Name _____ Period _____ Role (Circle one) Programmer/Driver

Name _____ Period _____ Role (Circle one) Programmer/Driver

Document Object Model

Your Tasks (Mark these off as you go)

- ☐ Explain the Document Object Model (DOM)
- ☐ Draw a DOM tree representation of an HTML document
- ☐ Have Ms. Pluska check off the above tasks
- ☐ Interpret the parent-child relationship of the DOM
- ☐ Write code to manipulate DOM elements
- ☐ Have Ms. Pluska check off the above tasks
- ☐ Receive credit for the group portion of this lab

☐ Explain the Document Object Model (DOM)

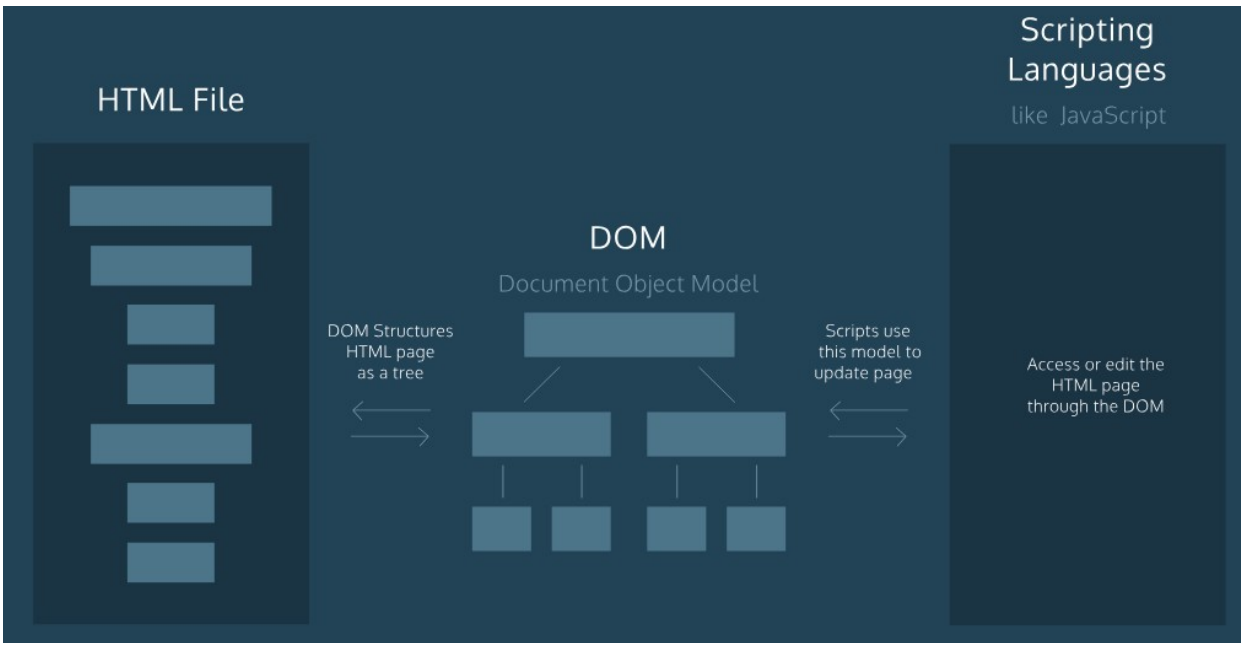
The Document Object Model, abbreviated DOM, is a powerful tree-like structure that allows programmers to conceptualize hierarchy and access the elements on a web page.

The DOM is one of the better-named acronyms in the field of Web Development. In fact, a useful way to understand what DOM does is by breaking down the acronym but out of order:

- The DOM is a logical tree-like **M**odel that organizes a web page's HTML **D**ocument as an **O**bject.

The DOM is a language-agnostic structure implemented by browsers to allow for web scripting languages, like JavaScript, to access, modify, and update the structure of an HTML web page in an organized way.

For this reason, we like to think of the DOM as the link between an HTML web page and scripting languages.



