EXTRA MIDTERM REVIEW

COMPUTER SCIENCE MENTORS 61A

March 12 to March 14, 2018

Midterm Review

This worksheet contains some extra problems beyond the weekly worksheet that may be good practice. The topics covered include:

- Lists (non-mutation)
- List Comprehensions
- Nonlocal
- Orders of Growth

This list is by no means exhaustive, and as we are not officially affiliated with the course, we cannot guarantee that these topics will show up on the midterm either. However, these are topics that historically do show up.

1. What would Python display? Draw box-and-pointer diagrams to find out.

   (a)
   ```
   L = [1, 2, 3]
   B = L
   B
   ```

   (b)
   ```
   A = L[1:3]
   L[0] = A
   L = L + A
   B
   ```

2. Write a list comprehension that accomplishes each of the following tasks.

   (a) Square all the elements of a given list, `lst`.

   (b) Compute the dot product of two lists `lst1` and `lst2`. *Hint*: The dot product is defined as lst1$[0] \cdot$ lst2$[0] +$ lst1$[1] \cdot$ lst2$[1] + \ldots +$ lst1$[n] \cdot$ lst2$[n]$. The Python **`zip`** function may be useful here.

   (c) Return a list of lists such that `lol = [[0], [0, 1], [0, 1, 2], [0, 1, 2, 3], [0, 1, 2, 3, 4]]`.

   (d) Return the same list as above, except now excluding every instance of the number 2: `lold = [[0], [0, 1], [0, 1], [0, 1, 3], [0, 1, 3, 4]])`.

3. (a) Draw the environment diagram that results from running the code.

```
def what(a, b):
    x = a
    def ha(ha):
        nonlocal x
        x = ha * 2
        return x
    return b(ha(x), x)

what(4, lambda x, y : x)
```

(b) Write the simplest possible function that does the same thing as `what` for any input `a`, `b`.

Computer Science Mentors CS61A Spring 2018: Chris Allsman and Jennie Chen, with
Ajay Raj, Alex Yang, Annie Tang, Brandon Fong, Catherine Han, Danelle Nachum, Elaine Park, Hyun Jae Moon, Kevin Tsang, Lindsay Yang, Michelle Cheung, Ryan Moughan, Ryan Roggenkemper, Shreya Sahoo, Surya Duggirala, Thomas Zhang

4. **Fast Exponentiation:** in this problem, we will examine a real-world algorithm used to improve the speed of calculating exponents.

(a) First, express the runtime of the naive exponentiation algorithm in big-O notation.

```
def exp(b, n):
    if n == 0:
        return 1
    else:
        return b * exp(b, n - 1)
```

(b) Now, express the runtime of the fast exponentiation algorithm in big-O notation.

```
def fast_exp(b, n):
    if n == 0:
        return 1
    elif n % 2 == 0: # Assume square runs in constant time
        return square(fast_exp(b, n // 2))
    else:
        return b * fast_exp(b, n - 1)
```

(c) What about this slightly modified version of `fast_exp`?

```
def fast_exp(b, n):
    for _ in range(50 * n):
        print("Killing time")
    if n == 0:
        return 1
    elif n % 2 == 0:
        return square(fast_exp(b, n // 2))
    else:
        return b * fast_exp(b, n - 1)
```

5. **Mysterious loops:** What is the order of growth in time for the following functions? Use big-O notation.

(a)

```
def mystery(n):
    for i in range(n):
        while i % 2 != 0:
            print(i)
            i = i - 1
        print("Done")
```

(b)

```
def fun(n):
    for i in range(n):
        for j in range(n * n):
            if j == 4:
                return -1
            print("Fun!")
```