Questions

Recursion mystery

```python
1  def rec_sum(nums):
2      if len(nums) == 0:
3          return 0
4      rest = rec_sum(nums[1:])
5      print(rest, end="", ")
6      return nums[0] + rest
7    print(rec_sum([8, 4, 2, 1]))
```

* 0, 1, 3, 7, 15

Point mysteries

```python
1  class Point:
2      def __init__(self, x, y)
3          self.x = x
4          self.y = y
5      def __repr__(self):
6          return "("+str(self.x)+","+str(self.y)+")"
7
8  p = Point(1, 5)
9  x = Point(p.y, p.x)
10  r = p
11  r.x = r.x + 1
12  x.x = p.x
13  print(p, x, r)
```

* (2, 5) (2, 1) (2, 5)
* r is an alias for p
  * p = Point(1, 5)
  * x = Point(p.y, p.x)
r = p

How many classes, instances, and variables?
- 1 class (Point), 2 instances ((1, 5) and (5, 1)), 3 variables (p, x, r)

- Election data from homework 3
  - House of Representatives, each line has the state, district, candidate, and number of votes
  - Represent all of this in a dictionary
    - states -> districts -> candidates -> votes
    - `fp = open("district_overall_2018.csv")`
      - `lines = fp.readlines()`
      - `fp.close()`
    - `for i in range(1, len(lines)):`
      - `lines[i] = lines[i].strip().split(“,“)`
    - `data = {}`
    - `for line in lines:`
      - `if line[1] in data and line[12] == "FALSE":`
        - `d = data[line[1]]`
      - `if line[7] in d:`
        - `d = d[line[7]]`
      - `if line[10] not in d:`
        - `d[line[10]] = 0`
      - `d[line[10]] += int(line[14])`
      - `elif line[12] == "FALSE":`
        - `data[line[1]] = {line[7]: {line[10]: int(line[14])}}`

  - think about/work through how you would approach the first three problems (getting started, representation, margin of victory) using this new representation

- Practice: sorted list
  - Define a SortedList class that has a list as an instance variable and an append method that ensures the list is always in sorted order after an element is added

```
1 class SortedList():
2     def __init__(self):
3         self.elems = []
4     def append(self, elem):
5         self.elems.append(elem)
6         self.elems.sort()
```

- Recursion
  - Game of tic-tac-toe, want to represent all the possible "pathways" through the game
  - see tic-tac-toe-state.py