CS 142 Midterm Examination

Spring 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 (10 points)

HTML is a markup language designed to support transferring documents around the world wide web using the Internet in its youth. Some Internet connections in those early days operated at speeds in the range of 30 to 40 bytes per second (i.e. 300 baud modem). A makeup language works by extending the document's content with annotating tags so the size of an HTML document is the size of the content plus the size of the tags. To reduce the size increase, HTML made common tags (e.g. `<p>` and `<a>`) as small as possible.

A. Given the original document's text content size and the size of all the HTML tags, we can compute an estimate of the total size of an HTML encoding of the document. Given your knowledge of HTML, how far off might this estimate be for the HTML encoding of an arbitrary document. Be sure to explain your answer.

   Although a markup language works by annotating the text with tags it must deal with the problem of text that looks like annotations. Some "escape" mechanism is needed. HTML chose to replace the symbols that are used in annotations with a 4 character sequence (e.g. "`<`" becomes "`&lt;`") so in the worse case the estimate could be off by a factor of 4 if every character needed to be represented by the escape sequence.

B. With the introduction of CSS, CSS style sheet bytes needed to be transferred over the Internet along with the HTML document bytes in order for the document to be displayed correctly. Like HTML tags, CSS stylesheets are relatively space-efficient. Would a highly styled HTML document (e.g. a document that overrides the browser's default styling) have a larger total transfer size including the CSS stylesheets when compared to doing the styling without CSS style sheets? Explain your answer.

   Yes, using CSS eliminates the need to repeatedly set styling attributes on the html elements. CSS mantra is "Don't repeat yourself (DRY)". So, the DRY principle in CSS significantly is expected to reduce the transfer size of highly styled documents.

Problem #2 (5 points)

Although the original HTML design was based on previous markup languages that were used to describe the formatting of documents, the HTML standard has moved away from using HTML for that role. Describe what the role of HTML is today if it isn't specifying the document's appearance.

HTML's role is to identify parts of a document, not specify the styling. Styling should be left to CSS.

Problem #3 (5 points)

Using CSS styling, a DOM element and its children can be marked invisible using two different CSS properties:

- Setting `visibility: hidden;`
- Setting `display: none;`

Both of these operations are reversible (e.g `visibility: visible;` or `display: block;`) so by periodically updating the DOM element we can cause part of an HTML to disappear and then reappear giving the user the illusion that the element is blinking. Which of the two approaches do you think would be more common to making something blink in a web application? Justify your answer.

Setting visibility property to hidden/visible would be better to make something blink. Although both will make the element disappear, setting the display to none will cause the element to be removed from the view and the HTML will be re-laid out without the element. For example, the page layout could move around with the elements below the hidden element moving into its space. With the visibility setting the element would not display but continue to occupy space in the layout so it would disappear and reappear without the page layout changing.

Problem #4 (8 points)

Given the following HTML document body:

```
<body class="b">
  <div class="d1">
    Div1
    <div class="d2">
        Div2
        <div class="d3">
         Div3
         <p class="p">Paragraph</p>
        </div>
    </div>
 </div>
</body>
```

A. If you had a CSS rule that looked like:
`.p { position: absolute; top 2px; left 2px;}`
State the class of the element that the paragraph element is positioned relative to. Explain your answer.

Class = "b". Looking up the DOM tree from the paragraph element, none of its ancestors are positioned so the position will be relative the top most element: the body tag with class="b".

B. Assume you wanted the paragraph to be positioned relative to the div with `class="d2"`. Show a CSS rule that would make that happen. Include an explanation of why it works.

```
.d2 {position: absolute;}
```

This rule turns this `div` with class="d2" to be the nearest positioned ancestor for the paragraph.

Problem #5 (8 points)

Part A.
In project 4, we end up starting the web application by going to the location http://localhost:3000/p5.html but used React to generate the view. The view that React generated has hyperlinks that look like `<a href="#/states">States</a>`. Identify the following components of the URL the browser goes to when that hyperlink is clicked.

