return vs print…

```
def dbl(x):
    return 2 * x
```

```
def trbl(x):
    print 2 * x
```

```
def happy(input):
    y = dbl(input)
    return y + 42
```

```
def sad(input):
    y = trbl(input)
    return y + 42
```


```
def friendly(input):
    y = dbl(input)
    print y, "is very nice!"
    return y + 42
```

Strings are in single or double quotes

Mapping with Python...

```
def dbl(x):
  """returns 2 * x"""
  return 2 * x

>>> map(dbl, [0, 1, 2, 3, 4])
[0, 2, 4, 6, 8]

def evens(n):
  myList = range(n)
  doubled = map(dbl, myList)
  return doubled
```

Alternatively….

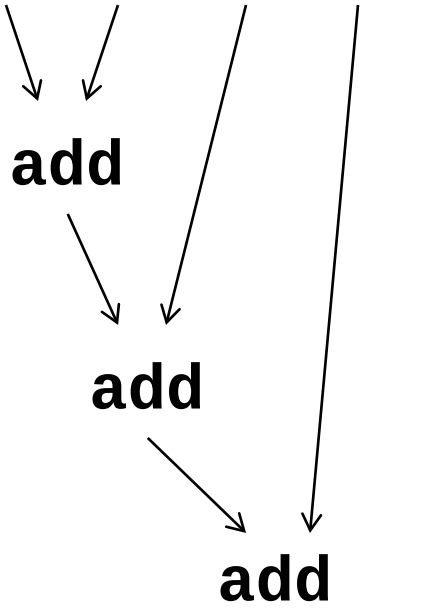
```
def evens(n):
  return map(dbl, range(n))
```

reduce-ing with Python...

```
def add(x, y):
  """returns x + y"""
  return x + y

>>> reduce(add, [1, 2, 3, 4])
10
```

reduce-ing with Python...

```
def add(x, y):
  """returns x + y"""
  return x + y

>>> reduce(add, [1, 2, 3, 4])
```



Try this…

Write a function called `span` that returns the difference between the maximum and minimum numbers in a list…

```
>>> span([3, 1, 42, 7])
41
>>> span([42, 42, 42, 42])
0
```

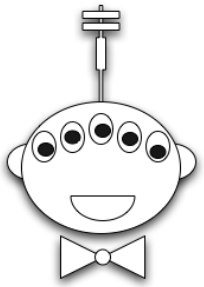
```
min(x, y)
max(x, y)
```

These are built-in to Python!

Try this...

1. Write a python function called `gauss` that takes as input a positive integer N and returns the sum
   $1 + 2 + \ldots + N$

2. Write a python function called `sumOfSquares` that takes as input a positive integer N and returns the sum
   $1^2 + 2^2 + 3^2 + \ldots + N^2$

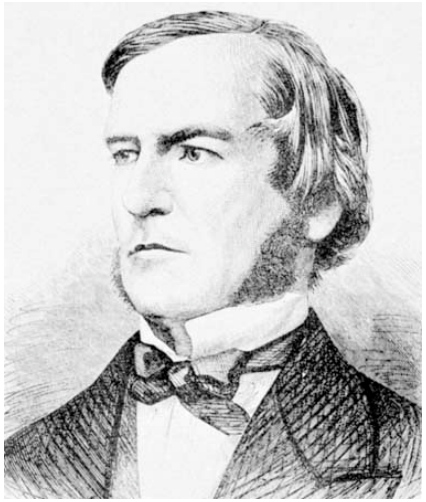


You can write extra "helper" functions too!

Booleans

```
>>> 3 == 1+2
True
>>> 42 == "spam"
False
>>> “spam” > 42
True
```

Strings!



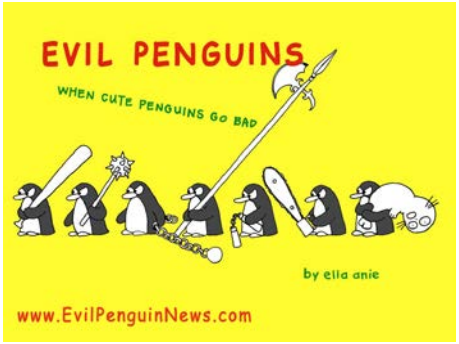
George Boole
1815-1864

Strings > lists > numbers

The “Truth” about Python’s Booleans

```
>>> True + 41
42
>>> 2 ** False == True
True
```

Demonstrating the True “power” of Falsity!



