Recursion == Happiness
Today’s True Story
### CS 5: Welcome!

<table>
<thead>
<tr>
<th>Administration</th>
<th>Course Syllabus</th>
<th>Exams</th>
<th>Pairs Policy</th>
<th>Submission Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using Python</td>
<td>On your machine</td>
<td>In your browser</td>
<td>CS5's text</td>
<td>httlacs text</td>
</tr>
<tr>
<td>Useful/Helpful</td>
<td>GradeScope</td>
<td>CS5 Piazza</td>
<td>Grutoring!</td>
<td>Piazza text</td>
</tr>
<tr>
<td>Related Courses</td>
<td>POM CS 51</td>
<td>CGU IST 341</td>
<td>CS 5 Green</td>
<td>Credit Labs</td>
</tr>
</tbody>
</table>

Homework Assignments and Labs
Office hours and grutoring

Ran’s Office Hours (Olin 1253): Mondays and Fridays 3-4 PM

Grutoring hours in the Linde Activities Center (LAC) everyday!

<table>
<thead>
<tr>
<th></th>
<th>I</th>
<th>J</th>
<th>K</th>
<th>L</th>
<th>M</th>
<th>N</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Tutoring!</td>
<td>Tutoring!</td>
<td>Tutoring!</td>
<td>Tutoring!</td>
<td>Tutoring!</td>
<td>Tutoring!</td>
<td>Tutoring!</td>
</tr>
<tr>
<td>Time</td>
<td>Thursdays</td>
<td>Thursdays</td>
<td>Fridays</td>
<td>Fridays</td>
<td>Saturdays</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time</td>
<td>8:00pm - 10:00pm</td>
<td>10:00pm - 12ish</td>
<td>1:00pm - 3:00pm</td>
<td>3:00 - 5:00pm</td>
<td>1:00pm - 3:00pm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Notes</td>
<td>in the LAC</td>
<td>in the LAC</td>
<td>in the LAC</td>
<td>in the LAC</td>
<td>(no green b/c LAB)</td>
<td>in the LAC</td>
<td></td>
</tr>
<tr>
<td>Location</td>
<td>in the LAC</td>
<td>in the LAC</td>
<td>in the LAC</td>
<td>in the LAC</td>
<td>(no green b/c LAB)</td>
<td>in the LAC</td>
<td></td>
</tr>
<tr>
<td>5 gold/black/green tutoring @ HMC</td>
<td>up to 2 CS5ers (green welcome!)</td>
<td>up to 2 CS5ers (green welcome!)</td>
<td>up to 2 CS5ers (no green - b/c lab)</td>
<td>up to 3 CS5ers (no green - b/c lab)</td>
<td>up to 3 CS5ers (green welcome!)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Grutor sign-ups in the gold spaces</td>
<td>David Sobek</td>
<td>Cat Ngo</td>
<td>Pinky King</td>
<td>Renata Paramastri</td>
<td>Siena Guerrero</td>
<td></td>
<td></td>
</tr>
<tr>
<td>here (at right...)</td>
<td>&lt;grutoring spot&gt;</td>
<td>Sydney Wallace</td>
<td>Anant Kandori</td>
<td>Gabby Giordano</td>
<td>Cole Kurashige</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF you were in green</td>
<td>not available</td>
<td>not available</td>
<td>not available</td>
<td>Noah Boorstin</td>
<td>Hanna Hoffman</td>
<td>Leana Yearwood</td>
<td></td>
</tr>
</tbody>
</table>
def h(x):
    return f(x) + x

def f(x):
    x = x-1
    return g(x)+1

def g(x):
    return x*2

def h(3):
    return f(3) + 3

def f(3):
    return g(2) + 1

def g(2):
    return 4

Two key points…
• Functions return to where they were called from
• Each function keeps its own values of its variables
Factorial with loops…

\[ n! = n \times (n-1) \times (n-2) \times \ldots \times 1 \]

def factorial(n):
    result = 1
    for k in range(1, n+1):
        result = result * k
    return result

No recursion needed here, but some problems desperately need recursion!
Recursion…

\[ n! = n \times (n-1) \times (n-2) \times \ldots \times 1 \]

\[ n! = n \times (n-1)! \]  “inductive definition”

\[ 0! = 1 \]  “base case”

Why is 0! = 1?
Math Induction = CS Recursion

**Math**

inductive definition

\[
\begin{align*}
0! &= 1 \\
n! &= n \times (n-1)!
\end{align*}
\]

