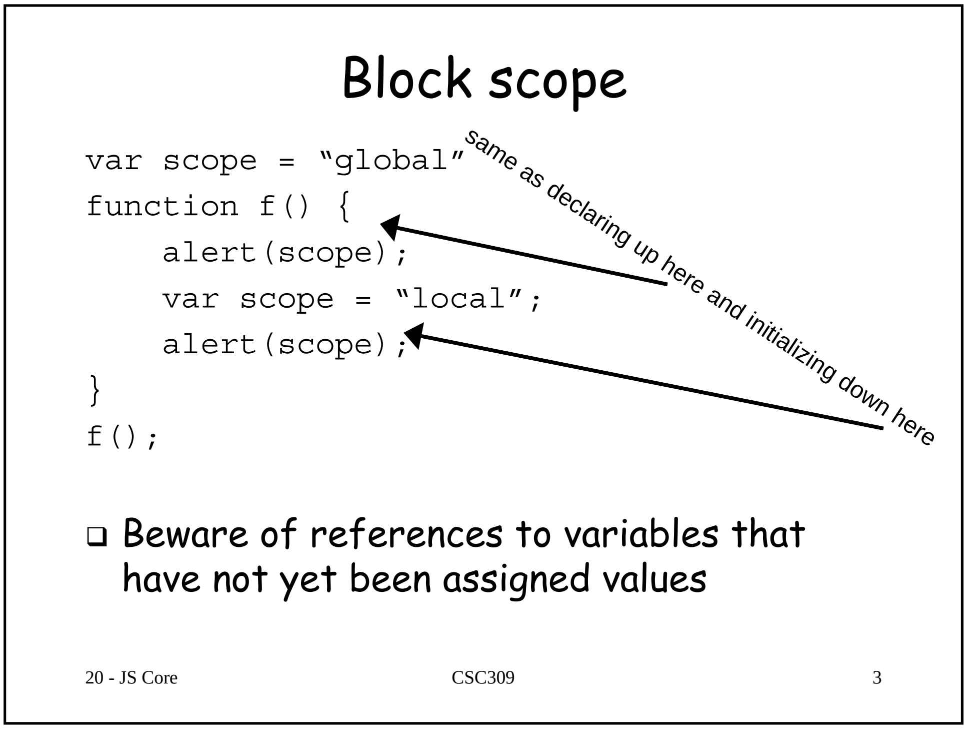
The JavaScript Language

- untyped
  - different than Java or C …
  - a variable can hold any type of value:
    - number (8-byte IEEE fp)
    - string
    - boolean
    - function (first-class data type)
    - object
    - Array (elements can be of mixed types)
  - … and can hold values of different types at different times during execution

JavaScript

- Variable declaration
  - var i = 12, msg = “hello”;
- If you omit a variable declaration:
  - automatically declared at global scope
- no block-level scope

```
function test() {
    if( 1 == 1 ) {
        var j = 12;
    }
    document.write(j);
}
```

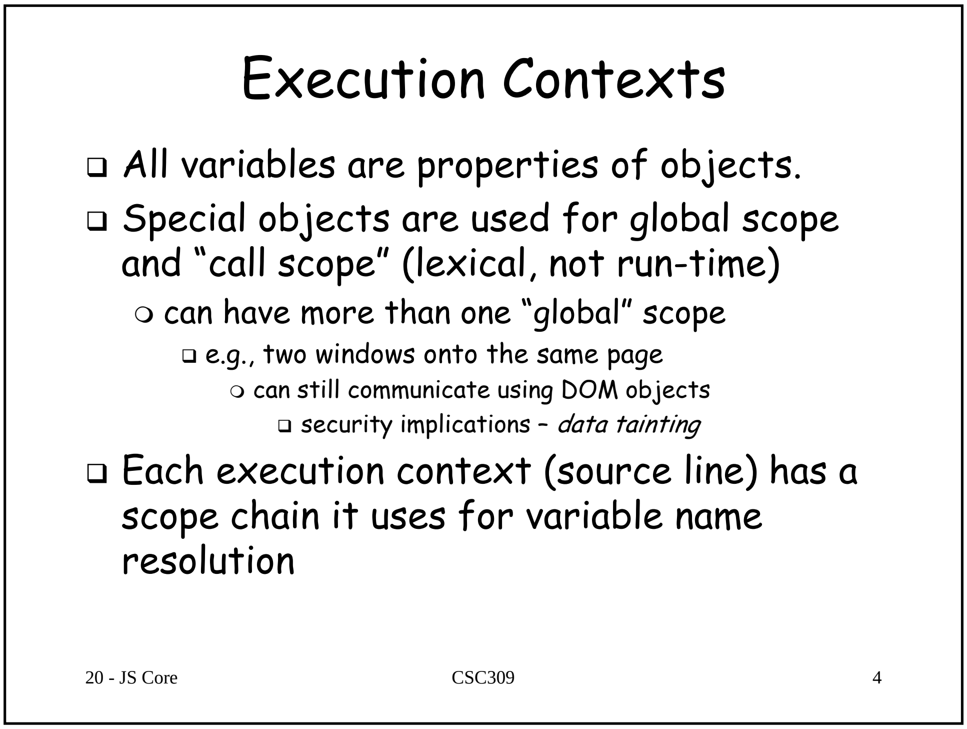


Block scope

```
var scope = “global”
function f() {
    alert(scope);
    var scope = “local”;
    alert(scope);
}
f();
```

