CS 61A Summer 2017 — Growth & Nonlocals

Mentoring 8: July 17, 2017

1 Nonlocal

1.1 (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`.

1.2 Draw the environment diagram that results from running the code.

```
def campa(nile):
    def ding(ding):
        nonlocal nile
        def nile(ring):
            return ding
    return nile(ding(1914)) + nile(1917)

ring = campa(lambda nile: 103)
```

1.3 The ping-pong sequence counts up starting from 1 and is always either counting up or counting down.

At element k, the direction switches if k is a multiple of 7 or contains the digit 7.

The first 30 elements of the ping-pong sequence are listed below, with direction swaps marked using brackets at the 7th, 14th, 17th, 21st, 27th, and 28th elements.

```
1 2 3 4 5 6 [7] 6 5 4 3 2 1 [0] 1 2 [3] 2 1 0 [-1] 0 1 2 3 4 [5] [4] 5 6
```

Implement `make_pingpong_tracker` which returns the next value in the pingpong sequence each time it is called.

```
def has_seven(n):
    """Returns whether a number, n, contains the digit, 7."""
    if n % 10 == 7:
        return True
    elif n < 10:
        return False
    else:
        return has_seven(n // 10)

def make_pingpong_tracker():
    """
    >>> output = []
    >>> x = make_pingpong_tracker()
    >>> for _ in range(9):
    ... output += [x()]
    >>> output
    [1, 2, 3, 4, 5, 6, 7, 6, 5]
    """

    index, current, add = 1, 0, True

    def pingpong_tracker():

        ___________________________________

        if add:

            ________________________________

        else:

            ________________________________

        if _______________________________:

            add = not add

        __________________________________

        __________________________________

    return pingpong_tracker
```

2 Linked Lists & Trees, Yet Again?

2.1 Consider the following linked list function.

```
def prepend(s, item):
    return link(item, s)
```

(a) What does this function do?

(b) Assume `s` is initially length n, how long does it take to `prepend` once? `prepend` twice? `prepend` n times?

2.2 What does this function do?

```
def append(s, item):
    if s is empty:
        return link(item)
    else:
        return link(first(s), append(rest(s), item))
```

2.3 Say we want to repeatedly insert some numbers to the end of a linked list.

```
def append_many(s, items):
    for item in items:
        append(s, item)
```

(a) Assuming `s` is initially length 1. How long will it take to complete the first insertion? The second? The nth?

(b) Give the total runtime in $\Theta(\cdot)$ notation if `s` is initially empty and `items` contains n items.

2.4 Consider the `word_finder` function below.

```
def word_finder(t, n, word):
    if root(t) == word:
        n -= 1
        if n == 0:
            return True
    for branch in branches(t):
        if word_finder(branch, n, word):
            return True
    return False
```

(a) What does this function do?

(b) If a tree has n total nodes, what is the total runtime for all searches in $\Theta(\cdot)$ notation?