

# Week 4: Forms, Media, & HTML 5 APIs

**INFSCI 2560**  
**Web Technologies & Standards**

# Reminder

## Assignment 1

- Due tonight at 11:59p
  - Late Policy: 10% per day, up to 3 days.
- Don't forget to complete the README!
  - Check your front-page with the W3C validators!
- *HAND-CODED*
  - No frameworks!

# Today's Topics

- HTML Forms
- Media Elements
- HTML 5 APIs

# HTML Forms

- The main mechanism for obtaining information from users
- A collection of widgets for user interaction
- Widgets can be assembled together to create *forms*
- You see forms all the time:
  - Every time you log into Pitt Passport
  - Online Shopping
  - Creating accounts on various social media sites
- Usually used for collecting user information in the browser and sending it to the server
  - But not necessarily

# A Simple Form

```
<form action="/server-endpoint" method="post">
  <div class="form name">
    <label for="name">Name:</label>
    <input type="text" id="name" name="user_name">
  </div>
  <div class="form email">
    <label for="mail">E-mail:</label>
    <input type="email" id="mail" name="user_mail">
  </div>
  <div class="form message">
    <label for="msg">Message:</label>
    <textarea id="msg" name="user_message"></textarea>
  </div>
</form>
```

Name:

E-mail:

Message:

# HTML Form Elements

<a href="#"><code>&lt;form&gt;</code></a>	Defines an HTML form for user input
<a href="#"><code>&lt;input&gt;</code></a>	Defines an input control
<a href="#"><code>&lt;textarea&gt;</code></a>	Defines a multiline input control (text area)
<a href="#"><code>&lt;label&gt;</code></a>	Defines a label for an <code>&lt;input&gt;</code> element
<a href="#"><code>&lt;fieldset&gt;</code></a>	Groups related elements in a form
<a href="#"><code>&lt;legend&gt;</code></a>	Defines a caption for a <code>&lt;fieldset&gt;</code> element
<a href="#"><code>&lt;select&gt;</code></a>	Defines a drop-down list
<a href="#"><code>&lt;optgroup&gt;</code> <code>&gt;</code></a>	Defines a group of related options in a drop-down list
<a href="#"><code>&lt;option&gt;</code></a>	Defines an option in a drop-down list
<a href="#"><code>&lt;button&gt;</code></a>	Defines a clickable button
<a href="#"><code>&lt;datalist&gt;</code></a>	Specifies a list of pre-defined options for input controls

- This is the collection of elements you can use to compose forms
- Each corresponds to a widget or behavior as implemented in the browser
- They don't always look the same in different browsers (Chrome, Safari, Firefox, Edge)
  - But they should behave the same
- For more detailed description check out the [W3 Schools HTML Form Elements](#) reference page

# The <form> Element

```
<form action="some_url"
      method="HTTP method">
    ...
    <Form elements>
    ...
</form>
```

- This is the basic structure of a form element
- The <form> element is a semantic container for HTML, like a <div> or <p>
- Has two important attributes that determine the information behavior of the form
  - **action** - Specifies a URL for where to send the data collected by the form
  - **method** - Specifies the HTTP Method (GET, POST) to use when sending form data

# The <fieldset> and <legend> Elements

```
<form>
  <fieldset>
    <legend>Personalia:</legend>
    Name: <input type="text"><br>
    Email: <input type="text"><br>
    Date of birth: <input type="text">
  </fieldset>
</form>
```

Personalia:

Name:	<input type="text"/>
Email:	<input type="text"/>
Date of birth:	<input type="text"/>

- Use the <fieldset> element to group form related elements in a form
  - Create a semantic collection
- Most browsers will draw a box around the sub-elements
  - You can use CSS to change the look
- Use the <legend> element to give a name to that collection of form elements
  - Will be displayed in the box



# The <button> Element

```
<button id="button1"  
type="button">Click  
Me!</button>
```

A rectangular button with a light gray background, a thin gray border, and a subtle gradient. The text "Click Me!" is centered on the button in a bold, black, sans-serif font.

