Section #4: Midterm Review

Written by Juliette Woodrow, Anna Mistele, John Dalloul, and Elyse Cornwall

Solutions

This week in section, you'll work through problems that will help you study for the midterm. This handout only has a few problems that we think are great practice problems. We've also released a Midterm Prep Handout with lots of practice problems and exam logistic information.

Mid-Quarter Evaluation

This week, please take some time to fill out the Mid-Quarter Evaluation. Nick and Elyse read each and every piece of feedback so that we can adjust the course to best meet your needs. This is also a place to provide valuable feedback about section.

Tracing Problems

Tracing problems test your understanding of how code runs without actually executing that code. You must "trace" through the code and determine what it will output. Tracing problems often appear on CS exams :)

Num Operations

For each ??, write what value comes from the Python expression on the above line.

```
>>> 1 + 2 - 3 + 4
??
>>> 3 + 4 * 2
??
>>> 2 * (5 + 3)
??
```

Function Tracing

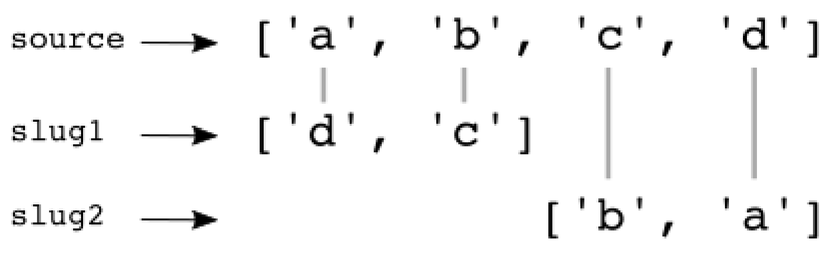
Similar to the above problem, this is a tracing problem. Rather than copying it into Pycharm and running it to see what it prints out, we are going to trace through it by hand to see how values are passed between functions. This may seem like a silly exercise, but tracing through code is a good skill to have, especially because you won't be able to run code on the midterm.

```
def b(y, x):
    z = x + y
    print("b variables: ", x, y, z)
    if z % 2 == 0: # if z is even
      return z + 1
    return z - 1

def main(): # code execution starts here
    x = 4
    y = 8
    z = x + y
    y = b(x, z)
    print("main variables: ", x, y, z)
```

Double Slug Encryption

Double-slug encryption uses one source list and *two* slug lists, all containing lowercase characters. The source list has an even length, and the two slugs are each half of its length. The **slug1** list holds the encrypted form of characters from the first half of the **source** list. The **slug2** list holds the encrypted form of characters from the second half of the **source** list. No character is in both **slug1** and **slug2**. Here is an example with a length-4 source list:



Here are some examples of encryption using our lists from above:

```
encrypt('bad') -> returns 'cda'
encrypt('abba') -> returns 'dccd'
```

Write the code for the **encrypt** function: Given the lists of lowercase characters: `source, slug1, slug2`, and a single character `ch` return the encrypted form of `ch`, or return `ch` unchanged if it is not in the `source` list. You can find the middle index of the `source` list, where the first character that would be encrypted by `slug2` will be, using:

```
midway = len(source) // 2 # in the diagram above, this is index 2
```

```python
def encrypt(source, slug1, slug2, ch):
    midway = len(source) // 2
    pass
```

Grid Problem

We have a grid representing Yellowstone park, and every square is either a person 'p', a yelling person 'y', a bear 'b', or empty None. Implement the following grid functions:

a. `is_scared(grid, x, y):` Given a grid and an in-bounds (x, y) location, return `True` if

1. There is a person or yelling person at that (x, y), and
2. There is a bear to the right of or above (x, y).

Return `False` otherwise. *Some diagrams might be helpful to visualize different Yellowstone grids!*

b. `all_run_away(grid):` When a person is scared of a bear, they begin yelling running away to the left. To implement this idea in code, you will loop over all coordinates in the grid, and make any scared people move to the left. Notice that we've provided the loops for you in the starter code below.

For each grid location (x, y), if `is_scared(grid, x, y)`:

1. Set the person stored in the grid to 'y' to signify that they've started yelling.
2. If the square to their left is in-bounds and empty, then move the person to that square, leaving their original square empty.

```python
def is_scared(grid, x, y):
    pass


def all_run_away(grid):
    for y in range(grid.height):
        for x in range(grid.width):
            pass
```