JavaScript 2

CS571: Building User Interfaces

Cole Nelson & Professor Yuhang Zhao

Before Lecture

- Clone today's code to your machine.
- Download and install Postman!

Ethical Hacking

Ask for permission first.

Running Code

Just double-click on the `index.html` file.

You are welcome to use utilities built into VS Code if you would like, but we won't cover them.

JavaScript 1 Recap

- The Web is made up of HTML, CSS, and JS!
  - HTML: structure
  - CSS: styling
  - JS: behavior
- CSS and JS can be applied to HTML inline, internal, or externally.
- Is this true for cs571.org? Let's look!

JavaScript 1 Recap

Use `document` to reference the DOM.

```
let title = document.getElementById("articleTitle");
let loginBtn = document.getElementsByName("login")[0];
let callouts = document.getElementsByClassName("callout"); // *
```

*class refers to a **CSS** class

We can add *event listeners* or read/modify *properties*.

StackBlitz

Using these DOM elements, we can change the title of the article, add an action for when the button is clicked, and make all of the callouts red.

```
title.innerText = 'My Website!';
loginBtn.addEventListener("click", () => {
  alert("You are advancing to the next part of the site...");
});

for (let callout of callouts) {
  callout.style.color = "red";
}
```

StackBlitz

For HW2...

You may need to manipulate other properties of the HTML elements, such as `innerText`, `innerHTML`, `className`.

Look up their documentation on MDN (or any other resource!)

What will we learn today?

- How to work with APIs and JSON data?
- How to write async functions?
- How to write declaratively?

What is JSON?

Definition: JavaScript Object Notation (JSON) is a structured way to represent text-based data based on JS object syntax.

Refresher: JS Objects

Definition: Objects are unordered collection of related data of primitive or reference types defined using key-value pairs.

```
const instructor = {
  firstName: "Cole",
  lastName: "Nelson",
  roles: ["student", "faculty"]
}
```

JSON Equivalent

```
{
  "firstName": "Cole",
  "lastName": "Nelson",
  "roles": ["student", "faculty"]
}
```

What's the difference? A JS Object is executable code; JSON is a language-agnostic representation of an object. There are also slight differences in syntax.

You can write comments in JS Objects...

```
const drinks = [
  {
    name: "Mimosa",
    ingredients: [
      {name: "Orange Juice", hasAlcohol: false},
      {name: "Champagne", hasAlcohol: true}
    ]
  },
  {
    name: "Vesper Martini", // shaken, not stirred
    ingredients: [
      {name: "Gin", hasAlcohol: true},
      {name: "Vodka", hasAlcohol: true},
      {name: "Dry Vermouth", hasAlcohol: true},
    ]
  }
]
```

... but not in JSON!

```
[
  {
    "name": "Mimosa",
    "ingredients": [
      { "name": "Orange Juice", "hasAlcohol": false },
      { "name": "Champagne", "hasAlcohol": true }
    ]
  },
  {
    "name": "Vesper Martini",
    "ingredients": [
      { "name": "Gin", "hasAlcohol": true },
      { "name": "Vodka", "hasAlcohol": true },
      { "name": "Dry Vermouth", "hasAlcohol": true }
    ]
  }
]
```

Conversion

Because JS Objects and JSON are so similar, it is easy to convert between them.

- `JSON.parse` JSON String → JS Object
- `JSON.stringify` JS Object → JSON string

Conversion Examples

Using `JSON.parse` and `JSON.stringify`.

```
const myObj = JSON.parse('{"name": "Cole", "age": 25}');
const myStr = JSON.stringify(myObj);

console.log(typeof myObj);
console.log(typeof myStr);
```

```
object
string
```

StackBlitz

What is an API?

Definition: An application programming interface (API) is a set of definitions and protocols for communication through the serialization and de-serialization of objects.

JSON is a language-agnostic medium that we can serialize to and de-serialize from!

How do we make an API request?

- Your browser!
- cURL
- Postman
- JavaScript

Try making an API request to...

- https://v2.jokeapi.dev/joke/Any?safe-mode
- https://cs571.org/api/f23/weekly/week02

In-Class Activity

Fetch from the Jokes and CS571 APIs using...

- Your browser!
- curl
- Postman

Request for JSON

- Requests can be `synchronous` or `asynchronous`.
- `asynchronous` requests are recommended as they are *non-blocking*. Typically, they use a *callback* when the data is received and lets the browser continue its work while the request is made.

More on synchronous/asynchronous requests

Making Asynchronous HTTP Requests

Two key methods: `XMLHttpRequest` (old) and `fetch` (new). `fetch` is a promise-based method.

- `Promise` objects represent the eventual completion/failure of an *asynchronous* operation and its resulting value.
- `async` / `await` — keywords to indicate that a function is *asynchronous* -- will learn later!

fetch()

```
fetch(url)
  .then((response) => response.json()) // implict return
  .then((data) => {
    // fetch has already parsed data from JSON to a JS object!
    // Do something with the data
  })
  .catch(error => console.error(error)) // Print errors
```

Fetching Jokes

`fetch()`

Fetch happens *asynchronously*.

```
fetch(url)
  .then((response) => response.json()) // implict return
  .then((data) => {
    console.log("Data takes time to fetch -- I won't print until much later!")
  })
  .catch(error => console.error(error)) // Print errors

console.log("I will print first!")
```

StackBlitz

`fetch()` from a CS571 API

```
fetch(url, {
  method: "GET",
  headers: {
    "X-CS571-ID": CS571.getBadgerId()
  }
})
.then(response => response.json())
.then(data => {
  // Do something with the data
})
.catch(error => console.error(error)) // Print errors
```

There is a database that maps your BID to a WISC ID!

Your Turn!

