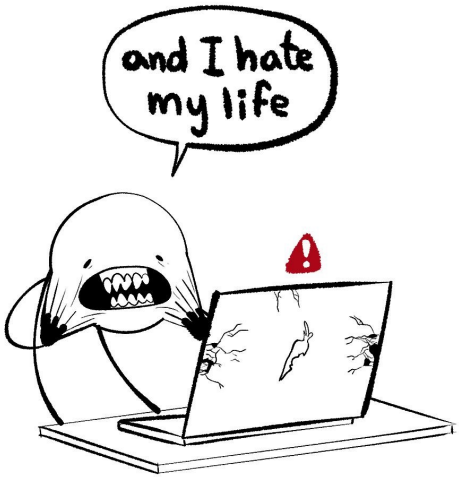
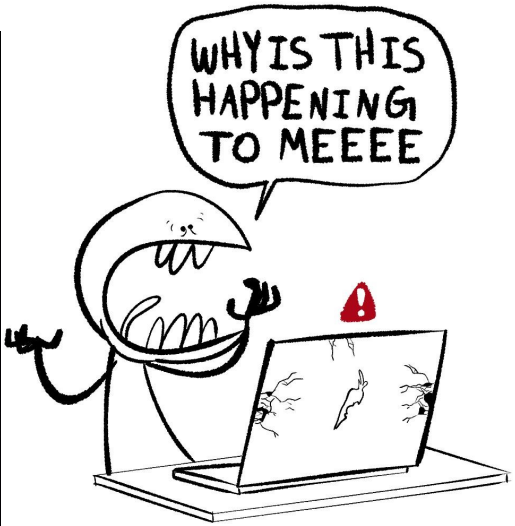
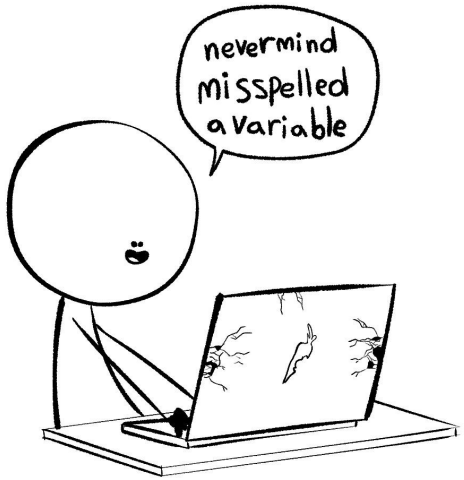
*me who just removed an error from my code

*my code:-

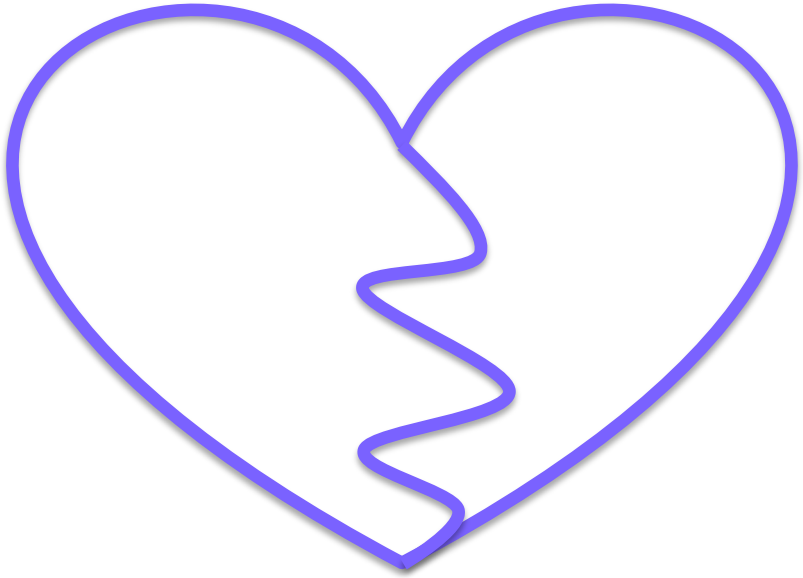






Reminders

It’s my last lecture!



