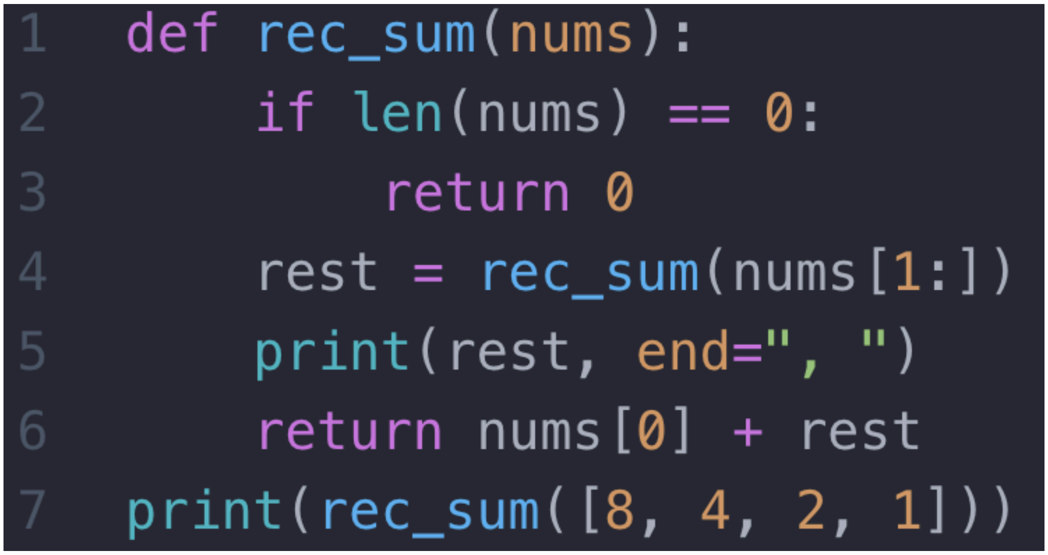
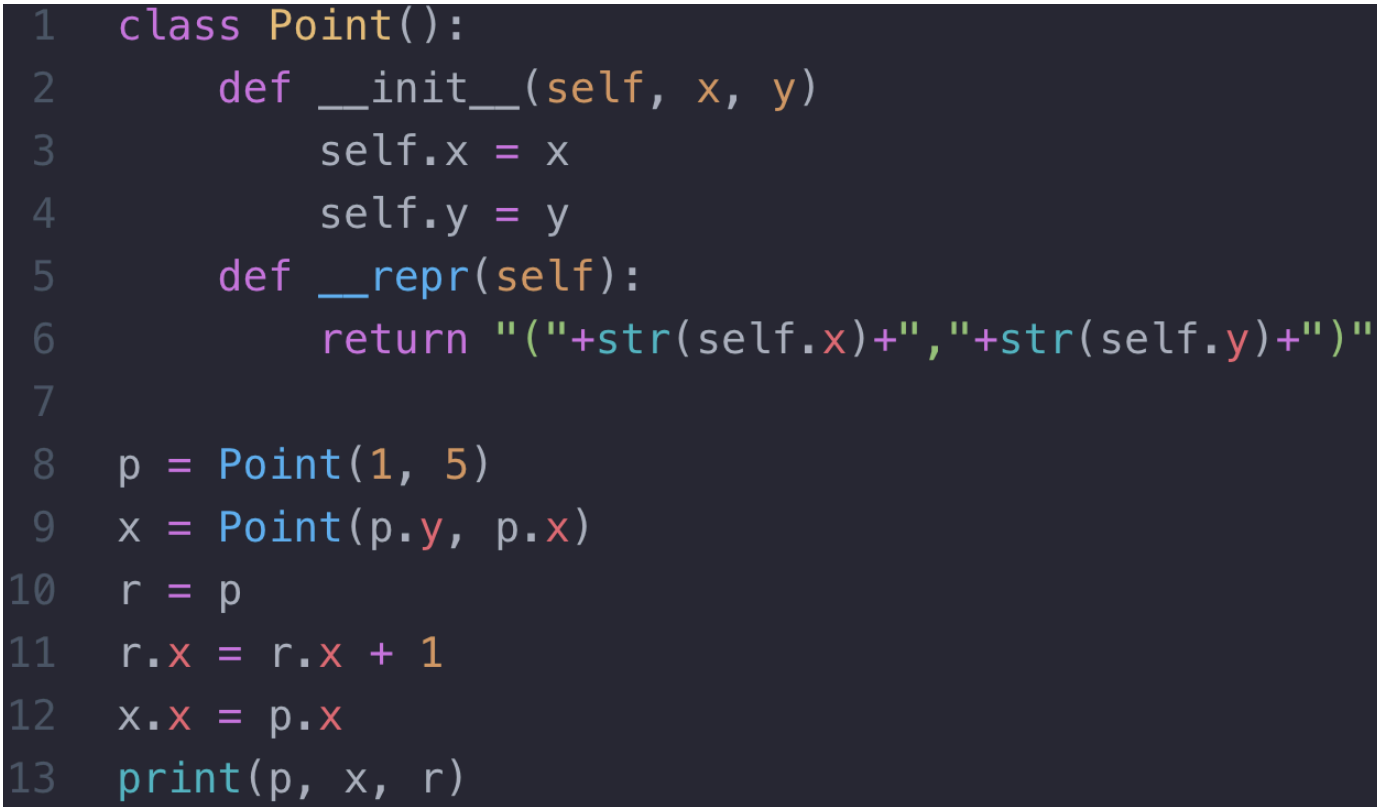
- Questions
- Recursion mystery