Fetch data from our API and do "interesting" things!

```
https://cs571.org/api/f23/weekly/week02
```

1. Can you get *any* data back?
2. Can you dynamically put my name on the HTML?
3. What are my favorite colors?
4. What semesters did I take more than 15 credits?
5. What are the names of my plants that survived?

Callback Functions

`then` and `catch` take a *callback function* as an argument.

Definition: A *callback function* (sometimes called a *function reference*) is passed into another function as an argument, which is then invoked inside the outer function to complete a routine or action.

More on callback functions

`processUserInput` takes a *callback function*.

```
const name = prompt('Please enter your name.');
function processUserInput(callback) {
  alert("Incoming message!")
  callback(name);
}
function greeting1(name) {
  alert('Hello ' + name);
}
const greeting2 = (name) => {
  alert('Whats up ' + name);
}
processUserInput(greeting1);
processUserInput(greeting2);
processUserInput((name) => alert("Welcome " + name));
```

StackBlitz | PythonTutor

Declarative vs. Imperative

Writing "clean code".

Declarative vs Imperative Programming

The following is imperative...

```
for (let obj of arr) { /* stmts */ }
```

The following is declarative...

```
arr.forEach((obj) => { /* stmts */ })
```

We typically prefer *declarative* programming over *imperative* programming.

Declarative vs Imperative Programming

Declarative array functions include `forEach`, `map`, `slice`, `concat`, `filter`, `some`, `every`, and `reduce`.

Today we'll learn about `forEach`, `filter`, and `map`!

Eventually, we'll learn how to use all of these!

`forEach`, `filter`, and `map`

All are used on *arrays*.

All take a callback as an argument.

This callback then takes an argument* representing the current element, e.g.

```
students.forEach((student) => /* */);
ids.map((sId) => /* */);
grades.filter((grade) => /* */);
```

* also an *optional* second arg of index `grades.filter((grade, i) => /* */)`

forEach

`forEach` performs a function on each element of an array.

```
["Bessy", "Rob", "Bartholomew"].forEach(name => {
  if (name.length > 5) {
    console.log(`${name} is a long name.`);
  } else {
    console.log(`${name} is a short name.`);
  }
});
```

StackBlitz

`filter`

`filter` performs a function on each element of an array and *returns* an array of those elements whose function call returned true.

```
const shortNames = ["Bessy", "Rob", "Bartholomew"].filter(name => {
  if (name.length <= 5) {
    return true;
  } else {
    return false;
  }
});
console.log(shortNames);
```

StackBlitz

`filter`

`filter` performs a function on each element of an array and *returns* an array of those elements whose function call returned true.

```
const shortNames = ["Bessy", "Rob", "Bartholomew"].filter(name => name.length <= 5);
const longNames = ["Bessy", "Rob", "Bartholomew"].filter(name => name.length > 5);
console.log(shortNames);
console.log(longNames);
```

StackBlitz

`map`

`map` performs a function on each element of an array and *returns* an array of the the return of those function calls.

```
const nameLengths = ["Bessy", "Rob", "Bartholomew"].map(n => n.length);
console.log(nameLengths);
```

StackBlitz

Chaining Declarative Functions

Of those with short names, how many letters are in their name?

```
["Bessy", "Rob", "Bartholomew"]
  .filter(name => name.length <= 5)
  .map(name => name.length);
```

StackBlitz

Chaining Declarative Functions

Of those with short names, what are their names and do they have a *very* short name?

```
["Bessy", "Rob", "Bartholomew"]
  .filter(n => n.length <= 5)
  .map(n => {
    return {
      name: n,
      isVeryShort: n.length <= 3
    }
  });
```

StackBlitz

Your turn!

Can you write declarative code within your fetch to...

1. Print out what are my favorite colors?
2. What semesters did I take more than 15 credits?
3. What are the names of my plants that survived?

List Favorite Colors

Imperative

```
for (const color of data.favColors) {
    console.log(color)
}
```

Declarative

```
data.favColors.forEach((color) => console.log(color));
```

Get Semester Credit History

Imperative

```
for (const sem of data.creditHistory) {
    if(sem.cred > 15) {
        console.log(sem.semester)
    }
}
```

Declarative

```
data.creditHistory
  .filter(sem => sem.cred > 15)
  .forEach(sem => console.log(sem.semester))
```

Get Alive Plant Names

Imperative

```
const plants = data.plants;
let alivePlants = [];
for (const plant in plants) {
    if(plants[plant].alive) {
        alivePlants.push(plant);
    }
}
console.log("Surviving plants...")
console.log(alivePlants);
```

Declarative?

Get Alive Plant Names

This is an **object**, not an array!

```
{
    "foto": {
        "alive": true,
        "type": "SUCCULENT"
    },
    "syn": {
        "alive": true,
        "type": "SUCCULENT"
    },
    "thesis": {
        "alive": false,
        "type": "BAMBOO"
    }
}
```

`Object.keys(obj)`

We can use `Object.keys` to get the keys of an object...

`Object.keys(plants)` -> `['foto', 'syn', 'thesis']`

Your turn! Using this, try to solve the problem *declaratively*.

Get Alive Plant Names

Declaratively

```
const plantNames = Object.keys(data.plants);
const alivePlants = plantNames.filter(name => data.plants[name].alive)
console.log(alivePlants);
```

Other `async` Functions

- `setInterval(callback, interval)` perform a callback function every interval milliseconds.*
- `setTimeout(callback, timeout)` perform a callback function in timeout milliseconds.*

Fetch Jokes (w/ `setInterval`)

Fetch Jokes (w/ `setInterval` and `setTimeout`)

* approximately

What did we learn today?

- How to work with APIs and JSON data.
- How to write async functions.
- How to write declaratively.

Questions?