CIS 122

Let's do that again!

Homework Review

- Most homework submitted
  - Will post homework solution
  - Will go over in more detail

- At least 1 pair submission
  - Would love to see more

- Generally correct, but a few common details
  - Include your name
  - Remember docstrings
  - Remember comments

Homework 1 Continued

- You wrote max, max3, max5
  - What about general max function?
- You wrote single character shifter
  - Could probably write 2-character shifter
  - What about arbitrary length text shifter?
- Don't have the right tools yet
  - Let's fix that

The Factorial Function

- Represented by the ! symbol

- Product of all numbers up to x
  - 3! = 3 * 2 * 1 = 6
  - 5! = 5 * 4 * 3 * 2 * 1 = 120

- Factorial gets really large really quickly
  - 10! = 3628800
  - 20! = 2432902008176640000
  - 30! = 265252859812191058636308480000000
  - You get the idea...

The Factorial Function

- How would we write a factorial function?

```
def factorial(x):
    if x==1:
        return 1
    elif x==2:
        return 1 * 2
    elif x==3:
        return 1 * 2 * 3
    elif ...
```

- This could take a while...

The Factorial Function

- Let's reexamine our problem
- Suppose we want to calculate 10!

$$10! = 10 * 9 * 8 * 7 * 6 * 5 * 4 * 3 * 2 * 1$$

The Factorial Function

- Let's reexamine our problem
- Suppose we want to calculate 10!

$$10! = 10 * 9 * 8 * 7 * 6 * 5 * 4 * 3 * 2 * 1$$

The Factorial Function

- Let's reexamine our problem
- Suppose we want to calculate 10!

$$10! = 10 * 9!$$

The Factorial Function

- Let's reexamine our problem

- Suppose we want to calculate 10!

$$10! = 10 * 9!$$

- If we knew 9 factorial, 10 factorial would be easy
    - But how do we calculate 9 factorial?

The Factorial Function

- Let's reexamine our problem
- Suppose we want to calculate 10!

10! = 10 * 9!

- If we knew 9 factorial, 10 factorial would be easy
  - But how do we calculate 9 factorial?

9! = 9 * 8 * 7 * 6 * 5 * 4 * 3 * 2 * 1

The Factorial Function

- Let's reexamine our problem
- Suppose we want to calculate 10!

$$10! = 10 * 9!$$

- If we knew 9 factorial, 10 factorial would be easy
    - But how do we calculate 9 factorial?

$$9! = 9 * 8!$$

The Factorial Function - Take Two

- It's hard to calculate x!
    - But x! is just x * (x-1)!
    - If we knew (x-1)!, it would be easy to find x!
    - Let's try writing that function again...

```
def factorial(x):
    answer = x * factorial(x-1)
    return answer
```

- How do we feel about this code?
    - Let's try drawing up a stack diagram...

The Factorial Function - Take Two

__main__

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

>>> x = factorial(2)
```

The Factorial Function - Take Two

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

>>> x = factorial(2)
```

__main__

```
    factorial → <func>
    x         → ???
```

The Factorial Function - Take Two

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

>>> x = factorial(2)
```

__main__

```
    factorial → <func>
    x         → ???
```

factorial

The Factorial Function - Take Two

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

>>> x = factorial(2)
```

__main__

```
    factorial → <func>
    x         → ???
```

factorial

```
    n         → 2
    answer    → ???
```

The Factorial Function - Take Two

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

>>> x = factorial(2)
```

__main__

```
    factorial → <func>
    x         → ???
```

factorial

```
    n         → 2
    answer    → ???
```

factorial

The Factorial Function - Take Two

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

>>> x = factorial(2)
```

```
__main__
    factorial → <func>
    x         → ???

factorial
    n         → 2
    answer → ???

factorial
    n         → 1
    answer → ???
```

The Factorial Function - Take Two

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

>>> x = factorial(2)
```

__main__

```
    factorial → <func>
    x         → ???
```

factorial

```
    n         → 2
    answer    → ???
```

factorial

```
    n         → 1
    answer    → ???
```

factorial

The Factorial Function - Take Two

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

>>> x = factorial(2)
```

__main__

```
    factorial → <func>
    x         → ???
```

factorial

```
    n      → 2
    answer → ???
```

factorial

```
    n      → 1
    answer → ???
```

factorial

```
    n      → 0
    answer → ???
```

The Factorial Function - Take Two

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

>>> x = factorial(2)
```

```
__main__
    factorial → <func>
    x         → ???

factorial
    n         → 2
    answer → ???

factorial
    n         → 1
    answer → ???

factorial
    n         → 0
    answer → ???
```

This could take a while...

The Factorial Function - Take Two

- We're making progress
    - Now our code is finite
    - But it doesn't terminate...

- Let's fix that
    - Need somewhere to stop
    - A **Base Case**

The Factorial Function - Take Three

- Let's pick a really easy case
  - We know 0 factorial is 1
  - If we see the input 0, we'll just return 1

