ATLS 4630/5630: Web Front-End Development
Week 3: Document Object Model Interaction

Output

What are some different ways to handle output in JavaScript?

- alert() creates a dialog box with output
- document.write can write to the web page while it's still open
- access and manipulate the DOM

Input

What are some different ways to handle input in JavaScript?

- prompt() provides a dialog box to for users to enter input
- HTML form
- DOM interaction

In object-oriented programming objects are things that can be described by:

- Properties – attributes or characteristics
  - Data values
- Methods – behavior or tasks an object can perform
  - Operations like functions
- Events – situations that happen
  - situations the object can send notifications about

The Document Object Model (DOM) is the standard object model that browsers use to store HTML documents. (slide)

- The window object serves as the global object, you access it by just typing "window". It's within this object that all of your JavaScript code is executed. Like all objects it has properties and methods.
- When referring to the location, history, and document objects we can leave off window because it is assumed.
- The DOM itself isn't JavaScript—it's a standard from the World Wide Web Consortium (W3C) that most browser manufacturers have adopted and added to their browsers.
- The DOM provides an application programming interface for HTML documents
  - Using JavaScript you can access and update the content, structure, and style of HTML elements
- The DOM stores web pages as an organized hierarchy of elements, attributes, and text
  - In the DOM these are called *nodes*
  - Every node is an object
- JavaScript can manipulate the contents of a page by accessing these objects
- Each node has a relationship to the other nodes just like a family tree.
  - parentNode
  - previousSibling
  - nextSibling
  - firstChild
  - lastChild
- Enables you to traverse the DOM and select elements in relation to each other.

Properties

Properties define the object's state at any given time. They give the object the ability to hold information. You can access the value of any property by *objectName.propertyName*

- document.title
  - Title of the document
- document.URL
  - Some properties are read only and can't be changed

You can access all CSS properties the only difference is you use camel case instead of a dash - document.body.style.backgroundColor is the background color of the document body

https://repl.it/@aileenjp/DOM-interaction

Example:

```
var color;
color=prompt("What color is your light saber?");
document.body.style.backgroundColor = color;
```

Methods

Methods are actions an object can perform. You can call a method on an object by *objectName.methodName (parameter list);*

- document.write("Hi there");
- Parameters allow us to send data to the object that it needs to execute the method.

Accessing DOM elements

- Methods that find elements in the DOM tree are called DOM queries.
- Methods that return a single element node
  - JavaScript can access a single element node given the value of its id attribute using *getElementById("id")*
    - document.getElementById('header');
    - returns the one element with the id 'header'
  - If your script needs to access an element more than once use a variable to store its location
    - var el = document.getElementById('header');
    - Stores the location of id 'header' in the variable 'el' so you can easily refer to it later in your program
  - JavaScript can access a single element node given a css selector using *querySelector("css selector")*
    - Returns only the first matching element it finds

Example:

```
document.getElementById('prequel').style.color="purple";
```

Changing the CSS display property is an easy way to hide something on a page. Note that it also moves the rest of the page up.

```
document.getElementById('prequel').style.display="none";
```

The className property adds a new class to the original trilogy div.

```
document.getElementById("original").className="highlight";
```

What's another way we could have made the text color is this div yellow?

- document.getElementById("original").style.color="#FFD700";

Accessing Multiple DOM elements

- Methods that return multiple elements store them in nodelists
  - all the returned elements in an array indexed from 0 to the nodelist's length-1
  - JavaScript can retrieve tag elements by selecting a tag name using *getElementsByTagName(tag)*
    - returns a nodelist of all elements on the page with that tag
    - could also use *querySelectorAll()*
  - JavaScript can retrieve class elements by selecting a class name using *getElementsByClassName(class)*
    - returns a nodelist of all elements on the page with that class

Example:
getElementsByTagName("li") returns all li tags in an array
You must access a node to change it, can't change the entire array in JavaScript

```
var movies=document.getElementsByTagName("li");
for(i=0; i<movies.length; i++)
{
  movies[i].style.color="#FFD700";
}
```

Creating DOM elements
There are two different ways to create, move, or remove HTML elements from the DOM
The innerHTML property can be used to retrieve and replace HTML content in an element. Assigning new HTML to the innerHTML property will replace all the HTML in that element.

1. Create a variable with your new HTML markup
2. Select the element whose content you want to update
3. Update content of the selected content with the new markup
   a. Set it to an empty string if you want to remove all the content and make it blank.

