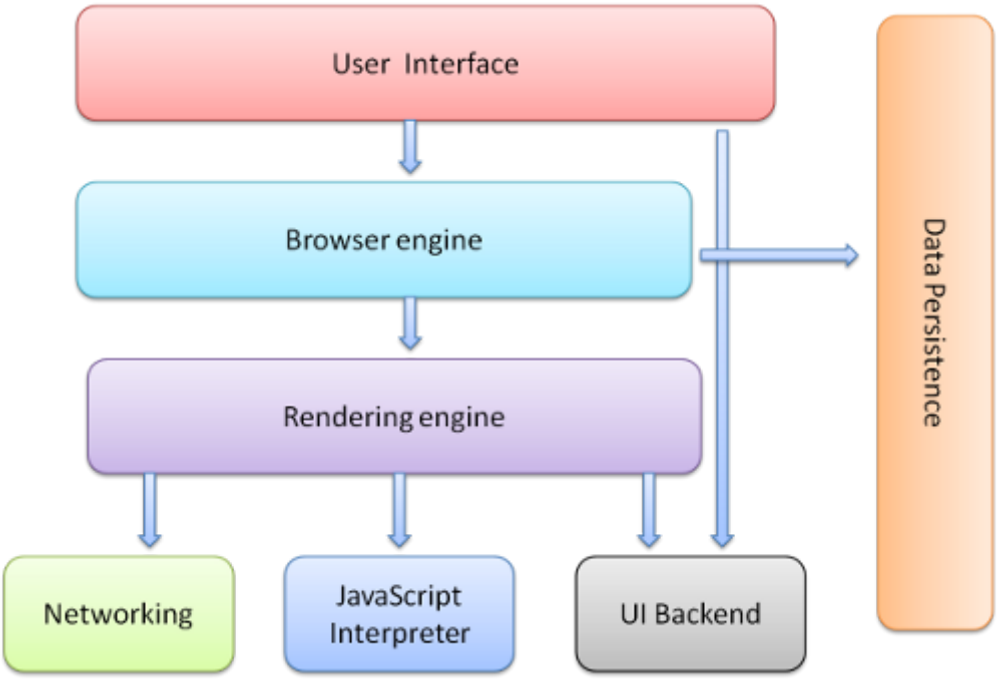
Objectives

Browser Basics

1. Explain the difference between the BOM (browser object model) and the DOM(document object model).

- The `Browser Object Model` is the hierarchy of browser objects - the document object is a part of the hierarchy. The DOM is also managed by the BOM.
- The `Document Object Model` is the collection of HTML elements or Nodes that can be accessed or manipulated.

2. Given a diagram of all the different parts of the Browser identify each part. Use the Window API to change the innerHeight of a user's window.



- `User Interface` : The browser interface that users interact with, includes everything except the requested page content (such as the address bar, bookmarks menu etc)
- `Browser Engine` : Manages the interaction between the UI and the rendering engine.
- `Rendering Engine` : Renders the requested page content. For example, if the requested content is HTML, the HTML and CSS will be parsed and rendered.
- `Networking` : Handles all netword calls, for example HTTP requests.
- `Javascript Interpreter` : Parses and executes our JS code.
- `UI Backend` : Uses Operating System user interface methods.

- **`Data Storage`** : Persistence of data stored in the browser, i.e. cookies.

```
// First Open a new window
// Arguments are: URL, Name, ...Features
newWindow = window.open(
  "http://www.google.com",
  "Google",
  "width=100, height=100"
);
// Use resizeTo() Window API method to change the width & height
newWindow.resizeTo(500, 500);
// conducted in pixels
```

3. Identify the context of an anonymous functions running in the Browser (the window).

- When we call an anonymous function in the global scope, the context will default to the **`global context`** .

4. 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)

```
<!DOCTYPE html>
<html>
  <head>
    <script type="text/javascript" src="someFile.js"></script>
  </head>
</html>
```

```
window.addEventListener("DOMContentLoaded", (event) => {
  // some code
});
```

- We can import a JS file into our HTML by placing a script tag into our head meta data and indicating a type and source URL.
- To ensure our code doesn't run until our page loads, we can add an event listener to our window object, calling for execution only after our DOM is fully loaded.

