The DOM

Warmup: Questions about arrays and the getBiggest function?

```
var arrayOfNums = [2, 7, 7, 3, 9, 0, 1, 6, 8, 3, 8, 4, 7, 9];

function getBiggest(array) {
```

Initialize a variable to keep track of the biggest item so far.

Use a for loop to look at each item in the array.

If the current item is bigger than the biggest one so far, then make the current item the biggest one.

After we get to the end of the array, return the variable with the biggest item.

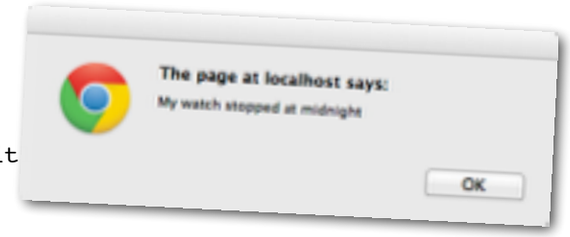
```
}

var biggest = getBiggest(arrayOfNums);
console.log("The biggest is: ", biggest);
```

Warmup: Questions objects and the to do objects?

Warmup

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Dr. Evel's Secret Code Page</title>
</head>
<body>
  <p id="code1">The eagle is in the</p>
  <p id="code2">The fox is in the</p>
  <p id="code3">snuck into the garden last night.</p>
  <p id="code4">They said it would rain</p>
  <p id="code5">Does the red robin crow at</p>
  <p id="code6">Where can I find Mr.</p>
  <p id="code7">I told the boys to bring tea and</p>
  <p id="code8">Where's my dough? The cake won't</p>
  <p id="code9">My watch stopped at</p>
  <p id="code10">barking, can't fly without umbrella.</p>
  <p id="code11">The green canary flies at</p>
  <p id="code12">The oyster owns a fine</p>
<script>
  var access = document.getElementById("code9");
  var code = access.innerHTML;
  code = code + " midnight";
  alert(code);
</script>
</body>
</html>
```

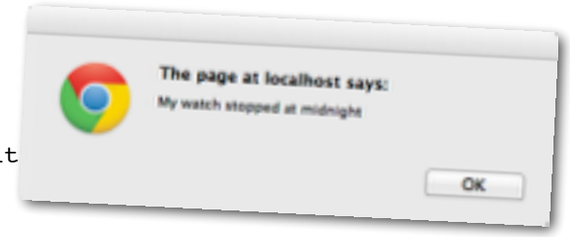
```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Dr. Evel's Secret Code Page</tit
</head>
<body>
  <p id="code1">The eagle is in the</p>
  <p id="code2">The fox is in the</p>
  <p id="code3">snuck into the garden last night.</p>
  <p id="code4">They said it would rain</p>
  <p id="code5">Does the red robin crow at</p>
  <p id="code6">Where can I find Mr.</p>
  <p id="code7">I told the boys to bring tea and</p>
  <p id="code8">Where's my dough? The cake won't</p>
  <p id="code9">My watch stopped at</p>
  <p id="code10">barking, can't fly without umbrella.</p>
  <p id="code11">The green canary flies at</p>
  <p id="code12">The oyster owns a fine</p>
<script>
  var access = document.getElementById("code9");
  var code = access.innerHTML;
  code = code + " midnight";
  alert(code);
</script>
</body>
</html>
```



Step through the code:

document.getElementById("code9")

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Dr. Evel's Secret Code Page</tit
</head>
<body>
  <p id="code1">The eagle is in the</p>
  <p id="code2">The fox is in the</p>
  <p id="code3">snuck into the garden last night.</p>
  <p id="code4">They said it would rain</p>
  <p id="code5">Does the red robin crow at</p>
  <p id="code6">Where can I find Mr.</p>
  <p id="code7">I told the boys to bring tea and</p>
  <p id="code8">Where's my dough? The cake won't</p>
  <p id="code9">My watch stopped at</p>
  <p id="code10">barking, can't fly without umbrella.</p>
  <p id="code11">The green canary flies at</p>
  <p id="code12">The oyster owns a fine</p>
<script>
  var access = document.getElementById("code9");
  var code = access.innerHTML;
  code = code + " midnight";
  alert(code);
</script>
</body>
</html>
```



Step through the code:

access.innerHTML

Looking a bit closer...

An object

A method

A string corresponding to the id of an element

```
document.getElementById("code9");
```

What is *document*?