Scheme: `http`

Host name: `localhost`

Port number: `3000`

Hierarchical portion: `p5.html`

Query parameters:

Fragment: `/states`

Part B.
Describe the processing done by the browser when it is given the above URL. Note this is asking what the *browser* does, not what the JavaScript environment does.

When given this URL, the browser notes that it is already on the URL page (http://localhost:3000/p5.html) so no fetch (GET request) is done. It does use the fragment part of the URL and does a lookup in the HTML document of a matching anchor tag (i.e. `<a name="/states">`). The browser doesn't find that anchor tag so doesn't change the scrolling of the window.

Although we didn't ask about this: If the JavaScript is running the React Router code, it is watching the URL. Using HashRouter it will examine the fragment part of the URL and see if there is a match in one of the React Router Route components. That could cause the rendered view of the States component.

Problem #6 (8 points)

A. When declaring a variable in JavaScript, the type of the variable is not specified. This is what the programming language experts call an *untyped* language. Although it is an untyped language, all JavaScript variables do have a type. How can the programmer know the type of a variable since it is not specified as part of the declaration?

JavaScript variables take on the type of the last value assigned to the variable. By determining the type of the last assignment to the variable a programmer can know it types. Note that the JavaScript typeof keyword would require adding code to run and returns the primitive JavaScript type (e.g. Arrays, Dates, RegEx are all objects).

B. JavaScript has some weirdnesses in it. If you take the number 9007199254740991 and add 1 to it in JavaScript, you get 9007199254740992. But if you add 2 to it, you will also get 9007199254740992. This is unlike most other programming languages and calculators where you get 9007199254740993. Explain why JavaScript can't add 2 correctly here.

JavaScript represents the number type using a double precision float point format commonly available on CPUs. The format using a scheme with a 52 bit fraction, 11 bit exponent, and 1 bit sign packed into a 64 bit word. Integers up to 2^53 = 9007199254740992 can be stored exactly but if you add 1 it will round it to the nearest value and result in 9007199254740992 again.

Note that although 9007199254740992 is the biggest "safe" integer is isn't the biggest number or biggest integer. `Number.MAX_VALUE` is `1.7976931348623157e+308`. Assume the problem is like a weird 53bit integer that doesn't wrap is incorrect. For example, 9007199254740991+3 rounds to 9007199254740994 which is correct.

Problem #7 (8 points)

Part A.
Some React components that have an event handler method named `clickHander` contain a line in the component's constructor that looks like:

```
this.clickHandler = this.clickHandler.bind(this);
```

This statement is being executed as part of an object allocation in which multiple objects are accessed or created. The objects include the newly allocated **component instance object** and the **component class prototype object.** List the sequence of object accesses done to these two objects when the above JavaScript statement is executed. Consider object reads that occur but return undefined. It could take less than 5 steps so leave any extra numbers blank.

1. **Read instance** trying to find 'clickHandler' returns undefined.
2. **Read prototype** to find 'clickHandler' returns the function. Creates a new function .bind.
3. **Write instance** adds 'clickHandler' to the instance pointing to the new function.
4.
5.

You can use the following words in bold in your steps:

- **Read instance** - read the 'clickHandler' property from the component instance object.
- **Read prototype** - read the 'clickHandler' property from the component class prototype object.
- **Write prototype** - write or create the 'clickHandler' property in the component class prototype object.
- **Write instance** - write or create the 'clickHandler' property in the component instance object.

Part B.
Describe the problem that led JavaScript programmers to start method functions by creating an alias name for the `this` variable such as: `let self = this;`

Javascript has a problem with method functions that define and use callback functions. They problem is the keyword `this` is available anywhere in the method function but its value inside the callback functions will be set by the code calling the callback function rather than be the correct `this` of the method function. By making a variable named `self` be a reference to `this` the method function can access the instance using the name `self` and have the code work everywhere in the method function including the callbacks.

