

CSCI E-33a (Web50)

Section 5

Ref: Lecture 5 (JavaScript)






Vlad Popil

Oct 12, 2022

Welcome!

About me:

Volodymyr “Vlad” Popil

-  Teaching Fellow for E-33a (since 2019)
-  Father of 4 y/o son
-  Master's (ALM) in Software Engineering
-  Software Engineer at Google
-  Born in Ukraine

Email: vlad@cs50.harvard.edu

Sections: Wed 8:30-10:00 pm ET

Office Hours: Fri 7:00-8:30 pm ET

Agenda

- Logistics
- Lecture review
- JS / Ajax
- Project 3
- Grading criteria (not exhaustive)
- Chrome Debugging - Javascript
- jshint
- `pycodestyle` , `pylint` (recap)
- Q&A

Logistics

Intro

- Refer to website: <https://cs50.harvard.edu/extension/web/2022/fall>
- Sections and office hours schedule on website sections
- Get comfortable with [command line](#) (cd, ls, cat, python ..., python -m pip ..., rm, touch)
- Text editor is usually sufficient to write code, BUT IDEs (e.g. VSCode) is faster!
- Zoom:
 - Use zoom features like raise hand, chat and other
 - Video presence is STRONGLY encouraged
 - Mute your line when not speaking (enable temporary unmute)
- Six projects:
 - Start early (or even better RIGHT AWAY!!!)
 - Post **and answer** questions on Ed platform
 - Remember: bugs can take time to fix
 - Grade $\rightarrow 3 \times \text{Correctness (5/5)} + 2 \times \text{Design [code] (5/5)} + 1 \times \text{Style [code] (5/5)}$ (Project 0 is an exception)
 - $15+10+5=30/30$ | e.g. Correctness can be 15, 12, 9, 6, 3, 0
 - Overall weights: projects (90%), section attendance (10%)
 - [Lateness policy](#) - 72 late hours combined for first 5 proj., then 0.1per minute => **16hrs 40 min**
 - Project 3 - Due Sunday, Oct 30th at 11:59pm EDT << **ONLY 19 FULL DAYS LEFT** >>
 - Another section

Reminders

- Sections/Office Hours:
 - Sections are recorded (published 72hrs), office hours are not
 - Real-time attendance is required of at least one section
 - Video and participation encouraged even more
- Section prep:
 - Watch lecture(s)
 - Review project requirements
- Office hours prep:
 - *~Write down your questions as you go, TODO, etc.~*
 - Come with particular questions

10,000 foot overview

- *Section 0 - SKIPPED*
- *Section 1+2 (Git + Python) - Chrome Dev Tools (Inspector), CDT (Network), Project 0, Grading aspects*
- *Section 3 (Django) - Env Config, Markdown, RegEx, IDEs, pycodestyle, Debugging, Project 1*
- *Section 4 (SQL, Models, Migrations) - VSCode, linting, DB modeling, Project 2*
- *Section 5 (JavaScript) - ~~eURL/Postman~~, jshint, CDT + IDE's Debugging, Project 3*
- *Section 6 (User Interfaces) - Animations, DB modeling, Pagination, Project 4*
- *Section 7 (Testing, CI/CD) - Test Driven Development, DevOps, Final Project*
- *Section 8 (Scalability and Security) - Cryptography, CAs, Attacks, App Deployment (Heroku)*

Most sections: material review, logistics, project criteria review, reminders, hints, etc.

Burning Questions?

Please ask questions, or topics to cover today!

Topics:

- *Last project (i.e. 2) issues re-assigning to FK, the only fix identified is to recreate the database:*
 - *Create FK with different name*
 - *Drop the origin field*
 - *Rename*

JavaScript

JavaScript

- Programming Language used for running client-side code.
- Useful for manipulating the Document Object Model (DOM)
- Employs Event-Driven Programming



Include JavaScript in page

- Write code between `<script>` tags in HTML
- Link to separate JavaScript file (preferred)

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Hello</title>
    <script>
      alert('Hello, world!');
    </script>
  </head>
  <body>
    <h1>Hello!</h1>
  </body>
</html>
```

```
<script src="counter.js"></script>
```

Functions

- Regular notation
- Arrow Notation
- Anonymous Functions

```
1 // Traditional function notation
2 function add(a, b) {
3     return a + b;
4 }
5
6 // Arrow notation
7 add = (a, b) => {
8     return a + b;
9 }
10
11 // Anonymous function
12 // (We've given it a name here, but no need to)
13 var anon = function(a, b) {
14     return a + b;
15 }
```

Variable Declaration

- var: Declares a function-scoped variable
- let: Declares a limited-scope variable
- const: Declares a constant value

