Review 5

Recursion

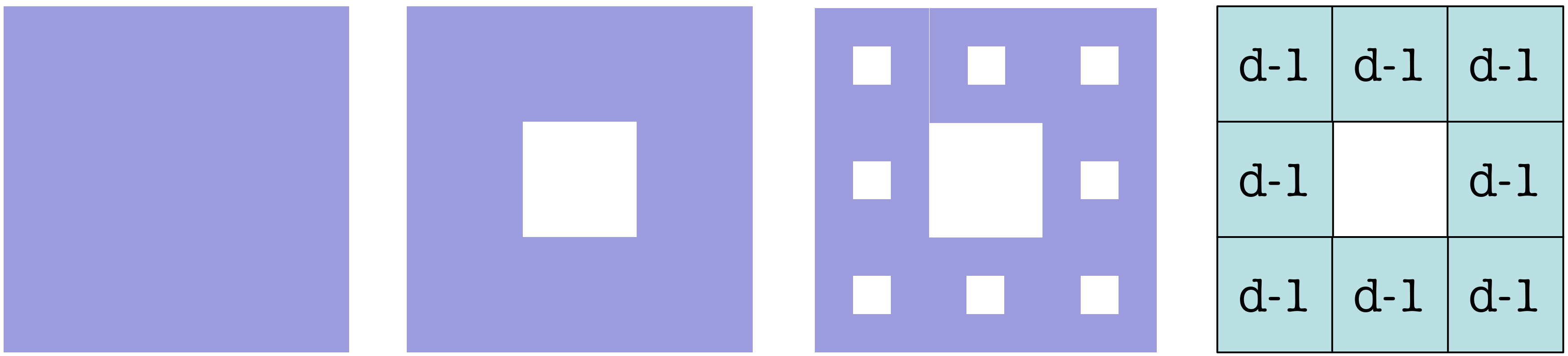
The Two Types of Recursion in CS 1110

- Recursive Definitions
  - The specification itself is recursive
  - Code simply implements the definition
  - **Example**: The shapes in A4
- Divide and Conquer
  - The specification is not recursive
  - But it involves data that can be broken up
  - **Example**: Most of Lab 8

Recursive Definition: Spring 2006

- The Sierpinski Carpet has the following form

depth 0 depth 1 depth 2 depth d



- Assume the following helper

```
def drawsquare(x,y,side):
    """Draws a square of length side centered at x,y
    Precondition: x,y,side are numbers >= 0"""
```

Recursive Definition: Spring 2006

```
def carpet(x,y,side,d) {
    """Draws a Sierpinski Carpet of depth d
     The carpet is has length side centered at x,y
     Precondition: x,y,side,d are numbers >= 0"""
```

Recursive Definition: Spring 2006

```
def carpet(x,y,side,d) {
    """Draws a Sierpinski Carpet of depth d"""
    if d == 0:
        drawsquare(x,y,side)
    else:
        carpet(x-side/3,y-side/3,side/3,d-1)
        carpet(x,y-side/3,side/3,d-1)
        carpet(x+side/3,y-side/3,side/3,d-1)
        carpet(x-side/3,y,side/3,d-1)
        carpet(x+side/3,y,side/3,d-1)
        carpet(x-side/3,y+side/3,side/3,d-1)
        carpet(x,y+side/3,side/3,d-1)
        carpet(x+side/3,y+side/3,side/3,d-1)
```

Three Steps for Divide and Conquer

1. Decide what to do on "small" data
   - Some data cannot be broken up
   - Have to compute this answer directly
2. Decide how to break up your data
   - Both "halves" should be smaller than whole
   - Often no wrong way to do this (next lecture)
3. Decide how to combine your answers
   - Assume the smaller answers are correct
   - Combining them should give bigger answer

Complement of an Integer

```
def complement(int n) {
    """Returns: the complement of the number n
    Each decimal digit in n is replaced by 10-n.
    Example: the result for 93723 is 17387.
    Precondition: n > 0 and int, and no digit of n is 0"""
```

