Presentation 15

More Recursion

Announcements for This Lecture

Assignments and Labs

- Need to be working on A4
  - Just reading it takes a while
  - Slightly longer than A3
  - Finish 1-3 before Tuesday
- **Labs**: lots of practice!
  - Many optional functions
  - Exam questions on Prelim 2
  - Great way to study

Other Announcements

- View the lesson videos
  - **Videos 17.6-17.11** for today
  - **Lesson 18** next time
  - Also **Videos 19.1-19.7**
  - Note this is a lot of videos
- **Exam graded by Saturday**
  - Will appear in GradeScope
  - Note Submission renamed

More Divide and Conquer

```
def decode(nlist):
    """
    Returns a string that represents the decoded nlist


    The nlist a list of lists, where each element is a character, number.
    The number is the number of times to repeat the character.

    Example: decode([['a',3],['h',1],['a',1]]) is 'aaaha'

    Precondition: nlist is a (possibly empty) nested list of two-element
    lists, where each list inside is a pair of a character and an integer
    """
    pass
```

More Divide and Conquer

```
def decode(nlist):
    """
    Returns a string that represents the decoded nlist

    The nlist a list of lists, where each element is a character, number.
    The number is the number of times to repeat the character.

    Example: decode([['a',3],['h',1],['a',1]])

    Precondition: nlist is a (possibly emp                                ,
    lists, where each list inside is a pair
    """
    pass
```

How Divide?

A: Cut in half

B: Pull off one elt.

C: Does not matter

More Divide and Conquer

```
def decode(nlist):
    """
    Returns a string that represents the decoded nlist


    The nlist a list of lists, where each element is a character, number.
    The number is the number of times to repeat the character.

    Example: decode([['a',3],['h',1],['a',1]])

    Precondition: nlist is a (possibly emp
    lists, where each list inside is a pair
    """
    pass
```

How Combine?

A: Add left, right

B: Add right, left

C: Something trickier

More Divide and Conquer

```
def encode(text):
    """
    Returns a nested list encoding the duplication of each character

    The returned list is a (possibly empty) nested list of two-element
    lists, where each list inside is a pair of a character and an integer.

    Example: encode('aaaha') is [['a',3],['h',1],['a',1]]

    Precondition: text is a (possibly empty) string
    """
    pass
```

More Divide and Conquer

```
def encode(text):
    """
    Returns a nested list encoding the duplication of each character

    The returned list is a (possibly empty) nested list of two-element
    lists, where each list inside is a pair of a character and an integer.

    Example: encode('aaaha') is [['a',3],['

    Precondition: text is a (possibly emp
    """
    pass
```

How Divide?

A: Cut in half

B: Pull off one elt.

C: Does not matter

More Divide and Conquer

```
def encode(text):
    """
    Returns a nested list encoding the duplication of each character

    The returned list is a (possibly empty) nested list of two-element
    lists, where each list inside is a pair of a character and an integer.

    Example: encode('aaaha') is [['a',3],['

    Precondition: text is a (possibly emp
    """
    pass
```

How Combine?

A: Add left, right

B: Add right, left

C: Something trickier

Here is a HARD One

```
def segregate(nlist):
    """
    Returns a tuple segregating nlist into negative and non-negative.

    This function returns a tuple (pos,rlist). The value rlist is a reordered copy of nlist
    where negatives come before the non-negatives. However, ordering  inside each
    part (negative, non-negatives) should remain EXACTLY as it is in nlist.

    The value pos indicates the first position of a non-negative number in rlist.
    If there are no non-negative numbers, pos is -1.

    Example: segregate([1, -1, 2, -5, -3, 0]) returns (3, [-1, -5, -3, 1, 2, 0])

    Precondition: nlist is a (possibly empty) list of numbers
    """
    pass
```

Here is a HARD One

```
def segregate(nlist):
    """
    Returns a tuple segregating nlist into negative and non-negative.

    This function returns a tuple (pos,rlist). The value rlist is a reordered copy of nlist
    where negatives come before the non-negatives. However, ordering  inside each
    part (negative, non-negatives) should remain EXACTLY as it is in nlist.

    The value pos indicates the first position of a non-negative number in rlist.
    If there are no non-negative numbers, pos is

    Example: segregate([1, -1, 2, -5, -3, 0]) retur

    Precondition: nlist is a (possibly empty) list
    """
    pass
```

How Divide?