The video below illustrates how the javascript can be applied to access and manipulate an HTML document.

https://www.youtube.com/watch?v=iwNUJU5D3aI&feature=emb_title

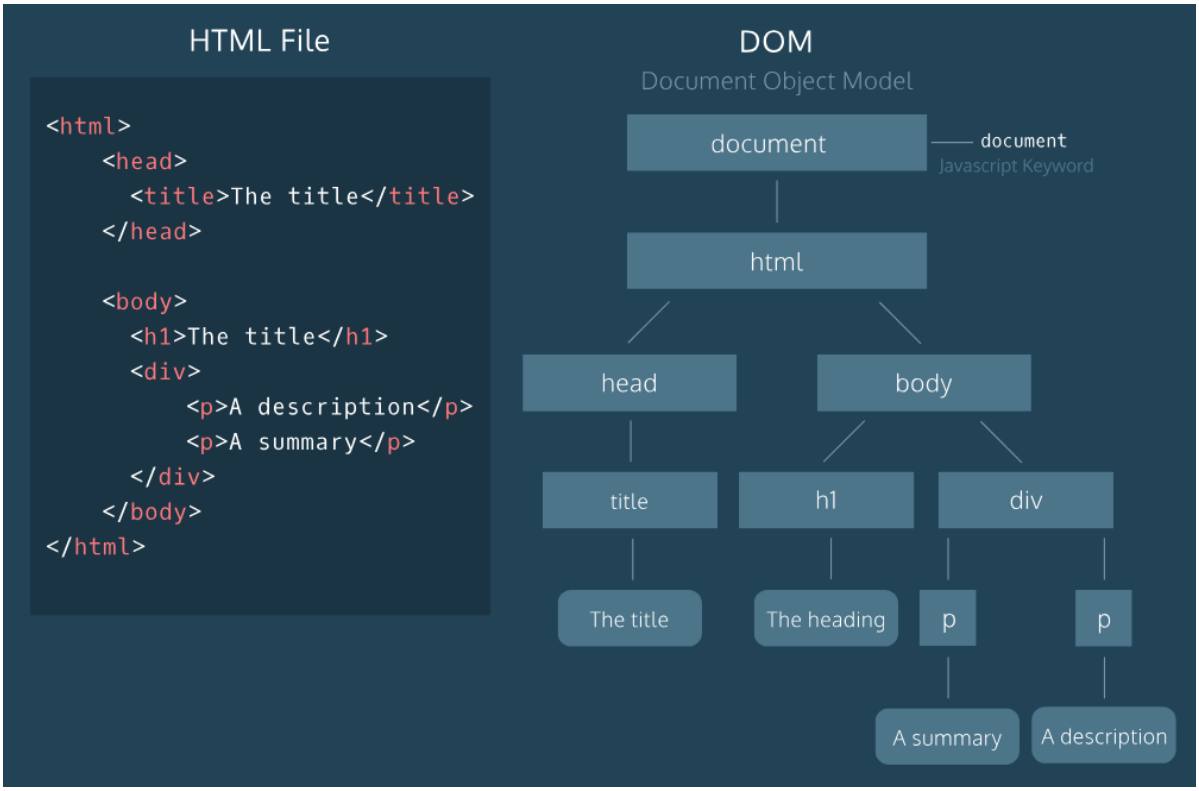
□ Draw a DOM tree representation of an HTML document

Tree-like modeling is used in many fields, including evolutionary science and data analytics. Perhaps you're already familiar with the concept of family trees: these charts represent the familial relationships amongst the descendants of a given family name.

The DOM tree follows similar logic to that of a family tree. A family tree is made up of family members and their relationships to the family name. In computer science, we would call each family member a *node*.

We define a *node* as an intersecting point in a tree that contains data. In the DOM tree, the top-most node is called the root node, and it represents the HTML document. The descendants of the root node are the HTML tags in the document, starting with the <html> tag followed by the <head> and <body> tags and so on.

The diagram below models the HTML document and labels the root element, which is the document. Observe the difference in the rectangular boxes and the curved boxes. These denote a difference in the types of nodes in the DOM structure.