- The button element creates clickable object in the page
- There is a browser determined default look
  - But this can be modified with CSS
- You can also put content *inside* the button
  - Can't do this with <input>
  - like Images, or other text elements

# Styling HTML buttons

```
.button {  
  background-color: #4CAF50;  
  border: none;  
  color: white;  
  padding: 15px 25px;  
  text-align: center;  
  font-size: 16px;  
  cursor: pointer;  
}  
.button:hover {  
  background-color: green;  
}
```

```
<button>Default Button</button>  
<button class="button">Styled  
Button</button>
```

Default Button

Styled Button

# The <label> Element

```
<form action="/action_page.php">
  <label for="steelers">Pittsburgh Steelers</label>
  <input type="radio" name="team"
    id="steelers" value="steelers"><br>
  <label for="ravens">Baltimore Ravens</label>
  <input type="radio" name="team"
    id="ravens" value="ravens"><br>
  <label for="other">Other</label>
  <input type="radio" name="team"
    id="other" value="other"><br><br>
  <input type="submit" value="Submit">
</form>
```

- Use the <label> element to associate a label with a form element
  - input, button, select, textarea
  - Good for screen readers
- The value of the **for** attribute of the <label> element should be equal to the value of the **id** attribute of the form element the <label> is associated with

Pittsburgh Steelers ☐

Baltimore Ravens ☐

Other ☐

Submit

# The <input> Element

```
<form action="/action_page.php">  
  First name:  
  <input type="text" name="FirstName"  
    value="Mickey"><br>  
  Last name:  
  <input type="text" name="LastName"  
    value="Mouse"><br>  
  <input type="submit" value="Submit">  
</form>
```

- **After the <form> element, the <input> element is the most important**
- Specifies a field where user can enter data
- Has several very important attributes that can affect behavior
  - **name**
  - **value**

First name:

Last name:

# Input Types - ~5 minute activity

Go to the MDN page that lists out the <input> form types.

<https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input>

- Visit <https://codepen.io/pen/>
- Create a basic form with 2-3 input types.
- Use one from the list to the right and another input type of your choice.

- Button
- Checkbox
- Date
- File
- Number
- Password
- Text

# Example input types

click

☐ Yes

Choose File No file chosen

0

....

initial

Week --, ----

- Button
- Checkbox
- Color
- File
- Number
- Password
- Text
- week

# Attributes common to all input types

- Name
- Value
- Type
- Autocomplete
- Disabled
- List
- Readonly
- Required
- Tabindex
- Autofocus

# The <textarea> Element

```
<textarea rows="5" cols="40"  
          maxlength="40">
```

Initial text in the <textarea> field  
will not be cut off at the maxlength.  
But the user won't be able to enter  
new text until this is deleted.  
</textarea>

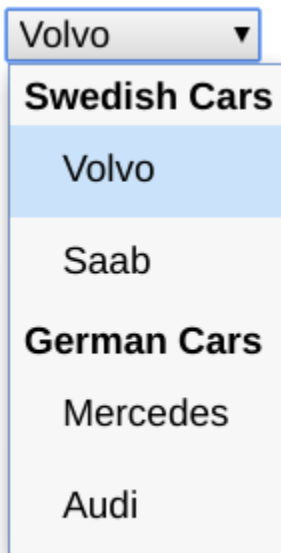
Initial text in the <textarea> field  
will not be cut off at the maxlength.  
But the user won't be able to enter new  
text until this is deleted.

- The <textarea> form element lets you define a multi-line text input area
- It can hold an "unlimited" amount of text
- Style in a monospace font (usually Courier, the typewriter font)
- Has several useful elements
  - **cols** - Specify the width
  - **rows** - Specify the number of lines
  - **maxlength** - Specify the max number of characters
  - and many others



# Drop Down Menus

```
<select multiple size=7>
  <optgroup label="Swedish">
    <option
value="volvo">Volvo</option>
    <option
value="saab">Saab</option>
  </optgroup>
  <optgroup label="German">
    <option
value="mercedes">Mercedes</option>
  >
    <option
value="audi">Audi</option>
  </optgroup>
</select>
```



- The `<select>` element can be used to create a menu of options that the user must choose.
- Specify the possible options by filling the `<select>` element with `<option>` elements
  - The text contents are for display
  - The **value** attribute, this is what gets sent
- You can group options together using the `<optgroup>` element
- You can also use the **multiple** and **size** attributes to do a multiple select

