6.009 Quiz 1: Practice Quiz B

Fall 2021

Name: **Answers**

Kerberos/Athena username:

4 questions 1 hour and 50 minutes

- Please **WAIT** until we tell you to begin.
- This quiz is closed-book, but you may use one 8.5 × 11 sheet of paper (both sides) as a reference.
- You may **NOT** use any electronic devices (including computers, calculators, phones, etc.).
- If you have questions, please **come to us at the front** to ask them.
- Enter all answers in the boxes provided. Work on other pages with QR codes may be taken into account when assigning partial credit.
- If you finish the exam more than 10 minutes before the end time, please quietly bring your exam to us at the front of the room. If you finish within 10 minutes of the end time, please remain seated so as not to disturb those who are still finishing their quizzes.

1 Program Tracing

Part a.

Consider the following piece of code, designed to create small 2-d grids of a particular form, where each element in the grid is `0`, except for elements along the diagonal of the grid, which are `1`:

```
01 |   def make_grid(n):
02 |       out = []
03 |       row = [0] * n
04 |
05 |       for i in range(n):
06 |           out.append(row)
07 |
08 |       for i in range(n):
09 |           out[i][i] = 1
10 |
11 |       return out
```

Consider also the following small test:

```
grid = make_grid(5)
for row in grid:
    print(row)
```

For this small example, the intended output is:

```
[1, 0, 0, 0, 0]
[0, 1, 0, 0, 0]
[0, 0, 1, 0, 0]
[0, 0, 0, 1, 0]
[0, 0, 0, 0, 1]
```

However, this program contains an error and does not produce the intended output! Answer the questions on the facing page about the nature of this issue.

What will be the actual output produced by the small test case on the previous page, given the `make_grid` function as written?

```
[1, 1, 1, 1, 1]
[1, 1, 1, 1, 1]
[1, 1, 1, 1, 1]
[1, 1, 1, 1, 1]
[1, 1, 1, 1, 1]
```

It turns out that a one-line change to the `make_grid` function on the previous page can cause this program to produce the correct output not only for the test case given on the previous page but for all values of `n`. Which line of the input should be changed, and what should it be changed to?

Replace line number [6] with the following code:

```
out.append(row[:])
```

Part b.

Consider the following piece of code and answer the questions about it on the facing page.

```
01 |   n = 307
02 |
03 |   def f1(n):
04 |       def g1(n):
05 |           return n
06 |       return g1
07 |
08 |
09 |   def f2():
10 |       def g2(n):
11 |           return n
12 |       return g2
13 |
14 |
15 |   def f3(n):
16 |       def g3():
17 |           return n
18 |       return g3
19 |
20 |
21 |   def f4():
22 |       def g4():
23 |           return n
24 |       return g4
25 |
26 |   h1 = f1(308)
27 |   h2 = f2()
28 |   h3 = f3(309)
29 |   h4 = f4()
30 |
31 |   print(h1(310))
32 |   print(h2(311))
33 |   print(h3())
34 |   print(h4())
```

Will the program on the facing page run to completion (Yes/No)? Yes

If yes, what will be printed to the screen when this program is run?
If no, briefly describe the nature of the issue that prevents the program from running to completion.

```
310
311
309
307
```

Now imagine that we add the following on line 25 in the code:

```
n = 312
```

After this change, will the program run to completion (Yes/No)? Yes

If yes, what will be printed to the screen when this program is run?
If no, briefly describe the nature of the issue that prevents the program from running to completion.

```
310
311
309
312
```

Worksheet (intentionally blank)

2 Tennis[1]

A game of tennis is won by a player who has at least four points, while also having at least two more points than the opponent. However, the way that these scores are reported is somewhat unusual:

- Scores from zero to three are reported as "love", "fifteen", "thirty", and "forty", respectively.
- A typical score is reported with both players' scores, so an example match score is "fifteen-thirty".
- If the score is tied and each player has two or fewer points, then the second score is given as "all", for example "thirty-all".
- If the score is tied and each player has at least three points, then the entire score is given as "deuce".
- If each player has at least three points but the game is not over, the score of the game is "Advantage" for the player in the lead.

You have hired a freelance programmer to write a piece of code for you to report tennis scores in this form, given a number of points. They implemented this behavior as a function `report_score`, which behaves correctly, as demonstrated below:

```
>>> report_score(0, 0)
'love-all'
>>> report_score(1, 0)
'fifteen-love'
>>> report_score(1, 1)
'fifteen-all'
>>> report_score(2, 1)
'thirty-fifteen'
>>> report_score(3, 1)
'forty-fifteen'
>>> report_score(3, 2)
'forty-thirty'
>>> report_score(3, 3)
'deuce'
>>> report_score(4, 3)
'advantage player 1'
>>> report_score(4, 4)
'deuce'
>>> report_score(4, 5)
'advantage player 2'
>>> report_score(4, 6)
'win for player 2'
```

However, their code (on the facing page) is a mess! You are tasked with refactoring the code so that it is easier to work with in the future. On the facing page, please circle areas of the code that are problematic, and write code next to each circle to replace that piece. If you have additional high-level structural changes that you would like to make, please note those as well, in the blank space on the page (and/or on the following page).

