# Lecture Notes 5
# JavaScript Document Object Model and Event Handling

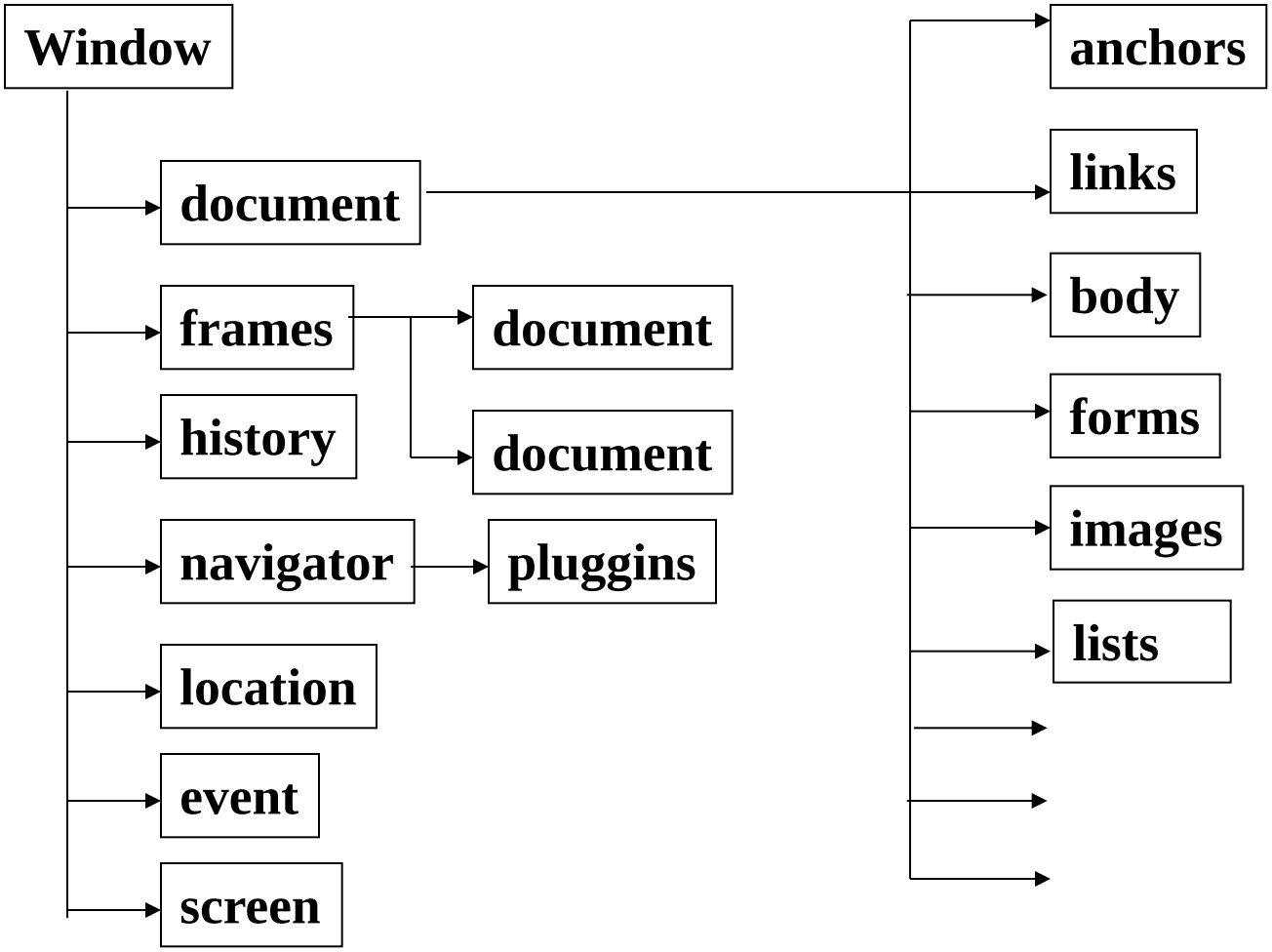
## Window and Document Objects

**Anand Tripathi**

**CSci 4131**

## Internet Programming

# Dynamic HTML: Object Model



# Window Control Methods (See Section 13.7)

- Many of these functions are subject to security privileges assigned to the downloaded code.
  - See Chapter 21 (Flanagan)
- open and close a window
- moveTo methods moves the upper-left corner of the window to the specified coordinates.
- moveBy moves a window x or y directions by given number of pixels.
- resizeTo and resizeBy

# Properties of the Window Object

The window object represents the browser's display window.

