The finals are graded, and I’m happy to report that it appears to have gone even better than the midterm did. Because of the shear amount of material you needed master in order to do well, I’m happy going with my traditional final exam curve, which places the median grade at the B/B+ border.

You can determine your letter grade by looking up your score in the following table:

<table>
<thead>
<tr>
<th>Range</th>
<th>Grade</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>97–100</td>
<td>A+</td>
<td>3</td>
</tr>
<tr>
<td>92–96</td>
<td>A</td>
<td>7</td>
</tr>
<tr>
<td>86–91</td>
<td>A−</td>
<td>2</td>
</tr>
<tr>
<td>78–85</td>
<td>B+</td>
<td>4</td>
</tr>
<tr>
<td>69–77</td>
<td>B</td>
<td>6</td>
</tr>
<tr>
<td>63–68</td>
<td>B−</td>
<td>4</td>
</tr>
<tr>
<td>58–62</td>
<td>C+</td>
<td>0</td>
</tr>
<tr>
<td>52–57</td>
<td>C</td>
<td>2</td>
</tr>
<tr>
<td>42–51</td>
<td>C−</td>
<td>2</td>
</tr>
<tr>
<td>00–41</td>
<td>D</td>
<td>1</td>
</tr>
</tbody>
</table>

Median = 78%
Solution 1—Python Strings

def lrs(s):
    """
    Extracts every possible nonempty substring from the one provided and sees if each is repeated a second time. While doing so, we keep track of the longest such substring and ultimately return that substring at the end.
    """
    longest = ""
    for lh in range(0, len(s)):
        for rh in range(lh + 1, len(s)):
            subs = s[lh:rh]
            if len(subs) > len(longest) and s[lh + 1:].find(subs) != -1:
                longest = subs
    return longest

Solution 2—Python Strings and Lists

def zigzag(cleartext, numRails):
    """
    Constructs the rails needed to encrypt the provided cleartext message as per the rail fence cipher algorithm.
    """
    rails = [""] * numRails
    pos = 0
    delta = 1
    for ch in cleartext:
        rails[pos] += ch
        for i in range(numRails):
            if i != pos:
                rails[i] += "."
        pos += delta
        if pos == 0 or pos == numRails - 1: delta *= -1
    return rails

Solution 3—Working with Python Dictionaries and Objects

def computeScore(words, scores):
    """
    Computes the sentiment score of list of provided words. Note that my solution uses get to surface a default value when a key is absent, as with:
    
    info = scores.get(word, {})
    
    An equivalent way of doing this is:
    
    info = {} if word not in scores else scores[word]
    
    Either approach is fine.
    """
    if len(words) == 0: return 0.0
    info = scores.get(words[0], {})
    score = info.get("polarity", 0.0)
for i in range(1, len(words)):
    info = scores.get(words[i - 1], {})
    intensity = info.get("intensity", 1.0)
    info = scores.get(words[i], {})
    polarity = info.get("polarity", 0.0)
    score += intensity * polarity
return score

Solution 4—Working with Python Dictionaries and Objects

class TestData:
    def __init__(self, filename):
        ""
        Parses the provided data file and builds up a dictionary for each
        state, where each state maps to another dictionary for each city
        within that state, and each state-city entry maps to a list
        of dictionaries, one per high school.
        The parsing also builds up a list of 101 lists of high
        schools, one for each adversity score. (We set aside space
        for 0, even though it's an invalid adversity score, just so that
        everything lines up nicely.
        ""
        self._cityStateMap = {}
        self._adversities = []
        for i in range(101):
            self._adversities.append([])
        with open(filename) as infile:
            while True:
                state = infile.readline().strip()
                if state == "": break
                if state not in self._cityStateMap:
                    self._cityStateMap[state] = {}
                while True:
                    line = infile.readline().strip()
                    if line == "": break
                    parts = line.split(",")
                    city = parts[0].strip()
                    if city not in self._cityStateMap[state]:
                        self._cityStateMap[state][city] = []
                    school = {
                        "name": parts[1].strip(),
                        "sat": int(parts[2].strip()),
                        "adversity": int(parts[3].strip())
                    }
                    self._cityStateMap[state][city].append(school)
                    self._adversities[school["adversity"]].append(school["name"])

    def getCityStateData(self, city, state):
        ""
        Returns the list of all high schools under the jurisdiction of
        the provided city and state. The error checking was required
        of this one, since some state and many cities could be missing.
        ""
        if state not in self._cityStateMap or city not in self._cityStateMap[state]:
            return []
        return self._cityStateMap[state][city]

    def getHighSchoolsWithAdversity(self, score):
        """
Returns the names of the high schools that have been assigned the supplied adversity score. While we expect error checking to guard against out-of-range scores, we did not require you to confirm the supplied score is an integer.

```python
if score <= 0 or score > 100: return []
return self._adversities[score]
```

Solution 5—Client-Side JavaScript [kudos to Jonathan Kula for this problem!]

```javascript
function BootstrapSearchEngine() {
    /* define frequently accessed interactors at upper level */
    let input = document.getElementById("search-text");
    let button = document.getElementById("search-button");
    let results = document.getElementById("search-results");

    function onButtonClick(e) {
        AsyncRequest("/articles")
            .addParam("search", input.value.trim())
            .setSuccessHandler(listSearchResults)
            .send();
    }

    function listSearchResults(response) {
        let articles = JSON.parse(response.getPayload());

        while (results.lastChild !== null) {
            results.removeChild(results.lastChild);
        }

        for (let i = 0; i < articles.length; i++) {
            let div = document.createElement("div");
            let h2 = document.createElement("h2");
            h2.appendChild(document.createTextNode(articles[i].title));
            div.appendChild(h2);
            let p = document.createElement("p");
            p.appendChild(document.createTextNode(articles[i].summary));
            div.appendChild(p);
            let a = document.createElement("a");
            a.setAttribute("href", articles[i].link);
            a.appendChild(document.createTextNode("Read more"));
            div.appendChild(a);
            results.appendChild(div);
        }
    }

    button.addEventListener("click", onButtonClick);
}
```