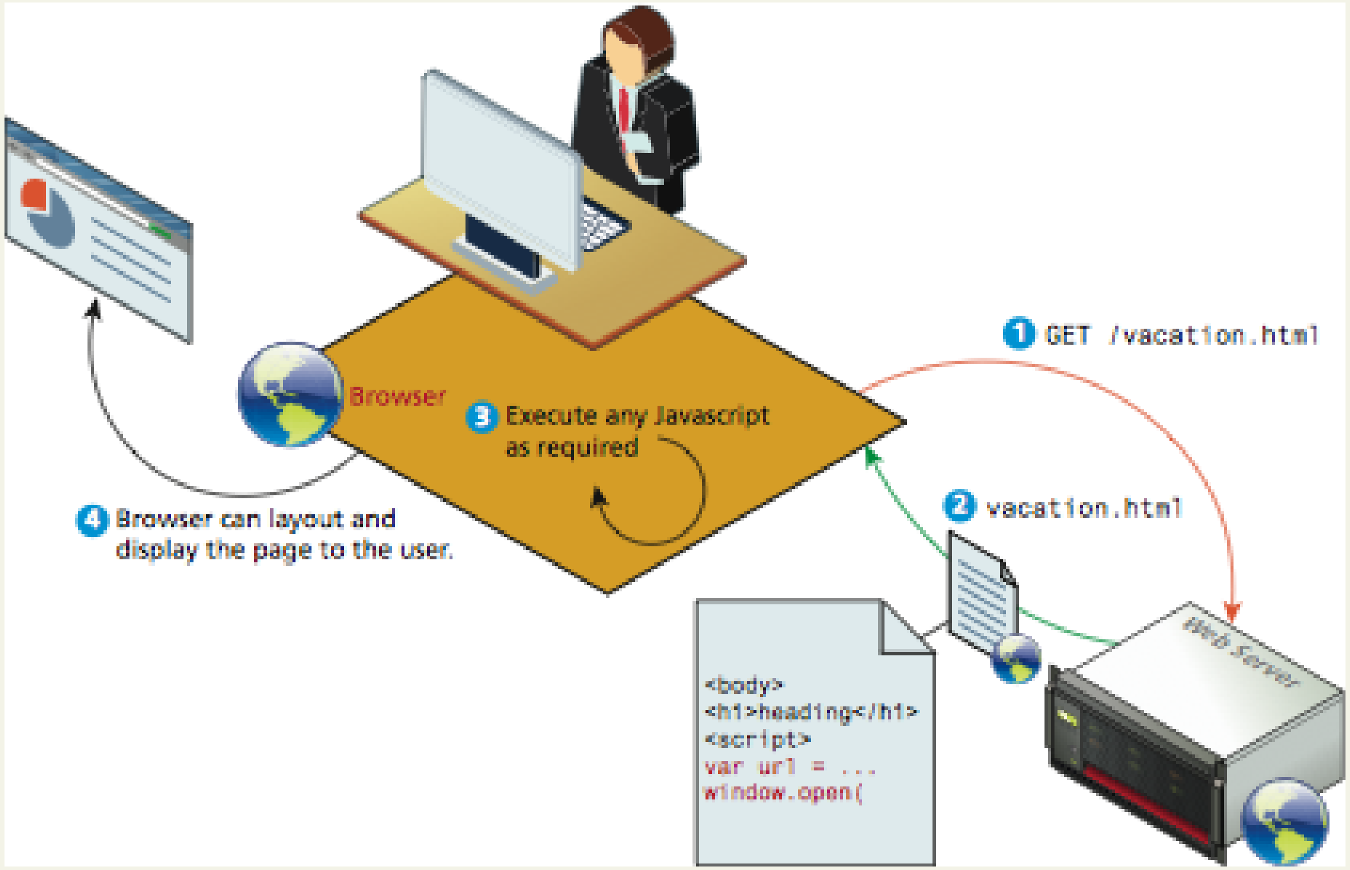
1. JavaScript 1: Language Fundamentals
2. Where Does JavaScript Go?
3. Variables and Data Types
4. JavaScript Output
5. Conditionals
6. Loops
7. Arrays
8. Objects

Chapter 8 cont.

Chapter 8

What is JavaScript & What Can It Do?

Client-Side Scripting



What is JavaScript & What Can It Do?

JavaScript's History

- JavaScript was introduced by Netscape in their Navigator browser back in 1996
- JavaScript that is supported by your browser contains language features
    - not included in the current ECMAScript specification and
    - missing certain language features from that specification

The latest version of ECMAScript is the Sixth Edition (generally referred to as ES6 or ES2015).

What is JavaScript & What Can It Do?

JavaScript and Web 2.0

- Early JavaScript had only a few common uses:
- 2000s onward saw more sophisticated uses for JavaScript
- **AJAX** as both an acronym and a general term
- Chapters 10 and 19 will cover AJAX in much more detail.

What is JavaScript & What Can It Do?

JavaScript in Contemporary Software Development

Chapter 8

Where Does JavaScript Go?

Inline JavaScript

Inline JavaScript refers to the practice of including JavaScript code directly within certain HTML attributes

```
<a href="JavaScript:OpenWindow();">more info</a>

<input type="button" onClick="alert('Are you sure?');" />
```

Where Does JavaScript Go?

Embedded JavaScript

Embedded JavaScript refers to the practice of placing JavaScript code within a <script> element

```
<script type="text/javascript">
        /* A JavaScript Comment */
        alert("Hello World!");
</script>
```

Where Does JavaScript Go?

External JavaScript

external JavaScript files typically contain function definitions, data definitions, and entire frameworks.

```
<head>
    <script type="text/javascript" src="greeting.js"></script>
</head>
```

Where Does JavaScript Go?

Users without JavaScript

- Web crawler
- Browser plug-in.
- Text-based client.
- Visually disabled client.
- The <NoScript> Tag

Chapter 8

Variables and Data Types

Variables in JavaScript are **dynamically typed**

This simplifies variable declarations, since we do not require the familiar data-type identifiers

Instead we simply use the **var** keyword

Variables and Data Types

Example variable declarations and Assignments

Defines a variable named abc

```
var abc;
```

Each line of JavaScript should be terminated with a semicolon

```
var def = 0;
```

A variable named def is defined and initialized to 0

```
def= 4 ;
```

def is assigned the value of 4

Notice that whitespace is unimportant

```
def   =
   "hello"   ;
```

def is assigned the value of "hello"

Notice that a line of JavaScript can span multiple lines

Variables and Data Types

Data Types

two basic data types:

- reference types (usually referred to as objects) and
- primitive types

Primitive types represent simple forms of data.

