CS 142 Midterm Examination

Winter Quarter 2023

You have 1.5 hours (90 minutes) for this examination; the number of points for each question indicates roughly how many minutes you should spend on that question. Make sure you print your name and sign the Honor Code below. During the examination you may consult two double-sided pages of notes; all other sources of information, including laptops, cell phones, etc. are prohibited.

I acknowledge and accept the Stanford University Honor Code. I have neither given nor received aid in answering the questions on this examination.

____
(Signature)

____
(Print your name, legibly!)

____@stanford.edu
(Stanford email account for grading database key)

Problem #1 (6 points)

The word "hyper" in Hypertext Markup Language (HTML) gets its name from the ability to embed entities where the user's mouse clicks generate actions. For example, the HTML "`a`" tag renders text so a different document is viewed when the user clicks on the text.
(e.g. `<a href="clickhappened">click me</a>`)

With event handling, arbitrary JavaScript code including displaying a different document can be bound to a mouse action so any HTML tag can be "hyper". For example, a user clicks on a "`b`" tag or "`p`" tag can have a mouse click switch to a different view:
(e.g. `<p onclick="do_clickhappened()">click me</p>`)

Although with JavaScript frameworks like React.js, we could generate HTML that never used the "a" tag but instead used event handling to switch the HTML document being rendered by the browser. Explain why a developer using React.js might prefer to use the old HTML “a” tag and keep the browser involved in the view switching rather than doing the view switching solely in JavaScript.

Problem #2 (8 points)

You are given the following HTML document that contains a placeholder (**XXX**) that can be set to any string:

```
<html>
   <head>
      <style type="text/css">
      p { position: XXX; top: 0; left: 0; }
      </style>
   </head>
   <body>
      <p>Paragraph 1</p>
      <p>Paragraph 2</p>
      <p>Paragraph 3</p>
    </body>
</html>
```

Describe how the HTML document would look when rendered with XXX set to the following values:

A. "static"
B. "absolute"
C. "relative"
D. "fixed"

Problem #3 (8 points)

Assume we have a React.js web application that was served from the URL `http://host/a.html` and contains the following generated hyperlinks:

```
  <a href="#/user/list">Show users</a>
  <a href="#full">Show full list</a>
```

Also, assume JavaScript click handling in the React.js runtime has broken so only the normal browser handling of "a" tags occurs. Assume the user clicked twice on the initial view. The first click was on "Show users" and they noticed no view changes (recall the JavaScript is broken). The second click was on "Show full list".

A. Describe what the browser processing would have been triggered by the first click (think about the processing the browser does on the click).
B. Show what the resulting location URL would be after the first click.
C. Show what the resulting location URL would be after the second click.

Problem #4 (7 points)

Consider the following ECMAScript code:

```
class Foo {
  methodA() { }
  static staticA() { }
}
let f = new Foo();
```

List all the JavaScript objects created by the code. For each object you list:

- Provide a JavaScript expression that accesses the object.
- State what properties (if any) the object contains.

Problem #5 (6 points)

In real-world math: `1 + 2 = 3,`

so by dividing both sides by `n`, we get that

`1/n + 2/n = 3/n`

will be true for all `n` != 0.

In JavaScript, it is true for many, but not all, values of `n`. Looking at the first 10 non-zero integers, the expression `1/n + 2/n == 3/n` evaluates to `false` for 20% of the numbers.

A. Explain why JavaScript arithmetic only gets 80% correct here.
B. If we changed the expression to `1/n + 2/n === 3/n` would there be any improvement in the result? Explain your answer.

Problem #6 (6 points)

Early JavaScript code conventionally used a variable name `self` when coding object-oriented programming methods containing callback functions. Explain how the introduction of shorthand for defining functions pushed by the advocates of functional programming (i.e. arrow function expressions) caused object-oriented programing developers to use the variable `self` less?

Problem #7 (6 points)

Assume you have the following HTML document:

```
<html id="id0">
   <body id="id1">
       <div id="id2">
          Div 1
          <div id="id3">
              Div 2
         </div>
       </div>
   </body>
</html>
```

Assume you looked up the DOM nodes by ID (i.e. `getElementById`) and recorded the IDs of the `offsetParent` and `parentNode`. Fill in the following table:

	`offsetParent.id`	`parentNode.id`
`id1`	body `offsetParent` is NULL by spec	
`id2`		
`id3`		

Problem #8 (8 points)

Assume you have the following HTML document (it's the same as Problem #7):

```
<html id="id0">
   <body id="id1">
       <div id="id2">
          Div 1
          <div id="id3">
              Div 2
         </div>
       </div>
   </body>
</html>
```

Assume you registered the same event handler on DOM nodes with IDs of `id0, id1, id2, id3` for both the capture and bubble phases (i.e., a total of 8 calls to `addEventListener)` You click on the line "Div 2" in the document. The event handler prints out the following:

1. Event phase ("bubble" or "capture")
2. `event.target.id`
3. `event.currentTarget.id`

In the table below, show what handler call prints you would expect to see. (Leave row blank if there are fewer calls than rows)

	Phase	`event.target.id`	`event.currentTarget.id`
1			
2			
3			
4			
5			
6			
7			
8			

Problem #9 (6 points)

HTML templates are commonly used for view generation in modern web application frameworks. These frameworks allow arbitrary complex expressions including function definitions and many line expressions to be embedded inside templates but good programming practices argue against long expressions. Explain what advantages of HTML templates are hurt by using long expressions.

Problem #10 (8 points)

React.js permits you to directly read component state (e.g. `this.state.foobar`) but strongly discourages you from directly writing to component state (e.g. `this.state.foobar = newvalue`) once a component has been created.

A. Explain the rationale behind this asymmetry.
B. What misbehavior would you expect to happen if you ignored this suggestion? Use the model, view, controller (MVC) pattern to describe the problem.

Problem #11 (9 points)

In functional programming, it is typically required that a function's return value only depend on the input parameters being passed to the function and that the function doesn't change anything in the global state (i.e. no side-effects). When building a React.js component, we can create some state by using a call to `useState` from React Hooks like so:

```
  let [val, setVal] = useState("init value");
```

A. React.js's `useState` can sometimes violate the functional programming requirement described above. Describe what would cause `useState` to return a different value for the same input parameter.
B. `useState` returns a JavaScript array. JavaScript supports arrays that are sparse and polymorphic. State if `useState` is using each of the following capabilities of JavaScript arrays. Briefly explain your answer showing your knowledge of the capability.
    a. sparse arrays
    b. polymorphic arrays

Problem #12 (6 points)

Many modern web applications will dynamically update themselves if you change the size of the browser window they are running in. If you start the web application in a large window and slowly make the window smaller and smaller you notice the web application shrinks the amount of white space until spacing gets tight and then changes so less content is being displayed. It repeats this pattern of white space compression and content removal as the window gets smaller.

Describe the underlying browser mechanisms that make this white space compression and content removal behavior possible.

Problem #13 (6 points)

Would you expect to get better test coverage when using all end-2-end tests or all unit tests? Explain your answer.