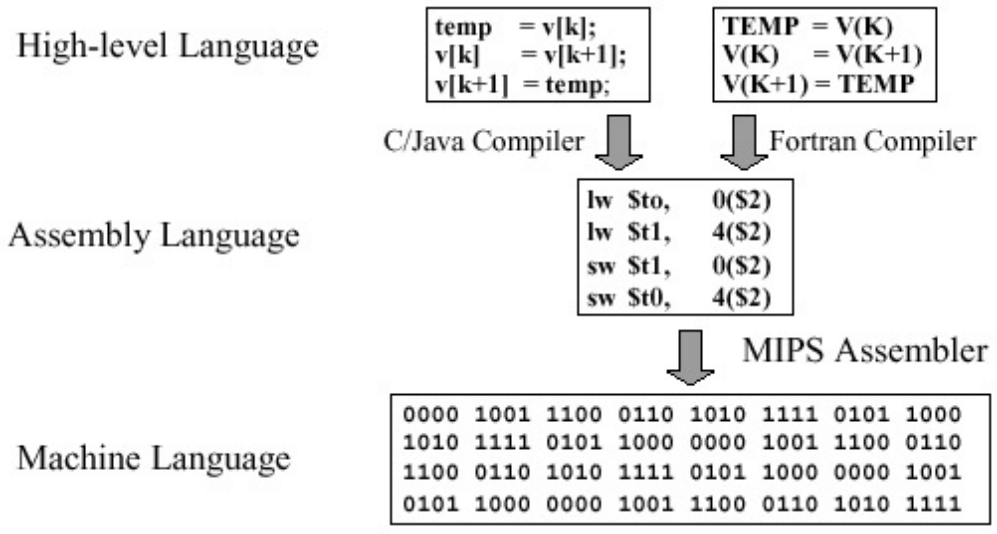
Reminders

Next week: Prof. Dodds comes to CS5 Green!

