Code Decomposition in Education

INFORMED CONSENT TO PARTICIPATE IN A RESEARCH PROJECT, “CODE DECOMPOSITION: A NEW APPROACH”.

A research project on code decomposition is being conducted by Nupur Garg, a graduate student in the Department of Computer Science and Software Engineering at Cal Poly, San Luis Obispo, under the supervision of Dr. Aaron Keen. The purpose of the study is to understand if code decomposition can be taught better in introductory courses through the use of an automated static-analysis tool.

You are being asked to take part in this study by completing the following online questionnaire. You will be asked to answer questions about your thoughts on code decomposition and how well this tool does at decomposing snippets of code. Your participation will take approximately 15-20 minutes. Please be aware that you are not required to participate in this research, you may omit any items that you prefer not to answer, and you may discontinue your participation at any time without penalty.

There are no risks anticipated with participation in this study. Your responses will be provided anonymously to protect your privacy. Potential benefits associated with the study include furthering the understanding of whether an automated tool can help improve code decomposition education.

If you have questions regarding this study or would like to be informed of the results when the study is completed, please feel free to contact Nupur Garg at ngarg@calpoly.edu or Aaron Keen at akeen@calpoly.edu. If you have concerns regarding the manner in which the study is conducted, you may contact Dr. Michael Black, Chair of the Cal Poly Institutional Review Board, at (805) 756-2894, mblack@calpoly.edu, or Dr. Dean Wendt, Dean of Research, at (805) 756-1508, dwendt@calpoly.edu.

If you agree to voluntarily participate in this research project as described, please indicate your agreement by completing and submitting the following questionnaire. Please print a copy of this consent form now for your reference, and thank you for your participation in this research.

* Required

1. *

Mark only one oval.

- ◯ Yes, I volunteer
- ◯ No, I do not volunteer *Stop filling out this form.*

Basic Demographic

This survey is aimed at individuals who have previously taken 3+ college course in computer science, have 1+ years of experience in industry, or are an instructor for computer science. This survey should take ~15 minutes.

2. **What situation best applies to your current situation** *

Mark only one oval.

- ◯ Cal Poly student taking either CPE 101 or CPE 103 (202) *Stop filling out this form.*
- ◯ Cal Poly student majoring in SE, CSC, CPE or minoring in CSC who has completed the equivalent of CPE 101, CPE 102, and CPE 103
- ◯ Non-Cal Poly student studying computer science (or related major) who has completed 3+ college courses
- ◯ Recently graduated student who studied computer science (or related major)
- ◯ Industry professional with 1+ years of experience in the industry coding fulltime
- ◯ Instructor for computer science (or related majors)

Coding Sample 1: Blur

Code decomposition (also known as functional decomposition) is the process of breaking a larger problem into smaller subproblems. In other words, functional decomposition is the process of making each function do one thing.

My tool, Earthworm, aims to provide concrete suggestions on how to improve the decomposition of Python code. This survey examines the suggestions provided by my tool for 3 code samples.

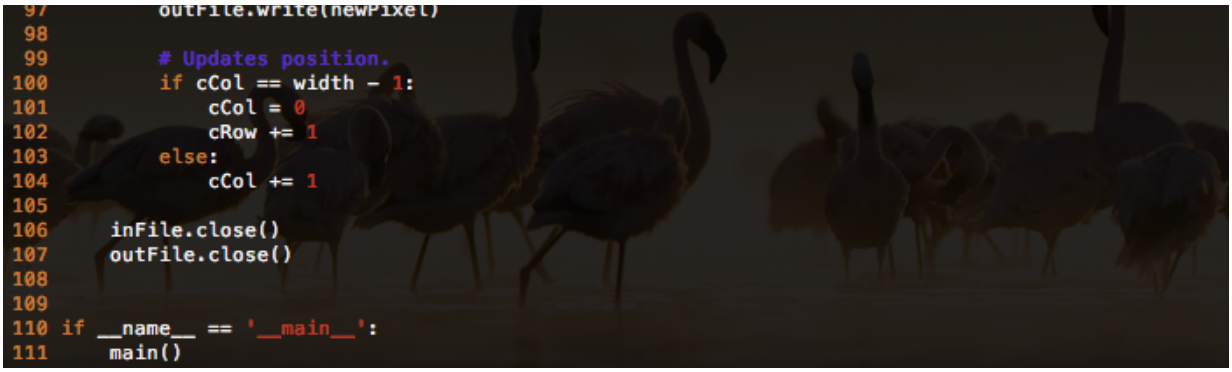
This program blurs the PPM image below. The current code sample contains one function - main() consisting of ~100 lines of code.

