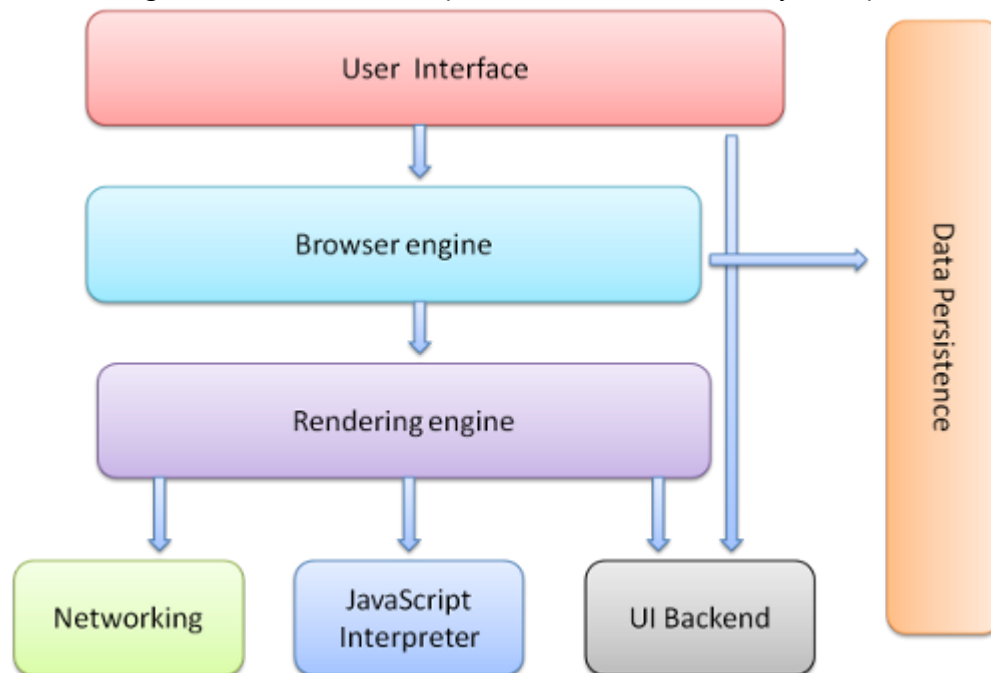


Week 4 Learning Objectives

Browser Basics

1. Explain the difference between the BOM (browser object model) and the DOM (document object model). - The DOM is the collection of nodes that represent the hierarchy of HTML elements that are rendered on the page. The BOM is the hierarchy of the browser objects, such as the window and the navigator of the browser. The DOM is thus a smaller part of the BOM.
2. Given a diagram of all the different parts of the Browser identify each part.



3. Use the Window API to change the innerHeight of a user's window.

```
newWindow = window.open("https://www.google.com", "Google Homepage", "width=100, height=100");
newWindow.resizeTo(400, 200)
```

4. Identify the context of an anonymous functions running in the Browser (the window). - We can double check by console logging `this` in the browser

```
let func = () => {
  console.log(this)
}
func();

// OR we can use an IIFE (Immediately Invoked Function Expression)

(() => {
```

```
console.log(this);  
})();
```

5. Given a JS file and an HTML file, use a script tag to import the JS file and execute the code therein when all the elements on the page load (using DOMContentLoaded)

```
<head>  
  <script type='text/javascript' src='exampleScript.js'></script>  
</head>
```

```
window.addEventListener('DOMContentLoaded', () => { ...code } )
```

