SIU CARBONDALE
SOUTHERN ILLINOIS UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

CS 350/491
INTRO TO JAVASCRIPT

Textbook Chapter 4

Overview of JavaScript: Origins

- Originally developed by Brendan Eich at **Netscape** as **LiveScript**
  - Joint venture with Sun Microsystems later in 1995
- **Standardized as** ECMA-262 by the *European Computer Manufacturers Association*
- Broadly supported by all browsers: etscape, Mozilla, Internet Explore, etc.
- We call JavaScript code **script**, not program
- This course covers ***client-side* scripting** (not programming)

Overview of JavaScript: the Language

- **Core: the heart of the language**
- **Client-side**
  - **Library of objects supporting browser control and user interaction**
- **Server-side** (not recommended nowadays, not to be discussed)
  - **Library of objects that support use in web servers**
- **User interactions through forms mostly**
- **The Document Object Model makes it possible to support dynamic HTML documents with JavaScript**
- **Much of what we will do with JavaScript is event-driven computation (Chapter 5) when JavaScript is**

Java and JavaScript

- **Differences with Java**
  - JavaScript has a different object model from Java
  - JavaScript is not strongly typed, nor object-oriented
    - Missing Key OO features: class-based inheritance and polymorphism
    - JS supports limited prototype-based inheritance
    - All objects derived from the ***Object***
- **Like Java and C++**
  - JS objects are collections of properties
  - objects are accessed through ***references***
- **Java 1.6 has support for scripting**
  - http://java.sun.com/javase/6/docs/technotes/guides/scripting/index.html
- **Mozilla Rhino is an implementation of JavaScript in Java**
  - http://www.mozilla.org/rhino/
- Don't get confused with Java!!

Uses of JavaScript

- **Provide (better) *alternative* to server-side programming**
  - **Servers are often overloaded**
  - **Client processing has quicker reaction time**
- **JavaScript can work with forms**
- **JavaScript can interact with the internal model of the web page – DOM (Document Object Model)**
- **JavaScript is used to provide more complex user interface than plain forms with HTML/CSS can provide**

General Syntactic Characteristics

- **All JavaScript scripts are embedded in HTML documents**
  - **Either directly, as in**

```
<script type = "text/javaScript">
-- JavaScript script –
</script>
```

  - **Or indirectly, as a file specified in the `src` attribute of `<script>`, as in**

```
<script type = "text/javaScript"
        src = "myScript.js">
</script>
```

- ***Language Basics*:**
  - *Identifier form*: begin with a letter or underscore, followed by any number of letters, underscores, and digits
    - Case sensitive
  - *25 reserved words*, plus future reserved words
  - *Comments*: both `//` and `/* … */`

General Syntactic Characteristics (continued)

- Scripts are usually hidden from browsers that do not include JavaScript interpreters by putting them in special comments

```
<!--
-- JavaScript script –
//-->
```

  - Also hides it from HTML validators

- Semicolons can be a problem
  - They are “somewhat” optional
  - Hare to always to judge when the end of the line can be the end of a statement – JavaScript puts a semicolon there (e.g. later)

General Syntactic Characteristics (continued)

- Identifiers
  - **Start with $, _, or a letter**
  - **Continue with $, _, letter or digit**
  - (only) **Identifiers are case sensitive!**
- Reserved words (not case sensitive)
  - **Such as null, true, false, etc**
- Comments *within* JavaScript code:
  - //
  - /* … */

Event-Driven Computation

- **User actions, such as mouse *clicks* and key *presses*, are referred to as *events***
- **The main task of most JavaScript programs is to *respond to events***
- **For example, a JavaScript program is often (but not only) used to validate data in a form before it is submitted to a server**
  - ***Caution:*** It is important that crucial validation still need be done on the server as it is relatively easy to bypass client-side controls
  - For example, a user might create a copy of a web page but remove all the validation code.

HTML/JavaScript Documents

- JavaScript code is *not standalone program*, but typically as embedded scripts in an XHTML document, subject to browser interpretation.
- Proper strategies must be used to ‘protect’ the JavaScript from the browser behavior
  - For example, comparisons present a problem since < and > are used to mark tags in HTML
  - Therefore (as another reason), JavaScript code is normally enclosed in XHTML comments

JavaScript Objects

- Objects are collections of *properties*
- Properties are either *data properties* or *method properties*
- Data properties are either *primitive* values or *references* to other objects
- Primitive values are often implemented directly in hardware
- The special *Object* object is the ancestor of all objects in a JavaScript program
  - Let's call this special object "The big 'O' object".
  - This Object itself has no data properties, but several method properties

Statement Syntax

- Statements can be terminated with a semicolon;
- However, the interpreter will insert the semicolon if missing at the end of a line and the statement seems to be complete
- But this can lead to a real problem, e.g.,

```
return
x;
```

- If a statement must be continued to a new line, make sure that the first line does not make a complete statement by itself!

