CGT 141/CPT 141 Lecture 17 Wk 11
Client-side Scripting: JavaScript

Background Information

- Object-oriented, client-side scripting language
- Originally developed by Netscape to extend the browser.
- Microsoft has its own "flavor" called Jscript. Difficult to distinguish between JavaScript/Jscript
- The official standard for scripting on the web is ECMAScript.
- Embedded in HTML (using <script> in <head>) and is interpreted at run time by a scripting engine in the browser that parses the code.

What does object-oriented programming (OOP) mean?

- An object is a piece of code (composed of code [instructions] and data [on which the code is executed]) that receives and sends message
- The primary difference between OOP and traditional programming is:
  - In traditional programming code and data are kept separate from one another
  - In OOP, code and data are merged into a single thing: an object
- In OOP, objects are defined by classes in that a class defines the things that an object can do (called methods) and the characteristics that object can have (properties)
- In most OOP languages there are three types of objects:
  - Intrinsic objects – objects that are "predefined" and you can readily use. Additionally, these are visually represented in the environment.
    - Examples: window, document, form are all objects in JavaScript
  - Extrinsic objects – objects that are "predefined" and you can readily use, but for which there exists no visual representation.
    - Examples: variables, arrays, etc.
  - User-defined objects – objects that the user creates by defining the class, that is, defining information about that object, such as the things it can do (methods) and what characteristics it has (properties).

Objects

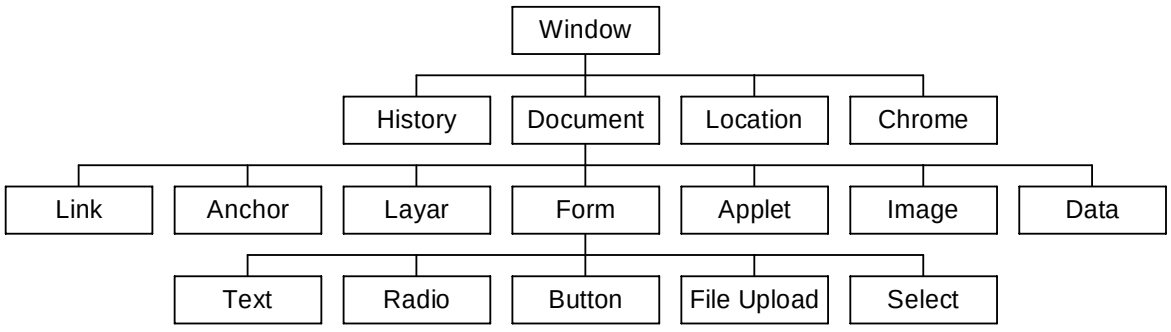
- Everything in OOP is about objects.
- To figure out how to program something with OOP you:
  - Determine what you want to do
    - This leads you to figure out what "object" you need to do it. For example, let's say you want to change the status bar of the browser window. Thus, the window object is the item you need to interact with.
    - Next, see if the window object has a method or property that will allow you to change the status bar (you can look this up in a book as you get started). You'll find no predefined behavior (method) for changing the status bar. However, you will find that the window object has a property call status that can be changed…thus to change the status bar of the browser window, you change the status property of the window object!

- Determine when you want to do it
  - Everything in an OOP environment occurs as a result of an event.
    - When the user interacts with something…
    - Or when something else happens in the environment…
  - If you figure out "when you want something to occur" it will tell you want object you need to attach the "what you want to do" to. In this case, let's say you want to have the status bar change (we already know what to do to make this happen) when the user rolls over a link.
    - Now we have to figure out if a hyperlink is "smart", so to speak. We want to see if a hyperlink knows when the user rolls over it. Indeed it has several events it can respond to (there are events associated with most of the intrinsic JavaScript objects) and the one we are interest in is onmouseover.
  - Thus we could write the following to change the status bar when a user rolls over a link:

```
<a href="mypage.html" onmouseover="window.status='Link 1';">Link 1</a>
```

  - Although we won't be getting into the nitty-gritty of JavaScript (you could spend an entire semester on it, as there is much you can do) remember:
    - Figure out what you want to do
      - Tells you the object, method or properties you need
    - Figure out when you want to do it
      - Tells you the event handler you need and the object to attach code to.

The Object Hierarchy



Example JavaScript

Given this code:

```
…
<form name=”form0” action=”processForm.html” method=”post”>
        <input type=”text” name=”login” value=”” />
        <input type=”password” name=”pword” value=”” />

        <input type=”submit” name=”submit” value=”Login” />
</form>
…
```

The following JavaScript will place the Insertion Point (IP) in the input box named “login” You should note that anything within the <script> … </script> tags is case-sensitive.

```
<script language=”javascript”>
        document.form0.login.focus();
</script>
```

Notice that the JavaScript code started with the document object from the hierarchy above. Using the dot operator (.), we accessed the form by using its name, form0. Again, using the dot operator, we accessed the input box by using its name, login. Once we have accessed the login box, we used the dot operator one more time to call the method focus() which sets the IP to whatever object that called it. Also note that each statement in JavaScript ends with a semicolon.

The most common place to put the <script> … </script> block is nested inside of the <head> … </head> tags. However, <script> blocks can go almost anywhere within an html document. The general rules for tag placement in html still apply, i.e.: everything else goes between <body> and </body>, and never place anything between certain tags, such as <tr> and <td> for example.

```
<head>
        …
        <script language=”javascript”>
                document.form0.login.focus();
        </script>
</head>
```

That being said, the second most common place to locate a <script> … </script> block is inside the <body> tags, at the end of the html. In this instance, the entire page will load, then the JavaScript at the bottom of the html will be evaluated.

```
<body>
        …
        …
        <script language=”javascript”>
                document.form0.login.focus();
        </script>
</body>
```

However, a <script> … </script> block can be located almost anywhere inside of the open and close body tags.