Complement of an Integer

```
def complement(int n) {
    """Returns: the complement of the number n
    Precondition: n > 0 and int, and no digit of n is 0"""
    # Small Data



    # Break it up and recurse



    # Combine answer

```

Complement of an Integer

```
def complement(int n) {
    """Returns: the complement of the number n
    Precondition: n > 0 and int, and no digit of n is 0"""
    # Small Data
    if n < 10:
        return 10 – n
    # Break it up and recurse
    left    = complement(n/10)
    right = 10 - n%10                    # complement(n % 10)
    # Combine answer
    return left*10+right
```

Combining Recursion and Loops

```
def deepsum(nested):
    """Returns: Sum of all numbers in nested list
    Examples:
      deepsum([1,2,3]) is 6
      deepsum([[1,2],[3]]) is 6
      deepsum([[1,[2,3]],[[[4]]]]) is 10
    Precondition: nested a nested list of ints (or empty)"""
```

Combining Recursion and Loops

```
def deepsum(nested):
    """Returns: Sum of all numbers in nested list
    Precondition: nested a nested list of ints (or empty)"""
    # Small Data



    # Recurse over EACH element in the list
```

Combining Recursion and Loops

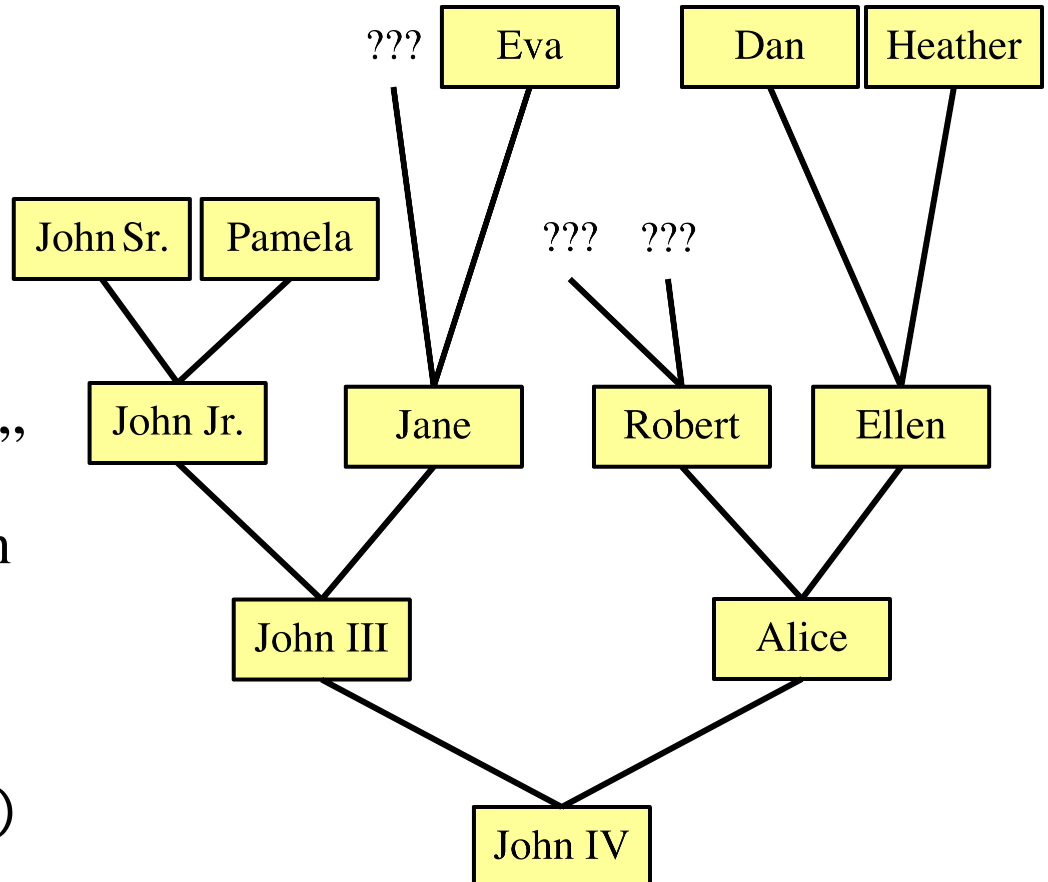
```
def deepsum(nested):
    """Returns: Sum of all numbers in nested list
    Precondition: nested a nested list of ints (or empty)"""
    # Small Data
    if len(nested) == 0:
        return 0
    # Recurse over EACH element in the list
```

