Recursion 1

Lab 10

Recursive thinking

- *Recursion* is a programming technique in which a function can call itself to solve a problem
- A *recursive definition* is one which uses the word or concept being defined in the definition itself
- In some situations, a recursive definition can be an appropriate way to express a concept
- Before applying recursion to programming, it is best to practice thinking recursively

Recursive programming

- A function in Python can call itself; if written that way, it is called a *recursive* function
- A recursive function solves some problem.

- The code of a recursive function should be written to handle the problem in one of two ways:
  - **Base case**: a simple case of the problem that can be answered directly; does not use recursion.
  - **Recursive case**: a more complicated case of the problem, that isn't easy to answer directly, but can be expressed elegantly with recursion; makes a recursive call to help compute the overall answer

Recursive factorial

- *factorial* can also be defined recursively:

$$f(n) = \begin{cases} n \geq 1 \Rightarrow n \times f(n-1) \\ n = 0 \Rightarrow 1 \end{cases}$$

- A factorial is defined in terms of another factorial until the basic case of 0! is reached

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

Q1

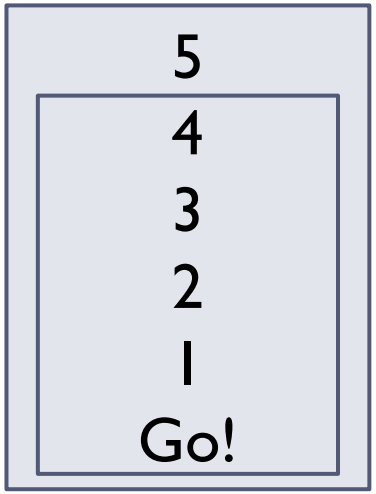
- count_down – Write a function which takes a number, n, and prints "the numbers n to 1 (inclusive) and "GO"
  - E.g. count_down(3) would print:
    - 3
    - 2
    - 1
    - Go!
  - What is the base case?
  - What is the recursive case?

Count Down – Thinking about the Problem

count_down(5) will print:

5
4
3
2
1
Go!

We can think about this as printing the value "5" and then printing count_down(4)



Count Down

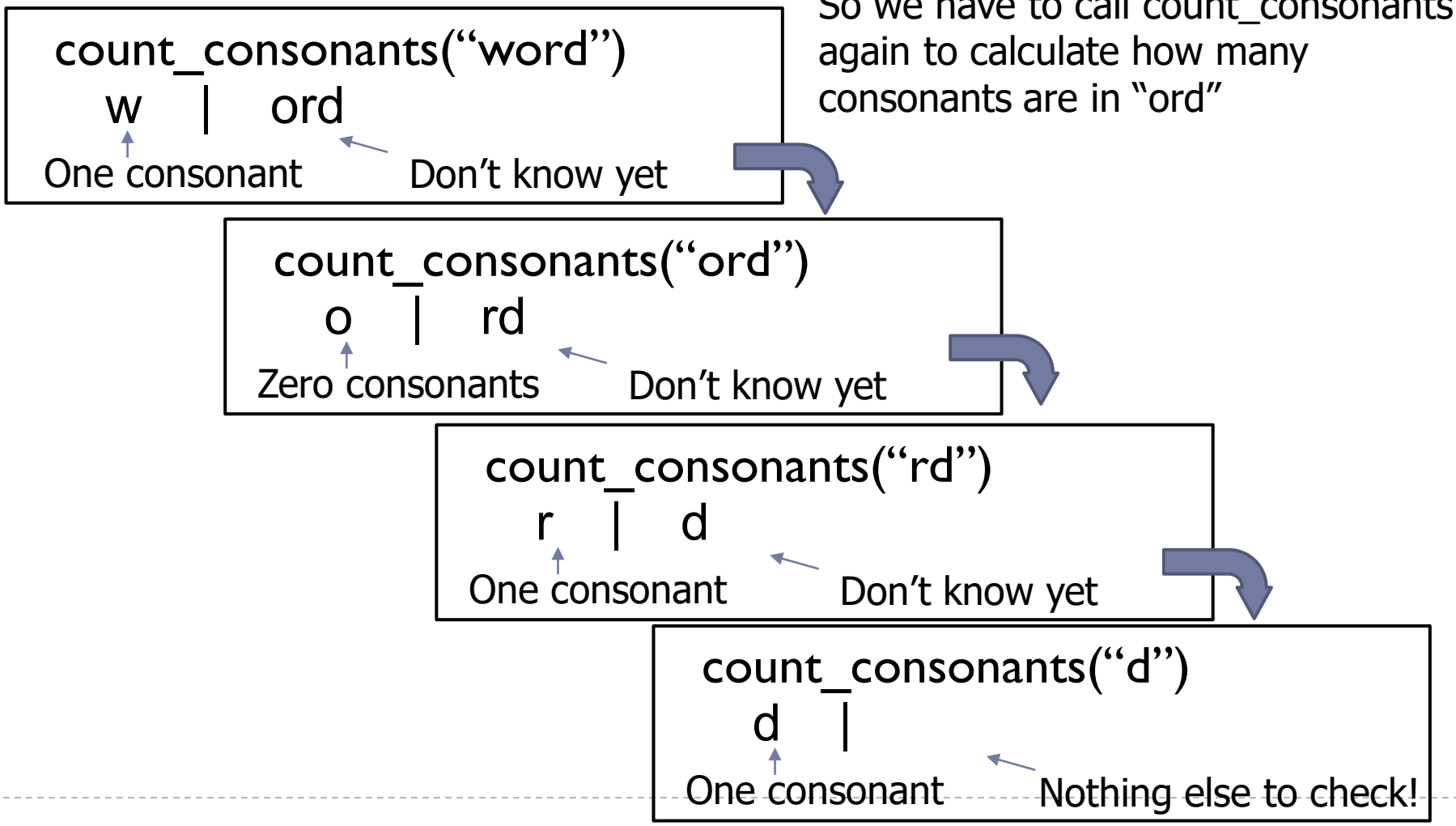
- We need to tell recursive functions when to stop

```
count_down(n):
        If n == 0:
                print "Go!"
        Else:
                print(n)

                Then do  count_down(n-1)
```