Python

```
>>> print("Hello World!")
Hello World!
```

Assembly

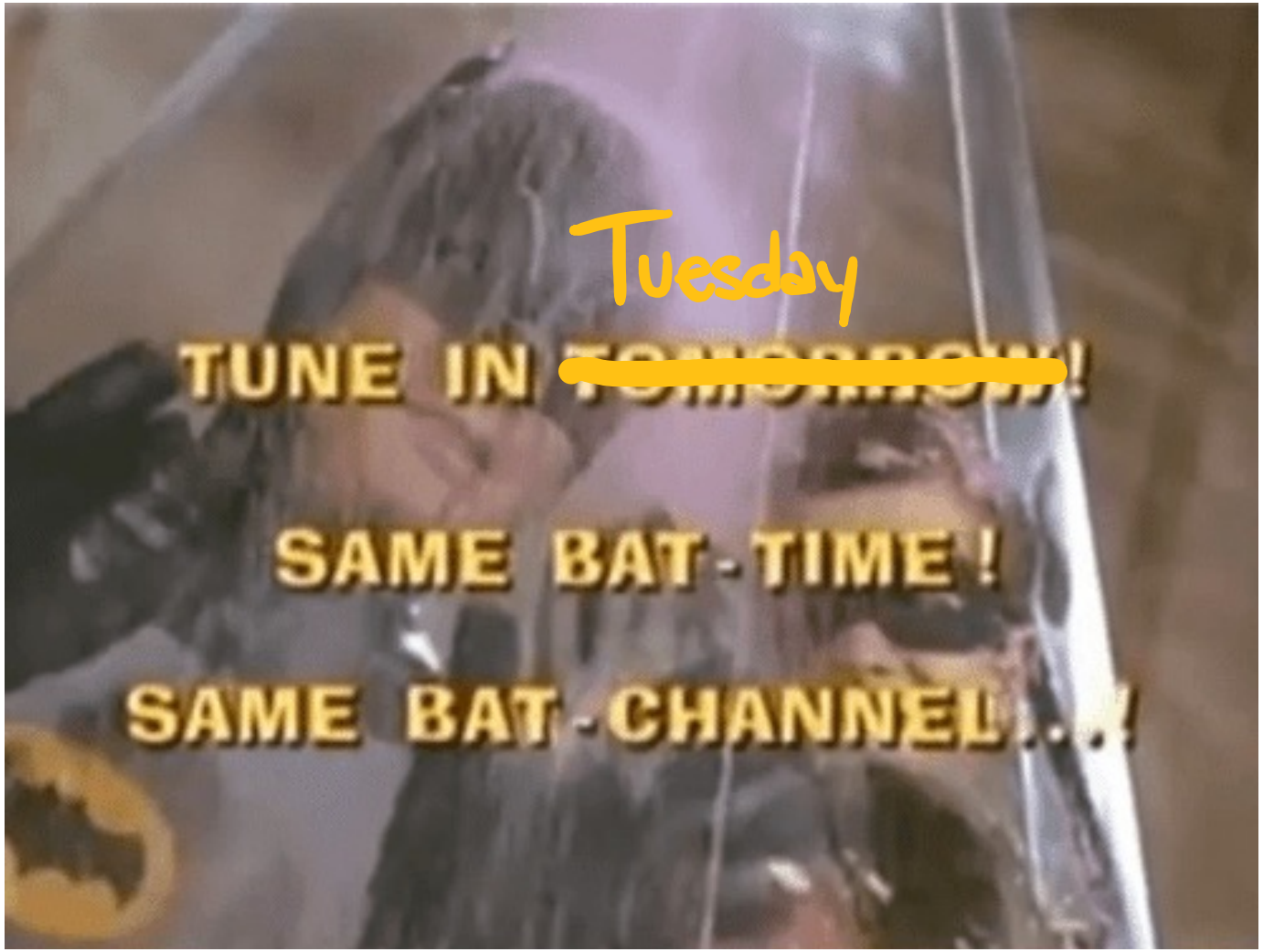


Shan 2460

https://www.cise.ufl.edu/~mssz/CompOrg/CDA-lang.html

Reminders

Next week: Prof. Dodds comes to CS5 Green!



Shan 2460

Reminders

It's my last lecture!

6	10/12/21 - Lec 12: Hmmm 1 (Z)	10/14/21 - Lec 13: Hmmm 2 (Z)	No HW over fall break	12
7	10/19/21 - Happy fall break!	10/21/21 - Lec 14: Recursion on Trees (E)	Homework 6/7	9
8	10/26/21 - Lec 15: UPGMA (E)	10/28/21 - Lec 16: More trees! (E)	Homework 8	10, 11
9	11/02/21 - Lec 17: RNA Folding (E)	11/04/21 - **Midterm**	Homework 9	
10	11/09/21 - Lec 18: Oops (E)	11/11/21 - Lec 19: Oops etc. (E)	Homework 10	CS For All Chapter 6
11	11/16/21 - Lec 20: Shapes! (E)	11/18/21 - Lec 21: Finishing up Oops (E)	Homework 11	CS For All Chapter 6
12	11/23/21 - Lec 22: Projects! (MJE)	11/25/21 - Happy Thanksgiving!	Project Descriptions	
13	11/30/21 - Theory 1 (G)	12/02/21 - Theory 2 (G)	Work on Projects	
14	12/07/21 - Theory 3 (G)	12/09/21 - Finale (MJE)	Work on projects	

