Quiz Section Week 5
April 26, 2016

Review

Programming: Matrices, files, more on functions, organizing programs

Topics (not guaranteed to be comprehensive!)

- Alignments
  - Reasons to align sequences
  - Needleman-Wunsch algorithm
  - Smith-Waterman algorithm
  - Effects of parameter variation (including gap penalties)
  - Testing for statistical significance of an alignment
- Phylogenetic trees
  - Rooted and unrooted topologies
  - Defining the best tree with UPGMA and Neighbor Joining
  - Concept of parsimony
  - Fitch algorithm: quantifying how parsimonious a tree is, assigning internal states
  - Finding the most parsimonious tree: Hill climbing w/ Nearest-Neighbor interchanges
  - Bootstrapping to quantify confidence in tree partitions
- Clustering
  - Defining a clustering problem
  - Hierarchical clustering
    - Impact of using single/complete/average linkage
  - K-means: Objective and algorithm
- General computation and programming
  - What is an algorithm
  - What is a search heuristic
  - Conceptual definitions of variable and function
  - Algorithm complexity with O(n) notation
  - Data types and converting between them
  - Program flow and control with conditional statements and loops

Phylogenetic trees

UPGMA/Neighbor Joining

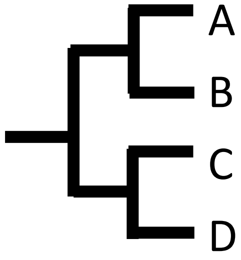
- Define the best tree: based on distance between leaves
- Find the best tree using: polynomial time algorithm to construct the best tree from a distance matrix

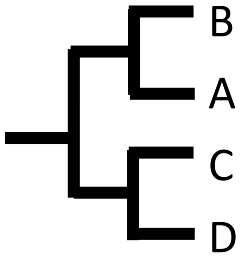
Parsimony approach

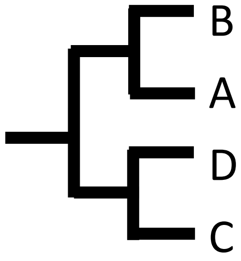
- Define the best tree: Minimum # of mutations required to traverse tree
- Find the best tree: by enumerating all trees (exhaustive search), or by heuristic approach like Nearest-Neighbor Interchange Hill-Climbing

Tree topologies

Are these the same tree?

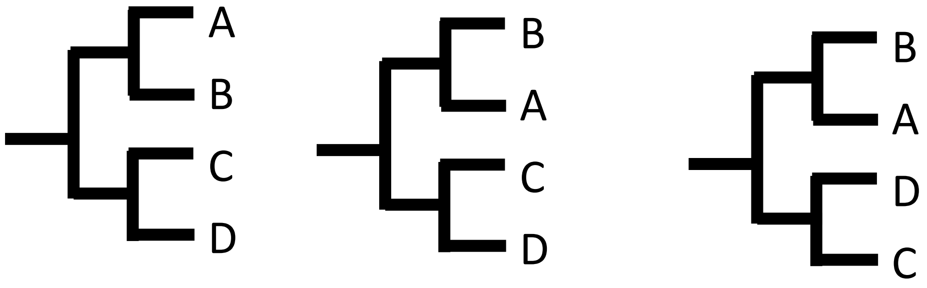




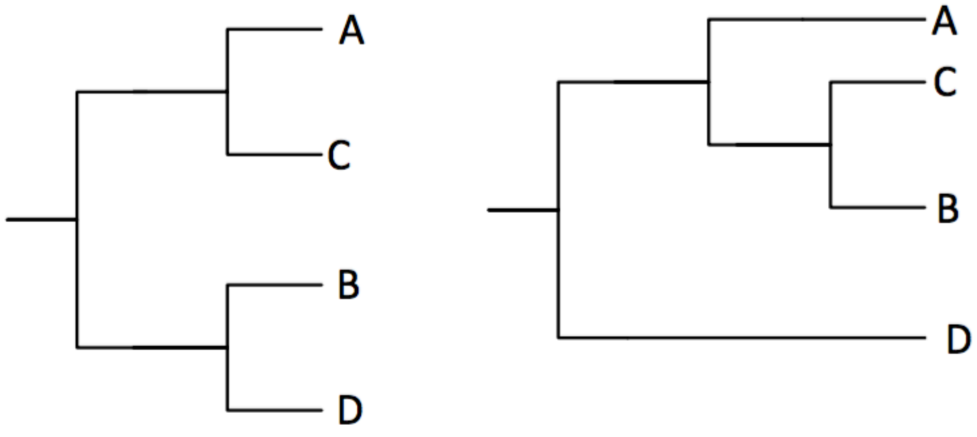


Tree topologies

Are these the same tree?



How about these?

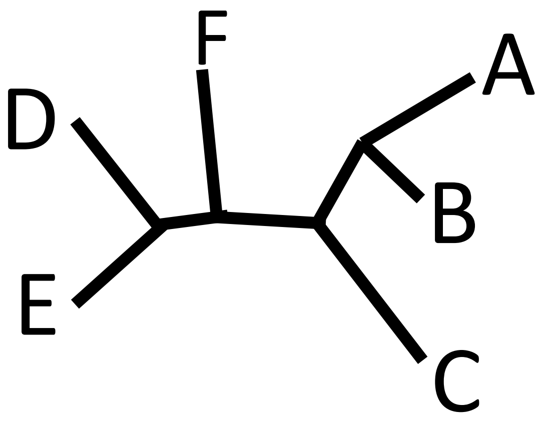


Counting tree topologies

For N leaves

# of unrooted topologies = 3*5*7*…*(2N-5)
# of branches = 2N-3

E.g. an unrooted tree with 6 nodes



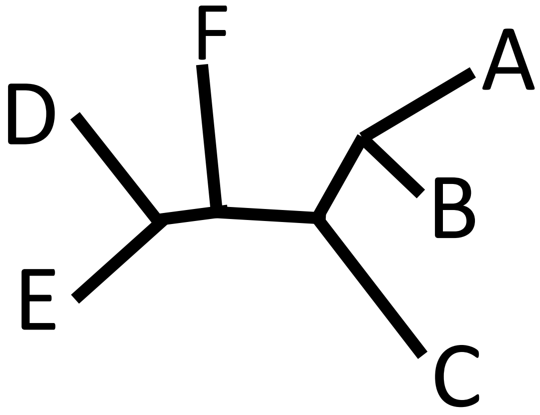
How many different topologies?

