 Claremont (AP): A CS professor at a small college in Southern California was rescued on Monday morning from a recursive function that was stuck in an infinite loop. “We’re very proud of our rescue team,” said a supervisor at the site of the incident. The professor had forgotten to put a base case in his function and got pulled into the program and was spinning at very high speed when we got there. We were called by an anonymous function. Fortunately, our elite rescue unit, code-named “Control C”, got there just in time. We mapped our way through his office, filtering through the immense debris, and were able to reduce the risk to the team by memo-izing the exact right process. The professor was recovering comfortably and the prognosis is that he may regain his sanity in the distant future.
True Story!

Caltech
Pasadena City College
True Story!

Caltech:
Pasadena City College
True Story!

Caltech
(Pasadena City College)
True Story!

Caltech = Pasadena City College
True Story!

Caltech \(\in\) Pasadena City College
True Story!

Caltech ⊆
Pasadena City College
Huntington Library
Caltech
(Pasadena City College)
Stuff

• Come to office hours to review concepts, get started on homework, ...
  • Mondays 10-12:30 PM
  • Fridays 3-4 PM (Great time to review concepts before embarking on HW!)

• Piazza

• Grutoring help
  • More on Sunday 8-10 PM!
  • Other times?
  • More care to check in with everyone
A quick aside about the **import** business!

from math import *

x = factorial(42)
foo = cos(x)
bar = pi/foo

import math

x = math.factorial(42)
foo = math.cos(x)
bar = math.pi/foo

How about a quick aside about foo and bar?
makePoly3 and funcAdd3

```python
>>> f = makePoly3([3, 2, 1])
3x^2 + 2x + 1
>>> f
<function <lambda> at 0x108396f28>

>>> f(10) \leftarrow 3*10^2 + 2*10 + 1
321
```

def foo(X):
    return 2 * X

g = lambda X: X+1
h = lambda X: X**2

```python
>>> j = funcAdd3([foo, g, h])
>>> j
<function <lambda> at 0x108396c42>
>>> j(3) \leftarrow 2*3 + (3+1) + 3^2 = 19
19
```
makePoly3 and funcAdd3 (in your notes)

```python
>>> f = makePoly3([3, 2, 1])
>>> f(10)  # 3*10**2 + 2*10 + 1
321

def makePoly3(L):
    """Takes a list of 3 coefficients [a, b, c] and returns the polynomial f(x) = ax**2 + bx + c """
    return lambda X:

def funcAdd3(L):
    """Takes a list of 3 functions and returns a new function that is their sum """
```
makePoly3 and funcAdd3

```python
>>> f = makePoly3([3, 2, 1])
>>> f(10) # 3*10**2 + 2*10 + 1
321

def makePoly3(L)
    """ Takes a list of 3 coefficients [a, b, c] and returns
    the polynomial f(x) = ax**2 + bx + c """
    return lambda X: L[0]*X**2 + L[1]*X + L[2]

def funcAdd3(L)
    """ Takes a list of 3 functions and returns
    a new function that is their sum """
    f1 = L[0]
    f2 = L[1]
    f3 = L[2]
    return lambda X: f1(X) + f2(X) + f3(X)

Alternatively: return lambda X: L[0](X) + L[1](X) + L[2](X)
```
**funcAdd and makePoly**

Now the lists can have any length! (Also in your notes)

```python
>>> f = funcAdd([f1, f2, f3, f4])

>>> f1 = makePoly([3, 2, 1])
>>> f2 = makePoly([6, 5, 2, 7])
>>> f1(10)
321
>>> f2(10)
6527

def funcAdd(funcList):
    if len(funcList) == 1:
        return ____________
    else:
        return lambda X:
```

Do funcAdd on this page and makePoly on the next page.
**funcAdd and makePoly**  

Now the lists can have any length!

```python
>>> f = funcAdd([f1, f2, f3, f4])

>>> f1 = makePoly([3, 2, 1])
>>> f2 = makePoly(6, 5, 2, 7])
>>> f1(10)
321
>>> f2(10)
6527

def funcAdd(funcList):
    if len(funcList) == 1: return funcList[0]
    else: return lambda X:
```

