An Brief Introduction to a web browser's

Document Object Model (DOM)

Lecture 20 – COMP110 – Spring 2018

Final Exams

- We will split up across this room and another. I will let you know when your assigned building and seat are posted!
- Section 1 - Saturday, December 8th at 12pm
- Section 2 - Thursday, December 13th at 4pm
- Do you have a pink slip for 3 exams in 24 hours?
  - Scan it on your phone and email it to me with the subject: **3 in 24 PINK SLIP**
  - Make-ups are nearest reading day prior at 9am
    - Section 1 - Thursday, December 6th at 9am - Sitterson 254
    - Section 2 - Wednesday, December 12th at 9am - Sitterson 254

1. PollEv: Given the Person class, what is printed when the main function runs?

```
class Person {
    name: string;
    age: number;

    constructor(name: string, age: number) {
        print("constructor");
        this.name = name;
        this.age = age;
    }

    getAge(): number {
        print("getAge");
        return this.age;
    }

    jump(): void {
        print(this.name);
    }
}
```

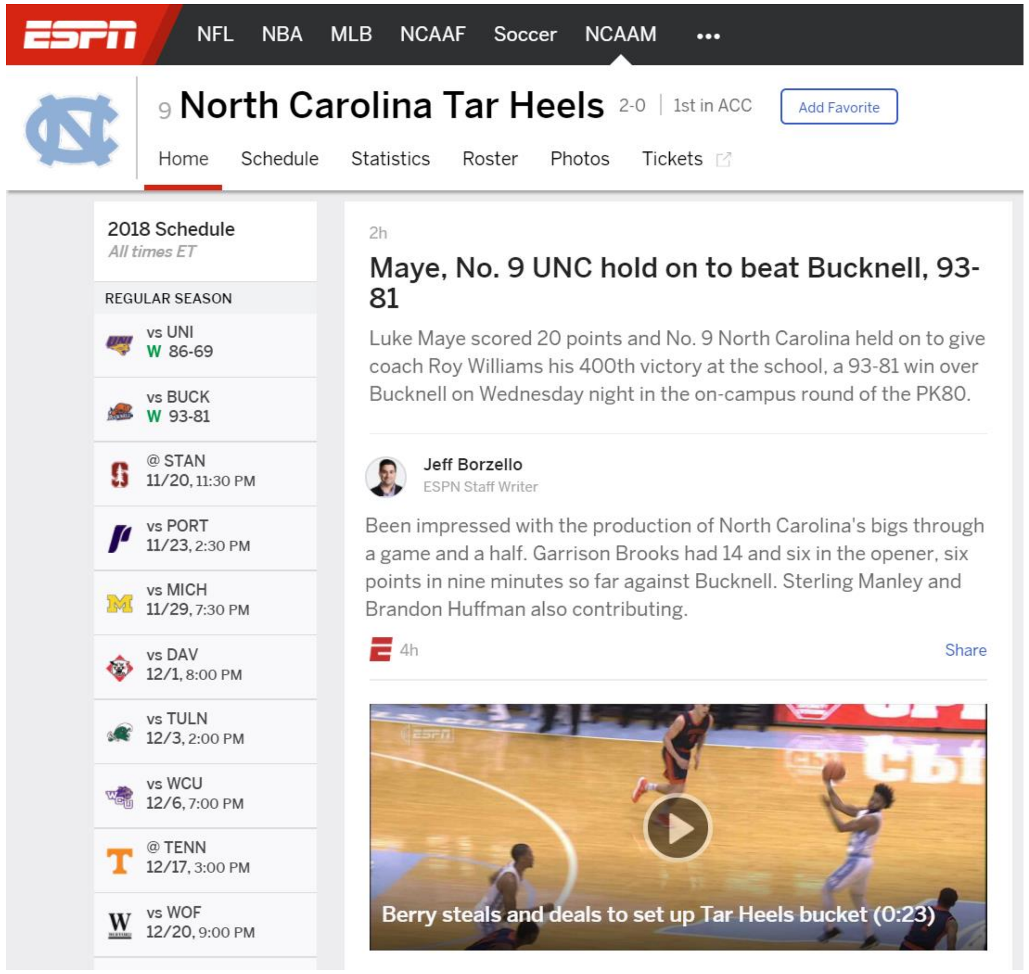
```
export let main = async () => {
    let p = new Person("Jane", 20);
    print(p.getAge());
    p.jump();
};

main();
```