Counting tree topologies

For N leaves

# of unrooted topologies = 3*5*7*…*(2N-5)
# of branches = 2N-3

E.g. an unrooted tree with 6 nodes



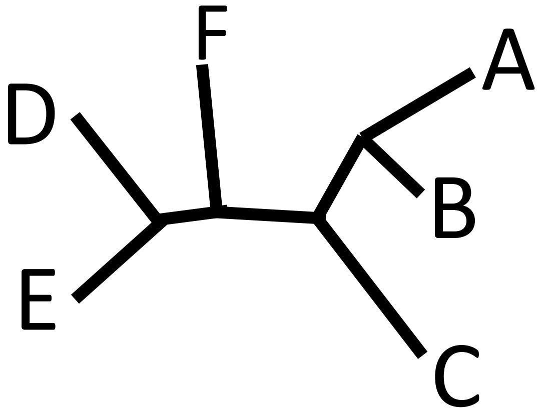
How many different topologies?
3*5*7 = **105**

Counting tree topologies

For N leaves

# of unrooted topologies = 3*5*7*…*(2N-5)
# of branches = 2N-3

E.g. an unrooted tree with 6 nodes



How many different topologies?
3*5*7 = **105**

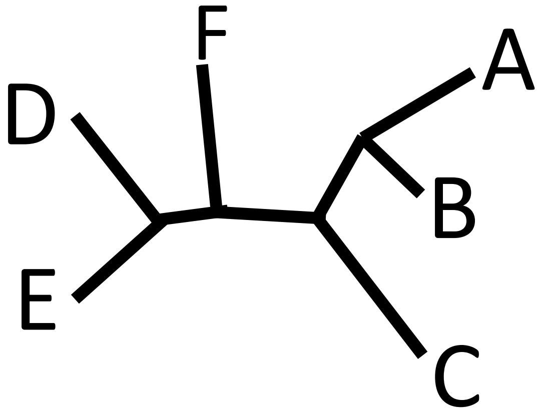
How many branches?

Counting tree topologies

For N leaves

# of unrooted topologies = 3*5*7*…*(2N-5)
# of branches = 2N-3

E.g. an unrooted tree with 6 nodes



How many different topologies?
3*5*7 = **105**

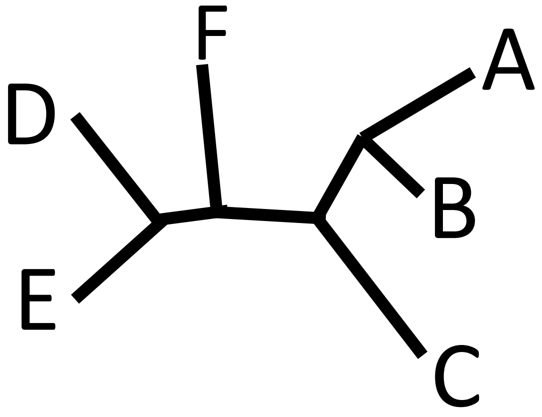
How many branches?
2N-3 = **9**

Counting tree topologies

For N leaves

# of unrooted topologies = 3*5*7*…*(2N-5)
# of branches = 2N-3

E.g. an unrooted tree with 6 nodes

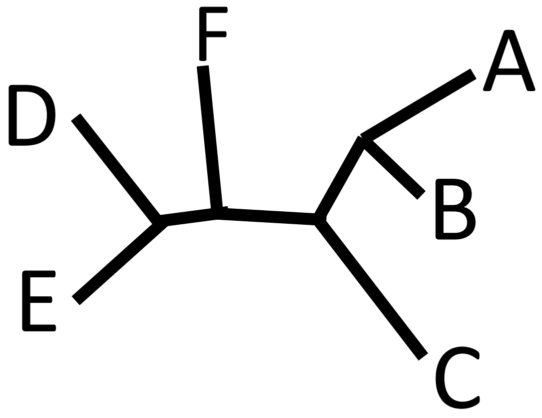


How many different topologies?
3*5*7 = **105**

How many branches?
2N-3 = **9**

The root could be placed on any branch

E.g. an unrooted tree with 6 nodes

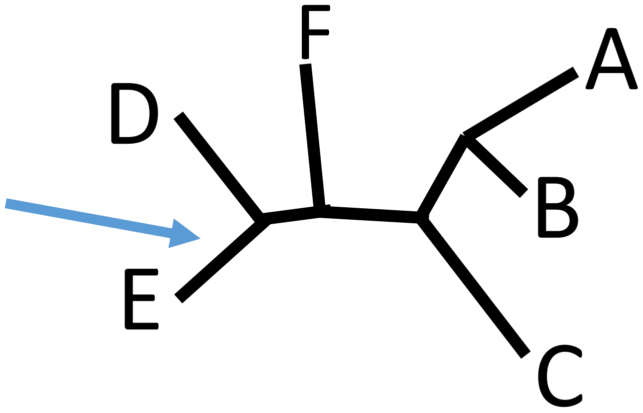


How many different topologies?