The following are some of the properties defined for a window object:

- open open a new window (pop-up)
- closed boolean which is true if the window has been closed
- document reference to the document object displayed in the window Through this reference the properties of a document object can be accessed.
- frames[ ] array of frames contained within the window
- history browsing history
- location points to Location object representing the URL of the document displayed in the window. Its "href" property can be set to load a new document.
- location-bar, scrollbars, statusbar, toolbar, personbar, menubar
- Various bar objects.
- name window's string name; can be used with TARGET attribute in HTML
- opener window object that opened this one.
  Status line text at the bottom of the browser.

- parent  if the current window is a frame, this contains a reference to the frame of the window that contains this one.
- self  self reference
- top  If the current window is a frame, this contains a reference to the window object of the top-level window.
- alert(), confirm(), and prompt() methods
- alert() prints a message in a box.
- confirm() asks for "yes" or "no" kind of response and the JavaScript execution is suspended until that response is received.
- prompt() asks for some user input. Execution is suspended until a response is received.
- focus() and blur()  to acquire or release keyboard focus for the window.
- moveBy() or moveTo()  Used for moving a window.
- setInterval() and clearInterval()  Schedule or cancel a function for periodic execution.
- setTimeout() and clearTimeout()  Set or cancel execution of a function to be executed once after a specified timeout in milliseconds.
- close  Close a window
- pageXOffset,  pageYOffset

# Opening a new window

- A new top level browser window can be created using window.open() method.

- It takes 3 parameters:
-

1. First is the URL of the document to be displayed there.
2. Second is the name of the window.
3. Third is the list of options for various status bars.

- Example:

```
var winID = window.open( "main.html", "mainWindow",
    "width=800,height=600,directories=no,location=yes,
    status=no,toolbar=yes,menubar=yes,scrollbars=yes,
    resizeable=yes" );

winID.alert("Hello");
```

The following parameters are set as yes/no:

- location, menubar, toolbar, scrollbars, resizeable, status, directories
- The following are set as number of pixels: height, width
- Click here to see this example.

# Window Closing

- Window.open( ) function returns an identifier for the new window.

  winID = window.open( … );

- A window opened by  a JavaScript can be closed by the same code:

  winID.close()

- A code can close the window in which it is running by

  window.close()

- If a code tries to close a window other than ones it created, the user is presented with a dialog box.

# Window Control Methods
## (See Section 13.7 of Flanagan Book)

- Keyboard focus:
  - focus( ) will acquire keyboard focus for a window
  - blur( ) relinquishes keyboard focus
- Scrolling
  - scrollBy( ) scrolls the document displayed in the window by given number of pixels left/right, up/down
  - scrollTo( ) scrolls to an absolute position

# Example: Creating a new window

**Click here to see the example working**

This example is form an older version of Deitel's book.

```
<script type = "text/javascript">
  var childWindow; // variable to control the child window
   function createChildWindow()
   {  // these variables all contain either "yes" or "no"
      // to enable or disable a feature in the child window
      var toolBar;
      var menuBar;
      var scrollBars;

         // determine whether the Tool Bar checkbox is checked
      if ( document.getElementById( "toolBarCheckBox" ).checked )
         toolBar = "yes";
      else   toolBar = "no";
```

