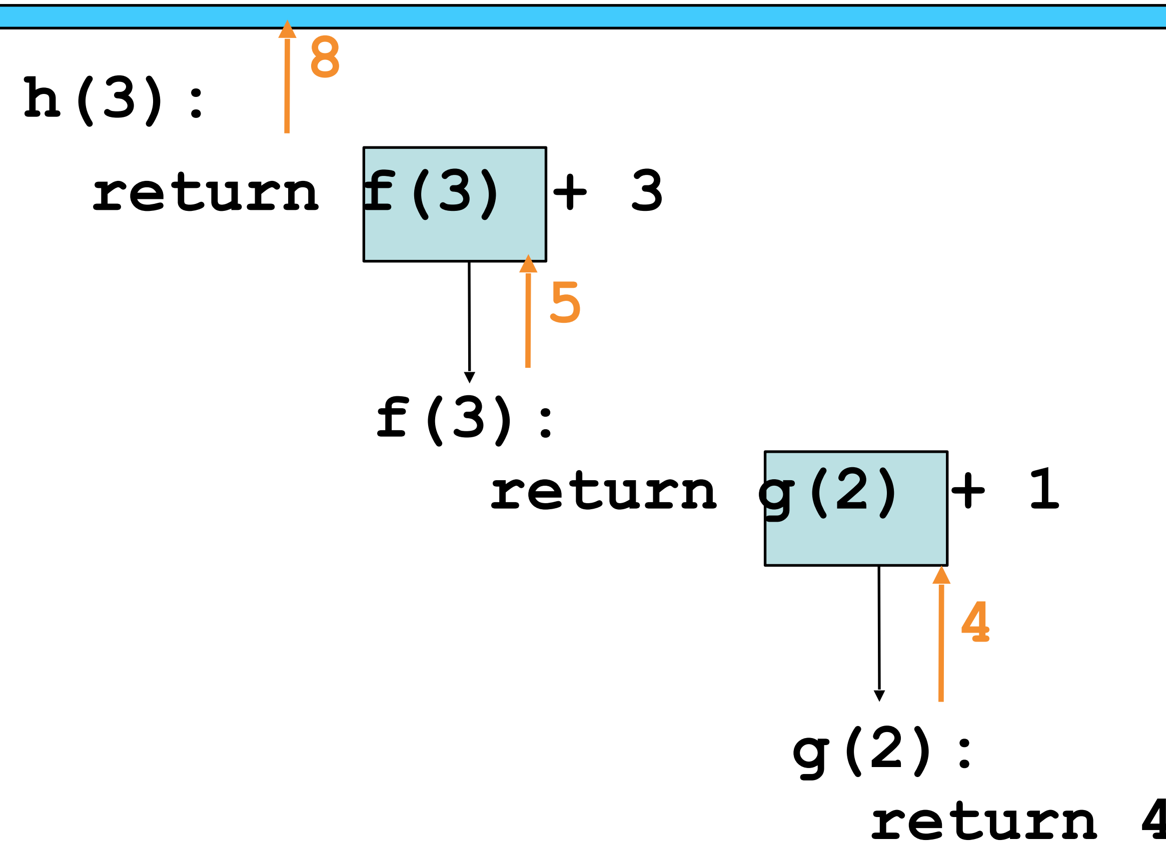
What Happens Inside a Function?