5. 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

- We can add a async or defer attribute to our script tag if we wish to execute our code when the entire page loads, not just the DOM.
- **Note**: Async and defer essentially do the same thing, Async is not supported in older browsers so you would have to use defer - however async is considered the standard go-to in most modern

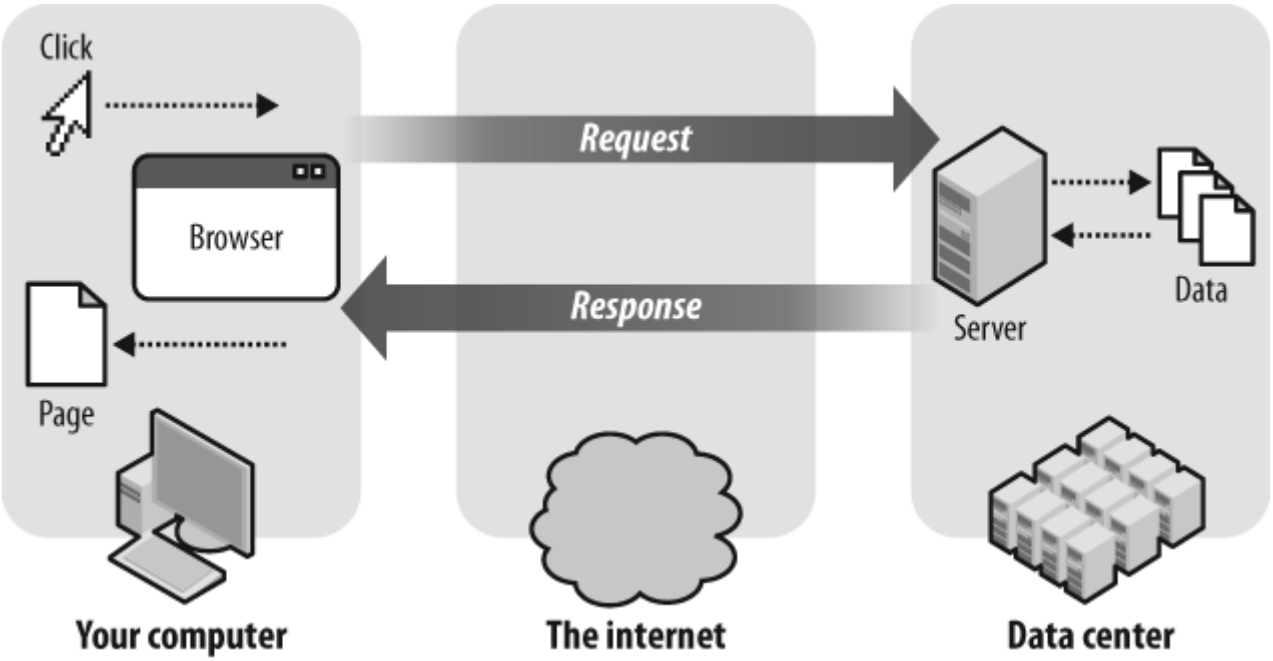
browsers.

```
<!DOCTYPE html>
<html>
  <head>
    <script async defer type="text/javascript" src="someFile.js"></script>
  </head>
</html>
```

6. Identify three ways to prevent JS code from executing until an entire HTML page is loaded

- **In order of preference**:
  1. Adding the async or defer attribute to your script tag.
  2. Use DOMContentLoaded event in our external JS file.
  3. Place the JS import script tag all the way at the bottom of your HTML file.

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



- The left side contains the **client-side** or **browser**
- The right side is the **server side** - a database where we can store data.
- The middle is the **internet**, which comprises a series of **client requests** and **server responses**.

8. 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

9. Given several detractors - identify which real-world situations could be implemented with the Web Storage API (shopping cart, forms saving inputs etc.)

- As a reminder, the Web Storage API allows browsers to store key-value pairs.
- Also note that cookies are different from Web Storage.
- Cookies are meant more for server side, while Web Storage is meant for client-use.
- Some useful mechanisms include:
    1. Storing info about user (shopping cart info, saved form inputs)
    2. Storing session information
    3. Storing persistent data (data that remains even after a refresh)
    4. Storing key-value pairs that aren't needed server side.

10. 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.

- You can delete cookies from your Dev Tools by going to the application tab and manually deleting them with right-click drop down menu.
- Just in case we need to do on the test, some students have mentioned that the Amazon demo only works if you are not logged-in.