Continued…

```
 // determine whether the Menu Bar checkbox is checked
      if ( document.getElementById( "menuBarCheckBox" ).checked )
         menuBar = "yes";
      else  menuBar = "no";
      // determine whether the Scroll Bar checkbox is checked
      if ( document.getElementById( "scrollBarsCheckBox" ).checked )
         scrollBars = "yes";
      else  scrollBars = "no";

      //create a new window with the selected features
      childWindow = window.open( "", "",
         ",toolbar = " + toolBar +
         ",menubar = " + menuBar +
         ",scrollbars = " + scrollBars );

      // enable  buttons
      document.getElementById( "closeButton" ).disabled = false;
      document.getElementById( "modifyButton" ).disabled = false;
      document.getElementById( "setURLButton" ).disabled = false;
   } // end function createChildWindow
```

```
function modifyChildWindow()
  {
     if ( childWindow.closed )
        alert( "You attempted to interact with a closed window" );
     else
        childWindow.document.write(
           document.getElementById( "textForChild" ).value );
  } // end function modifyChildWindow

  // close the child window

function closeChildWindow()
  {  if ( childWindow.closed )
        alert( "You attempted to interact with a closed window" );
     else   childWindow.close();

     document.getElementById( "closeButton" ).disabled = true;
     document.getElementById( "modifyButton" ).disabled = true;
     document.getElementById( "setURLButton" ).disabled = true;
  } // end function closeChildWindow
```

```
// set the URL of the child window to the URL in the parent window's
     myChildURL

function setChildWindowURL()
   {
      if ( childWindow.closed )
         alert( "You attempted to interact with a closed window" );
      else
         childWindow.location =
            document.getElementById( "myChildURL" ).value;
   } // end function setChildWindowURL

</script>
```

```
</head>
<body>
  <h1>Hello, this is the main window</h1>
  <p>Please check the features to enable for the child window<br/>

  <input id = "toolBarCheckBox" type = "checkbox" value = ""
    checked = "checked" />
    <label>Tool Bar</label>

  <input id = "menuBarCheckBox" type = "checkbox" value = ""
    checked = "checked" />
    <label>Menu Bar</label>

  <input id = "scrollBarsCheckBox" type = "checkbox" value = ""
    checked = "checked" />
    <label>Scroll Bars</label></p>
```

```
<p>Please enter the text that you would like to display  in the child
    window<br/>

  <input id = "textForChild" type = "text"
    value = "<h1>Hello, I am a child window.</h1> " />

  <input id = "createButton" type = "button"
    value = "Create Child Window" onclick = "createChildWindow()" />

  <input id= "modifyButton" type = "button" value = "Modify Child Window"
    onclick = "modifyChildWindow()" disabled = "disabled" />

  <input id = "closeButton" type = "button" value = "Close Child Window"
    onclick = "closeChildWindow()" disabled = "disabled" /></p>

  <p>The other window's URL is: <br/>
  <input id = "myChildURL" type = "text" value = "./" />
  <input id = "setURLButton" type = "button" value = "Set Child URL"
    onclick = "setChildWindowURL()" disabled = "disabled" /></p>
</body>
```

# Window's Location Object (13.8 Flanagan)

- Location object is a representation of the URL of the document being displayed in the window.
- href property contains the URL.
  - You can read or write it. Writing it will load a new document.
- You can obtain properties such as host, protocol, pathname etc for it.
- search property contains any portion of the Url following a question mark. This is usually the query-string for CGI.
- Using getArgs you can parse it.

# Window's Location Object (13.8 Flanagan)

- One can selectively change parts of the URL and reload a document or a new document.

- This object provides methods:
  - reload( )
  - assign( ) - load a new URL
  - replace( )
    - This will load the specified URL, and also replace the current entry in "history". Not possible to go back to the previous page.

# Properties of Window's Location Object)

Location object has the following properties

See http://www.w3schools.com/jsref/obj_location.asp

| | |
|---|---|
| hash | Sets or returns the anchor part (#) of a URL |
| host | Sets or returns the hostname and port number of a URL |
| hostname | Sets or returns the hostname of a URL |
| href | Sets or returns the entire URL |
| origin | Returns the protocol, hostname and port number of a URL |
| pathname | Sets or returns the path name of a URL |
| port | Sets or returns the port number of a URL |
| protocol | Sets or returns the protocol of a URL |
| search | Sets or returns the querystring part of a URL |

# Document Object

This provides functions for manipulating and accessing document and other objects contained in it.

