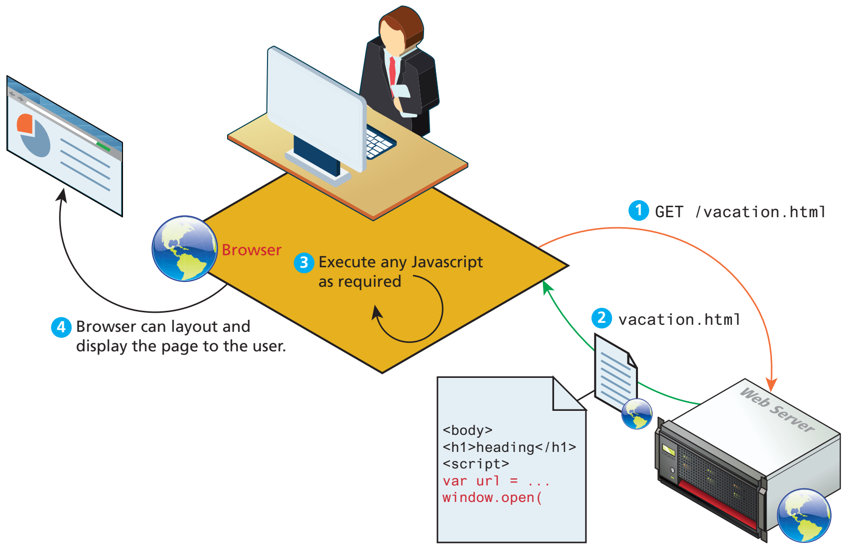
# Appendix C - Scripting Language

Introduction to CSS

- JavaScript Language Fundamentals
- Variables and Outputs
- Constructs, Arrays and Functions
- The DOM

## What is JavaScript & What Can It Do?

Client-Side Scripting


Browser
1 GET /vacation.html
2 vacation.html
3 Execute any Javascript as required
4 Browser can layout and display the page to the user.
Web Server

```
<body>
<h1>heading</h1>
<script>
var url = ...
window.open(
```

## 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>
```

## Variables and Data Types

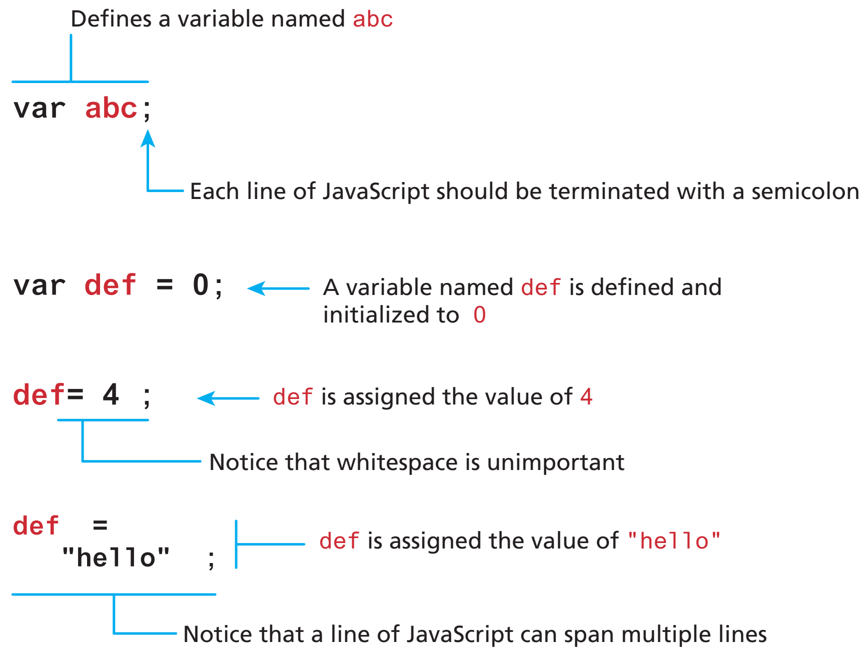
Variables in JavaScript are **dynamically typed**

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];
```