Primitive Types

- **Five primitive types**
  - Number
  - String
  - Boolean
  - Undefined
  - Null
- **Three predefined wrapper objects**
  - The three wrapper objects named the same as their corresponding primitive type names: Number, String, and Boolean (Is this confusing?)
  - As place holders for methods and properties relevant to the primitive values
  - As a convenient way to apply OO features to primitive values, e.g.,

```
Var num = 6;
Var str = num.toString();  // num  is implicitly coerced to a String object first!
```

  - Primitive values are often *coerced* to objects *as necessary*, and vice-versa
    - So users (you) can ignore this (though essential) difference

Primitive and Object Storage

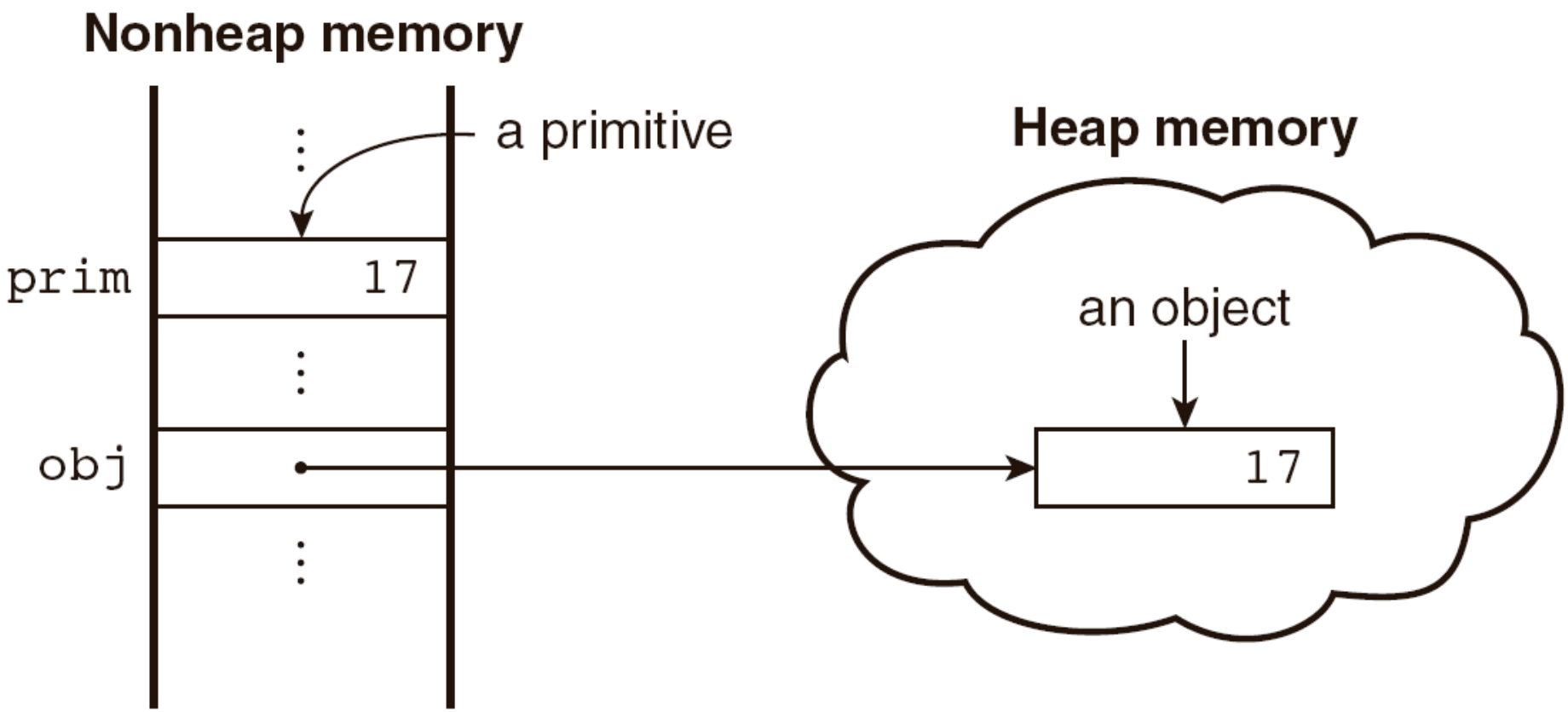


Figure 4.1 Primitives and objects

Numeric and String Literals

- **Numeric values are represented internally as double-precision, floating-point values**
  - Numeric literals can be either integer or float
  - Float values may have a *decimal* and/or *exponent*
- **A String literal is delimited by either single or double quotes**
  - There is no difference between single and double quotes
  - Certain characters need be *escaped* in strings
    - \' or \" --to use the quote in a string delimited by the same quotes
    - \\ -- to use a literal backslash
  - The empty string '' or "" has no characters