Notes

The Browser Object Model

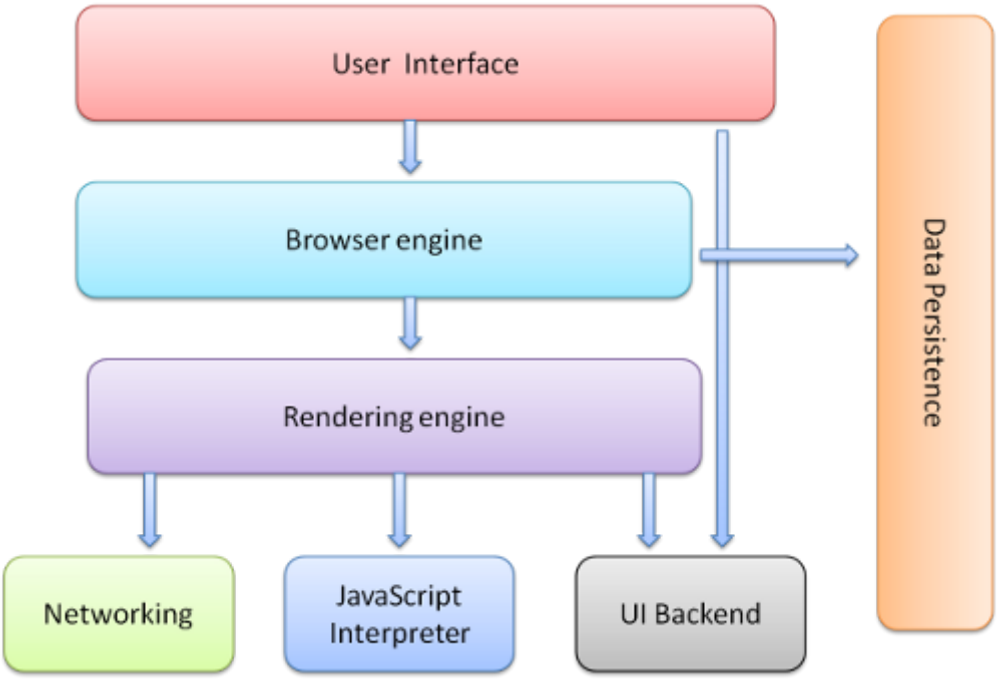
The DOM vs the BOM

- `Document Object Model` : Also known as the `DOM` contains a collection of nodes (HTML Elements), that can be access and manipulated.
    - The `document` object is a Web Page, and the DOM is the hierarchy of browser objects.
- `Browser Object Model` : Also known as the `BOM` , this is the hierarchy of browser objects.
- `Window Object` : Chief browser object which contains properties and methods that we can use to access different objects within the window.
    - `window.navigator` : returns a reference to the navigator object.
    - `window.screen` : returns a reference to the screen object associated with the window.
    - `window.history` : returns a reference to the history object.
    - `window.location` : gets and sets the location/current URL of the window object.

    - `window.document` : (can be shortened to document) returns a reference to the document that the window contains.

- All of the objects above can be shortened to their latter half (i.e. window.document => document)

The Browser Diagram

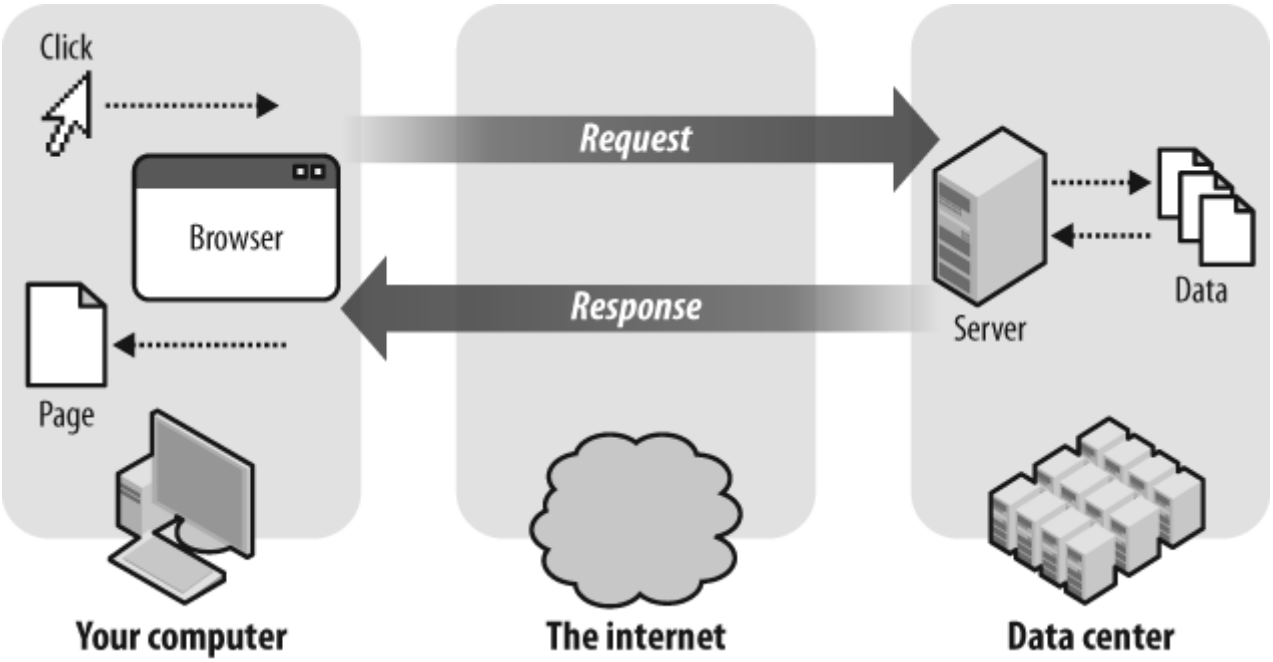


- `User interface` : Browser interface that the user interacts with such as the address bar, buttons, bookmarks, etc. Includes everything except for the requested page content.
- `Browser Engine` : Manages the interactions between the UI and the rendering machine.
- `Rendering Engine` : Displays and renders the requested page content.
- `Networking` : Handles network calls, such as HTTP requests.
- `Javascript Interpreter` : Parses and executes JS code.
- `UI Backend` : Used for drawing basic widgets like combo boxes and windows; uses operating system user interface methods.
- `Data Storage` : The persistence of data stored in the browser, such as cookies.