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Planets</title>
</head>
<body>
  <h1>Green Planet</h1>
  <p id="greenplanet">All is well</p>
  <h1>Red Planet</h1>
  <p id="redplanet">Nothing to report</p>
  <h1>Blue Planet</h1>
  <p id="blueplanet">All systems A-OK</p>
</body>
</html>
```

document is an object defined internally to the browser that gives you access to all the elements in your web page.

Let's use a simple example: planets.

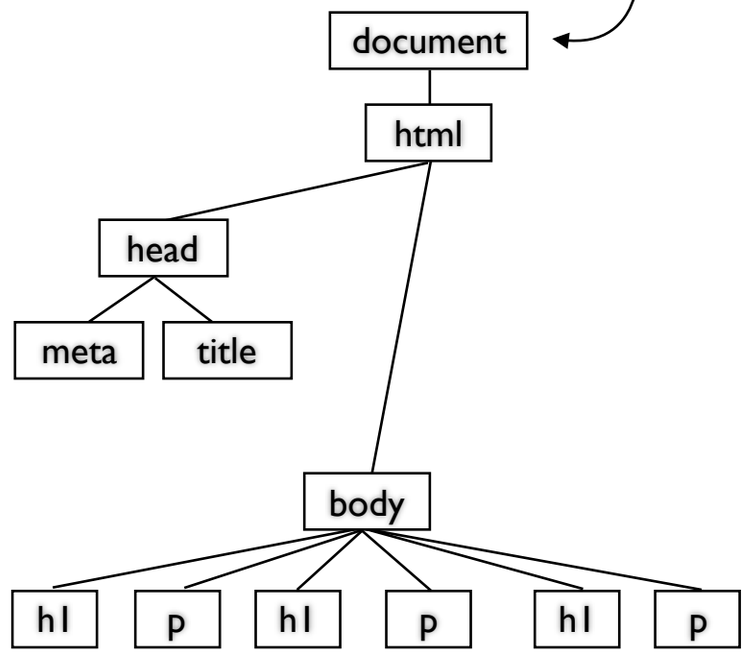
Go ahead and type this in and save the file as planets.html, and then load it in your browser.

What is *document*?

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Planets</title>
</head>
<body>
  <h1>Green Planet</h1>
  <p id="greenplanet">All is well</p>
  <h1>Red Planet</h1>
  <p id="redplanet">Nothing to report</p>
  <h1>Blue Planet</h1>
  <p id="blueplanet">All systems A-OK</p>
</body>
</html>
```

What you write.

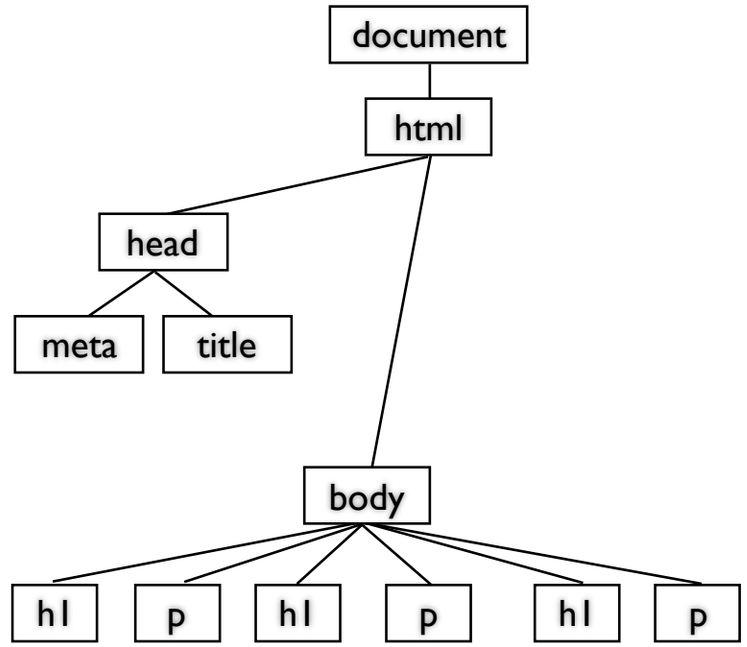
What the browser creates.



When the browser loads your web page, it creates an internal representation of your page, consisting of objects, each object representing an element on the page.

We call this internal representation of the page the "Document Object Model".
We often say "the DOM tree"... if you look at it, it's like an upside down tree.
It's also a "tree" in the computer science technical sense:
http://en.wikipedia.org/wiki/Tree_(data_structure)

The DOM is like an upside down tree:



All the elements in your page are in this tree, inside the browser.