A: Cut in half

B: Pull off one elt.

C: Does not matter

Working with Objects

```
def ancestors(p):
    """
    Returns the list of names of all ancestors of p

    The name of p should not be in the list (unless another ancestor has
    this name).  Duplicates names (e.g. ancestors with the same name)
    are okay.

    The list returned should be sorted alphabetically

    See family.py for examples

    Precondition: p is a Person and not None
    """
    pass
```

Working with Objects

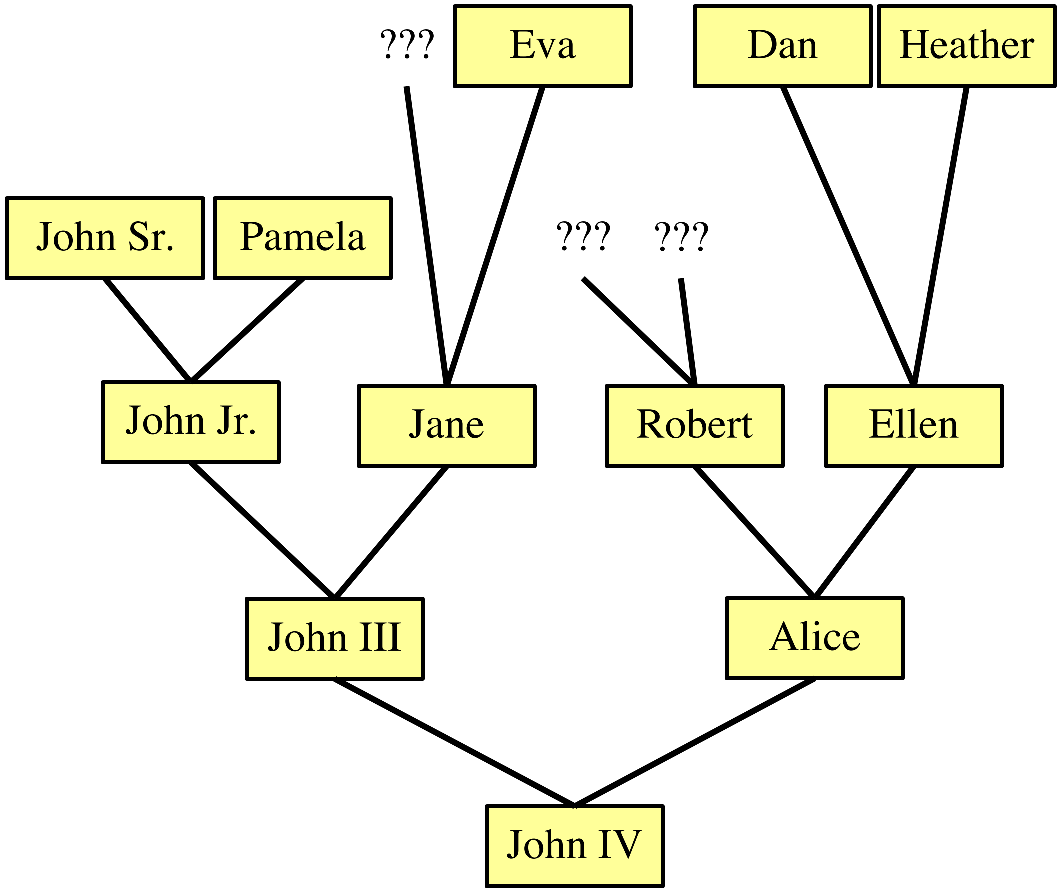
```
def ancestors(p):
    """
    Returns the list of names

    The name of p should not
    this name).  Duplicates na
    are okay.

    The list returned should b

    See family.py for example

    Precondition: p is a Perso
    """
    pass
```



Working with Objects

```
def ancestors(p):
    """
    Returns the list of names of all ancestors of p

    The name of p should not be in the list (unless another ancestor has
    this name).  Duplicates names (e.g. ancestors with the same name)
    are okay.

    The list returned should

    See family.py for exampl

    Precondition: p is a Pers
    """
    pass
```

Why is a helper needed?