Lists Revisited!

```
>>> L = [1, 42, 3, 4]
>>> L
[1, 42, 3, 4]
>>> L + 10
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: can only concatenate list (not "int") to list
>>> L + [50]
[1, 42, 3, 4, 50]
>>> L
[1, 42, 3, 4]
>>> L*2
[1, 42, 3, 4, 1, 42, 3, 4]
>>> M = [42, "hello", 3+2j, 3.141, [1, 2, 3, 4, 5, 6]]
```

L doesn't change!

Lists are "polymorphic"

Lists Revisited!

```
>>> L = [1, 42, 3, 4]
>>> L
[1, 42, 3, 4]
>>> L + 10
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: can only concatenate list (not "int") to list
>>> L + [50]
[1, 42, 3, 4, 50]
>>> L
[1, 42, 3, 4]
>>> L*2
[1, 42, 3, 4, 1, 42, 3, 4]
>>> M = [42, "hello", 3+2j, 3.141, [1, 2, 3, 4, 5, 6]]
```

L doesn't change!

Lists are "polymorphic"

Lists Revisited!

```
>>> L = [1, 42, 3, 4]
>>> L
[1, 42, 3, 4]
>>> L + 10
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: can only concatenate list (not "int") to list
>>> L + [50]
[1, 42, 3, 4, 50]
>>> L
[1, 42, 3, 4]
>>> L*2
[1, 42, 3, 4, 1, 42, 3, 4]
>>> M = [42, "hello", 3+2j, 3.141, [1, 2, 3, 4, 5, 6]]
```

L doesn't change!

Lists are "polymorphic"

Lists Revisited!

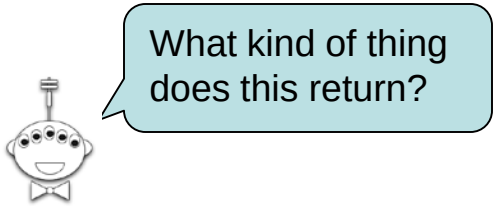
```
>>> L = [1, 42, 3, 4]
>>> L
[1, 42, 3, 4]
>>> L + 10
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: can only concatenate list (not "int") to list
>>> L + [50]
[1, 42, 3, 4, 50]
>>> L
[1, 42, 3, 4]
>>> L*2
[1, 42, 3, 4, 1, 42, 3, 4]
>>> M = [42, "hello", 3+2j, 3.141, [1, 2, 3, 4, 5, 6]]
```

L doesn't change!

Lists are "polymorphic"

List Indexing and Slicing!

```
            0   1   2   3
>>> M = [42, 3, 98, 37 ]

>>> M[0]

>>> M[2]

>>> M[0:2]

>>> M[0:3:2]

>>> M[1:]

>>> M[:-1]

>>> M[1:-2]

>>>
```



Try to reverse the list!

Strings Revisited

```
>>> S = “I love Spam!”
          0 1 2 3 4 5 6 7 8 9 1 1 1
                              0 1 2
>>> S[0]

>>> S[13]

>>> S[2:6]

>>> S[12:6:-1]
```

if, elif, else…

```
def special(x):
      """This function demonstrates the use
      of if and else"""
      if x == 42:
            return "Very special number!"
      else:
            return "Stupid, boring number."
```

```
def special(x):
      if x == 42:
            return "Very special number!"
      return "Stupid, boring number."
```

Alternatively??

Notice how lines with the same level of indentation are in the same code block!