Other Primitive Types

- **Boolean**
  - **Two values: true and false**
- **Null**
  - **As a primitive value, null, simply means "no value" (no content).**
  - `'null'` is the reserved word for this special value
  - null is as a special **object**: *typeof null* returns 'object'
  - Using ***undeclared&unassigned*** variable typically results in the `null` value
  - And doing so usually causes an error!!!!
- **Undefined**
  - **The undefined primitive value indicates *the nonexistence status* that a variable has not been assigned a value,**
  - However, undefined is not a reserved word (just indicates such a conception).
  - *undefined* indicates **more "uncertainness"**

Declaring Variables

- **JavaScript is *dynamically typed*, that is, variables do not have declared types and do not have to be declared!**
  - A variable can hold different types of values at different times during program execution
- **A variable is explicitly declared using the keyword var**

```
var counter,
index,
pi = 3.14159265,
quarterback = "Elway",
stop_flag = true;
```

- **An *undeclared* variable is one that is declared using var**

Numeric Operators

- **Standard arithmetic**
  - `+ * - / %`
- **Increment and decrement**
  - -- ++
  - **Increment and decrement <u>differ in effect</u> when used <u>before</u> and <u>after</u> a variable**
  - **Assume that *a* has the value 7 initially**
  - **(++a) * 3 will have the value 24**
  - **But (a++) * 3 has the value 21**
  - **In either case, *a* has the final value 8**

Precedence of Operators

Operators	Associativity
`++, --, unary +, unary -`	Right
`*, /, %`	Left
`+, -`	Left
`>, <, >= ,<=`	Left
`==, !=`	Left
`===,!==`	Left
`&&`	Left
`||`	Left
`=, +=, -=, *=, /=, &&=, ||=, %=`	Right

Example of Precedence

```
var a = 2,
b = 4,
c,
d;
c = 3 + a * b;
// * evaluates first, so c is now 11 (not24)
d = b / a / 2;
// '/' associates left, so d is now 1 (not4)
```

String Catenation

- **The operation + is the *string catenation* operation**
- **In many cases, other types are *automatically converted* to string before concatenating**

The `Number` Wrapper Object

- Properties
  - `MAX_VALUE` – the largest representable number
  - `MIN_VALUE`
  - `NaN`
  - `POSITIVE_INFINITY`
  - `NEGATIVE_INFINITY`
  - `PI`
- Operations resulting in errors return *`NaN`*
  - `isNaN(a)` yields true if *a* is `NaN`
    - `NaN` is not equal to itself – don't compare *nothing* with *nothing*!
    - Don't get consumed with `null`!
  - The *toString* method of `Number` converts a number to a string

The `Math` Wrapper Object – Very Useful!

- **An additional wrapper object**
- **Provides a collection of properties and methods useful for numeric values:**
  - **Not available from the Number wrapper object**
- **This includes the *trigonometric* functions such as `sin`, `cos`, `tan`, `atan`, `max`, `min`, etc.**
- **When used, these methods must be qualified, as in `Math.sin(x)`**

Implicit Type Conversion

- **JavaScript attempts to convert values in order to be able to perform operations**
- **"August" + 1977 causes the number to be converted to string and a concatenation then to be performed**
- **7 * "3" causes the string to be converted to a number and a multiplication to be performed**
- ***null* is converted to 0 in a numeric context, *undefined* to *NaN***
  - **Note null** has a meaning but **NaN** is meaningless!
- ***0* is interpreted as a Boolean *false*, all other numbers are interpreted as *true***
- **The empty string is interpreted as a Boolean *false*, all other strings (including "0" and " ") as Boolean *true***
- ***undefined*, *NaN* and *null* are all interpreted as Boolean *false***

Explicit Type Conversion

- **Explicit conversion of string to number**
  - **Number(aString)**
  - **Same as "aString – 0" (note "aString + 0" won't work, why?)**
  - **The string must begin with a number, followed by space or end of string**
    - **Note** not every string can be transformed into number!
- ***parseInt* and *parseFloat* convert the beginning of a string and do not cause an error if a non-space follows the numeric part**
  - These two string functions are not String functions (!!)
    - So do not make calls like String.parseInt('5') which is **wrong**!!

The `String` Wrapper Object

- **Only one (data) property: length**
  - Note this is not a method!
- **Character positions in strings begin at index 0**

Methods in the String Wrapper Object

Method	Parameters	Result
charAt	A number	Returns the character in the String object that is at the specified position
indexOf	One-character string	Returns the position in the String object of the parameter
substring	Two numbers	Returns the substring of the String object from the first parameter position to the second
toLowerCase	None	Converts any uppercase letters in the string to lowercase
toUpperCase	None	Converts any lowercase letters in the string to uppercase

The `typeof` Operator

- **Returns "number" or "string" or "boolean" for *primitive* types respectively**
- **Returns "object" for an object or null**
  - **null** is an object!!

