SUBJECT: Submitting a Form DATE:

The submit event

- triggers when a form is submitted
- can be submitted in 2 ways:
  1. clicking `<input type="submit">` or `<input type="image">`
  2. pressing Enter on an input field
- `event.preventDefault()` may cancel a form submit event if there are errors on the form

Example using both methods. The form will not be sent due to return false ⟶

```
<form onsubmit="alert('submit!');return false">
  First: Enter in the input field <input type="text" value="text"><br>
  Second: Click "submit": <input type="submit" value="Submit">
</form>
```

First: Enter in the input field text
Second: Click "submit": Submit

- A click event is still triggered when a form is sent by pressing the enter key.

⟶ it triggers on `<input type="submit">`

```
<form onsubmit="return false">
 <input type="text" size="30" value="Focus here and press enter">
 <input type="submit" value="Submit" onclick="alert('click')">
</form>
```

Focus here and press enter Submit

SUBJECT: Submitting a Form DATE:

Calling form.submit()

- submits a form to the server manually
- the submit event is not generated
  → it is assumed that if the programmer calls form.submit(), then the script already did all its processing

```
let form = document.createElement('form');
form.action = 'https://google.com/search';
form.method = 'GET';

form.innerHTML = '<input name="q" value="test">';

// the form must be in the document to submit it
document.body.append(form);

form.submit();
```

SUBJECT: FormData DATE:

Sending HTML Forms using FormData

- FormData is an object to represent HTML form data
-
```
let formData = new FormData([form]);
```

→ if a form is provided, the constructor automatically captures its fields

→ network methods (i.e. fetch) can accept a FormData obj. as a body

- sent out with Content-Type: multipart/form-data

Example sending a simple form →

```
<form id="formElem">
  <input type="text" name="name" value="John">
  <input type="text" name="surname" value="Smith">
  <input type="submit">
</form>

<script>
  formElem.onsubmit = async (e) => {
    e.preventDefault();

    let response = await fetch('/article/formdata/post/user', {
      method: 'POST',
      body: new FormData(formElem)
    });

    let result = await response.json();

    alert(result.message);
  };
</script>
```

* the server code (not shown) accepts the POST request & replies "User Saved"

SUBJECT: FormData Methods DATE:

Can modify fields in FormData w/these methods:

→ formData.append(name, value)
- adds a form field w/given name & value
- can append multiple same-named fields

→ formData.append(name, blob, fileName)
- adds a field w/given name as if it were <input type="file">

→ formData.set(name, value)
→ formData.set(name, blob, fileName)
- both set methods remove all fields w/given name, so make sure there aren't multiple fields w/the same name

→ formData.delete(name)
- removes field w/given name

→ formData.get(name)
- get the value of the field w/given name

→ formData.has(name)
- returns true if there is a field w/given name

SUBJECT: Sending Forms DATE:

- can iterate over FormData using for... of loop

```
let formData = new FormData();
formData.append('key1', 'value1');
formData.append('key2', 'value2');

// List key/value pairs
for(let [name, value] of formData) {
  alert(`${name} = ${value}`); // key1 = value1, then key2 = value2
}
```

Sending a Form w/a File

- always sent as Content-Type: multipart/form-data
  → this encoding also allows <input type="file"> fields to be sent

```
<form id="formElem">
  <input type="text" name="firstName" value="John">
  Picture: <input type="file" name="picture" accept="image/*">
  <input type="submit">
</form>

<script>
  formElem.onsubmit = async (e) => {
    e.preventDefault();

    let response = await fetch('/article/formdata/post/user-avatar', {
      method: 'POST',
      body: new FormData(formElem)
    });

    let result = await response.json();

    alert(result.message);
  };
</script>
```

John Picture: Choose File No file chosen Submit

SUBJECT: Sending Forms DATE:

Sending a form with Blob data

- dynamically-generated binary data (i.e. images) can be sent as a blob

  → blobs can be supplied via the fetch parameter body

- is often more convenient to send images as part of a form, not separately

  → handling multipart-encoded forms is also easier on the server

The image blob is added as if there were

<input type="file" name="image">

... the server will read it as if it were a regular form submission

```
<body style="margin:0">
 <canvas id="canvasElem" width="100" height="80" style="border:1px solid"></canvas>

 <input type="button" value="Submit" onclick="submit()">

 <script>
   canvasElem.onmousemove = function(e) {
     let ctx = canvasElem.getContext('2d');
     ctx.lineTo(e.clientX, e.clientY);
     ctx.stroke();
   };

   async function submit() {
     let imageBlob = await new Promise(resolve => canvasElem.toBlob(resolve, 'image/png'));

     let formData = new FormData();
     formData.append("firstName", "John");
     formData.append("image", imageBlob, "image.png");

     let response = await fetch('/article/formdata/post/image-form', {
       method: 'POST',
       body: formData
     });
     let result = await response.json();
     alert(result.message);
   }

 </script>
</body>
```

SUBJECT: Summary DATE:

Summary

FormData objects are used to capture HTML form and submit it using `fetch` or another network method.

We can either create `new FormData(form)` from an HTML form, or create an object without a form at all, and then append fields with methods:

- `formData.append(name, value)`
- `formData.append(name, blob, fileName)`
- `formData.set(name, value)`
- `formData.set(name, blob, fileName)`

Let's note two peculiarities here:

1. The `set` method removes fields with the same name, `append` doesn't. That's the only difference between them.
2. To send a file, 3-argument syntax is needed, the last argument is a file name, that normally is taken from user filesystem for `<input type="file">`.

Other methods are:

- `formData.delete(name)`
- `formData.get(name)`
- `formData.has(name)`