“CS 5 is good for your health!”

We’re currently covering Chapter 2 in the book. Sections 2.1-2.6 are Python “basics” and Sections 2.7-2.11 are about recursion.
“CS 5 is good for your health!”
Don’t worry, be happy...
Recursion warmup (in your notes)!

```python
>>> count(42, [1, [2, 42], 42, [[[[42]]]] ])
1

>>> deepCount(42, [1, [2, 42], 42, [[[[42]]]] ])
3

if type(blah) == list:  ...
```

Hey, look at me! I’m hiding really deep inside this list!
Notice that `deepCount` does NOT call `count`. It calls itself!
The idea... Given a list like L = [42, 57, 1, 3]
- Slice off the first element. Now we have a shorter list... [57, 1, 3]
- Use recursion to sort that list. Now we have... [1, 3, 57]
- Now, insert L[0] (Which is 42) into the right place in [1, 3, 57]...
  [1, 3, 42, 57]

```python
def insert(x, sortedList):
    ''' Takes a number and sorted list as input and returns a new list that has x
    inserted into the right place in the sorted list '''
    if sortedList == []: return [x]
    elif x < sortedList[0]: return [x] + sortedList
    else: return [sortedList[0]] + insert(x, sortedList[1:])

def sort(myList):
    if myList == []: return []
    else: return insert(myList[0], sort(myList[1:]))
```

Try this in your worksheet!
Insertion Sorting

>>> sort([42, 57, 1, 3])
[1, 3, 42, 57]

The idea... Given a list like L = [42, 57, 1, 3]
• Slice off the first element. Now we have a shorter list... [57, 1, 3]
• Use recursion to sort that list. Now we have... [1, 3, 57]
• Now, insert L[0] (Which is 42) into the right place in [1, 3, 57]...
  [1, 3, 42, 57]

def sort(myList):
    if myList == []: return []
    else: return insert(myList[0], sort(myList[1:]))

def insert(x, sortedList):
    ''' Takes a number and sorted list as input and returns a new list that has x
    inserted into the right place in the sorted list `'''
    if sortedList == []: return [x]
    elif x < sortedList[0]: return [x] + sortedList
    else: return [sortedList[0]] + insert(x, sortedList[1:])
Insertion Sorting

```python
>>> sort([42, 57, 1, 3])
[1, 3, 42, 57]
```

The idea... Given a list like \( L = [42, 57, 1, 3] \)

- Slice off the first element. Now we have a shorter list... \( [57, 1, 3] \)
- Use recursion to sort that list. Now we have... \( [1, 3, 57] \)
- Now, insert \( L[0] \) (Which is 42) into the right place in \( [1, 3, 57] \)... \( [1, 3, 42, 57] \)

```python
def sort(myList):
    if myList == []: return []
    else: return insert(L[0], sort(myList[1:]))

def insert(x, sortedList):
    ''' Takes a number and sorted list as input and returns a new list that has x inserted into the right place in the sorted list '''
    if sortedList == []: return [x]
    elif x < sortedList[0]: return [x] + sortedList
    else: return [sortedList[0]] + insert(x, sortedList[1:])
```

Choose a set of pegs of maximum total value, without choosing two adjacent pegs

>>> pegs([10, 3, 5, 15, 4])
25
def pegs(pegList):
    ''' Takes a list of numbers as input and returns a number representing the maximum score for that list in the pegs game. '''
    if pegList == []: return 0 # base case
    else:
        It = pegList[0]
        useIt = It + pegs(pegList[2:])
        loseIt = pegs(pegList[1:])
    return max(useIt, loseIt)
“Care Packages”

```python
>>> pegs([10, 3, 5, 15, 4])
25
```

```python
>>> pegsSolution([10, 3, 5, 15, 4])
[25, [10, 15]]
```

This is a list whose first element is a number and the second element is a list. We’ll call this kind of thing a “care package”
```python
def pegs(pegList):
    ''' Takes a list of numbers as input and returns a number representing the maximum score for that list in the pegs game. '''
    if pegList == []: return 0 # base case
    else:
        It = pegList[0]
        useIt = It + pegs(pegList[2:]
        loseIt = pegs(pegList[1:]
        return max(useIt, loseIt)

def pegsSolution(pegList):
    ''' Takes a list of numbers as input and returns a care package of the form [score, pegs] where pegs is a list of pegs that give that score '''
    if pegList == []: return _______
    else:
```

def pegs(pegList):
    ''' Takes a list of numbers as input and returns a number representing the maximum score for that list in the pegs game. '''
    if pegList == []: return 0 # base case
    else:
        It = pegList[0]
        useIt = It + pegs(pegList[2:])
        loseIt = pegs(pegList[1:])
        return max(useIt, loseIt)

def pegsSolution(pegList):
    ''' Takes a list of numbers as input and returns a care package of the form [score, pegs] where pegs is a list of pegs that give that score '''
    if pegList == []: return [0, []]
    else:
def pegs(pegList):
    ''' Takes a list of numbers as input and returns a number representing the maximum score for that list in the pegs game. '''
    if pegList == []: return 0 # base case
    else:
        It = pegList[0]
        useIt = It + pegs(pegList[2:])
        loseIt = pegs(pegList[1:])
        return max(useIt, loseIt)

