Decomposition

Modular Design

Announcements

Separation of Concerns

A design principle: Isolate different parts of a program that address different concerns.
A modular component can be developed and tested independently.

<table>
<thead>
<tr>
<th>Hog</th>
<th>Game Commentary</th>
<th>Player Strategies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hog Game Simulator</td>
<td>• Event descriptions</td>
<td>• Decision rules</td>
</tr>
<tr>
<td></td>
<td>• User input</td>
<td>• Strategy parameters</td>
</tr>
<tr>
<td></td>
<td>• Game rules</td>
<td>(e.g., margins &amp; number of dice)</td>
</tr>
<tr>
<td></td>
<td>• Ordering of events</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• State tracking to determine the winner</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ants</th>
<th>Actions</th>
<th>Tunnel Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ants Game Simulator</td>
<td>• Characteristics of different ants &amp; bees</td>
<td>• Entrances &amp; exits</td>
</tr>
<tr>
<td></td>
<td>• Order of actions</td>
<td>• Locations of insects</td>
</tr>
<tr>
<td></td>
<td>• Food tracking</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Game ending conditions</td>
<td></td>
</tr>
</tbody>
</table>
Example: Restaurant Search

Restaurant Search Data

Given the following data, look up a restaurant by name and show related restaurants.

```json
{
  "business_id": "gclB3ED6uk6wiLo5b_uA",  
  "name": "Cafe 3",  
  "stars": 2.0,  
  "price": 1,  
}
{
  "business_id": "WXKx2I2SEzBpeUGtDMCS8A",  
  "name": "La Cascada Taqueria",  
  "stars": 3.0,  
  "price": 2
}
...
{
  "business_id": "gclB3ED6uk6wiLo5b_uA",  
  "user_id": "xVocUuZkZTaAgCmHk3xYQ",  
  "stars": 1,  
  "text": "Cafe 3 (or Cafe Tre, as I like to say) used to be the bomb diggity when I first lived in the dorms but sadly, quality has dramatically decreased over the years....",  
  "date": "2012-01-19"
}
...
```

Discussion Question: Most Similar Restaurants

Implement `similar`, a `Restaurant` method that takes a positive integer `k` and a function `similarity` that takes two restaurants as arguments and returns a number. Higher `similarity` values indicate more similar restaurants. The `similar` method returns a list containing the `k` most similar restaurants according to the `similarity` function, but not containing `self`.

```python
def similar(self, k, similarity):
    """Return the K most similar restaurants to SELF, using SIMILARITY for comparison."""
    others = list(Restaurant.all)
    others.remove(self)
    return sorted(others, key=lambda r: -similarity(self, r))[:k]
```

`sorted(iterable, /, *, key=None, reverse=False)`

Return a new list containing all items from the iterable in ascending order. A custom key function can be supplied to customize the sort order, and the reverse flag can be set to request the result in descending order.
Linear-Time Intersection of Sorted Lists

Given two sorted lists with no repeats, return the number of elements that appear in both.

```python
def fast_overlap(s, t):
    # Return the overlap between sorted S and sorted T.
    >>> fast_overlap([3, 4, 6, 7, 9, 10], [1, 3, 5, 7, 8])
    2
    i, j, count = 0, 0, 0
    while i < len(s) and j < len(t):
        if s[i] == t[j]:
            count, i, j = count + 1, i + 1, j + 1
        elif s[i] < t[j]:
            i = i + 1
        else:
            j = j + 1
    return count
```

(Demo)