Document object contains other objects such as forms, images, links, anchors

We will study:

- Some example programs
- Events and Event Handling Model

# Document's "location" property

- Document object also has a property named "location"
- It is a read-only string, usually the same as window's location.href property.
- However, these two are not always equal.
- On redirection,
  - window.location is a refererence to a Location object
  - whereas document.location is a string containing the URL of the retrieved resource.

# Document Object

A document object has the following properties:

forms[] Array of form objects. These the forms objects numbered in the order of their presentation in the document.

anchors[] Array of anchor objects (it is a named position within a document – e.g. anchor to go to top or some section)

<a name="section1"> Section 1: Introduction </a>

links[] Array of all hypertex link objects appearing in the document.

iages[] Array of image objects.

It is always more convenient to refer to objects by names instead of array indices changing the order of these objects in a document may require changes in the code.

# Document Object Naming

```
<form name="f1">
  <input type="button"  value="PUSH Me">
</form>
```

Assuming that this is the first form in the HTML document, in JavaScript you can write:

```
document.forms[0]          // refer by indexing
```

OR

```
document.f1                // refer by name
```

# Document Properties

| | |
|---|---|
| alinkColor | Color of a link which is activated, while user is clicking on it |
| vlinkColor | Visited link color |
| linkColor | Color of unvisited links |
| lastModified | Time the document was last modified |
| bgColor | Background color |
| fgColor | Default text color |
| cookie | Using this JavaScript can read/write cookies |

# Document Properties

| | |
|---|---|
| referer | The URL of the document that contained this document's link |
| URL | URL of the document same a window.location.href except in cases of server redirection |
| | window.location.href contains requested URL |
| | document.URL actual URL loaded after redirection |
| location | Currently loaded document’s URL (same as URL) |
| title | Text under the title tag |
| domain | This property is used for security purposes |
| | similar to "same origin" policies |

# Dynamically Generated Documents

- JavaScript provides 4 methods to dynamically create HTML documents.
- Entirely new document can also be created.
- Using the following methods, a JavaScript program can output an HTML document

  write( )

  writeln( )

  open( )

  close( )

- Write methods should be used within the <script> tags. This part of the code is interpreted when the document is loaded and parsed.
- If you use document.write in an event handler, you would overwrite the current document completely, instead of appending text to it. This would also completely overwrite the event handler and other JavaScript code.
- You can write to a document in another window or frame.

# Links in a Document

- links property of the document object contains Link objects that represent the hypertext links in a document.
  - These are the <a> </a> elements.
- Similarly the following properties refer to arraus
  - images
  - Forms
  - anchors

# Example: Random Link

```
<a  href="http://www.cs.umn.edu" > Department page </a>
<br/>
<a  href="http://www.cnn.com" >  CNN </a>
<br/>
<a  href="http://www.abcnews.com" > ABC News </a>
<br/>
<a  href="http://www.yahoo.com" > Yahoo </a>
<br/>
<a  href=""
    onClick="this.href=
    document.links[Math.floor(Math.random()*document.links.length)].href"
>  Random Link </a>
```

Click here to execute this code.

# Example

```
function listlinks(d) {
    var newwin = window.open("", "linklist",
                    "menubar,scrollbars,resizable,width=600,height=300");

    for (var i = 0; i < d.links.length; i++) {
        newwin.document.write('<a href="' + d.links[i].href + '">')
        newwin.document.write(d.links[i].href);
        newwin.document.writeln("</a><br/>");
    }
    newwin.document.close();
}
```

See this example

# Form Objects

A form object has the following objects in its elements[] array.

