1. Short Answers (20pts total)

1. HTML: Happy Little DOM Trees! (2.5pts)

1. <p>
2. <section>
3. or
4.
5. <footer>

2. Web Accessibility (2pts) What are 2 different ways we have learned for making a website more accessible?

Possible answers include:

- Semantic tags
- Descriptive alt text on images
- Responsive layout (e.g. to support different screen resolutions for BVI users)
- Relative layout (% vs. px)
- Descriptive <label> tags in forms
- Color contrast with CSS to support different color visions
- Support for alternative navigation (e.g. keyboard)

3. Event Listeners (2pts) Consider the following JS program:

a	b	c	e

`ribbit!`
`meow!`
`woof!`
`quack!`
`meow!`
`woof!`
	`meow!`
	`woof!`
	`ribbit!`
	`quack!`
	`meow!`
	`woof!`
	`ribbit!`
		`ribbit!`
		`woof!`
			`meow!`
			`woof!`
			`woof!`
			`quack!`
			`meow!`
			`woof!`

4. JS Timers (2pts) Consider the following JS program:

a	b	c	d	e
1 doggo 2 doggo DUBS! 3 doggo 4 doggo DUBS! 5 doggo 6 doggo	1 doggo 2 doggo 3 doggo DUBS!	1 doggo 2 doggo DUBS! 1 doggo 2 doggo DUBS! 1 doggo 2 doggo	1 doggo 2 doggo DUBS! 3 doggo 4 doggo 5 doggo DUBS! 6 doggo	1 doggo 2 doggo DUBS! DUBS! DUBS! DUBS! DUBS! DUBS!

5. Node.js Error-Handling (2pts)

a. Provide a specific example where it would be more appropriate to return a 400 error instead of a 503 error in a web service:

Possible answers include specific types of invalid requests:
- Missing/invalid parameter required for a GET/POST request

or specific web service examples:
- Missing a required GET parameter for HW4/HW5
- Parameter value does not correspond to a found resource on the server (e.g. character directory)

b. Provide a specific example where it would be more appropriate to return a 503 error instead of a 400 error in a web service:

Possible answers include:
- Database connection is down
- User credentials for mysql2 are incorrect

6. GET vs. POST (2pts)

a. Provide an example where a GET request would be more appropriate than a POST request for a web service.

Possible answers include:
- Requests which don't send sensitive data
- Search queries
- Requests where we might want to cache/bookmark the url
- Request that solely gets information from a server

b. Provide an example where a POST request would be more appropriate than a GET request for a web service.

Possible answers include:
- Sending data with a form
- Requests where data is more sensitive and should be encrypted (e.g. login)
- Requests where the server will be modified

7. Validation Methods (2pts)

a. What is one advantage of validating user input on the client (HTML5 or JS) rather than on the server (Node)?

Possible answers include:

- It is faster to validate on the client than on the server since the validation doesn't have to wait for the input to be sent in a request
- Saves network/server resources with fewer requests to server
- It can lead to a better UI/UX experience due to quicker/custom feedback to a user about why their input is invalid
- It can help prevent malicious user input by preprocessing the input before sending to the server

b. What is one advantage of validating user input on the server as opposed to on the client?

Possible answers include:

- Validating user input on the server is more secure than validating on the client, client can't see source code vulnerabilities in server-side code
- Can check input against data on server such as username/passwords, which (hopefully) aren't on the client-side

8. Data Storage Trade-Offs (2pts)

a. Recall that `localStorage` can be used to store data on a user's browser. In 1-2 sentences, explain when it is more appropriate to store data with `localStorage` instead of with a SQL database.

Possible answers include:

- If you want to fetch information from the server and store it locally vs retrieving it again from the server each time you need it.
- If you want to store information across browser sessions so a user doesn't have to retrieve it from the server multiple times.
- Use localStorage if you want to store a set of user/machine specific preferences on the local machine

b. It is possible to store/process data on a server using .txt or .json files. What is one advantage of using SQL databases to store data instead?

