CSE 331
Software Design & Implementation

Winter 2025

Section 1 – HW1 and Tools

Welcome

- Let's all introduce ourselves:
  - Name and pronouns
  - Year
  - What other classes you are taking this quarter
  - How many times would you have to find a seagull in your place of living before thinking someone was intentionally putting them there, and why?

Administrivia

- **Software Setup:** Due this morning!
  - If you haven't finished it yet, still submit your screenshot when you do, don't worry about late days for this one
- **Knowledge Quiz:** Due Monday, Jan 13th @ 11pm
- **Homework 1:**
  - Due Wednesday, Jan 15th @ 11pm
  - Released this evening (usual cycle)

Coding Setup

Software we will use

- **Bash**: command-line shell (built-in on Mac, see course website to download Windows version)
  - Run `echo "${BASH_VERSION}"` to check for download
- **Git**: version control system (built-in on Mac, Windows version comes with Bash, above)
- **Node**: executes JavaScript code on the command-line (see link on course website to install)
  - Run `node -v` to check for download
- **NPM**: package manager (comes with Node, above)
- **VS Code** or the editor of your choice

Node Demo

- **Node**: executes JavaScript code on the command-line (see link on course website to install)
  - Run `node -v` to check for download

- Useful for playing with the JavaScript language

- Try this to see what it does (does it crash?)
  - first start `node` and then type this in:

```
const x = {a: 1, b: "two"};
console.log(x.c);
```

Git Demo

- **Git**: version control system (built-in on Mac, Windows version comes with Bash, above)

- Almost all professionals use some kind of version control system
  - git is probably the most popular today
  - git can be tricky to learn / understand

- We will only need it for getting the starter code
  - here is the command for sec1 (similar command for HW1)

```
git clone
https://gitlab.cs.washington.edu/cse331-25wi/materials/sec01.git
```

NPM Demo

- **NPM**: package manager (comes with Node)

- Used to
  - install all the libraries needed for our code
  - compile, test, and run our code

- Use this command to install the libraries needed for sec1

```
npm install --no-audit
```

(leaving off --no-audit will generate some **bogus** error messages)

VSCode Demo

- **VS Code** or the editor of your choice

- VS Code is relatively lightweight IDE
  - primary support for JavaScript and TypeScript (good for us)

- Extensions provide support for other languages and tools

NPM Start

- **NPM**: package manager (comes with Node)

- Use this command to start

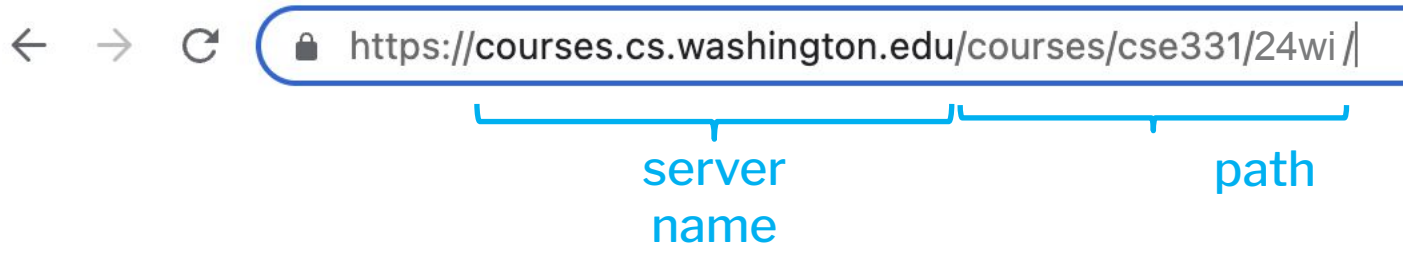
```
npm run start
```

- Then navigate to this URL in Chrome to see it work

```
http://localhost:8080
```

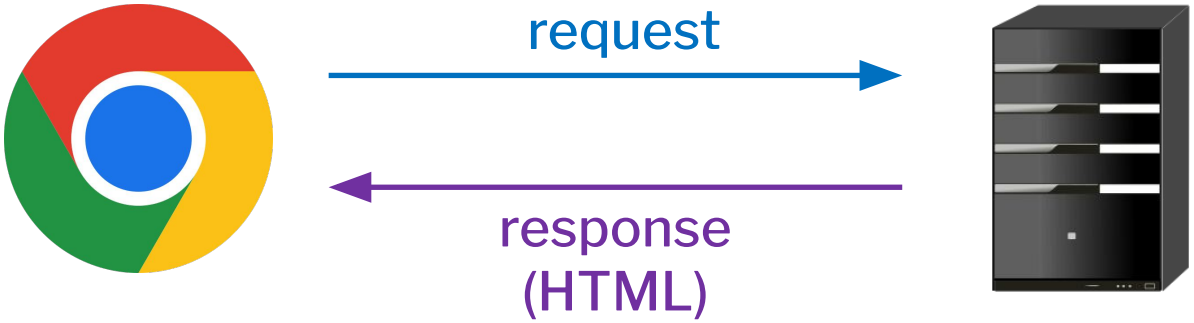
Browser Operation

- Browser reads the URL to find the server to talk to



- Contact the given server and request the given path:

