Announcements

- Homework
  - If you haven't gotten HW4 in yet, get it finished up! I'm around all afternoon today if you have any last little questions.
  - HW5 should be posted.
- I'm trying to get some grade reports generated this week so you can know where you are sitting at in the class currently.
  - I'll send out a blast on Campuswire when that happens
- Polling: `rembold-class.ddns.net`

Review Question

What would be the final printed value of the code to the right?

A) 50
B) 12
C) 10
D) This code would error out

```
def func(a, b=5, c=True):
    if c:
        return a + b
    else:
        return a * b

b = 2
print(func(10,False))
```

Functional Communication

- We've already mentioned using comments to communicate important ideas in your code to readers
- Communication even more important with the introduction of functions
  - What does this function even do?
  - What types of values can I pass into it?
  - What types of values does it return?
  - Are there any restrictions or qualifications about what can be input or output?
- Supposedly, all of this can be gleaned from the code, but it makes far more sense to present it all upfront
  - Introducing docstrings!

What's up doc?

- A docstring (or specification) is like an elaborate comment at the start of a function
- Is surrounded in triple quotes to inform the interpreter

```
def important_function(a,b,c):
    """ This is a docstring yo! """
    <code here>
```

Tis Binding

- Contract between function writer and function users (even if they are the same person!)
  - What does the function do in broad terms?
  - Assumptions:
    - What variable types are allowed as inputs?
    - Are there any restrictions or restraints on those input parameters?
  - Guarantees:
    - What will the program return under potential conditions?
    - How accurate will answers be?

Specific Benefits

- Teamwork
  - Individual team-members can work on different functions from the same program
  - Good specifications let them know that everything will work when they bring it all together
- Testing
  - Good specifications state exactly how a function should operate
  - Makes it easy to write small tests to ensure than everything is working as intended
  - Writing a few short tests early can save lots of time later

Functional Summary

- Functions are all about providing two main things:
  - Decomposition
    - Allows us to break a program into smaller chunks
    - Makes troubleshooting more efficient
    - Can reuse chunks of code in different settings
  - Abstraction
    - Hides details that are not currently relevant to the problem at hand
    - Let's us focus only on what is important

Recursion

- Recursion is the process of repeating items in a self-similar way

Recursion

- Recursion is the process of repeating items in a self-similar way
- Algorithmically: method of finding solutions to problems with a divide and conquer strategy

Recursion

- Recursion is the process of repeating items in a self-similar way
- Algorithmically: method of finding solutions to problems with a divide and conquer strategy
  - Reduce a problem to solving simpler versions of the same problem

Recursion

- Recursion is the process of repeating items in a self-similar way
- Algorithmically: method of finding solutions to problems with a divide and conquer strategy
  - Reduce a problem to solving simpler versions of the same problem
- Programatically: a technique of writing a function which *calls itself* within the body of the function.

Recursion

- Recursion is the process of repeating items in a self-similar way
- Algorithmically: method of finding solutions to problems with a divide and conquer strategy
  - Reduce a problem to solving simpler versions of the same problem
- Programatically: a technique of writing a function which *calls itself* within the body of the function.
  - We do NOT want infinite recursion

Recursion

- Recursion is the process of repeating items in a self-similar way
- Algorithmically: method of finding solutions to problems with a divide and conquer strategy
  - Reduce a problem to solving simpler versions of the same problem
- Programatically: a technique of writing a function which *calls itself* within the body of the function.
  - We do NOT want infinite recursion
    - Need 1 or more base cases that we can solve without recursion

Looping vs Recursion

Example

It can be illustrative to look at a very simple example of how a looping algorithm compares to a recursive algorithm.
Take the case of multiplication, where multiplying a value A by B is the same as "adding A to itself B"

$$A \times B = A_1 + A_2 + A_3 + \cdots + A_B$$

How would we approach this using both looping and recursion?