**Python (Functional)**

recursive function

```python
# recursive factorial
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```
Is Recursion Magic?

factorial(3):
    return 3 * factorial(2)
Is Recursion Magic?

factorial(3):

return 3 * factorial(2)

return 2 * factorial(1)

# recursive factorial
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
Is Recursion Magic?

factorial(3):
return 3 * factorial(2)

return 2 * factorial(1)

return 1 * factorial(0)

# recursive factorial
def factorial(n):
    if n == 0:
        return 1
    else:
        return n*factorial(n-1)
Is Recursion Magic?

factorial(3):
return 3 * factorial(2)

return 2 * factorial(1)

return 1 * factorial(0)

# recursive factorial
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
Is Recursion Magic?

factorial(3):
   return 3 * factorial(2)
      return 2 * factorial(1)
         return 1 * factorial(0)

# recursive factorial
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
Is Recursion Magic?

```python
# recursive factorial
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

factorial(3):
    return 3 * factorial(2)
        return 2 * factorial(1)
            return 1 * factorial(0)
```

# recursive factorial
```python
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```
Is Recursion Magic?

```python
# recursive factorial
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

factorial(3):
    return 3 * factorial(2)
    return 2 * factorial(1)
    return 1 * factorial(0)
    1
```

# recursive factorial
```python
# recursive factorial
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```

factorial(3):
    return 3 * factorial(2)
    return 2 * factorial(1)
    return 1 * factorial(0)
    1
A Tower of Fun!

Math

\[
tower(3) = 2^{2^2} \\
tower(4) = 2^{2^{2^2}} \\
tower(5) = 2^{2^{2^{2^2}}} \\
\]

Python (Functional)

```
# recursive tower
def tower(n):
```

# recursive function

Try this on your worksheet! (Note that \(x^y\) is computed using \(x**y\))
A Tower of Fun!

**Math**

tower(3) = \(2^{2^2}\)

tower(4) = \(2^{2^{2^2}}\)

tower(5) = \(2^{2^{2^{2^2}}}\)

**Python (Functional)**

recursive function

```python
# recursive tower function
def tower(n):
```
A Tower of Fun!

Math

tower(3) = \[2^{2^2}\]
tower(4) = \[2^{2^{2^2}}\]
tower(5) = \[2^{2^{2^{2^2}}}\]

inductive definition:

inductive case:
tower(n) = 2^{tower(n-1)}

base case:
tower(0) = 1

Python (Functional)

recursive function

```python
# recursive tower
def tower(n):
    """ Tower of n ""
    if n==0:
        return 1
    else:
```

A Tower of Fun!

Math

tower(3) = 2^2^2

tower(4) = 2^2^2^2

tower(5) = 2^2^2^2^2

inductive definition:

inductive case:
tower(n) = 2^{tower(n-1)}

base case:
tower(0) = 1

Python (Functional)

recursive function

```python
# recursive tower
def tower(n):
    """Tower of n""
    if n==0:
        return 1
    else:
        return 2**tower(n-1)
```

Demo!
Lists!

Python starts counting from 0…

```python
>>> M = [2, "alien", 42, ["spam","spamity","spam"] ]

>>> len(M)
4

>>> M[2]
42

>>> M[3]
['spam', 'spamity', 'spam']

>>> M[3][0]
'spam'

>>> M[1:]
['alien', 42, ['spam','spamity','spam'] ]
```

Hey! Cool!
I’m in a list!
Recursion on lists

```python
>>> len([1, 42, "spam"])
3
>>> len([1, [2, [3, 4]]])
2

def len(L):
    '''returns the length of list L'''
```
Computing the length of a list

```python
>>> len([1, 42, "spam"])
3
>>> len([1, [2, [3, 4]]])
2

def len(L):
    '''returns the length of list L'''
    if L == []: return 0  # base case
```
Computing the length of a list

>>> len([1, 42, "spam"])
3
>>> len([1, [2, [3, 4]]])
2

def len(L):
    '''returns the length of list L'''
    if L == []: return 0  # base case
    else: return 1 + len(L[1:])

What happens if I run len(42)?
Summing up the numbers in a list

```python
def sum(L):
    '''returns the sum of numbers in the list'''
```

Python has this built-in too!
Summing up the numbers in a list

```python
def sum(L):
    '''returns the sum of numbers in the list'''
    if L == []: return 0
```

The base case takes care of the simplest case we could get.
Summing up the numbers in a list

```python
>>> sum([1, 42, 7])
50
>>> sum([42])
42
>>> sum([])
0

def sum(L):
    '''returns the sum of numbers in the list'''
    if L == []: return 0
    else: return ???
```

Try finishing this in your notes (not worksheet)…
Summing up the numbers in a list

```python
>>> sum([1, 42, 7])
50
>>> sum([42])
42
>>> sum([])
0

def sum(L):
    '''returns the sum of numbers in the list'''
    if L == []: return 0
    else: return L[0] + sum(L[1:])
```
More tricks with lists...