[1]This problem is based on code by Emily Bache from `https://github.com/emilybache/Tennis-Refactoring-Kata`, available under the MIT/Expat license.

```
01 |   def report_score(player1_points, player2_points):
02 |       result = ""
03 |       temp_score=0
04 |
05 |       if (player1_points==player2_points):
06 |           tie_scores = {
07 |               0 : "love-all",
08 |               1 : "fifteen-all",
09 |               2 : "thirty-all",
10 |           }
11 |
12 |           result = "deuce"
13 |           for k in tie_scores:
14 |               if k == player1_points or k == player2_points:
15 |                   result = tie_scores[k]
16 |
17 |       elif (player1_points>=4 or player2_points>=4):
18 |           if player1_points >= player2_points + 2:
19 |               result = "win for player 1"
20 |           elif player2_points >= player1_points + 2:
21 |               result = "win for player 2"
22 |           elif player2_points == player1_points + 1:
23 |               result = "advantage player 2"
24 |           elif player1_points == player2_points + 1:
25 |               result = "advantage player 1"
26 |           elif player1_points == player2_points:
27 |               result = "deuce"
28 |
29 |       else:
30 |           for i in range(1,4):
31 |               if i==1:
32 |                   temp_score = player1_points
33 |               elif i == 2:
34 |                   result += "-"
35 |                   continue
36 |               else:
37 |                   temp_score = player2_points
38 |
39 |               if temp_score == 0:
40 |                   result += 'love'
41 |               if temp_score == 1:
42 |                   result += 'fifteen'
43 |               if temp_score == 2:
44 |                   result += 'thirty'
45 |               if temp_score == 3:
46 |                   result += 'forty'
47 |
48 |       return result
```

Additional Notes (if necessary):

There are many issues and opportunities for improvement here, some of which are listed below:

(a) On lines 13-15, there is no need to loop over all of the keys to find the given score (nor to compare both to player1's and player2's score, since we know they are equal). We could replace these lines with something like: `return tie_scores.get(player1_points, "deuce")`

(b) We have multiple places throughout the code where we map number of points to the words (0 to "love", 1 to "fifteen", etc). We can replace those with a single dictionary at the start of the file, and re-use that mapping throughout the code.

(c) The `result` variable is unnecessary; we can just return the values as soon as we know what they should be.

(d) The `temp_score` variable and the looping structure on lines 30-46 can be dramatically simplified, especially with the dictionary available to us if we implement the first suggested change above. No looping structure is necessary here; we can simply look up the word associated with player1's points and player2's points, and put a hyphen in between them.

(e) The check for the "deuce" condition on line 26 can never happen, so we can remove it.

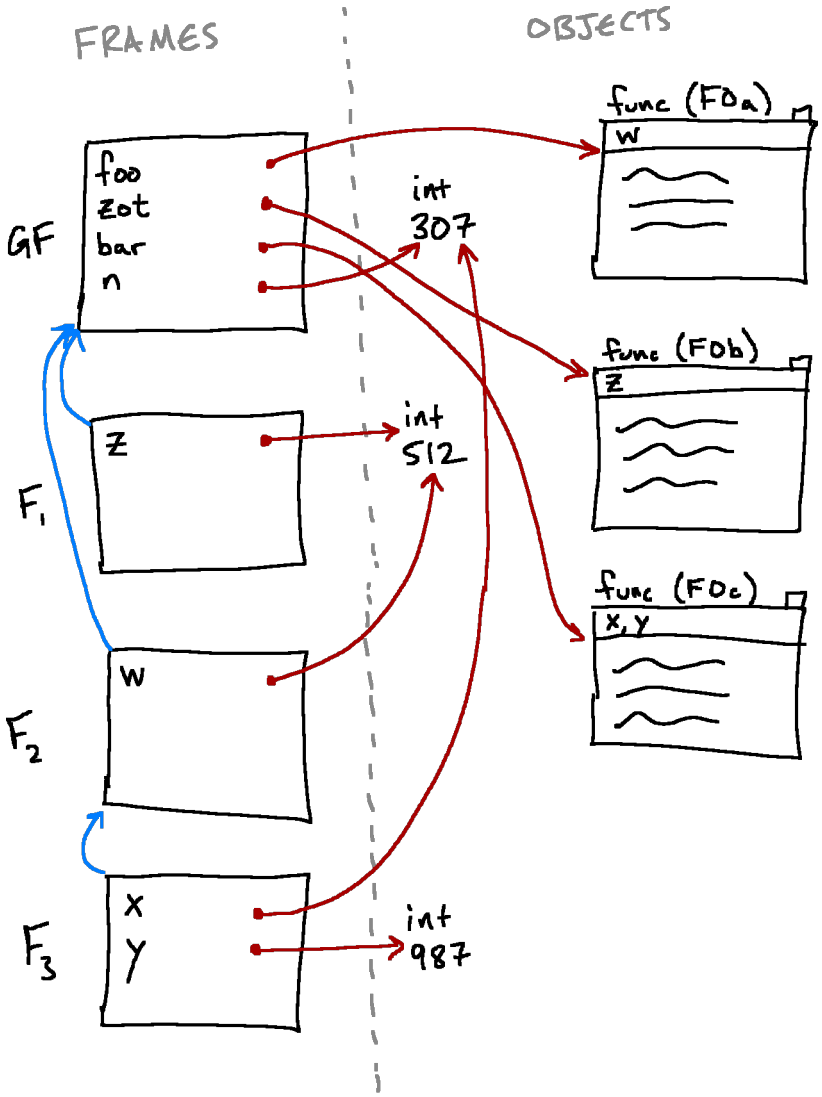
Here is one possible final result:

```
def report_score(player1_points, player2_points):
    scores = {
        0: 'love',
        1: 'fifteen',
        2: 'thirty',
        3: 'forty',
    }

    if player1_points == player2_points:
        if player1_points >= 3:
            return "deuce"
        else:
            return f"{scores[player1_points]}-all"

    elif max(player1_points, player2_points)>=4:
        leader = 1 if player1_points > player2_points else 2
        if abs(player1_points - player2_points) >= 2:
            return f"win for player {leader}"
        else:
            return f"advantage player {leader}"
    else:
        return f"{scores[player1_points]}-{scores[player2_points]}"
```