- **Two syntactic forms**
  - `typeof x`
  - `typeof(x)`

Assignment Statements

- Plain assignment indicated by =
- Compound assignment with
- `+= -= /= *= %= …`
- a += 7 means the same as
- a = a + 7

The Date Object

- A *Date* object represents a ***time stamp***, that is, **a point in time**
- A Date object is created with the new operator, e.g.,
  - var now= new Date();
  - This creates a Date object for the time at which it was created

The Date Object's Methods

toLocaleString	A string of the Date information
getDate	The day of the month
getMonth	The month of the year, as a number in the range of 0 to 11
getDay	The day of the week, as a number in the range of 0 to 6
getFullYear	The year
getTime	The number of milliseconds since January 1, 1970
getHours	The number of the hour, as a number in the range of 0 to 23
getMinutes	The number of the minute, as a number in the range of 0 to 59
getSeconds	The number of the second, as a number in the range of 0 to 59
getMilliseconds	The number of the millisecond, as a number in the range of 0 to 999

Date

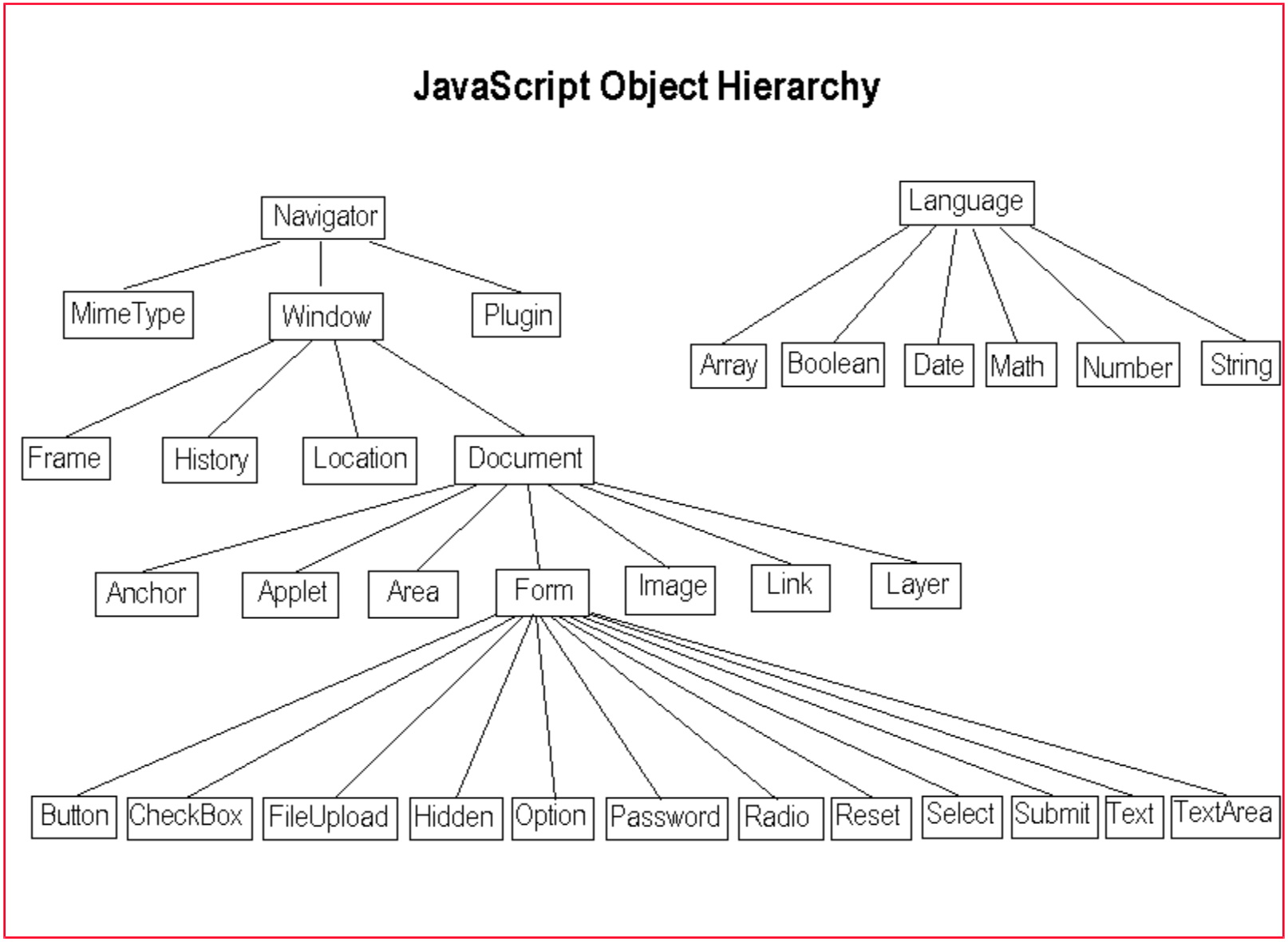
```
1   <!DOCTYPE html>
2   <html>
3   <body>
4   <script>
5       var d=new Date();
6       var year=d.getFullYear();
7       var month=d.getMonth();
8       var day=d.getDay();
9       var hours=d.getHours();
10      var min=d.getMinutes();
11      document.write("</br> "+ d);
12      document.write("</br> Year: " +year);
13      document.write("</br> Month: " +month);
14      document.write("</br> Day: " +day);
15      document.write("</br> hours: " +hours);
16      document.write("</br> Minutes: " +min);
17  </script>
18  </body>
19  </html>
```