Example:
The textContent property lets you access or update the text in an element

```
var films=document.getElementById("stories");
films.textContent = "Anthology Films";
```

The innerHTML property replaces all the HTML in the element

```
var newfilm="Anthology Films<ul><li>Rogue One</li></ul>"
films.innerHTML = newfilm;
```

As you create new DOM elements you can then continue to modify those. So if we add `id='rogue'` to our li tag above, now we can reference that id.

```
document.getElementById("rogue").style.color="yellow";
```

DOM manipulation

- You can create a new element using *createElement()*
  - var newp=document.createElement('p');
- You can create a new text node using *createTextNode()*
  - var text=document.createTextNode("Labs");
- Once a new element is created you need to add it to the document.
- You can add it as a child to an existing node using *appendChild( )*
  - newp.appendChild(text);
    - Adds the text node as a child of the newp object, and now you have a mini-tree of two nodes.
  - main.appendChild(newp);
    - adds newp to the existing document by making it a child of id main.
- You can insert a new element in an exact location using *insertBefore( )*
  - insertBefore() is useful if you don't want the new node to be a child
- You can make a copy of a node using *cloneNode( )*
- *removeChild( )* removes a node
  - Must be called from the parent node
- *replaceChild( )* replaces one node with another node
  - Must be called from the parent node
  - More convenient than cloning and replacing

Example:
Create an element node and a text node.
Append the text node to the li node.
Append that mini-DOM tree to the last child of the stories section which is the ul

```
var newitem=document.createElement("li");
var newfilm=document.createTextNode("Solo");
newitem.appendChild(newfilm);
films.lastChild.appendChild(newitem);
```

You could also use `firstElementChild` since stories only has one element child.

Events

A browser event is when something happens on a web page, an event occurs on an object in the DOM

- User interaction events occur when a user interacts with the browser's UI
  - load
- Keyboard events occur when a user interacts with the keyboard
  - keydown
- Mouse events occur when a user interacts with a mouse, trackpad, or touch screen

    - click
- Focus events occur when an element games or loses focus
    - focus
- Form events occur when a user interacts with a form element
    - submit

When an event has occurred, we say the event was *fired*
When an event has fired you can use JavaScript to handle the event

1. Select the element you want to respond to
2. Specify which event on the element you want to respond to
3. Call the code you want to run when the event occurs

Listeners

Event listeners are the best approach to handling events.

- HTML event handlers where the event is mixed in with the HTML is an old way to attach events but it's not good practice to mix HTML and JavaScript
- Traditional DOM event handlers are better but can only attach a single function to an event.

Event listeners using addEventListener() is the preferred approach and the only one you should be using.

- Can handle multiple functions and events on one element
- Older browsers and IE 5-8 did not support addEventListener() You can read about the workarounds in the book
- element**.addEventListener**('event', functionName, event flow);
- Event names don't need to precede with 'on'
- No parentheses after the function name
    - When the interpreter sees the parens after a function call it runs the function right away.
    - In an event handler you want to wait until the event triggers it.

Event listeners need to access objects in the DOM so the whole web page has to be loaded first.

- The javascript is at the bottom of the page to ensure that the page has loaded first so the id 'logo' can be found in the DOM

Example:

Let's change the top image when a user clicks on it.

```
function changeImage(){
  document.getElementById('logo').src='images/starwarsgalaxy.jpg';
}
var sw=document.getElementById("logo");
sw.addEventListener("click", changeImage, false);
```

Now let's do the same for our yoda image.

```
function changeImage2(){
  document.getElementById('yoda').src='images/starwars_episode4.png';
}
var yoda=document.getElementById("yoda");
```

```
yoda.addEventListener("click", changeImage2, false);
```

These two functions do the same general task.
Whenever you find yourself writing multiple functions that are similar there might be a better way to structure them.
Can you think of a way to implement these by generalizing these functions?

When designing a program you should think about the high-level functionality and abstracting as much of the functionality as possible.

- Use functions to create modular code
- Put functions in an external file so they can easily be linked to from multiple pages
- Create high level functions that are reusable
    - Use parameters for data the functions need instead of hard coding data in the function

Example:

```
function changeImage(imageId, imageName){
  document.getElementById(imageId).src=imageName;
}

var sw=document.getElementById("logo");
sw.addEventListener("click", function(){
  changeImage("logo", "images/starwarsgalaxy.jpg");
  }, false);

var yoda=document.getElementById("yoda");
yoda.addEventListener("click", function(){
  changeImage("yoda", "images/starwars_episode4.png");
  }, false);
```

If the function takes parameters you can't just put them in the function call in an event listener.
You have to wrap the function call in an anonymous function so the interpreter won't run the function right away it will just set up the event listener.

Lab: Create a web page that uses JavaScript to interact with the DOM in the following ways:

- manipulate at least one CSS property through the DOM
- manipulate at least one DOM property that is NOT a CSS property
- responds to at least two different types of events using event listeners(using addEventListener())

Your JavaScript should be in a function in an external js file.