if, elif, else…

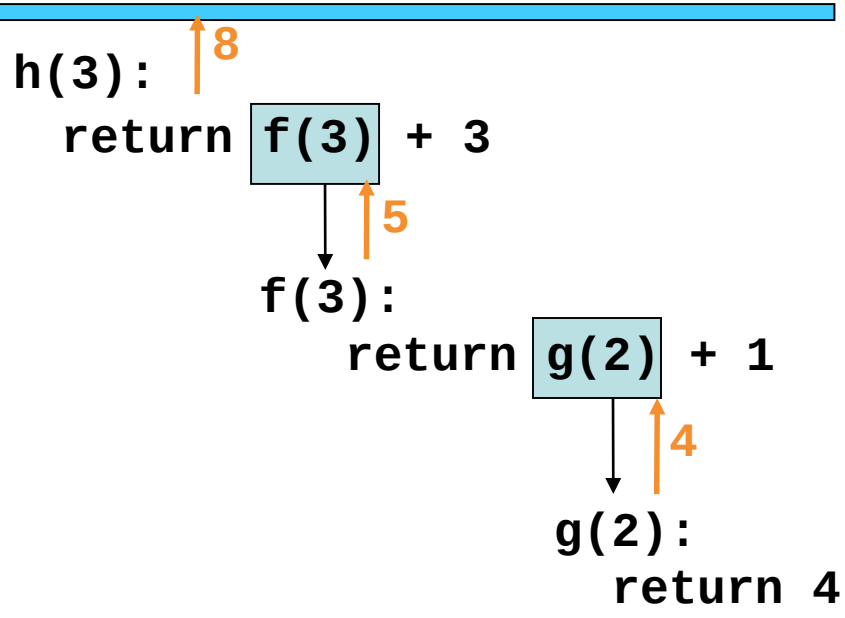
```
def superSpecial(x):
      """This function demonstrates the use
      of if, elif, and else"""
      if x < 42:
            return "Small number"
      elif x == 42 or x % 42 == 0:
            return "Nice!"
      elif 41 <= x <= 43:
            return "So close!"
      else:
            # We might do more stuff here…
            return "Yuck!"
```

Would swapping the order of these elif's give the same behavior?

Notice how lines with the same level of indentation are in the same code block!

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

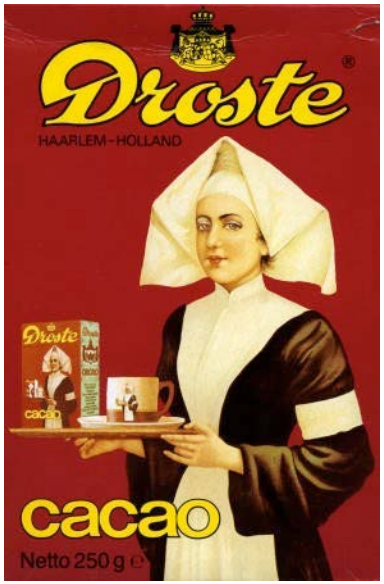
Recursion…

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

Recursion…

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

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

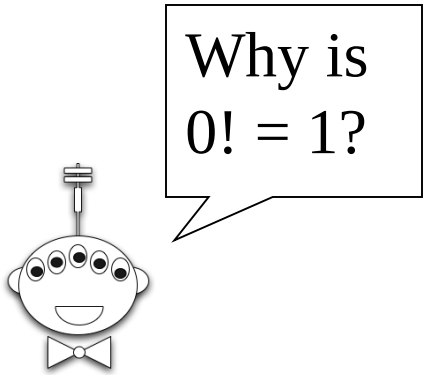


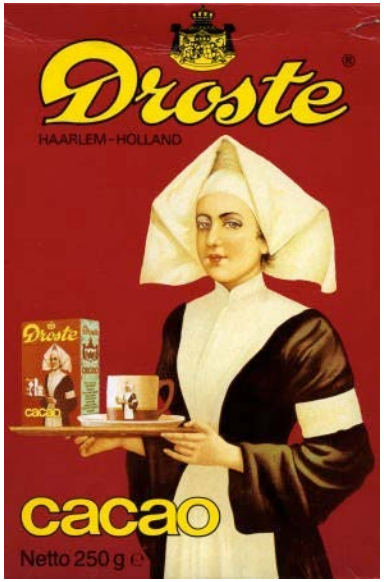
Recursion…

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

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

$0! = 1$ “base case”





