Trees and Human Evolution

orangutan

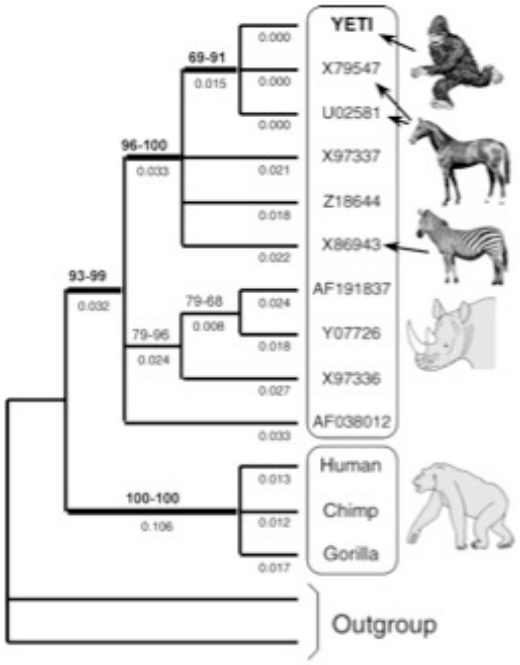
gorilla

human

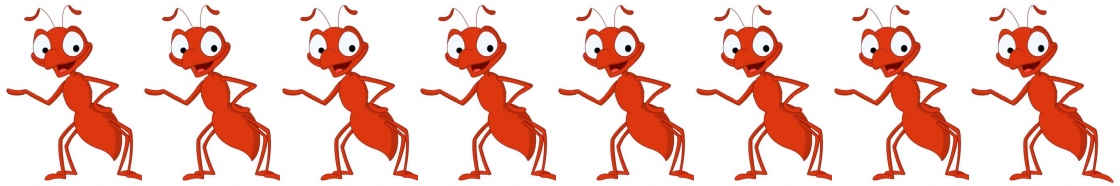
common chimpanzee

bonobo

Coming Soon to CS 5 Green



- Hmmm lab/homework
- Phylogenetics
- Midterm (in class 11/4)
- OOPs
- CS Theory
- End-of-semester projects!



Learning Goals

- Describe how data is stored in memory (just a peek)
- Introduce biological question
- Describe tree terminology and representation
- Practice writing functions on trees

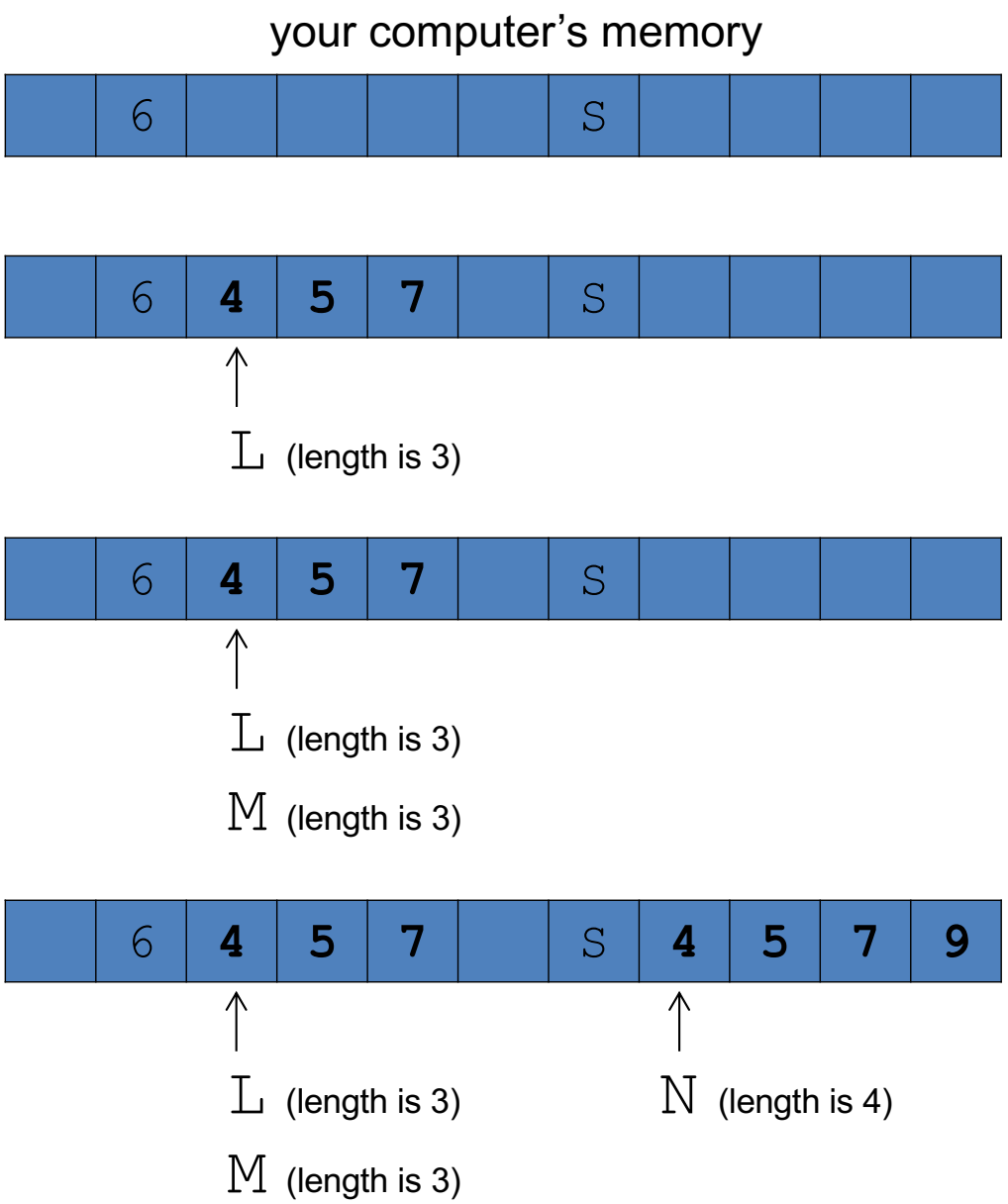
Lists Revisited

```
>>> L = [4, 5, 7]
```

```
>>> M = L
```