- **Boolean, Number, String, …**

Variables and Data Types

Reference Types

```
var abc = 27;
var def = "hello";
```

variables with primitive types

```
var foo = [45, 35, 25];
```

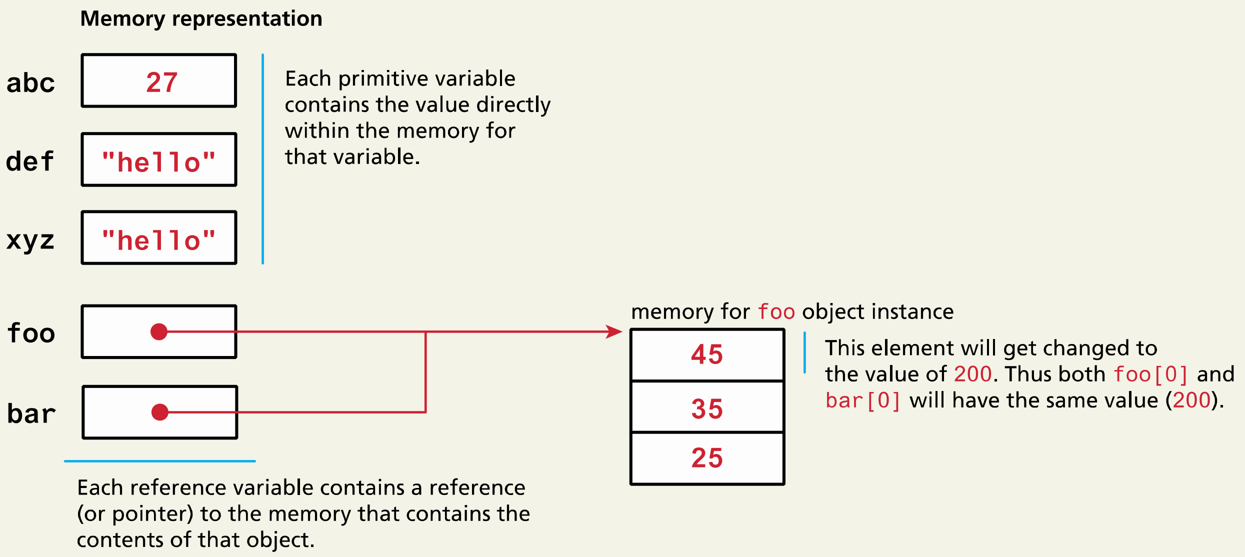
variable with reference type (i.e., array object)

```
var xyz = def;
var bar = foo;
```

these new variables differ in important ways (see below)

```
bar[0] = 200;
```

changes value of the first element of array



Each primitive variable contains the value directly within the memory for that variable.

This element will get changed to the value of 200. Thus both foo[0] and bar[0] will have the same value (200).

Each reference variable contains a reference (or pointer) to the memory that contains the contents of that object.

Chapter 8

JavaScript Output

```
alert("Hello world");
```

JavaScript Output

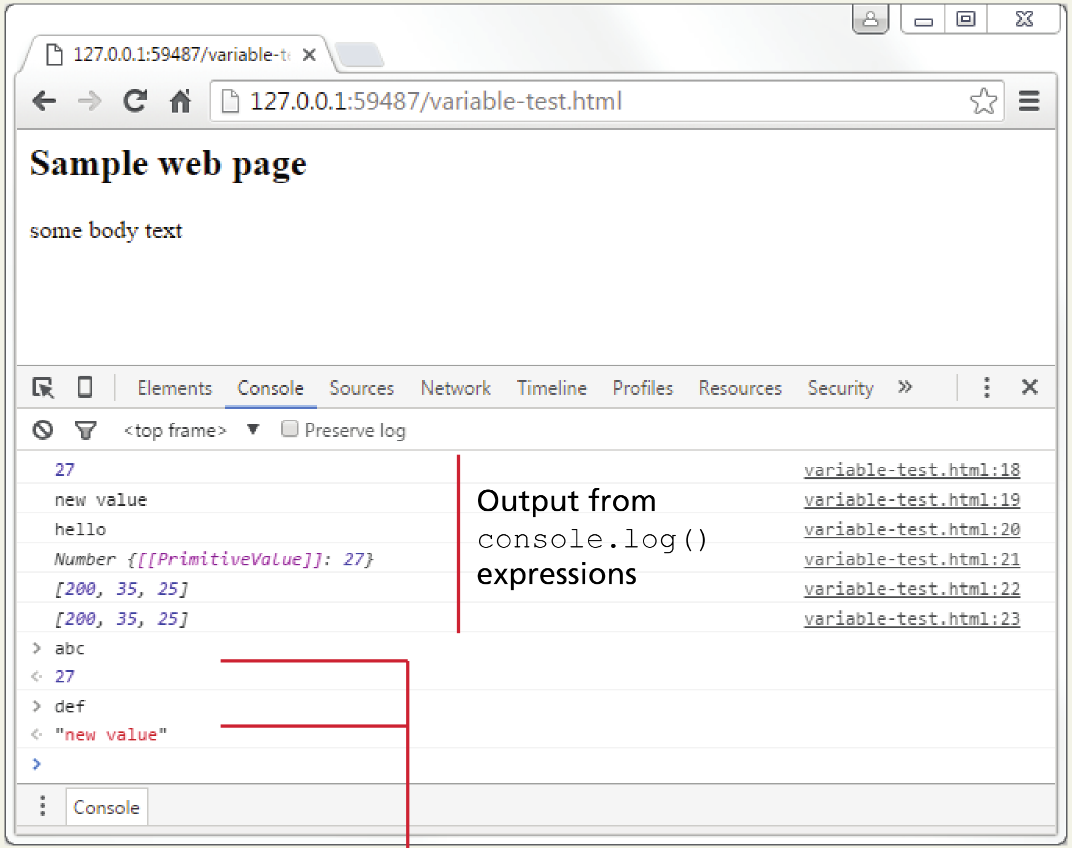
```
var name = "Randy";
document.write("<h1>Title</h1>");
// this uses the concatenate operator (+)
document.write("Hello " + name + " and welcome");
```

JavaScript Output

- alert() Displays content within a pop-up box.
- console.log() Displays content in the Browser’s JavaScript console.
- document.write() Outputs the content (as markup) directly to the HTML document.

