

# Many Ways to Do Form Validation with JS

- I won't teach just 1 "best" way
- You often have to implement according to design
  - Designers will have different ideas and goals
- I will show how to do different options
  - And offer pros/cons
- A set of options for you + designer to pick from

# Working with Forms with JS

- JS offers a lot of options base HTML/CSS does not
- Let's create an example form

```
<form class="register" action="/register" method="POST">
  <label class="register__label">
    <span class="register__name-label">Name</span>
    <input name="name" class="register__name">
  </label>
  <label class="register__label">
    <span class="register__email-label">Email</span>
    <input name="email" type="email" class="register__email">
  </label>
  <label class="register__label">
    <span class="register__confirm-label">Confirm Email</span>
    <input name="confirm" type="email" class="register__confirm">
  </label>
  <button type="submit" class="register__submit">Register</button>
</form>
```

# Events

- Validating Forms with JS
- Have code that reacts to **events**

```
const nameEl = document.querySelector('.register__name');  
nameEl.addEventListener('input', () => { //fat arrow function  
  console.log("typing happened");  
});
```

# Value

- Event object is passed to callback
- Event object `.target` is DOM Node of field
- `.value` property is current value of the field

```
const nameEl = document.querySelector('.register__name');

nameEl.addEventListener('input', (event) => {
  // Can decide if value is okay as they type
  console.log(event.target.value);
});
```

# Form Events

- There are MANY events
- Here are some common ones for forms
  - input
  - submit
  - focus
  - blur
  - change
  - keydown

# input event

- Typing on a text/textarea
- Changed selection on `<select>`
- Inconsistent on checkbox/radio
  - Use change event instead

```
const nameEl = document.querySelector('.register__name');

nameEl.addEventListener('input', (event) => {
  // Can decide if value is okay as they type
  console.log(event.target.value);
});
```

# submit event

- Fires on `<form>` when submitted
  - On `<form>` element, not on button!
  - Even though submit likely from a button
- `event.preventDefault()` stops submit

```
const formEl = document.querySelector('.register');
formEl.addEventListener('submit', (event) => {
  // event.target is the form, not the fields
  const isFormInvalid = true; // Put code to decide here
  if( isFormInvalid ) {
    // Put code to tell user what to fix here
    event.preventDefault();
  }
});
```

# invalid event

- Like `submit`, triggers on submit
  - When HTML validation not passed
- No `submit` in such case



# focus and blur events

- Fires when element gains/loses focus
- Does NOT propagate/"bubble"
  - This can complicate things later! (only a little)
- Used to validate a field after user LEAVES the field
  - Good UX because only complains after done
  - Poor UX because fixes require they go back
- Can get `blur` AND `submit` if they click submit

# change event

- Fires when a value changes
  - like `blur` on text
  - on selection for select/radio/checkbox

# keydown event

- Fires on keypress
- BEFORE key is added to field
- Fires even if key is modifier (Shift, Ctrl, etc)
- `event.preventDefault()` - key is not added to field
- Event object has info about the key pressed
  - `.key` - which key is pressed
  - `.shiftKey`, `.altKey`, `.ctrlKey`, `.metaKey`
  - `.isComposing` - translation inputs (Ex: Pinyin)
  - Event object, not `event.target`

# keydown example

```
// prevent "-" from being entered

inputEl.addEventListener('keydown', (event) => {
  if( event.key === "-" ) {
    event.preventDefault();
  }
});
```

- Cut and Paste/autofill can bypass
- Do not assume too much
  - Users enter data in many ways

# **How to inform user of problems?**

- Prevent submission
- Indicators
- Messages

# Preventing Submission

- Telling user and stopping submissions
  - Two different requirements
- Stop submission on `submit` event
  - Disabling button may not stop submission!
  - Enter on form field can submit!

# Visual Indicators Cannot be JUST color

- Ex: Put red border around invalid fields
  - Requires they see and distinguish red
  - Not good for color-blind or vision-impaired
- Indicators + Messaging better
- For indicator styling
  - Place class on field(s) and/or on form
  - Have CSS that selects for field

```
.invalid { /* class on field */  
  border: 1px solid red;  
}
```

# Messaging

- Text informing user of problem
- Can be at top of form
- Can change text of/change submit button
- Can be on each field
- UX is finding the way best for user
  - Not the easy way for developer



# Changing Text using JS (innerText)

- Pros:
  - Text updates dynamically
- Cons:
  - Error text lives in JS
  - Only plain text can change (not HTML)

# Using `innerText` dynamically

```
<div class="demo"></div>  
<button class="button__add">Add</button>
```

```
let count = 0;  
  
const buttonEl = document.querySelector('.button__add');  
const demoEl = document.querySelector('.demo');  
buttonEl.addEventListener('click', () => {  
  count += 1;  
  demoEl.innerText = count;  
});
```

# .innerText

- Change the text content of a DOM Node

```
<div class="demo"></div>
```

```
const demoEl = document.querySelector('.demo');  
demoEl.innerText = "Hello World";
```

- Set to empty string to remove

```
demoEl.innerText = "";
```

# Styling elements that use innerText

- Often you want errors to have styling
  - borders, padding, etc
- Don't want this visible when text is empty

```
.demo {  
  padding: 1rem;  
  background-color: #FF000033; /* red w/transparency */  
}  
  
.demo:empty { /* Only applies when element is empty */  
  display: none;  
}
```

# .innerHTML allows more than text

```
const demoEl = document.querySelector('.demo');  
demoEl.innerHTML = `<p>This is <b>Awesome!</b></p>`;
```

- As with `innerText`, set to "" to remove
- An element with child elements is not `:empty`
  - Even if those elements have no text

# Changing HTML using JS (innerHTML)

- Pros:
  - Allows more complex messaging
- Cons:
  - Puts HTML in JS
    - Harder to edit/maintain
  - Security issues if data isn't sanitized

# Summary - Forms with JS

- Events allow you to react at different times
- Can examine content of fields
- Can prevent submission
- Can change CSS to change styling
- Can display messages to the user

Powerful, but requires effort

- Detailed work
- Needed skills to distinguish yourself!

# Summary - Common Form Events

- `input` - Check as typed
- `keydown` - Edit WHILE typing
- `focus/blur` - Check after leaving
- `change` - Check after change complete
- `submit` - form event, check before submit
  - Should always be checked on submit



# Summary - Showing User Results

- Add/Remove text using `.innerText`
  - Pro: Secure
  - Con: Text in JS
- Add/Remove HTML using `.innerHTML`
  - Pro: Most Control
  - Cons: HTML in JS, security risk if user data
    - Never use user supplied data!