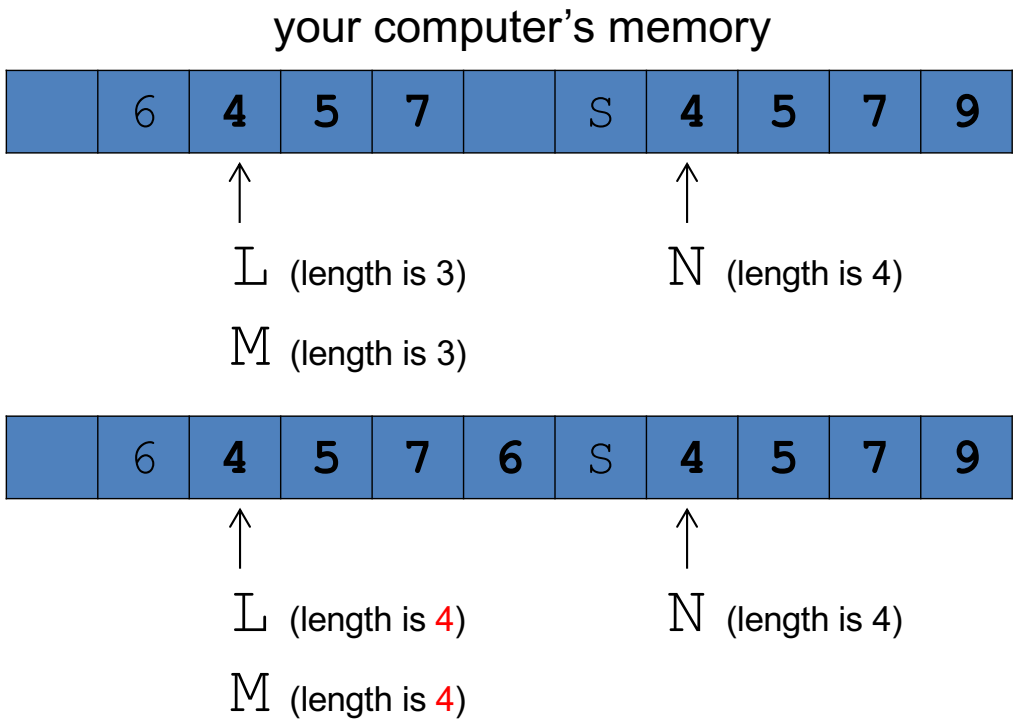
This is called "shallow copy"

```
>>> N = L + [9]
```



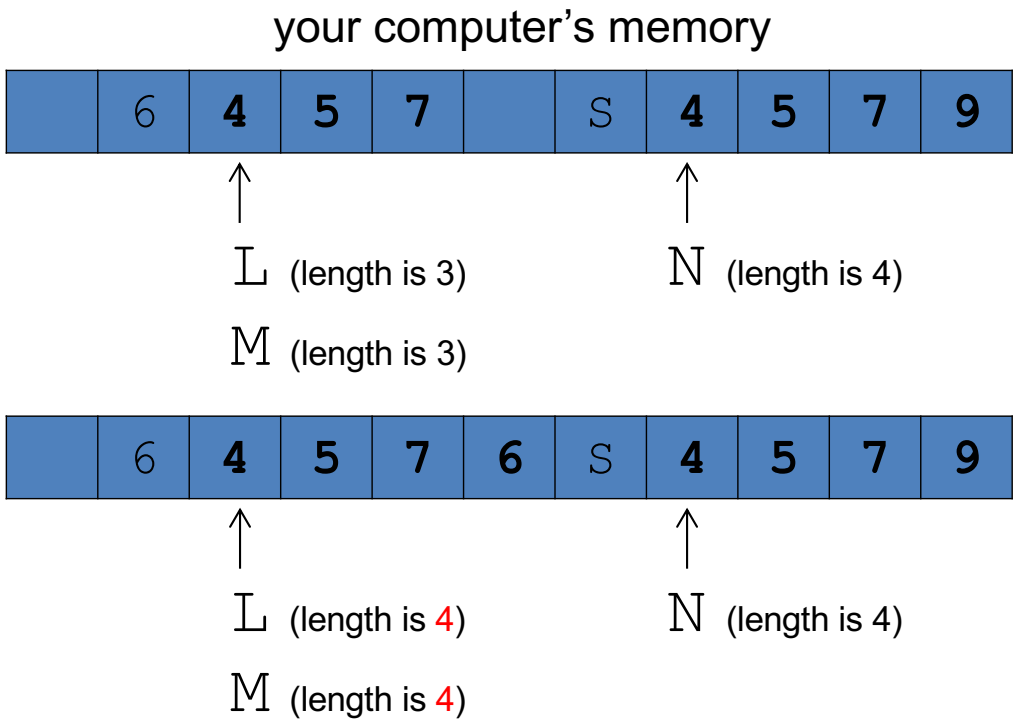
Lists Revisited


```
>>> L = [4, 5, 7]
>>> M = L
>>> N = L + [9]

>>> L.append(6)
>>> L
[4, 5, 7, 6]
>>> M

>>> N
```



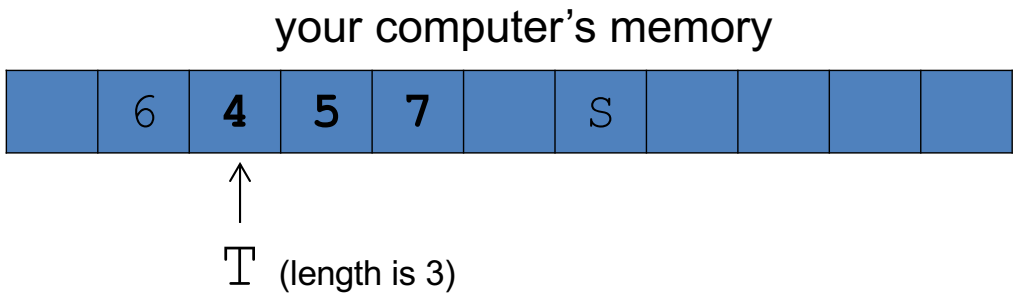
Lists Revisited


```
>>> L = [4, 5, 7]
>>> M = L
>>> N = L + [9]

>>> L.append(6)
>>> L
[4, 5, 7, 6]
>>> M
[4, 5, 7, 6]
>>> N
[4, 5, 7, 9]
```



Tuples are immutable lists

```
>>> T = (4, 5, 7)
>>> T[0]
4
>>> T[1:]
(5, 7)
>>> len(T)
3
>>> T[0] = 42
BARF!
>>> T.append(42)
BARF!
```



Learning Goals

- Describe how data is stored in memory (just a peek)
- Introduce biological question
- Describe tree terminology and representation
- Practice writing functions on trees

Neanderthals and Modern Humans

Neanderthal type specimen

the old man of La Chapelle

https://www.msu.edu/~heslipst/contents/ANP440/images/Neanderthal_1_langle.jpg
http://humanorigins.si.edu/evidence/human-fossils/fossils/la-chapelle-aux-saints
http://anthropologynet.files.wordpress.com/2007/06/neander-valley.jpg