```
// Declaring a Constant
```

```
const PI = 3.14;
```

```
// Declaring a function-scoped variable
```

```
var some_var = "Hello!";
```

```
// Declaring a limited scope variable
```

```
let i = 1;
```

Selecting DOM Elements

- `document.querySelector(query)` selects the **first** element matching the query
- `document.querySelectorAll(query)` selects all elements matching the query

Examples:

Query	Result
<code>...Selector('#header1');</code>	Selects element with ID header1
<code>...SelectorAll('.big');</code>	Selects all elements with class big
<code>...SelectorAll('image');</code>	Selects all images
<code>...Selector('form');</code>	Selects first form on the page

Manipulating DOM Elements

Attribute/Command	Meaning	Example
<code>element.innerHTML</code>	All HTML inside element	<code>header.innerHTML = "Hi!";</code>
<code>element.style.property</code>	The CSS styling of an element	<code>header.style.color = 'blue';</code>
<code>document.createElement(type)</code>	Creates new DOM element of specified type	<code>const item = document.createElement("li");</code>
<code>element.appendChild(child_name)</code>	Adds child element nested within parent	<code>listy.appendChild(item);</code>

Conditionals

- Use `===` for strict equality
(must be same type)
(preferred)
- Use `==` for normal equality
(can be different type,
must be same value)
- Example:
 - `3 == '3'` is True
 - `3 === '3'` is False

```
1  // Setting up variable:
2  let x = 4;
3
4  ✓ if (x < 0) {
5      |     console.log("negative");
6  ✓ } else if (x === 0) {
7      |     console.log("zero");
8  ✓ } else {
9      |     console.log("positive");
10  | }
```


JavaScript Objects

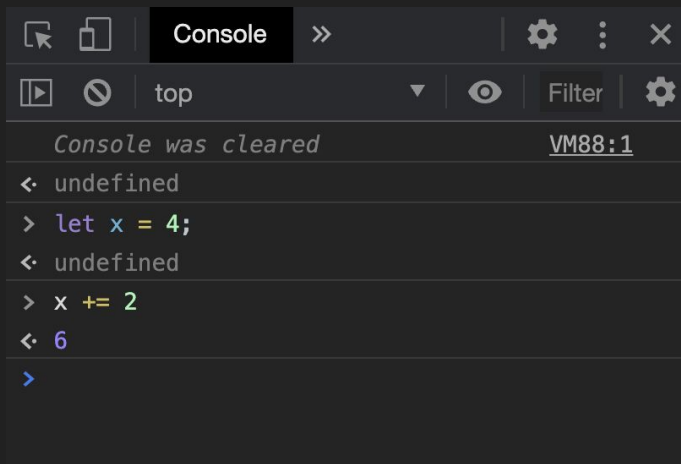
- Stores Key-Value pairs similar to a Python Dictionary

ages

```
1  let ages = {  
2      "Connor": 20,  
3      "Chris": 49,  
4      "Sam": 15  
5  };|
```

JavaScript Console

- Command-line like tool on web browsers where we can run JavaScript
- Use `console.log(x)` to print x to the console
- In Chrome: Right-Click -> Inspect ->



Timing

- `setInterval(function_name, milliseconds);`
- Ex: `setInterval(count, 1000);`

Local Storage

- Storage within the user's web browser
- Get item using:

```
localStorage.getItem("key_name");
```

- Set item using:

```
localStorage.setItem("key_name", value);
```

Application Programming Interfaces (APIs)

- A set of functions and values that clients can access through requests.
- Data typically transferred as a JSON (JavaScript Object Notation) object
- fetch function makes a web request and returns an HTTP response:

```
JS api.js >  getQuestion
1  getQuestion = () => {
2      const url = "https://jservice.io/api/random";
3      fetch(url).then(response => {
4          return response.json();
5      }).then(data => {
6          console.log(data);
7      })
8  }
```

Handle Form Submission

- Use the “submit” event to add a function to be run on submission of a form:

```
object.addEventListener("submit", event =>  
myFunction(event));
```
- Use the “value” attribute of each HTML element to access user input:

```
let f_name = document.querySelector("#first").value;
```
- Use the `event.preventDefault()` function in your function to prevent the default action of the form. (Works with any other event as well if using `addEventListener` function).