There are nine different types of node objects in the DOM tree. In our diagram, the node objects with the sharp-edge rectangles are of the type *Element*, while the rounded edge rectangles are of type *Text*, because they represent the text inside the HTML paragraph elements.

When trying to modify a web page, the script will mostly interact with the DOM nodes of type *Element*. Elements are the building units of HTML web pages, they contain everything between an opening tag and a closing tag. If the tag is a self-closing tag, then that is the element itself.

Draw a DOM tree to represent the html document below	
Index.html	**DOM tree**

`<!DOCTYPE html>`
`<html>`
`  <head>`
`  <title>About Me</title>`
`  </head>`
`  <body>`
`  <h1> My Hobbies </h1>`
 	
`  <h2> Soccer </h2>`
`  <h4> 5 years </h4>`
 	
`  </body>`
`</html>`	

☐ Have Ms. Pluska check off the above tasks

Before you continue have Ms. Pluska check off the above tasks

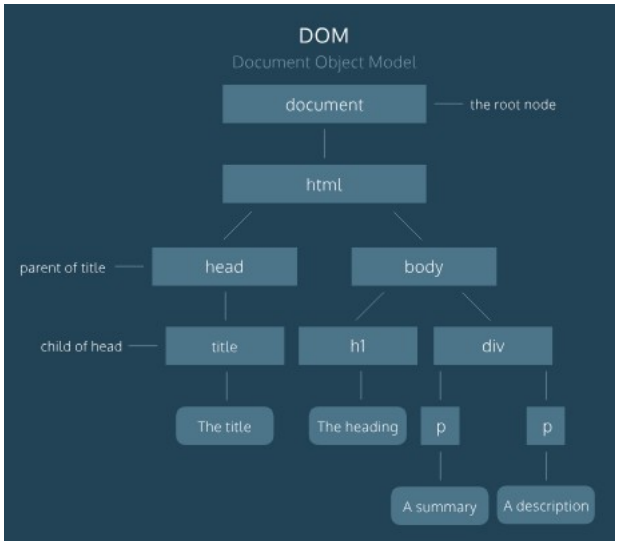
Do not continue until you have Ms. Pluska's (or her designated TA's) signature _____

☐ Interpret the parent-child relationship of the DOM

Following the metaphor of a family tree, let's define some key terminology in the DOM hierarchy:

- A *parent node* is the closest connected node to another node in the direction towards the root.
- A *child node* is the closest connected node to another node in the direction away from the root.

Knowing these terms will allow you to understand and discuss the DOM as a tree-like structure. In fact, you will also see this terminology used when referring to the nesting structure of HTML code. Programmers refer to elements nested inside other elements as the children elements and parent elements respectively.

Refer to the DOM tree below. Write the appropriate HTML elements int the **index.html** file so that it reflects the tree-diagram.	
DOM tree	**Index.html**
	

□ Access DOM elements

The *document* object in JavaScript is the door to the DOM structure. The *document* object allows you to access the root node of the DOM tree. Before you can access a specific element in the page, first you must access the document structure itself.

The *document* object allows scripts to access children of the DOM as properties.
For example, if you wanted to access the <body> element of a page in your script, you could access it as a property of the document by typing the code below,

```
var b = document.body;
```

Similarly, you could access the <title> element with the .titleproperty.

```
var t = document.title;
```

See a comprehensive list (https://developer.mozilla.org/en-US/docs/Web/API/Document) of all document properties.

Indicate what is printed to the console

Index.html

```
<!DOCTYPE html>
<html>
  <head>
  <script src = "App.js" defer></script>
  <title>About Me</title>
  </head>
  <body>
    <h1> My Hobbies </h1>

    <div id = “sports”>
        <h2>Soccer</h2>
        <h4>5 years</h4>

        <h2>Swimming</h2>
        <h4>2 years</h4>
    </sports>

  </body>
</html>
```

App.js

(a)
```
var h = document.head;
console.log(h);
```

(b)
```
var d = document.body;
var p = d.firstElementChild;
console.log(p);
```

(c)
```
var sports =
document.getElementById("sports");

console.log(sports.children);
```

(d)
```
var s =
document.getElementById("sports").f
irstElementChild;
console.log(s);
```

Manipulate DOM elements

When using the DOM in your script to access an HTML element, you also have access to all of that element's properties. This includes the ability to modify the contents of the element as well as its attributes and properties— that can range from modifying the text inside a *p* element to assigning a new background color to a *div*.