Trees and Human Evolution

orangutan

gorilla

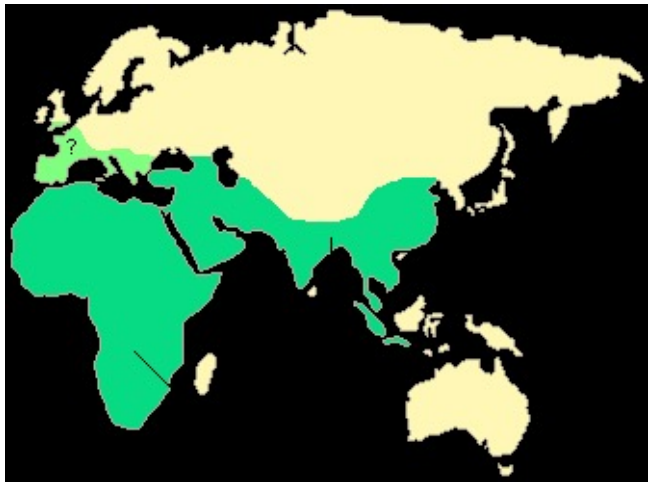
human

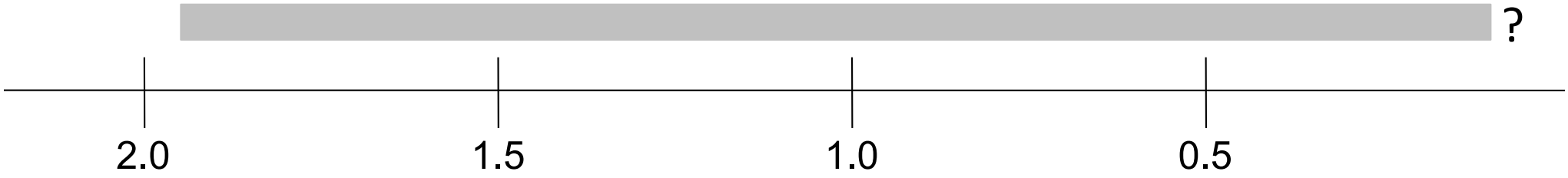
common chimpanzee

bonobo

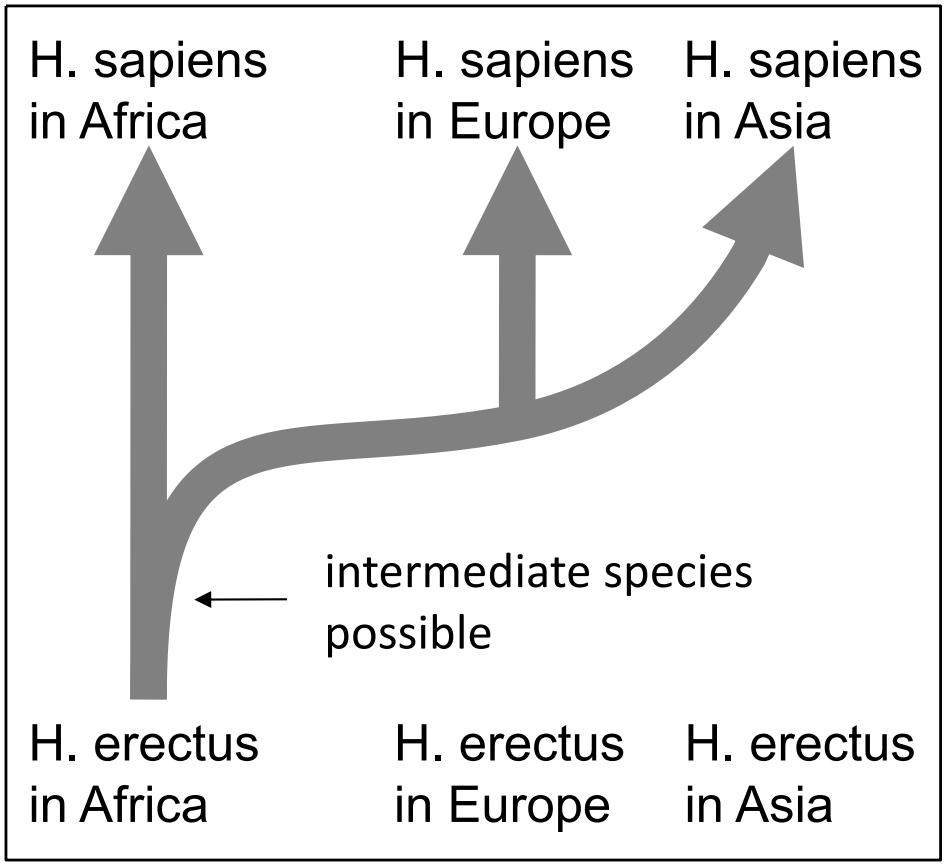
Homo erectus: first undisputed world traveler

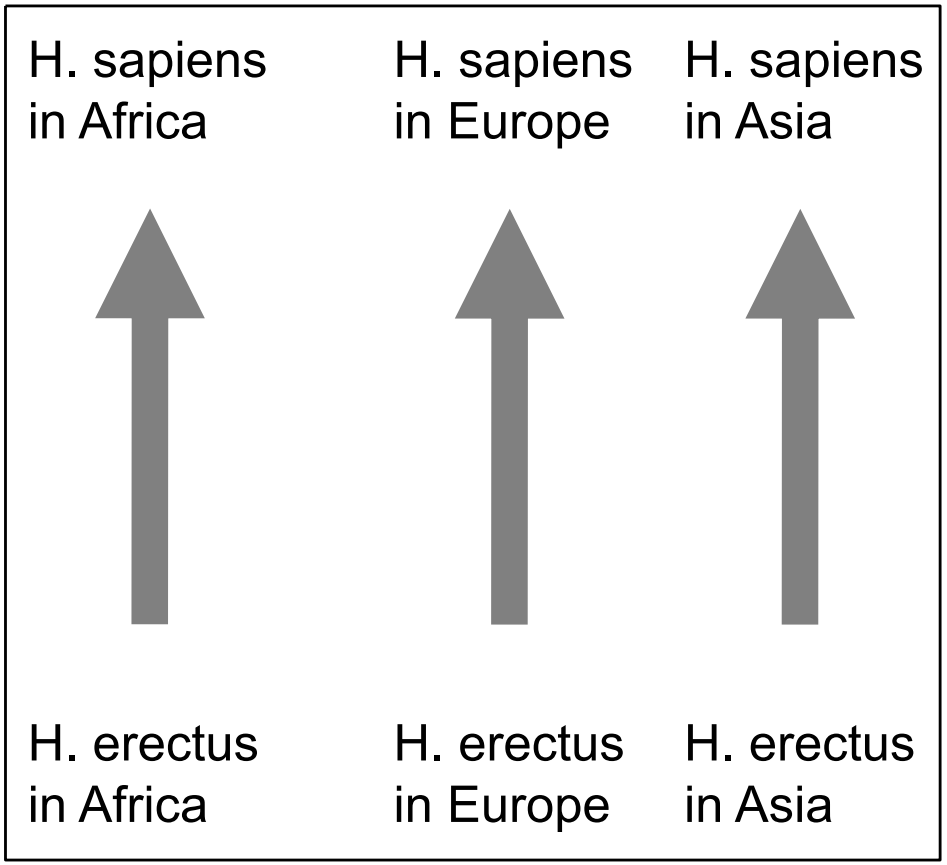
Sangiran 17, 1.3-1.0 MYA,
Sangiran Indonesia