Do funcAdd on this page and makePoly on the next page.
**funcAdd and makePoly**

Now the lists can have any length!

```python
>>> f = funcAdd([f1, f2, f3, f4])

>>> f1 = makePoly([3, 2, 1])
>>> f2 = makePoly([6, 5, 2, 7])
>>> f1(10)
321
>>> f2(10)
6527

def funcAdd(funcList):  
    if len(funcList) == 1: return funcList[0]  
    else: return lambda X: funcList[0](X) + funcAdd(funcList[1:])(X)
```

Do funcAdd on this page and makePoly on the next page.
funcAdd and makePoly  

Now the lists can have any length!

```python
>>> f = funcAdd([f1, f2, f3, f4])
```

```python
>>> f1 = makePoly([3, 2, 1])
>>> f2 = makePoly([6, 5, 2, 7])
>>> f1(10)
321
>>> f2(10)
6527
```

def funcAdd(funcList):
    if funcList == []: return lambda X: 
    else: return lambda X: funcList[0](X) + funcAdd(funcList[1:])(X)
```
funcAdd and makePoly

Now the lists can have any length!

```python
>>> f = funcAdd([f1, f2, f3, f4])

>>> f1 = makePoly([3, 2, 1])
>>> f2 = makePoly(6, 5, 2, 7])
>>> f1(10)
321
>>> f2(10)
6527

def funcAdd(funcList):
    if funcList == []: return lambda X: 0
    else: return lambda X: funcList[0](X) + funcAdd(funcList[1:])(X)
```

Do funcAdd on this page and makePoly on the next page.
funcAdd and makePoly

Now the lists can have any length!

```python
>>> f = funcAdd([f1, f2, f3, f4])

>>> f1 = makePoly([3, 2, 1])
>>> f2 = makePoly([6, 5, 2, 7])
>>> f1(10)
321
>>> f2(10)
6527

def funcAdd(funcList):
    if funcList == []: return lambda X: 0
    else: return lambda X: funcList[0](X) + funcAdd(funcList[1:])(X)

This can also be done with reduce!
```
funcAdd and makePoly

Now the lists can have any length!

>>> f1 = makePoly([3, 2, 1])
>>> f2 = makePoly(6, 5, 2, 7])
>>> f1(10)
321
>>> f2(10)
6527

def makePoly1(coeffList):
    if coeffList == []: return lambda X: __________
    else:
        return lambda X: ___________________________________

def makePoly2(coeffList):
    if coeffList == []: return lambda X: _________
    else:
        return lambda X:
def makePoly1(coeffList):
    if coeffList == []: return lambda X: 0
    else:
        return lambda X: X**(len(coeffList)-1)*coeffList[0] +

def makePoly2(coeffList):
    if coeffList == []: return lambda X: 0
    else:
        return lambda X:
def makePoly1(coeffList):
    if coeffList == []: return lambda X: 0
    else:
        return lambda X: X**(len(coeffList)-1)*coeffList[0]
                        + makePoly1(coeffList[1:])(X)

def makePoly2(coeffList):
    if coeffList == []: return lambda X: 0
    else:
        return lambda X: X*makePoly2(coeffList[:-1])(X)
                        + coeffList[-1]
Another aside: List Comprehensions

```python
>>> list(map(lambda X: X*2, range(10)))
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

>>> [X*2 for x in range(10)]
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```
```python
>>> list(filter(lambda x: x % 2 == 0, range(10))
[0, 2, 4, 6, 8]
```
```
>>> [x for x in range(10) if x % 2 == 0]
[0, 2, 4, 6, 8]
```
```
>>> list(map(lambda x: x**2,
            filter(lambda y: y%2 == 0, range(10))))
[0, 4, 16, 36, 64]
```
```
>>> [x**2 for x in range(10) if x%2 == 0]
[0, 4, 16, 36, 64]
```