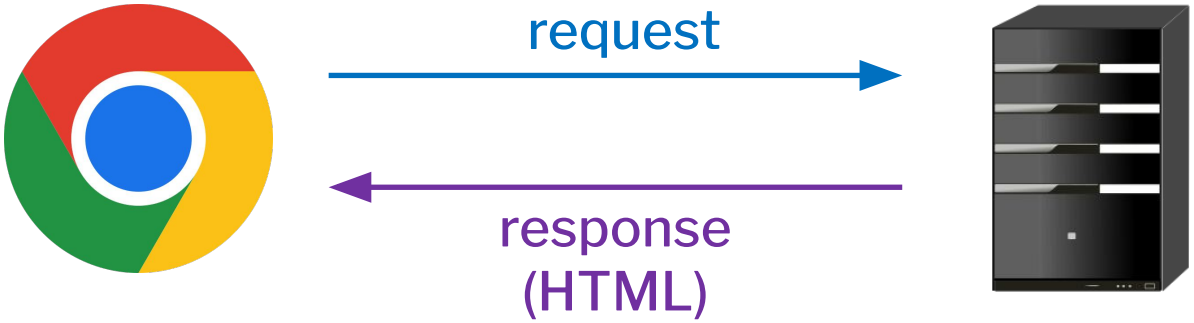
Browser Operation



- HTML page can load JavaScript
  - starter code's `index.html` includes `index.tsx`

- Each time the page loads, browser executes `index.tsx`

Development Environment



- "`npm run start`" starts a server that the browser can contact
  - server is running on this machine (localhost)
  - (more on servers later this quarter…)

- This server returns index.html but adds compiled JS into the page
  - also adds code to reload if the source code is changed!

Custom Server (Review)

- **Query Parameters** (e.g. `?name=Fred`) in requests (`req`)

```
const F = (req, res) => {
    if (req.query.name === undefined) {
        res.status(400).send({response: "Missing
'name'"});
        return;
    }
    res.send(`Hi, ${req.query.name}`);
}
```

HTTP Terminology (Review)

- **HTTP** Request include:
  - URL: Path and query parameters
  - Method: Get/Post
    - Get is used to read data on the server (can paste raw url in browser and get result back)
    - Post is used to change data on the server (cannot paste raw url in browser)
  - Body (for Post only)
    - used for sending large or non-string data to server
- **HTTP** Response Status Codes include:
  - 200 (ok)
  - 400-499 (Client error)
  - 500-599 (Server error)

Debugging

Bugs can be split into 2 stages:

- **Failure:** the incorrect behavior that is externally visible (e.g. visible to a user/client)
- **Defect ("the bug"):** the actual mistake in the code

Debugging is the *search* from the **failure** back to the **defect**

Example

```
let A = [3, 28, 7, 15, 12, 234, 89, 834];
let s = 0;
for (let i = 0; i <= A.length; i++) {
    s += A[i];
}
console.log(s);
```

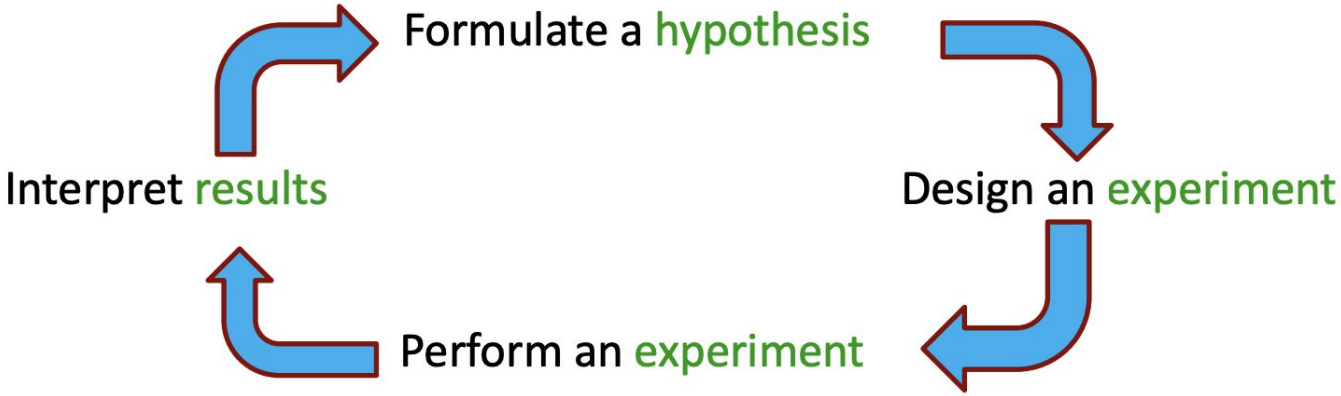
- **Failure:** prints NaN!
- **Defect:** “<=” should be “<” so we stay inside the array

