

**PRACTICE EXAM 1**  
**ANSWERS**

Tuesday, October 12, 2021  
10:10am – 11:40am (90 minutes)

**Username:** \_\_\_\_\_ **Student ID:** \_\_\_\_\_

**Name:** \_\_\_\_\_

**Honor Pledge:**

“I have neither given nor received unauthorized aid on this examination,  
nor have I concealed any violations of the Honor Code.”

**Signature:** \_\_\_\_\_

---

INSTRUCTIONS:

- The exam is closed book and notes except for one 8.5"x11" sheet of notes (both sides). All electronic device use is prohibited during the exam (calculators, phones, laptops, etc.).
  - Print your name, student ID number, and Username **LEGIBLY**. Sign the Honor Pledge. Your exam will not be graded without your signature.
  - Record your **USERNAME** at the top of each odd-numbered page in the space provided. This will allow us to identify your exam if pages become separated during grading.
  - Please write your answers in the space provided on the exam, and clearly mark your solutions. You may use the backs of the exam pages as scratch paper. Please do not use any additional scratch paper.
  - Partial credit will be awarded for partial solutions. If you leave a non-extra-credit portion of the exam blank, **you will receive one-third of the points for that small portion (rounded down)**. If you wish to make an answer to a sub-question blank, draw an “X” through the portion you wish to be blank. **DO NOT** cross out portions of the exam until you near the end because this mark cannot be undone.
  - Partial credit will be awarded for partial solutions.
  - **DO NOT** detach any pages from this booklet.
  - There should be 10 pages in this packet; if you are missing any pages, please let the instructor know.
-

**1. Web Architecture [11 Points]**

Consider the following output from the **client URL** (cURL) program for an HTTP request to `google.com`.

```
% curl -v google.com
* Trying 142.250.176.206...
* TCP_NODELAY set
* Connected to google.com (142.250.176.206) port 80 (#0)
> GET / HTTP/1.1
> Host: google.com
> User-Agent: curl/7.64.1
> Accept: */*
>
< HTTP/1.1 301 Moved Permanently
< Location: http://www.google.com/
< Content-Type: text/html; charset=UTF-8
< Date: Thu, 07 Oct 2021 00:00:17 GMT
< Expires: Sat, 06 Nov 2021 00:00:17 GMT
< Cache-Control: public, max-age=2592000
< Server: gws
< Content-Length: 219
< X-XSS-Protection: 0
< X-Frame-Options: SAMEORIGIN
<
<HTML><HEAD><meta http-equiv="content-type" content="text/html; charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="http://www.google.com/">here </A>.
</BODY></HTML>
* Connection #0 to host google.com left intact
* Closing connection 0
```

Label the key parts of the request and response. Explain what each part signifies in the space below.

- `>` indicates part of the HTTP request
- `<` indicates part of the HTTP response
- `GET / HTTP/1.1`: this is a GET request using HTTP version 1.1
- `HOST: google.com`: this is a request for something from google.com
- `User-Agent`: information about who/what is making the request
- The response has a status code of 301 meaning this is a redirect error (permanent)
- The correct location to request is `http://www.google.com`
- The format of the response body is `text/html` with a UTF-8 character set
- The body of the response is an HTML page with information about the correct place to go if the browser does not automatically redirect

**2. Web Applications** [16 Points]

A) (8 points) List four properties or characteristics often exhibited by modern web applications.

1. Web browser is the user interface
2. Perform actions and manipulate data without screen refreshes
3. Interactive (prompt response to interactions)
4. Stores data for the user
5. Makes asynchronous calls to the web server
6. May be available even when not on internet

B) (8 points) Identify a currently popular web application and describe how the properties you listed in the previous part apply to this application.

**Web Application:** Google Docs \_\_\_\_\_

1. The editor is visited via the website docs.google.com. There is no desktop app version.
2. One can type text and manipulate the document using the menus without the page ever refreshing.
3. There are no pauses as text is edited (either as one types or when highlighting text and changing formatting)
4. The contents of documents is stored on Google servers and can be accessed from any computer.

**3. HTML5 [13 Points]**

For each of the following tags, describe the type of information typically contained within.