2. Assume arrays have built-in filter, map, and reduce methods. What do you expect the value of the variable b to be after this code is evaluated?

```
let a = [1, 2, 3, 4];
let b = a.filter((x) => x > 2)
          .map((x) => x * 2)
          .reduce((memo, x) => memo + x, 0);
```

Document Object Model (DOM)

- Web pages are made up of a hierarchy of **objects** called the **Document Object Model or DOM**
- Single objects, like a **label** object or a text **input** field object, are added to container objects like a **form** or a **div**ision ("section") of a web page.
- Just like shapes, each object in the DOM has properties our code can manipulate:
  - background color
  - font-size
  - borders
  - positioning
  - and many, many more!



9 North Carolina Tar Heels 2-0 | 1st in ACC

2018 Schedule
All times ET

REGULAR SEASON

vs UNI	W 86-69
vs BUCK	W 93-81
@ STAN	11/20, 11:30 PM
vs PORT	11/23, 2:30 PM
vs MICH	11/29, 7:30 PM
vs DAV	12/1, 8:00 PM
vs TULN	12/3, 2:00 PM
vs WCU	12/6, 7:00 PM
@ TENN	12/17, 3:00 PM
vs WOF	12/20, 9:00 PM

2h

Maye, No. 9 UNC hold on to beat Bucknell, 93-81

Luke Maye scored 20 points and No. 9 North Carolina held on to give coach Roy Williams his 400th victory at the school, a 93-81 win over Bucknell on Wednesday night in the on-campus round of the PK80.

Jeff Borzello
ESPN Staff Writer

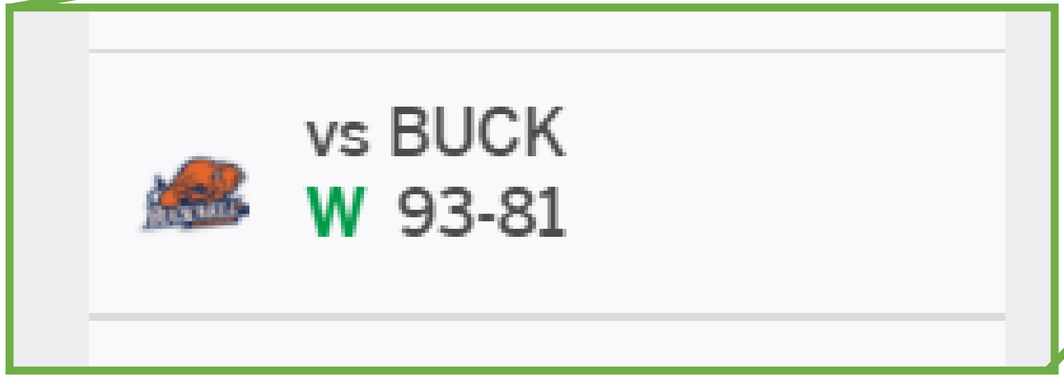
Been impressed with the production of North Carolina's bigs through a game and a half. Garrison Brooks had 14 and six in the opener, six points in nine minutes so far against Bucknell. Sterling Manley and Brandon Huffman also contributing.