Input ppm image (left), output blurred ppm image (right)

Sample code

```
 4 def main():
 5     file_name = "error"
 6     outFile = open("blurred.ppm", "w")
 7
 8     # Open file.
 9     try:
10         if len(argv) == 2:
11             file_name = argv[1]
12             reach = 4
13
14             inFile = open(file_name, 'r')
15         elif len(argv) == 3:
16             file_name = argv[1]
```

```
            reach = int(argv[2])

            inFile = open(file_name, 'r')
        else:
            print("Usage: python blur.py <image> <OPTIONAL:reach>")
            exit()
    except IOError:
        print("Unable to open %s" %file_name)
        exit()

    # Print header.
    header = inFile.readline()
    w_and_h = inFile.readline().split()
    width = int(w_and_h[0])
    height = int(w_and_h[1])
    MAX_COMP_NUM = int(inFile.readline())
    outFile.write(header + str(width) + " " + str(height) + "\n" + str(MAX_COMP_NUM) + "\n")

    # Read file.
    pixels = []
    rgb = []
    for line in inFile:
        line = line.split()
        for comp in line:
            if len(rgb) != 3:
                rgb.append(comp)
            if len(rgb) == 3:
                pixels.append(rgb)
                rgb = []

    picture = []
    i = 0
    for row in range(height):
        row = []
        for col in range(width):
            row.append(pixels[i])
            i += 1
        picture.append(row)

    cCol = 0
    cRow = 0
    for pixel in pixels:
        totalR = 0
        totalB = 0
        totalG = 0
        totalP = 0

        # Calculate lower bounds of neighborhood.
        lowCol = 0
        if cCol - reach > 0:
            lowCol = cCol - reach
        lowRow = 0
        if cRow - reach > 0:
            lowRow = cRow - reach

        # Calculate upper bounds of neighborhood.
        upCol = width
        if cCol + reach < upCol:
            upCol = cCol + reach
        upRow = height
        if cRow + reach < upRow:
            upRow = cRow + reach

        # Calculate neighborhood totals.
        rows = picture[lowRow:upRow]
        for row in rows:
            cols = row[lowCol:upCol]

            for neighbor in cols:
                totalR += int(neighbor[0])
                totalG += int(neighbor[1])
                totalB += int(neighbor[2])
                totalP += 1

        red = int(totalR / totalP)
        green = int(totalG / totalP)
        blue = int(totalB / totalP)

        # Write out new pixel.
        newPixel = str(red) + " " + str(green) + " " + str(blue) + "\n"
```



Suggestions

Each suggestion below indicates how you may be able to refactor the problematic code into a new function, with the function's parameters and return value provided.

3. **Do you believe the code decomposition of the code picture above should change?** *
   Mark only one oval.
   - ◯ Yes
   - ◯ No

Suggestion #1

line 36-104 (main):
parameters: height, inFile, outFile, reach, width
reason: Multiple variables defined prior to these instructions are not used in these line numbers.

Reference the image above for the section of code recommended for decomposition.

4. **Rate the suggestion's usefulness.** *
   Usefulness can be defined as how much this code decomposition improves overall reusability, readability, and testability.
   Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

5. **How helpful is the text of the suggestion?** *
   Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

6. **If this change were made, would it be easier to write meaningful tests for each task that the program performs?** *
   In other words, would making this change make it easier to develop unit tests (tests testing each functional unit) for the program.
   Mark only one oval.

   - ◯ Yes
   - ◯ No

Suggestion #2

line 28-33 (main):
parameters: inFile, outFile
returns: height, width
reason: Removing these instructions decreases number of paths of execution*
in the function - making the code more readable and testable.

