Discussion 4: February 18, 2025

Recursion

Many students find this topic challenging. Everything gets easier with practice. Please help each other learn.

Q1: Swipe

Implement swipe, which prints the digits of argument n, one per line, first backward then forward. The left-most digit is printed only once. Do not use while or for or str. (Use recursion, of course!)

```
def swipe(n):
    """Print the digits of n, one per line, first backward then forward.
    >>> swipe(2837)
    7
    3
    8
    2
    8
    3
    7
    0.00
    if n < 10:
        print(n)
    else:
        print(n % 10)
        swipe(n // 10)
        print(n % 10)
```

First print the first line of the output, then make a recursive call, then print the last line of the output.

Q2: Skip Factorial

Define the base case for the skip_factorial function, which returns the product of every other positive integer, starting with n.

```
def skip_factorial(n):
    """Return the product of positive integers n * (n - 2) * (n - 4) * ...

>>> skip_factorial(5) # 5 * 3 * 1
    15
    >>> skip_factorial(8) # 8 * 6 * 4 * 2
    384
    """

if n <= 2:
    return n
else:
    return n * skip_factorial(n - 2)</pre>
```

If n is even, then the base case will be 2. If n is odd, then the base case will be 1. Try to write a condition that handles both possibilities.

Q3: 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. Complete this recursive version of hailstone that prints out the values of the sequence and returns the number of steps.

```
def hailstone(n):
    """Print out the hailstone sequence starting at n,
    and return the number of elements in the sequence.
    >>> a = hailstone(10)
    10
    5
    16
    8
    4
    2
    1
    >>> a
    7
    >>> b = hailstone(1)
    >>> b
    1
    0.00
    print(n)
    if n % 2 == 0:
        return even(n)
    else:
        return odd(n)
def even(n):
    return 1 + hailstone(n // 2)
def odd(n):
    if n == 1:
        return 1
    else:
        return 1 + hailstone(3 * n + 1)
```

An even number is never a base case, so even always makes a recursive call to hailstone and returns one more than the length of the rest of the hailstone sequence.

An odd number might be 1 (the base case) or greater than one (the recursive case). Only the recursive case should call hailstone.

Document the Occasion

Please all fill out the attendance form (one submission per person per week).

Extra Questions

The questions below are optional but recommended if you would like some extra practice.

Q4: Is Prime

Implement is_prime that takes an integer n greater than 1. It returns True if n is a prime number and False otherwise. Try following the approach below, but implement it recursively without using a while (or for) statement.

```
def is_prime(n):
    assert n > 1
    i = 2
    while i < n:
        if n % i == 0:
            return False
        i = i + 1
    return True</pre>
```

You will need to define another "helper" function (a function that exists just to help implement this one). Does it matter whether you define it within is_prime or as a separate function in the global frame? Try to define it to take as few arguments as possible.

```
def is_prime(n):
    """Returns True if n is a prime number and False otherwise.
    >>> is prime(2)
    True
    >>> is_prime(16)
    False
    >>> is_prime(521)
    True
    0.00
    def check_all(i):
        "Check whether no number from i up to n evenly divides n."
        if i == n:
                        # could be replaced with i > (n ** 0.5)
            return True
        elif n % i == 0:
            return False
        return check_all(i + 1)
    return check_all(2)
```

Define an inner function that checks whether some integer between i and n evenly divides n. Then you can call it starting with i=2:

Q5: Function Repeater

Define a function $make_fn_repeater$ which takes in a one-argument function f and an integer x. It should return another function which takes in one argument, another integer. This function returns the result of applying f to x this number of times.

Make sure to use recursion in your solution.

```
def make_func_repeater(f, x):
    """
    >>> incr_1 = make_func_repeater(lambda x: x + 1, 1)
    >>> incr_1(2) #same as f(f(x))
    3
    >>> incr_1(5)
    6
    """
    def repeat(i):
        if i == 0:
            return x
        else:
            return f(repeat(i - 1))
    return repeat
```