```
def factorial(n):
    if n==0:
        return 1
    else:
        answer = n * factorial(n-1)
        return answer
```

- What happens when we run this code?
  - Back to the stack...

The Factorial Function - Take Three

__main__

```
def factorial(n):
    if n==0:
        return 1
    else:
        answer = n * factorial(n-1)
        return answer

>>> x = factorial(2)
```

The Factorial Function - Take Three

```
def factorial(n):
    if n==0:
        return 1
    else:
        answer = n * factorial(n-1)
        return answer

>>> x = factorial(2)
```

__main__

```
factorial → <func>
x         → ???
```

The Factorial Function - Take Three

```
def factorial(n):
    if n==0:
        return 1
    else:
        answer = n * factorial(n-1)
        return answer

>>> x = factorial(2)
```

__main__

```
factorial → <func>
x         → ???
```

factorial

The Factorial Function - Take Three

```
def factorial(n):
    if n==0:
        return 1
    else:
        answer = n * factorial(n-1)
        return answer

>>> x = factorial(2)
```

__main__

```
    factorial → <func>
    x         → ???
```

factorial

```
    n         → 2
    answer → ???
```

The Factorial Function - Take Three

```
def factorial(n):
    if n==0:
        return 1
    else:
        answer = n * factorial(n-1)
        return answer

>>> x = factorial(2)
```

__main__

```
factorial → <func>
x         → ???
```

factorial

```
n      → 2
answer → ???
```

factorial

The Factorial Function - Take Three

```
def factorial(n):
    if n==0:
        return 1
    else:
        answer = n * factorial(n-1)
        return answer

>>> x = factorial(2)
```

```
__main__
    factorial → <func>
    x         → ???

factorial
    n         → 2
    answer    → ???

factorial
    n         → 1
    answer    → ???
```

The Factorial Function - Take Three

```
def factorial(n):
    if n==0:
        return 1
    else:
        answer = n * factorial(n-1)
        return answer

>>> x = factorial(2)
```

__main__

```
    factorial → <func>
    x         → ???
```

factorial

```
    n      → 2
    answer → ???
```

factorial

```
    n      → 1
    answer → ???
```

factorial

The Factorial Function - Take Three

```
def factorial(n):
    if n==0:
        return 1
    else:
        answer = n * factorial(n-1)
        return answer

>>> x = factorial(2)
```

```
__main__
    factorial → <func>
    x         → ???

factorial
    n         → 2
    answer → ???

factorial
    n         → 1
    answer → ???

factorial
    n         → 0
    answer → 1
```

The Factorial Function - Take Three

```
def factorial(n):
    if n==0:
        return 1
    else:
        answer = n * factorial(n-1)
        return answer

>>> x = factorial(2)
```

```
__main__
    factorial → <func>
    x         → ???

factorial
    n         → 2
    answer → ???

factorial
    n         → 1
    answer → 1

factorial
    n         → 0
    answer → 1
```

The Factorial Function - Take Three

```
def factorial(n):
    if n==0:
        return 1
    else:
        answer = n * factorial(n-1)
        return answer

>>> x = factorial(2)
```

__main__

```
    factorial → <func>
    x         → ???
```

factorial

```
    n      → 2
    answer → 2
```

factorial

```
    n      → 1
    answer → 1
```

factorial

```
    n      → 0
    answer → 1
```

The Factorial Function - Take Three

```
def factorial(n):
    if n==0:
        return 1
    else:
        answer = n * factorial(n-1)
        return answer

>>> x = factorial(2)
```

__main__

```
    factorial → <func>
    x         → 2
```

factorial

```
    n      → 2
    answer → 2
```

factorial

```
    n      → 1
    answer → 1
```

factorial

```
    n      → 0
    answer → 1
```

Recursion

- Reducing a problem to a **smaller** version of itself

- "To understand recursion, you must first understand recursion"
  - Try googling "recursion"

- Two Components
  - Base Case
  - Recursive step

Base Case

- Some easy known case
    - Generally something small and trivial
    - 0! = 1
- Want to reduce all other problems down to this case
- Don't forget your base case
    - Code might break
    - Code might never terminate

Recursive Step

- Define the problem in terms of a **smaller** version of itself
    - How do I compute x factorial?
    - Compute (x-1) factorial and multiply by x

- What do we mean by smaller?
    - Closer to the base case
    - Eventually reduce to the base case

- What happens if our problem doesn't get smaller?
    - Code will never terminate
    - To compute x!, first compute x!

Recursion is all around us

- How do you do the dishes?

- Base case
    - If the sink is empty, you're done

- Recursive step
    - Wash one dish
    - Wash the rest of the dishes

Recursion is all around us

- How do I walk to school?
- Base case
  - If I'm at school, I'm done
- Recursive step
  - Take one step towards school
  - Walk the rest of the way to school

Recursion in Action

- Over to IDLE