Possible answers include:

- SQL databases are more secure than txt/JSON files
- Easier to handle multiple requests from clients to process/modify data in a database
- It is often more efficient (in terms of space and time) to manage data with SQL than with txt/JSON files
- Databases allow for transactions to allow concurrent users to modify parts of the database which can't be done with txt/JSON
- Easier to query/update data with SQL

9. Node FIle I/O (1.5pts)

Statement	Resolved Promise Value
`await fs.readdir("birds");`	`["eagle", "heron", "merlin"]`
`await globPromise("birds/merlin/i*");`	`["birds/merlin/image.png", "birds/merlin/info.txt"]`
`await globPromise("birds/*/*mp3");`	`["birds/eagle/call1.mp3", "birds/heron/call1.mp3", "birds/heron/call2.mp3", "birds/merlin/call1.mp3"]`

10. Regex (2pts) For each of the two regular expressions, circle all the string(s) below that match it:

i. `/^[irA]{3}b*[n]+b?$/`

- A3bnnnb
- rAin
- iiiBnB
- AAibnbb
- Airbnb

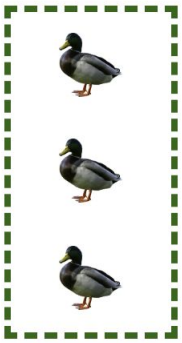
ii. `/^[A-Z]L(3){1,}..$/`

- AL3rt
- GL33..$
- XL33TX
- LLLL
- WIL

2. CSS Writing (13pts): Swimmin' in CSS

Part A: The Fountain Mallards

Expected output:



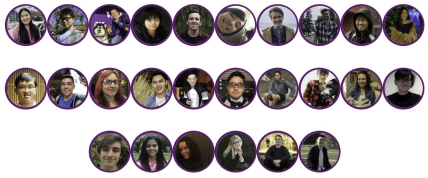
```
/* Write your CSS for Part A below */

#mallard-container {
    width: 100px;
    height: 200px;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: space-around;
    border: 4px dashed green;
}

#mallard-container img {
    height: 40px;
    width: 40px;
}
```

Part B: Instructor Party!

Expected output:



```
/* Write your CSS for Part B below */

#party-container {
    height: 200px;
    width: 80%;
    display: flex;
    flex-wrap: wrap;
    justify-content: center;
}

.party-pic {
    height: 40px;
    width: 40px;
    border-radius: 50%;
    border: 2px solid purple;
}
```

Part C: Button Container

Expected output (remember these are centered on the page in a #button-container that spans the width of the page):


```
/* Write your CSS for Part C below */

button {
    text-align: center;
    font-family: Verdana, sans-serif;
    font-size: 14pt;
    background-color: gold;
    color: indigo;
    width: 300px;
    margin-left: 10px;
    padding-top: 10px;
    padding-bottom: 10px;
}

#button-container {
    display: flex;
    justify-content: center;
}
```

3. JS with DOM (15pts): Jump in with JS!

```
"use strict";
(function() {
window.addEventListener("load", init);

// Images of all the CSE 154 instructors
const PICS = ["ann.jpg", "chao.jpg", "conner.jpg", "daniel.jpg", "hawk.jpg", ... ];

// Returns array with [randomTop, randomLeft] values, where each value is an
// integer for a position for a .duckie to fit in the #fountain.
function getDuckiePos() {
  // Details of function not provided here
}

/* BEGIN ONE POSSIBLE SOLUTION */
function init() {
  id("add-duckie").addEventListener("click", addDuckie);
  id("party-mode").addEventListener("click", partyMode);
}

function addDuckie() {
  let duckie = document.createElement("div");
  duckie.classList.add("duckie");
  let coordinates = getDuckiePos();
  let randomX = coordinates[0];
  let randomY = coordinates[1];
  duckie.style.top = randomY + "px";
  duckie.style.left = randomX + "px";
  id("fountain").appendChild(duckie);
}

function partyMode() {
  if (id("party-container").children.length > 0) {
    id("party-container").innerHTML = "";
    id("party-mode").innerText = "~Party On~";
  } else {
    for (let i = 0; i < PICS.length; i++) {
      let instructor = document.createElement("img");
      instructor.src = PICS[i];
      instructor.alt = "Party instructor!";
      instructor.classList.add("party-pic");
      id("party-container").appendChild(instructor);
    }
    id("party-mode").innerText = "Party Off";
  }
}
/* END SOLUTION */
```

4. Node.js Web Service (20pts): Club CSE 154!

Part A:

```
const fs = require("fs").promises;

async function getChars() {
  let files = await fs.readdir("public/data");
  let result = "";
  for (let i = 0; i < files.length; i++) {
    result = result + files[i] + "\n";
  }
  return result;
}
```