JavaScript Output

Chrome JavaScript Console



Sample web page

some body text

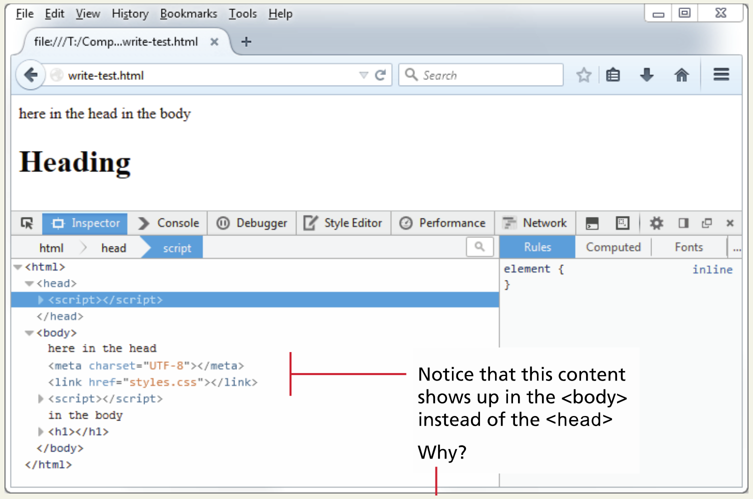
```
27                                              variable-test.html:18
new value                                       variable-test.html:19
hello                                           variable-test.html:20
Number {[[PrimitiveValue]]: 27}                 variable-test.html:21
[200, 35, 25]                                   variable-test.html:22
[200, 35, 25]                                   variable-test.html:23
> abc
< 27
> def
< "new value"
>
```

Output from console.log() expressions

Using console interactively to query value of JavaScript variables

JavaScript Output

Fun with document.write()



here in the head in the body

Heading

```
<html>
  <head>
    <script></script>
  </head>
  <body>
    here in the head
    <meta charset="UTF-8"></meta>
    <link href="styles.css"></link>
    <script></script>
    in the body
    <h1></h1>
  </body>
</html>
```

```
element {
}
inline
```

Notice that this content shows up in the <body> instead of the <head>

Why?

The appearance of this line will shift the following `write()` calls to the `<body>`

Chapter 8

Conditionals

If, else if, else

```
if (hourOfDay > 4 && hourOfDay < 12) {
        greeting = "Good Morning";
}
else if (hourOfDay >= 12 && hourOfDay < 18) {
        greeting = "Good Afternoon";
}
else {
        greeting = "Good Evening";
}
```

Conditionals

switch

```
switch (artType) {
        case "PT":
                output = "Painting";
                break;
        case "SC":
                output = "Sculpture";
                break;
        default:
        output = "Other";
}
```

Conditionals

Conditional Assignment

```
/* x conditional assignment */
```

$$\texttt{x = }\underbrace{\texttt{(y==4)}}_{\text{Condition}}\texttt{ ? }\underbrace{\texttt{"y is 4"}}_{\text{Value if true}}\texttt{ : }\underbrace{\texttt{"y is not 4"}}_{\text{Value if false}}\texttt{;}$$

```
/* equivalent to */
if (y==4) {
   x = "y is 4";
}
else {
   x = "y is not 4";
}
```

Conditionals

Truthy and Falsy

In JavaScript, a value is said to be **truthy** if it translates to true, while a value is said to be **falsy** if it translates to false.

- Almost all values in JavaScript are truthy
- false, null, "", '', 0, NaN, and undefined are falsy

Chapter 8

Loops

While and do . . . while Loops

```
var count = 0;
while (count < 10) {
        // do something
        //  ...
        count++;
}
count = 0;
do {
        // do something
        //  ...
        count++;
} while (count < 10);
```

Loops

For Loops

```
    initialization   condition   post-loop operation
for (var i = 0;      i < 10;     i++) {
  // do something with i
  // ...
}
```

Chapter 8

Arrays

Arrays are one of the most commonly used data structures in programming.

JavaScript provides two main ways to define an array.

- object literal notation
- use the Array() constructor

Arrays

object literal notation

The literal notation approach is generally preferred since it involves less typing, is more readable, and executes a little bit quicker

```
var years = [1855, 1648, 1420];

var countries = ["Canada", "France",

                 "Germany", "Nigeria",

                 "Thailand", "United States"];

var mess = [53, "Canada", true, 1420];
```

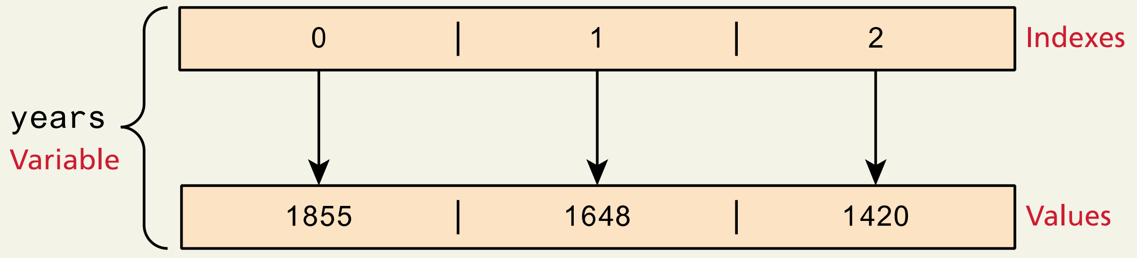
Arrays

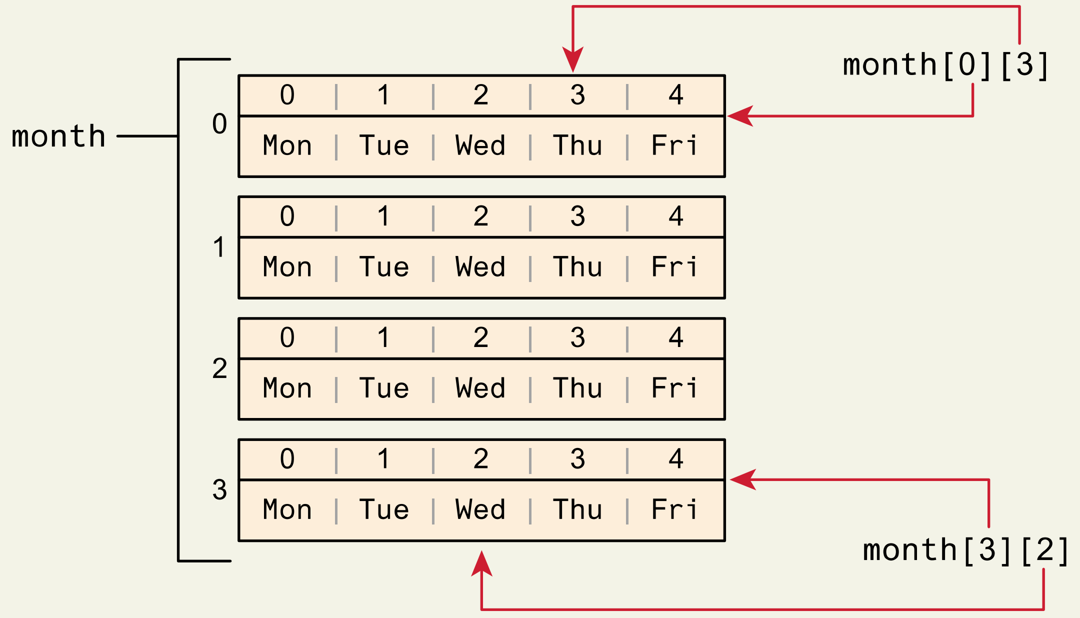
Some common features

- arrays in JavaScript are zero indexed
- [] notation for access
- .length gives the length of the array
- .push()
- .pop()
- concat(), slice(), join(), reverse(), shift(), and sort()

Arrays

Arrays Illustrated





Chapter 8

1 JavaScript 1: Language Fundamentals

2 Where Does JavaScript Go?

3 Variables and Data Types

4 JavaScript Output

5 Conditionals

6 Loops

7 Arrays

8 Objects

Objects

Object Creation—Object Literal Notation

```
var objName = {
        name1: value1,
        name2: value2,
        //  ...
        nameN: valueN
};
```

Objects

Object Creation—Object Literal Notation

Access using either of:

- objName.name1
- objName["name1"]

Objects

Object Creation—Constructed Form

```
// first create an empty object

var objName = new Object();

// then define properties for this object

objName.name1 = value1;

objName.name2 = value2;
```

Chapter 8 cont.