- Buttons
- Checkboxes
- Hidden
- Password
- Reset
- Select
- Submit
- Text
- Textarea

# Examples of Events

| Event | Occurs | For Objects |
|---|---|---|
| onclick | Mouse click by the user. Handler can return *false* to cancel the default action. | Links, buttons, elements |
| onload | Window document's body or image loading is completed. | Window, image |
| onmouseover | Mouse moves an element | Link, image |
| onmouseout | Mouse moves off an element. | Link, image |
| onreset | Form reset button is clicked | Form |
| onsubmit | Form submit button is clicked | Form |
| onchange | Change in value | select, checkbox |

# Link Event Handler

onclick event handler for a link is invoked when the user clicks on a hypertext link;

If the handler returns false, the browser does not follow the link

onmouseover

onmouseout

# Forms Object

- document.forms
  - Represents an array containing all forms object in the document
- document.forms[0] is the first form
- document.forms[document.forms.length-1]
  - is the last form
- Event Handlers are associated with the forms object as well as with the elements of the form

# Event Handler for Form

- These are defined in the “form” tag.
- onsubmit
  - It is executed when the form is submitted.
  - If the executed code returns false, the submission will NOT be made
- onreset
  - This is executed when the user tries to reset a form.

# Dynamic Updating of DOM Structure of an HTML document

- JavaScript can modify the structure of an HTML document currently displayed in the browser windoe.
  - Delete an element
  - Change attributes of an element
  - Create and add a new element
    - Set attributes of the new element
- See Chapter 12 of Deitel's book:
  - Document Object Model: Objects and Collections

# createElement

- JavaScript code can create a new element of the specified tag name
- For example:

```
var newDiv = createElement ( “div” )
varnewImage = createElement( imo” );
var anotherParagraph = createElement ( “p” )
```

# Other methods and node properties for modifying DOM tree structure

See Mozilla site for Node documentation

- parentNode - get pointer to the parent node of the specified node
- removeChild
- replaceChild( newchild, oldChild)
- appendNode
- insertBefore

# Images and Animation

**Click here to execute this example**

```
<head>
    <title>Deitel Book Cover Viewer</title>
    <style type = "text/css">
      .thumbs   { width: 192px;
                  height: 370px;
                  padding: 5px;
                  float: left }
      .mainimg  { width: 289px;
                   padding: 5px;
                   float: left }
      .imgCover { height: 373px }
      img       { border: 1px solid black }
    </style>
```

```
<script type = "text/javascript">
        <!--
        var interval = null; // keeps track of the interval
        var speed = 6; // determines the speed of the animation
        var count = 0; // size of the image during the animation

        // called repeatedly to animate the book cover
        function run( )  {
           count += speed;

        // stop the animation when the image is large enough
         if ( count >= 375 )  {
              window.clearInterval( interval );
              interval = null;
           } // end if

        var bigImage = document.getElementById( "imgCover" );
        bigImage.style.width = .7656 * count + "px";
        bigImage.style.height = count + "px";
        } // end function run
```

```
// inserts the proper image into the main image area and begins the animation
    function display( imgfile )
      {
        if ( interval )
          return;

        var bigImage = document.getElementById( "imgCover" );
        var newNode = document.createElement( "img" );
        newNode.id = "imgCover";
        newNode.src = "fullsize/" + imgfile;
        newNode.alt = "Large image";
                    newNode.className = "imgCover";
        newNode.style.width = "0px";
        newNode.style.height = "0px";
        bigImage.parentNode.replaceChild( newNode, bigImage );
        count = 0; // start the image at size 0
        interval = window.setInterval( "run()", 10 ); // animate
      } // end function display
    </script>
```

```
<body>
    <div id = "mainimg" class = "mainimg">
      <img id = "imgCover" src = "fullsize/iw3htp4.jpg"
              alt = "Full cover image" class = "imgCover"/>
    </div>
    <div id = "thumbs" class = "thumbs">
      <img src = "thumbs/iw3htp4.jpg" alt = "iw3htp4"
        onclick = "display( 'iw3htp4.jpg' )" />
      <img src = "thumbs/chtp5.jpg" alt = "chtp5"
        onclick = "display( 'chtp5.jpg' )" />
      <img src = "thumbs/cpphtp6.jpg" alt = "cpphtp6"
        onclick = "display( 'cpphtp6.jpg' )" />
      <img src = "thumbs/jhtp7.jpg" alt = "jhtp7"
        onclick = "display( 'jhtp7.jpg' )" />
      <img src = "thumbs/vbhtp3.jpg" alt = "vbhtp3"
        onclick = "display( 'vbhtp3.jpg' )" />
      <img src = "thumbs/vcsharphtp2.jpg" alt = "vcsharphtp2"
        onclick = "display( 'vcsharphtp2.jpg' )" />
    </div>
 </body>
```