same as declaring up here and initializing down here

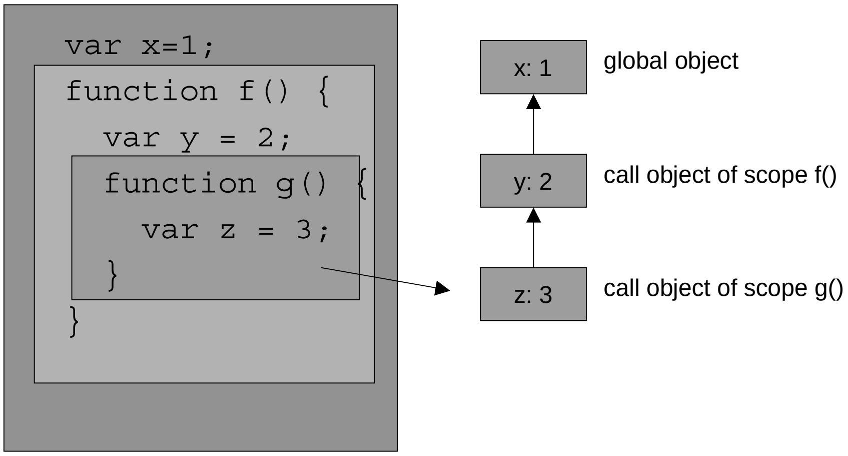
- ❑ Beware of references to variables that have not yet been assigned values



Execution Contexts

- ❑ All variables are properties of objects.
- ❑ Special objects are used for global scope and “call scope” (lexical, not run-time)
  - ❍ can have more than one “global” scope
    - ❑ e.g., two windows onto the same page
      - ❍ can still communicate using DOM objects
        - ❑ security implications – *data tainting*
- ❑ Each execution context (source line) has a scope chain it uses for variable name resolution

Scope Chain



```
var x=1;
function f() {
  var y = 2;
  function g() {
    var z = 3;
  }
}
```

x: 1 — global object

y: 2 — call object of scope f()

z: 3 — call object of scope g()

Implications of Lexical Scoping

```
function makefunc(x) {
  return function() { return x; }
}
a = [makefunc(0), makefunc(1), makefunc(2)];

alert(a[0]());     // displays 0
alert(a[1]());     // displays 1
alert(a[2]());     // displays 2
```

- A function reference is actually a reference to a “Closure” that has 2 properties:
  - a[0].__proto__: the function reference itself
  - a[0].__parent__: the scope object

Semicolons

- ❑ If a newline terminates a statement, then a semicolon is inserted for you automatically

```
a = 3;          a = 3         a = 3; b = 4
b = 4;          b = 4

a = 3           return        return;
b =             true;         true;
   4
```

(shudder…)

Literals

- ❑ Usual number, string, boolean literals
- ❑ Function literals (“lambda” functions)
  - ❍ `var square = function(x) { return x*x; }`
- ❑ Object literals
  - ❍ `var point = { x:2, y:4 };`
- ❑ Array literals
  - ❍ `var a = [1,“foo”,,true];`
- ❑ Regular expression literals
  - ❍ `var a = /[1-9][0-9]*/;`
    - ❑ creates object of type RegExp

Objects

- The language has no *class* construct
  - Objects are most like associative arrays

```
var point = new Object();
point.x = 2;  // equivalent to point[“x”] = 2
point[“y”] = 3;  // equivalent to point.y = 2
alert(point.x); alert(point.y);
for(var i in obj) document.write(i + “\n”)
```

- However, note a Javascript Object's definition is determined at *run time*. Unlike C++ or Java, it is possible to dynamically add new properties or methods and to change the binding of methods at runtime

Object Constructors

```
function Rectangle_area() {
    return this.width * this.heigth;
} /* “this” is a keyword used within a
  method to refer to the current object */

function Rectangle(w,h) {
    this.width = w;
    this.height = h;
    this.area = Rectangle_area;
}

var rec = new Rectangle(2,4);
document.write(rec.area());
```

