National University of Singapore
School of Computing
CS1010S: Programming Methodology

# Extra Practice 6 Solutions

For Questions 1-6, what are the output(s) of the code?

## Question 1

```
a = [1, 2, 0, 4, 5, 6]

for i in range(len(a)):
    if a[i] % 2 == 0:
        a.pop(i)
    print(a)
```

**Solution:**

```
[1, 2, 0, 4, 5, 6]
[1, 0, 4, 5, 6]
[1, 0, 5, 6]
[1, 0, 5]
IndexError: list index out of range
```

Click here for full visualization.

## Question 2

```
a = []

temp = [0]*4
for i in range(3):
    a.append(temp)

a[1][1] = 99
print(a) # [[0, 99, 0, 0], [0, 99, 0, 0], [0, 99, 0, 0]]
```

Click here for full visualization.

## Question 3

```
a = [1, 2, 3]
b = (1, 2, 3, a, 4, 5)
print(b) # (1, 2, 3, [1, 2, 3], 4, 5)
a.clear()
print(b) # (1, 2, 3, [], 4, 5)
a = [1]
print(b) # (1, 2, 3, [], 4, 5)
```

Click here for full visualization.

## Question 4

```
# CS1010S AY18/19 Sem 2 Finals
a = [1, 2]
a += [a]
print(a) # [1, 2, [...]] <- this ellipsis means referring to itself
b = a.copy() # shallow copy vs deep copy?
a[2] = 0
print(a) # [1, 2, 0]
print(b) # [1, 2, [1, 2, 0]]
```

Click here for full visualization.

## Question 5

```
a = [1, 2, 3]
b = [4, a]
c = [a, b, 15, b[0], b[1]]

c[1][0] = 99
print(b) # [99, [1, 2, 3]]
print(c) # [[1, 2, 3], [99, [1, 2, 3]], 15, 4, [1, 2, 3]]

c[3] = 100
print(b) # [99, [1, 2, 3]]
print(c) # [[1, 2, 3], [99, [1, 2, 3]], 15, 100, [1, 2, 3]]

a[1] = 200
print(a) # [1, 200, 3]
print(b) # [99, [1, 200, 3]]
print(c) # [[1, 200, 3], [99, [1, 200, 3]], 15, 100, [1, 200, 3]]

b[1][2] = 300
print(a) # [1, 200, 300]
print(b) # [99, [1, 200, 300]]
print(c) # [[1, 200, 300], [99, [1, 200, 300]], 15, 100, [1, 200, 300]]

d = [4, c.copy()]
c[2] = 500
c[0][2] = 500
print(d) # [4, [[1, 200, 500], [99, [1, 200, 500]], 15, 100, [1, 200, 500]]]
```

Click here for full visualization.

## Question 6

```
# CS1010X AY18/19 Special Term I Finals
a = [4, 3, 2, 1]
b = [1, 4, 3, 2]
c = [2, 1, 4, 3]
d = [3, 2, 1, 4]
a[0], a[1], a[2] = a, c, b
b[0], b[1], b[2] = a, b, c
c[0], c[1], c[2] = c, b, d
d[0], d[1], d[2] = c, d, a
print(a[0][1][0][2][1][2][2][2][3]) # 3
```

Click here for full visualization.

## Question 7: Minesweeper!

In Minesweeper, we have some mines placed in a 2-D field (starting from row 0, column 0) and store the positions of these mines within a list. For instance, the grid below.

```
grid = [["O","O","X","O"],  # row 0
        ["X","O","X","X"],
        ["O","X","O","O"]]  # row 2

bombs = [[0, 2], [1, 0], [1, 2], [1, 3], [2, 1]]
```

For the position at row 1 column 2, there are a total of 4 bombs in its vicinity. "Vicinity" in minesweeper means the bombs in a 3-by-3 grid with respect to its position in the middle.

(a) Define a function **grid_to_bombs** that takes in a grid as shown and returns the bombs list, containing the positions of all the bombs found in the grid (marked as "X"). **Sample Test:**

```
>>> grid_to_bombs(grid) == bombs
True
```

**Solution:**

```
def grid_to_bombs(grid):
    bombs = []

    for row_num in range(len(grid)):
        for col_num in range(len(grid[0])):
            if grid[row_num][col_num] == "X":
                bombs.append([row_num, col_num])

    return bombs
```

(b) Now, define a function **count_bombs** that takes in a row and a column, as well as a grid, and returns the number of bombs in the vicinity of that row and column. **Sample Test:**

```
>>> count_bombs(1, 2, grid)
4
```

**Solution:**

```
def count_bombs(row_num, col_num, grid):
    answer = 0
    top_bound = max(0, row_num - 1)
    left_bound = max(0, col_num - 1)
    right_bound = min(len(grid[0]), col_num + 2)
    down_bound = min(len(grid), row_num + 2)

    for r in range(top_bound, down_bound):
        for c in range(left_bound, right_bound):
            answer += int(grid[r][c] == "X")

    return answer
```

***Solution compiled by Russell Saerang.***