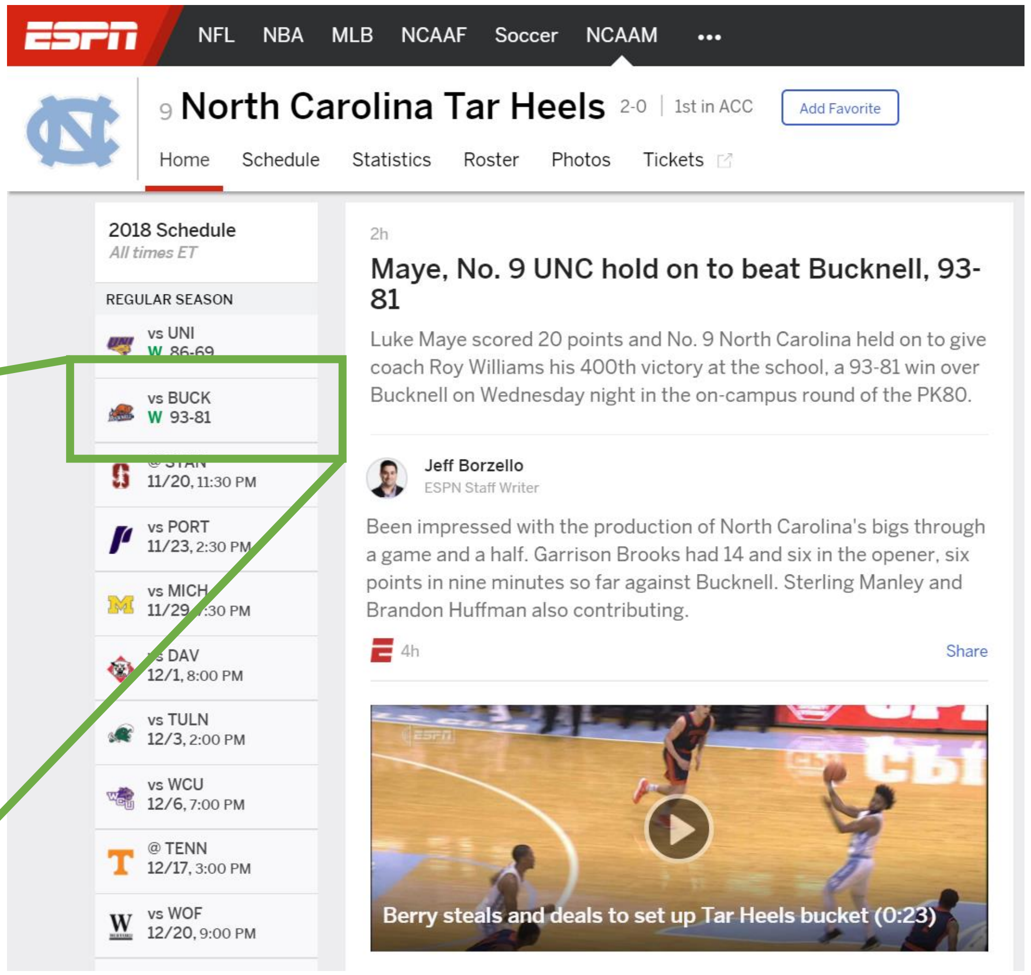
4h

Berry steals and deals to set up Tar Heels bucket (0:23)

Any given web page is made up of *many* objects

- Let's zoom in on one small part of the ESPN.com website and inspect the objects





9 North Carolina Tar Heels 2-0 | 1st in ACC

2018 Schedule
All times ET

REGULAR SEASON

vs UNI
W 86-69

vs BUCK
W 93-81

@ STAN
11/20, 11:30 PM

vs PORT
11/23, 2:30 PM

vs MICH
11/28, 7:30 PM

vs DAV
12/1, 8:00 PM

vs TULN
12/3, 2:00 PM

vs WCU
12/6, 7:00 PM

@ TENN
12/17, 3:00 PM

vs WOF
12/20, 9:00 PM

2h

Maye, No. 9 UNC hold on to beat Bucknell, 93-81

Luke Maye scored 20 points and No. 9 North Carolina held on to give coach Roy Williams his 400th victory at the school, a 93-81 win over Bucknell on Wednesday night in the on-campus round of the PK80.

