CS 142 Midterm Examination

Winter Quarter 2022

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 (5 points)

The HTML language standards group has added tags like `<article>`, `<section>`, `<footer>`, and `<summary>` that could have easily been implemented in older HTML using a `div` tag with a `class` attribute specifying the formatting that should be done for that type of document section.

Explain how adding these new tags to HTML makes sense (i.e. consistent with both the philosophy and practical usage of HTML) even though the formatting they direct can be done relatively easily with existing `div` tags.

Problem #2 (5 points)

When a web browser gets an HTTP response containing a text document rather than an HTML document, the browser switches off the HTML processing engine and shows the text document verbatim. An alternative approach would have been to assume the document is the body of an HTML document that is missing the `html` and `body` tags.

Explain what would happen if the browser treated the text document as the body of an HTML document and renders the text document using this approach. Describe what you expect the browser to display if given a commonly used text document such as a text-only email or a C++ header file. List any differences from what is shown using the verbatim approach.

Problem #3 (6 points)

Some CSS properties are inherited (e.g. `font-size`) and some properties are not inherited (e.g. `border`). For the inherited properties, does a DOM Node inherit from the `parentNode` or `offsetParent`? Explain your answer.

Problem #4 (6 points)

Assume you are given a deeply nested HTML document with no CSS styling and a global JavaScript variable named `element` that points to a DOM leaf node somewhere in the middle of the document many tree levels down from the root node.

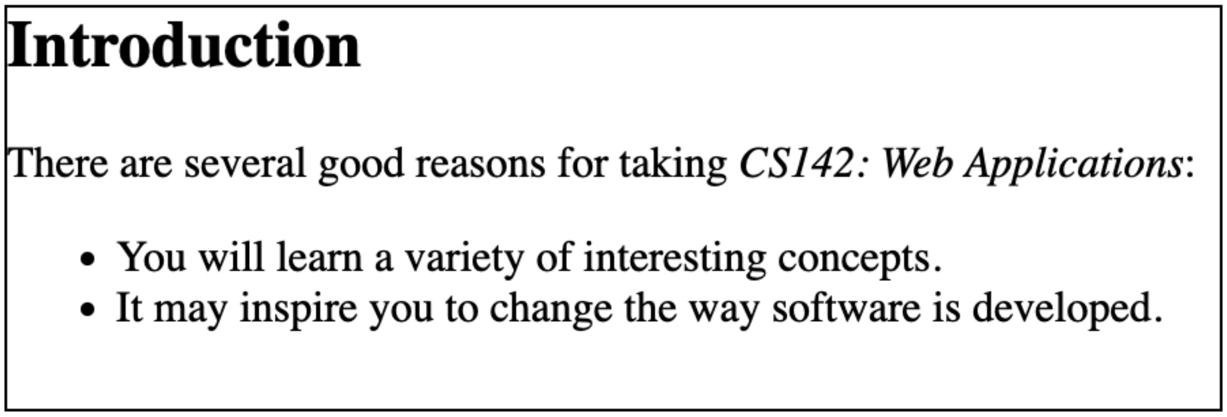
Show JavaScript code that changes `element.offsetParent` without explicitly assigning to `element.offsetParent`.

Problem #5 (8 points)

Assume you are given the following simple HTML document that has some styling on the body tag.

```
<html>
  <body style="border: 1px black solid;">
    <h2>Introduction</h2>
      <p>
         There are several good reasons for taking
         <i>CS142: Web Applications</i>:
      </p>
      <ul>
        <li>You will learn a variety of interesting concepts.</li>
        <li>It may inspire you to change the way software is developed.</li>
      </ul>
  </body>
</html>
```

It renders in a properly working browser to look like:



Introduction

There are several good reasons for taking *CS142: Web Applications*:

- You will learn a variety of interesting concepts.
- It may inspire you to change the way software is developed.

Assume you are given a browser that has a broken CSS implementation so the "border" properties switch to being inherited by all node elements (not #text nodes). Show your understanding of the implication of this by drawing the additional bounding boxes (if any) that would appear on the rendered view above.

Problem #6 (6 points)

When building traditional web pages (i.e. static HTML documents), the use of element styling (`style=`attributes) is discouraged in favor of using CSS style sheets for styling. When programming in React.js with JSX, element styling (style=attributes) is not discouraged. Explain the problem with using “`style=`attributes” in static HTML documents that is not present in JSX usage.