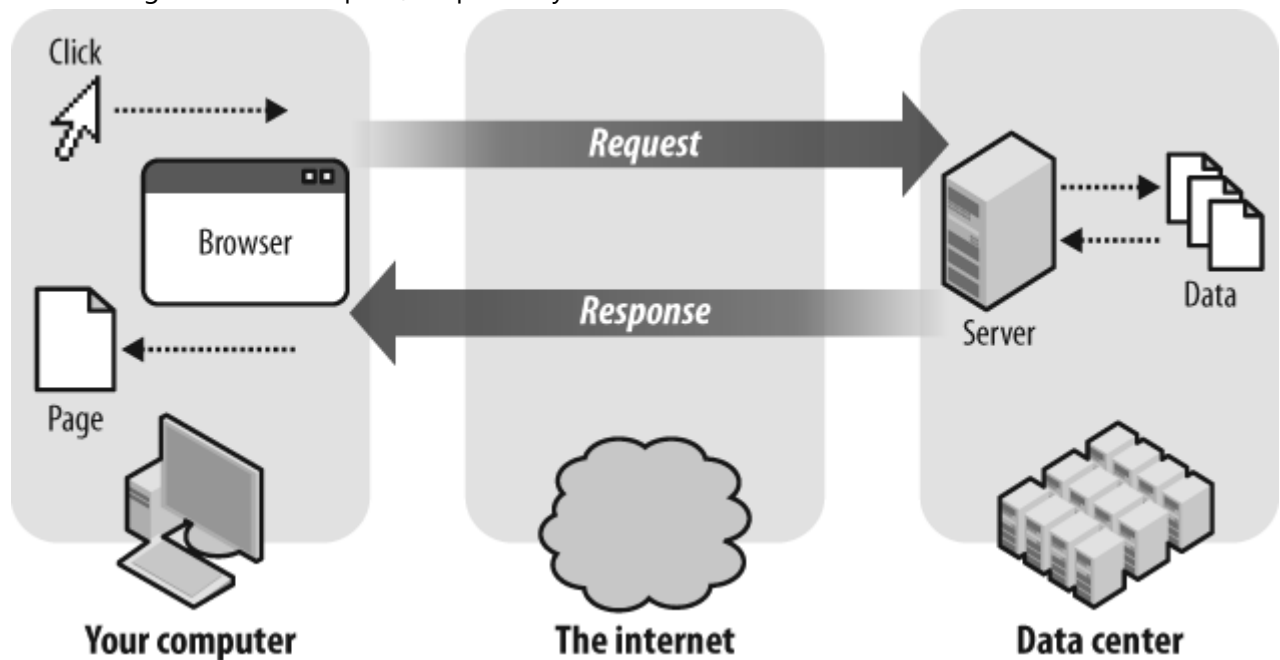
6. Given a JS file and an HTML file, use a script tag to import the JS file and execute the code therein when the page loads

```
<head>  
  <script type='text/javascript' src='exampleScript.js'></script>  
</head>
```

```
window.onload = () => { ...code }
```

7. Identify three ways to prevent JS code from executing until an entire HTML page is loaded - An event listener in the JavaScript file (`window.addEventListener('DOMContentLoaded', callback)` or `window.onload = callback`) - Have the script tag at the end of the HTML file - Use the keywords `async` and/or `defer` in the script tag

8. Label a diagram on the Request/Response cycle.



9. Explain the Browser's main role in the request/response cycle. - Parsing HTML,CSS, JS - Rendering that information to the user by constructing a DOM tree and rendering it)
10. Given several detractors - identify which real-world situations could be implemented with the Web Storage API - Shopping cart, form data, user preferences like a night mode
11. Given a website to visit that depends on cookies (like Amazon), students should be able to go to that site add something to their cart and then delete that cookie using the Chrome Developer tools in order to empty their cart. - Open Developer tools - Go to the Application tab - Open the Cookies section and either delete individual cookies or clear all

Element Selection

1. Given HTML that includes `<div id="catch-me-if-you-can">HI!</div>`, write a JavaScript statement that stores a reference to the HTMLDivElement with the id "catch-me-if-you-can" in a variable named "divOfInterest".

```
const divOfInterest = document.getElementById('catch-me-if-you-can');
```

2. Given HTML that includes seven SPAN elements each with the class "cloudy", write a JavaScript statement that stores a reference to a NodeList filled with references to the seven HTMLSpanElements in a variable named "cloudySpans".

```
const cloudySpans = document.querySelectorAll('span.cloudy');
// querySelector and querySelectorAll take in CSS selectors
// tag names are placed directly in the string: 'span'
// classes are preceded by a .: '.cloudy'
// ids use a #: '#catch-me-if-you-can'
// 'span.cloudy' is looking for only span tags that have the class 'cloudy'
```

```
// We could have used just '.cloudy', but if we had other tags with the same
// class they also would have been included in the collection.
```