Jeff Borzello
ESPN Staff Writer

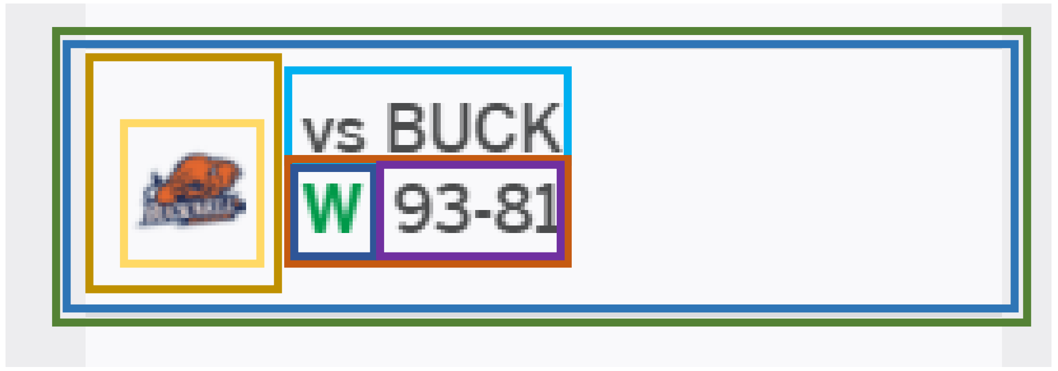
Been impressed with the production of North Carolina's bigs through a game and a half. Garrison Brooks had 14 and six in the opener, six points in nine minutes so far against Bucknell. Sterling Manley and Brandon Huffman also contributing.

4h

Berry steals and deals to set up Tar Heels bucket (0:23)

Any given web page is made up of *many* objects

- This single box on ESPN.com is composed of **8 objects!**



1. **List Item**
    2. **Anchor Link** (so that you can click it)
        3. **Division** for the logo
            4. **Image** of the Bucknell logo
        5. **Division** for the opponent "vs BUCK"
        6. **Division** for the game meta data
            7. **Division** for the result ("W")
            8. **Division** for the score

How do all the objects of a web page get initialized?

- Setting up all of a web pages' document objects (called elements) would be tedious to do via TypeScript (or JavaScript)

- Enter: Hypertext Markup Language or **HTML**

- HTML files describe the initial state of a web page's Document Object Model

```
<li>
    <a href="/game?gameId=400986059">
        <div class="logo">
            <img src="http://a.espncdn.com/.../el
        </div>
        <div class="game-info">vs BUCK</div>
        <div class="game-meta">
            <div class="game-result win">W</div>
            <div class="score">93-81</div>
        </div>
    </a>
</li>
```

HTML Element Syntax

- HTML is made up of nested **elements**
  - The syntax for an element is often called a "tag"
- For example, the **body** or "visible area" of a web page
  - Begins with the **opening tag: <body>**
  - Ends with the **closing tag: </body>**
- Inside of this web page's body is a **1st level heading** that says "Hello, world!"
  - Begins with the **opening tag: <h1>**
  - Ends with the **closing tag: </h1>**
- Learn more about HTML elements using MDN's great documentation: https://developer.mozilla.org/en-US/docs/Web/HTML/Element

```
<body>
    <h1>Hello, world!</h1>
</body>
```

Each HTML Element initializes a DOM Object

- When the web browser loads a web page, it initializes each HTML element into an object added to the DOM

- How? A recursive algorithm!

1. Each time it sees an opening tag, it constructs an object for that tag.

2. *If* the tag has nested tags, it recursively does the same thing and adds the returned tags to its group of descendant tags.

```html
<body>
    <h1>Hello, world!</h1>
</body>
```

An Element's Attributes initialize its Object's Properties

- Just like our shape objects had properties (fill, stroke, x, y, width, height, radius, etc), so do all DOM Objects

- A DOM object's properties are initialized in HTML using "attributes" in the opening tag of an HTML Element as shown below

```html
<body id="homePage" style="background:black">
   <h1 id="title" style="color:white">Hello, world!</h1>
</body>
```

- The **body** and **h1** elements each have two attributes defined: **id** and **style**.
- Each attribute's value will be assigned to the property of its DOM object.