def pegsSolution(pegList):
    ''' Takes a list of numbers as input and returns a care package of the form [score, pegs] where pegs is a list of pegs that give that score '''
    if pegList == []: return [0, []]
    else:
        It = pegList[0]
        option = pegsSolution(pegList[2:])
        score = option[0]
        pegs = option[1]
        useIt = ________________________________
    Be careful to call pegsSolution and not pegs!
```python
def pegs(pegList):
    ''' Takes a list of numbers as input and returns a number representing the maximum score for that list in the pegs game. '''
    if pegList == []: return 0 # base case
    else:
        It = pegList[0]
        useIt = It + pegs(pegList[2::])
        loseIt = pegs(pegList[1::])
        return max(useIt, loseIt)

def pegsSolution(pegList):
    ''' Takes a list of numbers as input and returns a care package of the form [score, pegs] where pegs is a list of pegs that give that score '''
    if pegList == []: return [0, []]
    else:
        It = pegList[0]
        option = pegsSolution(pegList[2::])
        score = option[0]
        pegs = option[1]
        useIt = [It + score, [It] + pegs]
        loseIt = ____________________________
        return useIt
```

Be careful to call pegsSolution and not pegs!
def pegs(pegList):
    ''' Takes a list of numbers as input and returns a number representing the maximum score for that list in the pegs game. '''
    if pegList == []: return 0  # base case
    else:
        It = pegList[0]
        useIt = It + pegs(pegList[2:])
        loseIt = pegs(pegList[1:])
        return max(useIt, loseIt)

def pegsSolution(pegList):
    ''' Takes a list of numbers as input and returns a care package of the form [score, pegs] where pegs is a list of pegs that give that score '''
    if pegList == []: return [0, []]
    else:
        It = pegList[0]
        option = pegsSolution(pegList[2:])
        score = option[0]
        pegs = option[1]
        useIt = [It + score, [It] + pegs]
        loseIt = pegsSolution(pegList[1:])
        if useIt[0] > loseIt[0]: return useIt
        else: return loseIt

Be careful to call pegsSolution and not pegs!
Comparing DNA with Longest Common Subsequence (LCS)

\[ \text{AGGACAT} \]
\[ \text{ATTACGAT} \]

\[ \text{>>> LCS(“AGGACAT”, “ATTACGAT”) } \]
\[ 5 \]

\[ \text{LCS(“spam”, “pims”) } \]

\[ \text{EGS = "GTACGTCGATAACTG"} \]
\[ \text{WGS = "TGATCGTCATAACGT"} \]

Schlitz gene
Try writing LCS

```python
def LCS(String1, String2):

# Base case(s)?
# First symbols match?
# Otherwise?
max(x, y) is built-in
```

If you happen to finish early, try writing `getLCS`...

```python
>>> getLCS("AGGACAT", "ATTACGAT")
[5, "AACAT"]
```
Try writing LCS

```python
def LCS(String1, String2):
    if String1 == "" or String2 == "": return 0
    elif String1[0] == String2[0):
        return 1 + LCS(String1[1:], String2[1:])
    else:
        option1 = LCS(String1, String2[1:])
        option2 = LCS(String1[1:], String2)
        return max(option1, option2)
```

Base case(s)?
First symbols match?
Otherwise?
\[ \text{max}(x, y) \text{ is built-in} \]
Try writing LCS

```python
>>> LCS("AGGACAT", "ATTACGAT")
5

def LCS(String1, String2):
    if String1 == "" or String2 == "": return 0
    elif String1[0] == String2[0]:
        return 1 + LCS(String1[1:], String2[1:])
    else:
        option1 = LCS(String1, String2[1:])
        option2 = LCS(String1[1:], String2)
        return max(option1, option2)
```

Base case(s)?
First symbols match?
Otherwise?
\[\text{max}(x, y)\text{ is built-in}\]
Try writing LCS

```python
def LCS(String1, String2):
    if String1 == "" or String2 == "": return 0
    elif String1[0] == String2[0]:
        return 1 + LCS(String1[1:], String2[1:])
    else: return LCS(String[1:], String2[1:])
```

This is a not-quite-right solution!
Try writing LCS

>>> LCS("AGGACAT", "ATTACGAT")
5

def LCS(String1, String2):
    if String1 == "" or String2 == "": return 0
    elif String1[0] == String2[0]:
        return 1 + LCS(String1[1:], String2[1:])
    else: return LCS(String1[1:], String2[1:])

LCS("spam", "pam") ->
Try writing LCS

```python
>>> LCS("AGGACAT", "ATTACGAT")
5

def LCS(String1, String2):
    if String1 == "" or String2 == "": return 0
    elif String1[0] == String2[0]:
        return 1 + LCS(String1[1:], String2[1:])
    else: return LCS(String1[1:], String2[1:])

LCS("spam", "pam") -> LCS("pam", "am") ->
LCS("am", "m") -> LCS("a", "") -> 0
```
>>> LCS("AGGACAT", "ATTACGAT")
5

def LCS(String1, String2):
    if String1 == "" or String2 == "": return 0
    elif String1[0] == String2[0]:
        return 1 + LCS(String1[1:], String2[1:])
    else:
        option1 =
        option2 =
        return max(option1, option2)
def LCS(String1, String2):
    if String1 == "" or String2 == "": return 0
    elif String1[0] == String2[0]:
        return 1 + LCS(String1[1:], String2[1:]
    else:
        option1 = LCS(String1, String2[1:])
        option2 = LCS(String1[1:], String2)
        return max(option1, option2)
The Sierpinski Fractal Triangle!