CS 5 Green

Learning Goals

- Motivate the need for “care packages” in recursion

Care packages with `change`

```
def change(target, coinsL):
    '''Accepts an integer and a list as inputs. Returns the fewest
    number of coins needed to make the target integer.'''
    # base case 1: no coins required
    if target == 0: return 0

    # base case 2: impossible to make change
    elif coinsL == []: return float('inf')
    else:
        # discard coin if it exceeds the target
        if coinsL[0] > target:
            return change(target, coinsL[1:])

        # try both using and losing a coin; return minimum req'd
        else:
            useIt  = 1 + change(target - coinsL[0], coinsL)
            loseIt = change(target, coinsL[1:])
            return min(useIt, loseIt)
```

```
>>> change(42, [25, 21, 1])
2
>>> showChange(42, [25, 21, 1])
[2, [21, 21]]
```

showChange

Answer will be added to slides after class

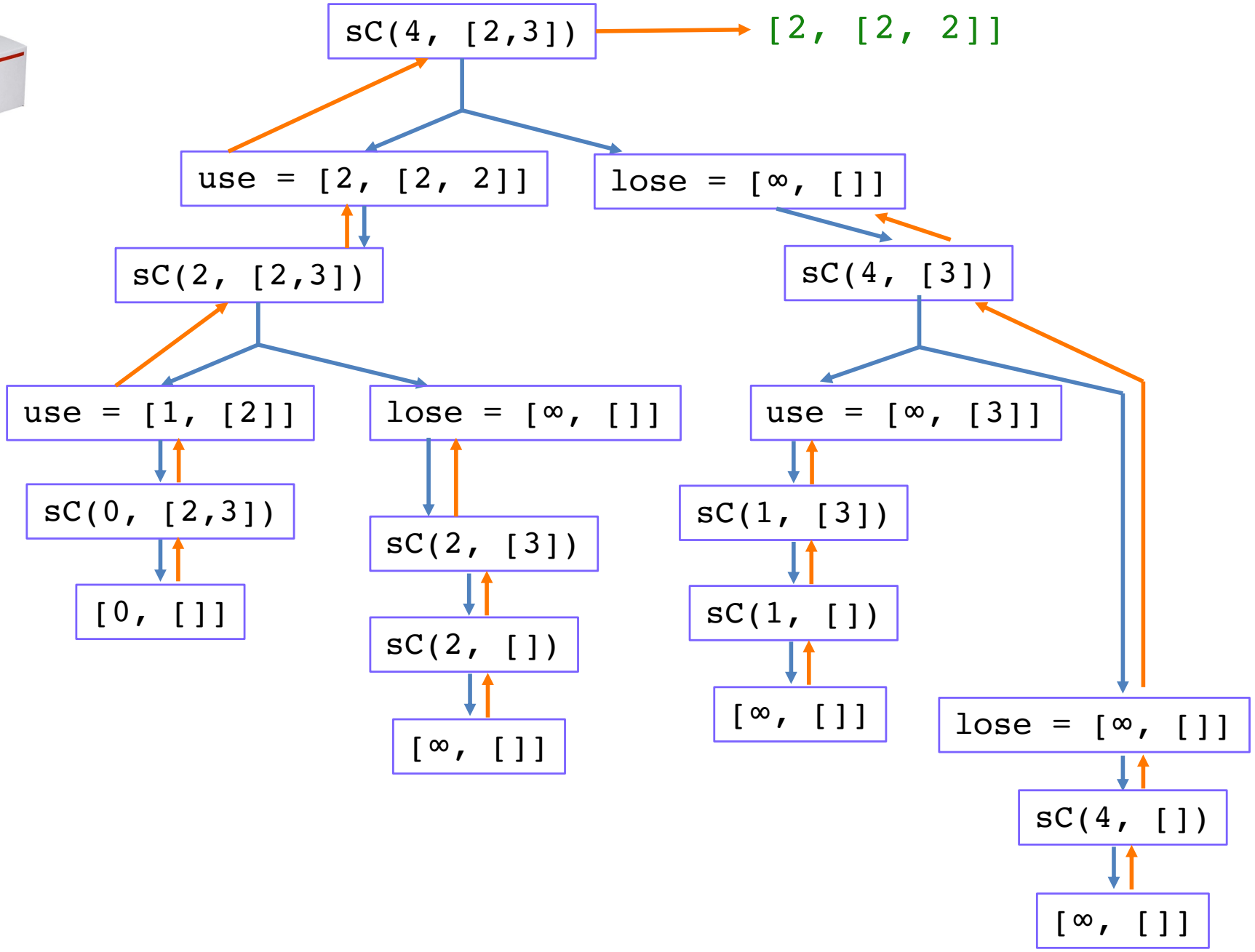
```
def showChange(target, coinsL):
    """Accepts a target integer and a list of coins as inputs. Returns a list
    containing two elements: the first element is the fewest number of coins
    required, and second element is a list of the actual coins used."""
```