The Looping Case

- Define a variable (often called a state variable to keep track of the current state of the multiplication and which gets updated each iteration
- Loop through the necessary number of iterations, updating the state variable each time

```
def mult_w_loop(A, B):
    total = 0
    for i in range(B):
        total += A
    return total
```

The Recursive Case

- Need a Recursive Step
  - How to reduce the problem to a simpler/smaller version of the same problem?

$$A \times B = A + \underbrace{A + A + A + \cdots + A}_{B-1}$$

$$A \times B = A + A \times (B - 1)$$

- Need a Base Case
  - When $B = 1$, $A \times B = A$

```
def mult_w_rec(A,B):
    if B == 1:
        return A
    else:
        return A + mult_w_rec(A, B-1)
```

Example!

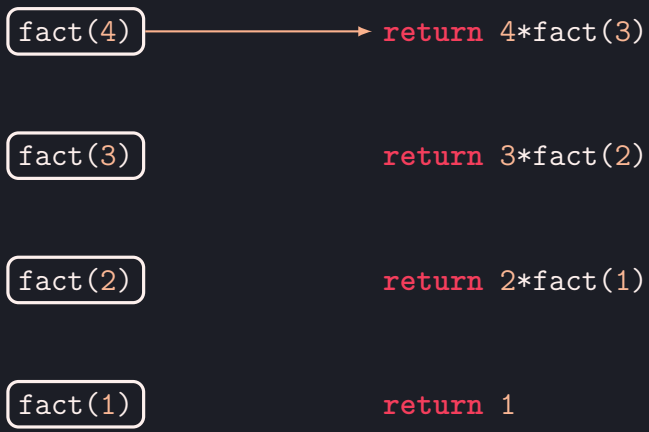
Example

The factorial of a number is defined as the product of all the positive integers equal to or smaller than that number. In variable form, this would look like:

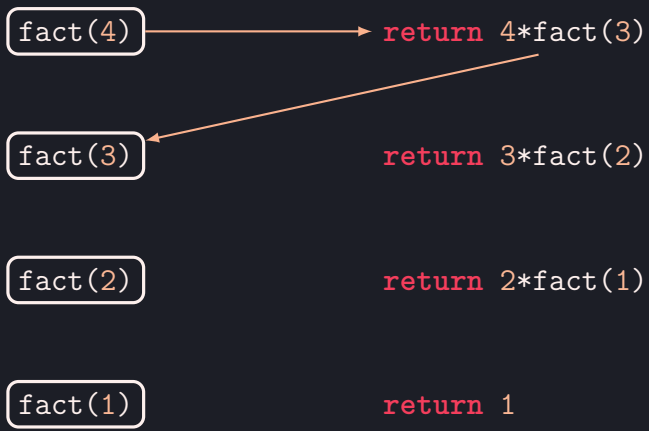
$$n! = n \times (n-1) \times (n-2) \times \cdots \times 1$$

Write a recursive function to return the factorial of any provided positive integer.