3 Environment Diagrams

Consider the following environment diagram, representing the state of a program's execution. In this diagram, notice that:

- There are exactly four variables bound in the global frame (`foo`, `zot`, `bar`, and `n`)
- There are exactly three function objects (which we have labeled with the names *FOa*, *FOb*, and *FOc*, and whose enclosing bodies and enclosing frames are intentionally left unspecified)
- There are exactly four frames: **GF** (the global frame), **F1** (which was created when calling *FOb*), **F2** (from calling *FOa*), and **F3** (from calling *FOc*).



Answer the following questions about the program that led to the environment diagram on the facing page. For any of the blanks, if there is not enough information given to answer the question (i.e., if the answer depends on information that cannot be discerned from the diagram), enter `NEI` (for "**N**ot **E**nough **I**nformation") in the box instead.

What is *FOa*'s enclosing frame? **GF**

What is *FOb*'s enclosing frame? **GF**

What is *FOc*'s enclosing frame? **F2**

The function call that led to the creation of **F1** happened while executing code in which frame? **GF**

The function call that led to the creation of **F2** happened while executing code in which frame? **F1**

The function call that led to the creation of **F3** happened while executing code in which frame? `NEI`

In the box below, enter any valid Python program that could have led to this environment diagram. Note that there are multiple possible answers.

```
n = 307

def foo(w):
    return lambda x, y: w+x+y

def zot(z):
    return foo(z)

bar = zot(512)
print(bar(n, 987))
```

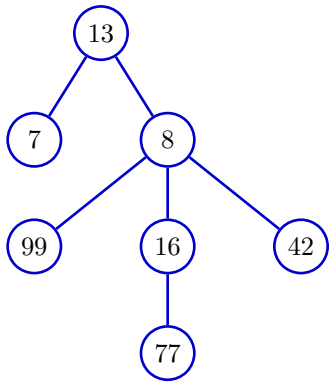
4 Tree Map

In this problem, we want to apply a given function to every value in a given tree, creating a **new tree** with the same structure as the original, but with values corresponding to the result of applying the function to every value. It should *not* mutate the input tree, nor any subelements thereof.

A node in the tree is represented as a list with the first element being the node's value and the rest of the list being the node's children (each of which is itself a tree). That is to say, our tree structure is a nested list structure.

For example, the following code (`tree1`) and picture represent the same tree.

```
tree1 = [13,
         [7],
         [8,
           [99],
           [16,
             [77]],
           [42]]]
```



For example, calling `tree_map(tree1, lambda x: x+1)` should result in the following tree (which we get by adding 1 to each element):

```
[14,
  [8],
  [9,
    [100],
    [17,
      [78]],
    [43]]]
```

Note that **although this example contains only integer values, the node values may, in general, be** *any type* **of Python object**.

Complete the definition of the `tree_map` function in the box:

```
def tree_map(tree, mapfunc):
    return [mapfunc(tree[0]), *(tree_map(tree[ix], mapfunc) for ix in range(1, len(tree)))]
```

Worksheet (intentionally blank)

Worksheet (intentionally blank)

Worksheet (intentionally blank)

Worksheet (intentionally blank)

Worksheet (intentionally blank)