Type Ahead

```html
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <title>Type Ahead
```

```
      const matchArray = findMatches(this.value, cities);
      const html = matchArray.map(place => {
        const regex = new RegExp(this.value, 'gi');
        const cityName = place.city.replace(regex, `<span 
class="hl">${this.value}</span>`);
        const stateName = place.state.replace(regex, `<span 
class="hl">${this.value}</span>`);
        return `
      <li>
        <span class="name">${cityName}, ${stateName}</span>
        <span class="population">${numberWithCommas(place.population)}</span>
      </li>
    `;
      }).join('');
      suggestions.innerHTML = html;
    }

    const searchInput = document.querySelector('.search');
    const suggestions = document.querySelector('.suggestions');

    searchInput.addEventListener('change', displayMatches);
    searchInput.addEventListener('keyup', displayMatches);


  </script>
</body>

</html>
```

Below is a brief explanation of the exercise that I have made for this challenge:

This exercise serves as practice for creating a live search feature with HTML, CSS, and JavaScript. It involves fetching city and state data, dynamically filtering results as users type, and updating suggestions in real-time. It reinforces skills in asynchronous data handling, regular expressions, and DOM manipulation for interactive web development.

JavaScript

1. **Fetch Data:**
   The script starts by declaring a constant endpoint that holds the URL of a JSON file containing city and state data. The fetch function is then used to retrieve this data asynchronously. Once the data is fetched, it is converted to JSON format using .json() and pushed into the cities array.

2. **Filter Function:**
   The findMatches function takes two parameters: wordToMatch (the user input) and cities array. It uses the filter method to match the input against city and state names. Regular expressions (RegExp) with the 'gi' flag are employed for case-insensitive global matching.

3. **Number Formatting:**
   The numberWithCommas function is defined to add commas to large numbers for better readability. This is achieved using the .replace method with a regular expression that targets the appropriate positions in the number.

4. **Display Matches:**
   The displayMatches function is triggered by the 'change' and 'keyup' events on the search input. It finds matches using findMatches, then dynamically generates HTML for the suggestions, highlighting the matched text. The results are added to the DOM by updating the innerHTML property of the suggestions element.

5. **Event Listeners:**
   Event listeners are added to the search input for both 'change' and 'keyup' events. These events trigger the displayMatches function, ensuring that the search suggestions are updated in real-time as the user types or changes the input.

6. **DOM Selection:**
   Finally, the script selects the search input and suggestions elements using document.querySelector, enabling interaction with these elements in the code. The search input triggers the display of matching suggestions, providing a responsive and user-friendly search experience.

What I have learned

This challenge taught me how to build a live search on a website using JavaScript. I learned to fetch and filter city data, use regular expressions for matching, and update the webpage dynamically based on user input.