A: It is needed to make list

B: It is needed to sort list

C: No helper is needed

Working with Objects

```
def ancestors(p):
    """
    Returns the list of names of all ancestors of p

    The name of p should not be in the list (unless another ancestor has
    this name).  Duplicates names (e.g. ancestors with the same name)
    are okay.

    The list returned should

    See family.py for exampl

    Precondition: p is a Pers
    """
    pass
```

Why is a helper needed?

A: It is needed to make list

[✓ B: It is needed to sort list]

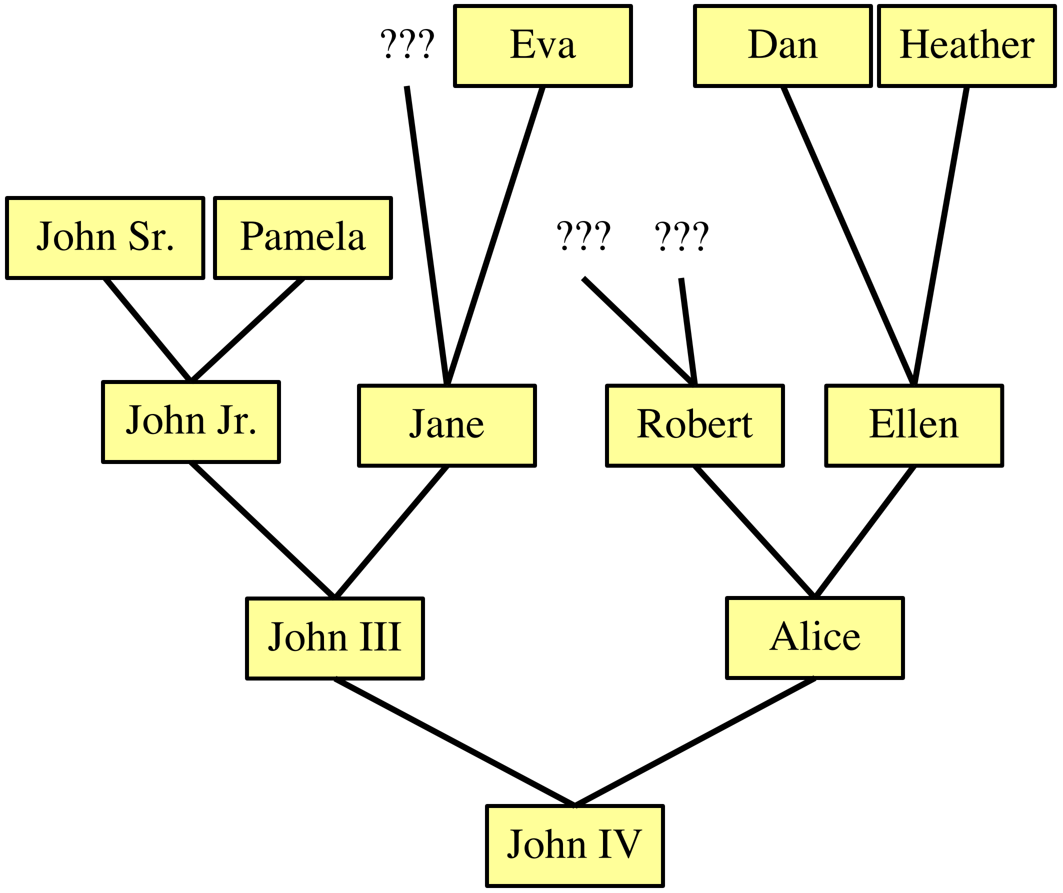
C: No helper is needed

Working with Objects

```
def related(p,q):
    """
    Returns True if Persons p
    If either p or q is None, th

    Two people are related if 
    tree (including themselve
    either they are the same 
    ancestor (parent, grandpa

    Preconditions: p and q are
    """
    pass
```



Working with Objects

```
def related(p,q):
    """
    Returns True if Persons p and q are related, False otherwise
    If either p or q is None, this function returns False.

    Two people are related if they have a common person in their family
    tree (including themselves). A recursive way of saying this is that
    either they are the sa
    ancestor (parent, gra

    Preconditions: p and
    """
    pass
```

How Divide?

