IbraFSG™ 3 - Week 7; List Comprehensions

Ibrahim Chehab

UTM RGASC

February 21, 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 *list comprehensions*.
- List Comprehensions are a way to create lists in Python using very little code. They are a very powerful tool that you will use in the future (A2)

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

- **List Comprehension:** A way to create lists in Python using very little code.

...Yeah that's about it for this week. Your UltraSheets™ should include more in-depth examples and documentation for the nooks and crannies of list comprehensions this week; Nothing *too* crazy :D

A Trick to Understanding List Comprehensions

List Comprehensions may seem scary at first, but they are actually quite simple. The trick to understanding them is to relate them to MAT102: Example:

$\{x^2 : x \in \{0, 1, 2, 3, 4, 5\}\}$ $\iff$

```
[x**2 for x in range(6)]
```

Fun Fact: You can include conditionals in list comprehensions, and items to be added if the condition is not met! Example:

```
[x**2 for x in range(6) if x % 2 == 0 else x**3]
```

would return a list like this:

```
[0, 1, 4, 27, 16, 125]
```

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...

Recall from last week our challenge problem:
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. . .

Continue solving this problem today:

```
def is_even(num: int) -> bool:
    """
    Method which uses mutual recursion to determine
   whether an integer is even or odd.
    """
    # TODO: Implement this method with a recursive
   call to is_odd


def is_odd(num: int) -> bool:
    """
    Method which uses mutual recursion to determine
   whether an integer is even or odd.
    """
      # TODO: Implement this method with a recursive
   call to is_even
```