* paths of execution - more commonly referred to as control flow paths, are all of the paths that might be traversed through a program during execution. In general this suggestion appears when conditionals or loops are present.

```
27      # Print header.
28      header = inFile.readline()
29      w_and_h = inFile.readline().split()
30      width = int(w_and_h[0])
31      height = int(w_and_h[1])
32      MAX_COMP_NUM = int(inFile.readline())
33      outFile.write(header + str(width) + " " + str(height) + "\n" + str(MAX_COMP_NUM) + "\n")
```

7. **Rate the suggestion's usefulness.** *
   Usefulness can be defined as how much this code decomposition improves overall reusability, readability, and testability.
   Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

8. **How helpful is the text of the suggestion?** *
   Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

9. **If this change were made, would it be easier to write meaningful tests for each task that the program performs?** *
   In other words, would making this change make it easier to develop unit tests (tests testing each functional unit) for the program.
   Mark only one oval.

   - ◯ Yes
   - ◯ No

Suggestion #3

line 38-45 (main):
parameters: inFile, pixels, rgb
returns: pixels
reason: Removing these instructions decreases number of paths of execution in the function - making the code more readable and testable.

```
35      # Read file.
36      pixels = []
37      rgb = []
38      for line in inFile:
39          line = line.split()
40          for comp in line:
41              if len(rgb) != 3:
42                  rgb.append(comp)
43              if len(rgb) == 3:
44                  pixels.append(rgb)
45                  rgb = []
```

10. **Rate the suggestion's usefulness.** *
Usefulness can be defined as how much this code decomposition improves overall reusability, readability, and testability.
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

11. **How helpful is the text of the suggestion?** *
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

12. **If this change were made, would it be easier to write meaningful tests for each task that the program performs?** *
In other words, would making this change make it easier to develop unit tests (tests testing each functional unit) for the program.
Mark only one oval.

- ◯ Yes
- ◯ No

Suggestion #4

line 41-45 (main):
parameters: comp, pixels, rgb
returns: pixels
reason: Removing these instructions decreases number of paths of execution in the function - making the code more readable and testable.

```
41              if len(rgb) != 3:
42                  rgb.append(comp)
43              if len(rgb) == 3:
44                  pixels.append(rgb)
45                  rgb = []
```

13. **Rate the suggestion's usefulness.** *

Usefulness can be defined as how much this code decomposition improves overall reusability, readability, and testability.
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

14. **How helpful is the text of the suggestion?** *

Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

15. **If this change were made, would it be easier to write meaningful tests for each task that the program performs?** *

In other words, would making this change make it easier to develop unit tests (tests testing each functional unit) for the program.
Mark only one oval.

- ◯ Yes
- ◯ No

Suggestion #5

line 49-54 (main):
parameters: height, i, picture, pixels, width
returns: picture
reason: Removing these instructions decreases number of paths of execution in the function - making the code more readable and testable.

```
47      picture = []
48      i = 0
49      for row in range(height):
50          row = []
51          for col in range(width):
52              row.append(pixels[i])
53              i += 1
54          picture.append(row)
```

16. **Rate the suggestion's usefulness.** *

Usefulness can be defined as how much this code decomposition improves overall reusability, readability, and testability.
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

17. **How helpful is the text of the suggestion?** *
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

18. **If this change were made, would it be easier to write meaningful tests for each task that the program performs?** *
In other words, would making this change make it easier to develop unit tests (tests testing each functional unit) for the program.
Mark only one oval.

◯ Yes
◯ No

Suggestion #6

line 58-106 (main):
parameters: cCol, cRow, height, inFile, outFile, picture, pixels, reach, width
reason: Removing these instructions decreases number of paths of execution in the function - making the code more readable and testable.

```
    for pixel in pixels:
        totalR = 0
        totalB = 0
        totalG = 0
        totalP = 0

        # Calculate lower bounds of neighborhood.
        lowCol = 0
        if cCol - reach > 0:
            lowCol = cCol - reach
        lowRow = 0
        if cRow - reach > 0:
            lowRow = cRow - reach

        # Calculate upper bounds of neighborhood.
        upCol = width
        if cCol + reach < upCol:
            upCol = cCol + reach
        upRow = height
        if cRow + reach < upRow:
            upRow = cRow + reach

        # Calculate neighborhood totals.
        rows = picture[lowRow:upRow]
        for row in rows:
            cols = row[lowCol:upCol]

            for neighbor in cols:
                totalR += int(neighbor[0])
                totalG += int(neighbor[1])
                totalB += int(neighbor[2])
                totalP += 1

        red = int(totalR / totalP)
        green = int(totalG / totalP)
        blue = int(totalB / totalP)

        # Write out new pixel.
        newPixel = str(red) + " " + str(green) + " " + str(blue) + "\n"
        outFile.write(newPixel)

        # Updates position.
        if cCol == width - 1:
            cCol = 0
            cRow += 1
        else:
            cCol += 1

    inFile.close()
    outFile.close()
```

19. **Rate the suggestion's usefulness.** *

Usefulness can be defined as how much this code decomposition improves overall reusability, readability, and testability.
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

20. **How helpful is the text of the suggestion?** *
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

21. **If this change were made, would it be easier to write meaningful tests for each task that the program performs?** *
In other words, would making this change make it easier to develop unit tests (tests testing each functional unit) for the program.
Mark only one oval.