Photo: https://www.flickr.com/photos/oscarparadela/6848647647/

The root is the document object,
the branches are the main elements
the leaves are the nested elements

The document object has several methods for accessing elements. The one you'll find yourself using most often at first is *getElementById*.

```
document.getElementById("greenplanet");
```

The document object has several methods for accessing elements. The one you'll find yourself using most often at first is *getElementById*.

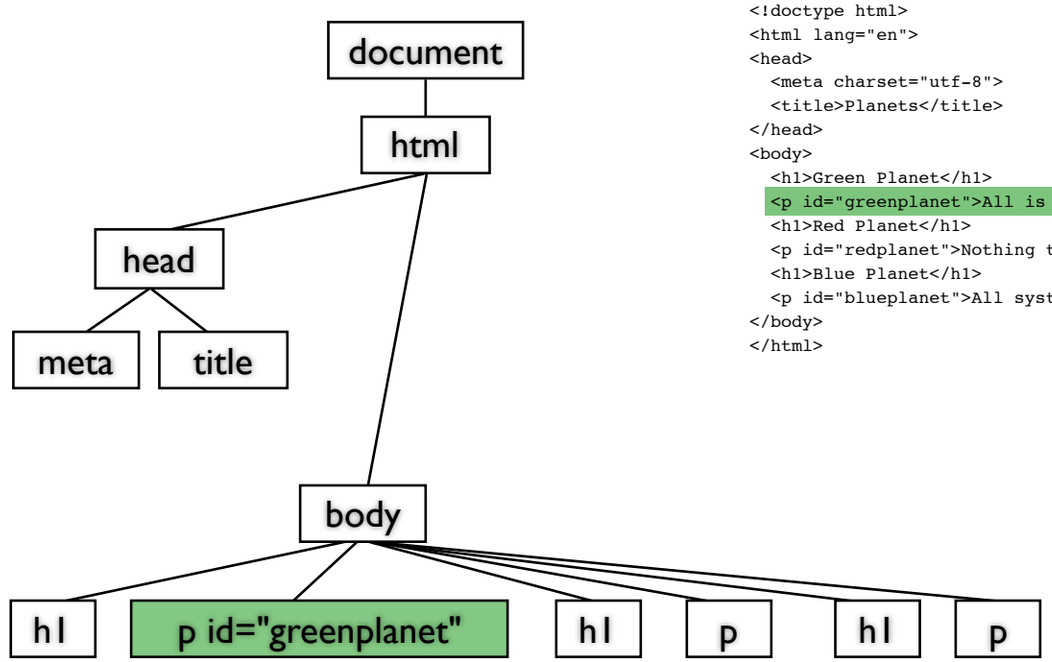
An object

A method

A string corresponding to the id of an element

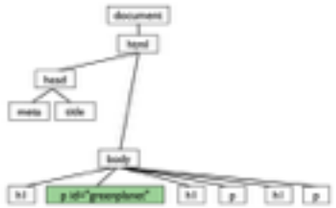
```
document.getElementById("greenplanet");
```

Note: remember that HTML elements should have unique ids. That is, there should only be one element with the id "greenplanet" in your page.




```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Planets</title>
</head>
<body>
  <h1>Green Planet</h1>
  <p id="greenplanet">All is well</p>
  <h1>Red Planet</h1>
  <p id="redplanet">Nothing to report</p>
  <h1>Blue Planet</h1>
  <p id="blueplanet">All systems A-OK</p>
</body>
</html>
```

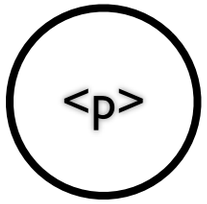
```
document.getElementById("greenplanet");
```



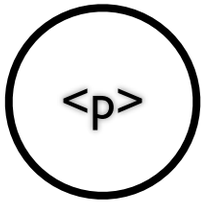
p id="greenplanet"

```
var planet = document.getElementById("greenplanet");
```

The value returned from calling the getElementById method of the document object is an element object.