The Request/Response Cycle

- Browsing the web is just a series of `requests` and `responses` .
- Think about the Request/Response cycle as the communication pattern between a client, browser, and a server.
    - An example is an `HTTP Request` , where we make a request to the server to fetch a URL.

The Request Response Cycle Diagram



- The left side is the `client` side, the middle is the `internet` , and right side is the `server` side.

The Browser's Role in the Request-Response Cycle

- The Browser also does other things besides letting users make requests.
  - Parses HTML, CSS, and JS.
  - Renders information to the user by constructing and rendering a DOM tree.
- The `Network Tab` in your `Developer's Tools` will allow you to view all the requests and responses.

Running JS Scripts in the Browser

Using the Window API

```
// windowTest.js

// Open a new window
newWindow = window.open("", "", "width=100, height=100");

// Resize the new window
newWindow.resizeTo(500, 500);
```

- **resizeTo()** : Windows API method that changes the height and width of a user's window - must be chained with a window.open.

Context, Scope, and Anonymous Function

- Scope is function-based & Context is object-based.
- `Execution Context` refers more to scope than to context.
  - Divided into two phases:
    - `Creation Phase` : The interpreter creates a variable object(activation object) - the scope chain is then initialized.
    - `Excution Phase` : Code is interpreted and executed.
- IIFE's are a type of closure.
- The important things to note about context are:
  - Every function has a context and scope.
  - Context is the object that `owns` the function.
  - Context is most often determined by how a function is invoked.

Running a script on DOMContentLoaded

```
window.addEventListener("DOMContentLoaded", (event) => {
  console.log("This script loaded when the DOM was ready.");
});
```

- DOMContentLoaded runs their script when the document has been loaded, without waiting for stylesheets, images, and subframes to load.

Running a script on page load

The alternative to DOMContentLoaded is, waiting for everything to load in the document before we run our script. For this we can use `window.onload`

```
window.onload = () => {
  console.log(
    "This script loaded when all the resources and the DOM were ready."
  );
};
```

Ways to prevent a script from running until page loads

Some methods include:

1. Using DOMContentLoaded in an externel JS File.
2. Put a script tag importing external code at the bottom of HTML file.
3. Add attribute to our script tag such as async or defer.

```
<script async src="scriptA.js"></script>

<script defer src="scriptB.js"></script>

<script async defer src="scriptC.js"></script>
```

Objectives

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");
// OR
const cloudySpans = document.getElementByClassName("cloudy");
```

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!".

```
<!DOCTYPE html>
<html>
  <head>
    <script type="text/javascript" src="header.js"></script>
  </head>
  <body></body>
</html>
```

```
function addHeader() {
  const header = document.createElement("h1");
  header.setAttribute("id", "sleeping-giant");
  header.appendChild(this.document.createTextNode("Jello-O, Burled!"));
  document.body.appendChild(header);
}
window.addEventListener('DOMContentLoaded', (event) {
    addHeader();
});
```

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.

```
<!DOCTYPE html>
<html>
  <head>
    <script type="text/javascript" src="someFile.js"></script>
  </head>
  <body></body>
</html>
```

```
function addFrog() {
  const newFrog = document.createElement("div");
  newFrog.setAttribute("lickable-frog");
  document.body.appendChild(newFrog);
}
window.addEventListener("DOMContentLoaded", (event) => {
  addFrog();
});
```

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.

```
<!DOCTYPE html>
<html>
  <head>
    <script type="text/javascript">
      window.addEventListener("DOMContentLoaded", (event) => {
        const Ul = document.createElement("ul");
        const Li = document.createElement("li");
        Li.setAttribute("id", "dreamy-eyes");
        Ul.appendChild(Li);
        document.body.appendChild(Ul);
      });
    </script>
  </head>
  <body></body>
</html>
```

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", (event) => {
  document.body.setAttribute("class", "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.

```
function listItems() {
  const ulEle = document.querySelectorAll("your-best-friend");
  const liEle = ulEle.querySelectorAll("li");
  liEle.forEach((ele) => {
    console.log(ele.innerText);
  });
}
window.addEventListener("DOMContentLoaded", (event) => {
  logList();
});
```

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.