# Look at the Structure

```
<form action="/server-endpoint" method="post">
  <div class="form name">
    <label for="name">Name:</label>
    <input type="text" id="name" name="user_name">
  </div>
  <div class="form email">
    <label for="mail">E-mail:</label>
    <input type="email" id="mail" name="user_mail">
  </div>
  <div class="form message">
    <label for="msg">Message:</label>
    <textarea id="msg"
name="user_message"></textarea>
  </div>
</form>
```

- Everything is contained within the `<form>` element
- Inside this form are two kinds of form elements
  - `<label>` - For defining a text label for the form widget
  - `<input>` - For creating an input box for users to type information
- The `<div>` elements are there to structure the form, break up the different widgets, and provide something for CSS styling

# Structuring Forms

```
<form>
  <fieldset>
    <legend>Fruit juice size</legend>
    <p>
      <input type="radio" name="size" id="size_1"
        value="small">
      <label for="size_1">Small</label>
    </p>
    <p>
      <input type="radio" name="size" id="size_2"
        value="medium">
      <label for="size_2">Medium</label>
    </p>
    <p>
      <input type="radio" name="size" id="size_3"
        value="large">
      <label for="size_3">Large</label>
    </p>
  </fieldset>
</form>
```

- There is no *one* right way to structure a form, depends on the data you want to collect
  - Use <fieldset> and <legends> to make semantically meaningful structures in your form
- Forms cannot be nested inside of other forms
- Use the **form** attribute on HTML form elements to bind the elements to the form, even when they aren't enclosed in a <form> element
  - So form elements can exist outside an encapsulating <form> element

[https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/How\\_to\\_structure\\_an\\_HTML\\_form](https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/How_to_structure_an_HTML_form)

# Sending form data

- When a user hits the "submit" button or input the browser will bundle all of the form elements into a data structure and send it to the URL specified by the **action** attribute
  - Send to the current page URL if no specified
- This will cause the page to reload, either going to the new URL or reloading the current page
- Sometimes you want to deal with the form data on the client side using javascript
- To do this you need to "capture" the implicit submit
- We will cover how to deal with form data on the server when we talk about server-side programming

# GET method

```
<form action="http://foo.com" method="get">
  <div>
    <label for="say">What greeting do you want to
say?</label>
    <input name="say" id="say" value="Hi">
  </div>
  <div>
    <label for="to">Who do you want to say it
to?</label>
    <input name="to" id="to" value="Mom">
  </div>
  <div>
    <button>Send my greetings</button>
  </div>
</form>
```

- **GET** - method will send the data inside the URL as HTTP parameters
- HTTP will look like:

```
GET /?say=Hi&to=Mom HTTP/2.0
```

```
Host: foo.com
```

- Notice how the name attribute of `<input>` relates to the data

# POST Method

```
<form action="http://foo.com" method="post">
  <div>
    <label for="say">What greeting do you want
to say?</label>
    <input name="say" id="say" value="Hi">
  </div>
  <div>
    <label for="to">Who do you want to say it
to?</label>
    <input name="to" id="to" value="Mom">
  </div>
  <div>
    <button>Send my greetings</button>
  </div>
</form>
```

- POST - method will send the data as key/value pairs in the body of the HTTP request

POST / HTTP/2.0

Host: foo.com

Content-Type: application/x-www-form-urlencoded

Content-Length: 13

say=Hi&to=Mom

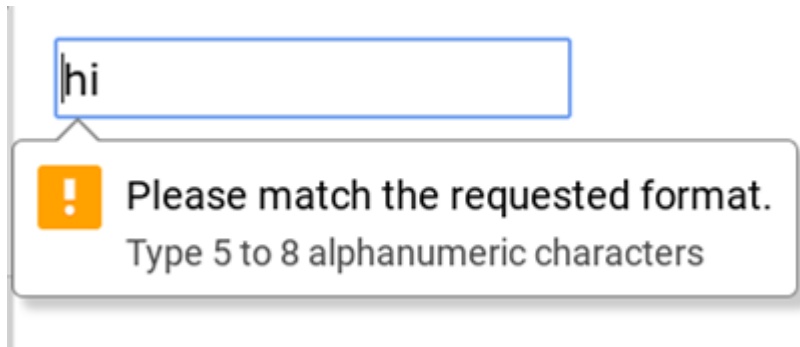
- Notice how the name attribute of `<input>` relates to the data