3*5*7 = **105**

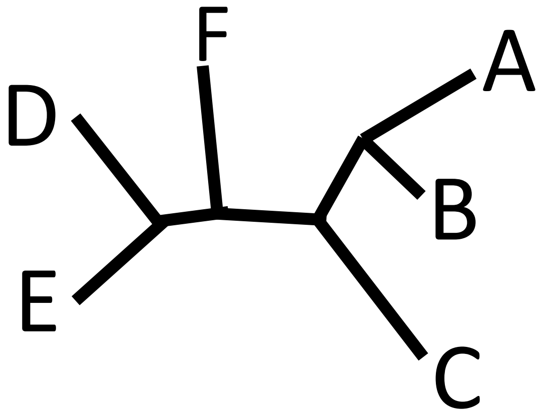
How many branches?

2N-3 = **9**



The root could be placed on any branch

E.g. an unrooted tree with 6 nodes

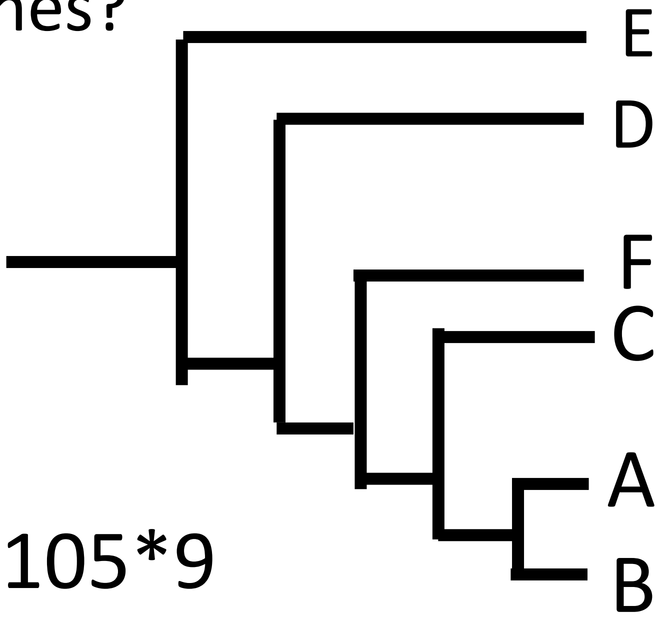


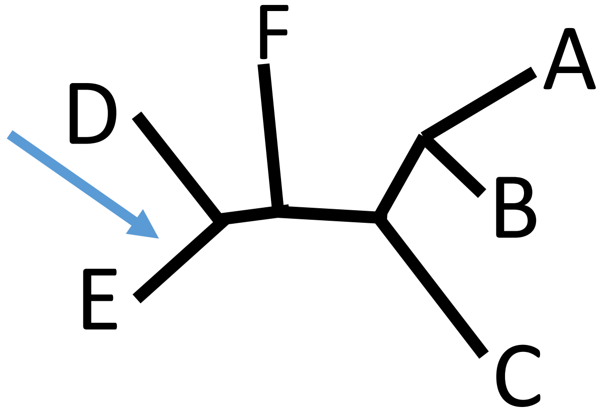
How many different topologies?

3*5*7 = **105**

How many branches?