Math Induction = CS Recursion

Math

inductive definition

$0! = 1$

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

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)
```

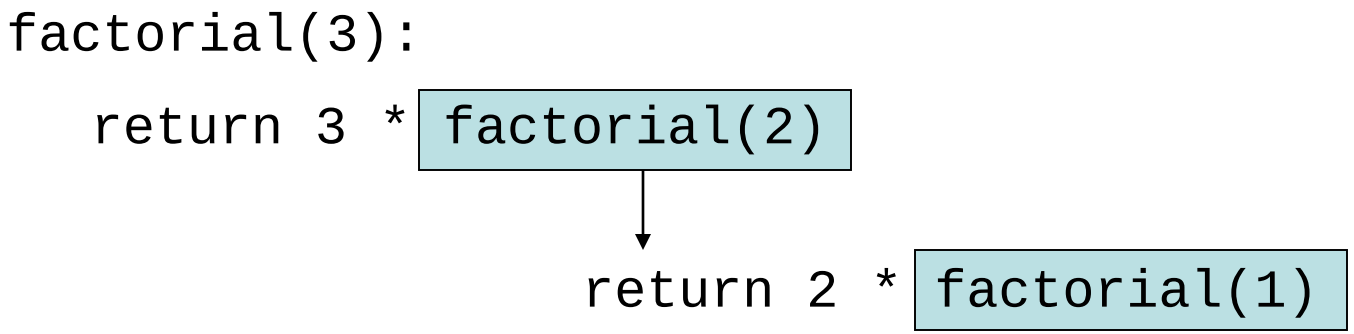
Is Recursion Magic?

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

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

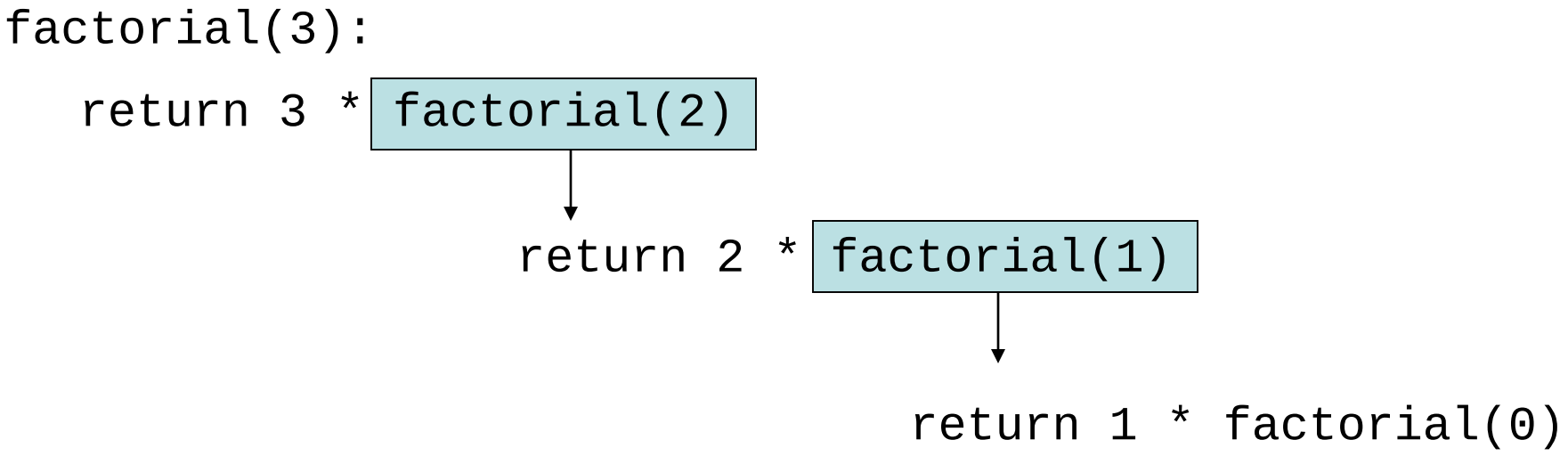
```
# recursive factorial
def factorial(n):
  if n == 0:
    return 1
  else:
    return n*factorial(n-1)
```

Is Recursion Magic?



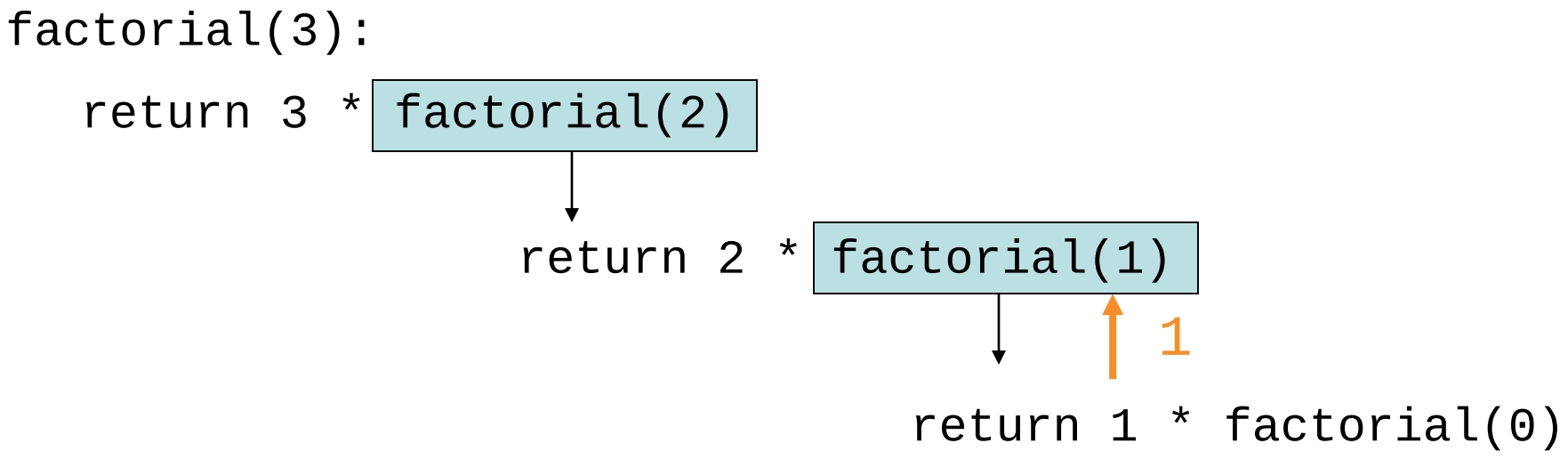

