Lisp and More

1. What do the following Lisp expressions output, when evaluated?
    a. `(list (+ 1 2) '(+ 1 2))`
    b. `(cons (+ 1 2) '(3 4))`
    c. `(+ 10 (car '(1 2 3)))`
    d. `(reverse (append '(1 2) '(3 4)))`
    e. `(cdddar (1 2 3 4 5 6 7))`
2. So far, you've gained exposure to a lot of languages: C++, Shell, Python, and Lisp. Consider each of the following scenarios. Which of these four languages would you choose for each, and why? There is no correct answer; the quality of your justification is what matters most.
    a. You want to create an interactive version of the CS 97 webpage, where there are checkboxes next to each assignment listed. You can click the checkbox to denote a completed assignment and it removes the assignment from the list.
    b. You are creating a video game graphics library that needs to process a large number of vectors quickly.
    c. You need to quickly write a program that will replace and reformat lines of text.
    d. You're a startup founder and you need a prototype of your product. Performance doesn't really matter; you just need a basic prototype ASAP so that you can start testing the market à la Lean Startup Methodology.
3. Translate the following C++ program into Emacs Lisp:

```
// Return the dot product of two 2x1 vectors, a and b, represented by
// integer arrays.
int dotProduct(int[] a, int[] b) {
    int result = a[0] * b[0] + a[1] * b[1];
    return result;
}
```

Python

When tackling all these problems, make sure to write code well: follow the snake_case convention for variables and functions instead of camelCase. Adhere to PEP 8 guidelines. Write comments as necessary, and make sure your variable and function names are descriptive. While correctness is the highest priority, if you can, try to be pythonic and keep your solutions as short and simple as possible.

Basic Problems

4. Write a function called `get_max(num0, num1)` that takes in two numbers and returns the greater of the two. Don't worry about type-checking.
   For example, `get_max(8, 2)` should return 8.

5. Write a function called `sum_range(lower, upper)` that takes in a lower bound and upper bound and returns the sum of [lower bound, upper bound). If the lower bound is greater than or equal to the upper bound, return 0. Assume both arguments are always integers.
   For example, `sum_range(3, 6)` should return 12.

6. Merge sort is a divide-and-conquer algorithm. In the conquering portion, you need to merge two smaller sorted lists into one large sorted list. Write a function named `merge_lists(li0, li1)` which takes in two **sorted** lists and merges them into one larger, sorted list. The lists may have different lengths. Don't worry about type-checking.
   For example, `merge_lists([1, 3, 5, 6], [2, 7])` should return `[1, 2, 3, 5, 6, 7]`.

7. Write a class named RegularPolygon. Each instance should have number_of_sides and side_length data members, which are passed in at initialization time. It should also have a member function get_perimeter() that calculates the shape's perimeter.
   For example, the following code snippet:

```
pentagon = RegularPolygon(5, 2.7)
print("This pentagon has {0} sides, a side length of {1}, and a
perimeter of {2}.".format(pentagon.number_of_sides,
pentagon.side_length, pentagon.get_perimeter()))
```

   should output: "This pentagon has 5 sides, a side length of 2.7, and a perimeter of 13.5."

Advanced Problems

The following questions are typical of technical interviews.

8. Write a function called `is_two_summable(nums, target)` that takes in a list of numbers and a target sum. It returns `True` if there exist two numbers in the list that sum to the target, and `False` otherwise.
   For example, `is_two_summable([1, 2, 3, 4], 7)` should return `True`, because 3 and 4 sum to 7. `is_two_summable([1, 2, 3, 4], 10)` should return `False`, because there are no two numbers in the list that sum to 10.

   Bonus: What is the time complexity of your algorithm? If N is the length of `nums`, then the optimal solution runs in $O(N)$ time.

9. Write a function called `num_jewels(jewels, stones)`. `jewels` is a string that contains characters that represent jewels. Every character in jewels is guaranteed to be unique, i.e. no character in `jewels` appears more than once. `stones` is also a string where each character represents a stone. You want to find out and return how many stones are actually jewels.

For example, `num_jewels("Ab","bbJJJqPJAaBb")` should return 4. The first parameter tells us that the character “A” and “b” are jewels. In the second parameter, “A” occurs 1 time and “b” occurs 3 times. Therefore, there are 4 jewels in the string.

Bonus: What is the time complexity of your algorithm? If N is the length of the longest parameter, then the optimal solution runs in $O(N)$ time.

10. (*This question was asked during a Google interview*) You’re playing a weird, one-dimensional version of whack-a-mole. On this machine, holes are arranged in a straight line. We can represent this using a list of Boolean values. For example, `[False, True, True, False]` means that there are four holes. The first and fourth are empty, whereas the second and third currently have a mole. You also have a very wide hammer that can whack multiple holes (next to each other) at the same time.

    Write a function called `max_moles(moles, hammer_width)` where moles is a list of Booleans that represents a configuration of empty/occupied holes at a single point in time as described above, and `hammer_width` is an integer representing the number of holes you can whack at once. Return the maximum number of moles that you can hit with a single whack, given the current configuration and the hammer’s width.

    For example, `max_moles([False, True, True, True, False], 4)` should return 3, because you can hit all 3 moles in the middle at once. `max_moles([True, False, True, False, True], 2)` should return 1, because with the hammer’s width you can only ever hit 1 mole with a single whack.

    Bonus: What is the time complexity of your algorithm? If N is the length of `moles`, then the optimal solution runs in $O(N)$ time.