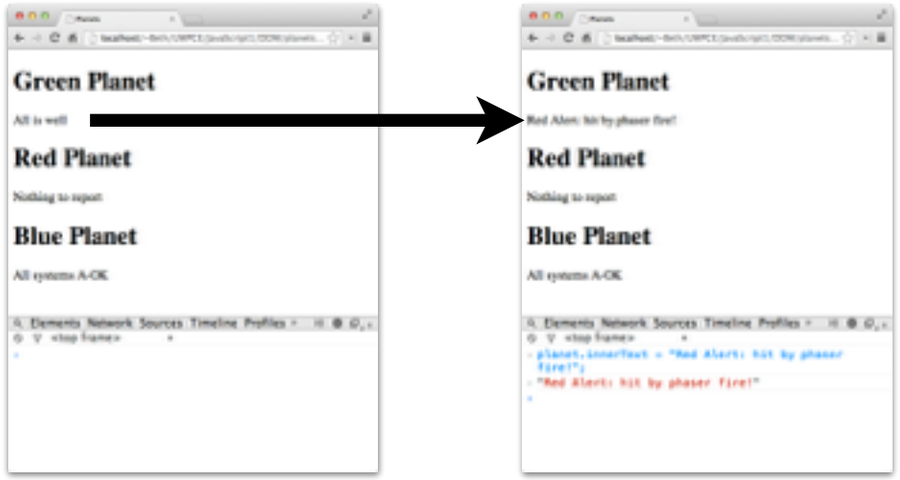


Remember: you can only get one element back using getElementById because you should have only ONE element in your HTML page with that id.

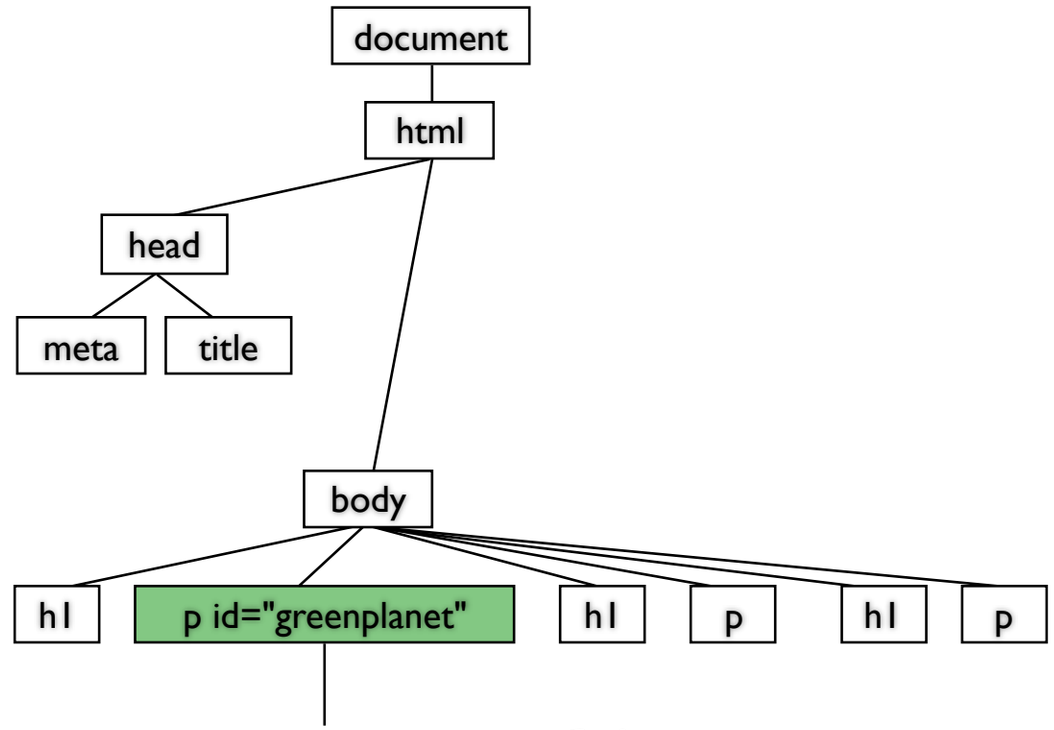


The element object that represents the <p> element in your HTML has properties, like *innerHTML*.