```
function randomNum(n) {
  return Math.floor(Math.random() * n);
}

function listGen() {
  const newLi = [];
  for (let i = 0; i < 6; i++) {
    newLi.push(`<li>${randomNum(n)}</li>`);
  }
  const ulEle = document.getElementById("your-worst-enemy");
  ulEle.innerHTML = newLi.join("");
}
window.addEventListener("DOMContentLoaded", (event) => {
  listGen();
});
```

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.

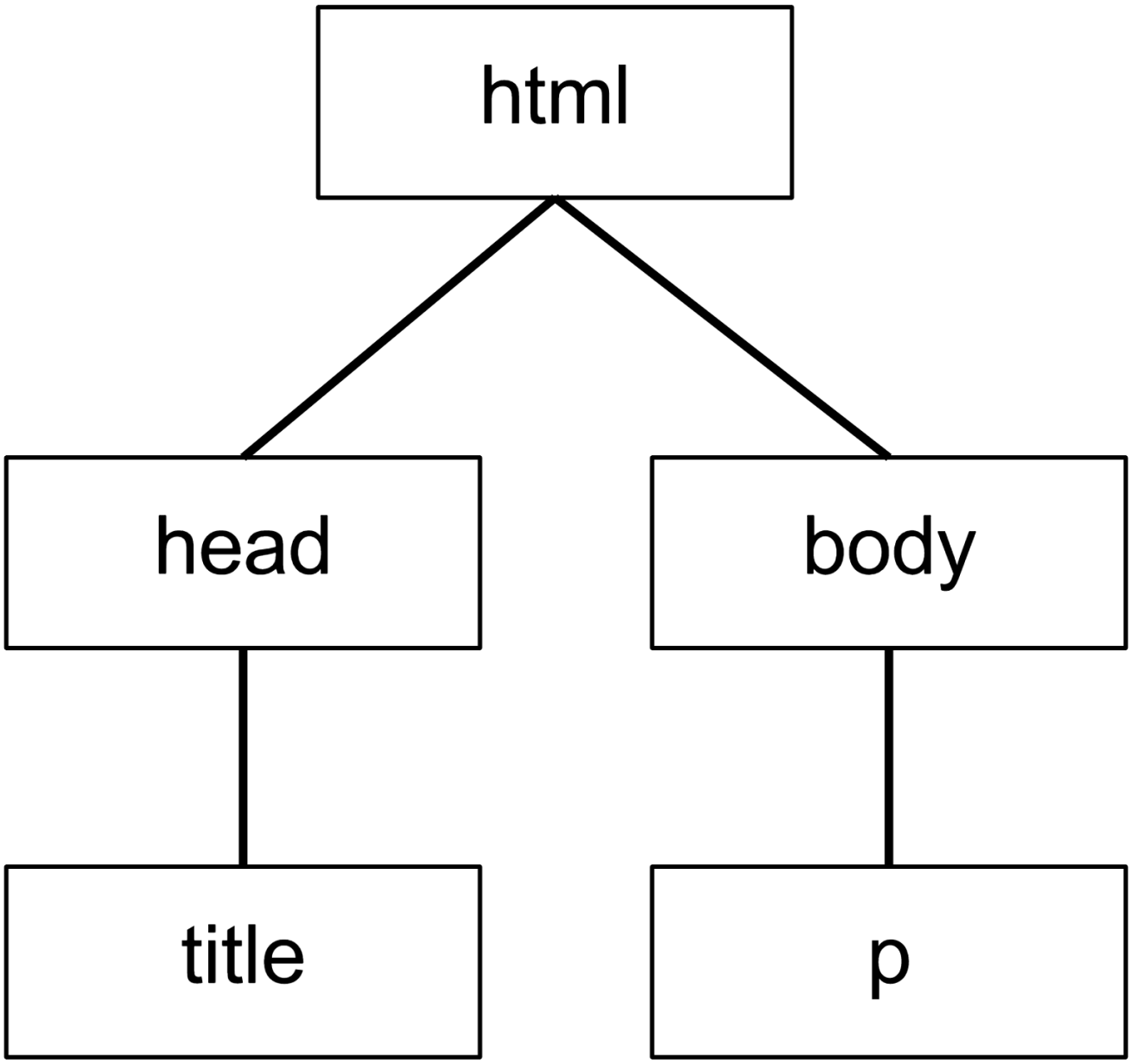
```
function getTime() {
  const now = new Date();
  const hourStr = String(now.getHours());
  const minStr = String(now.getMinutes());
  const secStr = String(now.getSeconds());
  return `${hourStr}:${minStr}:${secStr}`;
}

window.addEventListener("DOMContentLoaded", (event) => {
  const title = document.getElementsByTagName("title")[0];
  title.innerHTML = getTime();

  setInterval(function () {
    title.innerHTML = getTime();
  }, 1000);
});
```

Notes

Element Selection Placement 1

What is the DOM?

- The DOM is an object oriented representation of an HTML document or Web Page, meaning that the document is represented as objects or nodes.



> The DOM Hierarchy

Referncing the DOM

- **`document.getElementById`** : used to reference a single html element via an id.