Problem #8 (6 points)

Considering using the JavaScript class keyword like:

```
class Foo {
   methodA() { console.log("methodA"); }
   static methodB() { console.log("methodB"); }
};
let f = new Foo();
```

When executing the above code the JavaScript runtime ends up creating three functions (`Foo`, `methodA`, and `methodB`). The function `Foo` is directly accessible in the current scope (i.e. `typeof Foo == 'function'`) but the `methodA` and `methodB` references are stored in different objects associated with the class. Describe what object each of the methods is stored in and show a JavaScript expression that would call the function.

A. methodA:

methodA is a class method, so it is stored in the prototype object of the class in JavaScript.. That prototype object is accessible with the expression `Foo.prototype`. The JavaScript expression to call this method is `f.methodA()`.

B. methodB:

methodB is a static method, so it is stored in the class as a property that can be accessed as `Foo`. We can call this method by `Foo.methodB()`.

Problem #9 (6 points)

Assume you are given a HTML document with the following body:

```
<body class="b">
   <div class="d1">
      Div1
      <div class="d2">
         Div2
         <div class="d3">
            Div3
            <p class="p">Paragraph</p>
         </div>
      </div>
   </div>
</body>
```

A. How deep would the DOM tree be for the above HTML document? Justify your answer by listing the class names starting at the root and going to class p.

5 levels deep with class names: b -> d1 -> d2 -> d3 -> p
6 levels deep with class names: b -> d1 -> d2 -> d3 -> p -> textNode (no class name)

B. Describe a change you could make to the HTML document that would extend the depth of the DOM tree by two levels?

Change " Paragraph " to " Paragraph text "

Problem #10 (6 points)

A. Can a capture phases event handler prevent a bubble phases event handler on the same event from running when the event is raised? If so, explain how. If not, explain why not.

Yes. Use `stopPropagation` method function call on the event object passed to the handler.

B. Can a bubble phases event handler prevent a capture phases event handler on the same event from running when the event is raised? If so, explain how. If not, explain why not.

No. Because the capture phase happens before the bubble phase. When the event handler registered on the bubble phase gets called, the capture phase has already completed and all the capture phase event handlers have already been called.

Problem #11 (5 points)

One of the challenges of web applications is they need to work in the browser that is provided by the user. Even though web standards for HTML and JavaScript have moved forward rapidly, web applications can not assume the user's browser has been updated to track these standards. In fact, it is not uncommon for browsers to be years behind the standards.

In the first generation of web applications using static HTML documents, browser compatibility hampered the ability to use new features of HTML. React.js has eagerly embraced using new JavaScript features. Explain why React.js can get away with doing this.

The standard React toolchains use Babel or a similar transpiler to convert the new features in the JavaScript to do something equivalent using only old JavaScript features. Because of this, using new features doesn't constrain what browsers the React code can run in.

Problem #12 (5 points)

In React we can register event handlers on 'click' events and use that mechanism to support switching views. When the user clicks on something the event handler can cause the new view to be rendered. When using React to build single applications we don't use this approach. Describe the approach we use and what advantages it has over using event handlers for view switching as described above. List at least two advantages.

We will encode information in the URL and replace the buttons with event listeners with deep linking. Using this way, we can maintain a history of visited URLs, refresh the page without losing any data, share the URL with others to see the same view, pass in parameters in the URL to render the corresponding view.

Problem #13 (5 points)

Explain how the CSS grid system (display: grid;) helps in responsive web design.

CSS grid system allows you to add relative measures which automatically adjusts to different screen sizes. For example, one can specify the width to be 10% of the screen rather than say, 10px. In addition, it allows you to pack elements into rows or columns to be adjusted relative to the screen size all together.

Problem #14 (5 points)

HTML templates have long been used to specify views into web applications. Developers like that the templates make it easy to see the HTML structure of the view and where and how the dynamic data fits in. Explain how support for internationalization interferes with some of these benefits of templates.

By adding support for internationalization, you lose the advantage of having the HTML template match exactly what will be seen on the screen. This is especially true for some languages that are right-to-left (e.g. Arabic) or with longer words (e.g. German), which can possibly change how the HTML will be rendered in the window (due to text overflow, spacing, alignment, etc.).