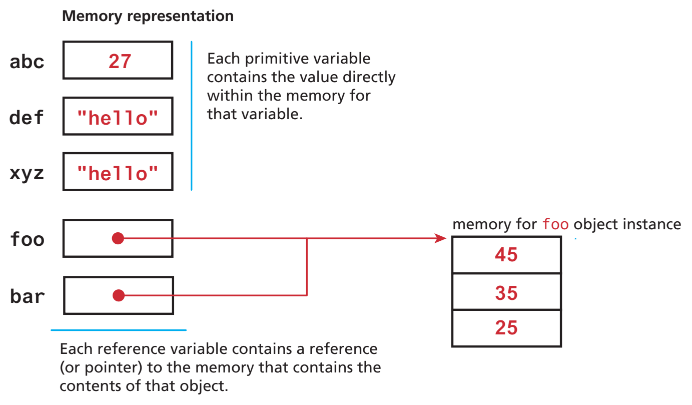
variable with reference type (i.e., array object)

```
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

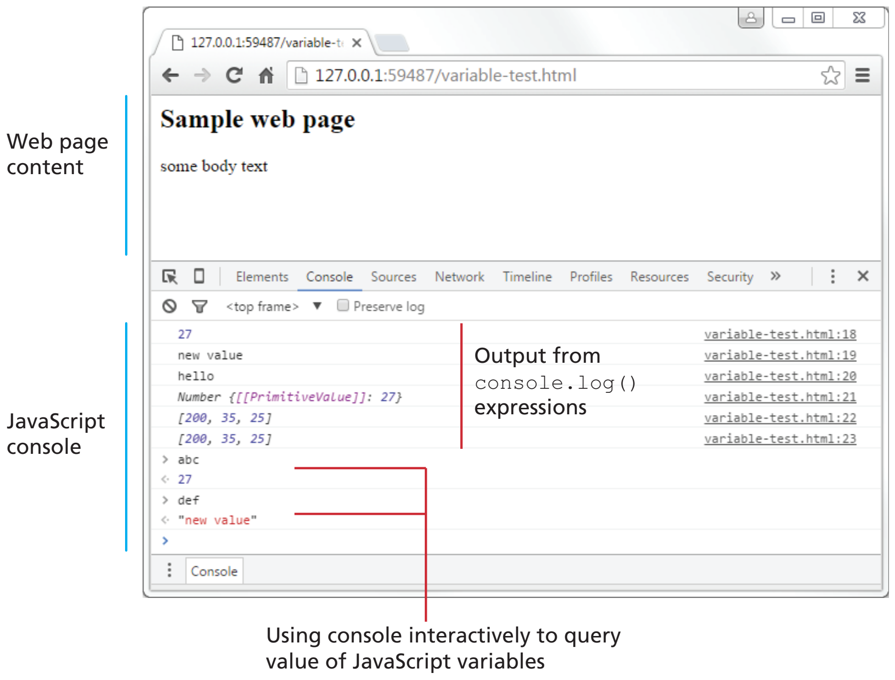

Memory representation
abc
27
def
"hello"
xyz
"hello"
Each primitive variable contains the value directly within the memory for that variable.
foo
bar
memory for foo object instance
45
35
25
Each reference variable contains a reference (or pointer) to the memory that contains the contents of that object.

## 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


Web page content
JavaScript console
127.0.0.1:59487/variable-test.html
Sample web page
some body text
Elements Console Sources Network Timeline Profiles Resources Security
<top frame>
Preserve log
Output from console.log() expressions
Console
Using console interactively to query value of JavaScript variables

```
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"
>
```

## 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 */
```

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

```
/* 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

## 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
    // ...
}
```

## 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];
```

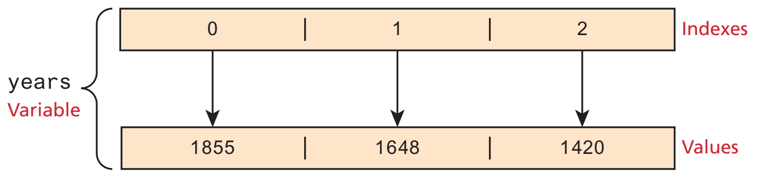
## Arrays

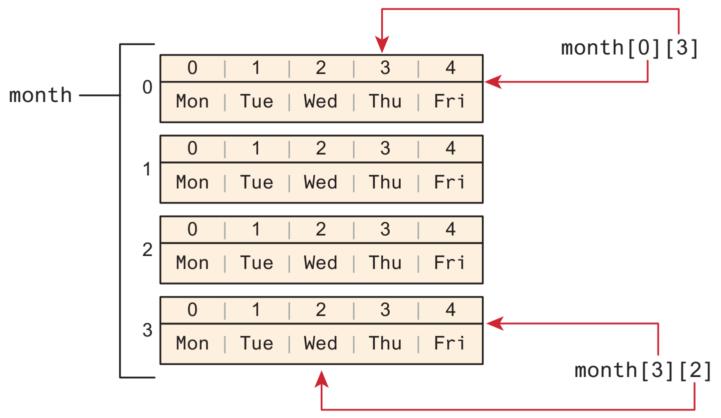
Some common features

- 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





## Objects

Object Creation—Object Literal Notation

**J**ava**S**cript **O**bject **N**otation (JSON) is a way of recursively defining objects in JS as `name:value` pairs, where `value` can be another JSON object or array

```
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;
```

## 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);
```