2N-3 = **9**

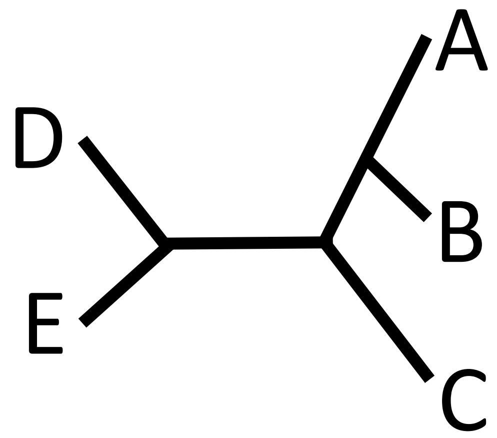




Total options = 105*9

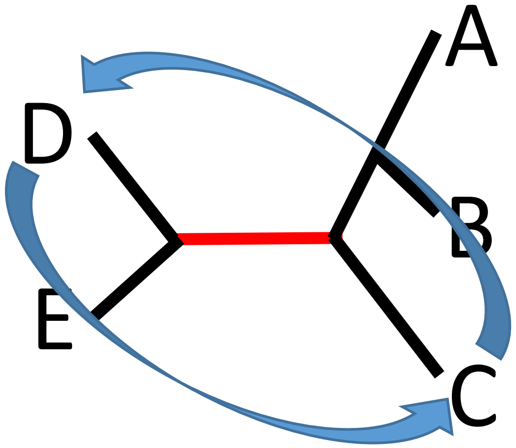
= 945

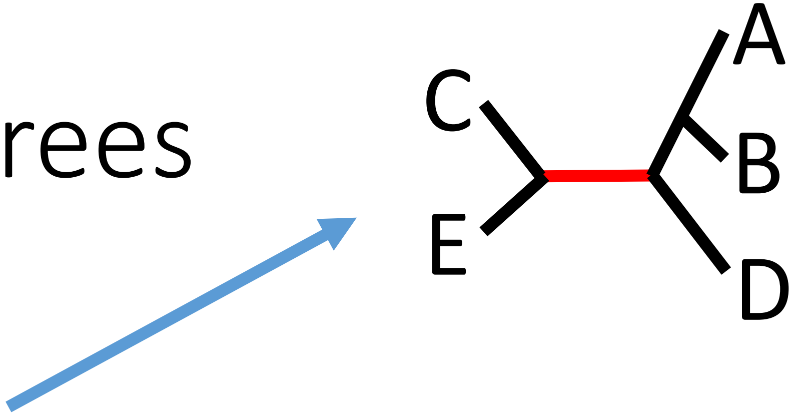
Nearest Neighbor Interchange trees



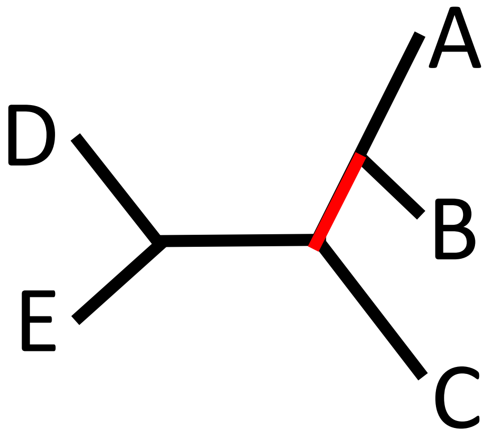
For each internal branch generate two variant trees that swap the relationships of the four outside branches

Nearest Neighbor Interchange trees



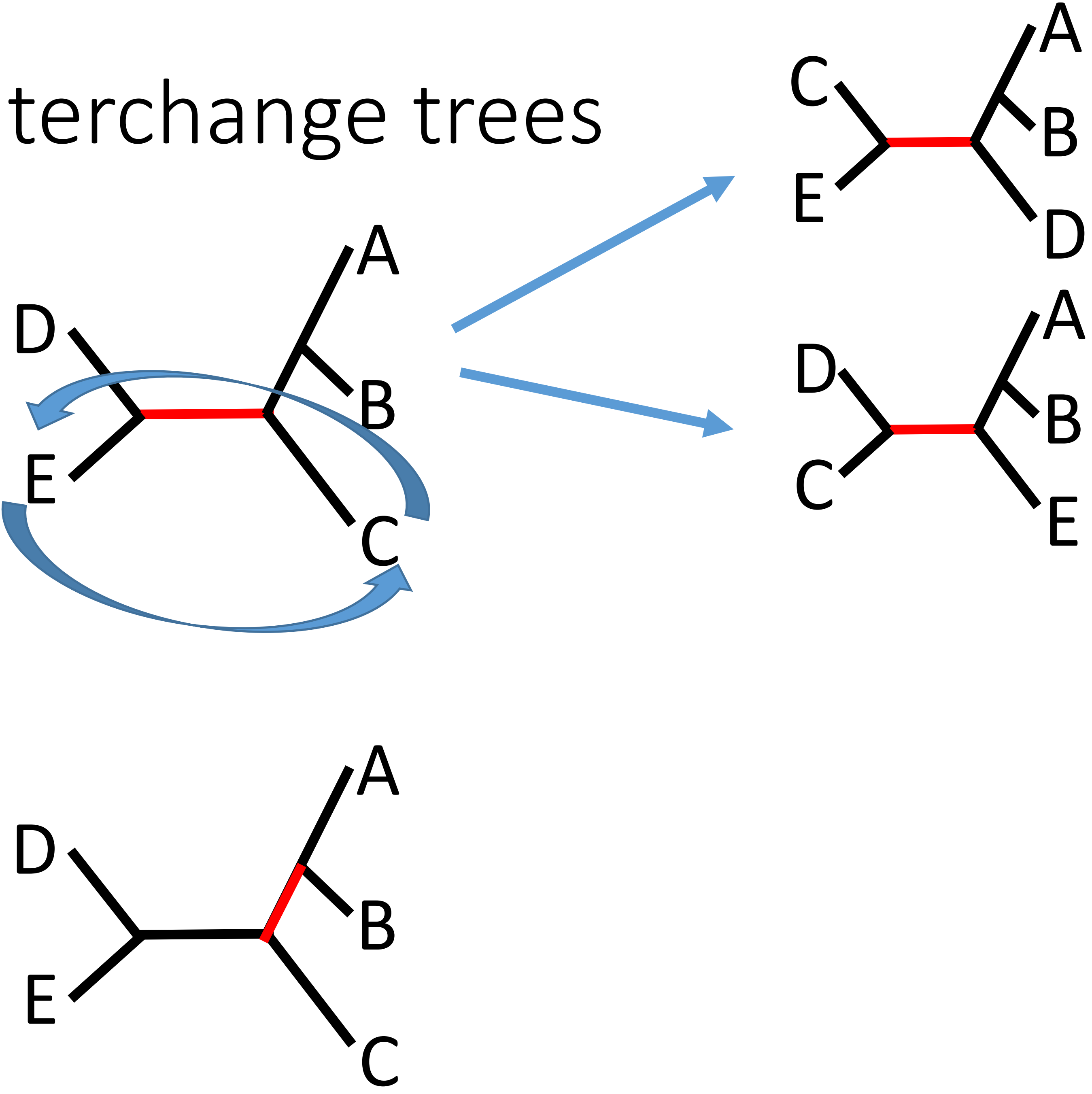


For each <span style="color:red">internal branch</span> generate two variant trees that swap the relationships of the four outside branches