Combining Recursion and Loops

```
def deepsum(nested):
    """Returns: Sum of all numbers in nested list
    Precondition: nested a nested list of ints (or empty)"""
    # Small Data
    if len(nested) == 0:
        return 0
    # Recurse over EACH element in the list
    accum = 0
    for item in nested:
        if type(item) == list:
            accum = accum + deepsum(item)
        else:
            accum = accum + item
    return accum
```

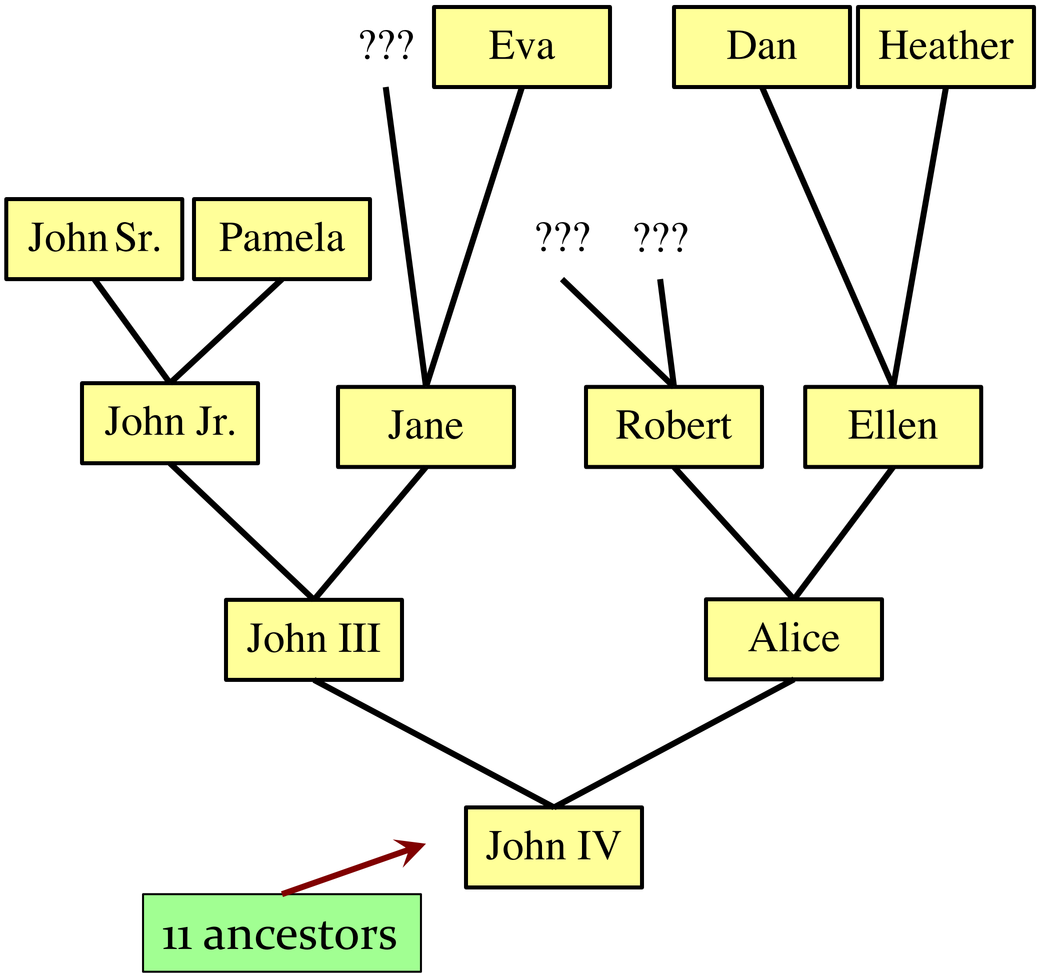
Recursion and Objects

- Class Person (`person.py`)
  - Objects have 3 attributes
  - `name`: String
