

# What is a form?

- User can enter data (text, checkbox, select)
- Data is sent to server, server returns new page

More advanced version:

- JS submits data
- Gets a response without a new page

We will not do the more advanced version (6250 topic)

- Implementation is less of a UI/UX concern
- Form HTML/CSS is same either way

# **Two ways forms are used**

A Form can be used to search/read data

- Ex: Entering a Google search
- Ex: Searching for a product on Amazon
- Ex: Looking up a user on social media

A Form can be used to create/edit data

- Ex: Making a social media post
- Ex: Adding a Like/Reaction to a social media post
- Ex: Adding a question to a forum

# Form Element

Foundation of a form:

```
<form action="/some/url/" method="POST">  
  <button type="submit">Submit</button>  
</form>
```

- Sends any data from form to `/some/url`
  - Fully Qualified/absolute/relative url
- Similar to a link (navigates)
- Form itself has no inherent UI (block)

# Form Method

```
<form action="/some/url/" method="POST">
```

- **method** decided by server
  - **GET** to search/read
  - **POST** to change/edit
- **GET** sends data in url **query parameters**
  - `<a href="">` link is always a GET request
- **POST** sends data in **body** of request
  - Not visible in URL
  - Not secured from user!
- Both default to sending data as **url encoded**

# URL Encoding

- Converts text to not break a url
- Forms CAN use other encodings
  - Particularly for file uploads
  - Url encoding appropriate for simple text data
- Easy test for URL encoding:
  - Have a form with GET action
    - Look at url afterward
  - Check Network request Payload in DevTools

# How to URL Encode

- Forms do this automatically
- Manual process:
  - Form fields are `name=value` pairs (no spaces)
  - Multiple name=value pairs separated by `&`
  - Special chars become `%` then 2 digit hex ASCII
    - `+` is `%2b`
    - space becomes `%20` or `+`
    - `?` is `%3f`
    - `&` is `%26`
    - `%` is `%25`
    - `#` is `%23`

# Input element

```
<input name="demo1" type="text"/>  
<input name="demo2" type="checkbox"/>  
<input name="demo3" value="cat" type="radio"/>  
<input name="demo3" value="cats" type="radio"/>
```

 ☐ ☐ ☐

When data is sent, it is sent as key/value pairs

```
demo1=whatwastyped  
demo2=on  
demo3=cats
```

url-encoded:

```
demo1=whatwastyped&demo2=on&demo3=cats
```

# Input text field

```
<input name="demo1" type="text"/>
```

## Notable attributes

- `type` (text is default)
- `name`
- `value`
- `placeholder`
- `disabled`
- `readonly`
- (validation covered later)
- (a11y covered later)



# Other text-like inputs

Change `type` for related text-like inputs

- `password` (hides characters from view)
- `hidden` (hides field, passes value)

Recent(ish) additions:

- `color` (graphical input, textual value)
- `date` (text or cal input, textual value)
- `email`
- `number`
- `search`
- `tel`
- `time`
- `url`

# Checkbox

- Sends value (default "on") when checked
- Doesn't send field on submit at all if not checked
- `checked` attribute to pre-select

```
<input type="checkbox" name="it-is"/>  
<input type="checkbox" name="already" checked/>
```



# Radio buttons

- Only one of same name can be selected
- Name didn't age well
- Uses `checked` as well
- No unselecting, only changing selection

```
<input type="radio" name="favorite" value="maru">  
<input type="radio" name="favorite" value="nyan">  
<input type="radio" name="favorite" value="grumpy">  
<input type="radio" name="meh" value="labrador" checked>  
<input type="radio" name="meh" value="poodle">  
<input type="radio" name="meh" value="retriever">
```



# Select dropdowns

- `<option>` tags inside a `<select>` element
- `name` of `<select>`, `value` of `<option>`
- `selected` attribute on `<option>`
  - Or first one (always a selection)
- `value` is sent, not content
  - unless no value (always have a value)

```
<select name="cats">
  <option value="rule">Rule the World</option>
  <option value="awesome">Are Awesome</option>
  <option value="inspire">Inspire Me</option>
</select>
```

Rule the World ▼

# Textarea element

- NOT an `<input>`
  - mostly same attributes
- Content is value, not `value` attribute
- Multiline input
- Default resizable (CSS `resize` can change)
- Is a natural `inline-block`
- `wrap` attribute (either "soft" or "hard") sets if newlines in value where it wrapped

```
<textarea name="blahblah"></textarea>
```

# Label element

Forms should have text labels describing them

- Don't use `placeholder` for this

`<label>` element for that text

Two ways to use `<label>`

- With a `for` element w/value of input `id` attribute
- As an ancestor of the input/textarea/radio group
  - No `for` needed (and no `id` needed)

Not only text, but selecting label selects the field

- Great for accessibility, esp on mobile!

## More on label

- You should always have `<label>` elements
  - Important for a11y and usability
- If `<label>` separate from field element
  - MUST have `for` attribute on label
  - MUST have matching `id` on field element
- If field element inside `<label>`
  - No need for `for`/`id`
  - PROBABLY want a `<span>` around the text
  - MUST have exactly one field input per label
- Choice heavily impacts how to style!