showChange

Answer will be added to slides after class

```
def showChange(target, coinsL):
    """Accepts a target integer and a list of coins as inputs. Returns a list
    containing two elements: the first element is the fewest number of coins
    required, and second element is a list of the actual coins used."""
    # bc1: no coins needed
    if target == 0: return [0, []]

    # bc2: impossible to make change
    elif coinsL == []: return [float("inf"), []]

    # discard coin if it exceeds the target
    elif coinsL[0] > target: return showChange(target, coinsL[1:])

    else:
        # try using a coin
        useIt = showChange(target - coinsL[0], coinsL)
        useIt[0] += 1                  # update the number of coins used
        useIt[1].append(coinsL[0])     # update the list with the coin used

        # try losing a coin
        loseIt = showChange(target, coinsL[1:])

        # return the care package with the fewest coins used
        if useIt[0] < loseIt[0]: return useIt
        else: return loseIt
```


PRIORITY MAIL
PRIORITY MAIL


sC(4, [2,3])
[2, [2, 2]]
use = [2, [2, 2]]
lose = [∞, []]
sC(2, [2,3])
sC(4, [3])
use = [1, [2]]
lose = [∞, []]
use = [∞, [3]]
sC(0, [2,3])
sC(2, [3])
sC(1, [3])
[0, []]
sC(2, [])
sC(1, [])
[∞, []]
[∞, []]
lose = [∞, []]
sC(4, [])
[∞, []]

`change` and `showChange` return different `types`

```
>>> change(42, [25, 21, 1])
2


>>> showChange(42, [25, 21, 1])
[2, [21, 21]]
```

Another example with `subset`

```
def subset(target, inputL):
    """Accepts an integer and a list of integers as inputs. Returns a
    boolean indicating whether or not there exists a subset of numbers
    in the list that adds up to the target."""
    # bc1: if our target reaches 0, then a subset exists
    if target == 0: return True

    # bc2: if our list becomes empty, then a subset does not exist
    elif inputL == []: return False

    # discard a number if it exceeds the target; check remaining
    elif inputL[0] > target: return subset(target, inputL[1:])

    # otherwise, try both using and losing the first number
    else:
        useIt = subset(target - inputL[0], inputL[1:])
        loseIt = subset(target, inputL[1:])
        return useIt or loseIt
```

hw5pr1: `memoSubset` (10pts)

showSubset

worksheet


```
def showSubset(target, inputL):
    """Accepts an integer and a list of integers as inputs. Returns a two-item
    list: the first item is a boolean indicating whether or not there exists a
    subset of numbers in the list that adds up to the target, the second item is
    a list of the subset of numbers used."""
    # bc1:
    if target == 0:

    # bc2:
    elif inputL == []:


    elif inputL[0] > target:


    else:
```

```
>>> showSubset(9,[2,3,5])
[False, []]
```

```
>>> showSubset(10,[2,3,5])
[True, [2, 3, 5]]
```

showSubset

worksheet S

```
def showSubset(target, inputL):
    """Accepts an integer and a list of integers as inputs. Returns a two-item
    list: the first item is a boolean indicating whether or not there exists a
    subset of numbers in the list that adds up to the target, the second item is
    a list of the subset of numbers used."""
    # bc1: if our target reaches 0, then a subset exists
    if target == 0: return [True, []]

    # bc2: if our list becomes empty, then a subset does not exist
    elif inputL == []: return [False, []]

    # discard the number if it exceeds the target; check remaining
    elif inputL[0] > target: return showSubset(target, inputL[1:])

    # otherwise, try both using and losing the number
    else:
        useIt = showSubset(target - inputL[0], inputL[1:])
        useIt[1].append(inputL[0])    # keep track of the used number

        loseIt = showSubset(target, inputL[1:])

        if useIt[0]: return useIt
        else: return loseIt
```