- ◯ Yes
- ◯ No

Suggestion #7

line 81-97 (main):
parameters: lowCol, lowRow, outFile, picture, totalB, totalG, totalP, totalR, upCol, upRow
reason: Removing these instructions decreases number of paths of execution
in the function - making the code more readable and testable.

```
80          # Calculate neighborhood totals.
81          rows = picture[lowRow:upRow]
82          for row in rows:
83              cols = row[lowCol:upCol]
84
85              for neighbor in cols:
86                  totalR += int(neighbor[0])
87                  totalG += int(neighbor[1])
88                  totalB += int(neighbor[2])
89                  totalP += 1
90
91          red = int(totalR / totalP)
92          green = int(totalG / totalP)
93          blue = int(totalB / totalP)
94
95          # Write out new pixel.
96          newPixel = str(red) + " " + str(green) + " " + str(blue) + "\n"
97          outFile.write(newPixel)
```

22. **Rate the suggestion's usefulness.** *
Usefulness can be defined as how much this code decomposition improves overall reusability, readability, and testability.
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

23. **How helpful is the text of the suggestion?** *
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

24. **If this change were made, would it be easier to write meaningful tests for each task that the program performs?** *
In other words, would making this change make it easier to develop unit tests (tests testing each functional unit) for the program.
Mark only one oval.

- ◯ Yes
- ◯ No

Suggestion #8

line 91-97 (main):
parameters: outFile, totalB, totalG, totalP, totalR
reason: Removing these instructions decreases number of paths of execution in the function - making the code more readable and testable.

```
91          red = int(totalR / totalP)
92          green = int(totalG / totalP)
93          blue = int(totalB / totalP)
94
95          # Write out new pixel.
96          newPixel = str(red) + " " + str(green) + " " + str(blue) + "\n"
97          outFile.write(newPixel)
```

25. **Rate the suggestion's usefulness.** *
Usefulness can be defined as how much this code decomposition improves overall reusability, readability, and testability.
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

26. **How helpful is the text of the suggestion?** *
Mark only one oval.

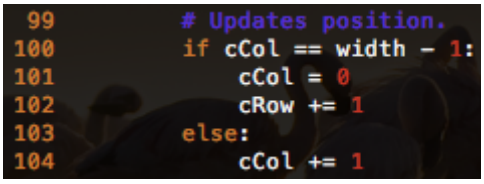
	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

27. **If this change were made, would it be easier to write meaningful tests for each task that the program performs?** *
In other words, would making this change make it easier to develop unit tests (tests testing each functional unit) for the program.
Mark only one oval.

- Yes
- No

Suggestion #9

line 100-104 (main):
parameters: cCol, cRow, width
returns: cCol, cRow
reason: Removing these instructions decreases number of paths of execution in the function - making the code more readable and testable.



```
 99          # Updates position.
100          if cCol == width - 1:
101              cCol = 0
102              cRow += 1
103          else:
104              cCol += 1
```

28. **Rate the suggestion's usefulness.** *
Usefulness can be defined as how much this code decomposition improves overall reusability, readability, and testability.
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

29. **How helpful is the text of the suggestion?** *
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

30. **If this change were made, would it be easier to write meaningful tests for each task that the program performs?** *
In other words, would making this change make it easier to develop unit tests (tests testing each functional unit) for the program.
Mark only one oval.