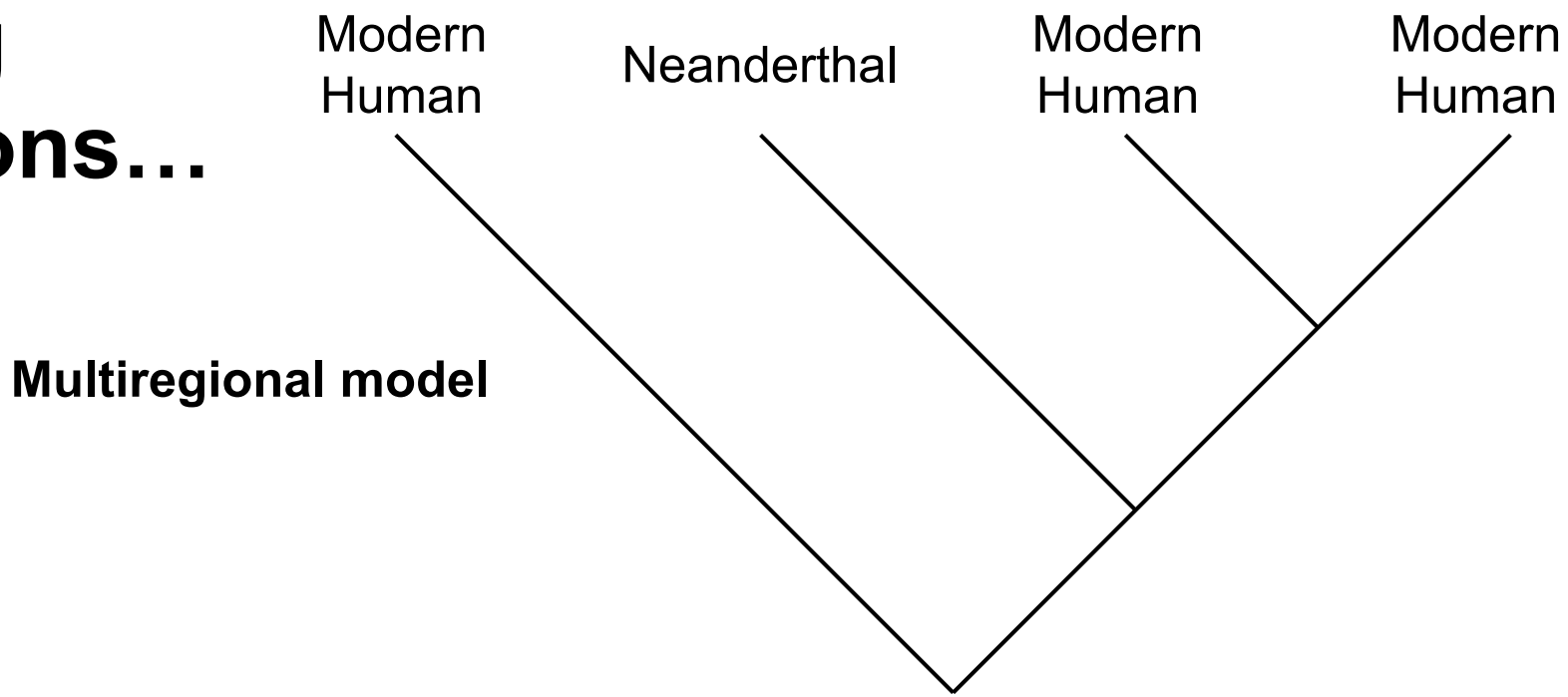


Out of Africa vs. multiregional origin of modern humans

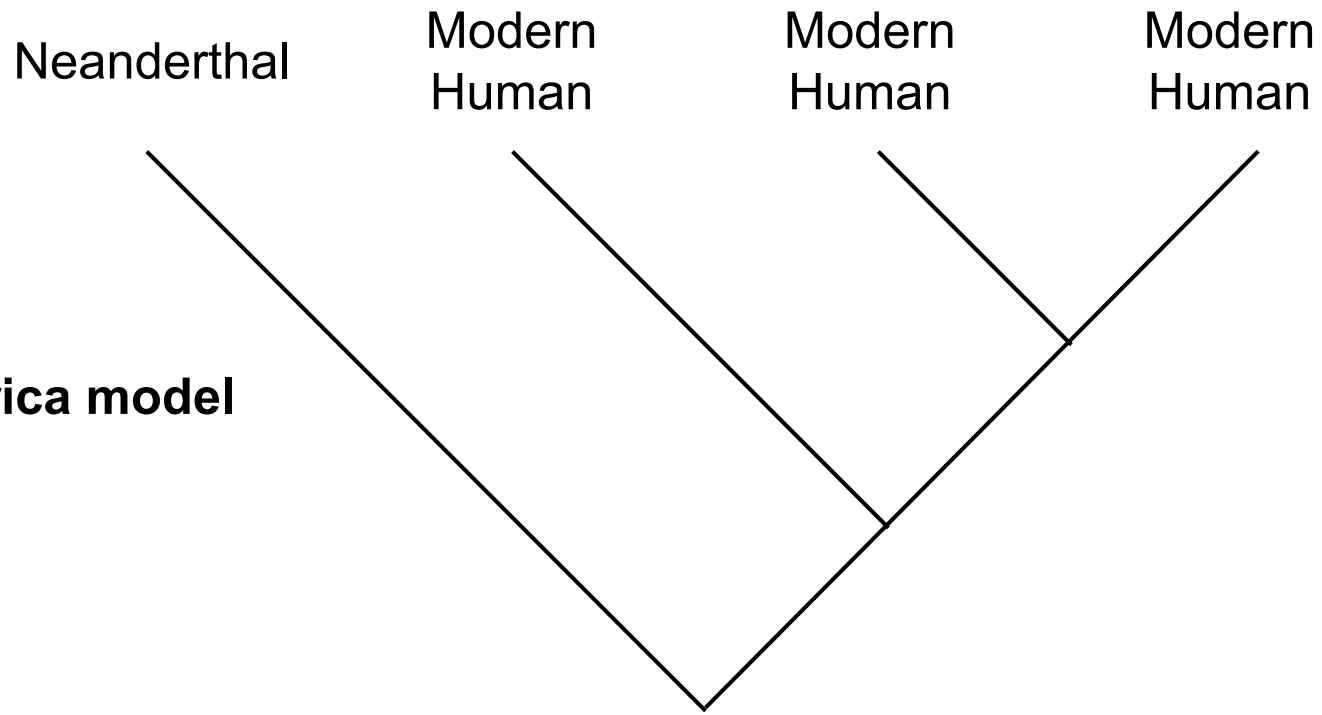




Differing predictions…



Multiregional model

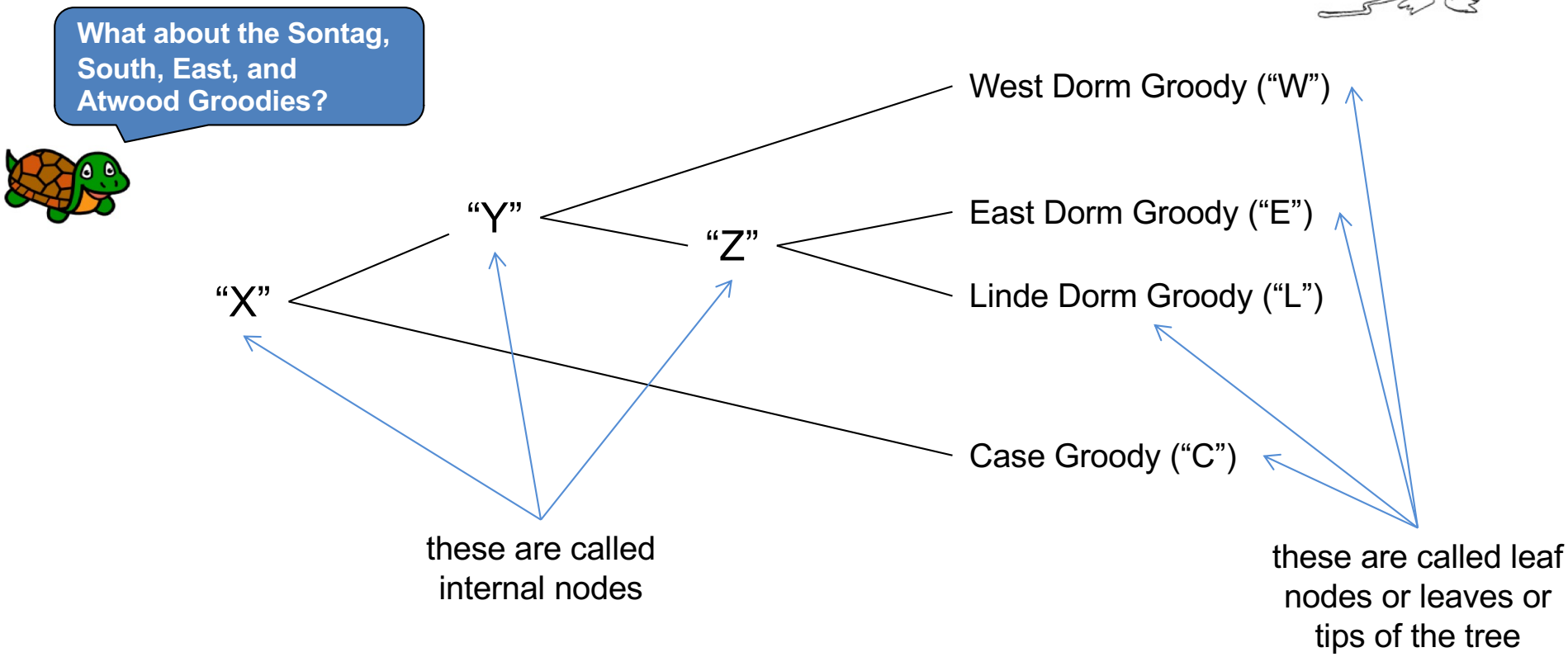


Out of Africa model

Learning Goals

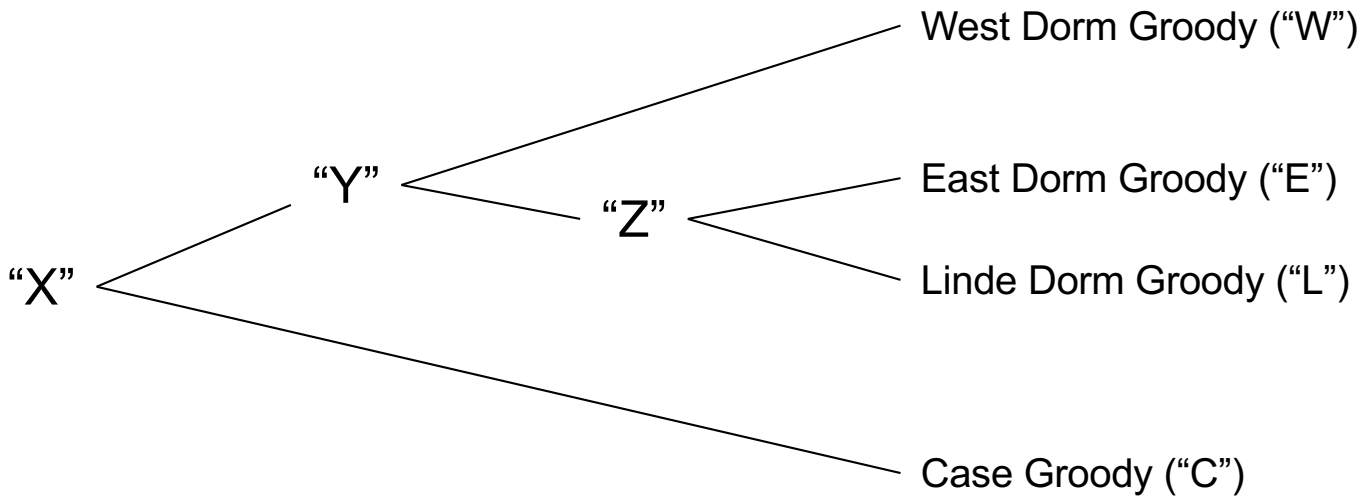
- Describe how data is stored in memory (just a peek)
- Introduce biological question
- Describe tree terminology and representation
- Practice writing functions on trees

Phylogenetic Trees



How do we represent this in Python?

RLR (root, left, right) format



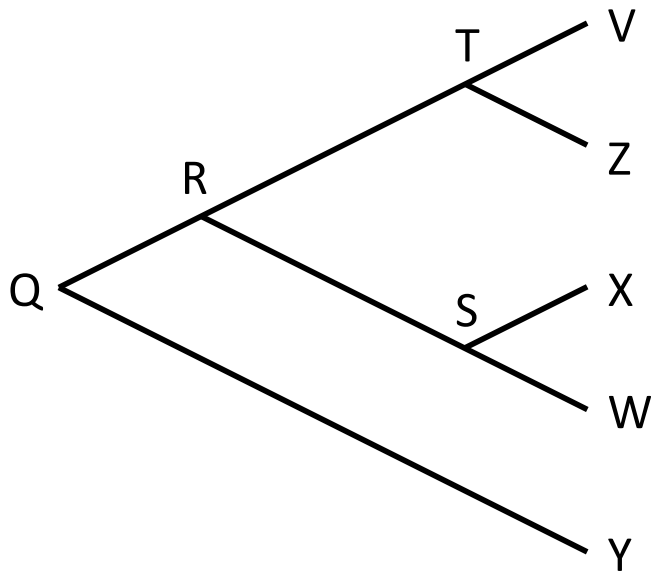

```
groodies = ("X",
                ("Y",
                      ("W", (), ()),
                      ("Z",
                            ("E", (), ()),
                            ("L", (), ())
                      )
                ),
                ("C", (), ())
           )
```

Draw this tree…


```
tr = ('Q',
           ('R',
                ('T',
                     ('V',(),()),
                     ('Z',(),()),
                ),
                ('S',
                     ('X',(),()),
                     ('W',(),()),
                )
           ),
          ('Y',(),())
     )
```