Prototype Inheritance Hack

```
function Circle(x,y,r) {
    this.x = x; this.y = y; this.r = r;
}

Circle.prototype.pi = 3.1415926534;
Circle.prototype.area = function() {
    return this.pi * this.r * this.r;
}
```

- Will look up the property in the prototype object if it's not defined in the object itself.
- Writing will create a local copy if property is defined in the prototype object; useful for creating instances that differ from the standard

Default Methods

- contructor
  - refers to the constructor function used to create an object,
  - e.g. `function Circle(x,y,r) {`

    `this.x = x; this.y = y; this.r = r;}`
- toString()
  - automatically called for conversions to string
- toValue()
  - automatically called for conversions to numbers

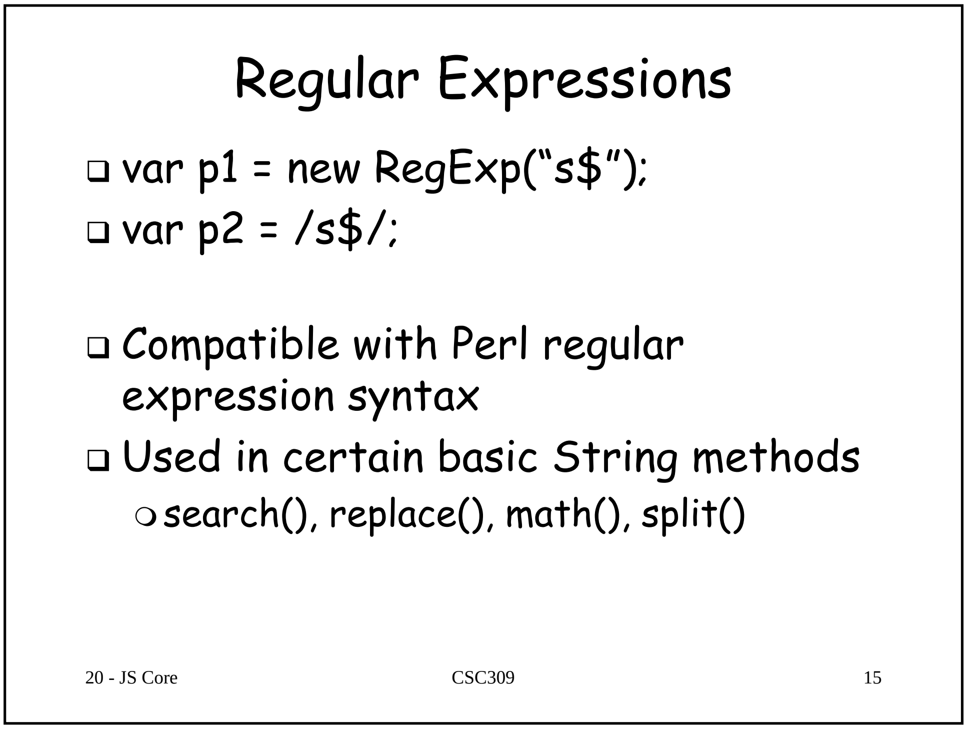
Arrays

```
var a = new Array();      // empty array
var b = new Array("dog", 3, 8.4);
var c = new Array(10); // array of size 10
var d = [2, 5, 'a', 'b'];

c[15] = "hello";    // implicit extension
```

Array Properties and methods

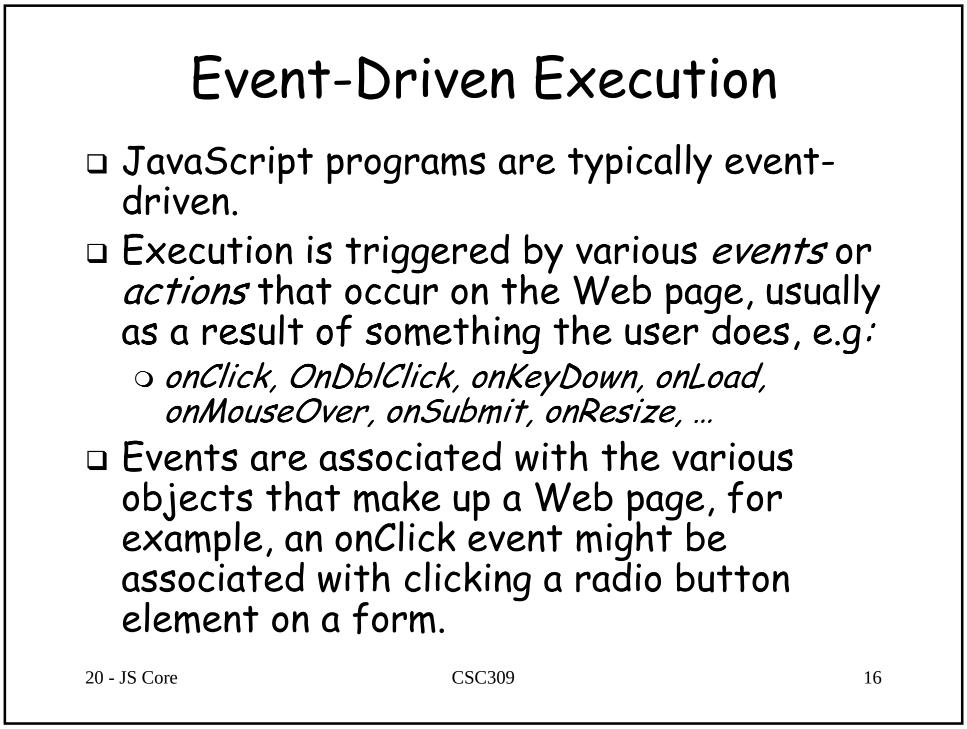
- ❑ length
- ❑ join()
- ❑ reverse()
- ❑ sort()
- ❑ concat()
- ❑ slice()
- ❑ splice()
- ❑ push() / pop()
- ❑ shift() / unshift()
- ❑ toString()