Functions

Function Declarations vs. Function Expressions

Functions are the building block for modular code in JavaScript.

```
function subtotal(price,quantity) {
        return price * quantity;
}
```

The above is formally called a **function declaration,** called or invoked by using the () operator

```
var result = subtotal(10,2);
```

Functions

Function Declarations vs. Function Expressions

```
// defines a function using a function expression

var sub = function subtotal(price,quantity) {

        return price * quantity;

};

// invokes the function

var result = sub(10,2);
```

It is conventional to leave out the function name in function expressions

Functions

Anonymous Function Expressions

```
// defines a function using an anonymous function expression
var calculateSubtotal = function (price,quantity) {
        return price * quantity;
};
// invokes the function
var result = calculateSubtotal(10,2);
```

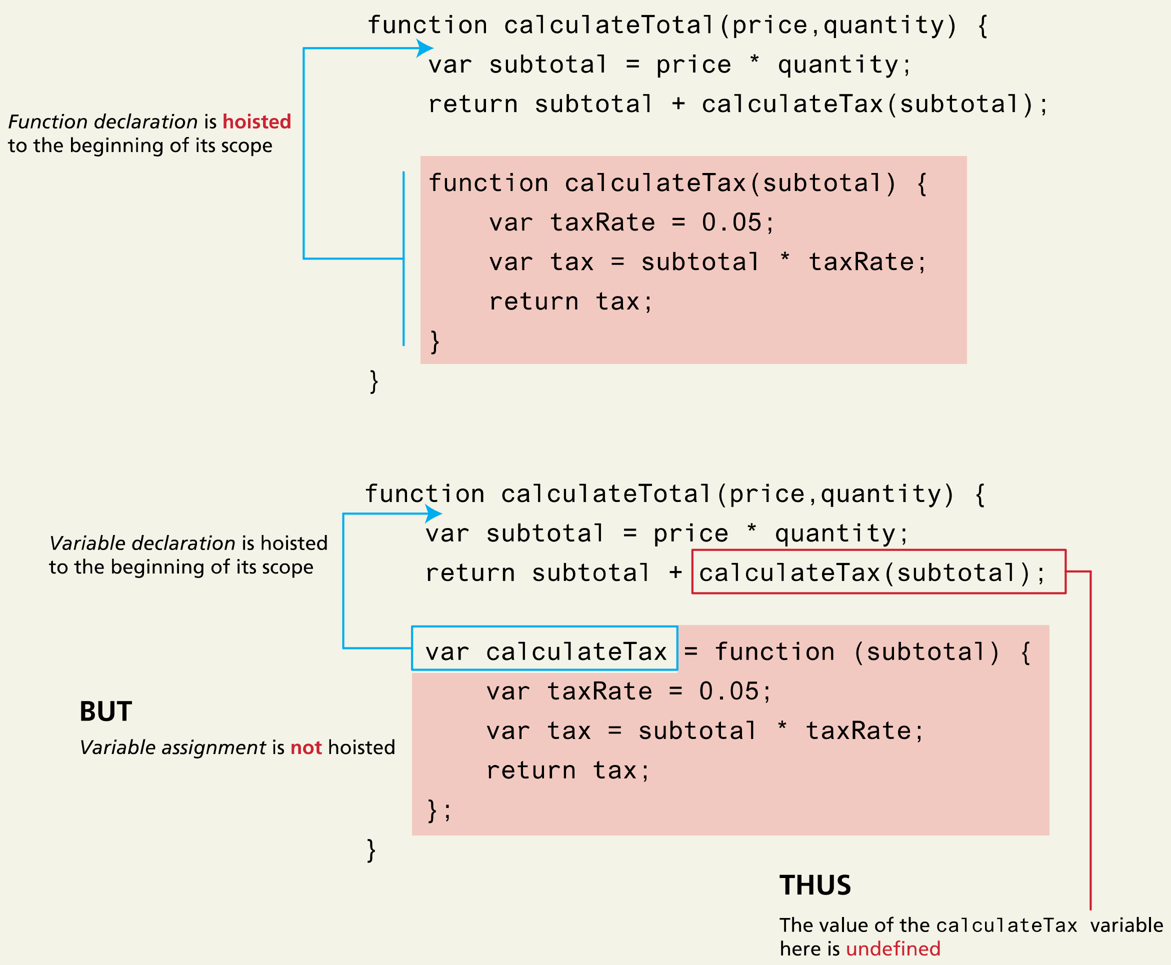
Functions

Nested Functions

```
function calculateTotal(price,quantity) {
        var subtotal = price * quantity;
        return subtotal + calculateTax(subtotal);
        // this function is nested
        function calculateTax(subtotal) {
                var taxRate = 0.05;
                var tax = subtotal * taxRate;
                return tax;
        }
}
```

Functions

Hoisting in JavaScript



Function declaration is **hoisted** to the beginning of its scope

```
function calculateTotal(price,quantity) {
   var subtotal = price * quantity;
   return subtotal + calculateTax(subtotal);

   function calculateTax(subtotal) {
      var taxRate = 0.05;
      var tax = subtotal * taxRate;
      return tax;
   }
}
```

Variable declaration is hoisted to the beginning of its scope

BUT

Variable assignment is **not** hoisted

```
function calculateTotal(price,quantity) {
   var subtotal = price * quantity;
   return subtotal + calculateTax(subtotal);

   var calculateTax = function (subtotal) {
      var taxRate = 0.05;
      var tax = subtotal * taxRate;
      return tax;
   };
}
```

THUS

The value of the `calculateTax` variable here is undefined

Functions

Callback Functions

```
var calculateTotal = function (price, quantity, tax) {
    var subtotal = price * quantity;
    return subtotal + tax(subtotal);
};
```