Debugging Tips

- **Check the easy stuff**
  - check files are saved
  - restart server/computer/VS Code (this works more than you think)
- **Create a minimal example that demonstrates problem**
  - need a way to reliably reproduce the failure
  - shrink the input that fails
- **Look for common silly mistakes**
  - comparing records with ===
  - misspelling the name of a method you were implementing usually caught by a type checker
  - passing arguments in the wrong order

Debugging Tips

- **Make sure it is a bug!**
  - check the spec carefully
- **Be thoughtful & systematic**
  - don't just try random changes
  - write down what you have tried
  - don't try the same thing again and again
  - think of experiments that reduce search space

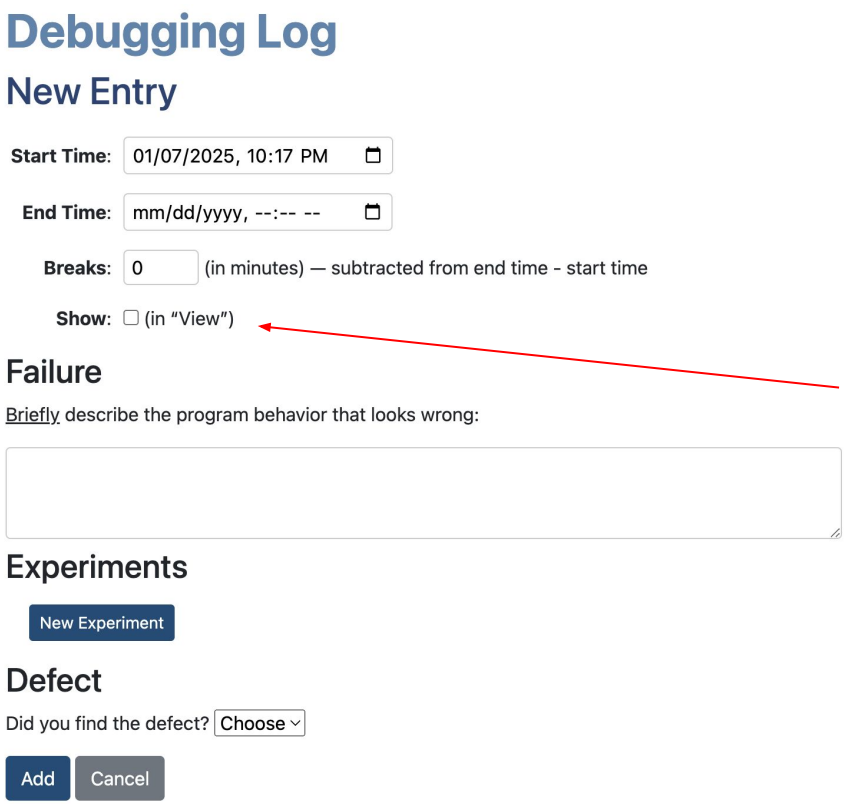


Debugging Tips

- **Try explaining the problem to someone / something**
  - can even be a rubber duck (“rubber ducking”)
  - Talking through the problem often helps you spot it – this happens all the time
- **Get some sleep!**
  - the later it gets, the dumber you get
  - often don’t realize it until 4–5am
  - Common to wake up and instantly see the problem

Bug Journalling

Practice Log: https://comfy.cs.washington.edu/service/hw1-practice
Homework Log: https://comfy.cs.washington.edu/service/hw1



Be sure to check this box for the bug entry to show up when you view the debugging log

Bug Journaling

- **Before you start debugging:** Add a bug log entry and document the error and start time of when you started debugging
- **After you have finished debugging:** Fill out the log entry with your findings from the debugging session. This will include:
  - Any experiments you ran to find the bug
  - If you found the defect (if so, where)
  - If type checking would have prevented the bug and why
- 1 bug should be 1 log entry. Do not split a single bug into multiple log entries.
- Be sure to check the “Show” checkbox so that the entry will show up when you view the debugging journal
- For initial HWs, you must debug for at least 4 hours and at most 6 hours regardless of whether you fix the bug or not

Your Turn!

- Take a look at the worksheet and fix those bugs!
- Be sure to document your debugging process and any experiments you did to find the bug