# Label HTML Structure

```
<form action="..." method="...">  
  <label for="name">Name:</label>  
  <input id="name" name="name">  
  <button type="submit">Submit</button>  
</form>
```

```
<form action="..." method="...">  
  <label>  
    <span>Name:</span>  
    <input name="name">  
  </label>  
  <button type="submit">Submit</button>  
</form>
```



# Checkboxes are visually complicated

- Labels may be to right OR left of input
  - (in English ltr flow)
- Checkboxes and Radio buttons are...different

Do you change your HTML to match?

- Or fix with only CSS?
- No good answer yet

# Fieldset and legend

- A `<fieldset>` element groups 1+ labels and fields
- A child `<legend>` element labels the fieldset

Styling these in different browsers can be challenging

- `flexbox` and `grid` haven't always worked on these
- Assorted a11y labelling problems
- Investigate before committing to it

# What to consider with a form

## Communication

- What am I filling in?
- What is required?
- What is the expected value?

## Validation

- What did I do wrong?
- What do I fix it?
- Where is it?

# **What to consider with form a11y**

## Accessibility

- Are fields identifiable?
- Do visuals translate?
- Are controls usable?
- Is positioning confusing?

# Form Communication

What am I filling in?

- Do I even notice the fields?

What is required?

- The more we ask, the more often they give up
- \* convention (backed with hint)

What is the expected value?

- Syntax?
- Data type in general

# Form Validation

Ensuring data is acceptable

- May be done before data is sent
  - HTML-based validation
  - JS-based validation
- May be done after data is sent
  - Server returns a form requesting changes

Make sure they know what to fix and how to fix!

- Per field hints is best
- Often a top-level indication that fixes are needed

# Bad Validation is the worst UX!

- Can leave user unable to proceed
- Names are just one example:
  - <https://shinesolutions.com/2018/01/08/falsehoods-programmers-believe-about-names-with-examples/>
- Consider "O'Brian"
  - Very common last name
  - Very commonly stopped by forms

Validation should **help** the user

# Form Accessibility

Forms are often most important part for usability

- Often the worst accessibility

Great visuals may not translate well!

- Don't fall in love with effect until you are sure

Screen readers read text

- Do they know what to read?
- Does it have the necessary context?

More on a11y later



# Styling Forms

- Surprisingly complex
- Makes sense:
  - Both Input and Output
  - More active communication
- Whitespace very important!
- Indentation(`margin`), `line-height`

# Columns

- Need to associate text labels with input
- 2 major approaches:
  - Two column (horizontal)
  - One column (vertical)
- Here "column" = place for label OR field
  - **NOT Grid Columns!**
  - Confusing, but we are talking about if labels/fields are in same column or not

# Form Layout: 2 Column

- Labels on one side (column)
- Fields on other side (column)

Pros/Cons:

- Arguably better for longer forms
  - But long forms themselves are poor UX
- Disliked in the current designer meta

Important details:

- Avoid big gaps between label and field
- Manage Checkboxes (exceptions?)

# Form Layout: 1 Column

Labels above (or below - ick) fields

Pros/Cons:

- Some tests show users are faster to fill out
  - Better "conversion rates"
    - Fewer leave before finishing
  - Field closer to label text
- Checkboxes still a pain

# How to create "columns"?

- DO NOT USE `float`
  - Lots of bad examples out there
- Otherwise, many answers
  - `inline-block` with `width`
  - `flex`
  - `grid` with various `grid-column: span X`
- Checkboxes and Radio can change things!
- `<label>` containing or sibling of `<input>`?

Learn how to layout anything

- Not one way to do it

# Other Form variations

- Multi-step form
  - Break form (in any layout) into multiple forms
  - Can "breadcrumb" or "step" navigation
  - Lulls the user (or lulz the user, ha!)
- Accordion or Folding form
  - Form in one page, but broken among collapsible sections.

# HTML Form Validation

Making sure the data entered is correct

- Server side
  - MUST HAPPEN!
  - Can be slow
  - Backend devs are less UI oriented
- Client side
  - For convenience, not security
  - Faster
  - Can be HTML-based or JS-based

# HTML-based Form validation

- Enforced by browser
  - Some may dislike default behavior
  - limited intelligence/combinations/UI
- CSS can alter/extend behavior

Example: `required` attribute

- What if you have multiple required fields?



# regex pattern validation

`pattern` attribute validates field content

- Using a "Regular Expression" (Regex)
- A lot of hate in the world against Regex
  - A powerful syntax to describe text patterns
- Many like various "regex tool" websites to help
  - I prefer to learn the actual rules
- HTML validation errors do not give details
  - Use `title` attribute to help
  - Additional a11y requirements later

See </readings/js/regex.md> for more on Regex

# CSS for validation

`:required` pseudo-selector matches required inputs

- e.g. `input:required`

`:invalid` pseudo-selector matches invalid inputs

- e.g. `input:invalid`
- Even if you've never had a chance to enter them
  - This limits the usefulness without JS

JS can read the state of fields

- Can add/remove classes to offer detailed styling
- More later