Answer will be added to slides after class

```
>>> show_subset(9,[2,3,5])
[False, []]
```

```
>>> show_subset(10,[2,3,5])
[True, [2, 3, 5]]
```

sub(8, [2,3,5])
[T, [5, 3]]
use = [F, [2]]
lose = [T, [5,3]]
sub(6, [3,5])
sub(8, [3,5])
use = [F, [3]]
lose = [F, []]
use = [T, [5,3]]
sub(3, [5])
sub(6, [5])
sub(5, [5])
lose = [F, []]
sub(3, [])
use = [T, [5]]
lose = [F, []]
sub(8, [5])
[F, []]
sub(0, [])
sub(5, [])
use = [F, [5]]
lose = [F, []]
[F, []]
use = [F, [5]]
sub(1, [])
sub(6, [])
sub(3, [])
lose = [F, []]
[F, []]
[F, []]
[T, []]
[F, []]
sub(8, [])
[F, []]

A final example with LCS

```
def LCS(stringA, stringB):
    """Accepts two strings as input. Returns the length of
    the Longest Common Substring."""
    # base case
    if stringA == "" or stringB == "": return 0

    # add 1 if first characters match; check remaining
    elif stringA[0] == stringB[0]:
        return 1 + LCS(stringA[1:], stringB[1:])

    # try dropping 1 char from each string; return max
    else:
        option1 = LCS(stringA, stringB[1:])
        option2 = LCS(stringA[1:], stringB)
        return max(option1, option2)
```

```
>>> fancyLCS("human", "chimpanzee")
[4, 'h#man', '#h#m#an###']
```

In your notes...


```
def fancyLCS(S1, S2):
    """Accepts two strings as inputs. Returns a three item list
    where the first item is the LCS length (int), and the second
    and third items are the "pounded out" versions of S1 and S2."""
    if S1 == "":

    elif S2 == "":

    elif S1[0] == S2[0]:









    else:
```

In your notes...


```
def fancyLCS(S1, S2):
    """Accepts two strings as inputs. Returns a three item list
    where the first item is the LCS length (int), and the second
    and third items are the "pounded out" versions of S1 and S2."""
    if S1 == "":   return [0, "", len(S2) * "#"]   # bc1

    elif S2 == "": return [0, len(S1) * "#", ""]   # bc2

    elif S1[0] == S2[0]: # if first chars match, then ...
        match = fancyLCS(S1[1:], S2[1:]) # ... recurse ...
        match[0] += 1                    # ... update LCS number ...
        match[1] = S1[0] + match[1]      # ... update pounded S1 ...
        match[2] = S2[0] + match[2]      # ... update pounded S2.
        return match

    else: # if first chars mismatch, then ...
        option1 = fancyLCS(S1[1:], S2) # ... drop char from S1 ...
        option1[1] = "#" + option1[1]  # ... "pound out" S1 ...

        option2 = fancyLCS(S1, S2[1:]) # ... drop char from S2 ...
        option2[2] = "#" + option2[2]  # ... "pound out" S2 ...

        # ... return the better option
        if option1[0] > option2[0]: return option1
        else: return option2
```

Answer will be added to slides after class

hw5pr2: `superLCS` (20pts)

```
>>> superLCS("human", "chimpanzee")
[4, '-hu-m-an---', 'ch-impanzee']
```