Regular Expressions

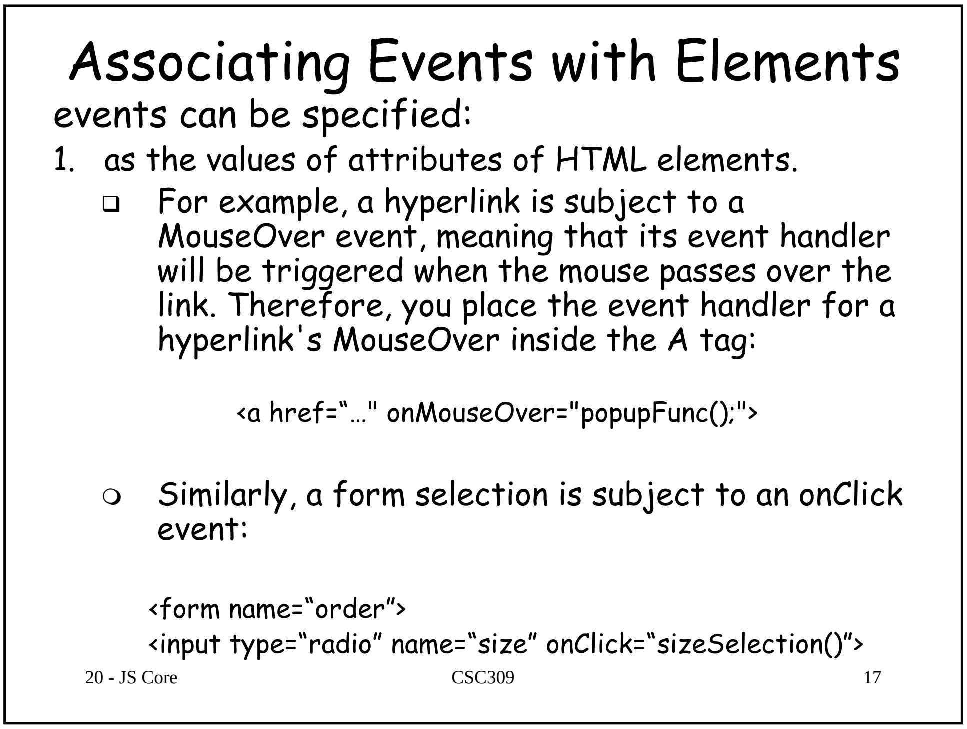
- ❑ var p1 = new RegExp(“s$”);
- ❑ var p2 = /s$/;

- ❑ Compatible with Perl regular expression syntax
- ❑ Used in certain basic String methods
  - ❍ search(), replace(), math(), split()



Event-Driven Execution

- ❑ JavaScript programs are typically event-driven.
- ❑ Execution is triggered by various *events* or *actions* that occur on the Web page, usually as a result of something the user does, e.g*:*
  - ❍ *onClick, OnDblClick, onKeyDown, onLoad, onMouseOver, onSubmit, onResize, …*
- ❑ Events are associated with the various objects that make up a Web page, for example, an onClick event might be associated with clicking a radio button element on a form.



Associating Events with Elements

events can be specified:

1. as the values of attributes of HTML elements.
   - ❑ For example, a hyperlink is subject to a MouseOver event, meaning that its event handler will be triggered when the mouse passes over the link. Therefore, you place the event handler for a hyperlink's MouseOver inside the A tag:

```
<a href=“…" onMouseOver="popupFunc();">
```

   - ❍ Similarly, a form selection is subject to an onClick event:

```
<form name=“order”>
<input type=“radio” name=“size” onClick=“sizeSelection()”>
```