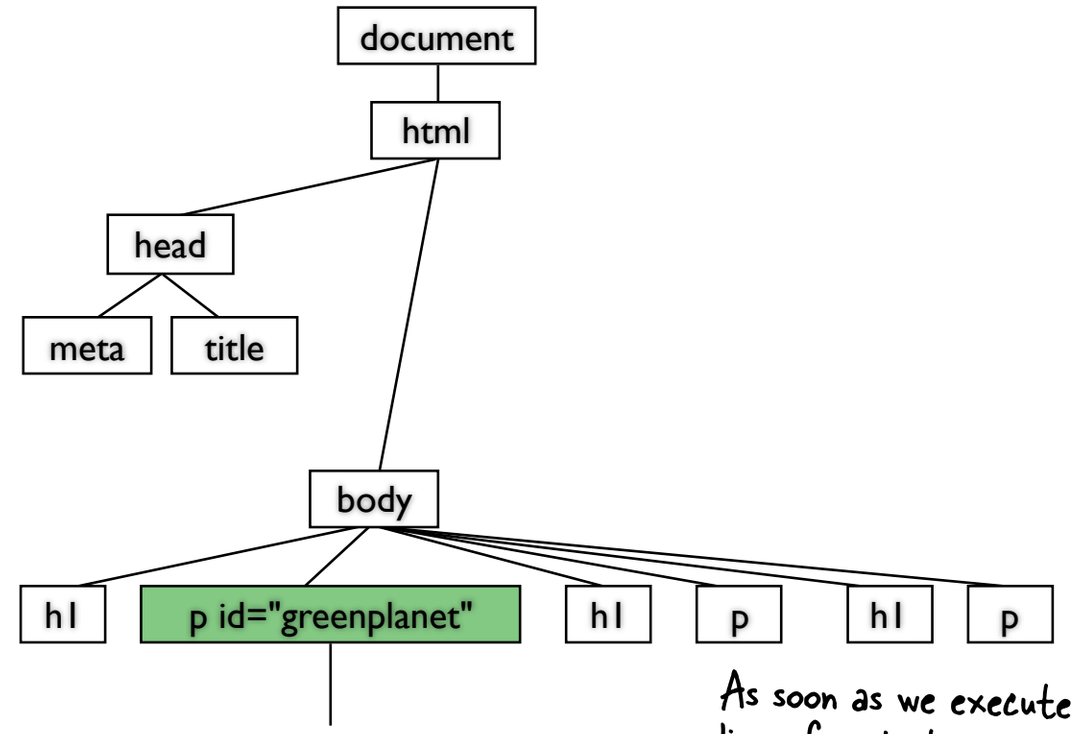
```
var planet = document.getElementById("greenplanet");
planet.innerHTML = "Red Alert: hit by phaser fire!";
```



Notice that as soon as the code is executed to set the innerHTML of the planet <p> element, your page changes to reflect the new content.



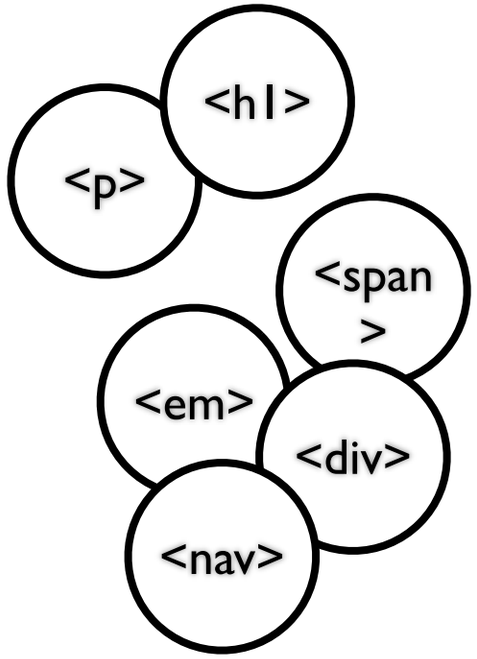
Before we change the content of the <p> element, the content is "All is well".



"Red Alert: hit by phaser fire!"

As soon as we execute the line of code to change the content with innerHTML, the DOM changes, and your page changes.

Whenever you get an element from the DOM using the document.getElementById method, you get back an *element object*.



Element objects have properties and methods to:

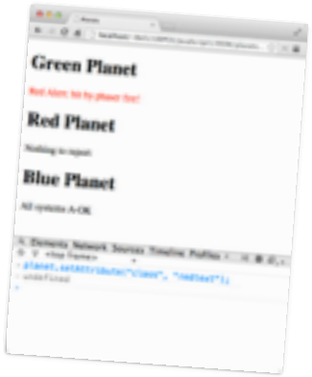
- Change the content of the element (text or HTML)
- Read an attribute
- Add an attribute
- Change an attribute
- Remove an attribute
- ... and more

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Planets</title>
</head>
<body>
  <h1>Green Planet</h1>
  <p id="greenplanet">All is well</p>
  <h1>Red Planet</h1>
  <p id="redplanet">Nothing to report</p>
  <h1>Blue Planet</h1>
  <p id="blueplanet">All systems A-OK</p>
  <script>
    var planet = document.getElementById("greenplanet");
    planet.innerHTML = "Red Alert: hit by phaser fire!";
  </script>
</body>
</html>
```

Go ahead and type this in now.

Load the page. What do you see in the first paragraph under "Green Planet"?

Let's add a "class" attribute to <p> to change the style of the element:



- First, add this style to your HTML in the <head>:

```
<style>
    .redtext { color: red; }
</style>
```

- Next, we'll use the element object's setAttribute method to add a "class" attribute to the <p> element.