```
# recursive factorial
def factorial(n):
  if n == 0:
    return 1
  else:
    return n*factorial(n-1)
```

Is Recursion Magic?



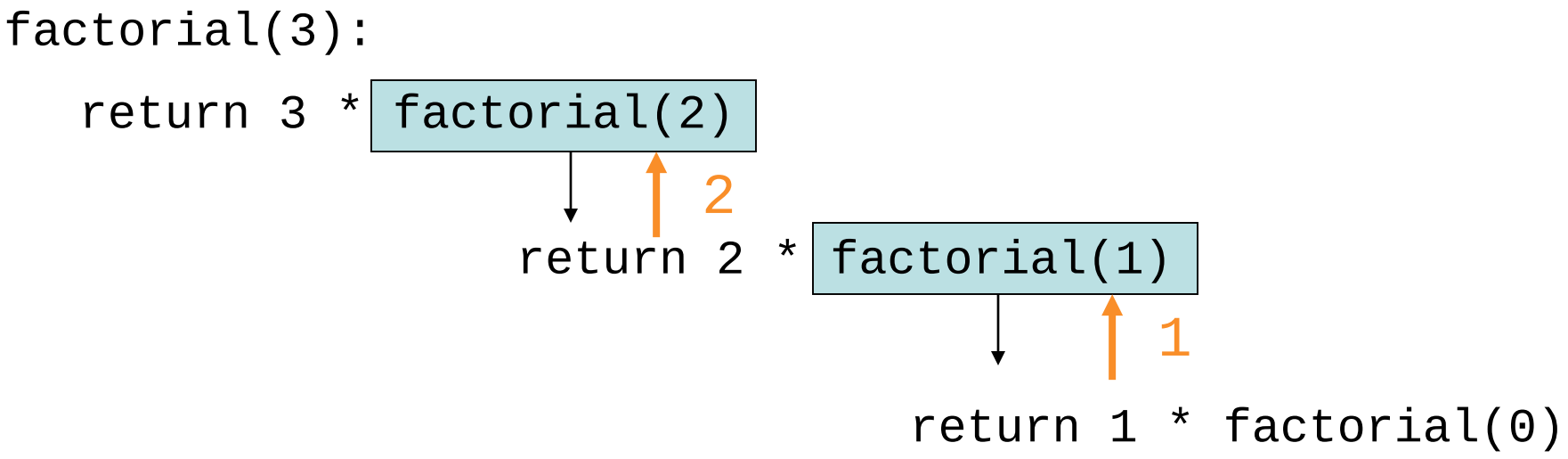