Draw this tree…


```
tr = ('Q',
          ('R',
               ('T',
                    ('V',(),()),
                    ('Z',(),()),
               ),
               ('S',
                    ('X',(),()),
                    ('W',(),()),
               )
          ),
         ('Y',(),())
     )
```

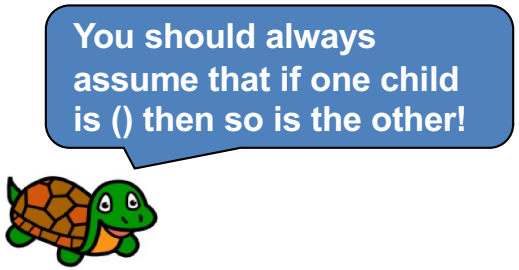


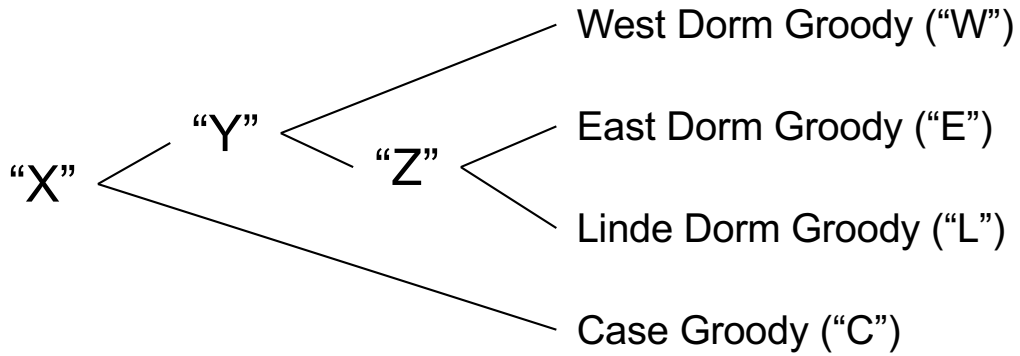
Learning Goals

- Describe how data is stored in memory (just a peek)
- Introduce biological question
- Describe tree terminology and representation
- Practice writing functions on trees

How many nodes are in this tree?


```
>>> node_count(("Yoohoo",(),()))
1
>>> node_count(groodies)
7
```



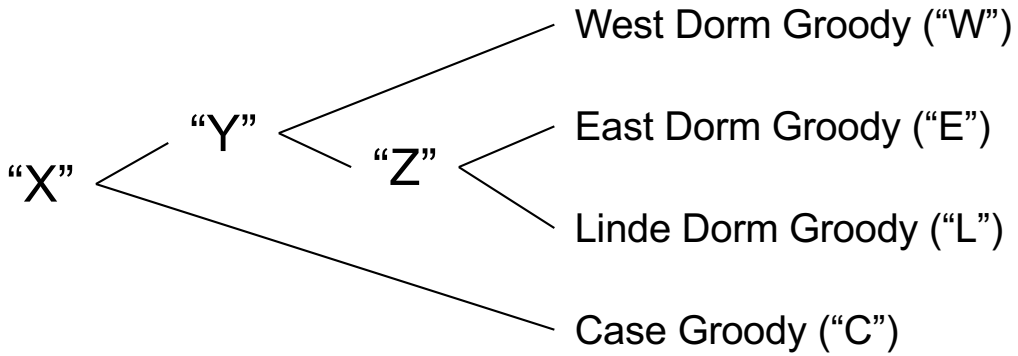


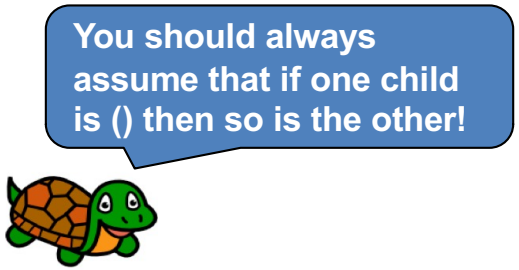

```
def node_count(tree):
    """Returns the total number of nodes in the given tree."""
```

Fill this in (in your notes)!

How many nodes are in this tree?

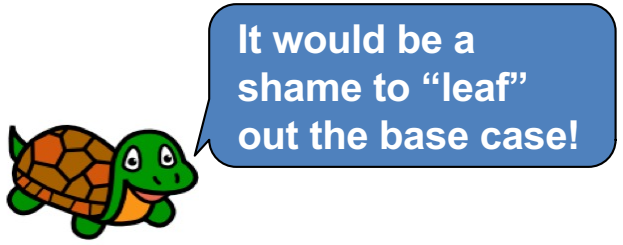

```
>>> node_count(("Yoohoo",(),()))
1
>>> node_count(groodies)
7
```



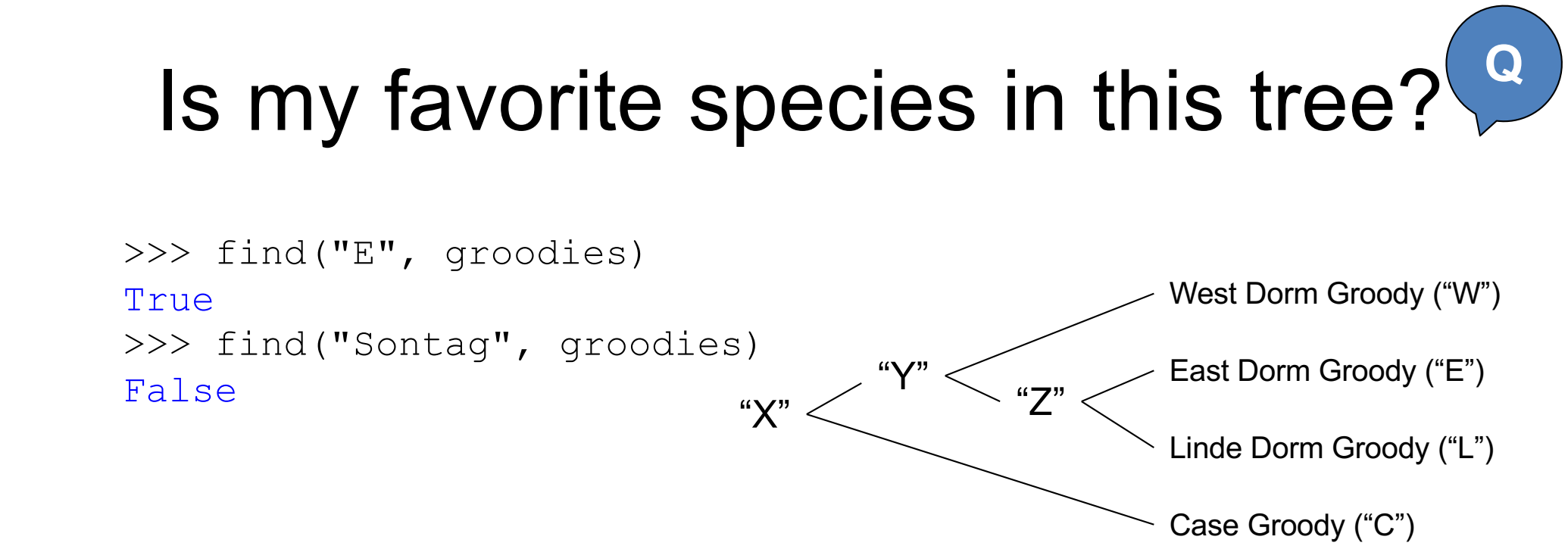



```
def node_count(tree):
    """Returns the total number of nodes in the given tree."""
    root, left, right = tree # root = tree[0], left = tree[1], right = tree[2]
    if left == (): return 1  # a leaf
    else:                    # an internal node
        return 1 + node_count(left) + node_count(right)
```

Fill this in (in your notes)!