Taken from <https://developer.mozilla.org>:

- A) `<article>` The `<article>` HTML element represents a self-contained composition in a document, page, application, or site, which is intended to be independently distributable or reusable (e.g., in syndication).
- B) `<aside>` The `<aside>` HTML element represents a portion of a document whose content is only indirectly related to the document's main content. Asides are frequently presented as sidebars or call-out boxes.
- C) `<div>` The `<div>` HTML element is the generic container for flow content. It has no effect on the content or layout until styled in some way using CSS (e.g. styling is directly applied to it, or some kind of layout model like Flexbox is applied to its parent element).
- D) `<figure>` The `<figure>` HTML element represents self-contained content, potentially with an optional caption, which is specified using the `<figcaption>` element.
- E) `<footer>` A `<footer>` typically contains information about the author of the section, copyright data or links to related documents.
- F) `<header>` The `<header>` HTML element represents introductory content, typically a group of introductory or navigational aids.
- G) `<li>` The `<li>` HTML element is used to represent an item in a list. It must be contained in a parent element: an ordered list (`<ol>`), an unordered list (`<ul>`), or a menu (`<menu>`).
- H) `<nav>` The `<nav>` HTML element represents a section of a page whose purpose is to provide navigation links, either within the current document or to other documents.
- I) `<ol>` The `<ol>` HTML element represents an ordered list of items — typically rendered as a numbered list.
- J) `<p>` The `<p>` HTML element represents a paragraph. Paragraphs are usually represented in visual media as blocks of text separated from adjacent blocks by blank lines and/or first-line indentation, but HTML paragraphs can be any structural grouping of related content, such as images or form fields.
- K) `<span>` The `<span>` HTML element is a generic inline container for phrasing content, which does not inherently represent anything.
- L) `<strong>` The `<strong>` HTML element indicates that its contents have strong importance, seriousness, or urgency. Browsers typically render the contents in bold type.
- M) `<ul>` The `<ul>` HTML element represents an unordered list of items, typically rendered as a bulleted list.

**4. Accessibility [5 Points]**

Explain the purpose of `aria` attributes in HTML tags. Give one concrete example and describe its impact on the web browser.

Taken from <https://developer.mozilla.org>:

Accessible Rich Internet Applications (ARIA) is a set of attributes that define ways to make web content and web applications (especially those developed with JavaScript) more accessible to people with disabilities. It supplements HTML so that interactions and widgets commonly used in applications can be passed to assistive technologies when there is not otherwise a mechanism.

As one example, `aria-labelledby` specifies a DOM element that provides information about the current element.

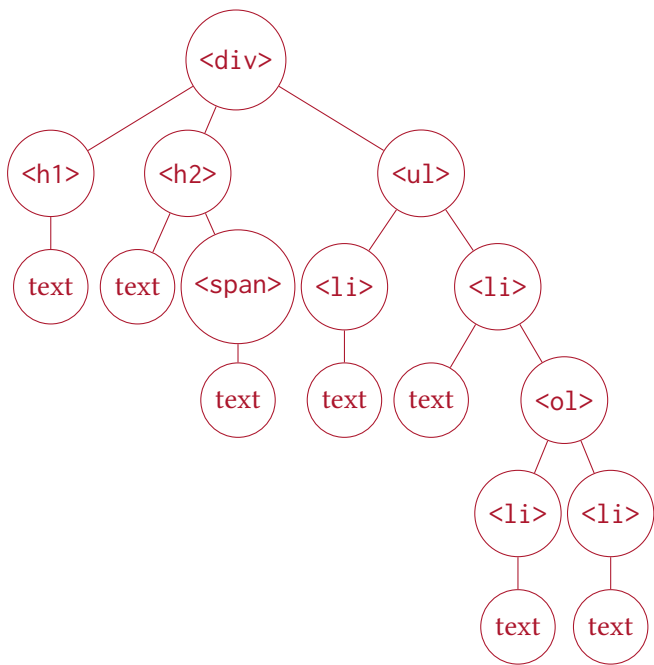
**5. The DOM and Cascading Style Sheets [10 Points]**

A) (8 points) Draw the Document Object Model (DOM) that would be generated from the following snippet of HTML. Include all opening tags from the code snippet in your drawing. You may omit the text contained within the HTML elements.

```

1 <div>
2   <h1>CS 332</h1>
3   <h2>Topics <span>in this class</span></h2>
4   <ul>
5     <li>Web Architecture</li>
6     <li>
7       Content Display and Styling
8       <ol>
9         <li>HTML5</li>
10        <li>CSS</li>
11      </ol>
12    </li>
13  </ul>
14 </div>

```



B) (3 points) Write a CSS rule that changes the color of the list item containing HTML5 (and only HTML5) in the document above to **red**.

```
div ul li ol li:first-child { color: red; }
```

**6. Positioning** [10 Points]

A) ( 5 points) Explain how to position DOM elements next to each other using Flexbox layout. How can you make this layout *responsive*?

First, there must be a parent, container node for the DOM elements that should be positioned horizontally. This parent node should have the `display: flex;` CSS property set on it. While it is the default, it is also possible to explicitly specify the direction of child elements, which is `flex-direction: row;` in this case.

To make this responsive, there are `flex-grow` and `flex-shrink` properties that can be set to allow the children to grow/shrink accordingly. This often needs to be combined with `flex-basis` to properly size the children.

B) (5 points) Describe two similarities and two differences between *relative* and *absolute* positioning.