```
<div id=""”catch-me-if-you-can”"">HI!</div>

const divOfInterest = document.getElementById(“catch-me-if-you-can”)
```

- **`document.querySelectorAll`** : used to reference multiple html elements via a class.

```
<span class="“cloudy”"></span>
<span class="“cloudy”"></span>
<span class="“cloudy”"></span>
<span class="“cloudy”"></span>
<span class="“cloudy”"></span>
<span class="“cloudy”"></span>
<span class="“cloudy”"></span>

const cloudySpans = document.querySelectorAll("span.cloudy");
```

- This generates a static **`Node List`** ; A Node List is not an array but it is possible to iterate over one using for Each.
- Static means changes in the DOM do not affect it's contents.

Creating New DOM Elements

```
const addElement = () => {
  // create a new div element
  const newElement = document.createElement("h1");

  // set the h1's id
  newElement.setAttribute("id", "sleeping-giant");

  // and give it some content
  const newContent = document.createTextNode("Jell-O, Burled!");

  // add the text node to the newly created div
  newElement.appendChild(newContent);

  // add the newly created element and its content into the DOM
  document.body.appendChild(newElement);
};
// run script when page is loaded
window.onload = addElement;
```

- We can also use JS to create elements to the HTML page!

Element Selection Part 2

Inserting Elements in JS File and Script Block

```
const addElements = () => {
  // create a new div element
  const newElement = document.createElement("h1");

  // set the h1's id
  newElement.setAttribute("id", "sleeping-giant");

  // and give it some content
  const newContent = document.createTextNode("Jell-O, Burled!");

  // add the text node to the newly created div
  newElement.appendChild(newContent);

  // add the newly created element and its content into the DOM
  document.body.appendChild(newElement);

  // append a second element to the DOM after the first one
  const lastElement = document.createElement("div");
  lastElement.setAttribute("id", "lickable-frog");
  document.body.appendChild(lastElement);
};
// run script when page is loaded
window.onload = addElements;
```

Referencing a JS File vs. Using a Script Block

- You can also put JS directly into your HTML document by placing it in the script tag

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Cool Website</title>
    <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>
  <body></body>
</html>
```

Element Selection Placement 3

- We previously used `window.onload` to load functions after our pages loaded to ensure all of our objects are within the DOM.
- JS does not need to wait for Stylesheets, images, and subframes to load before running.
- **`DOMContentLoaded`** : event that fires when the initial HTML document has been completely loaded and parsed, without waiting for stylesheets, images, and subframes to finish loading.

```
window.addEventListener("DOMContentLoaded", (event) => {
  document.body.className = "i-got-loaded";
});
```

- Here we are using DOMContentLoaded to add CSS Classes to page elements immediately after they are loaded.

Element Selection Placement 4

Console Logging Element Values

```
<!DOCTYPE html>
<html>
  <head> </head>
  <body>
    <ul id="your-best-friend">
      <li>Has your back</li>
      <li>Gives you support</li>
      <li>Actively listens to you</li>
      <li>Lends a helping hand</li>
      <li>Cheers you up when you're down</li>
      <li>Celebrates important moments with you</li>
    </ul>
  </body>
</html>

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 Element.innerHTML

```
<!DOCTYPE html>
<html>
  <head>
    <script type="text/javascript" src="example.js"></script>
  </head>
  <body>
    <ul id="your-worst-enemy"></ul>
  </body>
</html>

// 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;
});
```

Inserting a Date Object into the DOM

```
<title id="title"></title>;

const date = new Date();

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);
});
```

Objectives

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

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

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.

```
const toggle = () => {
  const checkbox = document.getElementById("on-off");
  const message = document.getElementById("now-you-see-me");
  checkbox.addEventListener("click", (event) => {
    if (checkbox.checked) {
      message.style.display = "block";
    } else {
      message.style.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".

```
const stopPage = () => {
  const input = document.getElementById("stopper");
  if (input.value === "STOP") {
    alert("Phew, you stopped it!");
  } else {
    document.body.style.backgroundColor = "red";
    alert("You died.");
  }
};

window.addEventListener("DOMContentLoaded", (event) => {
  setTimeout(stopPage, 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.

```
const bgColor = () => {
  const input = document.getElementById("fancypants");
  input.addEventListener("focus", (event) => {
    input.style.backgroundColor = "#E8F5E9";
  });
  input.addEventListener("blur", (event) => {
    input.style.backgroundColor = "white";
  });
};
window.addEventListener("DOMContentLoaded", (event) => {
  bgColor();
});
```

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.

