Recursive Void function

What will this output?

```
def hello(n):
    if n > 1:
        print 'hello'
        hello(n-1)  # recursive case
    else:
        print 'hello'
        return      # base case

hello(5)
```

Output:

```
hello
hello
hello
hello
hello
```

Observing Flow of Execution

Add print statements to see what happens:

```
def hello(n):
    space = ' ' * (2 * n)
    print space, 'starting hello', n
    if n > 1:
        print space, 'recursing hello', n
        hello(n-1)
    else:
        print space, 'base hello', n
        return
hello(5)
```

Output:

```
          starting hello 5
          recursing hello 5
        starting hello 4
        recursing hello 4
      starting hello 3
      recursing hello 3
    starting hello 2
    recursing hello 2
  starting hello 1
  base hello 1
```

Recursive Valued Function

Definition of the counter() function:

```
counter(0) = 0
counter(n) = n + counter(n-1), n > 0
```

Similar to definition of factorial:

```
0! = 1
n! = n (n-1)!, n > 0
```

A Python Counter() Function

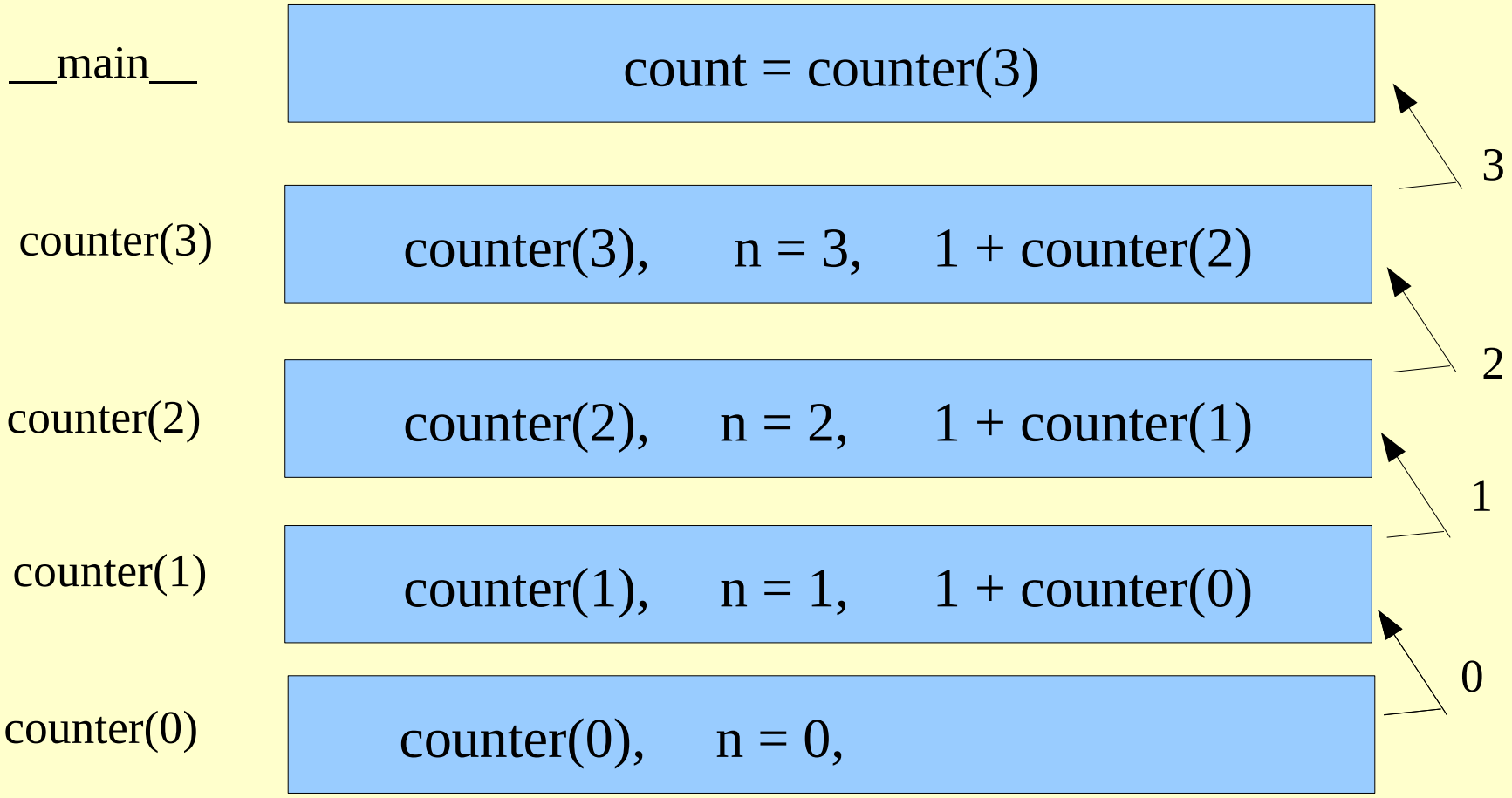
What would this output?

```
def counter(n):
    if n > 0:
        count = 1 + counter(n-1)  # recursive case
        return count
    else:
        return 0                  # base case


count = counter(100)
print 'count = ' , count
```

Output: `count = 100`

Recursion Stack



Observing Flow of Execution

What would this output?

```
def counter(n):
    space = ' ' * (2 * n)
    print space, 'counter', n
    if n > 0:
        count = 1 + counter(n-1)
        print space, 'returning', count
        return count
    else:
        print space, 'returning 0'
        return 0

count = counter(5)
print 'count = ', count
```

Output:

```
counter 5
 counter 4
  counter 3
   counter 2
    counter 1
     counter 0
     returning 0
    returning 1
   returning 2
  returning 3
 returning 4
returning 5
count =  5
```

Don't Need the Local Variable

The recursion stack stores the intermediate values.

```
def counter(n):
    if n > 0:
        return 1 + counter(n-1)   # recursive case
    else:
        return 0                  # base case

count = counter(100)
print 'count = ' , count
```

A Base 2 Log Function

- Not all recursion involves n-1
- Definition of the $\log_2 n$ function for integer n of the form $n = 2^k$:

base: $\log_2 1 = 0$

recursive: $\log_2 n = 1 + \log_2 n/2$

$2^0 = 1$

$2^1 = 2$

$2^2 = 4$

$2^3 = 8$

A Python $\log_2$ Function

Not a very useful function. Works only for powers of 2. What would this output?

```
def log_base2(n):
    if n > 1:
        return 1 + log_base2(n/2)  # recursive case
    else:
        return 0                   # base case


print 'log_2(8) = ', log_base2(8)
print 'log_2(64) = ', log_base2(64)
print 'log_2(1) = ', log_base2(1)
```

Output:

```
log_2(8) =  3
log_2(64) =  6
log_2(1) =  0
```

A Python $\log_2$ Function

Works only for powers of 2. What would this output?

```
def log_base2(n):
    if n > 1:
        return 1 + log_base2(n/2)  # recursive case
    else:
        return 0                   # base case


print 'log_2(8.0) = ', log_base2(8.0)
print 'log_2(8.5) = ', log_base2(8.5)
```

Output:

```
log_2(8.0) = 3
log_2(8.5) = 4
```

In Class Exercises

- Write a recursive function to compute 2^n for positive integers n.
- Write a recursive function to compute x^n for positive integers n.
- Use incremental development!