3. Given an HTML file with HTML, HEAD, TITLE, and BODY elements, create and reference a JS file that in which the JavaScript will create and attach to the BODY element an H1 element with the id "sleeping-giant" with the content "Jell-O, Burled!".

```
window.addEventListener('DOMContentLoaded', () => {
  const message = document.createElement('h1');
  message.setAttribute('id', 'sleeping-giant');
  const text = document.createTextNode('Jell-O, Burled!')
  message.appendChild(text);
  // The following could be used instead of creating and appending a text node
  // message.innerHTML = 'Jello-O, Burled!';
  document.body.appendChild(message);
})
```

4. Given an HTML file with HTML, HEAD, TITLE, SCRIPT, and BODY elements with the SCRIPT's SRC attribute referencing an empty JS file, write a script in the JS file to create a DIV element with the id "lickable-frog" and add it as the last child to the BODY element.

```
window.addEventListener('DOMContentLoaded', () => {
  const lastElement = document.createElement('div');
  lastElement.setAttribute('id', 'lickable-frog');
  document.body.appendChild(lastElement);
})
```

5. Given an HTML file with HTML, HEAD, TITLE, SCRIPT, and BODY elements with no SRC attribute on the SCRIPT element, write a script in the SCRIPT block to create a UL element with no id, create an LI element with the id "dreamy-eyes", add the LI as a child to the UL element, and add the UL element as the first child of the BODY element.

```
<head>
  <script type='text/javascript'>
    const addListElement = () => {
      const listElement = document.createElement("ul");
      const listItem = document.createElement("li");
      listItem.setAttribute("id", "dreamy-eyes");
      listElement.appendChild(listItem);
      document.body.prepend(listElement);
    };
    window.onload = addListElement;
  </script>
</head>
```

6. Write JavaScript to add the CSS class "i-got-loaded" to the BODY element when the window fires the DOMContentLoaded event.

```
window.addEventListener('DOMContentLoaded', () => {
  document.body.className = 'i-got-loaded';
  // document.body.classList.add('i-got-loaded');
})
```

7. Given an HTML file with a UL element with the id "your-best-friend" that has six non-empty LIs as its children, write JavaScript to write the content of each LI to the console.

```
window.addEventListener("DOMContentLoaded", event => {
  const parent = document.getElementById("your-best-friend");
  const childNodes = parent.childNodes;
  for (let value of childNodes.values()) {
    console.log(value);
  }
  // Using a standard forEach, we could also look at the innerHTML of each li
  // childNodes.forEach(child => {
  //   console.log(child.innerHTML)
  // })
});
```

8. Given an HTML file with a UL element with the id "your-worst-enemy" that has no children, write JavaScript to construct a string that contains six LI tags each containing a random number and set the inner HTML property of ul#your-worst-enemy to that string.

```
// generate a random number for each list item
const getRandomInt = max => {
  return Math.floor(Math.random() * Math.floor(max));
};

// listen for DOM ready event
window.addEventListener("DOMContentLoaded", event => {
  // push 6 LI elements into an array and join
  const liArr = [];
  for (let i = 0; i < 6; i++) {
    liArr.push("<li>" + getRandomInt(10) + "</li>");
  }
  const liString = liArr.join(" ");

  // insert string into the DOM using innerHTML
  const listElement = document.getElementById("your-worst-enemy");
  listElement.innerHTML = liString;
});
```

- If we wanted to append each child as it was created instead of constructing the entire HTML we could do so:

```
const getRandomInt = max => {
  return Math.floor(Math.random() * Math.floor(max));
};

window.addEventListener("DOMContentLoaded", event => {
  const listElement = document.getElementById("your-worst-enemy");
  for (let i = 0; i < 6; i++) {
    const liItem = document.createElement('li');
    liItem.innerHTML = getRandomInt(10);
    listElement.appendChild(liItem);
  }
});
```

9. Write JavaScript to update the title of the document to the current time at a reasonable interval such that it looks like a real clock.

```
window.addEventListener("DOMContentLoaded", event => {
  const title = document.getElementById("title");
  const time = () => {
    const date = new Date();
    const seconds = date.getSeconds();
    const minutes = date.getMinutes();
    const hours = date.getHours();

    title.innerHTML = hours + ":" + minutes + ":" + seconds;
  };
  setInterval(time, 1000);
});
```