Follow-Along:
Assigning an Attribute to the Body in creeds-tweets.html

```html
<body style="background: yellow">
```

Accessing the DOM from TypeScript

- The global **`document`** variable gives us a reference to the DOM
  - Just like the `window` variable we've used before.
- It is an object of type `HTMLDocument` and has properties and methods we can call on it
- **`body: HTMLElement`** - reference to the body element we just styled
- **`getElementById(id: string): HTMLElement`** – find an element by its id attribute
- **`createElement(tag: string): HTMLElement`** – create a new HTML Element
- **`appendChild(child: HTMLElement): void`** – appends an element to the document

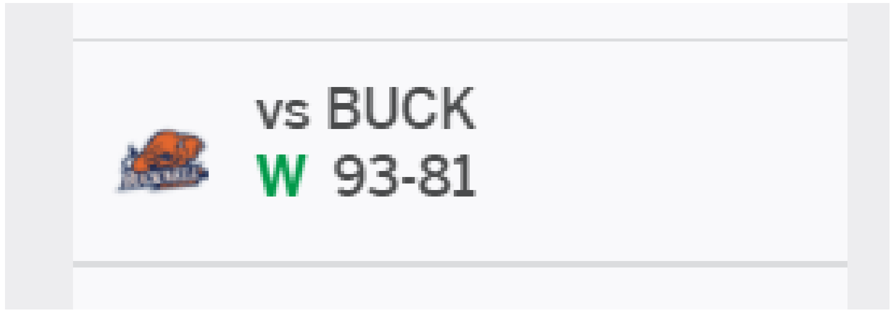
Follow-along: Accessing the DOM from TS

We've seen how to change **body**'s style property's background using an HTML attribute, now let's do it from TypeScript in code.

```
export let main = async () => {
       document.body.style.background = "lightgray";
}

main();
```

Let's look at the HTML for that box on ESPN

```
<li>
    <a href="https://espn.com/page-for-game">
        <div class="logo">
            <img src="...elided...">
        </div>
        <div class="game-info">vs BUCK</div>
        <div class="game-meta">
            <div class="game-result win">W</div>
            <div class="score">93-81</div>
        </div>
    </a>
</li>
```



- If the HTML above initializes the 8 objects that represent this portion of the web page... where are the attributes setting the x, y, colors, fonts, etc of these objects?
- There's *another language,* called cascading-stylesheets (CSS), which makes it easy to consistently assign style properties to many objects at once.
- We will not explore CSS in COMP110. There are great, free resources on-line!

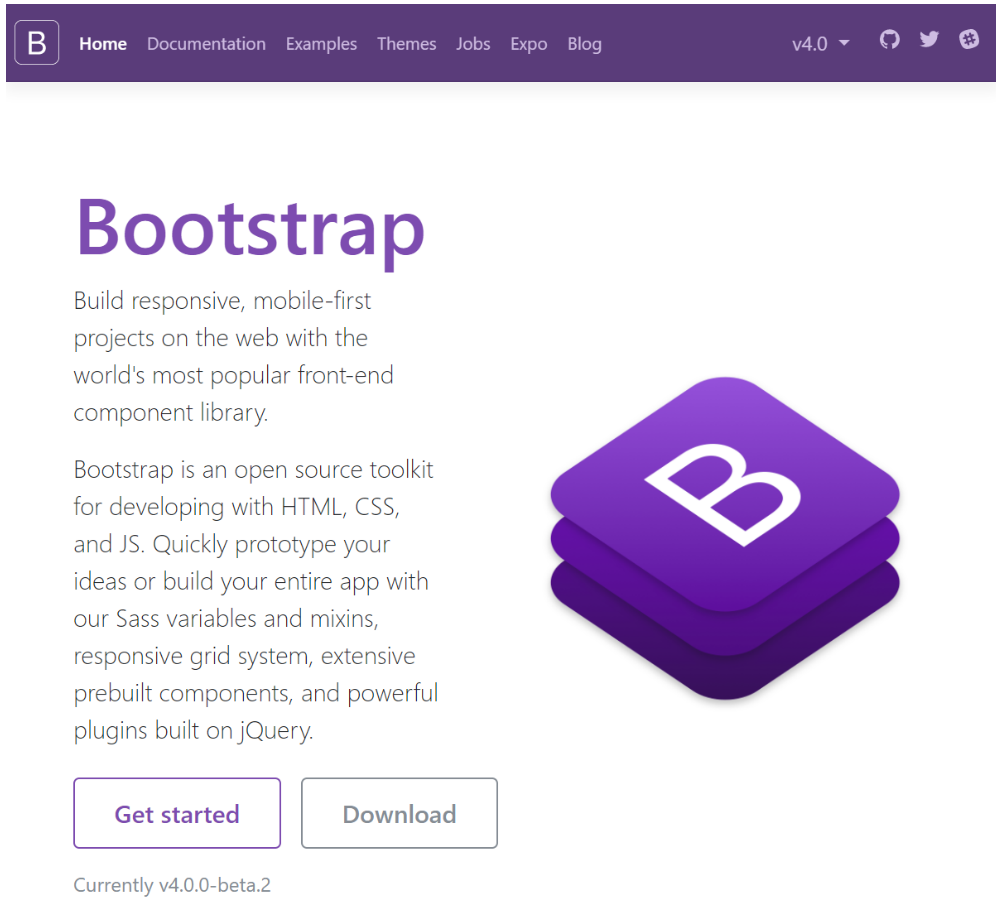
How can you start creating web pages that look OK ***without*** knowing CSS?