Part B:

```
const fs = require("fs").promises;

async function getCharInfo(chardir) {
  let char_info = (await fs.readFile("public/data/" + chardir + "/info.txt",
"utf-8")).split("\n");
  let result = {
    "name": char_info[0],
    "series": char_info[1],
    "description": char_info[2]
  };
  result["appearances"] = (await fs.readFile("public/data/" + chardir + "/appearances.txt",
"utf-8")).split("\n");
  return result;
}
```

Part C:

```
const express = require("express");
const fs = require("fs").promises;
const app = express();

app.get("/club154", async function (req, res) {
  try {
    let mode = req.query.mode;
    if (mode === "all") {
      let chars = await getChars();
      res.type("text").send(chars);
    } else if (mode === "lookup") {
        let name = req.query.char;
        if (!name) {
          res.type("text").send("Please pass in a character name.");
        } else {
          let info = await getCharInfo(name);
          res.json(info);
        }
    } else {
      res.status(400).type("text").send("Please pass in a mode of all or lookup.");
    }
  } catch (err) {
    res.type("text");
    if (err.code === "ENOENT") {
      res.status(400).send("Please pass in a valid character name.");
    } else {
      res.status(500).send("Something went wrong on the server. Please try again later.");
    }
  }
});
```

5. JS with Fetch (15pts): Gotta Fetch ‘em All

```
(function () {
  "use strict";

  const URL = "/club154";

  window.addEventListener("load", init);

  /* BEGIN ONE POSSIBLE SOLUTION */
  function init() {
    fetch(URL + "?mode=all")
      .then(checkStatus)
      .then(populateChars)
      .catch(handleError);
  }

  function populateChars(response) {
    response = response.split("\n");
    for (let i = 0; i < response.length; i++) {
      if (response[i] !== "") {
        let charData = response[i];
        let img = document.createElement("img");
        img.src = "data/" + charData + "/avatar.png";
        img.alt = charData;
        img.id = charData;
        id("playground").appendChild(img);
        img.addEventListener("click", showChar);
      }
    }
  }

  function showChar() {
    let name = this.id;
    fetch(URL + "?mode=lookup&name=" + name)
      .then(checkStatus)
      .then(JSON.parse)
      .then(populateCharInfo)
      .catch(handleError);
    if (qs(".selected")) {
      qs(".selected").classList.remove("selected");
    }
    this.classList.add("selected");
  }

  /* CONTINUES ON NEXT PAGE */
```

```
  function populateCharInfo(response) {
    id("name").innerText = response.name;
    id("series").innerText = response.series;
    id("description").innerText = response.description;
    id("appearances").innerHTML = "";
    for (let i = 0; i < response.appearances.length; i++) {
      let appearance = response.appearances[i];
      let li = document.createElement("li");
      li.innerText = appearance;
      id("appearances").appendChild(li);
    }
    id("spotlight").classList.remove("hidden");
  }

  function handleError() {
    id("error").classList.remove("hidden");
    id("spotlight").classList.add("hidden");
    if (qs(".selected")) {
      qs(".selected").classList.remove("selected");
    }
  }
  /* END SOLUTION */
})();
```

6. SQL and Node.js (12pts): Message in a Bottle

Part A: Basic SQL Queries/Statements

i.

```
SELECT DISTINCT name FROM posts WHERE question LIKE "%my code%" ORDER BY name;
```

ii.

```
INSERT INTO posts (id, name, question, category)
VALUES(541, "Piaz Za Rocks", "Will this be tested?", "other");
```

Or

```
INSERT INTO posts
VALUES(541, "Piaz Za Rocks", "Will this be tested?", "other");
```

iii.

```
DELETE FROM posts WHERE question LIKE "%code does not work%";
```

Part B: Node.js with SQL using mysql

For the following "/`update`" endpoint:

i. Briefly explain why this endpoint is vulnerable to malicious clients.

- It is prone to SQL injection (e.g. DROP TABLE statements) without prepare/execute.

Ii.

```
const db = mysql.createPool(cnctInfoObj);

app.post("/update", async function (req, res) {
  // assume id and newQuestion exist
  let id = req.body.id;
  let newQuestion = req.body.newQuestion;

  let str = "UPDATE posts SET question = ? WHERE id = ?";

  try {
    await db.query(str, [newQuestion, id]);
    res.type("text").send("Updated the database!");
  } catch (err) {
    res.type("text");
    res.status(500).send("Something went wrong on the server. Please try later.");
  }
});
```