Mon Apr 21 2014 10:18:31 GMT-0500 (Central Daylight Time)
Year: 2014
Month: 3
Day: 1
hours: 10
Minutes: 18

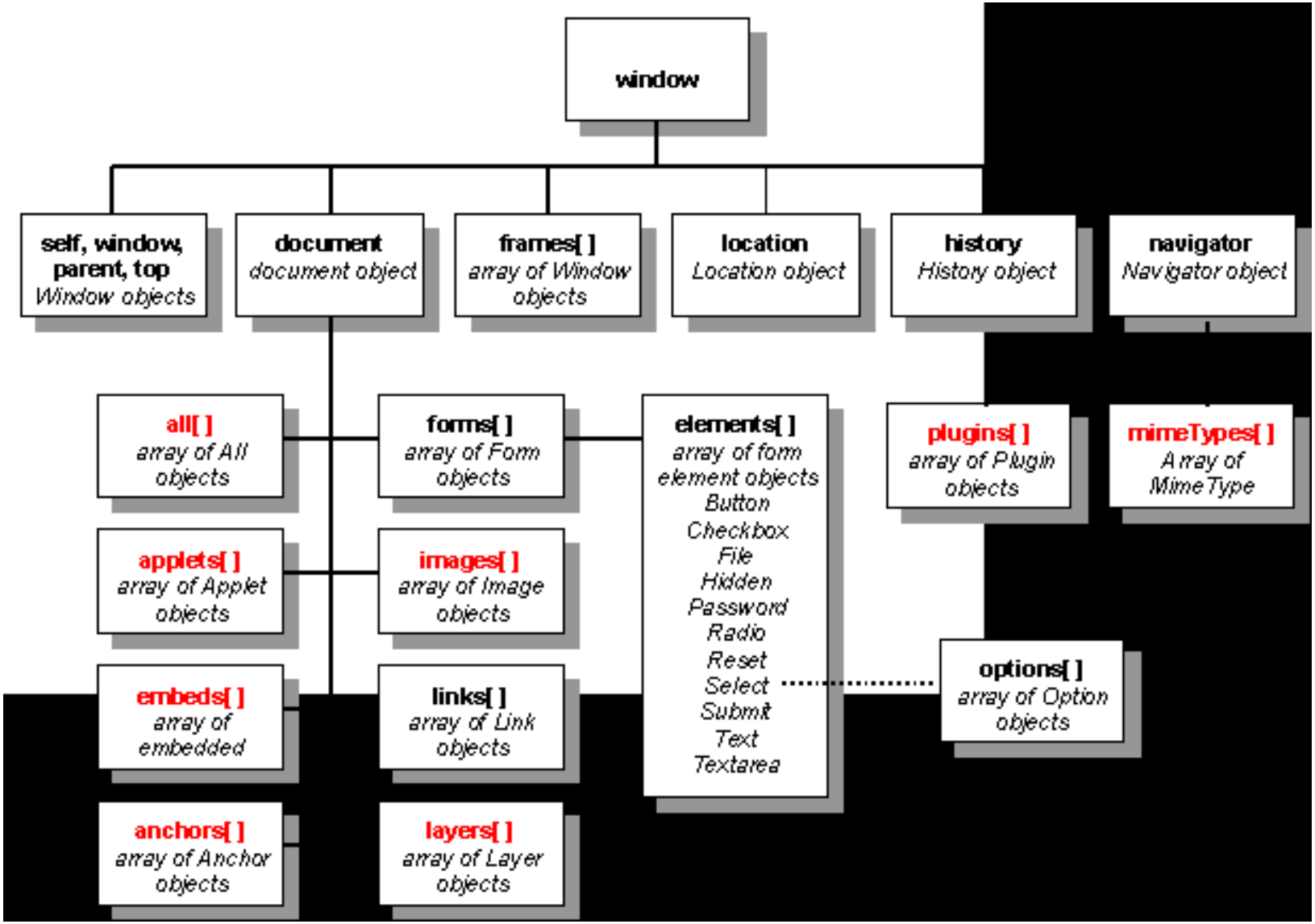
Window and Document

- **The Window object represents the window in which the document containing the script is being displayed**
- **The Document object represents the document being displayed using DOM**
- **Window (at the top of the hierarchy) has the property,**
  - `document` **that refers to the Document object being displayed**
- **The Window object is the *default* object for JavaScript,** so properties and methods of the Window object may be used without qualifying with the class name
- **JavaScript Objet Hierarchy (next slide)**

