Week 5: Forms, Media, & HTML 5 APIs

INFSCI 2560
Web Technologies & Standards

Good Evening!

Go to the course website

View Assignment 1

Review the Rubric and Cover Sheet

Do you have any questions?

Other Housekeeping Items

- Assignment 1 due Monday, September 30 by 11:59p
  - Late Policy: 10% per day, up to 3 days.
  - Don't forget to complete the cover sheet
- Activity due date now extended to Wednesdays at 11:59p.
- Exam 1 - October 14, 2019
  - The goal is to assess your knowledge and application of the topics/concepts covered in class.
  - Format (1 hour)
    - 30-40 Questions (Multiple Choice, Short Answer, Write code snippets)
  - How to study
    - Write code snippets
    - Be familiar with content covered during lectures and activities
    - Review activities and assignments

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>
```

HTML Form Elements

<form>	Defines an HTML form for user input
<input>	Defines an input control
<textarea>	Defines a multiline input control (text area)
<label>	Defines a label for an <input> element
<fieldset>	Groups related elements in a form
<legend>	Defines a caption for a <fieldset> element
<select>	Defines a drop-down list
<optgroup>	Defines a group of related options in a drop-down list
<option>	Defines an option in a drop-down list
<button>	Defines a clickable button
<datalist>	Specifies a list of pre-defined options for input controls
<output>	Defines the result of a calculation

- 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

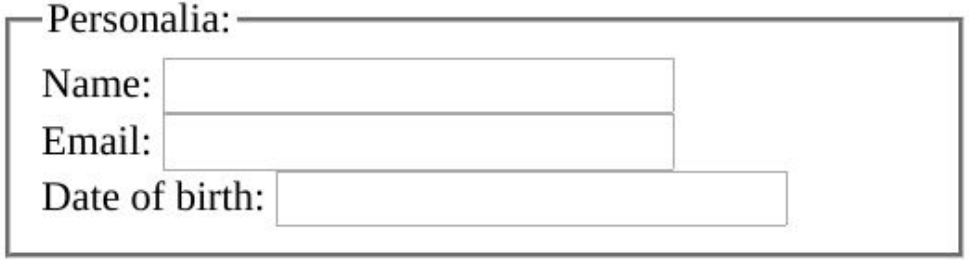
The <form> Element

```
<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 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>
```



- 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>
```

- 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

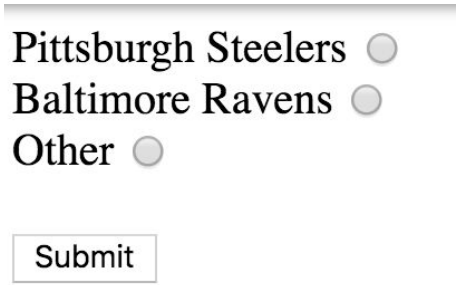
```
.button {
  background-color: #4CAF50;
  border: none;
  color: white;
  padding: 15px 25px;
  text-align: center;
  font-size: 16px;
  cursor: pointer;
}
.button:hover {
  background-color: green;
}
```

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

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>
```

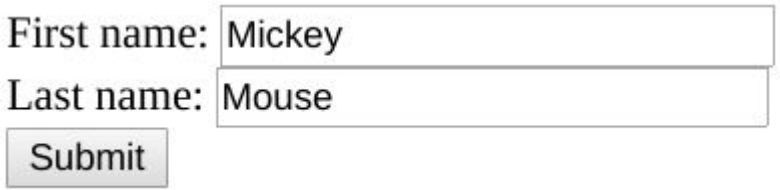
- Use the <label> element to associate a label with a form element
  - input, button, select, textarea
  - Good for screen readers
- The value of the **for** attribute of the <label> element should be equal to the value of the **id** attribute of the form element the <label> is associated with



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**



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

Input Types - ~5 minute activity

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

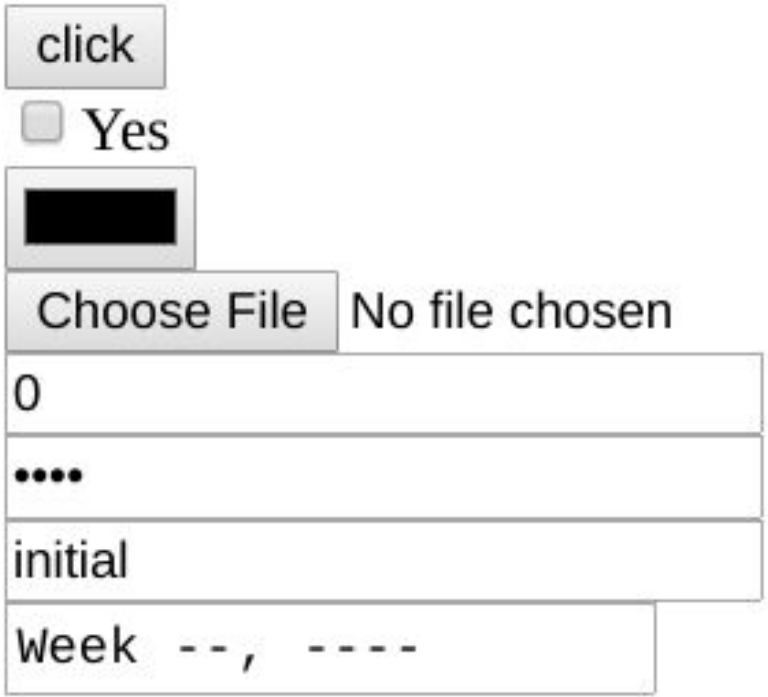
https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input

Using a plain text editor create a basic form with 2-3 input types. Use one from the list to the right and another input type of your choice.

If you have time, open the page in two browsers and see if there are any differences in how the browser displays the input type.

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

Example input types



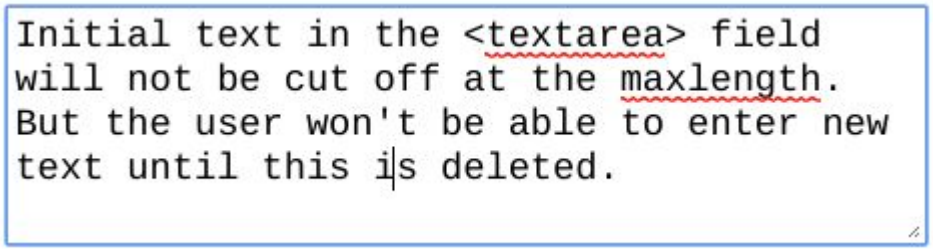
- Button
- Checkbox
- Color
- File
- Number
- Password
- Text
- week

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>
```