2 The local parameter variable `tax` is a reference to the `calcTax()` function

```
var calcTax = function (subtotal) {
    var taxRate = 0.05;
    var tax = subtotal * taxRate;
    return tax;
};
```

1 Passing the `calcTax()` function object as a parameter

We can say that `calcTax` variable here is a callback function

```
var temp = calculateTotal(50,2,calcTax);
```

Functions

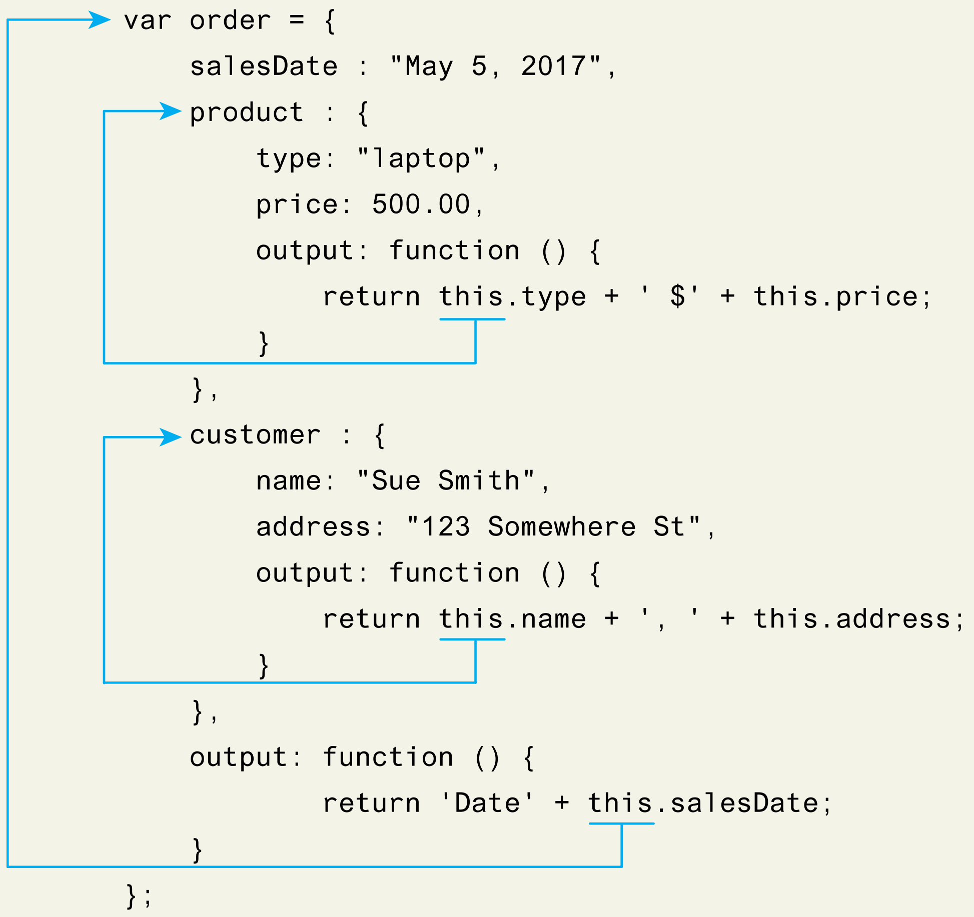
Callback Functions

Passing an anonymous function definition as a callback function parameter

```
var temp = calculateTotal( 50, 2,
                           function (subtotal) {
                              var taxRate = 0.05;
                              var tax = subtotal * taxRate;
                              return tax;
                           }
                         );
```

Functions

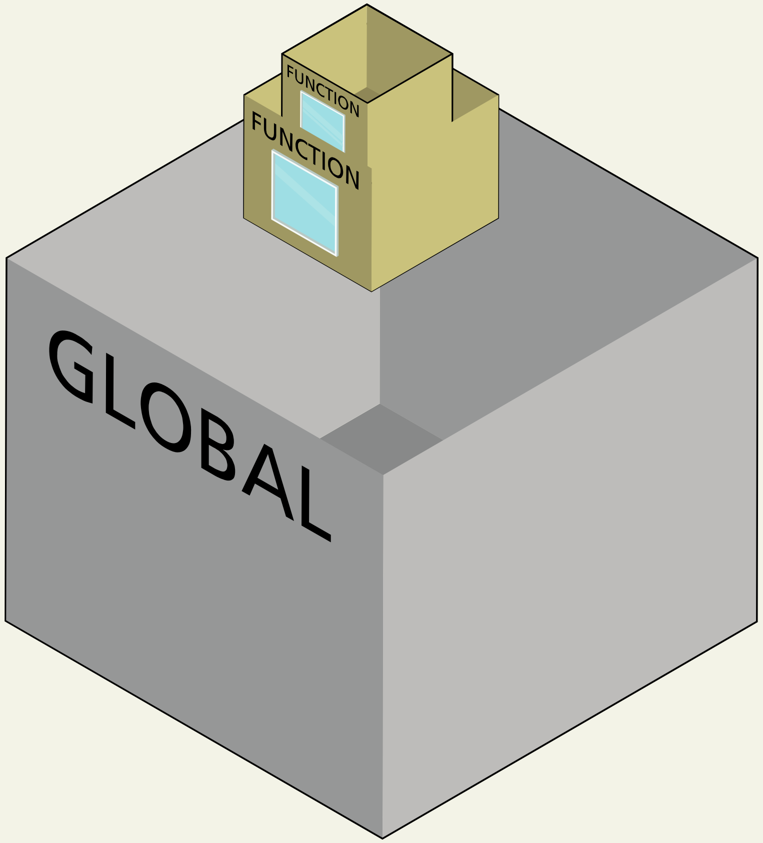
Objects and Functions Together



```
var order = {
    salesDate : "May 5, 2017",
    product : {
        type: "laptop",
        price: 500.00,
        output: function () {
            return this.type + ' $' + this.price;
        }
    },
    customer : {
        name: "Sue Smith",
        address: "123 Somewhere St",
        output: function () {
            return this.name + ', ' + this.address;
        }
    },
    output: function () {
            return 'Date' + this.salesDate;
    }
};
```