- Yes
- No

General Questions

Below is the revised code decomposition after making the suggestions listed above.

```
from sys import *
import math


# Prints header.
def print_header(inFile, outFile):
    header = inFile.readline()
    w_and_h = inFile.readline().split()
    width = int(w_and_h[0])
    height = int(w_and_h[1])
    MAX_COMP_NUM = int(inFile.readline())
    outFile.write(header + str(width) + " " + str(height) + "\n" + str(MAX_COMP_NUM) + "\n")
    return height, width


# Read file.
def read_file(inFile, pixels, rgb):
    pixels = []
    rgb = []
    for line in inFile:
        line = line.split()
        for comp in line:
            if len(rgb) != 3:
                rgb.append(comp)
            if len(rgb) == 3:
                pixels.append(rgb)
                rgb = []
    return pixels


# Format pixels as a list of list as laid out in the picture.
def get_formatted_pixels(height, pixels, width):
    picture = []
    i = 0
    for row in range(height):
        row = []
        for col in range(width):
            row.append(pixels[i])
            i += 1
        picture.append(row)
    return picture


# Calculate red, green, blue values.
def calculate_rgb(outFile, totalB, totalG, totalP, totalR):
    red = int(totalR / totalP)
    green = int(totalG / totalP)
    blue = int(totalB / totalP)

    # Write out new pixel.
    newPixel = str(red) + " " + str(green) + " " + str(blue) + "\n"
    outFile.write(newPixel)


# Calculate neighborhood totals.
def calculate_neighbors(lowCol, lowRow, outFile, picture,
                        totalB, totalG, totalP, totalR, upCol, upRow):
    rows = picture[lowRow:upRow]
    for row in rows:
        cols = row[lowCol:upCol]

        for neighbor in cols:
            totalR += int(neighbor[0])
            totalG += int(neighbor[1])
            totalB += int(neighbor[2])
            totalP += 1
    calculate_rgb(outFile, totalB, totalG, totalP, totalR)


# Updates position.
def update_position(cCol, cRow, width):
    if cCol == width - 1:
        cCol = 0
        cRow += 1
    else:
        cCol += 1


# Processes file.
def process_file(inFile, outFile, reach, width):
```

```
    pixels = read_file(inFile, pixels, rgb)
    picture = get_formatted_pixels(height, pixels, width)

    cCol = 0
    cRow = 0
    for pixel in pixels:
        totalR = 0
        totalB = 0
        totalG = 0
        totalP = 0

        # Calculate lower bounds of neighborhood.
        lowCol = 0
        if cCol - reach > 0:
            lowCol = cCol - reach
        lowRow = 0
        if cRow - reach > 0:
            lowRow = cRow - reach

        # Calculate upper bounds of neighborhood.
        upCol = width
        if cCol + reach < upCol:
            upCol = cCol + reach
        upRow = height
        if cRow + reach < upRow:
            upRow = cRow + reach

        calculate_neighbors(lowCol, lowRow, outFile, picture,
            totalB, totalG, totalP, totalR, upCol, upRow)

        cCol, cRow = update_position(cCol, cRow, width)


def main():
    file_name = "error"
    outFile = open("blurred.ppm", "w")

    # Open file.
    try:
        if len(argv) == 2:
            file_name = argv[1]
            reach = 4

            inFile = open(file_name, 'r')
        elif len(argv) == 3:
            file_name = argv[1]
            reach = int(argv[2])

            inFile = open(file_name, 'r')
        else:
            print("Usage: python blur.py <image> <OPTIONAL:reach>")
            exit()
    except IOError:
        print("Unable to open %s" %file_name)
        exit()

    # Processes file.
    height, width = print_header(inFile, outFile)
    process_file(inFile, outFile, reach, width)

    inFile.close()
    outFile.close()


if __name__ == '__main__':
    main()
```

31. **Do you believe this code should be decomposed further?** *

In other words, are there areas of the code that should come up as suggestions but did not?
Mark only one oval.

- ◯ Yes
- ◯ No

32. **If you answered yes to the previous question, what line numbers in the revised code would you make into a separate function.**

33. **What would you change to make the text of the suggestions more useful?**

34. **Any other thoughts/comments on this code sample?**

Coding Sample 2: 2048

This program is part of the solution for a command line version of 2048.