Problem #7 (8 points)

Below we list pairs containing an HTML hyperlink and a URL that resulted from a click on the hyperlink (i.e. resultant URL).

```
(a)   <a href="/a/b/c.html">Click</a> http://localhost:999/a/b/c.html
(b)   <a href="a/b/c.html">Click</a>  http://localhost:999/a/b/c.html
(c)   <a href="a/b/c.html">Click</a>  http://localhost:999/z/a/b/c.html
(d)   <a href="#foo">Click</a>        http://localhost:999/a/b/c.html#foo
```

None of the URLs in the hyperlinks are full URLs meaning the browser's current location URL of the page containing the hyperlink is used in determining the resultant URL. For each of the pairs (a)–(d), describe what the browser's current location URL must have been to have the given resultant URL. Note many possible current locations could cause the listed resultant URL. Your answer should describe the possible values for all the parts of the current URL (i.e. scheme, hostname, port number, hierarchical portion, query parameters, fragment).

(a)

(b)

(c)

(d)

Problem #8 (7 points)

JavaScript ES6 extensions added some familiar syntax to define classes but continued to use the object-oriented programming conventions long-used in JavaScript. The below syntax defines a JavaScript class (`Foo`) that has a member property (`member_prop`) and a member method (`member_method`). Code allocates an instance of the class Foo into the variable `f`.

```
class Foo {
  static static_prop = 1;
  constructor() {
     this.member_prop = 2;
  }
  member_method() { console.log("member method call"); }
}

let f = new Foo();
```

(a) Show the JavaScript expression that would add one to `member_prop`.

(b) Show the JavaScript expression that would add two to `static_prop`.

(c) By maintaining compatibility with the old object-oriented programming conventions, these classes exhibit some weird behavior. For example, although we can call the `member_method` function by the expression `f.member_method()`, we can also smash the `member_method` by replacing it with a number:

```
f.member_method = 32;
```

After the assignment `f.member_method` is now the number 32 rather than a function. Although this is consistent with the dynamic typing of JavaScript, executing:

```
delete f.member_method;
```

cause it to go back to being the original method function rather than `undefined`. Explain this type of weird behavior.

Problem #9 (6 points)

When looking at a DOM node other than the root of the tree, there are two pointers that point at nodes further up the tree towards the root (`parentNode` and `offsetParent`), following either of these pointers up the tree will eventually get to the `body` tag node. Considering these two paths up the tree, answer the following questions:

(a) Are the nodes in the parentNode path always in the offsetParent path? Explain your answer.

(b) Are the nodes in the offsetParent path always in the parentNode path? Explain your answer.

Problem #10 (9 points)

Consider the following HTML document:

```
<html>
 <body>
  <div id="one">
   One
   <div id="two">
    Two
    <div id="three">
     Three
     <div id="four">
      Four
     </div>
    </div>
   </div>
  </div>
 </body>
</html>
```

We run the following JavaScript on the document:

```
const divs = window.document.getElementsByTagName('div');
for (let i = 0; i < divs.length; i++) {
   divs[i].addEventListener("click", (e) =>
     console.log('bubble',
                 e.currentTarget.id,
                 e.currentTarget == e.target,
                 e.currentTarget.textContent.replace(/[\s]/g, '')),
     false);
   divs[i].addEventListener("click", (e) =>
     console.log('capture',
                 e.currentTarget.id,
                 e.currentTarget == e.target,
                 e.currentTarget.textContent.replace(/[\s]/g, '')),
     true);
}
```

Hint: the third parameter in `addEventListener` is named `useCapture`.

problem continued on next page…

Problem #10 continued

Describe what the output would be if the user were to click on the word "Three" in the document.

Problem #11 (6 points)

AngularJS showed the usefulness of binding JavaScript values in controllers to expressions in an HTML template. A programmer could simply change the value of a JavaScript variable and AngularJS would detect and re-render the component with the updated template expressions.

Although this binding was a hit with programmers, it required AngularJS to compare all the expressions in the templates to see if some state had changed every time some JavaScript code ran. For views with much state, this checking for changed JavaScript state variables got too expensive to be useful.

Explain the mechanism ReactJS used to be able to employ a similar binding of JavaScript variables to HTML template expressions yet didn't suffer large overheads when only small amounts of the state changed.

Problem #12 (6 points)

Explain how a ReactJS web application can appear to behave like an old-style web application (where the user clicked on hyperlinks to navigate between view "pages" and uses HTML forms to input data), yet be considered a single page application.

Problem #13 (6 points)

If you resize a modern web app in a window, it is not unusual to see it just shrink to fit in the smaller window but if you make the window small enough it will re-layout the app to better work in the small window. Explain the browser's mechanism that web app used to have this behavior.

Problem #14 (6 points)

Internationalization (I18N) and Accessibility (ARIA) are two important properties of a web application that frequently don't make it into the first release of a web application. When adding features to a web application, the work can be considered independent if the changes required don't interact with each other and features can be added by teams working concurrently. State whether Internationalization (I18N) and Accessibility (ARIA) are dependent or independent from each other, and explain.