```python
>>> myList = [42, 10, 5, "spam"]
```

0 1 2 3
-4 -3 -2 -1

```python
>>> myList[0]
42
```

```python
>>> myList[-4]
42
```

```python
>>> myList[-1]
"spam"
```

```python
>>> myList[100]
IndexError: list index out of range
```
More tricks with lists...

>>> myList = [42, 10, 5, "spam"]

>>> myList[0:3]  # stops one short of 3
[42, 10, 5]

>>> myList[0:-1] # stops one short of -1
[42, 10, 5]

>>> myList[1:]   # from 1 to the end
[10, 5, "spam"]

>>> myList[:2]   # from start to one short of 2
[42, 10]
More tricks with lists...

```python
>>> myList = [42, 10, 5, "spam"]
```

```plaintext
0   1   2   3
-4  -3  -2  -1
```

```python
>>> myList[100]
IndexError: list index out of range
```

```python
>>> myList[1:100]
[10, 5, "spam"]
```

You can’t get hurt when you slice!*  
(*In Python)
Adding lists

>>> myList = [42, 47, 23]
>>> newList = myList + 100
BARF!
>>> newList = myList + [100]
>>> newList
[42, 47, 23, 100]
>>> myList
[42, 47, 23]
>>> newList = newList + newList
>>> newList
[42, 47, 23, 100, 42, 47, 23, 100]
Adding lists

>>> myList = [42, 47, 23]
>>> newList = myList + 100
BARF!
>>> newList = myList + [100]
>>> newList
[42, 47, 23, 100]
>>> myList
[42, 47, 23]
>>> newList = newList + newList
>>> newList
[42, 47, 23, 100, 42, 47, 23, 100]
Adding lists

```python
>>> myList = [42, 47, 23]
>>> newList = myList + 100
BARF!
>>> newList = myList + [100]
>>> newList
[42, 47, 23, 100]
>>> myList
[42, 47, 23]
>>> newList = newList + newList
>>> newList
[42, 47, 23, 100, 42, 47, 23, 100]
```
Reversing a list

```python
def reverse(L):
    return L[::-1]
```

```python
>>> reverse([1, 2, 3, 4])
[4, 3, 2, 1]
```
Reversing a list

```python
>>> reverse([1, 2, 3, 4])
[4, 3, 2, 1]
```

def reverse(L):
    if L == []: return ???  # base case
Reversing a list

```python
def reverse(L):
    if L == []: return []  # base case
    else:
        return ???
```

```python
>>> reverse([1, 2, 3, 4])
[4, 3, 2, 1]
```

Try this in your notes (not worksheet)
Reversing a list

>>> reverse([1, 2, 3, 4])
[4, 3, 2, 1]

def reverse(L):
    if L == []: return []  # base case
    else:
        return reverse(L[1:]) + [L[0]]
Something that’s (very) hard to do **without** recursion…

```python
>>> reverse([1, [2, [4, 5], 6], 7])
[7, [2, [4, 5], 6], 1]

>>> deepReverse([1, [2, [4, 5], 6], 7])
[7, [6, [5, 4], 2], 1]

>>> removeAll(42, [67, 42, [42, [42, 43]], 47])
[67, [42, [42, 43]], 47]

>>> deepRemoveAll(42, [67, 42, [42, [42, 43]], 47])
[67, [[43]], 47]
```
Recursion = :^)“To understand recursion, you must first understand recursion” - anonymous Mudd alum
Strings!

>>> myDNA = "AATGCCGTGCTT"

>>> len(myDNA) # built-in len function!
12

>>> myDNA[0]
'A'

>>> myDNA[3]
'G'

>>> myDNA[20]
IndexError: string index out of range
Strings...