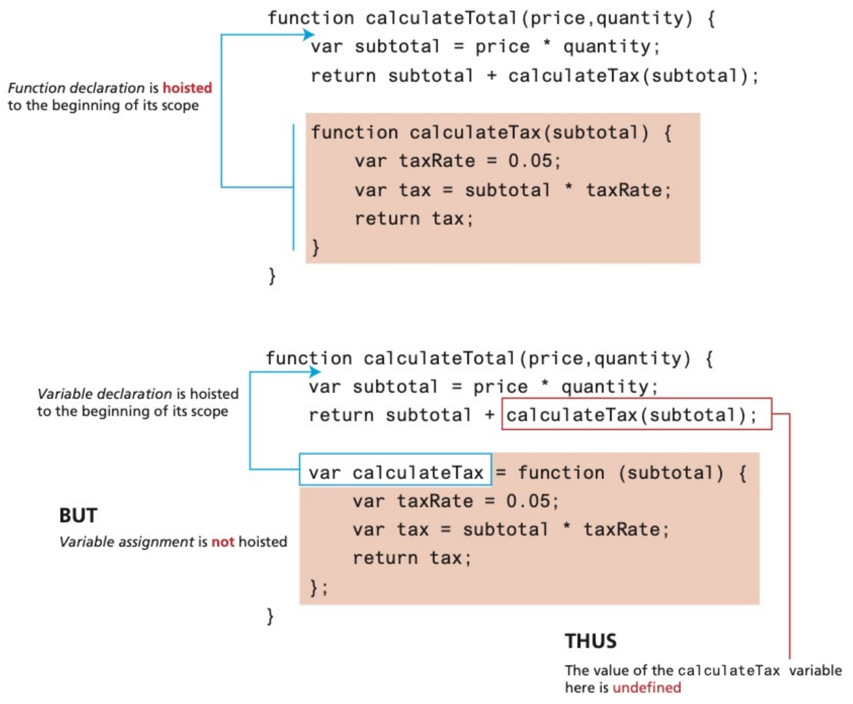
## Functions

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



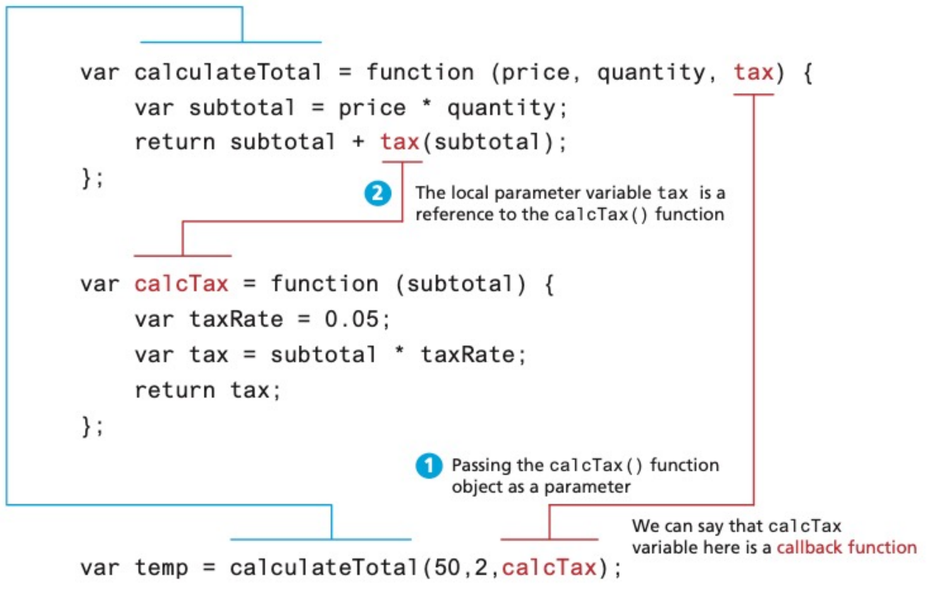
```
function calculateTotal(price,quantity) {
   var subtotal = price * quantity;
   return subtotal + calculateTax(subtotal);

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

```
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;
   };
}
```

*Function declaration* is **hoisted** to the beginning of its scope
*Variable declaration* is hoisted to the beginning of its scope
**BUT**
*Variable assignment* is **not** hoisted
**THUS**
The value of the calculateTax variable here is undefined

## Functions

Callback Functions

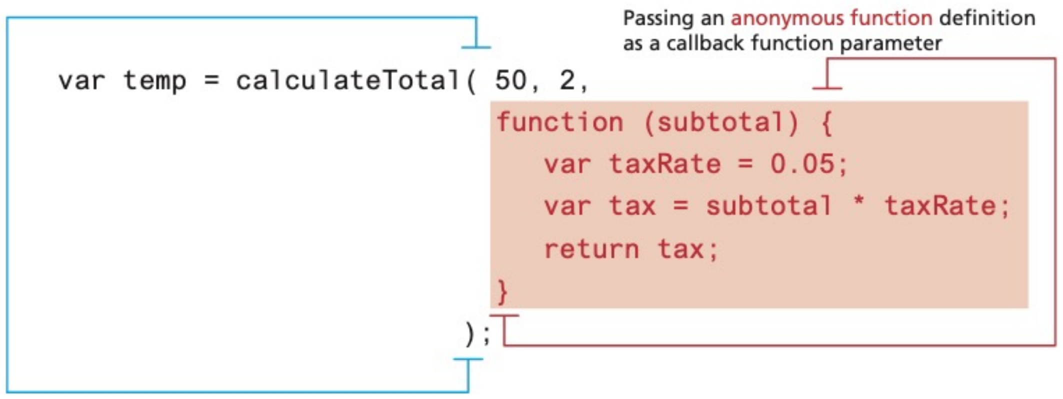


```
var calculateTotal = function (price, quantity, tax) {
    var subtotal = price * quantity;
    return subtotal + tax(subtotal);
};