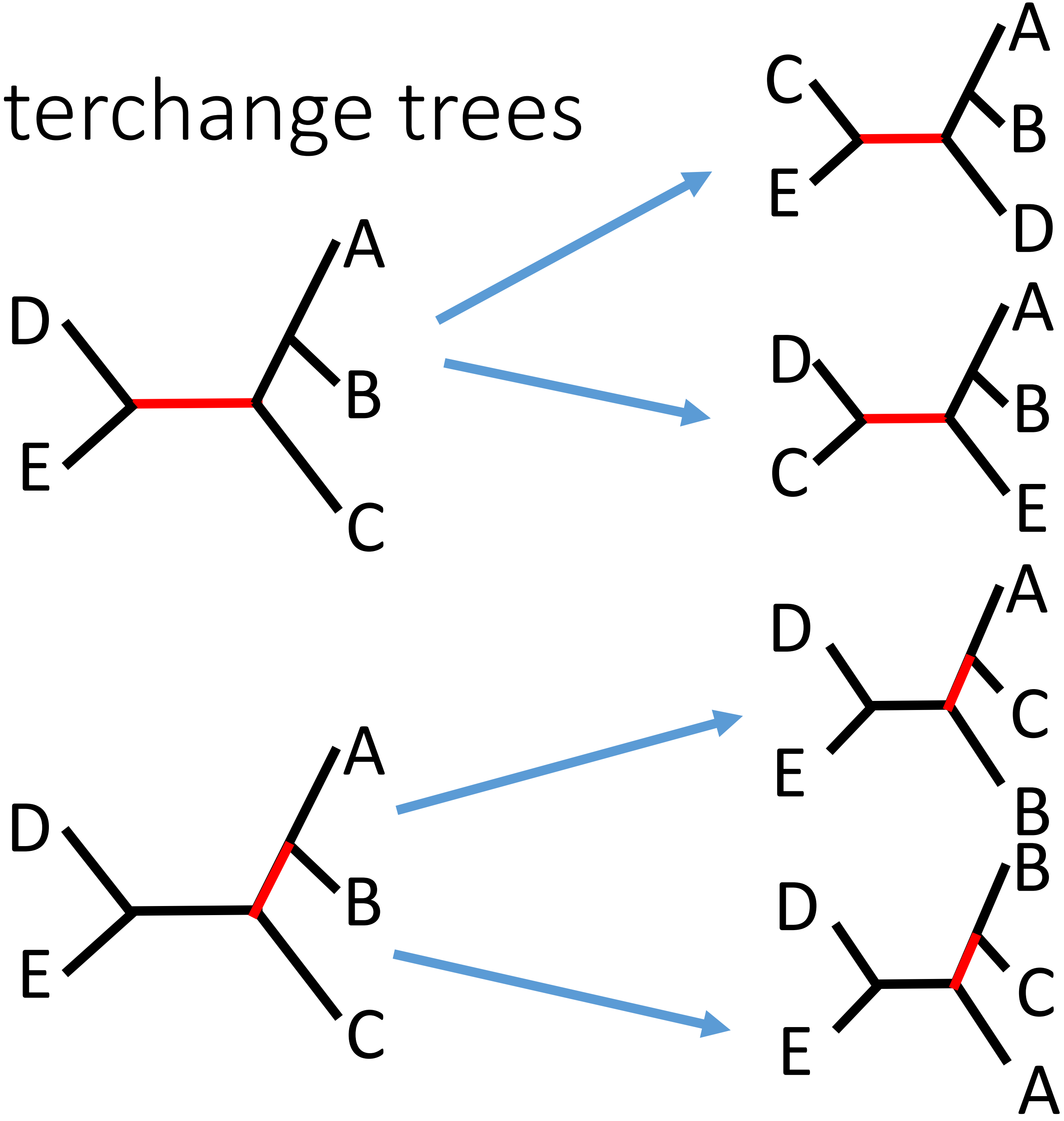
Nearest Neighbor Interchange trees

For each internal branch generate two variant trees that swap the relationships of the four outside branches



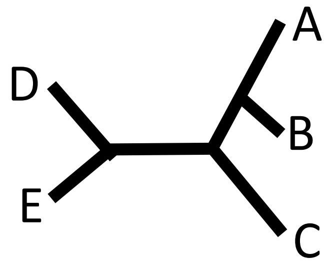
Nearest Neighbor Interchange trees

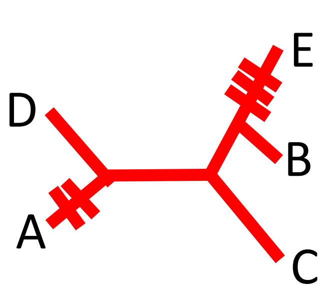
For each internal branch generate two variant trees that swap the relationships of the four outside branches