The solution logic is adapted from github repository yangshan/2048-python (https://github.com/yangshun/2048-python/blob/master/logic.py)

Code sample

```
# Moves the board.
def move_board(game, direction):
    has_shift = False
    has_merge = False

    if direction == 'w':
        print('... UP ...')
        game = transpose(game)
        game, has_shift = shift_left(game)
        game, has_merge = merge(game)
        game, _ = shift_left(game)
        game = transpose(game)
    elif direction == 'a':
        print('... LEFT ...')
        game, has_shift = shift_left(game)
        game, has_merge = merge(game)
        game, _ = shift_left(game)
    elif direction == 's':
        print('... RIGHT ...')
        game = reverse(game)
        game, has_shift = shift_left(game)
        game, has_merge = merge(game)
        game, _ = shift_left(game)
        game = reverse(game)
    elif direction == 'z':
        print('... DOWN ...')
        game = reverse(transpose(game))
        game, has_shift = shift_left(game)
        game, has_merge = merge(game)
        game, _ = shift_left(game)
        game = transpose(reverse(game))
    else:
        print('... INVALID MOVE ...')

    made_move = has_shift or has_merge
    return (game, made_move)


# Prints the board.
def print_board(board):
    for row in board:
        for col in row:
            print('{0} '.format(col), end=' ')
        print()
    print()


def main():
    # Initialize board.
    board = new_game(size=4)
    add_two(board)
    add_two(board)
    print_board(board)

    # Loop until game state ended.
    game_state = 0

    while game_state == 0:
        direction = input('Enter direction to move (w = UP, a = LEFT, s = RIGHT, z = DOWN): ')
        board, made_move = move_board(board, direction)
        if made_move:
            add_two(board)

            # Prints the board.
            print_board(board)

            # Set variables for next loop.
            game_state = get_game_state(board)

    # Prints the results of the game.
    if game_state == 1:
        print('YOU WON!')
    else:
        print('YOU LOST')

if __name__ == '__main__':
    main()
```

Suggestions

Each suggestion below indicates how you may be able to refactor the problematic code into a new function, with the function's parameters and return value provided.

35. **Do you believe the code decomposition of the code picture above should change?** *
Mark only one oval.

- ◯ Yes
- ◯ No

Suggestion #1

line 114-118 (move_board):
parameters: game
returns: game, has_merge, has_shift
reason: The same set of variables are referenced in all instructions in the given line numbers.

```
112     if direction == 'w':
113         print('... UP ...')
114         game = transpose(game)
115         game, has_shift = shift_left(game)
116         game, has_merge = merge(game)
117         game, _ = shift_left(game)
118         game = transpose(game)
```

36. **Rate the suggestion's usefulness.** *
Usefulness can be defined as how much this code decomposition improves overall reusability, readability, and testability.
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

37. **How helpful is the text of the suggestion?** *
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

38. **If this change were made, would it be easier to write meaningful tests for each task that the program performs?** *
In other words, would making this change make it easier to develop unit tests (tests testing each functional unit) for the program.
Mark only one oval.

- ◯ Yes
- ◯ No

Suggestion #2

line 121-123 (move_board):
parameters: game
returns: game, has_merge, has_shift
reason: The same set of variables are referenced in all instructions in the given line numbers.

```
119     elif direction == 'a':
120         print('... LEFT ...')
121         game, has_shift = shift_left(game)
122         game, has_merge = merge(game)
123         game, _ = shift_left(game)
```

39. **Rate the suggestion's usefulness.** *
Usefulness can be defined as how much this code decomposition improves overall reusability, readability, and testability.
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

40. **How helpful is the text of the suggestion?** *
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

41. **If this change were made, would it be easier to write meaningful tests for each task that the program performs?** *
In other words, would making this change make it easier to develop unit tests (tests testing each functional unit) for the program.
Mark only one oval.

- ◯ Yes
- ◯ No

Suggestion #3

line 126-130 (move_board):
parameters: game
returns: game, has_merge, has_shift
reason: The same set of variables are referenced in all instructions in the given line numbers.

```
124     elif direction == 's':
125         print('... RIGHT ...')
126         game = reverse(game)
127         game, has_shift = shift_left(game)
128         game, has_merge = merge(game)
129         game, _ = shift_left(game)
130         game = reverse(game)
```

42. **Rate the suggestion's usefulness.** *

Usefulness can be defined as how much this code decomposition improves overall reusability, readability, and testability.
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

43. **How helpful is the text of the suggestion?** *

Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

44. **If this change were made, would it be easier to write meaningful tests for each task that the program performs?** *

In other words, would making this change make it easier to develop unit tests (tests testing each functional unit) for the program.
Mark only one oval.