```
var planet = document.getElementById("greenplanet");
planet.innerText = "Red Alert: hit by phaser fire!";
planet.setAttribute("class", "redtext");
```

Add this code to your planets.html file, and reload the page.

Your turn:

Write code to get the "redplanet" and "blueplanet" elements and change the style and/or content of these elements to whatever you like.

Try using document.getElementById to get an element that doesn't exist in the page. What happens? E.g.:

```
var planet = document.getElementById("pinkplanet");
console.log(planet);
planet.innerHTML = "I'm a pink planet";
```

When you try to get an element that doesn't exist, null is returned.
Think of null as meaning "an object that doesn't exist".

null doesn't have any properties, so you can't access the innerHTML or innerText properties.
So when you try to set the innerHTML property of an object that doesn't exist, you get an error.

If you use document.getElementById to try to get an element that doesn't exist in the page, you get back the value *null*.

The value of planet is null.

```
var planet = document.getElementById("pinkplanet");
console.log(planet);
planet.innerHTML = "I'm a pink planet";
```

So this code causes an error. If you want to be sure you've got an element, you need to make sure it's not null!

```
var planet = document.getElementById("pinkplanet");
if (planet != null) {
    console.log(planet);
    planet.innerHTML = "I'm a pink planet";
} else {
    console.log("Error! No pink planet");
}
```

We'll talk a lot more about what null is later.

Where you put your script matters...

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Planets</title>
  <style>
    .redtext { color: red; }
  </style>
  <script>
    var planet = document.getElementById("greenplanet");
    planet.innerHTML = "Red Alert: hit by phaser fire!";
    planet.setAttribute("class", "redtext");
  </script>
</head>
<body>
  <h1>Green Planet</h1>
  <p id="greenplanet">All is well</p>
  <h1>Red Planet</h1>
  <p id="redplanet">Nothing to report</p>
  <h1>Blue Planet</h1>
  <p id="blueplanet">All systems A-OK</p>
  <script>
    var planet = document.getElementById("greenplanet");
    planet.innerHTML = "Red Alert: hit by phaser fire!";
    planet.setAttribute("class", "redtext");
  </script>
</body>
</html>
```

Move the script from the bottom of your page up to the <head>.

Remember that the browser handles your page top down.
So when the script is at the bottom of your page, the browser loads the page,
creates the Document Object Model (DOM) - that is the internal representation of the page -
and then executes the script as the very last thing before the end of the page.

What happens when we move the script to the top of the page??
Now, the script will be executed FIRST - before the rest of the page is loaded, and before the DOM is created.

What do you think will happen?

Save your file, and reload the page to find out....

```
> Uncaught TypeError: Cannot set property 'innerHTML' of null
```

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Planets</title>
  <style>
    .redtext { color: red; }
  </style>
  <script>
    var planet = document.getElementById("greenplanet");
    planet.innerHTML = "Red Alert: hit by phaser fire!";
    planet.setAttribute("class", "redtext");
  </script>
</head>
<body>
  <h1>Green Planet</h1>
  <p id="greenplanet">All is well</p>
  <h1>Red Planet</h1>
  <p id="redplanet">Nothing to report</p>
  <h1>Blue Planet</h1>
  <p id="blueplanet">All systems A-OK</p>
</body>
</html>
```

When we try to get the "greenplanet" element, it's not there because the DOM hasn't been created yet.

The script is executed before the DOM is created
So, when we try to get the <p> element with the id "greenplanet" it doesn't exist yet.
So planet is null.

When we try to set the innerHTML property of null, we get an error!

So, do we always have to put our code at the bottom?

No... We can use an *event handler*.

We can tell the browser to execute code only after the page is loaded and the DOM is created and ready to go.

```
function init() {
    var planet = document.getElementById("greenplanet");
    planet.innerText = "Red Alert: hit by phaser fire!";
    planet.setAttribute("class", "redtext");
}
window.onload = init;
```

How onload works:

1. Create a function containing the code you want to execute *after* the page has loaded and the DOM is ready:

```
function init() {
    var planet = document.getElementById("greenplanet");
    planet.innerText = "Red Alert: hit by phaser fire!";
    planet.setAttribute("class", "redtext");
}
```

2. Assign the function (using its name) to the window object's onload property:

```
window.onload = init;
```

The browser loads the page and executes the JavaScript at the top, which defines the function init and assigns the function to window.onload.

Once the page has completed loading, the browser then executes whatever function is assigned to window.onload. This function changes the page.

When the browser executes the code (as the page loads):

The function init is defined (but not executed).

