CSM!!!

↳ Piazza @580

MT!!!

↳ Practice @ 736

Recursion:

3 things! 1 1 2 3 5 def fib(n):

fib(_)

1) Base case

↳ "smallest" input

↳ what input requires no work?

2) recursive call

↳ fib(———)

↑ changing input

recursive leap of faith

3) do the last bit of work

fib(n-1) + fib(n-2)

def func1( )

return func1

def func2()

func2()

Q3: (Tutorial) Warm Up: Recursive Multiplication

These exercises are meant to help refresh your memory of topics covered in lecture and/or lab this week before tackling more challenging problems.

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?

Your Answer: "What input do I know the answer to immediately?"

m or n == 0 → return 0 m or n == 1, return other

Log in to save your work! (https://disc.cs61a.org/oauth/login)

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?

(in relation to our end goal) m·n

Your Answer:

Log in to save your work! (https://disc.cs61a.org/oauth/login)

```
1   def multiply(m, n):
2       """ Takes two positive integers and returns their product using recursion.
3       >>> multiply(5, 3)
4       15
5       """
6       "*** YOUR CODE HERE ***"
```

(m-1)(n) m·n

(m-1)(n)

mn - n + n → m·n

m(n-1) + m m·n

2·10 → 1·10 vs. 10·2

9·2 → 8·2 → 7·2

```
if m==0 or n==0:
    return 0
elif m==1:
    return n
elif n==1:
    return m
else:
    return multiply(m-1, n) + n
```

m-1 n
m-2 n
m-3 n
⋮
1

m · 1 m-x · 1
m-1 · 1
m-1 · 1
1 · 1

Q4: (Tutorial) Recursive Hailstone

Recall the `hailstone` function from Homework 1. 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.

```
def hailstone(n):
    """Print out the hailstone sequence starting at n, and return the number of elements
    >>> a = hailstone(10)
    10
    5
    16
    8
    4
    2
    1
    >>> a
    7
    """
    "*** YOUR CODE HERE ***"
```

n//2

3*n+1

~~hailstone(n-1)~~

hailstone(n//2)

hailstone(3n+1)

```
print(n)
if n == 1:
    return 1
elif n % 2 == 0:
    return hailstone(n//2) + 1
else:
    return hailstone(3*n+1)+1
```

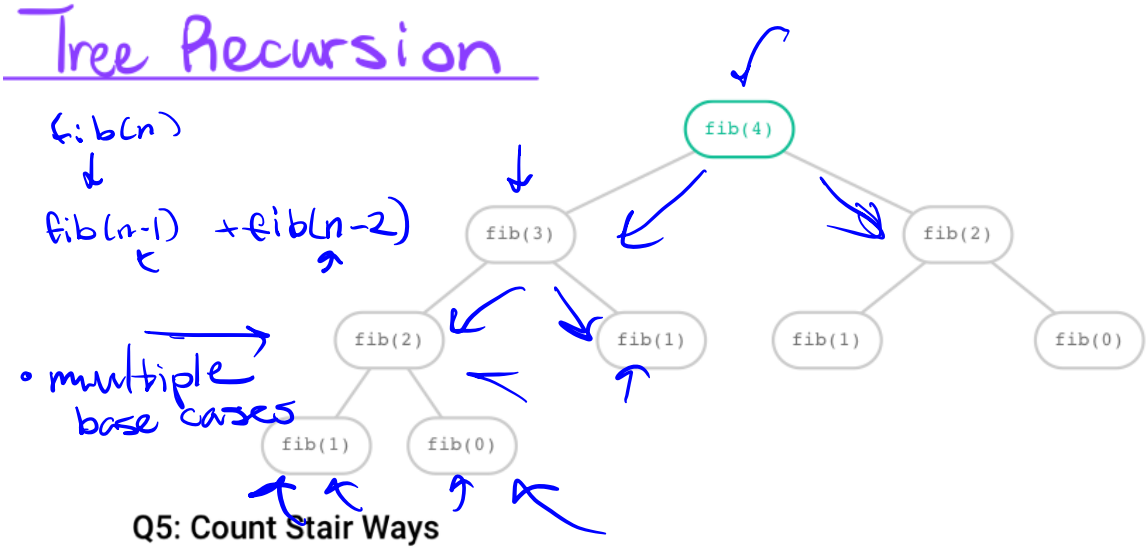
linear recursion

```
def tail_hailstone(n, total):
    if n == 1
        return total
    tail_hailstone(n//2, total+1)
return tail_hailstone(n, 1)
```

Tree Recursion

fib(n)

fib(n-1) + fib(n-2)



- multiple base cases

Q5: Count Stair Ways

Imagine that you want to go up a flight of stairs that has `n` steps, where `n` is a positive integer. You can either take 1 or 2 steps each time. In this question, you'll write a function `count_stair_ways` that solves this problem. Before you code your approach, consider these questions.

How many different ways can you go up this flight of stairs? → what we're trying to find out

Your Answer: "How many options do I have for my next move?"

Log in to save your work! (https://disc.cs61a.org/oauth/login)

What's the base case for this question? What is the simplest input?

Your Answer:

Log in to save your work! (https://disc.cs61a.org/oauth/login)

What do `count_stair_ways(n - 1)` and `count_stair_ways(n - 2)` represent?

Your Answer:

Log in to save your work! (https://disc.cs61a.org/oauth/login)

Fill in the code for `count_stair_ways`:

```
def count_stair_ways(n):
    """Returns the number of ways to climb up a flight of
    n stairs, moving either 1 step or 2 steps at a time.
    >>> count_stair_ways(4)
    5
    """
    "*** YOUR CODE HERE ***"
```

Q6: (Tutorial) Count K

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):
    """ Counts the number of paths up a flight of n stairs
    when taking up to and including k steps at a time.
    >>> 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
    """
    "*** YOUR CODE HERE ***"
```