- 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 se
- 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 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

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

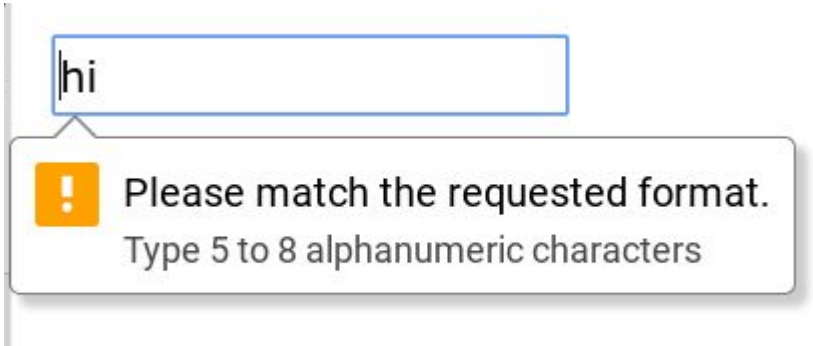
```
window.addEventListener("load",function() {
     // add a "submit" listener to a form element with id "my-form"
     document.getElementById('my-form').addEventListener("submit",function(e) {
          // this code will prevent the form submission
          e.preventDefault(); // before the code

           /* 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

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

- 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

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

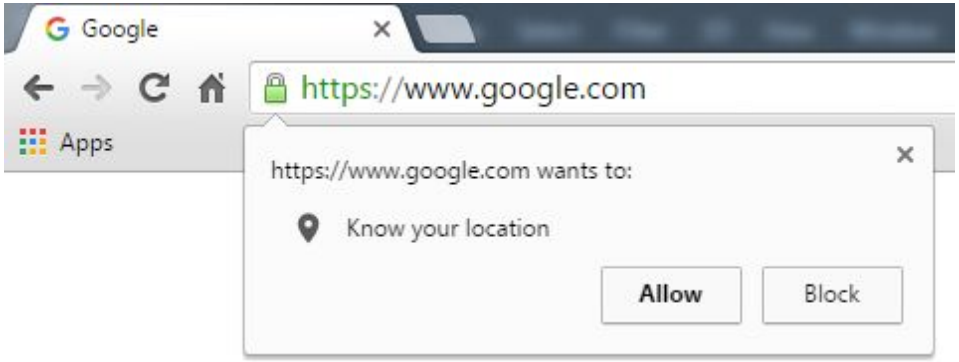
```
<video width="320" height="240" autoplay>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
Your browser does not support the video tag.
</video>
```

- 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

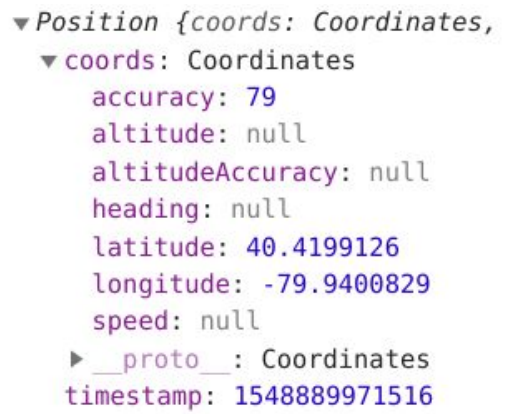


- HTML 5 introduces a Geolocation API that get the latitude and longitude of the device's current position.
- User must give permission to a page in order to get results
  - PRIVACY WARNING!
- Methods
  - Geolocation.getCurrentPosition()
  - Geolocation.watchPosition()
  - Geolocation.clearWatch()

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);
  }
```



[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

Activity 4

https://glitch.com/~2560-activity4

Due Wednesday, September 26 by 11:59pm