Nearest Neighbor interchange keeps you from stepping too far in hill-climbing

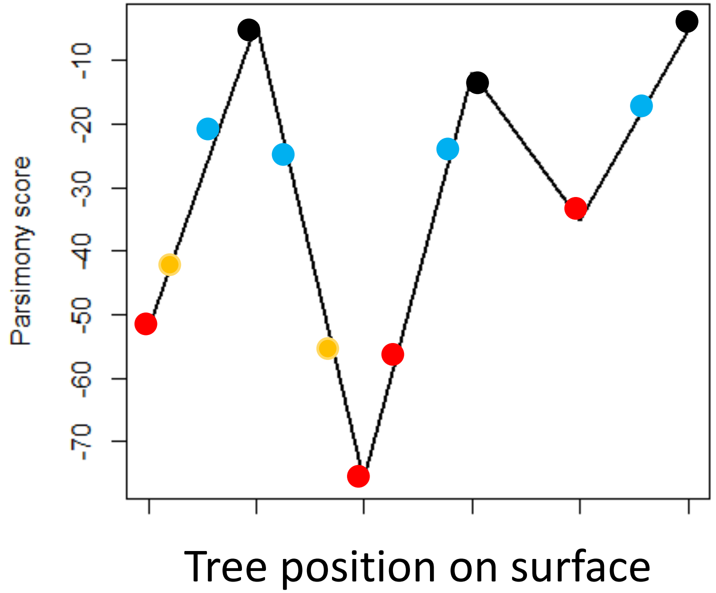
Clades A and B are very closely related



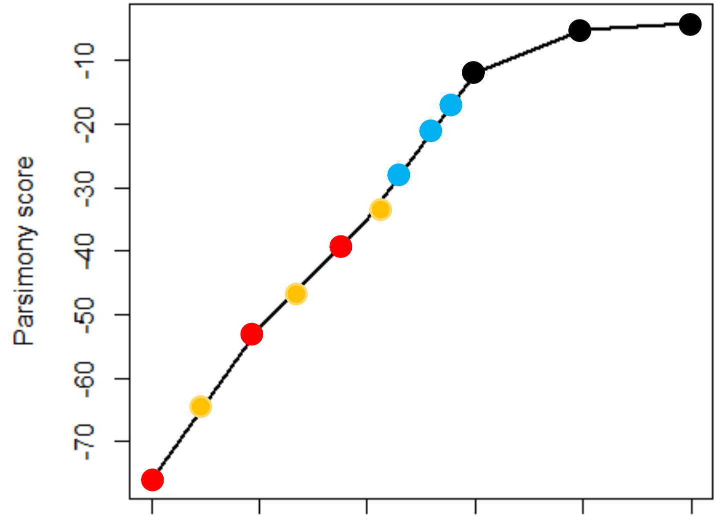


Tree	Negative parsimony Score
1	-12
2	-4
3	-5
4	-76
5	-52
6	-30

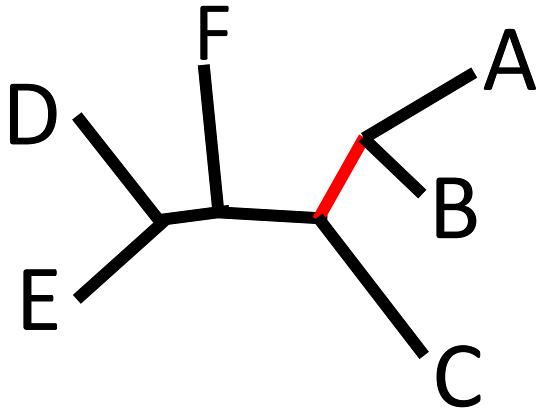
Random designation of neighbors



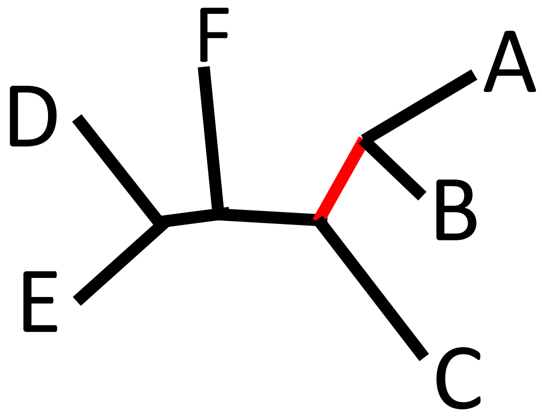
NNI designation

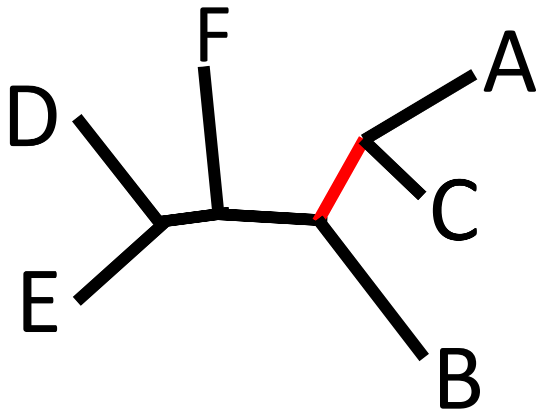


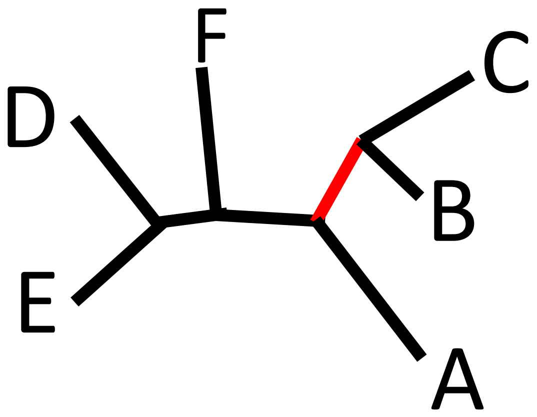
NN Practice: Draw both interchanges from swapping this branch



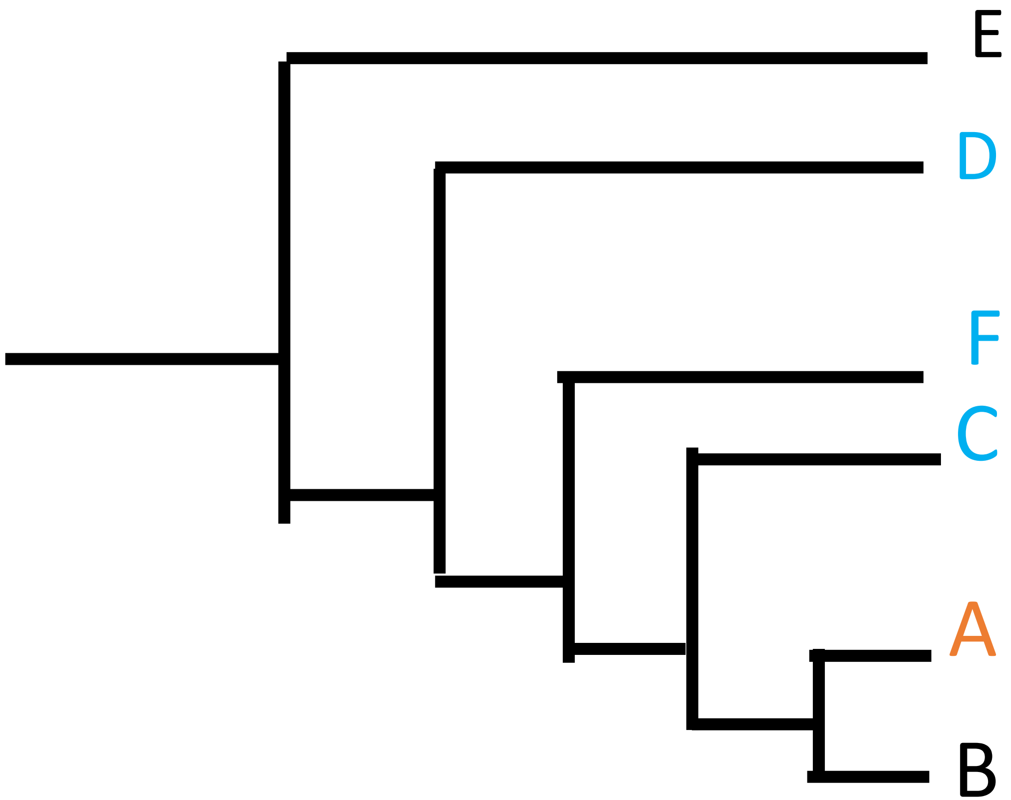
NN Practice: Draw both interchanges from swapping this branch