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

def g(x):
    return x*2

def h(x):
    if x%2 == 1:        # x odd
        return f(x) + x
    else:               # x even
        return f(f(x))
```



Two key points…

- Functions return to where they were called from
- Each function keeps its own values of its variables

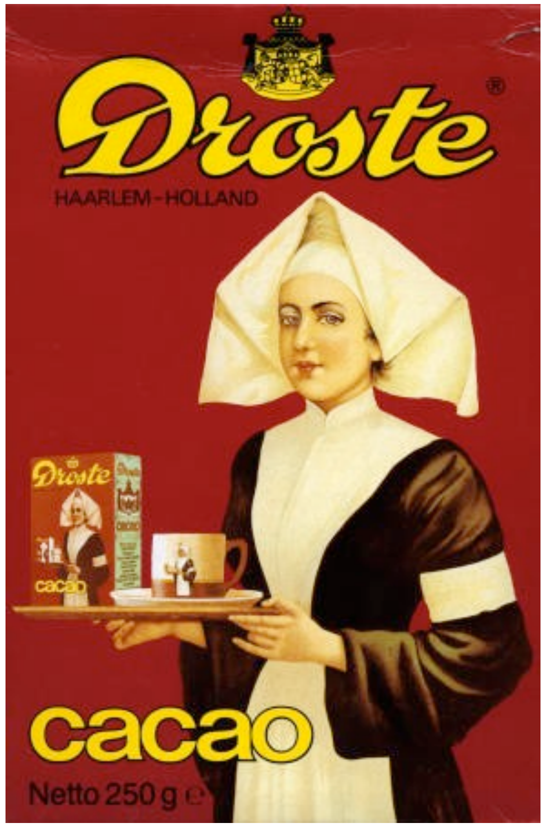
Recursion…

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

Recursion…

`n! = n×(n-1)×(n-2)×…×1`

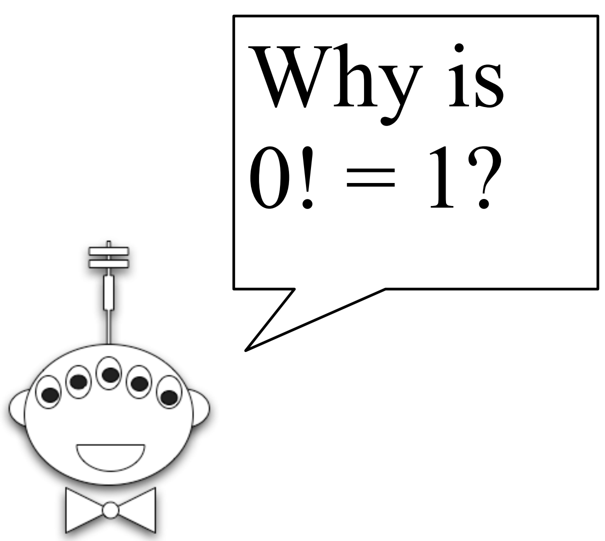
`n! = n×((n-1)!)`   "inductive definition"

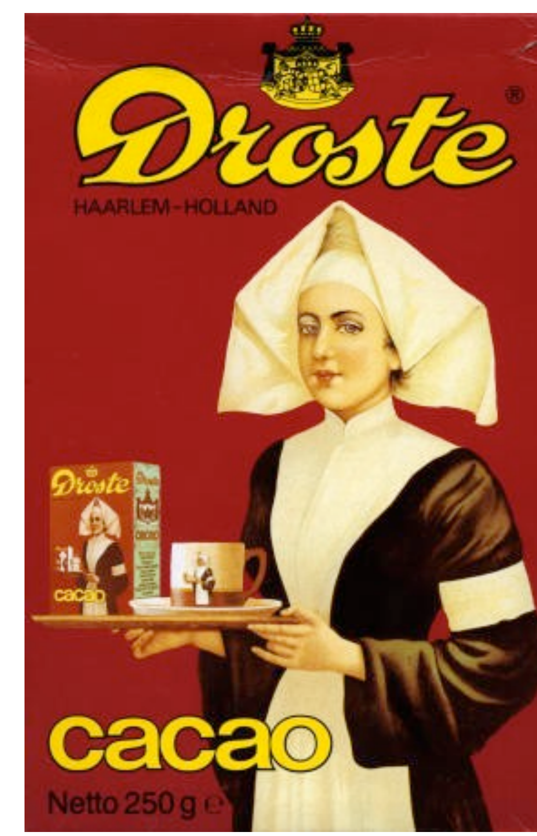


Recursion…

n! = n×(n-1)×(n-2)×…×1

n! = n×((n-1)!)	"inductive definition"
0! = 1	"base case"





Math Induction = CS Recursion

Math

inductive definition

```
0! = 1

n! = n × (n-1)!)
```

Python (Functional)

recursive function

```
# 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)
```

"To understand recursion, you must first understand
" - anonymous

Mudd alum

```
# 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)
```

"To understand recursion, you must first understand recursion" - anonymous Mudd alum

```
# 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)
```

```
# 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)
                                      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)
                        2
                 return 2 * factorial(1)
                                      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):               6
    return 3 * factorial(2)
                                2
                  return 2 * factorial(1)
                                             1
                             return 1 * factorial(0)
```

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

A Tower of Fun!

Math

$\texttt{tower(3)} = 2^{2^{2}} = 2^4 = 16$

$\texttt{tower(4)} = 2^{2^{2^{2}}} = 2^{16}$

$\texttt{tower(5)} = 2^{2^{2^{2^{2}}}} = 2^{(2^{16})}$

inductive definition:

Python (Functional)

recursive function

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

Aside: tower using reduce

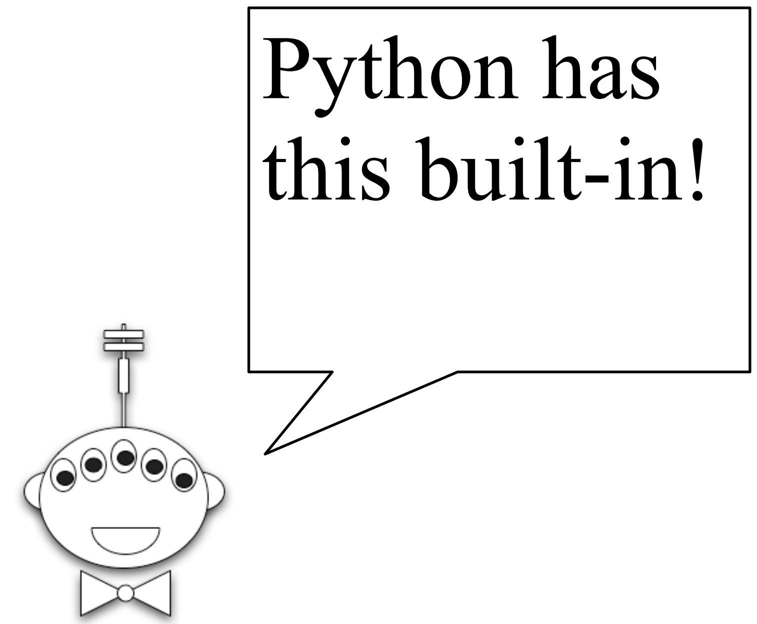
```
def pow(x, y):
    return x**y

>>> reduce(pow, [2, 2, 2, 2])
???


>>> 2 ** 3 ** 2
510. # which is 2**(3**2),
    # not (2**3)**2
```

Computing the length of a list

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




```
def len(lst):
    """returns the length of lst"""
```

Hint: view the list recursively, as `[first] + rest`