Web Science
Quiz 1: March 1, 2022

Enter your answers directly into this document (with the exception of #2 and #3).
All answers should be In Your Own Words, using complete sentences with proper spelling and grammar.

Save this document as: answers.docx (or .odf or .pdf) (-5 if wrong name). For all questions other than #2 and #3, you will not receive any credit for answers not placed in this document.

When finished with the quiz, put everything you wrote (this document, all code, etc.) on GitHub into a branch in your lab repo named: quiz1 (-5 if submitted incorrectly). **Do not submit your node_modules folder! (-15 if you submitted the node_modules folder)**

1. **Short answers** (25 points): (Answer in complete sentences, explain your answers)

    a. (5) How can I determine the type of device that my page is being displayed on? Give two examples of why I might care.

While it's easy to grab the user agent to find out the type of the device the page is being displayed on, this isn't recommended because the information is often out of date or incorrect. Instead, it's better to manually detect the features you need. The first example of this is a media query for screen size. This is helpful because if you're on a phone, the screen size will be significantly smaller (even though the actual pixel count might be large), so you would want your layout to be different. While this is the main one, another example would be if you optimized the navigation of a page around being able to hover with a mouse, it might make sense to completely change how you get around your website if you are on a phone.

    b. (5) What is a package-lock.json file? What is it used for? Is it required?

package-lock.json is a file that is generated when using npm. It stores all the exact versions and specifications of the packages that were used in the project. This is different from package.json, which only has the dependencies. package-lock.json is helpful because that way, npm install will always give the exact same environment no matter who is using it or when you set it up. However, it is not required.

    c. (5) What is npm? How does it work? Why is it used?

npm is the Node Package Manager, which is (often) bundled with node. It's used to set up node applications, install modules/packages, and manage dependencies. It works by reading the package.json and package-lock.json files in the project and installing all the modules/packages that are required for the project locally into the project (unless you manually install the module/package globally). It also modifies package.json and package-lock.json when you add new dependencies. It's used because it's an easy way to set up a project in node.js. Additionally, because the user installs the dependencies themselves, only the package json files need to be included in whatever version control system you are using, which saves a lot of space and hassle.

d. (10) Describe **in detail** the sequence(s) of transaction(s) for a frontend to request data from some external entity via Node.

The sequence will typically go as follows:

- The frontend makes a GET HTTP request (a different verb can be used if applicable) to an endpoint on the backend of the website
  - The backend does not need to be in the same filesystem
- The backend api will then make another request to the external entity.
- The external entity will (hopefully) respond with whatever data that was requested
- The backend will manipulate/format the data to the specifications that are expected (if it has to) and then send the response to the frontend
- The frontend will get the response and access the data it requested.

2. **Coding question**: (40 points) Create a webserver in node.js, name your server: server.js. You may use Express, but you *may not use a generator* – (i.e., NOT express-generator), which will serve a simple frontend (in the technologies of your choosing). The frontend will provide an input field for ZIP code and a series of buttons that issue GET and/or POST requests when clicked to the Node server. (frontend: 10 points)

   Upon entering a ZIP code and clicking the "Temperature" button, your application should send a POST request to http://localhost:3000/temperature. Node should then get the current temperature for that ZIP code (I bet you have an API for that!) and send the frontend back that information. The frontend should then output a sentence that says the name of the location and whether it is Freezing (<33F), Cold (between 33 and 50), Warm (between 51 and 80) or Hot (>80) – display the corresponding message in a unique color for each category. (temperature sequence: 10 points)

   Upon clicking the "Is RPI windy?" button, your application should send a GET request to http://localhost:3000/wind. Node should get wind speed information for Troy, NY, via that API and send that information back to the frontend. Have the frontend display this information in a unique color. (wind sequence: 10 points)

   Creativity matters; don't just give me an empty white page with a text entry form box and two buttons. Go beyond the minimum (but remember that creativity doesn't have to be visual). If you need to, write a short README file that tells me what I should consider for creativity. (creativity: 10 points)

   You may use any and all libraries you want for this coding question.

3. (15) Ensure the package.json file for Q2 has no errors when I run npm install & run your code.

4. (20) Provide **two** different explanations of the code below. The first explanation should be a high-level explanation (no less than four complete sentences) outlining what this code does to someone who has no coding experience. The second explanation should be a *detailed* one explaining line-by-line what the code does. If there are any errors in the code, fix them.

```
var net = require('net');

var sockets = []; // Fixed from == to =

var s = net.Server(function(socket) {
   sockets.push(socket);

   socket.on('data', function(d) {
      for(var i=0; i<sockets.length;i++) {
         if (sockets[i]==socket) continue;
         sockets[i].write(d);
      }
   });
   socket.on('end', function() {
      var i=sockets.indexOf(socket);
      sockets.splice(i,1);
   });
});

s.listen(8080);
```

High level:

This is the code for a server. All it does is have a list of all the people that are currently on the server. Then, if somebody sends data to the server, this code allows that data to be sent to everybody else on the server. The code also correctly removes somebody from the list if they were to stop being connected to the server. It does all this by working off of a bunch of code somebody else made called “net”.

Line by line:

```
// Tell node that we are using the "net" module
var net = require('net');

// Initialize an empty array called sockets
var sockets = [];

// Set up a net server that will run a function whenever somebody
connects
var s = net.Server(function (socket) {
  // Add the socket that just connected to the array of sockets
  sockets.push(socket);

  // Set up an event listener for a socket sending new data
  socket.on('data', function (d) {
    // Loop over all the sockets
    for (var i = 0; i < sockets.length; i++) {
      // If we're on the socket that sent the data, ignore it so the
socket doesn't send data to itself
      if (sockets[i] == socket) continue;

      // Send the data to the other sockets
      sockets[i].write(d);
    }
  });
  // When a socket stops it's connection
  socket.on('end', function () {
    // Find the index that the socket that ended is at
    var i = sockets.indexOf(socket);

    // Remove that socket from the list of sockets
    sockets.splice(i, 1);
  });
});

// Server s will listen for new connections on port 8080
s.listen(8080);
```

5. (+5) What is the name of the RPI-developed chat protocol popular in the 1990s?

HipChat I think