A: By mother, father

B: By siblings (brother, sister)

C: Not a divide-and-conquer

Working with Objects

```
def related(p,q):
    """
    Returns True if Persons p and q are related, False otherwise
    If either p or q is None, this function returns False.

    Two people are related if they have a common person in their family
    tree (including themselves). A recursive way of saying this is that
    either they are the sa
    ancestor (parent, gra

    Preconditions: p and
    """
    pass
```

How Divide?

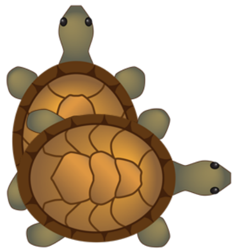
A: By mother, father

B: By siblings (brother, sister)

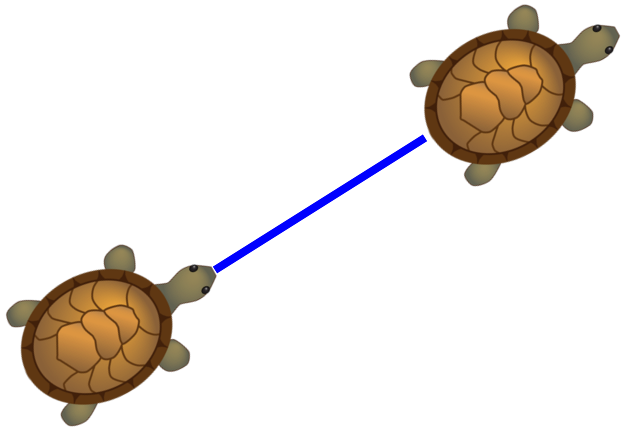
[✓ C: Not a divide-and-conquer]

Turtle Demo!

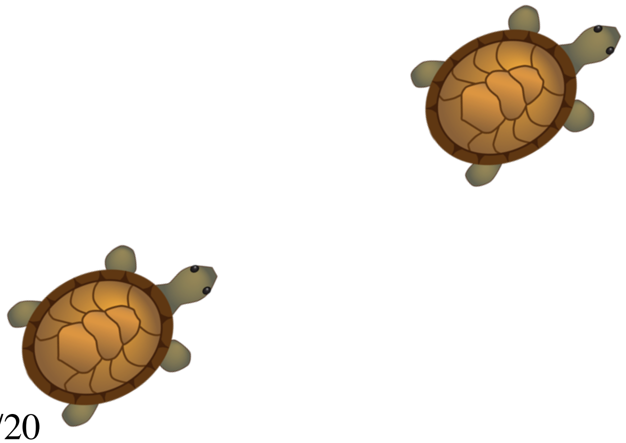
Turn



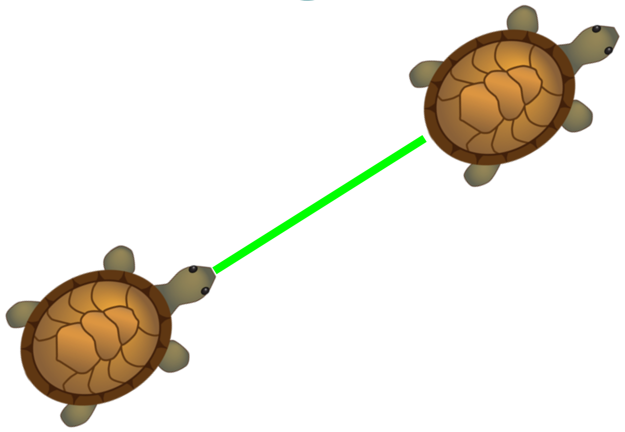
Draw Line



Move



Change Color



Questions?