Event Handling

1. Given an HTML page that includes `<button id="increment-count">I have been clicked 0 times</button>`, write JavaScript that increases the value of the content of `span#clicked-count` by 1 every time `button#increment-count` is clicked.

```
window.addEventListener("DOMContentLoaded", event => {
  // getElementById returns a single element (the first that it finds that has
  the specified id)
  const button = document.getElementById("increment-count");
  const count = document.getElementById("clicked-count");
  button.addEventListener("click", event => {
    count.innerHTML = `${event.detail}`;
  });
});
```

- event.detail will reset if we pause clicking on the button. If we want a persistent count, we can keep track of a variable:

```
window.addEventListener("DOMContentLoaded", event => {
  const button = document.getElementById("increment-count");
  const count = document.getElementById("clicked-count");
  let clicks = 0;
  button.addEventListener("click", event => {
    clicks++;
    count.innerHTML = clicks;
  });
});
```

2. Given an HTML page that includes `<input type="checkbox" id="on-off"><div id="now-you-see-me">Now you see me</div>`, write JavaScript that sets the display of `div#now-you-see-me` to "none" when `input#on-off` is checked and to "block" when `input#on-off` is not checked.

```
window.addEventListener("DOMContentLoaded", event => {
  // store the elements we need in variables
  const checkbox = document.getElementById("on-off");
  const divShowHide = document.getElementById("now-you-see-me");
  // add an event listener for the checkbox click
  checkbox.addEventListener("click", event => {
    // use the 'checked' attribute of checkbox inputs
    // returns true if checked, false if unchecked
    if (checkbox.checked) {
      // if the box is checked, show the div
      divShowHide.style.display = "block";
      // else hide the div
    } else {
      divShowHide.style.display = "none";
    }
  });
});
```

- We also could have used a CSS file that had styling for ``show`` and ``hide`` classes, then added and removed the classes instead of changing the style directly.

```
window.addEventListener("DOMContentLoaded", event => {
  // store the elements we need in variables
  const checkbox = document.getElementById("on-off");
  const divShowHide = document.getElementById("now-you-see-me");
  // add an event listener for the checkbox click
```

```
checkbox.addEventListener("click", event => {
  // use the 'checked' attribute of checkbox inputs
  // returns true if checked, false if unchecked
  if (checkbox.checked) {
    // if the box is checked, show the div
    divShowHide.className = "show";
    // else hide the div
  } else {
    divShowHide.className = "hide";
  }
});
});
```

```
.show {
  display: block;
}
.hide {
  display: none;
}
```

3. Given an HTML file that includes `<input id="stopper" type="text" placeholder="Quick! Type STOP">`, write JavaScript that will change the background color of the page to cyan five seconds after a page loads unless the field input#stopper contains only the text "STOP".

```
window.addEventListener("DOMContentLoaded", event => {
  const stopCyanMadness = () => {
    // get the value of the input field
    const inputValue = document.getElementById("stopper").value;
    // if value is anything other than 'STOP', change background color
    if (inputValue !== "STOP") {
      document.body.style.backgroundColor = "cyan";
    } else {
      // If we successfully typed in "STOP" I added an indicator to show that it
      worked
      console.log("Thank you for stopping the madness");
    }
  };
  setTimeout(stopCyanMadness, 5000);
});
```

4. Given an HTML page with that includes `<input type="text" id="fancypants">`, write JavaScript that changes the background color of the textbox to #E8F5E9 when the caret is in the textbox and turns it back to its normal color when focus is elsewhere.

```
window.addEventListener("DOMContentLoaded", event => {
  const input = document.getElementById("fancypants");
```



```
input.addEventListener("focus", event => {
  event.target.style.backgroundColor = "#E8F5E9";
});
input.addEventListener("blur", event => {
  event.target.style.backgroundColor = "initial";
});
});
```

- One important thing to note here is that the event listeners for the `focus` and `blur` events are on the input tag directly. In our project this week we had these event listeners on a parent element because we wanted to apply the same action to many child input tags. In order for us to do this we had to pass in the third argument `true` to `addEventListener` because the `focus` and `blur` events do not bubble. The `true` argument tells `addEventListener` to trigger the callback on the capture phase instead of on the standard bubble phase. (We could also use the `focusin` and `focusout` events, which do bubble.)

