

CS16 Section 0 Mini-Assignment

Due in your section the week of 1/28 - 2/1

Please bring a hard copy of your answers to section!

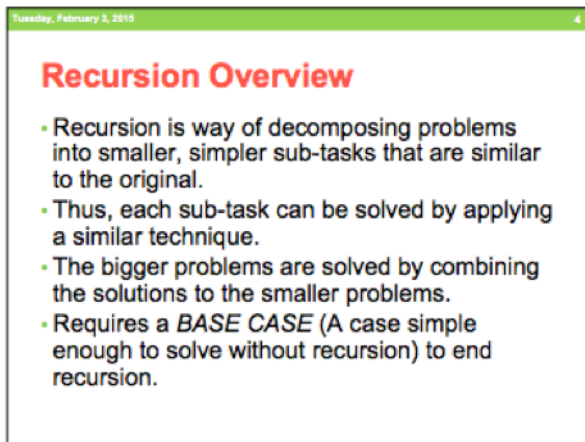
Problem 1

Please answer the following questions about yourself:

- What is your class year?
- What are you interested in academically?
- What are you hoping to get out of CS16 and CS16 section?
- What are you looking to get out of the mentorship?
- What else do you want us to know about you?

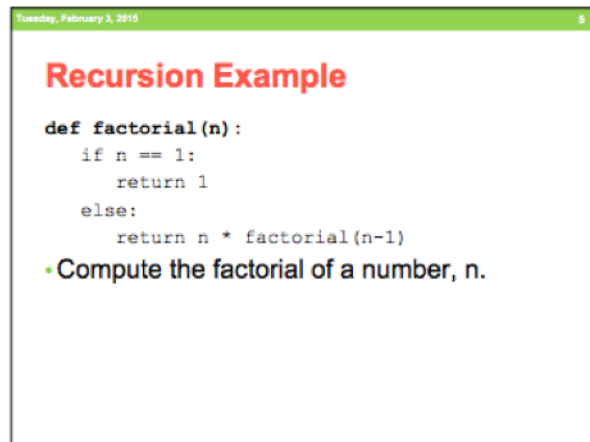
Problem 2

Read the following slides and use the call stack example to answer the question at the end.



Slide 4: Recursion Overview

- Recursion is way of decomposing problems into smaller, simpler sub-tasks that are similar to the original.
- Thus, each sub-task can be solved by applying a similar technique.
- The bigger problems are solved by combining the solutions to the smaller problems.
- Requires a *BASE CASE* (A case simple enough to solve without recursion) to end recursion.



Slide 5: Recursion Example

```
def factorial(n):  
    if n == 1:  
        return 1  
    else:  
        return n * factorial(n-1)
```


- Compute the factorial of a number, n .

Tuesday, February 3, 2015 6

Recursion Simulation

Calculate 3 factorial!

```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n*factorial(n-1)
```



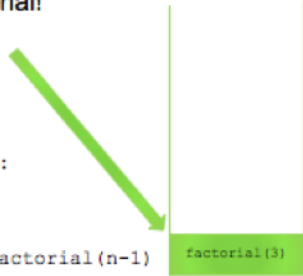
Call Stack

Tuesday, February 3, 2015 7

Recursion Simulation

Calculate 3 factorial!

```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n*factorial(n-1)
```




Call Stack

Tuesday, February 3, 2015 8

Recursion Simulation

- $n \neq 1$, so we return $n \cdot \text{factorial}(n-1)$ which includes a call to $\text{factorial}(2)$.

```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n*factorial(n-1)
```



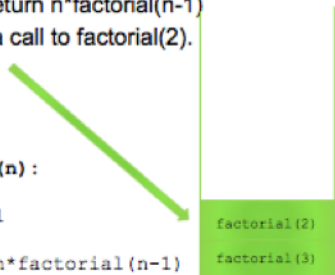
Call Stack

Tuesday, February 3, 2015 9

Recursion Simulation

- $n \neq 1$, so we return $n \cdot \text{factorial}(n-1)$ which includes a call to $\text{factorial}(2)$.

```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n*factorial(n-1)
```




Call Stack

Tuesday, February 3, 2015 10

Recursion Simulation

- n is still $\neq 1$, so we return $n \cdot \text{factorial}(n-1)$ again. In this case, there's a call to $\text{factorial}(1)$.

```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n*factorial(n-1)
```



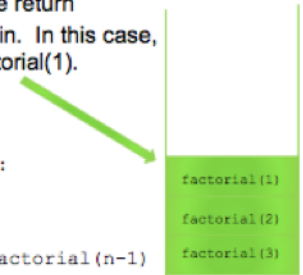
Call Stack

Tuesday, February 3, 2015 11

Recursion Simulation

- n is still $\neq 1$, so we return $n \cdot \text{factorial}(n-1)$ again. In this case, there's a call to $\text{factorial}(1)$.

```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n*factorial(n-1)
```



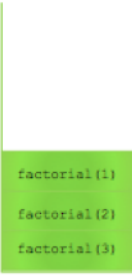
Call Stack

Tuesday, February 3, 2015 12

Recursion Simulation

- Now $n = 1$, so we return 1!

```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n*factorial(n-1)
```




Call Stack

Tuesday, February 3, 2015 13

Recursion Simulation

- Now $n = 1$, so we return 1!

```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n*factorial(n-1)
```




Call Stack

Tuesday, February 3, 2015 14

Recursion Simulation

- So factorial(2) returns $2 * \text{factorial}(1)$, Which we now know is $2 * 1$, so factorial(2) returns 2, and is removed from the call stack!

```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n*factorial(n-1)
```



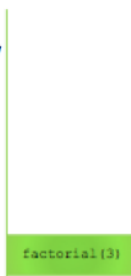
Call Stack

Tuesday, February 3, 2015 15

Recursion Simulation

- Now factorial(3) is at the "top" of the call stack, so we're back in factorial(3).
- Return $3 * \text{factorial}(2)$, which we now know is $3 * 2$.
- Factorial(3) returns 6 and removes itself from the call stack.

```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n*factorial(n-1)
```




Call Stack

Tuesday, February 3, 2015 16

Recursion Simulation

- Now factorial(3) is at the "top" of the call stack, so we're back in factorial(3).
- Return $3 * \text{factorial}(2)$, which we now know is $3 * 2$.
- Factorial(3) returns 6 and removes itself from the call stack.

```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n*factorial(n-1)
```




Call Stack

Tuesday, February 3, 2015 17

Recursion Simulation

- Now our call stack is empty, and we know that factorial(3) is 6!

```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n*factorial(n-1)
```



Call Stack

Now that you have read the slides, go through the following code for a function $\text{fib}(n)$ that finds the n th number of the Fibonacci Sequence. Draw out the call stack for $\text{fib}(4)$. Follow the format used in the factorial example from the slides.

```
def fib(n):
    if n == 0:
```

```
        return 0
    if n == 1:
        return 1
    return fib(n-1) + fib(n-2)
```