```
function init() {
    var planet = document.getElementById("greenplanet");
    planet.innerText = "Red Alert: hit by phaser fire!";
    planet.setAttribute("class", "redtext");
}
window.onload = init;
```

The function init is assigned to window.onload. Notice that we use just the name "init". We leave off the () because we don't want to call init; we just want to assign it to window.onload.

When the page has loaded, and the DOM is ready and *event* is fired: the *load* event.

```
function init() {
    var planet = document.getElementById("greenplanet");
    planet.innerText = "Red Alert: hit by phaser fire!";
    planet.setAttribute("class", "redtext");
}
window.onload = init;
```

A function assigned to the window.onload property is the "load" event handler. This function is called when the "load" event is fired.

A handler is just a function that gets called when there's an event. We'll talk a lot more about events later.

Assigning a function to a variable (or property):

```
function init() {
    var planet = document.getElementById("greenplanet");
    planet.innerText = "Red Alert: hit by phaser fire!";
    planet.setAttribute("class", "redtext");
}
window.onload = init;
```

Assigning a function by name to a property of window is just like assigning a function to a property of an object like fido.

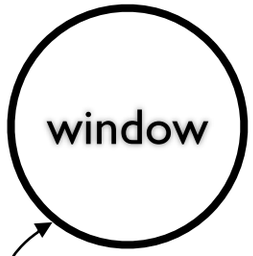
```
var fido = {
    name: "Fido",
    weight: 12,
    breed: "Mixed",
    bark: function() {
      console.log("Woof woof!");
    }
};
```

In both cases, we're assigning a function to the property of an object.

Update your code and reload the page:

```
function init() {
    var planet = document.getElementById("greenplanet");
    planet.innerText = "Red Alert: hit by phaser fire!";
    planet.setAttribute("class", "redtext");
}
window.onload = init;
```

What is *window*?

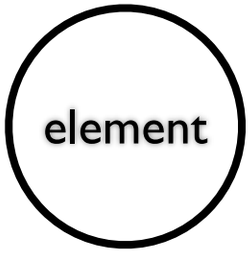


window is your "window" into all JavaScript related to the browser. This object is the "global object"; everything you do in JavaScript related to the browser and your page is done by using window's properties and methods.

window is your secret to programming the browser with JavaScript.

Use this object to access every browser-related property. *document* is one of these objects. *window.onload* is a method that's called when the page has completed loading. *window* is also the "global object".

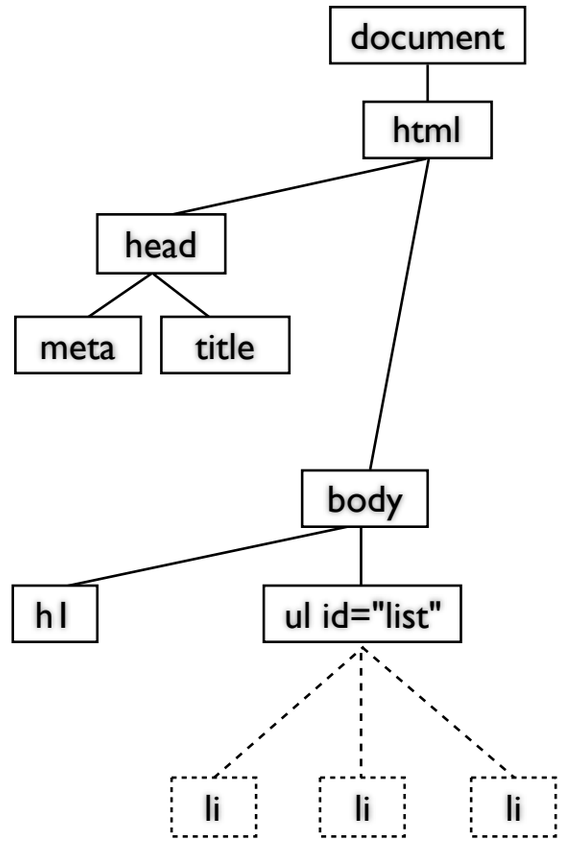
Use this object to access properties and methods of the page. *document.getElementById* is a method you can use to get an element from the page, as an element object.



When you have access to an element object, you can use the its properties and methods to change the content and attributes of an element. For instance, use *innerHTML* to change the content of an element.

Things you can do with the DOM:

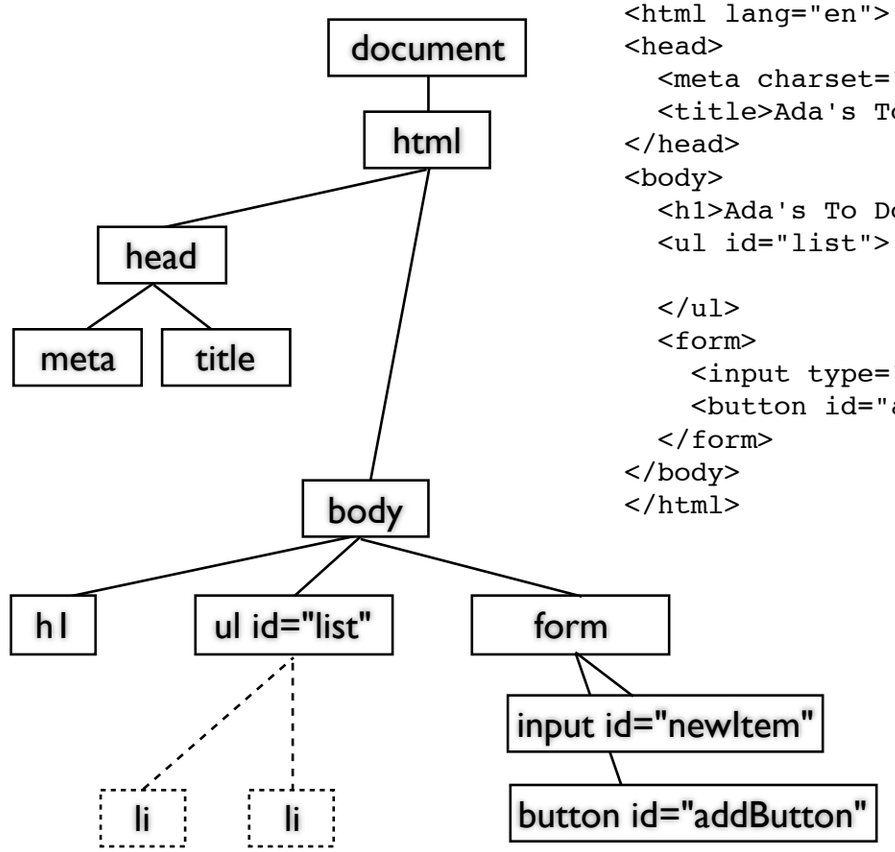
- Get elements from the DOM, by id (you've already seen this), by class, by tag name, and by CSS selector.
- Create and add new elements to the DOM (and thus, the page).
- Remove elements from the DOM.
- Traverse the elements in the DOM.
- Detect clicks on form buttons, focus in a form input, and more.
- Find out how far a page has been scrolled in the browser window.
- And much more!