JavaScript Object Hierarchy (1)



JavaScript Object Hierarchy (2)



Screen Output and Keyboard Input

- Standard output for JavaScript embedded in a web page is the browser window displaying the page
- The *write* (and *writeln*) method of the *Document* object writes its parameters to the browser window
  - Both append to the document currently displayed
- The output is interpreted as HTML by the browser
- Therefore, if a line break is needed in the output, interpolate <br/> into the output

Writing to The HTML Document (Output)

```html
<html>
<body>
<h1>Output</h1>
<script>
    var age = 35;
</script>
<p>The age is <script>document.write(age)</script></p>
</body>
</html>
```

Output

The age is **35**

```html
<html>
<body>
<h1>Output</h1>
<script>
    var age = 35;
    document.write("<p>The age is:<b> " +age+"</b></p>");
</script>
</body>
</html>
```

Output

The age is **35**

Concatenation

- var person = “My”;
- var information = “Age”;
- var age = 35;

My Age is 35

- document.write(person + “ “ + information + “ is “ + age);

The alert Method

- **The alert method opens a dialog box with a message**
- **The output of the alert is *not* HTML (for *convenience*) so you must use the new line character "\n" to break a line.**
- ***alert* does not return any value**

```
alert("The sum is:" + sum + "\n");
```

The confirm Method

- The confirm methods displays a message provided as a parameter
  - The confirm dialog has two buttons: OK and Cancel
- If the user presses OK, true is returned by the method
- If the user presses Cancel, false is returned

```
var question =
   confirm("Do you want to continue this download?");
```

JS-Demo : name2.html

- **Join 2 names**

```
1  <html>
2  <body>
3  <P> name2.html: joins 2 names</p>
4  <script>
5      var firstname, secondname, result;
6      firstname = prompt("Enter first name");
7      secondname = prompt("Enter last name");
8      result = firstname + " " + secondname // + means "join" here
9      alert("hello, " + result); // and here
10 </script>
11 </body>
12 <html>
```

JS-Demo:demo .html

```html
<html>
<body>
<h1>Output_Example</h1>
<script>document.write ('<style> table{border:1px solid;} tr{color:red;} </style>')</script>
    <table>
        <tr>
        <script>document.write('<td>')</script>
        CS-221-001 Internet and Mobile Computing
        <script>document.write('</td>')</script>
        </tr>
        <tr>
        <script>document.write('<td>')</script>
        CS-350-491 Web Application Development
        <script>document.write('</td>')</script>
        </tr>
    </table>

</body>

</html>
```

The prompt Method

- This method displays its string argument in a dialog box
  - A second argument provides a default content for the user entry area
- The dialog box has an area for the user to enter text
- The method returns a string: the text entered by the user

- Example:

```
name = prompt("What is your name?", "");
```

Control Statements

- A ***compound statement*** in JavaScript is a sequence of 0 or more statements enclosed in ***curly braces {…}***
  - Compound statements can be used as components of control statements allowing multiple statements to be used where, syntactically, a single statement is specified or expected
- A ***control construct*** (or a control statement) includes some control expression(s) and the corresponding statements (often compound statements)

Control Expressions

- **A control expression has a <u>Boolean</u> value**
  - **An expression with a non-Boolean value used in a control statement will have its value <u>converted to Boolean</u> automatically**
- **Comparison operators**

- **== != < <= > >= (<u>equality comparison</u>)**

- Boolean operators
- && || !

- Warning! A boolean *<u>object</u>* normally evaluates to true
  - **Unless the object is *null* or *undefined* or *NaN***

Selection Statements

- **The *if-then* and *if-then-else* are similar to that in other programming languages, especially C/C++/Java**

JS-Demo#4 : area.html

- **area.html: computes area of circle**

```
1  <html>
2  <body>
3  <p> area.html: computes area of circle</p>
4  <script>
5          var rad= prompt("Enter radius:");
6          if (rad == "" || rad == null)
7              {
8                  alert("Please enter a valid number");
9              }
10         else
11             {
12             rad=parseFloat(rad);
13             document.write("radius = " + rad + ", area = " + 3.14 * rad * rad);
14             alert("radius = " + rad + ", area = " + 3.14 * rad * rad);
15             }
16 </script>
17 </body>
18 <html>
```

The switch Statement Syntax

```
switch (expression) {
case value_1:
     // statement(s)
case value_2:
     // statement(s)
...
[default:
     // statement(s)]
}
```