```
const passwordChecker = () => {
  const password = document.getElementById("password");
  const confirm = document.getElementById("confirm-password");
  const form = document.getElementById("signup-form");
  form.addEventListener("submit", (event) => {
    if (password.value === confirm.value) {
      alert("Form Submitted!");
    } else {
      event.preventDefault();
      alert("Passwords must match!");
    }
  });
};

window.addEventListener("DOMContentLoaded", (event) => {
  passwordChecker();
});
```

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.

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.

```
const selectDiv = () => {
  document.body.addEventListener("click", (event) => {
    console.log(event.target.id);
  });
};
window.addEventListener("DOMContentLoaded", selectDiv);
```

8. Identify the definition of the bubbling principle.

- When an event happens on an element, it first runs the handlers on it, then on it's parent, then all the way up on other ancestors.

Objectives

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

the content of span#clicked-count by 1 every time button#increment-count is clicked.

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

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.**

```
const toggle = () => {
  const checkbox = document.getElementById("on-off");
  const message = document.getElementById("now-you-see-me");
  checkbox.addEventListener("click", (event) => {
    if (checkbox.checked) {
      message.style.display = "block";
    } else {
      message.style.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".**

```
const stopPage = () => {
  const input = document.getElementById("stopper");
  if (input.value === "STOP") {
    alert("Phew, you stopped it!");
  } else {
    document.body.style.backgroundColor = "red";
    alert("You died.");
  }
};

window.addEventListener("DOMContentLoaded", (event) => {
  setTimeout(stopPage, 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.

```
const bgColor = () => {
  const input = document.getElementById("fancypants");
  input.addEventListener("focus", (event) => {
    input.style.backgroundColor = "#E8F5E9";
  });
  input.addEventListener("blur", (event) => {
    input.style.backgroundColor = "white";
  });
};
window.addEventListener("DOMContentLoaded", (event) => {
  bgColor();
});
```

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.

```
const passwordChecker = () => {
  const password = document.getElementById("password");
  const confirm = document.getElementById("confirm-password");
  const form = document.getElementById("signup-form");
  form.addEventListener("submit", (event) => {
    if (password.value === confirm.value) {
      alert("Form Submitted!");
    } else {
      event.preventDefault();
      alert("Passwords must match!");
    }
  });
};

window.addEventListener("DOMContentLoaded", (event) => {
  passwordChecker();
});
```

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.

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.

```
const selectDiv = () => {
  document.body.addEventListener("click", (event) => {
    console.log(event.target.id);
  });
};
window.addEventListener("DOMContentLoaded", selectDiv);
```

8. Identify the definition of the bubbling principle.

- When an event happens on an element, it first runs the handlers on it, then on it's parent, then all the way up on other ancestors.

Objectives

JSON

1. Identify and generate valid JSON-formatted strings

- JSON formatted strings can contain [] for arrays, and {} for objects.
- Valid JSON values: objects, arrays, strings, numbers, booleans, and null.
- Keys to Objects must be double quoted in JSON.

2. Use JSON.parse to deserialize JSON-formatted strings

```
const JSONString = '[1, 2, 3, 4, "apple", "banana, "pear"]';
const myList = JSON.parse(JSONString);
console.log(myList[4]); // "apple"
```

3. Use JSON.stringify to serialize JavaScript objects

```
const fruits = ["apple", "banana", "pear"];
const JSONString = JSON.stringify(fruits);
console.log(JSONString); // ["apple", "banana", "pear"]
```

4. Correctly identify the definition of "deserialize"

- When we take some data and turn it into a string, so our program can send it to another computer.

5. Correctly identify the definition of "serialize"

- When we take some text and turn it into data.

Storage

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

```
localStorage.setItem("eatz", "I <3 falafel");
```

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

```
localStorage.getItem("paper-trail");
```

Notes

Cookies and Web Storage

Cookies

What is a cookie?

- A small filed stored on a user's computer that holds a bite-sized amount of data, under 4KB. The are included with HTTP requests.

What are cookies used for?

- Store stateful information about a user (personal info, browser habits, history, form input information.)
- Storing a **session cookie** on user login/validation. (These are lost once the browser window is closed)
- **Persistent Cookies** : can be used to ensure a cookie survives past a specified expiration date.

How to create a cooke in Javascript

```
document.cookie = aNewCookieHere;
```

- Syntax for creating a new cookie.

```
const firstCookie = "favoriteCat=million";
document.cookie = firstCookie;
document.cookie; // Returns "favoriteCat=million"
```

- We can also delete a cookie by setting an expiration date like so.