```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Ada's To Do List</title>
</head>
<body>
  <h1>Ada's To Do List</h1>
  <ul id="list">

  </ul>
</body>
</html>
```

We want to create new list items and add them to the list.

list.html STEP THROUGH CODE




```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Ada's To Do List</title>
</head>
<body>
  <h1>Ada's To Do List</h1>
  <ul id="list">

  </ul>
  <form>
    <input type="text" id="newItem">
    <button id="addButton">Add</button>
  </form>
</body>
</html>
```

We want to create new list items with text from the form input and add them to the list when we click the button.

list.html STEP THROUGH CODE

Project

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Ada's To Do List</title>
</head>
<body>
<header>
    <h1>Ada's To Do List</h1>
</header>
<section>
    <form>
        <input type="text" id="task" placeholder="task">
        <button id="addTask">add</button>
    </form>
    <ul id="list">

    </ul>
</section>
</body>
</html>
```

```
<li>
  <span class="check"></span>
  <span>to do item</span>
</li>
```

Todo list manager: Silver

- Form to create a new to do item.
  - You can assign an event handler to a button using the button's onclick property.
  - You can get a value from an <input> element by using it's value property.
  - You can check to see if the input value is null or "" to make sure you don't add empty input values to the list.
- Add new to do items to a list.
  - To create a new element, use document.createElement(...)
  - To add that element to the page, use element.appendChild(...)

Todo list manager: Platinum

- Form to create a new to do item.
- Add new to do items to a list.
- If you click on a list item change it to done.
- Use <li> classes "done" and "notdone" to keep track of if an item is done or not.
- If you mark all the items in a list "done", ask the user if they want to remove all the items in the list.
  - Use querySelectorAll to find all items with the class "notDone"
  - Look up removeChild to see how to remove a child from a list.
  - Look up firstElementChild to see how to find out if there are any items left in the list.