The *switch* Statement Semantics

- The expression is evaluated
- The value of the expressions is compared to the value in each case in turn
- If no case matches, execution begins at the *default* case
- Otherwise, execution continues with the statement following the case
- Execution continues until either the end of the switch is encountered or a `break` statement is executed

Loop Statements

- **Loop statements in JavaScript are similar to those in C/C++/Java**
- **While**

  `while` (*control expression*)

  statement or compound statement
- **For**

  `for` (*initial expression; control expression; increment expression*)

  statement or compound statement
- **do/while**

  `do` *statement or compound statement*

  `while` (*control expression*)

The `while` Statement Semantics

- The control expression is evaluated
- If the control expression is true, then the statement is executed
- These two steps are <u>repeated until the control expression becomes false</u>
- At that point the while statement is finished

While loop demo

```
1  <html>
2  <body>
3  <h1> While loop - Demo </h1>
4  <script>
5      var i=0;
6      while (i<=5)
7          {
8              document.write ('Number '+i+' = '+i+'</br>');
9              i++;
10         }
11 </script>
12 </body>
13 </html>
```

While loop - Demo

Number 0 = 0
Number 1 = 1
Number 2 = 2
Number 3 = 3
Number 4 = 4
Number 5 = 5

The `for` Statement Semantics

- **The *initial* expression is evaluated**
- **Then the *control expression* is evaluated**
- **If the control expression is true, the statement is executed**
- **Then the increment expression is evaluated**
- **The previous three steps are repeated as long as the control expression remains true**
- **When the control expression becomes false, quit the entire statement**
- **E.g.,**

```
for (var count = 0; count < 110000; count++)
    {   . . . }
```

For loop - Demo

```
1  <html>
2  <body>
3  <h1> For loop - Demo </h1>
4  <script>
5      for (var i=0; i<=5; i++)
6          {
7              document.write ('Number '+i+' = '+i+'</br>');
8          }
9  </script>
10 </body>
11 </html>
```

For loop - Demo

Number 0 = 0
Number 1 = 1
Number 2 = 2
Number 3 = 3
Number 4 = 4
Number 5 = 5

The `do/while` Statement Semantics

- The statement is executed <u>first</u> (so at least once)
- The control expression is then evaluated
- If the control expression is true, the previous steps are repeated
- This procedure continues until the control expression becomes <u>false</u>

Object Creation and Modification

- The *new* expression is used to create an object
  - This includes a call to a *constructor*
  - The *new* operator creates a blank object, the *constructor* creates and initializes all properties of the object
- Properties of an object are accessed using a dot notation: *object.property*
- Properties are not variables, so they are not declared
- The number of properties of an object may vary dynamically in JavaScript

Dynamic Properties

- Create (using new) my_car and add some properties

```
// Create an Object object
var my_car = new Object();
// initialize the make property
my_car.make = "Ford";
// initialize model
my_car.model = "Contour SVT";
```

- The **delete** operator can also be used to delete a property from an object
  - **delete** my_car.model

The "for-in" Loop

- **Specifically for JavaScript, very useful!**
- **Syntax**

  for (*identifier* **in** *object*)

  statement or compound statement
- The loop lets the identifier take on each property in turn in the object
- Printing the properties in my_car:

```
for (var prop in my_car)
  document.write("Name: ", prop, "; Value: ",
    my_car[prop], "<br />");
```

- Result would be like the following:

```
Name: make; Value: Ford
Name: model; Value: Contour SVT
```

Arrays

- An Array is a list of elements indexed by a numerical value – JS treats it as an Object
- Array indexes in JavaScript begin at 0
- Dynamic size: array size can be modified even after created!
- The concept here is much more flexible than in C++/Java

Array Demo

```
1  <html>
2  <body>
3  <h1> Array - Demo </h1>
4  <script>
5      var names=["Mary","Max","Mark"];
6      for (var i=0;i<names.length;i++)
7          {
8          document.write('Name('+ i +') = '+ names[i] + '</br>');
9          }
10 </script>
11 </body>
12 </html>
```

Array - Demo

Name(0) = Mary
Name(1) = Max
Name(2) = Mark

`Array` Object Creation

- **Arrays can be created using the <u>new Array</u> operation**
  - **new Array with one parameter creates an empty array of the specified <u>number of elements</u>**
    - **new Array(10)**
  - **new Array with <u>two or more</u> parameters creates an array with the specified parameters <u>as elements</u>**
    - **new Array(10, 20)**
- **Literal arrays can be specified using square brackets to include a list of elements**
  - **var alist = [1, “ii”, “gamma”, “4”];**
- **The elements in an array can be of <u>different types</u>**

Characteristics of `Array` Objects

- The length of an array is one more than the highest index to which a value has been assigned or the initial size (*Array* created with one argument), whichever is larger
- Assignment to an index greater than or equal to the current length simply increases the length of the array
- Only assigned elements in an array occupy space, e.g.,
  - Suppose an array was created using new Array(200), and only elements at indices 150 through 174 were assigned values
  - In this case, only the 25 assigned elements are allocated storage, the other 175 would not be allocated storage