  - `mom`: Person (or `None`)
  - `dad`: Person (or `None`)
- Represents the "family tree"
  - Goes as far back as known
  - Attributes `mom` and `dad` are `None` if not known
- **Constructor**: `Person(n,m,d)`
  - Or `Person(n)` if no mom, dad



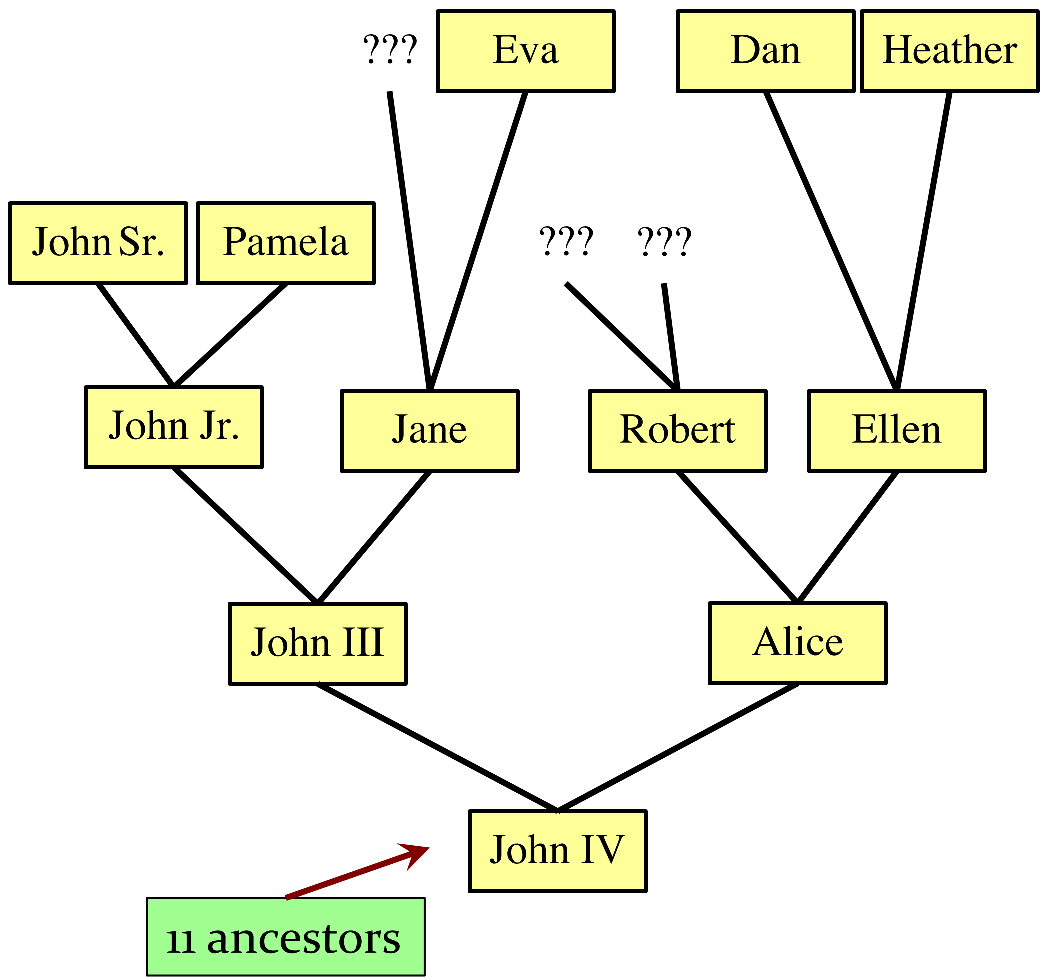
Recursion and Objects

```
def num_ancestors(p):
    """Returns: num of known ancestors
    Pre: p is a Person"""
    # Small Data
    # No mom or dad (no ancestors)


    # Break it up and recurse
    # Has mom or dad
    # Count ancestors of each one
    # (plus mom, dad themselves)
    # Add them together


    # Combine
```