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

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

2 The local parameter variable tax is a reference to the calcTax() function
1 Passing the calcTax() function object as a parameter
We can say that calcTax variable here is a callback function

## Functions

Callback Functions

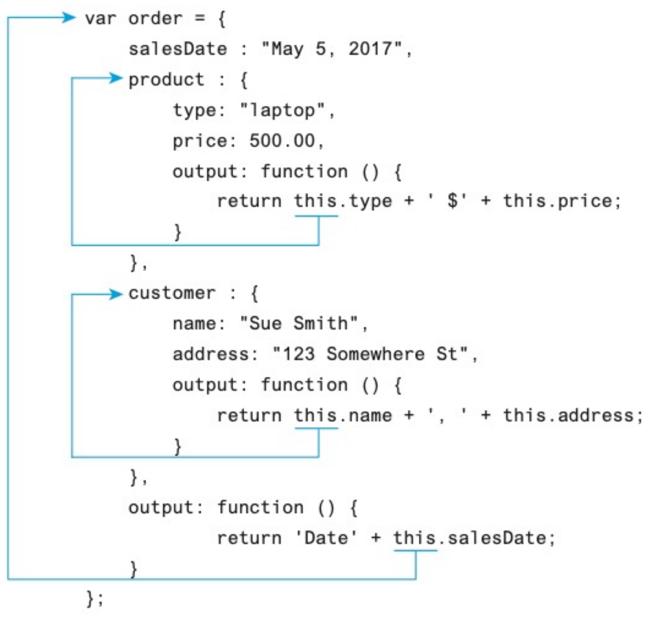


```
var temp = calculateTotal( 50, 2,
                 function (subtotal) {
                    var taxRate = 0.05;
                    var tax = subtotal * taxRate;
                    return tax;
                 }
              );
```

Passing an anonymous function definition as a callback function parameter

## Functions

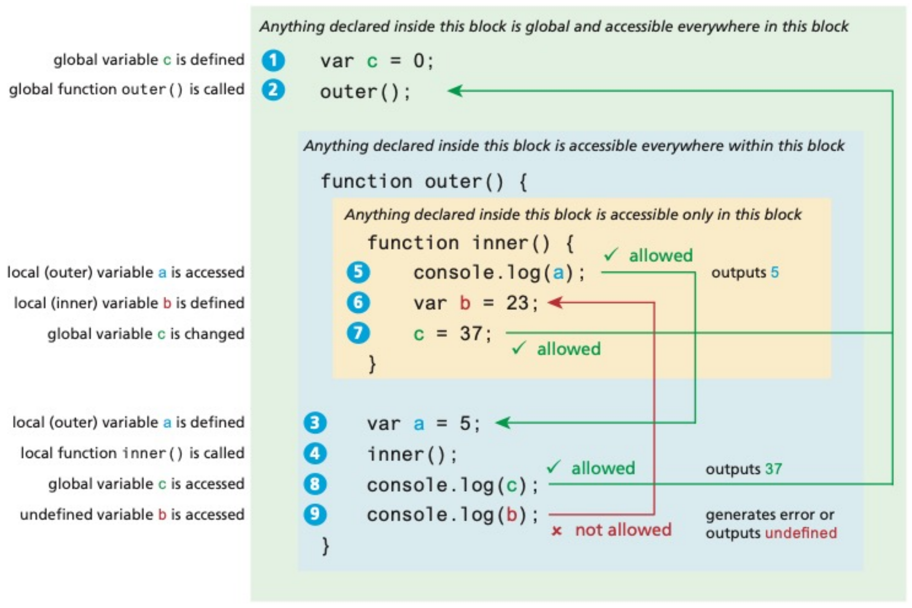
Objects and Functions Together



```
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

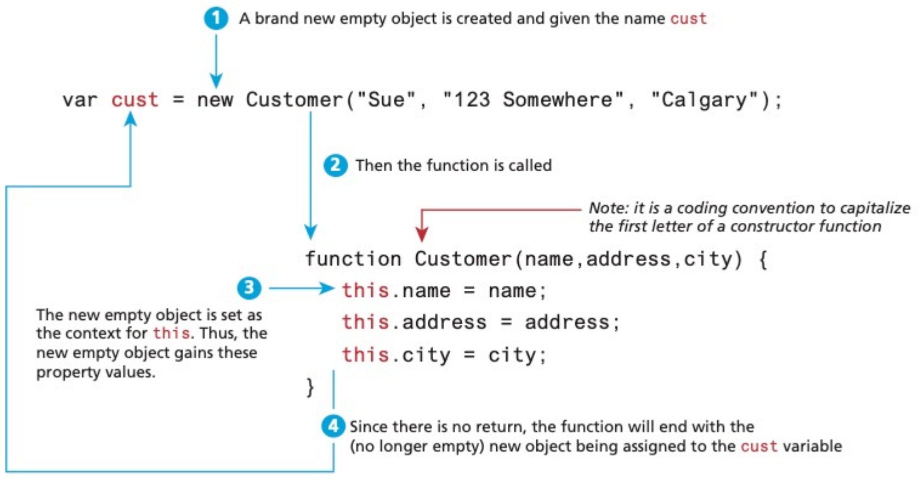