# Catching the Implicit Submit

```
window.addEventListener("load",function() {  
    // add a "submit" listener to a form element with id "my-form"  
    document.getElementById('my-form').addEventListener("submit",function(e) {  
        // this code will prevent the form submission  
        e.preventDefault(); // before the code  
  
        /* do what you want with the form here*/  
  
        // Should be triggered on form submit  
        console.log('Form Submitted');  
    });  
});
```

- By default all forms will send data to the server and reload the page
- You can "catch" this default behavior and redirect it using Javascript event listeners
- Use this code in Activity 4

# Form Validation



- When filling out forms, you often want to restrict what kinds of data the user can enter
  - required fields
  - phone numbers
  - password length and complexity
- Form validation allows you to apply constraints to user entered data
- Validation can happen server side or client side
  - client side can used HTML5 or Javascript
  - We will talk about server side validation later

[https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Form\\_validation](https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Form_validation)



# HTML 5 Form Validation

```
<!--
```

Use the pattern attribute to specify

```
"zero or more(*) alphanumeric  
characters([a-zA-Z0-9])" and the minlength  
and maxlength attributes to specify length  
-->
```

```
<input type="text" pattern="[a-zA-Z0-9]*"  
  minlength="5" maxlength="8" required  
  title="Type 5 to 8 alphanumeric characters">
```

[https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Form\\_validation](https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Form_validation)

- You can use **validation attributes** on form elements to specify rule for inputs
- Some specific elements have built-in validators
  - `type=email`, `type=url`
- Also assigns the `:value` and `:invalid` CSS pseudo-classes so you can define your own styles to elements
- Attributes:
  - **required** - Indicates the form field is required
  - **minlength** and **maxlength** - minimum and maximum character length for the field
  - **pattern**

# Media Elements

# HTML Multimedia

- `<video>` element allows you to embed a video.

```
<video src="myvideo.mp4" controls> <p>Your browser doesn't support HTML5 video. Here is a  
<a href="rabbit320.webm">link to the video</a> instead.</p> </video>
```

- `<iframe>` element allows you to embed other web documents into your web page.
  - iFrames are a common target for hackers.
  - Read more on [Security Concerns](#)
- `<svg>` element allows you to create shapes using XML.

[https://developer.mozilla.org/en-US/docs/Learn/HTML/Multimedia\\_and\\_embedding/Video\\_and\\_audio\\_content](https://developer.mozilla.org/en-US/docs/Learn/HTML/Multimedia_and_embedding/Video_and_audio_content)

# Audio

```
<audio controls>
```

```
  <source src="horse.ogg" type="audio/ogg">
```

```
  <source src="horse.mp3" type="audio/mpeg">
```

Your browser does not support the audio element.

```
</audio>
```

- You can add audio to web pages easily with the <audio> element
  - No more flash!
- The main challenge with HTML audio is the video formats
  - MP3
  - WAV
  - Ogg
- Chrome and Firefox support all three, Safari support MP3 and WAV, Internet Explorer only supports MP3
  - Ogg is a Free standard, but not very popular
- For more info check out the [W3 Schools guide on HTML audio](#)

# Video

```
<video width="320" height="240" autoplay>
```

```
  <source src="movie.mp4" type="video/mp4">
```

```
  <source src="movie.ogg" type="video/ogg">
```

Your browser does not support the video tag.