- Use a pre-built theme or framework!

- We'll use the popular Bootstrap CSS framework today
  - https://getbootstrap.com/
  - Great documentation and many alternative themes you can swap-in

- Beginning with its "Quick Start" documentation and "Starter Template" HTML, you can hit the ground running:
  - https://getbootstrap.com/docs/4.0/getting-started/introduction/

- As you add elements to your web page, you can browse through component documentation to find examples, e.g.:
  - Page Layout: https://getbootstrap.com/docs/4.0/layout/grid/
  - Forms: https://getbootstrap.com/docs/4.0/components/forms/



Bootstrap

Build responsive, mobile-first projects on the web with the world's most popular front-end component library.

Bootstrap is an open source toolkit for developing with HTML, CSS, and JS. Quickly prototype your ideas or build your entire app with our Sass variables and mixins, responsive grid system, extensive prebuilt components, and powerful plugins built on jQuery.

Currently v4.0.0-beta.2

How can our TypeScript code interact with a **specific element** in the DOM?

- Any HTML element can be given an "ID attribute" that will serve as its "identifier"

HTML

```
<div id="tweets"></div>
```

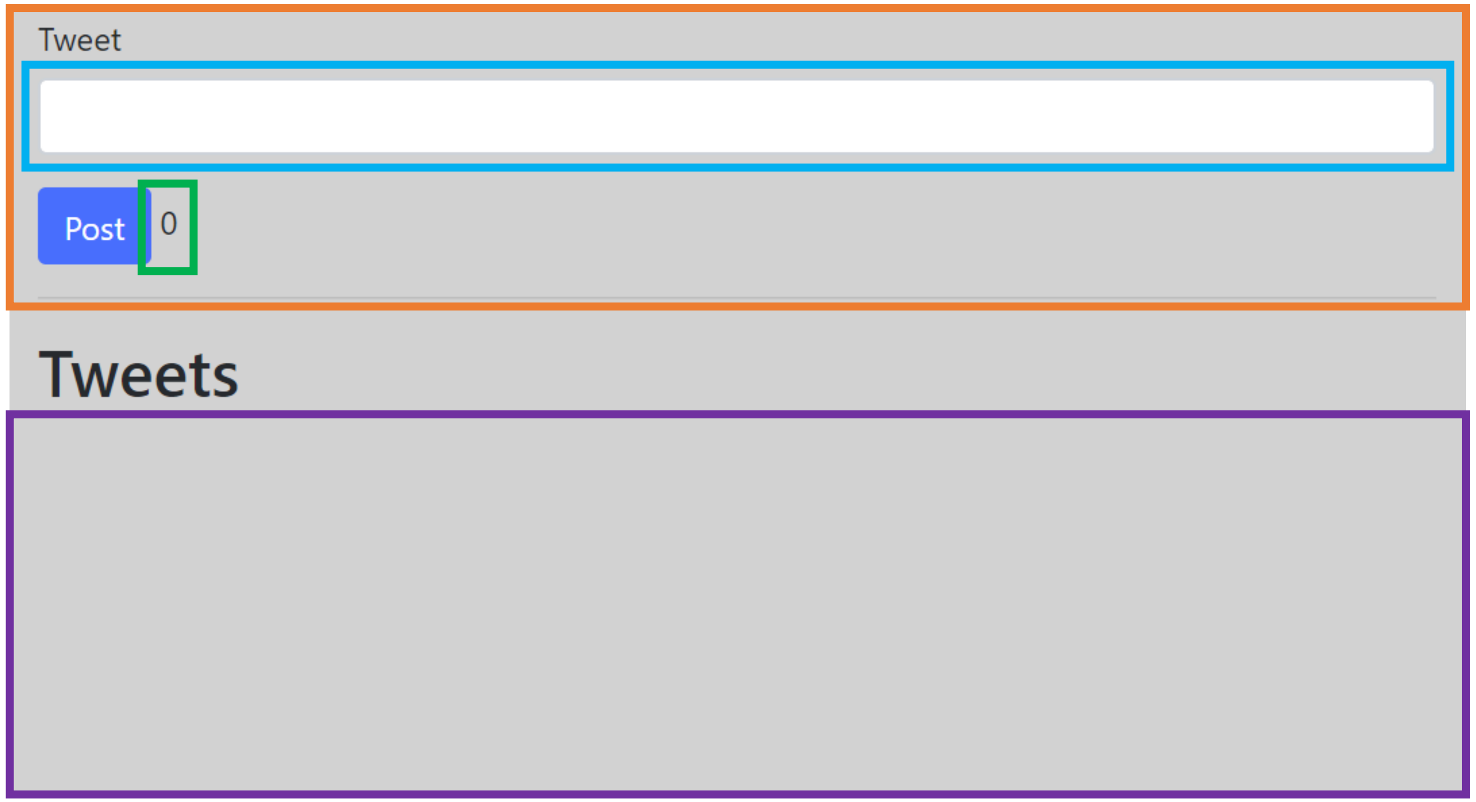
- By assigning an **`id`** attribute to an element, we can "get" that element from the DOM via a method call.

TypeScript

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

The **id** Attributes of our App's Elements

- `text`
  - The <input type="text"> field where we'll write a tweet
- `characters`
  - The <span> element where the # of characters typed will show
- `form`
  - The <form> container that the text field and post button are on
- `tweets`
  - The <div> container where we will post our tweets