```
var c = 0;
outer();

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
5 local (outer) variable a is accessed
6 local (inner) variable b is defined
7 global variable c is changed
3 local (outer) variable a is defined
4 local function inner() is called
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

Function Constructors



1 A brand new empty object is created and given the name cust
2 Then the function is called
Note: it is a coding convention to capitalize the first letter of a constructor function
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

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

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

* The constructor function can also be created using the ES6 class notation

## Object Prototypes

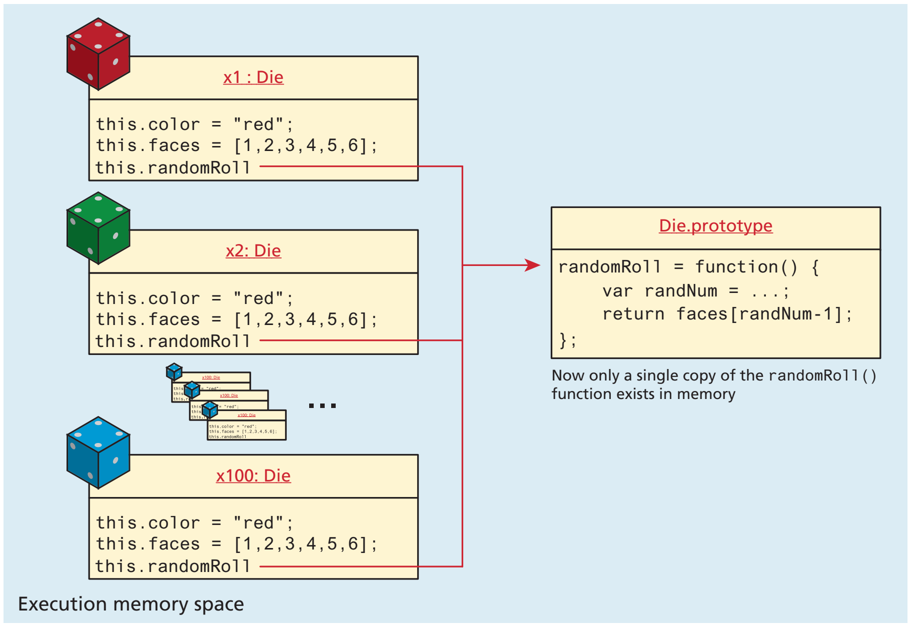
There’s a better way

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

Using Prototypes reduces duplication at run time.


x1 : Die
x2: Die
x100: Die
Die.prototype
Now only a single copy of the randomRoll() function exists in memory
Execution memory space

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

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

## Object Prototypes

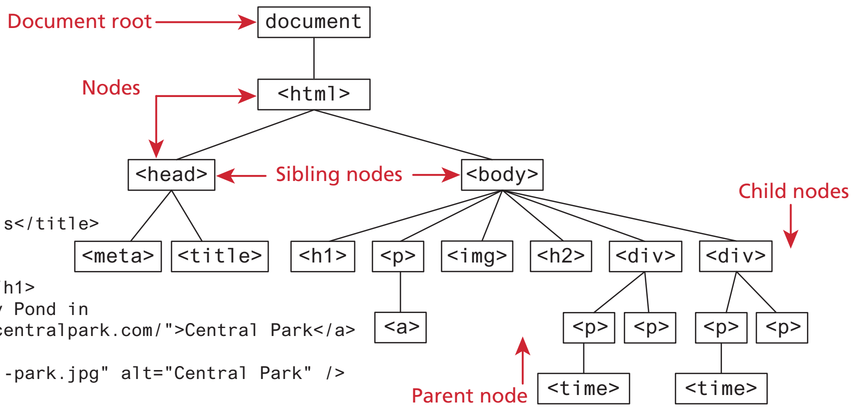
Using Prototypes to Extend Other Objects

```
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");
```

## The Document Object Model (DOM)

### Overview

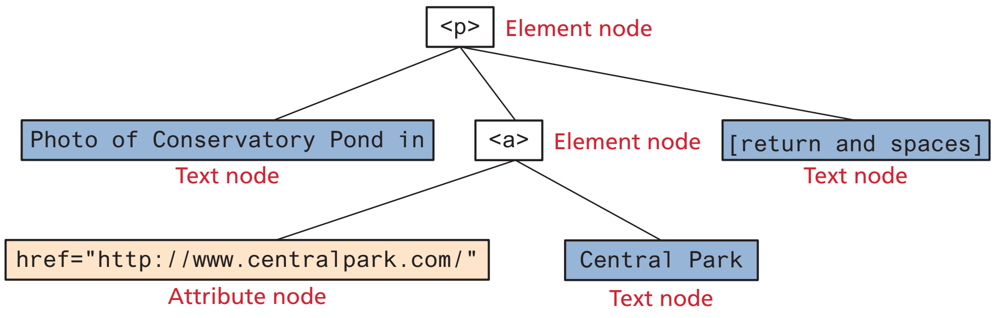
```
<html>
<head>
   <meta charset="utf-8">
   <title>Share Your Travels</title>
</head>
<body>
   <h1>Share Your Travels</h1>
   <p>Photo of Conservatory Pond in
      <a href="http://www.centralpark.com/">Central Park</a>
   </p>
   <img src="images/central-park.jpg" alt="Central Park" />
   <h2>Reviews</h2>
   <div id="latestComment">
      <p>By Ricardo on <time>2016-05-23</time></p>
      <p>Easy on the HDR buddy.</p>
   </div>
   <div>
      <p>By Susan on <time>2016-11-18</time></p>
      <p>I love Central Park.</p>
   </div>
</body>
</html>
```