Functions

Scope in JavaScript

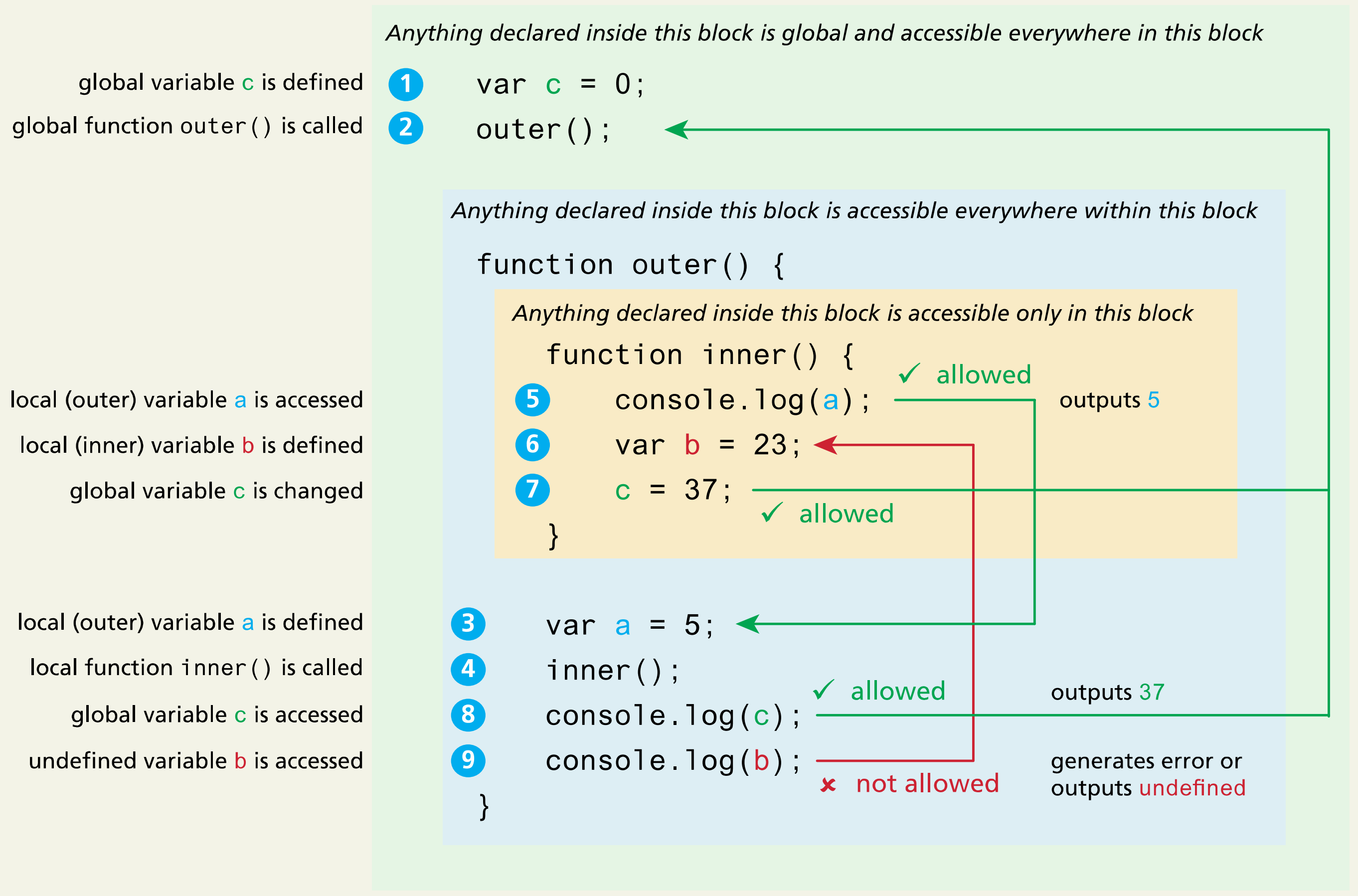


utside

of the global box

Functions

Scope in JavaScript



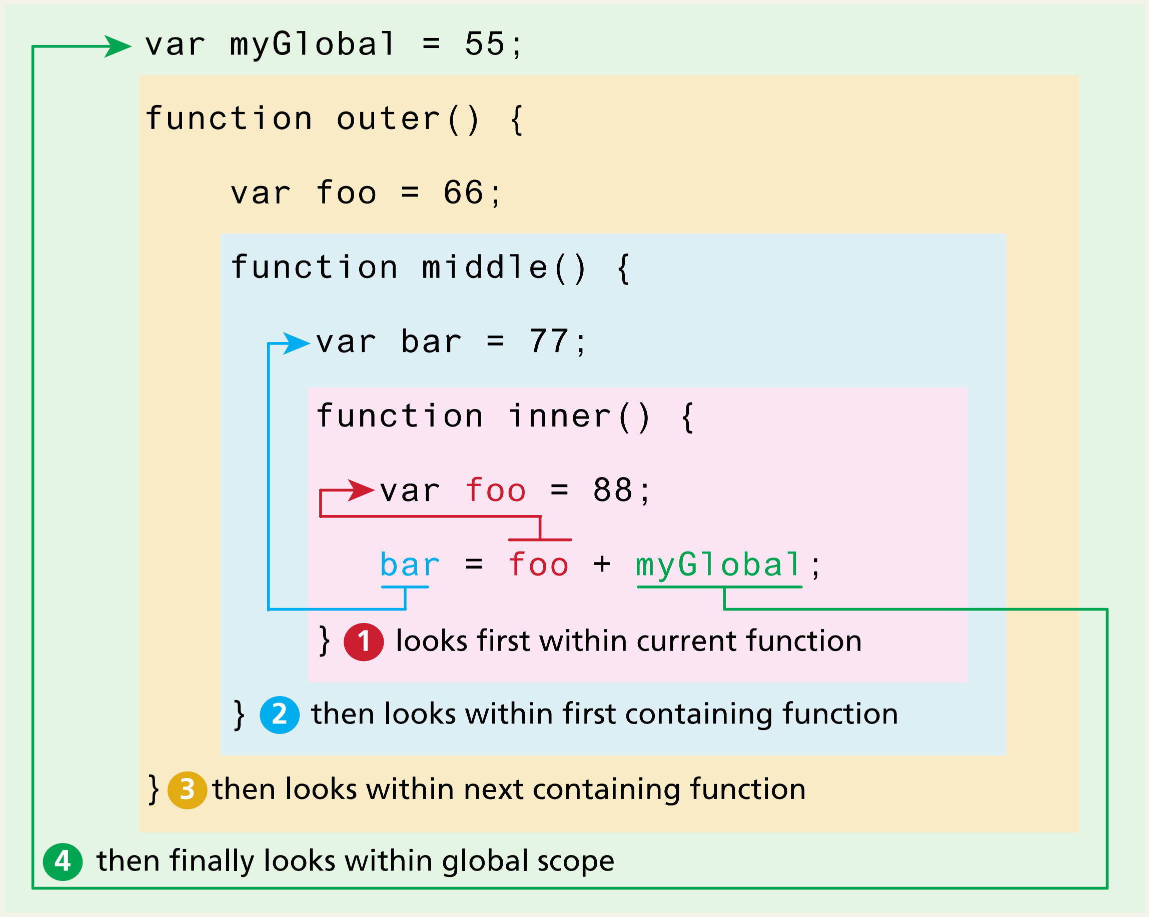
```
var c = 0;
outer();

function outer() {
    function inner() {
        console.log(a);
        var b = 23;
        c = 37;
    }

    var a = 5;
    inner();
    console.log(c);
    console.log(b);
}
```