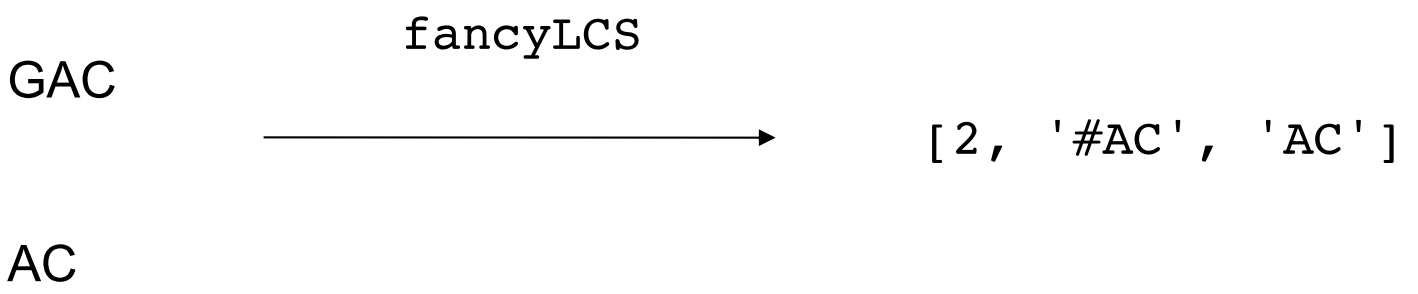
```
-hu-m-an---
ch-impanzee
```

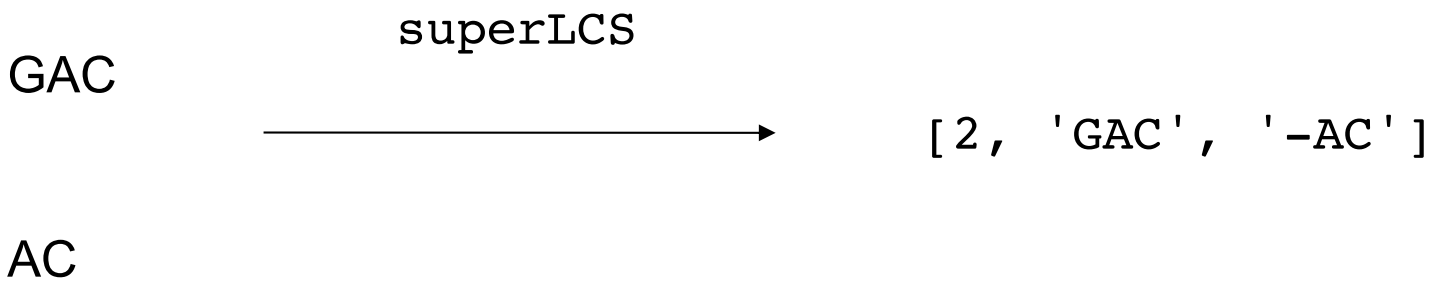
```
>>> superLCS("A","AT")
[1, 'A-', 'AT']
```

```
A-
AT
```

```
>>> superLCS("CG","G")
[1, 'CG', '-G']
```

```
CG
-G
```


fancyLCS
GAC
AC
[2, '#AC', 'AC']


superLCS
GAC
AC
[2, 'GAC', '-AC']

What is an alignment anyway?

Alignment representation

```
S1  GCCTGG-
S2  ACC-GGA
```

GCCTGG → **A**CCTGG	(change G to A)
	(keep the C)
	(keep the C)
ACC**T**GG → ACCGG	(delete the T)
	(keep the G)
	(keep the G)
ACCGG → ACCGG**A**	(insert A)

hw5pr3 + hw5pr4 (60pts): `align` and `memoAlignScore`

```
>>> alignScore("GCCTGG", "ACCGGA", -4, dnamat)
11
```

`alignScore` only returns score

```
>>> align("GCCTGG", "ACCGGA", -4, dnamat)
[11, 'gCCtGG-', 'aCC-GGa']
```

`align` returns score **and** alignment

Reminder:

- Lecture feedback form (https://forms.gle/aPmkpXDUTp4Xo4CV7)