```
event.preventDefault();
```
- Note: `return false;` was used in lecture in place of `event.preventDefault`, but that can be problematic at times.

Style

Major Style/Design Issues

- Keep indentation consistent
- **camelCase** JS typically used instead of **snake_case** (Python)
- Keep spacing consistent
- Don't repeat code!!!
- Use separate functions when it improves readability

Questions?

Demo

Javascript

Demo

AJAX

Demo

Project

Project 3

- Start early!!!
- Make a checklist of requirement and check all before submission
- Make sure there's no bugs (**especially those that don't happen all time**)
- Focus on functionality (NOT PRETTINESS)!!!

Project 3

- Refactor functions to:
 - add email to the mailbox (render email: create element and append)
 - archive current email (and go back to inbox) - takes T/F
 - compose
 - reply email -> compose email and plug in some values
 - view single email
 - send email
- Styling the inbox entry:
 - CSS classes
 - `` for timestamp
 - CSS child selector

Project 3

- Reading mailbox (inbox/sent/archive)
 - on POST email add **`event.preventDefault();`**
 - Style your 'read' emails, e.g.:

```
if (email.read) {  
    row.classList.add('email-read');  
}
```

- Add event listeners to each email:

```
email.addEventListener('click', function() {  
    view_email(email.id);  
});
```

- When loading mailbox make sure to hide the single view as well

Project 3

- Showing single email:
 - You can display button depending on which mailbox you are observing (or by fetching info)
 - Changing visibility →
`myArchiveElement.style.display = if globalMailbox === 'inbox' ? 'inline-block' : 'none';`
 - When showing single email, flush all values before fetching, in case no data or partial data returned
 - Make sure to mark email as read when landing on the page
 - Breaking ``n`` with `
`: OR? `<pre></pre>`
 - `email.body.split("\n").forEach(line => {`
 - 1) insert line
 - 2) insert `
`
 - }

Project 3

- Showing single email HTML:
 - separate entries for : From/To/Subject/Timestamp
 - buttons: reply/(archive|unarchive)
 - Also the div for body

Project 3

- Archiving
 - Simple method that calls backend:
 - No global value you need to pass T/F and email id
- Reply:
 - If `subject.slice(0, 4) !== 'Re: '`
 `subject += 'Re: '`
 - `.....element.value= `

 On ${email.timestamp}
${email.sender} wrote:\n${curEmail.body}`;`
 - hi

On ____ wrote: "Hello"

Design

What can be considered (not exclusively):

- Proper refactoring (copy-paste is usually a no-no)
- Use of constants/vars:
 - 1. const
 - 2. let
 - ~~00000. var~~
- Proper use of functions
- More reasonable solution
- Code/file structure
- `fetch('/email/' + id)` -> `fetch(`/email/${id}`)`

Design (continued)

What can be considered (not exclusively):

- Repetitive use of `querySelector`?
- Proper data structures
- `==` vs `===` ?
 - `const x = 5`
 - `const y = '5'`
 - `x == y -> T`
 - `x === y -> F`
- Code repetition

Style

What can be considered (not exclusively):

- *jshint (indentations, line breaks, long lines)*
- COMMENTS!
- Naming for variable, function, files, etc.:
 - getemailbyid -> get_email_by_id (Python convention)
 - getEmailById (JS convention)
- Consistency is the key!

Style (continued)

What can be considered (not exclusively):

- ‘ vs “ consistency
- camelCase(c*, Javascript, Java) vs snake_case (Python)
- == vs ===

jshint

- UI:
 - <https://jshint.com/>
- CLI:
 - brew update
 - brew doctor
 - brew install node
 - npm install -g jshint
 - In ~/.jshintrc add:
 - {
 - "esversion": 6
 - }

pycodestyle (formerly pep8)

- `python -m pip install pycodestyle`
- `pycodestyle app.py --max-line-length=120`

pylint (checks beyond style, but including)

- `python -m pip install pylint`
- `pip install pylint-django`
- `pylint app.py --load-plugins pylint_django`

Chrome Developer Tools (Network)

In Chrome:

1. Right click
2. Inspect
3. → Demo

Extremely powerful! Let's try Javascript...

Random Tips

- Video Speed Controller (Chrome Extension)

- Spotify + Hulu + Showtime >> \$5
- GitHub Education Pack
- Windows licence (<https://harvard.onthehub.com>)
- Chrome Tabs
- DSA:
 - [Video by CS50: https://www.youtube.com/watch?v=ODCj1B_ScQI&list=channel/CS50](https://www.youtube.com/watch?v=ODCj1B_ScQI&list=channel/CS50)
 - Leetcode / AlgoExpert / Etc.
 - Stanford Algorithms Specialization (EdX link /Coursera) - more theory (time consuming)
 - e22 seems good!
 - e20 + e 124 (combo) - HARD!
- System Design:
 - Grokking System Design
 - Alex Xu - System Design

Resources

- <https://github.com/vpopil/e33a-sections-fall-2022>

CSCI E-33a (Web50)

Section 5

Ref: Lecture 5 (JavaScript)

Vlad Popil

Oct 12, 2022