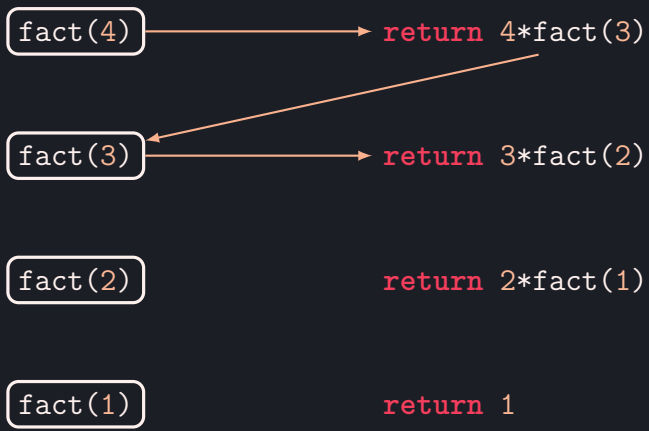
Factorial in Pictures



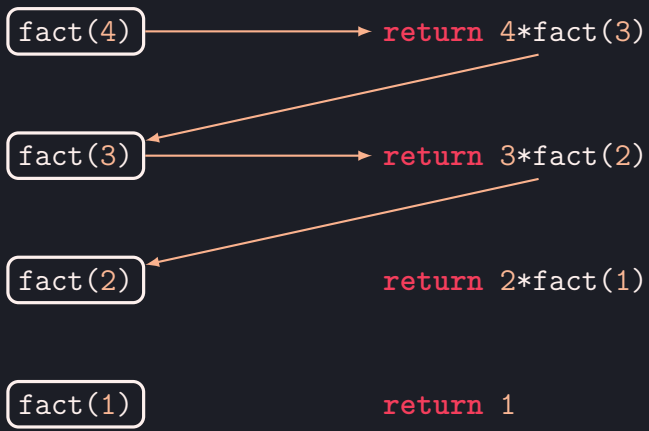
Factorial in Pictures



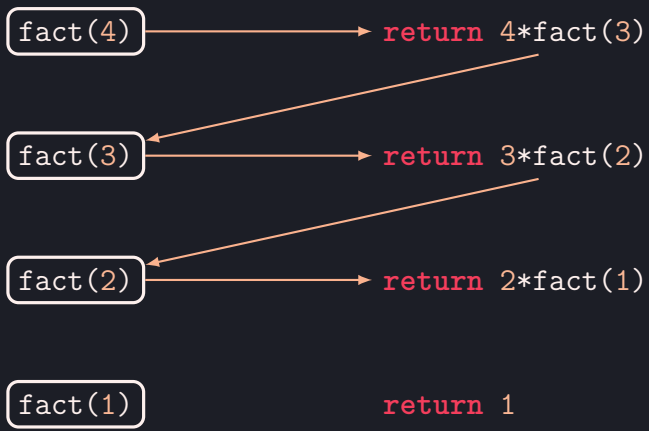
Factorial in Pictures



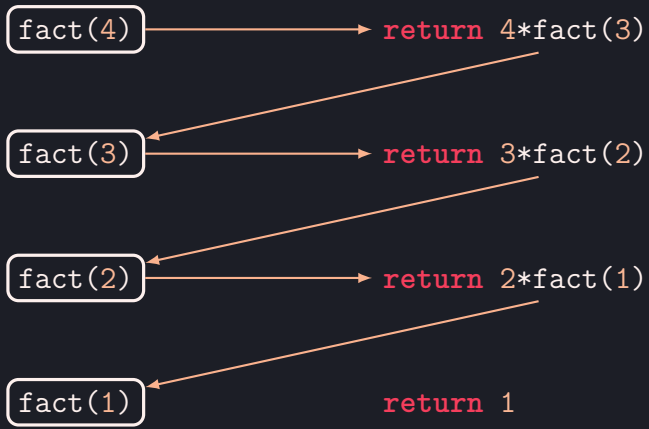
Factorial in Pictures



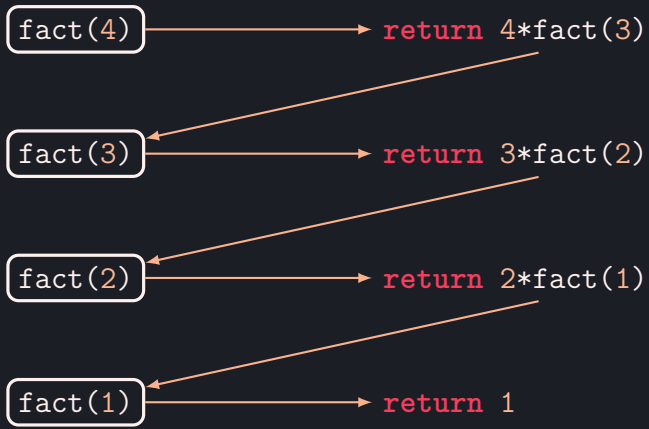
Factorial in Pictures



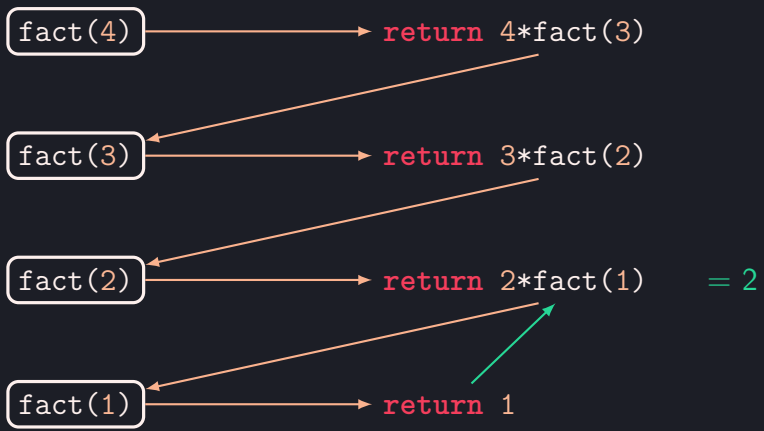
Factorial in Pictures



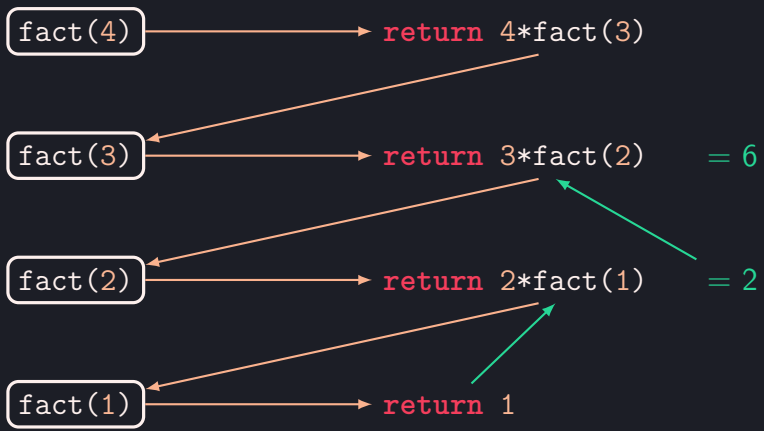
Factorial in Pictures



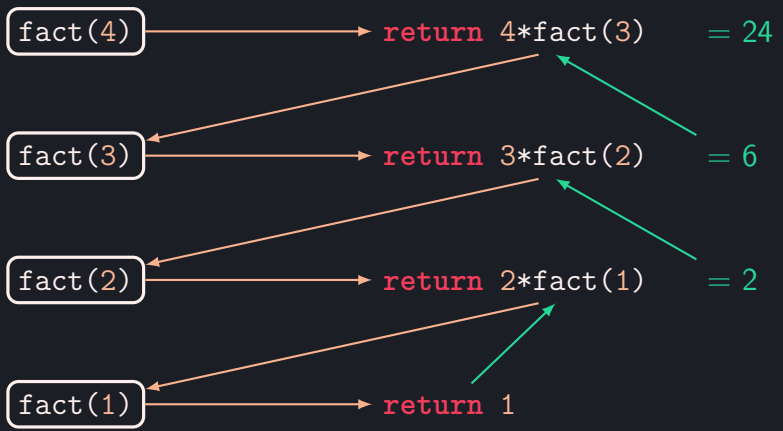
Factorial in Pictures



Factorial in Pictures



Factorial in Pictures



Some Observations

- Each call of the function (even the same function) *creates its own scope*.
- Variable values defined in a particular scope are not changed by further recursive calls
- Once a function returns its value, the flow of control passes back to the previous scope

Iteration vs Recursion

```
def fact_iter(n):
  prod = 1
  for i in range(1,n+1):
    prod *= i
  return prod
```

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

- Recursion may be simpler or more intuitive in some cases
- Recursion might be more efficient from a programmer's POV
- Recursion may not be efficient at all from a computer's POV