**Practice 4 - Recursion**

**Practice 1** - Write a function that takes two numbers m and n and returns their product. Assume m and n are positive integers. **Use recursion**, not mul or \*!

**Hint: 5\*3 = 5 + 5\*2 = 5 + 5 + 5\*1.**

For the base case, what is the simplest possible input for multiply?

For the recursive case, what does calling multiply(m - 1, n) do?  
What does calling multiply(m, n - 1) do? Do we prefer one over the other?

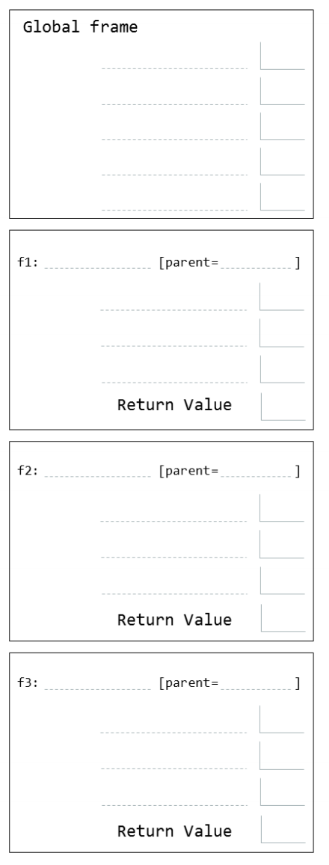
| def multiply(m, n):  """ >>> multiply(5, 3)  15  """  if n == 1: return m  else: return m + multiply(m, n-1) |
| --- |

**Practice 2** - Draw an environment diagram for the following code:

| def rec(x, y):  if y > 0:  return x \* rec(x, y - 1)  return 1  rec(3, 2) |
| --- |

Bonus question: what does this function do?

***Note****: This problem is meant to help you understand what really goes on when we make the ”recursive leap of faith”. However, when approaching or debugging recursive functions, you should avoid visualizing them in this way*



**Practice 3** - Below function returns True if a positive integer n is a prime number and False otherwise.

| def is\_prime(n):  """  >>> is\_prime(10)  False  >>> is\_prime(7)  True  """  if n == 1:  return False  k = 2  while k < n:  if n % k == 0:  return False  k += 1  return True |
| --- |

Now, implement it recursively!

| def is\_prime(n):  """ >>> is\_prime(7)  True  >>> is\_prime(10)  False  >>> is\_prime(1)  False  """  def prime\_helper(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_):  if \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  elif \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  else: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  return \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| --- |

## Tree Recursion

**Practice 4** - You want to go up a flight of stairs that has n steps. You can either take 1 or 2 steps each time. How many different ways can you go up this flight of stairs? Write a function count\_stair\_ways that solves this problem. Assume n is positive.

Before we start, what’s the base case for this question? What is the simplest input?

What do count\_stair\_ways(n - 1) and count\_stair\_ways(n - 2) represent?

Use those two recursive calls to write the recursive case:

| def count\_stair\_ways(n):  """  >>> count\_stair\_ways(1)  1  >>> count\_stair\_ways(2)  2  >>> count\_stair\_ways(3)  3  >>> count\_stair\_ways(4)  5  """ |
| --- |

**Practice 5** - Consider a special version of the count\_stairways problem, where instead of taking 1 or 2 steps, we are able to take up to and including k steps at a time.

Write a function count\_k that figures out the number of paths for this scenario. Assume n and k are positive.

| def count\_k(n, k):  """ >>> count\_k(3, 3) # 3, 2 + 1, 1 + 2, 1 + 1 + 1  4  >>> count\_k(4, 4) 8 >>> count\_k(10, 3)  274  >>> count\_k(300, 1) # Only one step at a time  1  """ |
| --- |