Changing the contents of an element

You can access and set the contents of an element with the *.innerHTML* property.
For example, the following code reassigns the inner HTML of the body element to the text 'The cat loves the dog':

```
document.body.innerHTML = "The cat hates the dog";
```

The *.innerHTML* property can also add any valid HTML, including properly formatted elements. The following example assigns an *h2* element as a child inside the *<body>* element:

```
document.body.innerHTML = '<h2>This is a heading</h2>';
```

What if we wanted to select a specific element? The DOM interface allows us to access a specific element with CSS selectors. CSS selectors define the elements to which a set of CSS rules apply, but we can also use these same selectors to access DOM elements with our script! Selectors can include the name of the tag, a class, or an ID.

The *.querySelector()* method allows us to specify a CSS selector and then returns the first element that matches that selector. The following code would return the first paragraph in the document.

```
document.querySelector('p');
```

Another option, if you want to access elements directly by their *id*, you can use the aptly named *.getElementByID()* function:

```
document.getElementById('bio').innerHTML = 'The description';
```

The example above selects the element with an ID of 'bio' and set its *.innerHTML* to the text 'The description'.

(a) Indicate what the page looks like after the following javascript is applied.
(b) The elements with id = time, do not display as expected. Fix the code so they display properly.

Index.html

```
<!DOCTYPE html>
<html>
  <head>
  <script src = "App.js" defer></script>
  <title>About Me</title>
  </head>
  <body>
    <h1 id = "about"> About Me </h1>

    <h2 id = "sport"> Soccer </h2>
    <h4 id = "time"> 5 years </h4>

    <h2 id = "hobbie">Baking</h2>
    <h4 id = "time"> 10 years </h4>

    <h2 id = "pet">Cat</h2>
    <h4 id = "time"> 6 years </h4>
  </body>
</html>
```

App.js

```
document.getElementById("sport").innerHTML = "Softball";

document.getElementById("time").innerHTML = "3 years";

document.getElementById("about").innerHTML = "Wigglesworth";

var p2 =
document.getElementById("pet");
p2.innerHTML = "Gecko";
```

Changing the style of an element

Another way to modify an element is by changing its CSS style. The .style property of a DOM element provides access to the inline style of that HTML tag.

The syntax follows an *element.style.property* format, with the property representing a CSS property. For example, the following code selects the element with *id=about* and assigns blue as the background-color:

```
var about = document.getElementById("about");
about.style.backgroundColor = "blue";
```

Write code to style the content as shown

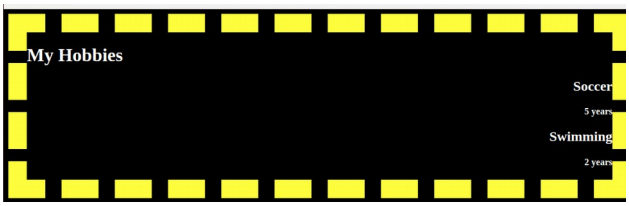
Index.html

```
<!DOCTYPE html>
<html>
  <head>
  <script src = "App.js"
defer></script>
  <title>About Me</title>
  </head>
  <body>
    <h1 id = "about"> About Me
</h1>

    <h2 id = "sport"> Soccer </h2>
    <h4 id = "time"> 5 years </h4>

    <h2 id = "hobbie">Baking</h2>
    <h4 id = "time"> 10 years </h4>

    <h2 id = "pet">Cat</h2>
    <h4 id = "time"> 6 years </h4>
  </body>
</html>
```



App.js

☐ Receive Credit for the group portion of this lab

- Make sure both you and your partner have completed the above tasks
- Indicate the names of all group members.
- Have Ms. Pluska check off the group tasks
- Submit your lab to the needs to be graded folder to receive credit for the group portion of this lab.
- Do not submit your lab until you have Ms. Pluska's (or her designated TA's) signature

_____