Recursion and Objects

```python
def num_ancestors(p):
    """Returns: num of known ancestors
    Pre: p is a Person"""
    # Small Data
    if p.mom == None and p.dad == None:
        return 0

    # Break it up and recurse
    moms = 0
    if not p.mom == None:
        moms = 1+num_ancestors(p.mom)
    dads = 0
    if not p.dad== None:
        dads = 1+num_ancestors(p.dad)
    # Combine
    return moms+dads
```



Extra Problems

- Use recursion to find minimum element in a list
  - Small data is easy
  - Hard part is combine
- Given list, use recursion to check if it is sorted
  - Small data is easy
  - Again, hard part is combine
- Given a string s, list all the permutations of s:
  - `'XZY'` → `'XZY'`, `'XYZ'`, `'ZXY'`, `'ZYX'`, `'YXZ'`, `'YZX'`
  - This one is a little trickier

One Last Problem

```
class FacebookProfile(object):
    """name [str]: name of this profile
       friends [list of FacebookProfile]: friends list"""
```

We want to answer the question:

- Is this profile at most 6 degrees away from Kevin Bacon?
- In other words, is Kevin Bacon a friend of a friend of a friend of a friend of a friend of a friend?

Specification (Method inside class FacebookProfile):

```
    def sixDegreesOfBacon(self):
        """Returns: True if this FacebookProfile is at most 6
        degrees away from Kevin Bacon; False otherwise"""
```

6-Degrees of Kevin Bacon

```
class FacebookProfile(object):
    …
    def sixDegreesOfBacon(self):
        """Returns: True if this FacebookProfile is at most 6 degrees away from Kevin Bacon"""


    def sixDegreesHelper(self,n):
        """Returns: True if this FacebookProfile is at most n degrees away from Kevin Bacon
        Precondition: n > 0 an int"""
```

6-Degrees of Kevin Bacon

```
class FacebookProfile(object):
    …
    def sixDegreesOfBacon(self):
        """Returns: True if this FacebookProfile is at most 6 degrees away from Kevin Bacon"""
        return self.sixDegreesHelper(6)

    def sixDegreesHelper(self,n):
        """Returns: True if this FacebookProfile is at most n degrees away from Kevin Bacon
        Precondition: n > 0 an int"""
        # Small Data




        # Break it up, recurse and combine
```

6-Degrees of Kevin Bacon

```
class FacebookProfile(object):
    ...
    def sixDegreesOfBacon(self):
        """Returns: True if this FacebookProfile is at most 6 degrees away from Kevin Bacon"""
        return self.sixDegreesHelper(6)

    def sixDegreesHelper(self,n):
        """Returns: True if this FacebookProfile is at most n degrees away from Kevin Bacon
        Precondition: n >= 0 an int"""
        # Small Data
        if self.name == 'Kevin Bacon':
            return True
        if n == 0:
            return False
        # Break it up, recurse and combine
```

6-Degrees of Kevin Bacon

```
class FacebookProfile(object):
    ...
    def sixDegreesOfBacon(self):
        """Returns: True if this FacebookProfile is at most 6 degrees away from Kevin Bacon"""
        return self.sixDegreesHelper(6)

    def sixDegreesHelper(self,n):
        """Returns: True if this FacebookProfile is at most n degrees away from Kevin Bacon
        Precondition: n > 0 an int"""
        # Small Data
        if self.name == 'Kevin Bacon':
            return True
        if n == 0:
            return False
        # Break it up, recurse and combine
        for f in self.friends:
            if f.sixDegreesHelper(n-1):
                return True
        return False
```

Questions?