## The Document Object Model (DOM)

### Nodes and NodeLists

```
<p>Photo of Conservatory Pond in
   <a href="http://www.centralpark.com/">Central Park</a>
</p>
```



## The Document Object Model (DOM)

Selection Methods

Classic

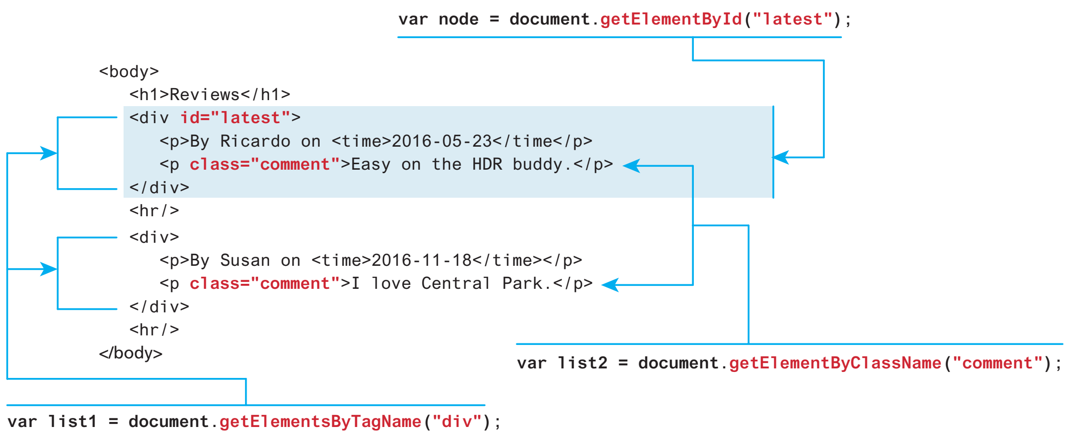
- getElementById()
- getElementsByTagName()
- getElementsByClassName()

Newer

- querySelector() and
- querySelectorAll()

## The Document Object Model (DOM)

Selection Methods



```
var node = document.getElementById("latest");
```

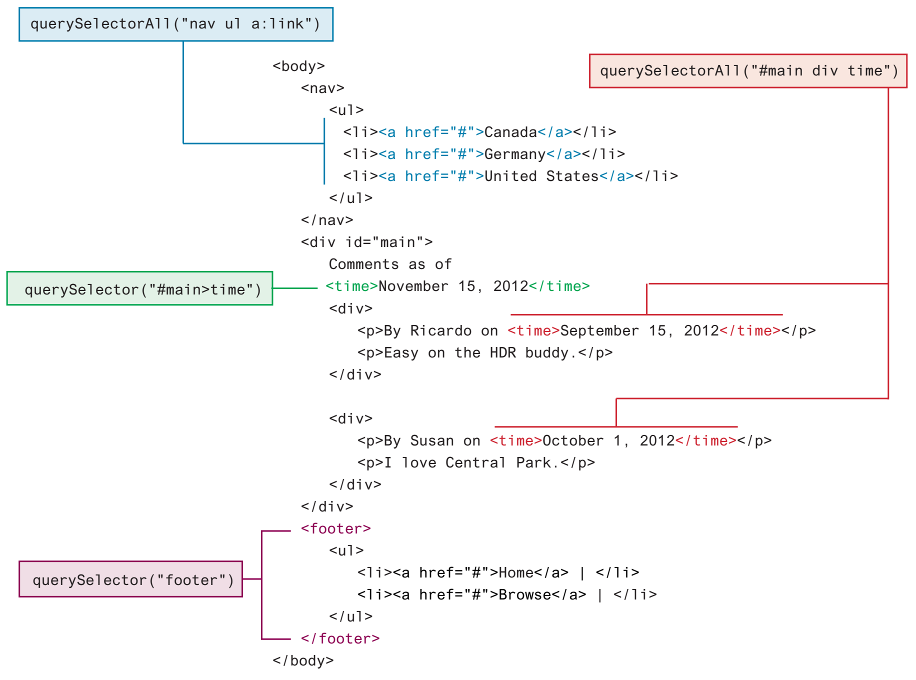
```
<body>
   <h1>Reviews</h1>
   <div id="latest">
      <p>By Ricardo on <time>2016-05-23</time</p>
      <p class="comment">Easy on the HDR buddy.</p>
   </div>
   <hr/>
   <div>
      <p>By Susan on <time>2016-11-18</time></p>
      <p class="comment">I love Central Park.</p>
   </div>
   <hr/>
</body>
```

```
var list2 = document.getElementByClassName("comment");
```

```
var list1 = document.getElementsByTagName("div");
```