Is my favorite species in this tree?

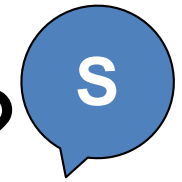



```
>>> find("E", groodies)
True
>>> find("Sontag", groodies)
False
```

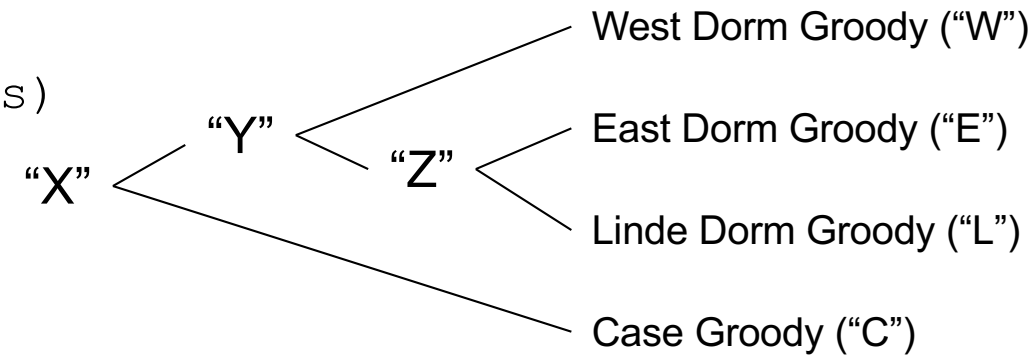


```
def find(species, tree):
    """Returns True if species is in tree and False otherwise."""
    root, left, right = tree
```

Is my favorite species in this tree?




```
>>> find("E", groodies)
True
>>> find("Sontag", groodies)
False
```

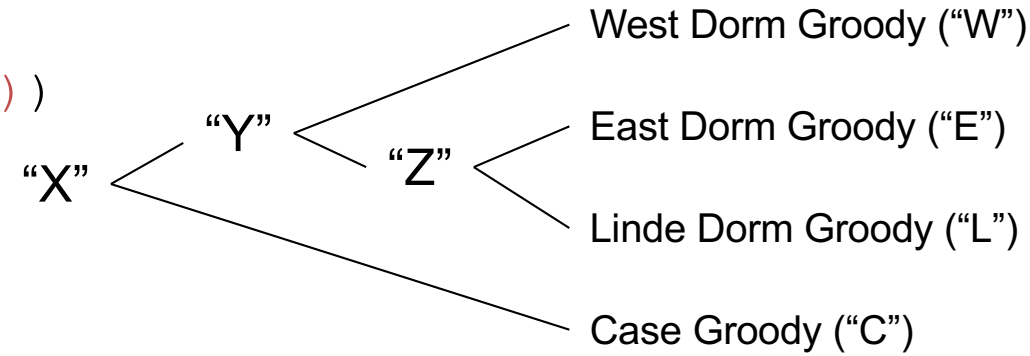



```
def find(species, tree):
    """Returns True if species is in tree and False otherwise."""
    root, left, right = tree
    if root == species: return True  # found it at the root!
    if left == (): return False
    else:
        return find(species, left) or find(species, right)
```

height

Q

```
>>> height(groodies)
3
>>> height(("spam", (), ()))
0
```



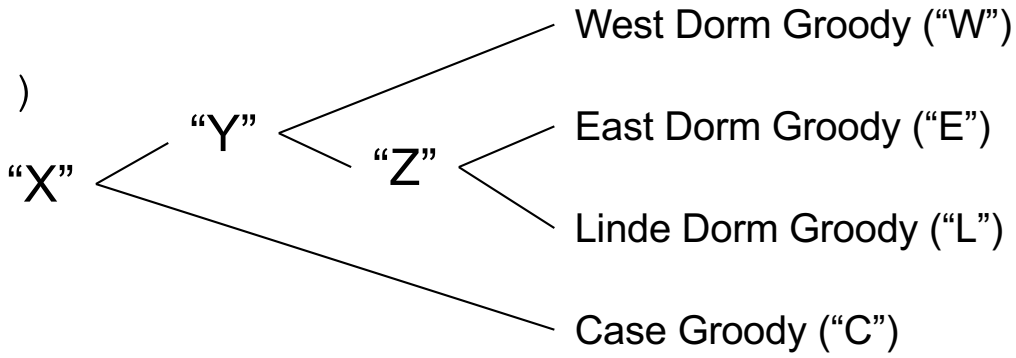
The height of a tree is the length of the path from the root to the deepest node in the tree.

```
def height(tree):
    """Returns the height of the given Tree."""
    root, left, right = tree
```

Worksheet

height


```
>>> height(groodies)
3
>>> height(("spam", (), ()))
0
```



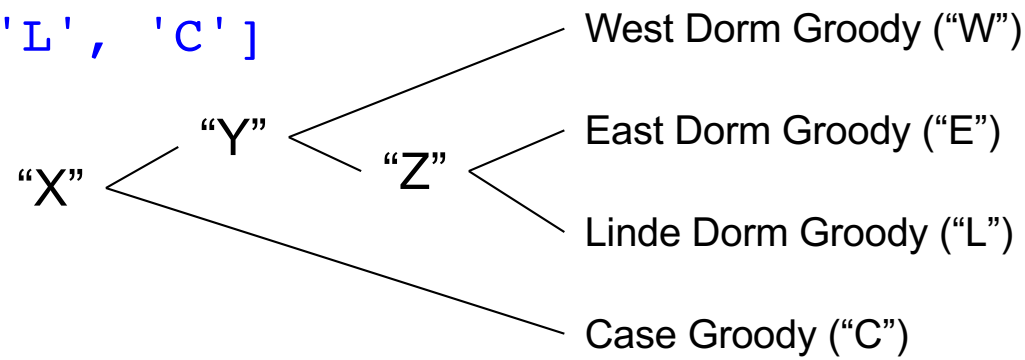
The height of a tree is the length of the path from the root to the deepest node in the tree.

```
def height(tree):
    """Returns the height of the given Tree."""
    root, left, right = tree
    if left == (): return 0  # a leaf
    else:                    # an internal node
        return 1 + max(height(left), height(right))
```