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

  - 0, 1, 3, 7, 15

- Point mysteries



```python
class Point():
    def __init__(self, x, y)
        self.x = x
        self.y = y
    def __repr(self):
        return "("+str(self.x)+","+str(self.y)+")"

p = Point(1, 5)
x = Point(p.y, p.x)
r = p
r.x = r.x + 1
x.x = p.x
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)

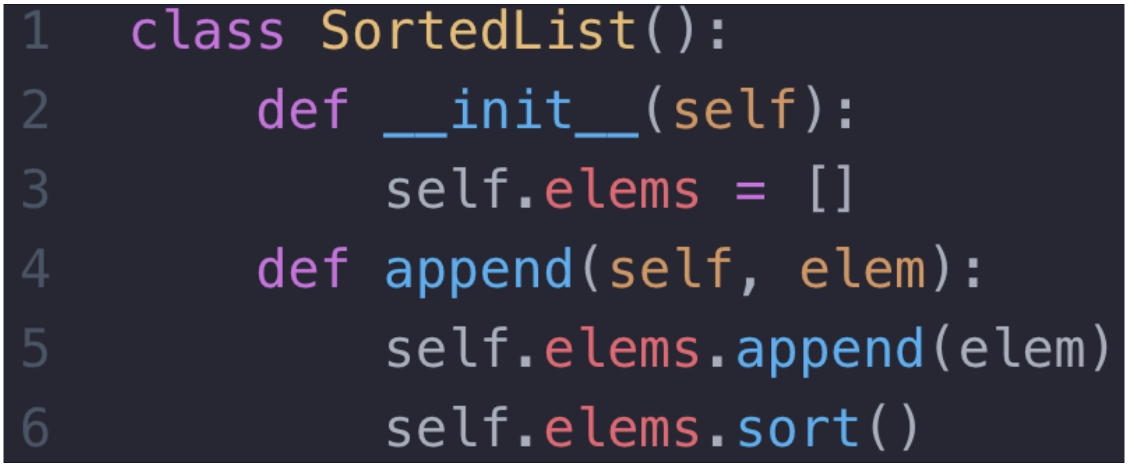
Election data from homework 3

- 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])
            else:
                d[line[7]] = {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
- 

  ```
  class SortedList():
      def __init__(self):
          self.elems = []
      def append(self, elem):
          self.elems.append(elem)
          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