```
# recursive factorial
def factorial(n):
  if n == 0:
    return 1
  else:
    return n*factorial(n-1)
```

Is Recursion Magic?



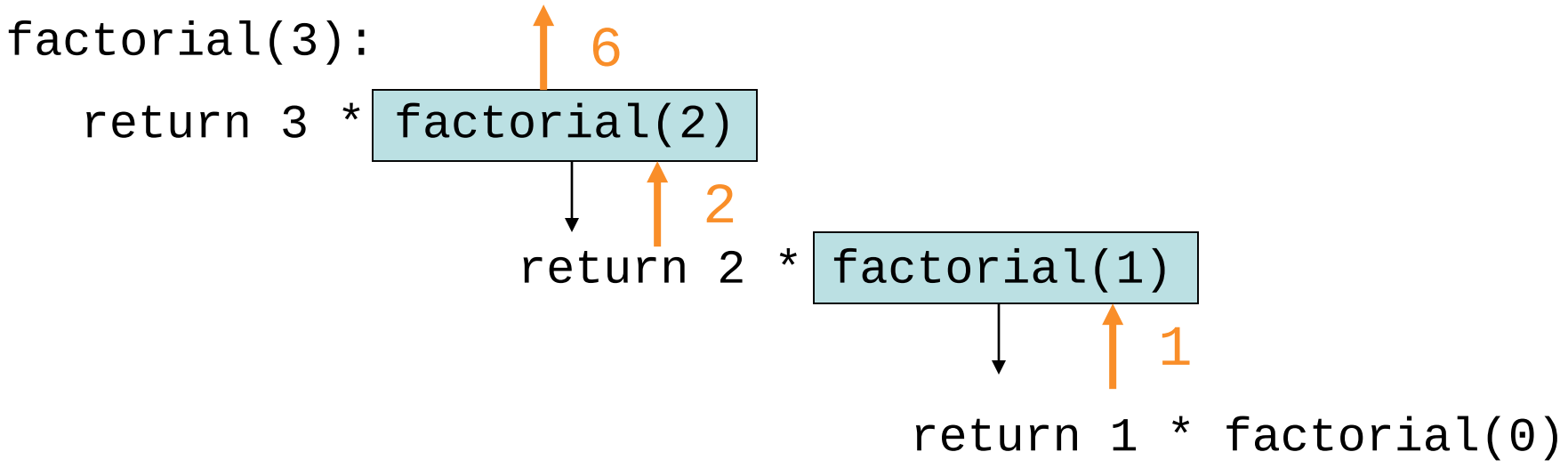

```
# recursive factorial
def factorial(n):
  if n == 0:
    return 1
  else:
    return n*factorial(n-1)
```

Is Recursion Magic?




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

Is Recursion Magic?




```
# 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

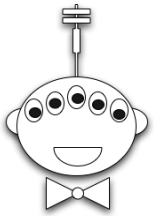
```
# recursive factorial
def tower(n):
```

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`

Reversing a list

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

def reverse(lst):
  """returns a new list that is the
     reverse of the input list"""
```

Reversing a list

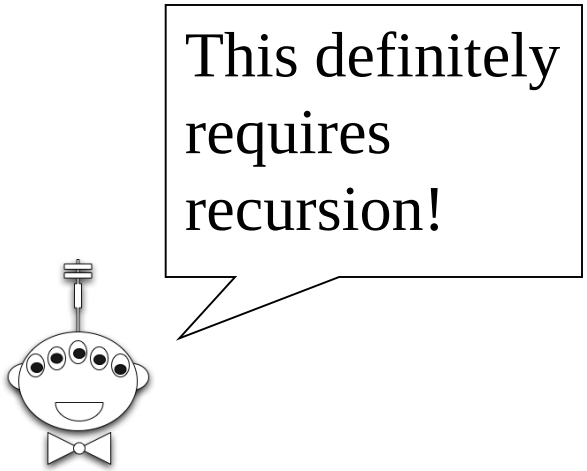
```
>>> reverse([1, [2, [4, 5], 6], 7])
```

Deep Reversing a list

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

Deep Reversing a list

```
>>> 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])
```




```
if type(L) == type([]): # True if L is a list and False otherwise
```

Recursion = :^)

Recursion, conditional statements, and lists suffice to give us a Turing-complete programming language!

Variables, assignment (=), if, while, etc. are all unnecessary!