# OBJECT ORIENTATION BASED ON PROTOTYPES

- "This object is just like that object."
  - The other object is the prototype object.

- Object constructor "new Object()" creates a new empty object (i.e. an object with no properties).

- In general, to create a new object one has to call

  new someObjectConstructorFunction()

# OBJECT ORIENTATION BASED ON PROTOTYPES

- Consider objects of type rectangle:
- First look at the constructor for creating new rectangle objects.

```
function Rectangle (w, h) {
    this.width = w;   /* this refers to the new Rectangle object */
    this.height = h; /*We added two properties to this object*/
}
```

To create a new rectangle named "box":

```
var box = new Rectangle(2, 5);
```

# Defining Methods of an Object

- A function can be added/deleted as an object's property list.
- Example:

```
var box  =  new Rectangle(2, 5);
function compute_area () {
    return ( this.width * this.height );
}
Box.area = compute_area ;
```

- A function is also an object.
- Suppose that we have an object named "box" of type Rectangle.
- We can now associate the function "compute_area" with the object named "box".
- We will add a property named "area" to the function object.
- This method can now be invoked later as shown below:

```
var A = box.area() ;
```

# Adding Function Properties in the Constructor

- **Function properties to an object can be added in its constructor**
- **This is shown below using the Rectangle constructor.**

```
function compute_area () {
    return ( this.width * this.height );  }
function compute_perimeter () {
    return ( return 2* (this.width + this.height) ); }

function Rectangle (w, h) {
    this.width = w;    this.height = h;
    this.area = compute_area;
    this.perimeter = compute_perimeter;
    this.resize = set_size;
}
```

- The above constructor will store with each newly created Rectangle object three new properties, which are methods of the object.
- We should note that this requires extra memory storage for the object.
- All Rectangle objects have these three properties identical. This means that this extra storage is wasteful.
- At a later point, if we wish to redefine any of these three methods, we will have to update that property in each of the existing Rectangle objects.

- Prototype objects allow us to solve these problems by storing all common properties with the prototype object associated with a constructor function.

# PROTOTYPE OBJECTS

For every constructor function, there is one prototype object.

For example: Rectangle.prototype

We can add properties to a prototype object as shown below:

```
Rectangle.prototype.compute_area =
    function () { return ( this.width * this.height ) }

Rectangle.prototype.compute_perimeter =
    function () { ( return 2* (this.width + this.height) ); } ;
```