`Array` Methods

- **join(seperator)** *// into a string*
- **reverse()**
- **sort(sortfunc)**
- **concat (array1, …, arrayN)**
- **slice(start, end)**
- **…**

Check the details and usage of these methods at

http://www.w3schools.com/jsref/jsref_obj_array.asp

Array Methods for Dynamic List Operations

- **push**
  - **Add to the end**
- **pop**
  - **Remove from the end**
- **unshift**
  - **Add to the front**
- **shift**
  - **Remove from the front**

Two-dimensional Arrays

- **A two-dimensional array is an <u>array of arrays</u>**
  - **This needs not even be rectangular shaped: different rows could have different lengths!**

Function -- Fundamentals

- **Function definition syntax**
  - **A function definition consist of a header and a compound statement**
  - **A function header:**
    - **function *function-name*(*optional-formal-parameters*)**
- **The return statements**
  - **A return statement causes a function to cease execution and to pass control to the caller**
  - **A return statement may include a value which is sent back to the caller**
    - **This value may be used in an expression by the caller**
  - **A return statement without a value implicitly returns *undefined***
- **Function *call* syntax**
  - **Function name followed by parentheses and any actual parameters**
  - **Function call may be used as an expression or part of an expression**
- **Functions must be defined before use in the page header**

Functions are Objects!

- **Functions are objects in JavaScript!**
- **Functions may, therefore, be assigned to variables and to object properties**
  - **Object properties that have function values are methods of the object**
- **Example**

```
function fun() {
  document.write("This surely is fun! <br/>");
}
  ref_fun = fun; // Now, ref_fun refers to the fun object
  fun(); // A call to fun
  ref_fun(); // Also a call to fun
```

areafcn.html: computes area of circle using function

```
1  <html>
2  <body>
3  <p> areafcn.html: computes area of circle using function</p>
4  <script>
5      var rad = prompt('Enter radius');
6          if (rad == "" || rad == null)
7          {
8           alert('Please enter a valid number:');
9          }
10         else
11         {
12          rad=parseFloat(rad);
13          alert ('radius = ' + rad + ', area = '+ area(rad));
14         }
15
16     function area(r) {
17         return 3.141592654 * r * r
18     }
19 </script>
20 </body>
21 </html>
```

Local Variables

- The *scope* of a variable is the range of statements over which the variable is visible
- A variable not declared using *var* has global scope, *i.e.*, visible throughout the page, even if used only inside a function definition
- A variable declared with var outside a function definition also has global scope
- A variable declared with var inside a function definition has local scope, visible only inside the function
  - **If a global variable has the *same name*, it is**
  - *hidden/inaccessible* inside the function definition

Parameters

- **Parameters named in a function header are called *formal parameters***
- **Parameters used in a function call are called *actual parameters***
- **Parameters are passed by value**
  - The value is copied from actual to formal parameters
  - For an **object** parameter, the passed value is in fact a reference, so the function body can actually change the object, and the change persists after the function returns
  - However, *direct assignment* to the formal parameter itself inside the function will not change the actual parameter

Parameter Passing Example

```
function fun1(my_list) {
      var list2 = new Array(1, 3, 5);
      my_list[3] = 14;
      my_list = list2;
      ...
}
...
var list = new Array(2, 4, 6, 8)
fun1(list);
```

- **The first assignment changes the *list* array passed from the caller.**
- **The second assignment has no effect to the actual parameter.**
- ***So what is in list now?***
- ***Pass by reference* can be simulated by passing an <u>array</u> containing the value.**

Parameter Checking

- JavaScript checks neither the type nor the number of parameters in a function call
  - Formal parameters have no type specified
  - Extra actual parameters are ignored
  - If there are fewer actual parameters than formal parameters, the extra formal parameters remain undefined
- This is typical for scripting languages
- There is a "secret second channel" between caller and the called functions – the *argument* array
- This array holds all of the actual parameters, whether there are more/less of them than there are formal parameters

The `sort` Method of Array, Revisited

- Can use a function parameter to affect how to sort array elements
  - The parameter typically is a function taking two parameters
  - The function returns a negative value to indicate the first parameter should come before the second
  - The function returns a positive value to indicate the first parameter should come after the second
  - The function returns 0 to indicate the first parameter and the second parameter are equivalent as far as the ordering is concerned

Constructors

- Constructors are functions that create and initialize properties for new objects
- A constructor typically uses the keyword *this* in the body to reference the object being created
- Method properties are properties that refer to functions
  - Other methods (functions) may also use the *this* to refer to the object
- Example

```
function car(new_make, new_model, new_year) {
    this.make = new_make;
    this.model = new_model;
    this.year = new_year;
} // then we can apply new to this constructor
X = new car("Honda", "Pilot", "2003");
```