5. Given an HTML page that includes a form with two password fields, write JavaScript that subscribes to the forms submission event and cancels it if the values in the two password fields differ.

```
window.addEventListener("DOMContentLoaded", event => {
  // get the form element
  const form = document.getElementById("signup-form");

  const checkPasswordMatch = event => {
    // get the values of the pw field and pw confirm field
    const passwordValue = document.getElementById("password").value;
    const passwordConfirmValue = document.getElementById("confirm-password")
      .value;
    // if the values are not equal, alert the user
    // otherwise, submit the form
    if (passwordValue !== passwordConfirmValue) {
      // prevent the default submission behavior
      event.preventDefault();
      alert("Passwords must match!");
    } else {
      alert("The form was submitted!");
    }
  };
  // listen for submit event and run password check
  form.addEventListener("submit", checkPasswordMatch);
});
```

6. Given an HTML page that includes a div styled as a square with a red background, write JavaScript that allows a user to drag the square around the screen. - Important aspects of this LO are knowing that we

have to add the 'draggable' property to the element, as well as knowing the basics of what events are associated with a drag-n-drop. -

- **dragstart**: listened for on the element that is being dragged, fired when the user clicks and starts to move - **dragenter**: fired when a draggable element crosses the boundary into this element - **dragleave**: fired when a draggable element crosses the boundary out of this element - **dragover**: fired every few hundred milliseconds while a draggable object is over this element - **drop**: fired when a draggable object is released over this element - Reference the example on aaronline for how these are applied: <https://open.appacademy.io/learn/js-py---feb-2020-online/week-4-feb-2020-online/drag-n-drop-api> - Reference the MDN docs for more info on all of the different events: https://developer.mozilla.org/en-US/docs/Web/API/HTML_Drag_and_Drop_API - Don't worry about implementation of a drag event in a coding environment, just know the basics of the events and how to make something draggable.

7. Given an HTML page that has 300 DIVs, create one click event subscription that will print the id of the element clicked on to the console.

```

window.addEventListener("DOMContentLoaded", event => {
  // Add a click event listener on the document's body
  document.body.addEventListener("click", event => {
    // console.log the event target's ID
    console.log(event.target.id);
  });
});

```

8. Identify the definition of the bubbling principle. - When an event occurs on an element, any handlers that respond to that event on the target are invoked, then on the parent element, then on its parent, all the way up the DOM hierarchy. We can prevent this bubbling by invoking `event.stopPropagation()`.

JSON

1. Identify and generate valid JSON-formatted strings - In most cases we can just add "" around our data. Whenever we are reference a string, we have to escape a " character with ". - {key1: "value", key2: 12} becomes '{"key1": "value", "key2": 12}' - Important to note that the keys in this object are strings, so we add in the escaped quotes around each key. {key1: "value"} is the same as '{"key1": "value"}
2. Use JSON.parse to deserialize JSON-formatted strings

```

const objJSON = '{"key1\\": \"value\\", \"key2\\": 12}';
const parsedObj = JSON.parse(objJSON);

```

3. Use JSON.stringify to serialize JavaScript objects


```

const obj = {key1: "value", key2: 12};
const objJSON = JSON.stringify(obj);

```

4. Correctly identify the definition of "deserialize" - Taking a string and turning it into data. We say string here because we are working with JSON. More broadly, it could refer to another format, such as bits, that we are turning into data.
5. Correctly identify the definition of "serialize" - Taking data and turning it into a string format in order to send or store more easily. (Same applies with the usage of strings, could be a different format if we're not talking about JSON)

Storage

1. Write JavaScript to store the value "I  falafel" with the key "eatz" in the browser's local storage.

```
localStorage.setItem('eatz', 'I  falafel')
```

2. Write JavaScript to read the value stored in local storage for the key "paper-trail".

```
localStorage.getItem('paper-trail')
```