# RULES TO SEARCH FOR SOME PROPERTY OF AN OBJECT

1. First look for a property in the object's property list.

2. If not found there, then search the prototype object's property list.

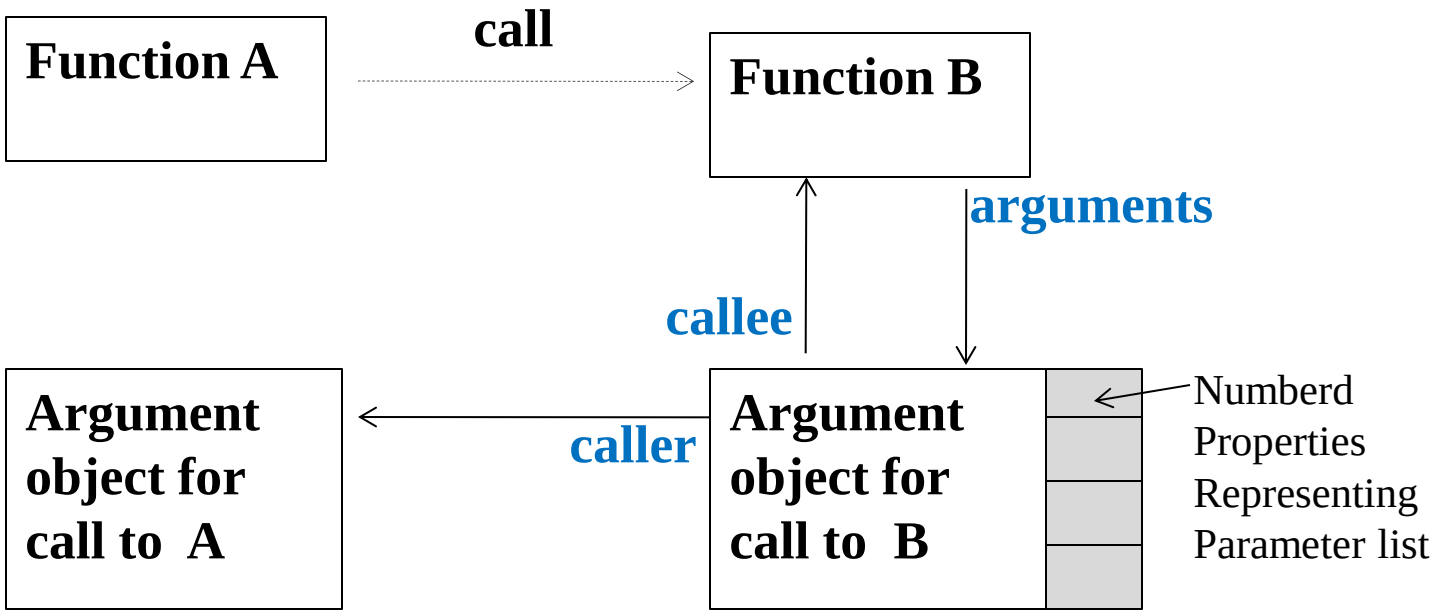
3. The "this" reference in any prototype method refers to the original object.

# ARGUMENT OBJECTS

- Every function call results in the creation of an "argument" object which has several properties related to that function call.
- Argument objects have some predefined named properties.
- Argument objects also have numbered properties (as an array), which represent the arguments passed in the call.
- NAMED PROPERTIES OF AN ARGUMENT OBJECT:
  - callee  This is reference to the function object that is being executed by this call.
  - caller This is a reference to the argument object of the function that called this function.
  - "arity" is the property of a function object. It represents the expected number of arguments.

# An argument object is created on call to a function

- Suppose that function A calls function B



# EXAMPLE OF ARGUMENT OBJECT USAGE

```
function f ( x, y, z ) {
    if ( arguments.length ! = 3 ) {
        alert ( "Number of arguments passed does not match the
    expected number");
    }
}
```

The test in the above code could be written as:

```
    ( arguments.length != arguments.callee.arity )
```

Another example: recursive call of an anonymous function:

```
function (x) {
    if ( x > 1 )
        return ( x * arguments.callee(x-1) );
return x;
}
```

# Adding New Properties to Function Objects

Consider the following example:

```
// counter property will act like a static variable

sequenceNumberGenerator.counter = 0;
function sequenceNumberGenerator( ) {
    return sequenceNumberGenerator.counter++ ;
}
alert ( sequenceNumberGenerator() );
alert ( sequenceNumberGenerator() );
alert ( sequenceNumberGenerator() );
```

See example here.

# Variable Scope

Click here to see this example

```
<html>
<head>
     <title>
        Javascript Variable Scope
     </title>
</head>
<body>
   <form name="myForm">
       <input type="button"  value="Click to declare counter and initialize to 0"
                    onClick=" count = 0;  alert( count ); " > <br/>
      <input type="button"  value="Click to increment counter"
                    onClick="count++; alert(  count )" >
   <br/>
   </form>
</body>
</html>
```