We can view creeds-tweets.html to see each of these elements and its id being established in HTML.

`document`'s `getElementById` method

- Its return type is:
  `HTMLElement | null`

```
document.getElementById("tweets")
```

- This means it will either return to you an HTMLElement *or* **`null`**
  - There is no guarantee an element with that **id** actually exists in your DOM.

- What if we *know* it's not **`null`**?
  - We can override the possibility of null with a "type assertion"

```
document.getElementById("tweets") as HTMLElement
```

Part I) Updating Character Count (1 / 3)

- Let's setup the text field so that when we type into it, our character count updates. To do this, we need to establish references to two objects in the DOM:

1. The input element where we type our text. Its id is "text".
2. The span of text element where we update the character count. Its id is "characters".

```
let text = document.getElementById("text") as HTMLInputElement;
let characters = document.getElementById("characters") as HTMLElement;
```

Type Assertions

- As a programmer you often *know* more about the program you're writing than the programming language does
- For example, you *know* there's an element with a specific **id**
- Additionally, you will usually *know* the specific **type** of **`HTMLElement`** you're working with. For example, here the element with id "tweets" is the subtype **`HTMLDivElement`**

```html
<div id="tweets"></div>
```

- We can also assert this knowledge using a type assertion:

```typescript
let tweets = document.getElementById("tweets") as HTMLDivElement;
```

- Why go to this effort? Some types (like text input fields) have properties (the value the user has typed in) defined only on the type that we need access to!

