UPGMA

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>6</td>
<td>5</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>10</td>
<td>9</td>
<td>4</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>9</td>
<td>8</td>
<td>5</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>
Fitch algorithm: What are we doing?

- The *small* parsimony problem
- Analyzing a *single* tree
  - Min changes required (parsimony score)
  - Parsimonious assignment of internal node traits
Parsimony score: 2
A few missed questions:

- Scoping – (if you don’t know what it is, ignore me for a moment, we’ll discuss again later) – in Python, there is no block-level scope, variables declared in loops and conditionals are in the same scope as surrounding code.

- String find all – short answer, no built-in function for this, but you could easily build your own after we discuss loops!
**for loops**

`for` loops let you repeatedly apply the same code to all elements in a list

```python
>>> l = [1, 2, 3]
>>> print l[0]
>>> print l[1]
>>> print l[2]
```
for loops

for loops let you repeatedly apply the same code to all elements in a list

```python
>>> l = [1, 2, 3]
>>> for item in l:
...     print item
1
2
3
```
Also works for strings

DNA = 'actg'
for i in DNA:
    print i

a
c
act
g
for loop

- Allows you to perform an operation on each element in a list.

```python
for <target> in <object>:
    <statement>
    <statement>
    ...
    <statement>
```

- Must be defined in advance
- Variable name available inside loop
- Repeated block of code
- Must be indented
For loop examples:

DNA = 'AGTCGA'

base 0 is A
base 1 is G
base 2 is T
base 3 is C
base 4 is G
base 5 is A
Solution

>>> index = 0
>>> for base in DNA:
...   print "base", index, "is", base
...   index = index + 1
...
For loop examples:

```python
text_list = [2, 3, 4, 5]
sum of all numbers?
```
For loop examples:

```python
num_list = [2, 3, 4, 5]

sum of all numbers?

sum = 0
for v in num_list:
    sum = sum + v
print 'The sum is:', sum
```
The `range()` function returns a list of integers covering a specified range

\[
\text{range}([\text{start,}] \text{ stop [,step]}))
\]

- `range(5)`
  
  \[0, 1, 2, 3, 4\]

- `range(2,8)`
  
  \[2, 3, 4, 5, 6, 7\]

- `range(-1, 2)`
  
  \[-1, 0, 1\]

- `range(0, 8, 2)`
  
  \[0, 2, 4, 6\]

- `range(0, 8, 3)`
  
  \[0, 3, 6\]

- `range(6, 0, -1)`
  
  \[6, 5, 4, 3, 2, 1\]
Range can be useful in a for loop

```python
num_list = [0, 1, 2, 3, 4, 5, 6]
# print only every other number:
for i in range(0, 7, 2):
    print num_list[i]
```
Nested loops

nucs = 'ACGT'
for i in nucs:
    for j in nucs:
        print i, j
Terminating a loop

break # jumps out of the most internal loop

>>> for integer in range(0,3):
    if (integer == 1):
        break
    print integer
...

Output?
Terminating a loop

continue # skips to end of current
    # iteration of the loop

>>> for integer in range(0,3):
  ...   if (integer == 1):
  ...     continue

  ...   print integer

...
Sample problem

Write a program `add-arguments.py` that reads any number of integers from the command line and prints the cumulative total for each successive argument.

```
> python add-arguments.py 1 2 3
1
3
6

> python add-arguments.py 1 4 -1
1
5
4
```