What does this do? Try writing a list comprehension equivalent!
>>> list(filter(lambda X: X % 2 == 0, range(10)))
[0, 2, 4, 6, 8]

>>> [X for X in range(10) if X % 2 == 0]
[0, 2, 4, 6, 8]

>>> list(map(lambda X: X**2,
          filter(lambda Y: Y%2 == 0,
                range(10))))
[0, 4, 16, 36, 64]

>>> [X**2 for X in range(10) if X%2 == 0]
[0, 4, 16, 36, 64]

You waited until NOW to tell us about this!?
Measuring similarity/difference between strings...

```python
>>> LCS("AAAA", "AGAAA")
4

>>> LCS("AAAA", "AGAAATCTCTCTCTCTCTCTCTCTCTC")
4

>>> ED("AAAA", "AGAAA")
1

>>> ED("AAAA", "AGAAATCTCTCTCTCTCTCTCTCTCTC")
23
```

For Edit Distance, a smaller score means more similar!
ED: Edit Distance

>>> ED(“qspam”, “scramble”)
6
qspam -> (delete q)
spam -> (substitute p with c)
scam -> (insert r)
scram -> (insert b)
scramb -> (insert l)
scramble

Permitted “morphing” operations:
- insert
- delete
- substitute

6 steps

def ED(S1, S2):
    if S1 == ‘’:
        elif S2 == ‘’:
            elif S1[0] == S2[0]:
                else:  # substitute, insert, or delete!
                    substitute =
                    insert =
                    delete =
                    return min(substitute, insert, delete)
def ED(S1, S2):
    if S1 == '': return len(S2)
    elif S2 == '': return len(S1)
    elif S1[0] == S2[0]: return ???
    else:  # substitute, insert, or delete!
def ED(S1, S2):
    if S1 == '': return len(S2)
    elif S2 == '': return len(S1)
    elif S1[0] == S2[0]: return ED(S1[1:], S2[1:])
    else:  # substitute, insert, or delete!
def ED(S1, S2):
    if S1 == '': return len(S2)
    elif S2 == '': return len(S1)
    elif S1[0] == S2[0]: return ED(S1[1:], S2[1:]);
    else:  # substitute, insert, or delete!
        substitute = 1 + ED(S1[1:], S2[1:]);
def ED(S1, S2):
    if S1 == '': return len(S2)
    elif S2 == '': return len(S1)
    elif S1[0] == S2[0]: return ED(S1[1:], S2[1:])
    else:  # substitute, insert, or delete!
        substitute = 1 + ED(S1[1:], S2[1:])
        insert =
        delete =
def ED(S1, S2):
    if S1 == '': return len(S2)
    elif S2 == '': return len(S1)
    elif S1[0] == S2[0]: return ED(S1[1:], S2[1:])
    else:  # substitute, insert, or delete!
        substitute = 1 + ED(S1[1:], S2[1:])
        insert = 1 + ED(S1, S2[1:])
        delete =
def ED(S1, S2):
    if S1 == '': return len(S2)
    elif S2 == '': return len(S1)
    elif S1[0] == S2[0]: return ED(S1[1:], S2[1:])
    else:  # substitute, insert, or delete!
        substitute = 1 + ED(S1[1:], S2[1:])
        insert = 1 + ED(S1, S2[1:])
        delete = 1 + ED(S1[1:], S2)
    return min(substitute, insert, delete)
Is ED Symmetric?
Spel Cheker Demo! (HW 3, Problem 2)
Power Set!

```python
>>> powerset([2, 3])
[[], [3], [2], [2, 3]]

>>> powerset([1, 2, 3])
[[], [3], [2], [2, 3], [1], [1, 3], [1, 2], [1, 2, 3]]

>>> powerset([1])
[[], [1]]

>>> powerset([])

This really demonstrates the power of functional programming!

Note that the order within each sublist is important, but the order in which the sublists are presented is not!