Anything declared inside this block is global and accessible everywhere in this block
Anything declared inside this block is accessible everywhere within this block
Anything declared inside this block is accessible only in this block
1 global variable c is defined
2 global function outer() is called
3 local (outer) variable a is defined
4 local function inner() is called
5 local (outer) variable a is accessed
6 local (inner) variable b is defined
7 global variable c is changed
8 global variable c is accessed
9 undefined variable b is accessed
✓ allowed
outputs 5
✓ allowed
✓ allowed
outputs 37
✗ not allowed
generates error or outputs undefined

Functions

Scope in JavaScript

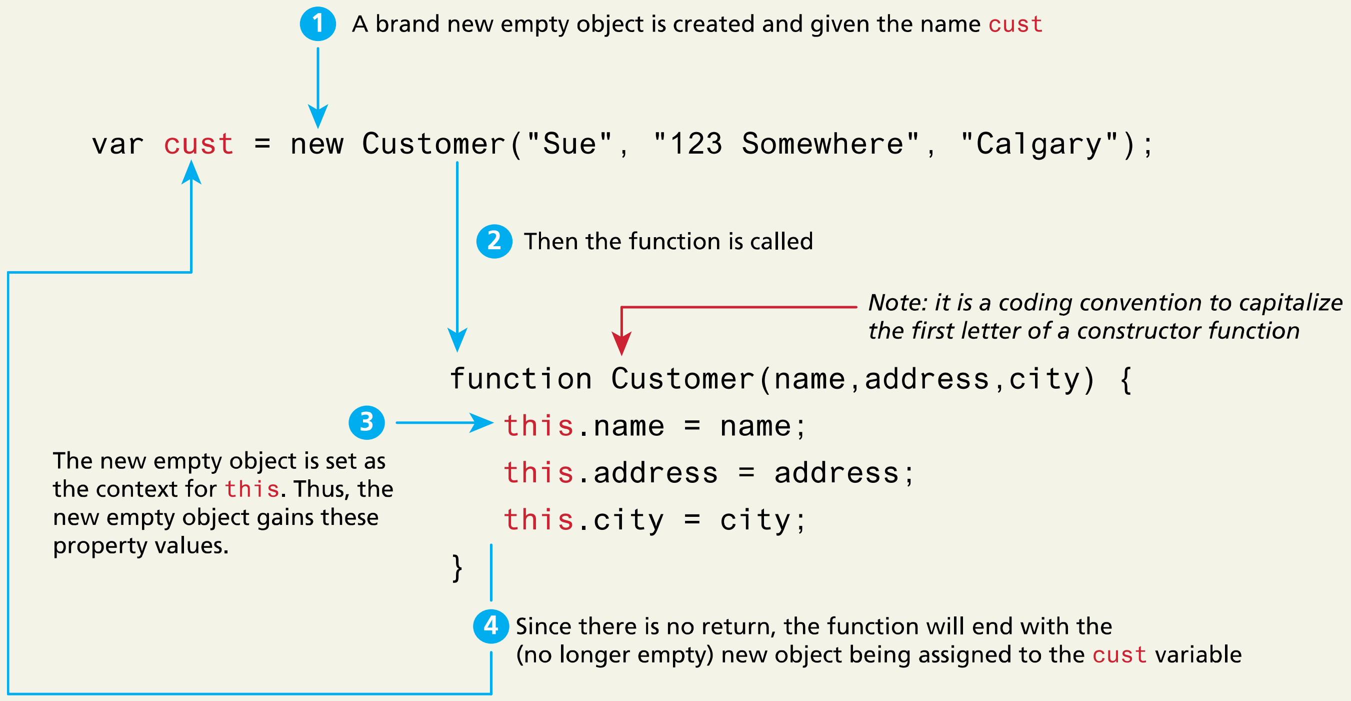
Remember that scope is determined at design-time



```
var myGlobal = 55;
function outer() {
   var foo = 66;
   function middle() {
      var bar = 77;
      function inner() {
         var foo = 88;
         bar = foo + myGlobal;
      } 1 looks first within current function
   } 2 then looks within first containing function
} 3 then looks within next containing function
4 then finally looks within global scope
```

Functions

Function Constructors



1 A brand new empty object is created and given the name cust

```
var cust = new Customer("Sue", "123 Somewhere", "Calgary");
```

2 Then the function is called

Note: it is a coding convention to capitalize the first letter of a constructor function

```
function Customer(name,address,city) {
   this.name = name;
   this.address = address;
   this.city = city;
}
```

3 The new empty object is set as the context for this. Thus, the new empty object gains these property values.

4 Since there is no return, the function will end with the (no longer empty) new object being assigned to the cust variable

Chapter 8 cont.

9 Functions

10 Object Prototypes

11 Summary

Object Prototypes

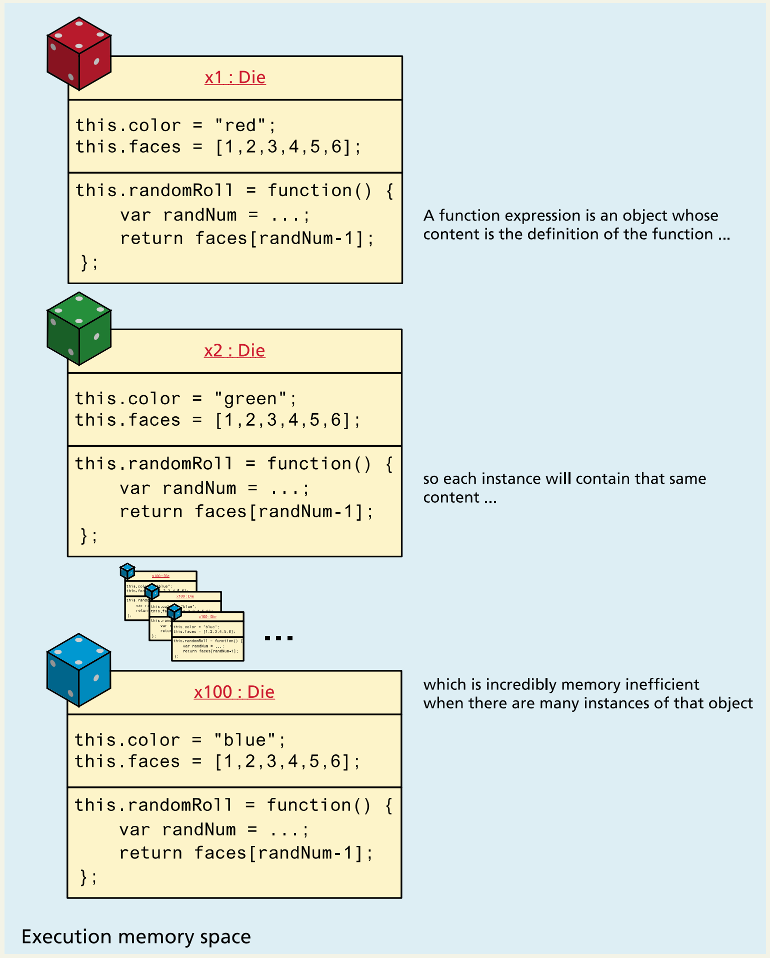
There's a better way

While the constructor function is simple to use, it can be an inefficient approach for objects that contain methods.

