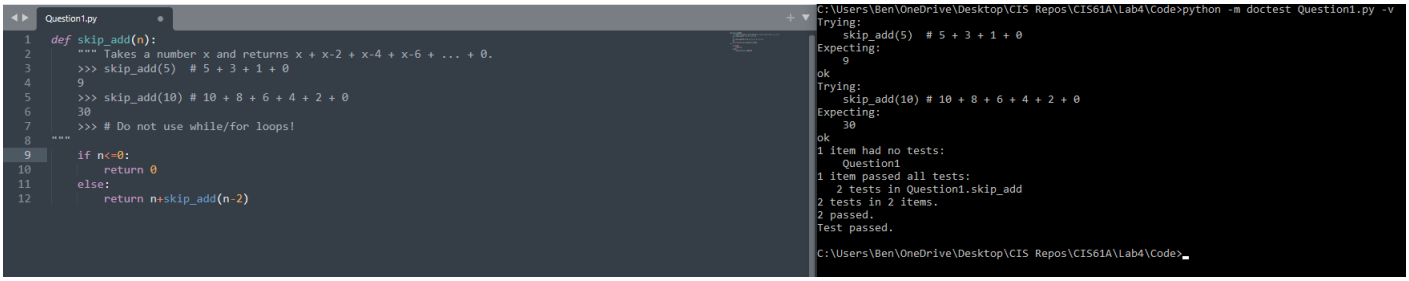
CIS 61 :: Lab 04 - Recursion - Template

Student Name:

Instructions: Use Recursion to find solutions to below functions. Attached screenshots of your code and your test run. Make sure the screenshots are readable.

Q1: Skip Add

Write a function `skip_add` that takes a single argument `n` and computes the sum of every other integer between `0` and `n`. Assume `n` is non-negative.