```
document.cookie = "favoriteCat=; expires = Thu, 01 Jan 1970 00:00:00 GMT";
document.cookie; // ""
```

- You can also manually delete cookies by going into your Dev Tools and right clicking on a cookie.

Web Storage API

- **Web Storage API** : Provides mechanisms by which browsers can store key-value pairs, in a much more intuitive fashion than using cookies.
- Local Storage & Session Storage are included.
- **Local Storage** :

    - Stores data with no expiration date
    - Deleted when clearing the browser cache
    - Has the maxiumum storage limit in the browser
- **Session Storage** :
    - Stores data only for a single session
    - Stores until a browser window or tab is closed
    - Never transfers data to the server
    - Has a storage limit of 5MB (cookies are 4KB)

```
let field = document.getElementById("field");

if (sessionStorage.getItem("autosave")) {
  field.value = sessionStorage.getItem("autosave");
}

field.addEventListener("change", function () {
  sessionStorage.setItem("autosave", field.value);
});
```

- Both methods use getItem() and setItem()

```
if (!localStorage.getItem("bgcolor")) {
  populateStorage();
}
setStyles();

const populateStorage = () => {
  localStorage.setItem("bgcolor", document.getElementById("bgcolor").value);
  localStorage.setItem("font", document.getElementById("font").value);
  localStorage.setItem("image", document.getElementById("image").value);
};

const setStyles = () => {
  var currentColor = localStorage.getItem("bgcolor");
  var currentFont = localStorage.getItem("font");
  var currentImage = localStorage.getItem("image");

  document.getElementById("bgcolor").value = currentColor;
  document.getElementById("font").value = currentFont;
  document.getElementById("image").value = currentImage;

  htmlElem.style.backgroundColor = "#" + currentColor;
  pElem.style.fontFamily = currentFont;
  imgElem.setAttribute("src", currentImage);
};
```

1. Our conditional checks to see if our local storage contains a data item called `bgcolor`
2. If it does contain the data:
    - We run setStyles() to update the page styles.
3. If it does not contain the data:
    - We first run populateStorage() to set the items, and then run setStyles() to update them.

When would we use the Web Storage API?

- Ideal for storing multiple key-value pairs.
- This Data can only be saved as strings.
- Some common use cases are:
    - Storing information about a shopping cart.
    - Saving input data on forms.
    - Tracking user preferences & buying habits.
- We can access our storage by going to the **Application Tab** on our dev tools.

JSON

`JSON` is an open-standard file format that "uses human-readable text to transmit objects consisting of key-value pairs and array data types".

JSON is a format!

- Remember that JSON is simply a format for data - it's just text!
- *JSON is just a string, it's just text*

Javascript Literal Value	JSON
true	"true"
false	"false"
12.34	"12.34"
null	"null"

String literals in JSON

- **JSON always uses double quotes for strings.**
- `Serialization` : When we turn some data into string to be sent to another computer.
- `Deserialization` : When we take some text and turn it into data.

Using the Built-in JSON Object

- **JSON.stringify(value)** : truns the value passed into it into a string.
- **JSON.parse(str)** : turns a JSON-formatted string into a JS object.

```
const array = [1, 'hello, "world"', 3.14, { id: 17 }];
console.log(JSON.stringify(array));
// prints [1, "hello, \"world\"", 3.14, {"id":17}]


const str = '[1,"hello, \\"world\\"",3.14,{"id":17}]';
console.log(JSON.parse(str));
// prints an array with the following entries:
//   0: 1
//   1: "hello, \"world\""
//   2: 3.14
//   3: { id: 17 }
```

- You will almost never write raw JSON, most of the times you will just use JSON.stringify(), or call a data service that returns JSON formatted content.

Brain Teaser

What will the following print?

```
const a = [1, 2, 3, 4, 5];
console.log(a[0]); // 1

const s = JSON.stringify(a);
console.log(s[0]); // [

const v = JSON.parse(s);
console.log(v[0]); /// 1
```

Web Storage

- One Typical use for local storage is caching data fetched from a server onto the client side.

Storing Data in Local Storage

- Web Storage exists in the window as object containing key-value pairs.

Reading Data in Local Storage

```
localStorage.setItem('eatz', 'I <3 falafel');
localStorage.setItem('coffee', 'black');
localStorage.setItem('doughnuts', '["glazed", "chocolate", "blueberry",
"cream-filled"]');

const eatz = localStorage.getItem('eatz');
const coffee = localStorage.getItem('coffee');
const doughnuts = localStorage.getItem('doughnuts');

console.log(eatz); // 'I <3 falafel'
console.log(coffee); // 'black'
console.log(doughnuts); // '["glazed", "chocolate", "blueberry", "cream-filled"]'
```

- Use setItem() to assign a key-value pair to local storage.
- Use getItem() to fetch the value of a key, which is passed in as it's argument.

JSON and Local Storage

- We can construct a JS value or object from JSON by parsing it.

```
const doughnuts = JSON.parse(localStorage.getItem("doughnuts"));
```

--