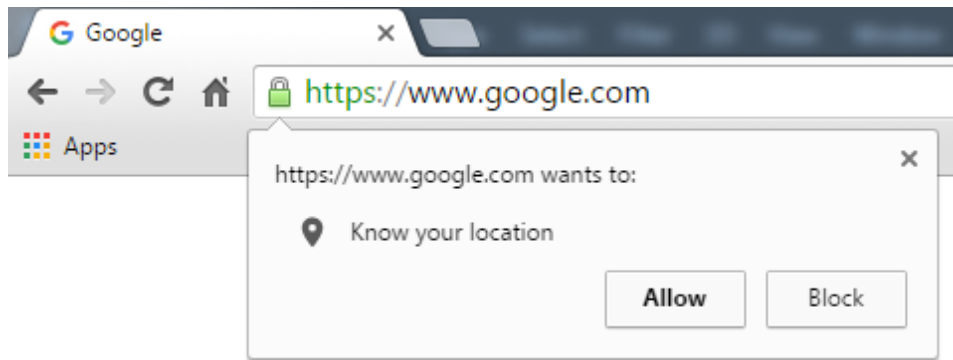
```
</video>
```

- You can add movies to web pages easily with the <video> element
  - No more flash!
- The main challenge with HTML video is the video format
  - MP4
  - WebM
  - Ogg
- Chrome and Firefox support all three, Safari and Internet Explorer only support MP4
  - MP4 is a proprietary format so Mozilla doesn't like it
- For more information check out the [W3 Schools page on HTML Video](#)

**BREAK**

# HTML 5 APIs

# HTML Geolocation



- HTML 5 introduces a Geolocation API that get the latitude and longitude of the device's current position.
- User must give permission to a page in order to get results
  - PRIVACY WARNING!
- Methods
  - `Geolocation.getCurrentPosition()`
  - `Geolocation.watchPosition()`
  - `Geolocation.clearWatch()`

<https://developer.mozilla.org/en-US/docs/Web/API/Geolocation>



# Geolocation Example

```
if (!navigator.geolocation) {  
    status.textContent = 'Geolocation is not supported by your browser';  
} else {  
    status.textContent = 'Locating...';  
    navigator.geolocation.getCurrentPosition(success, error);  
}
```

```
▼ Position {coords: Coordinates,  
  ▼ coords: Coordinates  
    accuracy: 79  
    altitude: null  
    altitudeAccuracy: null  
    heading: null  
    latitude: 40.4199126  
    longitude: -79.9400829  
    speed: null  
    ► __proto__: Coordinates  
    timestamp: 1548889971516
```

[CodePen Example](#)

# Local Storage



- HTML local storage provides two objects for storing simple, unstructured data on the client
  - *window.localStorage* - stores data with no expiration date
  - *window.sessionStorage* - stores data for one browser session. For *structured* data, there is the indexedDB API
- Methods
  - `setItem()`
  - `getItem()`
  - `removeItem()`
  - `clear()`

<https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage>

<https://developer.mozilla.org/en-US/docs/Web/API/Window/sessionStorage>

# LocalStorage Example

```
// Store
localStorage.setItem("username", "tedmonds")
localStorage.setItem("first name", "tonya")
localStorage.lastName = "Edmonds"
// Retrieve
var username =
localStorage.getItem("username")
> tedmonds
localStorage.lastName
> Edmonds
// Remove
localStorage.removeItem("username")
```

- The localStorage object preserves data "forever"
- <Key, Value> pairs
- Data persists even when the browser window is closed
- Everything is stored as strings
  - So you will need to convert numeric datatypes
- Both localStorage and sessionStorage **can** use up to 10MB of data, combined
- Synchronous
- **Not secure**

# IndexedDb

- For *structured* data, there is the indexedDB API
  - Full, transactional database system
  - Uses Javascript objects
  - Stored as key-value pairs
  - Asynchronous so it won't block pages

## LocalStorage vs IndexedDb

- LocalStorage was designed for smaller amounts of data
- IndexedDb provides indexes which allow you to run queries on your data
- LocalStorage is a lightweight solution
- IndexedDb is much more powerful and has a complex API
- IndexedDb requires knowledge of relational database concepts

[https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB\\_API](https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API)

# Activity 4

1. Go to <https://glitch.com/~infsci2560-fall2022-activity4> and REMIX IT
2. Complete the 5 tasks
  - a. Look at [README.md](#) and the HTML Source for instructions
3. Submit the Glitch *project* URL
  - a. *Project URL* - <https://glitch.com/~infsci2560-fall2022-activity4> << THIS
  - b. *Live URL* - <https://infsci2560-fall2022-activity4.glitch.me/>