## The Document Object Model (DOM)

**Query Selector**


querySelectorAll("nav ul a:link")
querySelectorAll("#main div time")
querySelector("#main>time")
querySelector("footer")

```
<body>
   <nav>
      <ul>
        <li><a href="#">Canada</a></li>
        <li><a href="#">Germany</a></li>
        <li><a href="#">United States</a></li>
      </ul>
   </nav>
   <div id="main">
      Comments as of
     <time>November 15, 2012</time>
      <div>
         <p>By Ricardo on <time>September 15, 2012</time></p>
         <p>Easy on the HDR buddy.</p>
      </div>

      <div>
         <p>By Susan on <time>October 1, 2012</time></p>
         <p>I love Central Park.</p>
      </div>
   </div>
   <footer>
      <ul>
         <li><a href="#">Home</a> | </li>
         <li><a href="#">Browse</a> | </li>
      </ul>
   </footer>
</body>
```

## The Document Object Model (DOM)

**Element Node Object**

Element Node object represents an HTML element in the hierarchy, contained between the opening <> and closing </> tags for this element. Every node has

- classList
- className
- Id
- innerHTML
- Style
- tagName

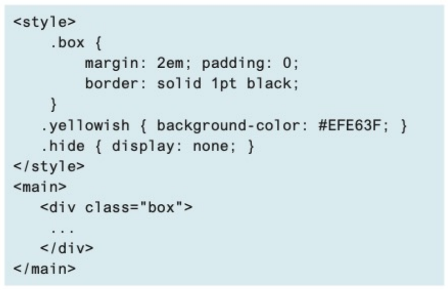
## The Document Object Model (DOM)

More common (not universal) properties

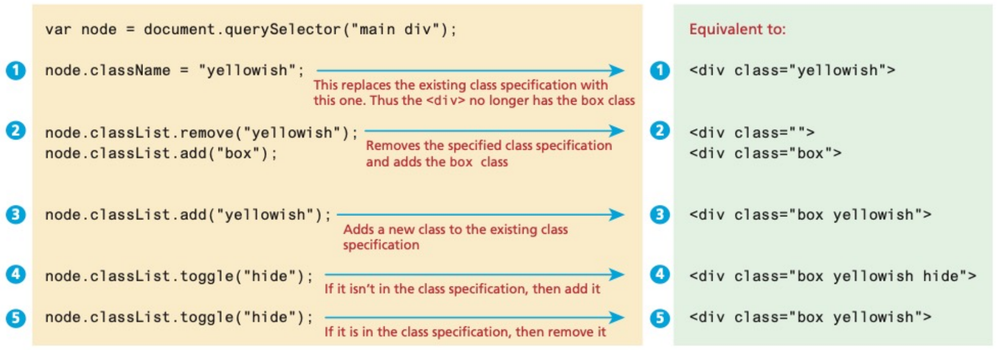
- href
- name
- src
- value

## Modifying the DOM

Changing an Element's Style



```
<style>
    .box {
        margin: 2em; padding: 0;
        border: solid 1pt black;
    }
   .yellowish { background-color: #EFE63F; }
   .hide { display: none; }
</style>
<main>
   <div class="box">
    ...
   </div>
</main>
```



```
var node = document.querySelector("main div");
```

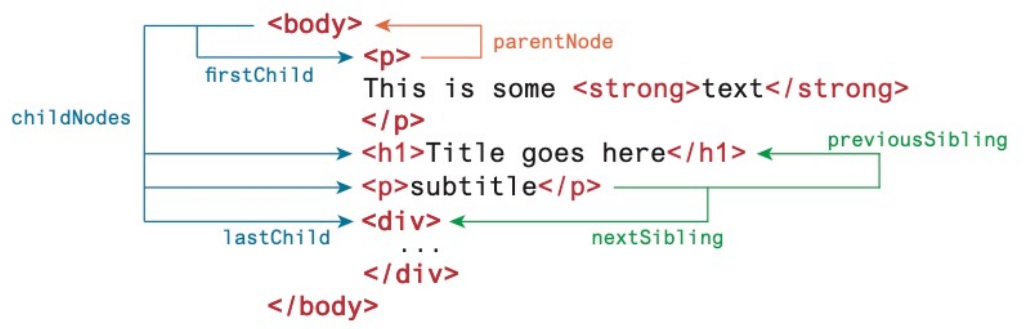
1. `node.className = "yellowish";` → This replaces the existing class specification with this one. Thus the `<div>` no longer has the box class
2. `node.classList.remove("yellowish");` `node.classList.add("box");` → Removes the specified class specification and adds the box class
3. `node.classList.add("yellowish");` → Adds a new class to the existing class specification
4. `node.classList.toggle("hide");` → If it isn't in the class specification, then add it
5. `node.classList.toggle("hide");` → If it is in the class specification, then remove it

Equivalent to:

1. `<div class="yellowish">`
2. `<div class="">` `<div class="box">`
3. `<div class="box yellowish">`
4. `<div class="box yellowish hide">`
5. `<div class="box yellowish">`