Similarities:

- 1) Both create coordinate systems
- 2) Both allow for the element to be shifted from its original position

Differences:

- 1) Relative positioning moves the item from normal layout positions whereas absolute positioning completely remove the element from normal layout.
- 2) In relative positioning, the space of the original element is retained. Absolute positioning closes up this space

**7. Higher-Order Functions** [20 Points]

- A) (10 points) Write a function, `multiplyEvens`, in JavaScript that takes in an array of numbers as an argument and returns the product of the elements whose values are divisible by two. For example, `multiplyEvens([1,2,3,4])` would return 8. You may use any built-in functions you wish, but this is solvable using only the function provided on the last page of this exam.

```
function multiplyEvens(arrayA) {  
  
    return arrayA.filter( (e) => e % 2 == 0 ).reduce( (acc, e) => {  
        return acc * e;  
    }, 1);  
  
}
```

- B) (10 points) Write a function, `insertListItems`, in JavaScript that takes in a DOM element, and an array of strings and inserts each of these strings as a list item into the provided DOM element. For example, `insertListItems(document.querySelector("#foo"), ["a", "b", "c"])` would insert three list items (i.e., `<li>` elements) into the DOM element with an ID of "foo".

```
function insertListItems(parent, list) {  
  
    list.forEach( (val) => {  
        const node = document.createElement("li");  
        node.innerText = val;  
        parent.appendChild(node);  
    });  
  
}
```

**8. JavaScript** [15 Points]

A) (5 points) What is an anonymous function? What are two different ways to create an anonymous function in JavaScript?

An anonymous function is a function value that is never bound to a variable. They are created as a function expression directly where it is used, passed into a function, or returned. An anonymous function can either be created using the function keyword or using the `() => {}` arrow notation.

B) (5 points) Write a JavaScript object literal named `todo` that contains a description of “Practice for the exam,” a due date of “2021-10-12,” and a priority of “high.” Store the date as a `Date` object.

```
1   const todo = {
2     description: "Practice for the exam",
3     due: new Date("2021-10-12"),
4     priority: "high"
5   };
```

C) (5 points) Consider the following constructor function:

```
1   const Todo = function(description, dueDate, priority) {
2     this.desc = description;
3     this.due = dueDate;
4     this.priority = priority;
5   };
```

Write a `toString` method that will apply to all `Todo` objects and returns a nicely formatted string of the `todo` item.

```
1   Todo.prototype.toString = function() {
2     return `Todo: (${this.priority} priority) ${this.description}, ` +
3     `Due: ${this.due}`;
4   };
```



## JavaScript Reference

These methods are correct and function as specified by the Mozilla JavaScript reference (<https://developer.mozilla.org/>). Parameters specified with [] are optional.

```
1  /*
2  * The map() method creates a new array with the results of calling a provided
3  * function on every element in this array.
4  *
5  * callback is invoked with three arguments: the value of the element, the index
6  * of the element, and the Array object being traversed.
7  *
8  * If a thisArg parameter is provided to map, it will be passed to callback when
9  * invoked, for use as its this value.
10 */
11 arr.map(callback[, thisArg])
12
13 /*
14 * The filter() method creates a new array with all elements that pass the test
15 * implemented by the provided function.
16 *
17 * callback is invoked with three arguments: the value of the element, the index
18 * of the element, and the Array object being traversed.
19 *
20 * If a thisArg parameter is provided to map, it will be passed to callback when
21 * invoked, for use as its this value.
22 */
23 arr.filter(callback[, thisArg])
24
25 /*
26 * The reduce() method applies a function against an accumulator and each value
27 * of the array (from left-to-right) to reduce it to a single value.
28 *
29 * reduce executes the callback function once for each element present in the
30 * array, excluding holes in the array, receiving four arguments: previousValue,
31 * currentValue, currentIndex, and array
32 *
33 * The first time the callback is called, previousValue and currentValue can be
34 * one of two values. If initialValue is provided in the call to reduce, then
35 * previousValue will be equal to initialValue and currentValue will be equal to
36 * the first value in the array. If no initialValue was provided, then
37 * previousValue will be equal to the first value in the array and currentValue
38 * will be equal to the second.
39 *
40 * Note: If initialValue isn't provided, reduce will execute the callback
41 * function starting at index 1, skipping the first index. If initialValue is
42 * provided, it will start at index 0.
43 */
44 arr.reduce(callback[, initialValue])
45
46 /*
47 * The indexOf() method returns the first index at which a given element can be
48 * found in the array, or -1 if it is not present.
49 *
50 * indexOf() compares searchElement to elements of the Array using strict
51 * equality (the same method used by the === or triple-equals operator).
52 */
53 arr.indexOf(searchElement[, fromIndex = 0])
```

This page is intentionally left blank.  
You may use this page as working space.