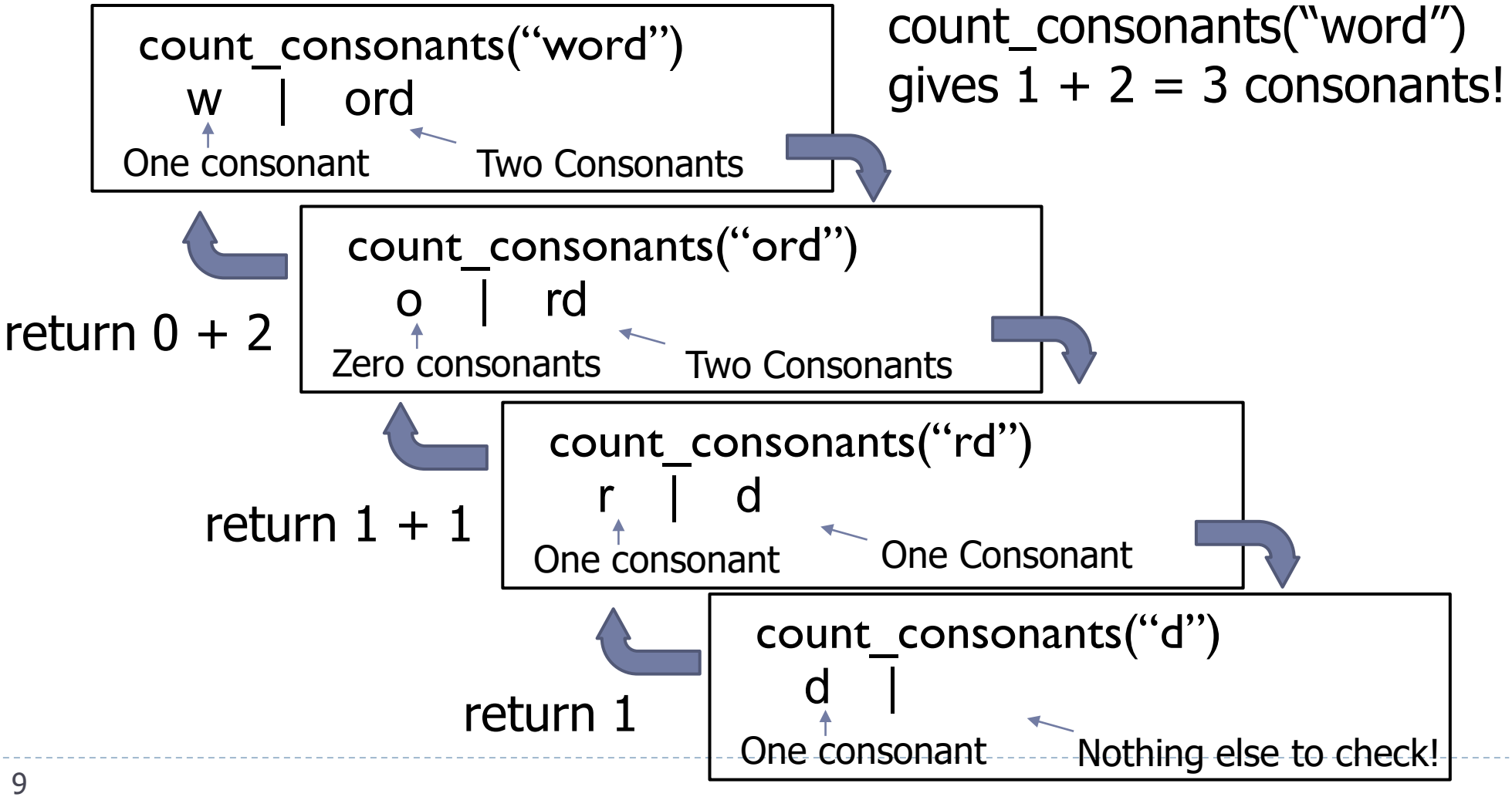
count_consonants

- Work Flow:



count_consonants

- Work Flow:



get_max_list

- Work Flow:

return 5

```
get_max_list([1, 5, 4, -9])
   Max(1, get_max_list([5, 4, -9]))
```

return 5

```
get_max_list([5, 4, -9])
   Max(5, get_max_list([4, -9]))
```

return 4

```
get_max_list([4, -9])
   Max(4, get_max_list([-9]))
```

return -9

```
get_max_list([-9])
   -9 # one thing in the list
```

Coderunner Tips

- Q1: The function just needs to print – it doesn't need to return anything
- Q2 & Q3: These are very similar – think about how you can change the recursive line to reverse the string (swap the order you add things)
- Q4: Think about how one of the input values can change to go through each number
- Q6: the base case can be when you find a lower-case letter
- Q10: you might have to go backwards through the equation