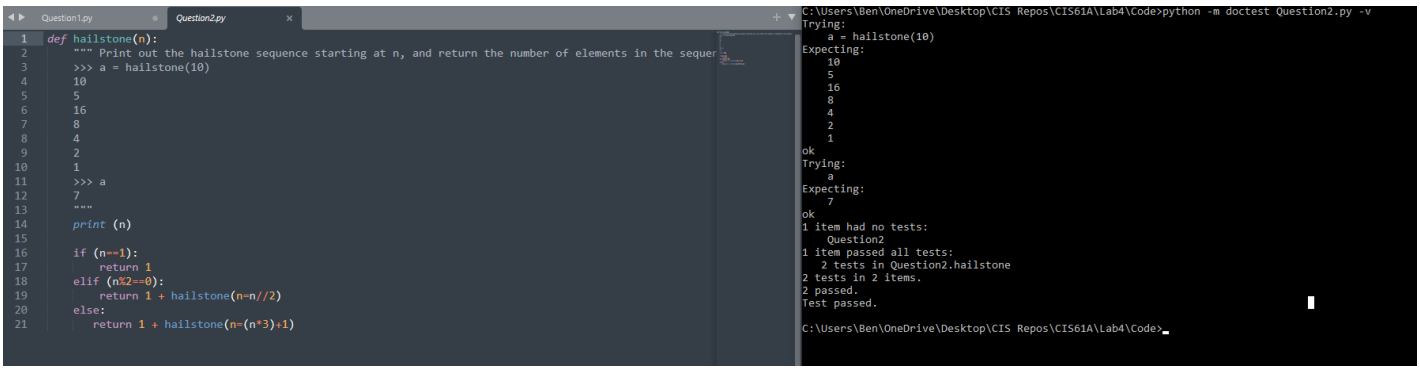
`Code:`

`Test Run:`

Q2: Hailstone

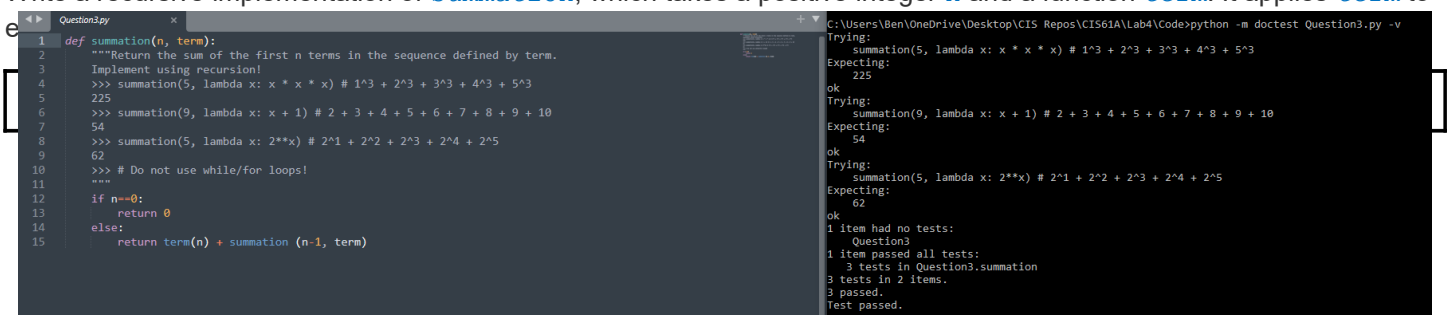
Recall the `hailstone` function from Lab 2. First, pick a positive integer `n` as the start. If `n` is even, divide it by 2. If `n` is odd, multiply it by 3 and add 1. Repeat this process until `n` is 1. Write a recursive version of hailstone that prints out the values of the sequence and returns the number of steps.

Hint: When taking the recursive leap of faith, consider both the return value and side effect of this function.



Q3: Summation

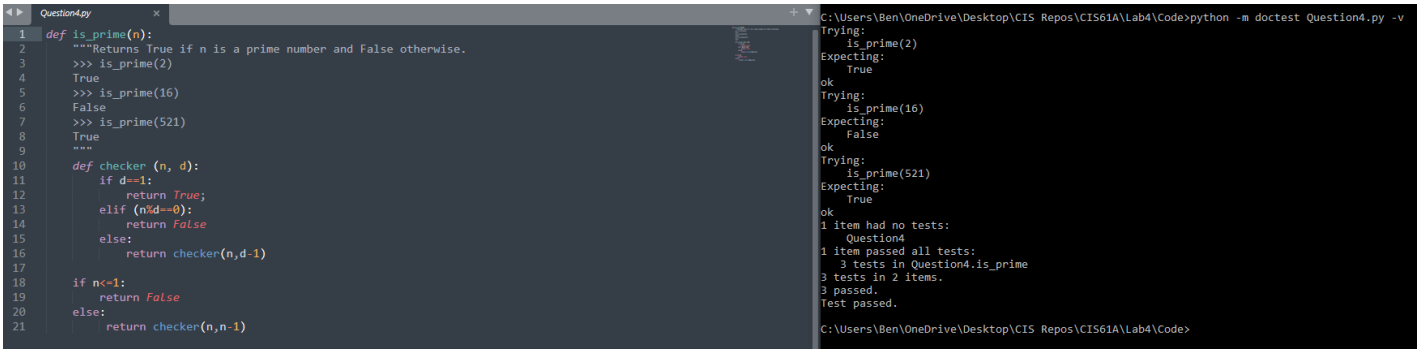
Write a recursive implementation of `summation`, which takes a positive integer `n` and a function `term`. It applies `term` to e



Q4: Is Prime

Write a function `is_prime` that takes a single argument `n` and returns `True` if `n` is a prime number and `False`otherwise. Assume `n > 1`. We implemented this iteratively before, now time to do it recursively!

Hint: You will need a helper function! Remember helper functions are useful if you need to keep track of more variables than the given parameters, or if you need to change the value of the input