Worksheet

node_list

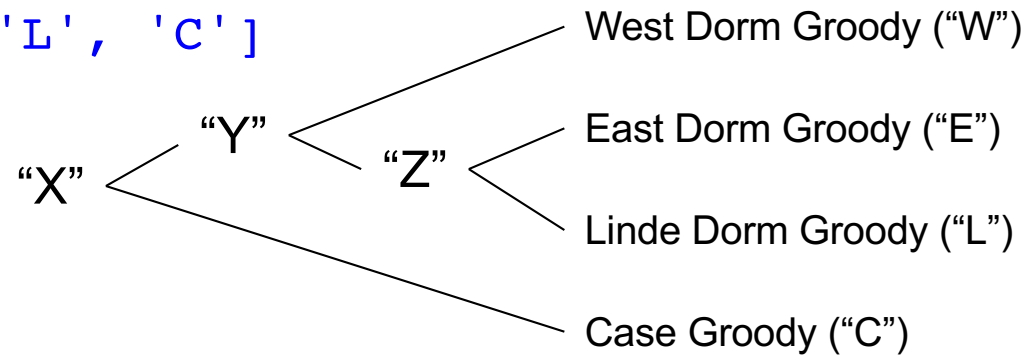

```
>>> node_list(groodies)
['X', 'Y', 'W', 'Z', 'E', 'L', 'C']
```




```
def node_list(tree):
    """Returns the list of nodes in a given tree."""
    root, left, right = tree
```

node_list

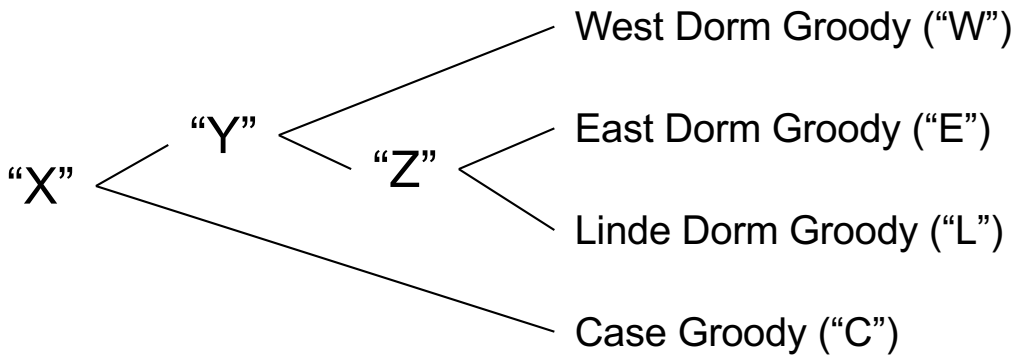

```
>>> node_list(groodies)
['X', 'Y', 'W', 'Z', 'E', 'L', 'C']
```




```
def node_list(tree):
    """Returns the list of nodes in a given tree."""
    root, left, right = tree
    if left == (): return [root]
    else:
        return [root] + node_list(left) + node_list(right)
```

leaf_list

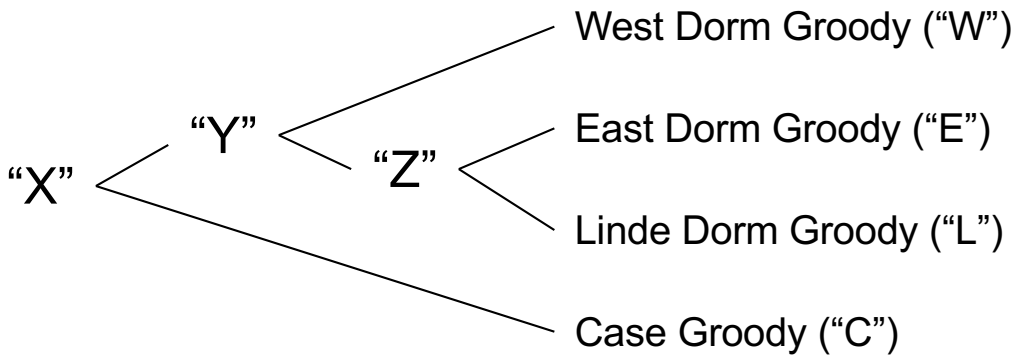

```
>>> leaf_list(groodies)
['W', 'E', 'L', 'C']
```




```
def leaf_list(tree):
    """Returns the list of leaves in a given Tree."""
    root, left, right = tree
```

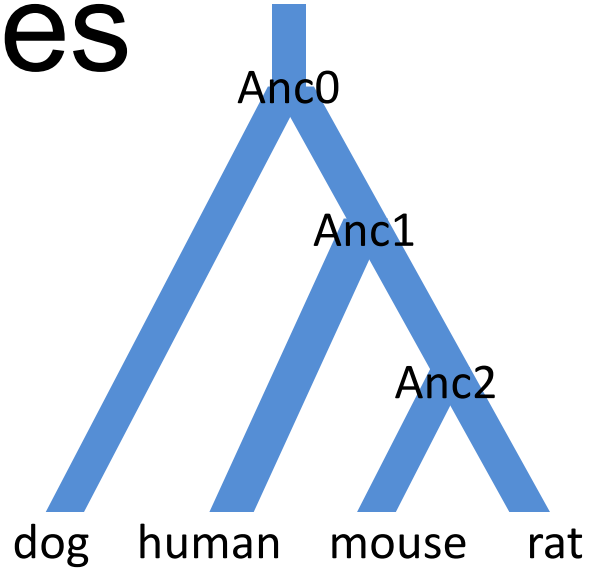
leaf_list


```
>>> leaf_list(groodies)
['W', 'E', 'L', 'C']
```




```
def leaf_list(tree):
    """Returns the list of leaves in a given Tree."""
    root, left, right = tree
    if left == ():  return [root]
    else:
        return leaf_list(left) + leaf_list(right)
```

Traversing Trees



- Print name of every node in the tree so that parents always appear…
  - *before* children (preorder)

    Anc0
    dog
    Anc1
    human
    Anc2
    mouse
    rat

  - *after* children (postorder)

    dog
    human
    mouse
    rat
    Anc2
    Anc1
    Anc0

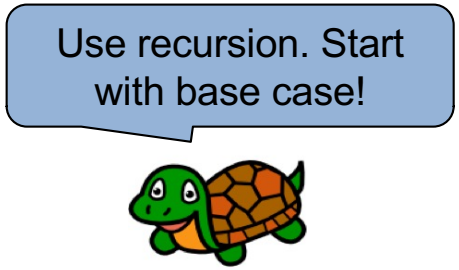
Traversing Trees

Preorder (parents first)

```
def preorder_print(tree):
    root, left, right = tree
```

Postorder (parents after)

```
def postorder_print(tree):
    root, left, right = tree
```



Traversing Trees

Preorder (parents first)

```
def preorder_print(tree):
    root, left, right = tree
    if left == ():
        print(root)
    else:
        print(root)
        preorder_print(left)
        preorder_print(right)
```

base case

print then recur

Postorder (parents after)

```
def postorder_print(tree):
    root, left, right = tree
    if left == ():
        print(root)
    else:
        postorder_print(left)
        postorder_print(right)
        print(root)
```

recur then print

Use recursion. Start with base case!