```python
>>> myDNA = "AATGCCGTGCTT"

>>> myDNA[0:4]
'AATG'

>>> myDNA[3:7]
'GCCG'

>>> myDNA[1:]
'ATGCCGTGCTT'

>>> myDNA[:4]
'AATG'

>>> myDNA[10:42]
'TT'

>>> foo = ''
>>> foo[1:]
''
```
Palindrome?

```python
>>> pal("radar")
True
>>> pal("amanaplanacanalpanama")
True
>>> pal("spam")
False

def pal(myString):
    ''' returns True if myString is a palindrome and False otherwise '''
    # base case?
    # general case?
```

A solution is on the next slide, but try to avert your eyes for now!
def pal(myString):
    ''' returns True if myString is a palindrome and False otherwise '''
    if len(myString) <= 1: return True # base case
    elif myString[0] != myString[-1]:
        return False
    else: return pal(myString[1:-1])
len revisited

```python
>>> len([1, 42, "spam"])
3
>>> len([1, [2, [3, 4]]])
2
```

def len(L):
    '''returns the length of list L'''
    if L == []: return 0  # base case
    else: return 1 + len(L[1:])

Hint... Recall...
>>> foo = ''
>>> foo[1:]
''

What happens if I run `len("spam")`?
What's the fix?
def len(L):
    '''returns the length of list L'''
    if L == []: return 0  # list base case
    elif L == '': return 0  # string base case
    else: return 1 + len(L[1:])

OR...

def len(L):
    if L == [] or L == '': return 0
    else: return 1 + len(L[1:])
Try the next few functions in your worksheet…

- min
- member
- Insertion sort!

Don’t rush! No need to finish these.
>>> min([372, 112, 42, 451])
42
>>> min([16])
16

def min(L):
    ''' returns smallest value in the list L '''

Assume that the list always has at least one number in it! You can use the built-in len function to find the length of the list.
>>> member(42, [1, 3, 5, 42, 7])
True
>>> member(42, ["spam", "is", "yummy"])
False

def member(thing, myList):
    ''' return True if thing in myList and False otherwise. '''
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 '''
```

def sort(myList):
  ```
Minimum!

```python
>>> min([372, 112, 42, 451])
42

>>> min([16])
16

def min(L):
    ''' returns smallest value in the list L '''
    if len(L) == 1: return L[0]  # base case!
    elif L[0] < min(L[1:]): ???
    else: return min(L[1:])
```
>>> min([372, 112, 42, 451])
42
>>> min([16])
16

```python
def min(L):
    ''' returns smallest value in the list L '''
    if len(L) == 1: return L[0] # base case?
    elif L[0] < min(L[1:]): return L[0]
    else: return ???
```

Assume that the list always has at least one number in it! Use `len` as a helper function!
Minimum!

```python
>>> min([372, 112, 42, 451])
42
>>> min([16])
16

def min(L):
    ''' returns smallest value in the list L '''
    if len(L) == 1: return L[0]  # base case?
    elif L[0] < min(L[1:]): return L[0]
    else: return min(L[1:])
```

This isn’t very efficient! How come?
Minimum!

>>> min([372, 112, 42, 451])
42

>>> min([16])
16

This isn’t very efficient!

def min(L):
    ''' returns smallest value in the list L '''
    if len(L) == 1: return L[0] # base case?
    else:
        minRest = min(L[1:]):
        if L[0] < minRest: return L[0]
        else: return minRest
member

>>> member(42, [1, 3, 5, 42, 7])
True
>>> member(42, ["spam", "is", "yummy"])
False

def member(thing, myList):
    ''' return True if thing in myList and False otherwise. '''
member

>>> member(42, [1, 3, 5, 42, 7])
True
>>> member(42, [“spam”, “is”, “yummy”])
False

def member(thing, myList):
    if myList == []: return ???  # base case
member

>>> member(42, [1, 3, 5, 42, 7])
True
>>> member(42, [“spam”, “is”, “yummy”])
False

def member(thing, myList):
    if myList == []: return False  # base case
    elif thing == myList[0]: return True
member

>>> member(42, [1, 3, 5, 42, 7])
True
>>> member(42, ["spam", "is", "yummy"])  
False

def member(thing, myList):
    if myList == []: return False    # base case
    elif thing == myList[0]: return True   
    else: return member(thing, myList[1:])

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 '''
```

def sort(myList):
```
Next week...

“Fancier” recursion including...

- string similarity
- fractals
- ... and more!

```python
>>> similarity("spam", "scramble")
5
spam -> scam -> scram -> scramb -> scrambl -> scramble
```