Q5: GCD

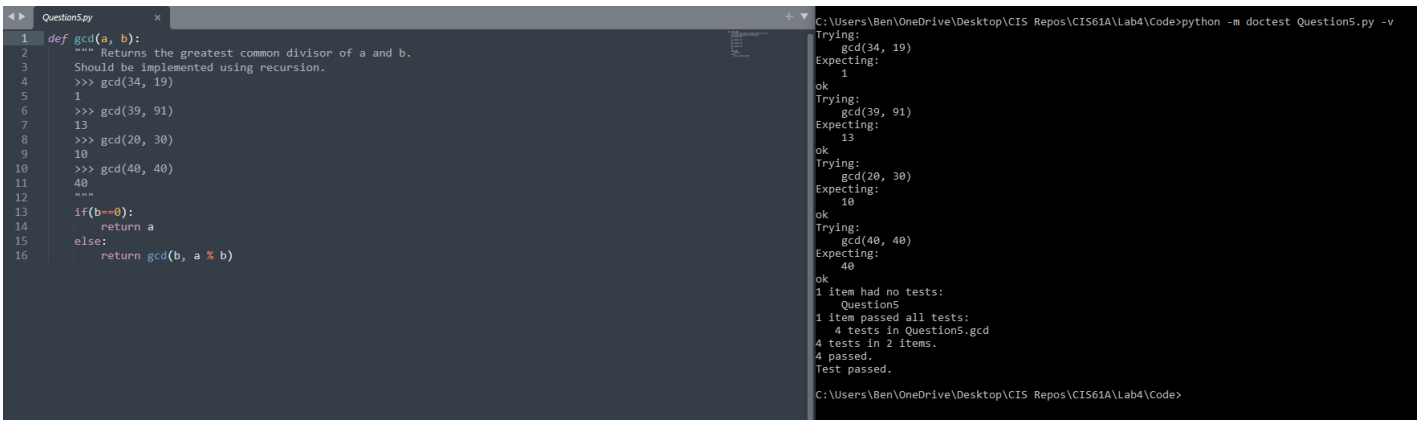
The greatest common divisor of two positive integers `a` and `b` is the largest integer which evenly divides both numbers (with no remainder). Euclid, a Greek mathematician in 300 B.C., realized that the greatest common divisor of `a` and `b` is one of the following:

- the smaller value if it evenly divides the larger value, or
- the greatest common divisor of the smaller value and the remainder of the larger value divided by the smaller value

In other words, if `a` is greater than `b` and `a` is not divisible by `b`, then

```
gcd(a, b) = gcd(b, a % b)
```

Write the `gcd` function recursively using Euclid's algorithm.



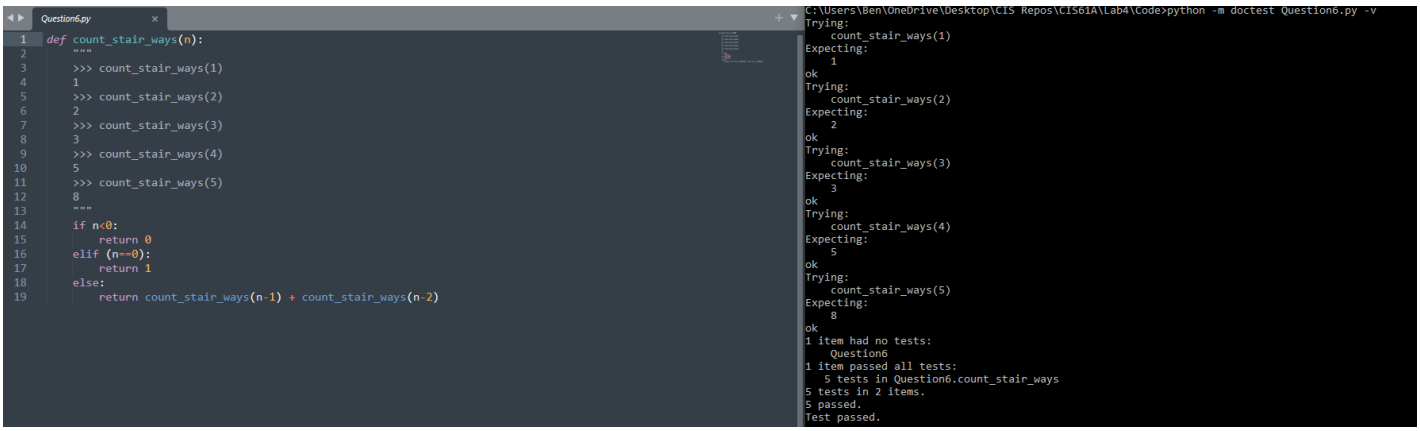
Q6: Count Stairs

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:



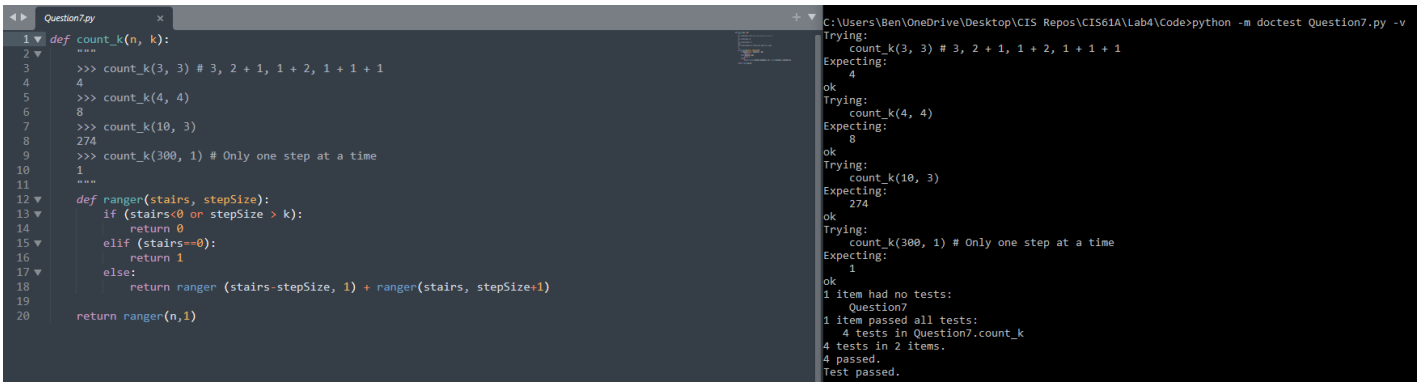
```
Question6.py
def count_stair_ways(n):
    """
    >>> count_stair_ways(1)
    1
    >>> count_stair_ways(2)
    2
    >>> count_stair_ways(3)
    3
    >>> count_stair_ways(4)
    5
    >>> count_stair_ways(5)
    8
    """
    if n<0:
        return 0
    elif (n==0):
        return 1
    else:
        return count_stair_ways(n-1) + count_stair_ways(n-2)
```

```
C:\Users\Ben\OneDrive\Desktop\CIS Repos\CIS61A\Lab4\Code>python -m doctest Question6.py -v
Trying:
    count_stair_ways(1)
Expecting:
    1
ok
Trying:
    count_stair_ways(2)
Expecting:
    2
ok
Trying:
    count_stair_ways(3)
Expecting:
    3
ok
Trying:
    count_stair_ways(4)
Expecting:
    5
ok
Trying:
    count_stair_ways(5)
Expecting:
    8
ok
1 item had no tests:
    Question6
1 item passed all tests:
   5 tests in Question6.count_stair_ways
5 tests in 2 items.
5 passed.
Test passed.
```

Q7: Count Stairs with k steps

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.
Tip: You may need to use a while loop in your solutions.



```
Question7.py
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
    """
    def ranger(stairs, stepSize):
        if (stairs<0 or stepSize > k):
            return 0
        elif (stairs==0):
            return 1
        else:
            return ranger (stairs-stepSize, 1) + ranger(stairs, stepSize+1)

    return ranger(n,1)
```

```
C:\Users\Ben\OneDrive\Desktop\CIS Repos\CIS61A\Lab4\Code>python -m doctest Question7.py -v
Trying:
    count_k(3, 3) # 3, 2 + 1, 1 + 2, 1 + 1 + 1
Expecting:
    4
ok
Trying:
    count_k(4, 4)
Expecting:
    8
ok
Trying:
    count_k(10, 3)
Expecting:
    274
ok
Trying:
    count_k(300, 1) # Only one step at a time
Expecting:
    1
ok
1 item had no tests:
    Question7
1 item passed all tests:
   4 tests in Question7.count_k
4 tests in 2 items.
4 passed.
Test passed.
```