Prototypes are an essential syntax mechanism in JavaScript, and are used to make JavaScript behave more like an object-oriented language.

Object Prototypes

Methods get duplicated…



x1 : Die
x2 : Die
x100 : Die
...
A function expression is an object whose content is the definition of the function ...
so each instance will contain that same content ...
which is incredibly memory inefficient when there are many instances of that object
Execution memory space

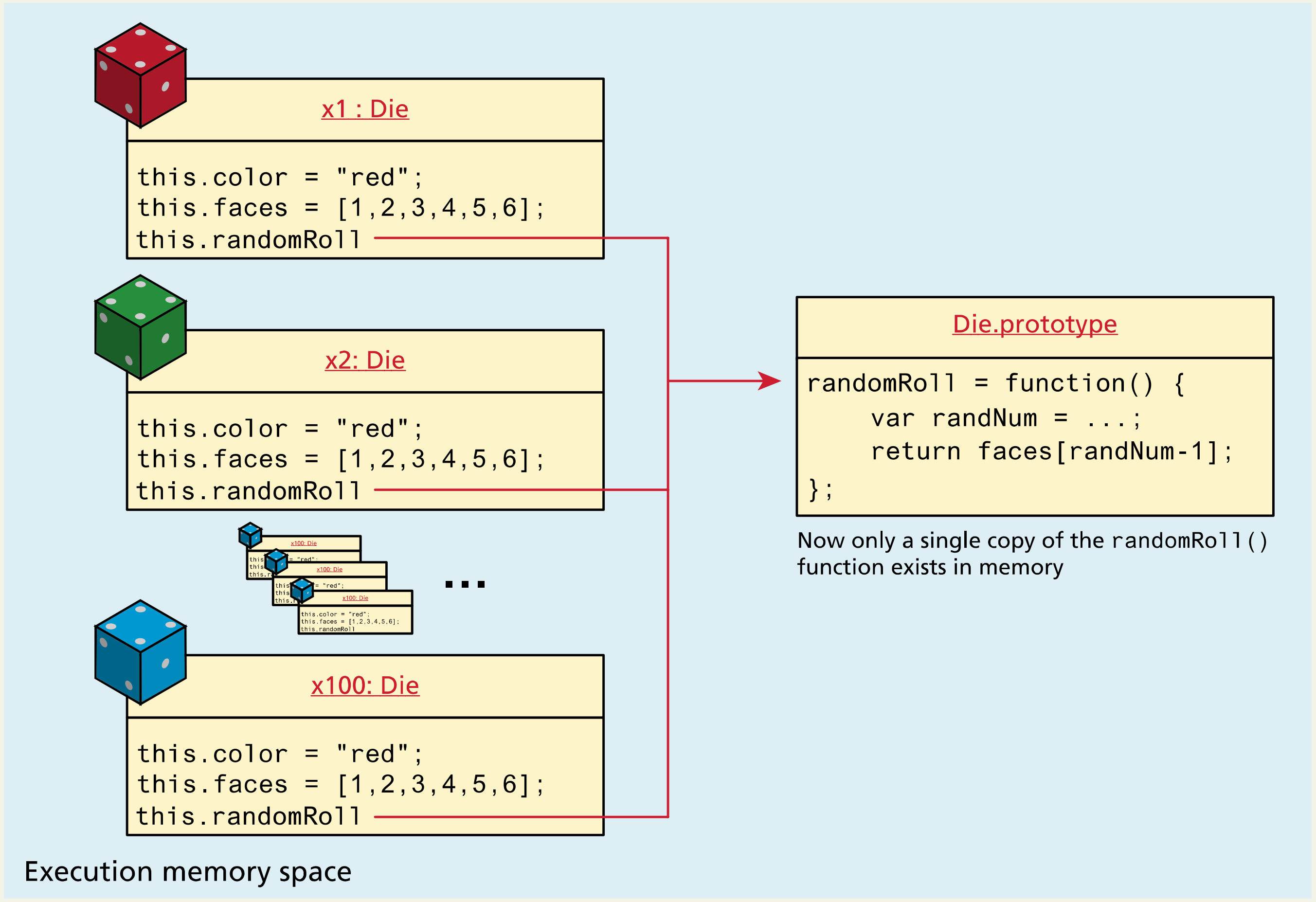
```
this.color = "red";
this.faces = [1,2,3,4,5,6];

this.randomRoll = function() {
    var randNum = ...;
    return faces[randNum-1];
};
```

```
this.color = "green";
this.faces = [1,2,3,4,5,6];

this.randomRoll = function() {
    var randNum = ...;
    return faces[randNum-1];
};
```

```
this.color = "blue";
this.faces = [1,2,3,4,5,6];

this.randomRoll = function() {
    var randNum = ...;
    return faces[randNum-1];
};
```

Object Prototypes

Using Prototypes reduces duplication at run time.



```
this.color = "red";
this.faces = [1,2,3,4,5,6];
this.randomRoll
```

```
randomRoll = function() {
    var randNum = ...;
    return faces[randNum-1];
};
```

Now only a single copy of the randomRoll() function exists in memory

Object Prototypes

Using Prototypes to Extend Other Objects

```
String.prototype.countChars = function (c) {
        var count=0;
        for (var i=0;i<this.length;i++) {
                if (this.charAt(i) == c)
                        count++;
                }
        return count;
}
var msg = "Hello World";
console.log(msg + "has" + msg.countChars("l") + " letter l's");
```

Chapter 8 cont.

Summary

Key Terms

ActionScript	ES2015	libraries
Adobe Flash	ES6	loop control variable
anonymous functions	exception	method
assignment	expressions	minification
AJAX	external JavaScript files	module pattern
applet	falsy	namespace conflict
arrays	fail-safe design	problem
arrow functions	for loops	objects
associative arrays	functions	object literal notation
browser extension	function constructor	primitive types
browser plug-in	function declaration	property
built-in objects	function expression	prototypes
callback function	inline JavaScript	reference types
client-side scripting	immediately-invoked	scope (local and global)
closure	function	strict mode
conditional assignment	Java applet	throw
dot notation	JavaScript frameworks	truthy
dynamically typed	JavaScript Object Notation	try. . . catch block
ECMAScript	JSON	undefined
embedded JavaScript	lexical scope	variables

Summary

Questions?