Part I) Updating Character Count (2 / 3)

- Next we'll write the event handler to change the "innerText" of the characters span to be the length of the string in the input box

It will look like other keyboard event handlers:

```
let updateLength = (event: KeyboardEvent): void => {
   characters.textContent = text.value.length + " chars";
};
```

- Notice that the string data the user has typed in to the form field is accessible to us via the **`text`** element's **`value`** property!

Part I) Updating Character Count (3 / 3)

- Finally, in the main function, we'll assign to the text input's onkeyup event handler a reference to the **updateLength** function we just wrote

```
export let main = async () => {

  document.body.style.background = "lightgray";

  // Bind Event Handlers
  text.onkeyup = updateLength;

};
```

Part II) Posting Tweets (1 / 3)

- When we submit the form (by either pressing the submit button or by just pressing enter while still in the text field) by default the whole page currently refreshes.

- How can we **prevent** this **default** behavior?

- We need to write a function to handle the form's "**submit event**"

- First, let's establish references to the form and the division of the page we want our tweets to post to.

```
let form = document.getElementById("form") as HTMLFormElement;
let tweets = document.getElementById("tweets") as HTMLDivElement;
```

Part II) Posting Tweets (2 / 3)

- Then, we need to write a generic event handler that will get called when our form is submitted.

```
let postTweet = (event: Event): void => {
   event.preventDefault();

   let tweet = document.createElement("p");
   tweet.textContent = text.value;
   tweets.appendChild(tweet);
};
```

- The first line is preventing the default behavior of the form from occurring (i.e. stopping the page from refreshing)
- Then, we're using the **`createElement`** method to create a new object in the DOM of type **p**aragraph. It's "**`innerText`**" property is its contents.
- Finally, we're appending the element to our tweets container.

Part II) Posting Tweets (3 / 3)

- Finally, we need to tell our form element to call the postTweet function when a submit event occurs.

```
export let main = async () => {
  body.style.background = "lightgray";

  // Bind Event Handlers
  text.onkeyup = updateLength;
  form.onsubmit = postTweet;
};
```

Part 3) Clearing Tweets (1 / 2)

- After a tweet has posted, we should clear the text input box and reset the character counter
- First, let's clear the text field by assigning an empty string to its value property:

```
text.value = "";
```

Optional Parameters (1/3)

- You can make parameters to **functions**, **methods**, and **constructors** optional by placing a **?**-mark character after their names

- For example:

```
constructor(r: number, cx?: number, cy?: number) {
   ...
}
```

- Any optional parameters must be declared *after* required parameters.

Optional Parameters (2/3)

- Specifying a parameter like:
  ```
  x?: number
  ```
- Changes x's type inside of the function/method/constructor to be *either* the type specified *or* undefined:
  ```
  number | undefined
  ```
- Therefore, in order to make use of an optional parameter, you must conditionally check to be sure it is not undefined:
  ```
  if (x !== undefined) {
    // Do something useful with x
  }
  ```

Optional Parameters (3/3)

- Optional parameters are very often used with constructors
  - This allows classes to have optional properties...
  - ...*and* makes it easy to override them when a new object is constructed.
  - You've already used this! Circle's constructor has optional center x and y.

Part 3) Clearing Tweets (2 / 2)

- Then we should call the **updateLength** function... but wait! It has a required **KeyboardEvent** parameter.

- Let's make that an optional parameter...

```
function updateLength(event?: KeyboardEvent): void {
```

- Now we can call **updateLength** after resetting the text field's value:

```
text.value = "";
updateLength();
```

Creating Interactive Web Pages

- There are a lot of little details to sort through when you transition to writing interactive web pages. It can feel overwhelming.
  - HTML Elements (and there are so many of them!)
  - Attributes
  - Event Handlers
  - Styling
- The good news is the web has *tons* of tutorials and reference material
- Best way forward is to first imagine what you want the web page to have on it, then how you want it to be able to respond to user input, and start breaking the problem down into smaller components. Pick one component to begin with and start tinkering, searching, and iterating to bring it alive.