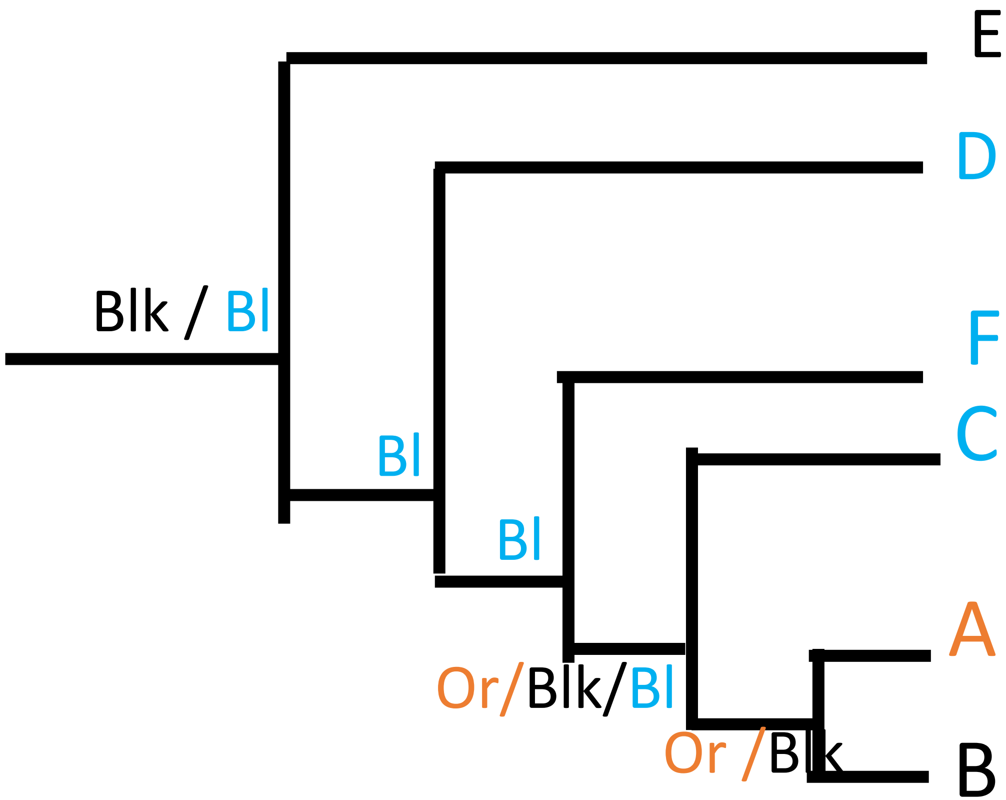




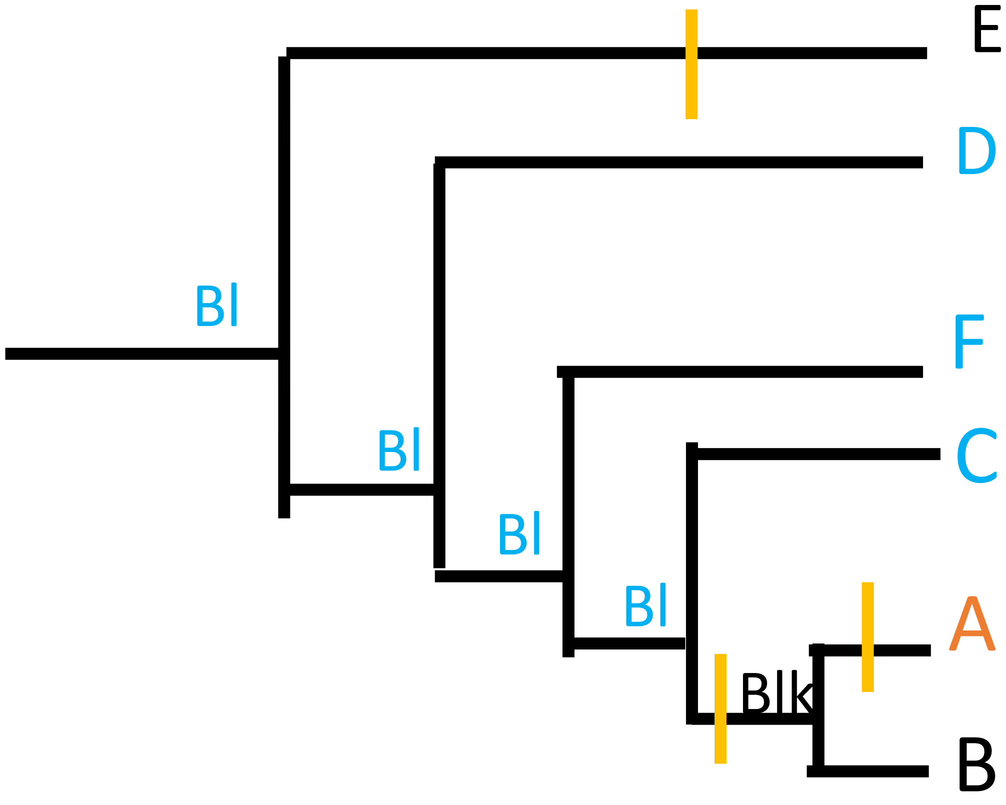
Fitch algorithm practice



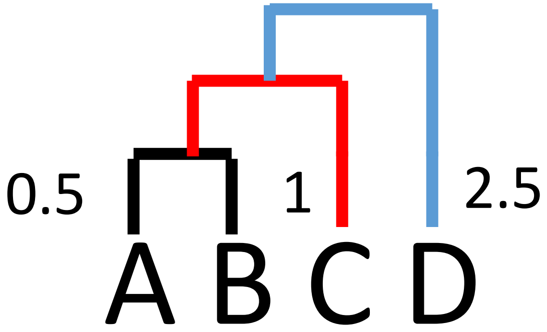
Fitch algorithm practice: bottom-up



Fitch algorithm practice: top-down



Hierarchical clustering with complete linkage example



	A	B	C	D
A	0	1	2	4
B	1	0	2	5
C	2	2	0	5
D	4	5	5	0

	A,B	C	D
A,B	0	2	5
C	2	0	5
D	5	5	0

	A,B,C	D
A,B,C	0	5
D	5	0

Programming note: 2D matrices in Python

- List of lists!
- Each row is a different list

	A	B	C	D
A	0	1	2	4
B	1	0	2	5
C	2	2	0	5
D	4	5	5	0

```
matrix = [ [0, 1, 2, 4], [1, 0, 2, 5], …]
print matrix[0][1]
1
print matrix[0]
[0, 1, 2, 4]
```

Reminder: “Big O” notation for complexity
What is the time complexity in O() to compute the sum of a list?

```
# x is a list of length N
sum = 0
for v in x:
    sum = sum + v
print 'The sum is:', sum
```

Directly proportional to # of items in list! `O(N)`

How about the time complexity in O() to compute the sum of an NxN matrix?

```
# x is a list of N lists
# each list has N elements
sum = 0
for row in x: # Do this N times
    for v in row: # N times again
        sum = sum + v
print 'The sum is:', sum
# The answer is O(N^2)
```

Given a list of 2D points, compute their center

```
points = [ (1,2), (3,4), (5,6), (7,8) ]
# center point is ( mean_x, mean_y )
mean_x = 0.0
mean_y = 0.0
for i in range(0,len(points)):
     mean_x += points[i][0]
     mean_y += points[i][1]

center = ( mean_x/len(points), mean_y/len(points) )
print center
```

```
(4.0, 5.0)
```

Reading data from a file in Python

```
fin = open('qs5.txt', 'r') # ‘r’ stands for
‘read’
all_lines = []
for line in fin: # In a for loop, fin acts
like a list of strings
    print line
    all_lines.append(line)
fin.close() # Lets the computer know it can
free up resources used to read the file
print all_lines
```

Alternative file-reading structure

```
my_open_file = open(sys.argv[1])
s1 = my_open_file.readline().strip()
s2 = my_open_file.readline().strip()
```

Note: if in a different directory, have to supply **file path**