## Modifying the DOM

Meet the family



```
<body>
<p>
This is some <strong>text</strong>
</p>
<h1>Title goes here</h1>
<p>subtitle</p>
<div>
...
</div>
</body>
```

## Events

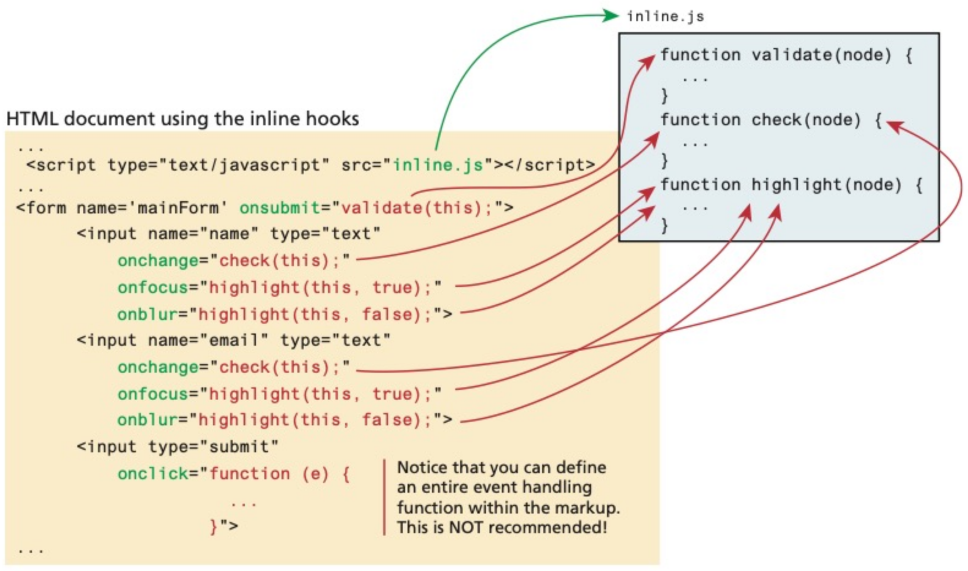
JavaScript event is an action that can be detected by JavaScript

- Many of them are initiated by user actions
- some are generated by the browser itself.

We say that an event is *triggered* and then it is *handled* by JavaScript functions

## Events

Event-Handling Approaches – Inline Hook


HTML document using the inline hooks

```
...
 <script type="text/javascript" src="inline.js"></script>
...
<form name='mainForm' onsubmit="validate(this);">
      <input name="name" type="text"
          onchange="check(this);"
          onfocus="highlight(this, true);"
          onblur="highlight(this, false);">
      <input name="email" type="text"
          onchange="check(this);"
          onfocus="highlight(this, true);"
          onblur="highlight(this, false);">
      <input type="submit"
          onclick="function (e) {
                        ...
                    }">
...
```

inline.js

```
function validate(node) {
   ...
}
function check(node) {
   ...
}
function highlight(node) {
   ...
}
```

Notice that you can define an entire event handling function within the markup. This is NOT recommended!

## Events

Event-Handling Approaches – Event Property Approach

```
var myButton = document.getElementById('example');

myButton.onclick = alert('some message');
```

## Events

Event-Handling Approaches – Event Listener Approach

```
var myButton = document.getElementById('example');

myButton.addEventListener('click', alert('some message'));

myButton.addEventListener('mouseout', funcName);
```

## Events

Event-Handling Approaches – Event Listener Approach (anon function)

```
myButton.addEventListener('click', function() {

       var d = new Date();

       alert("You clicked this on "+ d.toString());

});
```

## Events

Event Object

When an event is triggered, the browser will construct an event object  that contains information about the event.

```
div.addEventListener('click', function(e) {

        // find out where the user clicked

        var x = e.clientX;

        …
```

## Events

Event Object

- bubbles - Indicates whether the event bubbles up through the DOM
- cancelable - Indicates whether the event can be cancelled
- target - The object that generated (or dispatched) the event
- type - The type of the event (see next slides)

## Event Types

Mouse Events

- click - The mouse was clicked on an element
- dblclick - The mouse was double clicked on an element
- mousedown - The mouse was pressed down over an element
- mouseup - The mouse was released over an element
- mouseover - The mouse was moved (not clicked) over an element
- mouseout - The mouse was moved off of an element
- mousemove - The mouse was moved while over an element

## Event Types

Keyboard Events

- keydown - The user is pressing a key (this happens first)
- keypress - The user presses a key (this happens after keydown)
- keyup - The user releases a key that was down (this happens last)

## Event Types

Form Events

- Blur
- Change
- Focus
- Reset
- select
- Submit

## Event Types

Frame Events

- abort - An object was stopped from loading
- error - An object or image did not properly load
- load - When a document or object has been loaded
- resize - The document view was resized
- scroll - The document view was scrolled
- unload - The document has unloaded

## Forms

**Submitting Forms**

We can use JavaScript to submit a form by selecting the form object from the DOM and invoking the `submit()` function.

```
var formExample = document.getElementById("loginForm");

formExample.submit();
```