IbraFSG™ 3 - Week 7; List Comprehensions

Ibrahim Chehab

UTM RGASC

February 14, 2024

Table of Contents

Welcome back to IbraFSGs™

- Welcome back to IbraFSGs™! Hello to new people and welcome back to tenured members.
- This week we will be discussing recursion.
- Recursion will haunt you for the rest of CSC148 as you delve into the world of trees, graphs, and other recursive data structures.
- It will also re-appear in CSC236 as a main character, where you will find closed-forms for recursive functions, prove correctness of recursive algorithms, and much more.

Example: This is the closed-form of the Fibonacci sequence:

$$F(n) = \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right) \tag{1}$$

A Recap of the UltraSheet™

- An UltraSheet™ is a "cheat sheet" that you compile for yourself to review course materials
- It acts like your own personalized textbook chapter
- It is a great way to review for tests and exams, and find gaps in your knowledge
- UltraSheets™ help with type 1 and 2 questions

Key Terms for the week

The following are key terms for this week which you should be able to define. It is highly reccomended that you add these to your UltraSheet™

- Stack Overflow
- Stack frame
- Base case (Exact same thing as MAT102)
- (Maximum) Recursion depth
- Recursive Call
- Divide and Conquer (Not strictly important for this week, however is important to understand for future weeks + CSC236)

Practice Problem I: DeepCopying a nested list

We know from our knowledge of the `List.copy` method that it only creates a shallow copy of the list. This means that if we have a nested list, the inner lists will be copied by reference, and not by value.
Implement a function `deep_copy` that takes a list of integers and/or lists of integers, and returns a deep copy of the list.

```
def deep_copy(lst: Union[List, int]) -> Union[List,
    int]:
  """
  Return a deep copy of lst.
  """
```

Listing 1: Method signature of the deep_copy method

Hint: Most recursive functions work on the following principle: What is the least amount of work I can do in this recursive call before giving the rest of the work to the next recursive call?

Practice Problem II: Iterative to Recursive

Convert the following iterative function to a recursive function:

```
def ibranatchi_iterative(n: int) -> int:
  if n == 0:
    return 0
  elif n == 1:
    return 1
  elif n == 2:
    return 5

  sequence = [0, 1, 5]
  for i in range(3, n + 1):
    next_value = sequence[i - 1] * 2 * sequence[i - 2]
    - 5 * sequence[i - 3]
    sequence.append(next_value)

  return sequence[-1]

def ibranatchi_recursive(n: int) -> int:
  # TODO: Implement this recursively:
```

Debrief: Space and Time Complexity

Hash the following out in your groups:

- What is the time complexity of the iterative function?
- What is the time complexity of the recursive function?
- What is the space complexity of the iterative function?
- What is the space complexity of the recursive function?

A final challenge...

The following method is a recursive method that isn't intuitive right away. It is out of the scope of CSC148, however will serve as great practice for understanding recursion.

Problem Statement: Sharon Goodwin delves into Python in her free time. She's trying to create a series of recursive functions that mutually recurse over each other to determine whether a positive integer is even or odd. Help her create these two functions.

RESTRICTIONS:

- You are **NOT** allowed to use **ANY** of Python's integer operations **EXCEPT** subtraction.
- You may **NOT** use **Modulo**.
- Each function must have **EXACTLY** one base-case.
- You **MUST** use mutual recursion.
- You **MAY NOT** use any helper methods.

A final challenge...

```
def is_even(num: int) -> bool:
    """
    Method which uses mutual recursion to determine
    whether an integer is even or odd.
    """
```

```
def is_odd(num: int) -> bool:
    """
    Method which uses mutual recursion to determine
    whether an integer is even or odd.
    """
```