CS 61A
Fall 2023

Recursion

Discussion 3: September 13, 2023

Getting Started

VERY IMPORTANT: In this discussion, don't use a Python interpreter to run code until you are sure you are correct. Figure things out and check your work by *thinking* about what your code will do. Not sure? Talk to your group!

To get help from a TA, send a message to the `discuss-queue` channel with the @discuss tag and your discussion group number.

What everyone will need: * A way to view this worksheet (on your phone is fine) * A way to take notes (paper or a device)

What the group will need: * Someone to join Discord.

Suggestion: After Midterm 1, some students are looking for more effective ways to study. One great option is to meet up with your discussion group outside of class to review practice problems together. Now is a great time to schedule a time and place for some extra group practice of old Midterm 1 questions. This is optional and not everyone needs to come, but if there are Midterm 1 topics that haven't totally clicked yet, next week is a perfect time to review them.

Everything in this course builds on prior topics, and it's going to be hard to keep up if you don't have a solid understanding of the foundational topics from Midterm 1.

Remember, it's ok if someone hasn't learned everything yet and needs more time to master the course material. The whole point of the course is for students to learn things they don't already know. Please support each other in the process.

Recursion

Ok, time to discuss problems! Remember to work together. Everyone in the group should understand a solution before the group moves on.

Q1: Recursion Mystery

Given the function below, what will be the output of `f(3256)`?

```
def f(n):
    if n < 10:
        print(n)
    else:
        print(n % 10)
        f(n // 10)
        print(n % 10)
```

6 5 2 3 2 5 6

When you've all agreed on an answer, write your response in your group's channel's text chat so that the course staff can provide feedback.

Q2: Skip Factorial

Define the base case for the `skip_factorial` function.

```
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)
```

```
def skip_factorial(n):
    if n < 3:
        return n
    else:
        return n * skip_factorial(n - 2)
```

Q3: 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
```

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
    """
    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)
```

Finally, write a docstring for the helper function that describes what it does. Don't just write, "it helps implement `is_prime`." Instead, describe its behavior. When you're done, paste the text of that docstring in your group's channel's text chat.

Note: This worksheet is a problem bank—most TAs will not cover all the problems in discussion section.

Q4: 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)
    1
    >>> b
    1
    """
    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)
```

Once your group has converged on a solution, it's time to practice your ability to describe your own code. Pick a presenter, then send a message to the `discuss-queue` channel with the @discuss tag, your discussion group number, and the message "Hailing all course staff!" and a member of the course staff will join your voice channel to hear your description.

You'll need your whole discussion group for the next question. If anybody isn't caught up with you, spend some time helping them out.

Q5: Sevens

The Game of Sevens: Players in a circle count up from 1 in the clockwise direction. (The starting player says 1, the player to their left says 2, etc.) If a number is divisible by 7 or contains a 7 (or both), switch directions. Numbers must be said on the beat at 60 beats per minute. If someone says a number when it's not their turn or someone misses the beat on their turn, the game ends.

For example, 5 people would count to 20 like this:

```
Player 1 says 1
Player 2 says 2
Player 3 says 3
Player 4 says 4
Player 5 says 5
Player 1 says 6  # All the way around the circle
Player 2 says 7  # Switch to counterclockwise
Player 1 says 8
Player 5 says 9  # Back around the circle counterclockwise
Player 4 says 10
Player 3 says 11
Player 2 says 12
Player 1 says 13
Player 5 says 14 # Switch back to clockwise
Player 1 says 15
Player 2 says 16
Player 3 says 17 # Switch back to counterclockwise
Player 2 says 18
Player 1 says 19
Player 5 says 20
```

Play a few games. Post the highest number your group reached without making a mistake or missing a beat in your group's channel's text chat.

Then, implement `sevens` which takes a positive integer `n` and a number of players `k`. It returns which of the `k` players says `n`. You may call `has_seven`.

Note: This worksheet is a problem bank—most TAs will not cover all the problems in discussion section.

```
def sevens(n, k):
    """Return the (clockwise) position of who says n among k players.

    >>> sevens(2, 5)
    2
    >>> sevens(6, 5)
    1
    >>> sevens(7, 5)
    2
    >>> sevens(8, 5)
    1
    >>> sevens(9, 5)
    5
    >>> sevens(18, 5)
    2
    """
    def f(i, who, direction):
        if i == n:
            return who
        if i % 7 == 0 or has_seven(i):
            direction = -direction
        who = who + direction
        if who > k:
            who = 1
        if who < 1:
            who = k
        return f(i + 1, who, direction)
    return f(1, 1, 1)

def has_seven(n):
    if n == 0:
        return False
    elif n % 10 == 7:
        return True
    else:
        return has_seven(n // 10)
```

Get help! There's a lot to keep track of in this question. If your group is unsure of how to proceed, don't just start trying things in the interpreter to see what works; ask for help in the `discuss-queue` channel. We'll also review this question in Friday's lecture, so if you don't finish, that's ok.

Document the Occasion

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

Extra Challenge

This last recursion puzzle is just for fun. Work on it as a team if you have time. If you'd like a hint, send a message to the `discuss-queue` channel with the @discuss tag, your discussion group number, and the message "Hint me!"

Q6: Karel the Robot

Karel the robot starts in the corner of an `n` by `n` square for some unknown number `n`. Karel responds to only four functions: - `move()` moves Karel one square forward if there is no wall in front of Karel and errors if there is. - `turn_left()` turns Karel 90 degrees to the left. - `front_is_blocked()` returns whether there is a wall in front of Karel. - `front_is_clear()` returns whether there is no wall in front of Karel.

Implement a `main()` function that will leave Karel stopped halfway in the middle of the bottom row. For example, if the square is 7 x 7 and Karel starts in position (1, 1), the bottom left, then Karel should end in position (1, 4) (three steps from either side on the bottom row). Karel can be facing in any direction at the end. If the bottom row length is even, Karel can stop in either position (1, `n // 2`) or (1, `n // 2 + 1`).

Important You can only write `if` or `if`/`else` statements and function calls in the body of `main()`. You may not write assignment statements, def statements, lambda expressions, or while/for statements.