Writing data to a file

```
fout = open('output.txt', 'w') # ‘w’
stands for ‘write)
fout.write('Hello! How')
fout.write(' are you?\nI'm fine.') #
‘\n’ starts a
new line
fout.close()
```

Useful function: Split a string into its constituent words

```
s = 'Wherefore art thou Romeo?'
words = s.split()# Returns a list of substrings
print words
['Wherefore', 'art', 'thou', 'Romeo?']

# split() can use any arbitrary string to split by
words = s.split('r')
print words
['Whe', 'efo', 'e a', 't thou Romeo?']
```

One way to structure your program

python analyze_sequence_pairs.py inputfile.txt outputfile.txt

```
fin = open(sys.argv[1],'r')
fout = open(sys.argv[2],'w')
seqs = []
for line in fin:
     seqs.append(line.rstrip()) # gets rid of \n at the end of
the line
print seqs
answer = calculate_jukes_cantor(seqs[0], seqs[1])
fout.write( seqs[0] + ' ' + seqs[1] + ' ')
fout.write( str(answer) + '\n')
fin.close()
fout.close()
```

More on functions

Providing default values for arguments

```
  def less_than(myList, num=4):
       new_list = []
       for x in myList:
            if x < num:

       new_list.append(x)
       return new_list

>>> less_than([12,3,7]) # will use default value for num
   [3]
>>> less_than([12,3,7], num = 8)

   [3,7]
```

Scope of a variable

- Variables created in the main part of your program can be accessed anywhere (**global** scope)
- Variables created within functions are only accessible within that function (**local** scope)

```
new_list = [0,1,2]

def less_than(myList, num = 4):
      #new_list = []
      for x in myList:
            if x < num:
                  new_list.append(x)
      return new_list

print new_list #Error
anotherList = [3,7,12]
print less_than(anotherList)
```

Don't do this!! You'll
confuse yourself
Define all your functions at
the beginning of your
program, use local variables

Exercise if time: Save these sequences as a file, read it in, calculate # of As,Ts,Cs,Gs in each sequence

ATGGGGGACTACTGGGGGTTCCCCC
CTGACTTTTAGTACGTCATGGCATA