- ◯ Yes
- ◯ No

Suggestion #4

line 133-137 (move_board):
parameters: game
returns: game, has_merge, has_shift
reason: The same set of variables are referenced in all instructions in the given line numbers.

```
131     elif direction == 'z':
132         print('... DOWN ...')
133         game = reverse(transpose(game))
134         game, has_shift = shift_left(game)
135         game, has_merge = merge(game)
136         game, _ = shift_left(game)
137         game = transpose(reverse(game))
```

45. **Rate the suggestion's usefulness.** *

Usefulness can be defined as how much this code decomposition improves overall reusability, readability, and testability.
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

46. **How helpful is the text of the suggestion?** *
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

47. **If this change were made, would it be easier to write meaningful tests for each task that the program performs?** *
In other words, would making this change make it easier to develop unit tests (tests testing each functional unit) for the program.
Mark only one oval.

- ◯ Yes
- ◯ No

Suggestion #5

line 157-159 (main):
parameters: board
reason: The same set of variables are referenced in all instructions in the given line numbers.

```
155     # Initialize board.
156     board = new_game(size=4)
157     add_two(board)
158     add_two(board)
159     print_board(board)
```

48. **Rate the suggestion's usefulness.** *
Usefulness can be defined as how much this code decomposition improves overall reusability, readability, and testability.
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

49. **How helpful is the text of the suggestion?** *
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

50. **If this change were made, would it be easier to write meaningful tests for each task that the program performs?** *
In other words, would making this change make it easier to develop unit tests (tests testing each functional unit) for the program.
Mark only one oval.

- ◯ Yes
- ◯ No

Suggestion #6

line 168-174 (main):
parameters: board
returns: game_state
reason: The same set of variables are referenced in all instructions in the given line numbers.

```
167         if made_move:
168             add_two(board)
169
170             # Prints the board.
171             print_board(board)
172
173             # Set variables for next loop.
174             game_state = get_game_state(board)
```

51. **Rate the suggestion's usefulness.** *
Usefulness can be defined as how much this code decomposition improves overall reusability, readability, and testability.
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

52. **How helpful is the text of the suggestion?** *
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

53. **If this change were made, would it be easier to write meaningful tests for each task that the program performs?** *
In other words, would making this change make it easier to develop unit tests (tests testing each functional unit) for the program.
Mark only one oval.

- ◯ Yes
- ◯ No

Suggestion #7

line 177-180 (main):
parameters: game_state
reason: Removing these instructions decreases number of paths of execution in the function - making the code more readable and testable.

```
176     # Prints the results of the game.
177     if game_state == 1:
178         print('YOU WON!')
179     else:
180         print('YOU LOST')
```

54. **Rate the suggestion's usefulness.** *

Usefulness can be defined as how much this code decomposition improves overall reusability, readability, and testability.
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

55. **How helpful is the text of the suggestion?** *

Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

56. **If this change were made, would it be easier to write meaningful tests for each task that the program performs?** *

In other words, would making this change make it easier to develop unit tests (tests testing each functional unit) for the program.
Mark only one oval.

- ◯ Yes
- ◯ No

General Questions

Below is the revised code decomposition after making the suggestions listed above.

```
107 # Moves the board up.
108 def move_board_up(game):
109     game = transpose(game)
110     game, has_shift = shift_left(game)
111     game, has_merge = merge(game)
112     game, _ = shift_left(game)
113     game = transpose(game)
114
115
116 # Moves the board left.
117 def move_board_left(game):
118     game, has_shift = shift_left(game)
119     game, has_merge = merge(game)
120     game, _ = shift_left(game)
121
122
123 # Moves the board right.
124 def move_board_right(game):
125     game = reverse(game)
126     game, has_shift = shift_left(game)
127     game, has_merge = merge(game)
128     game, _ = shift_left(game)
129     game = reverse(game)
130
131
132 # Moves the board down.
133 def move_board_down(game):
134     game = reverse(transpose(game))
135     game, has_shift = shift_left(game)
136     game, has_merge = merge(game)
137     game, _ = shift_left(game)
138     game = transpose(reverse(game))
139
140
141 # Moves the board.
```

```
# Moves the board.
def move_board(game, direction):
    has_shift = False
    has_merge = False

    if direction == 'w':
        print('... UP ...')
        move_board_up()
    elif direction == 'a':
        print('... LEFT ...')
        move_board_left()
    elif direction == 's':
        print('... RIGHT ...')
        move_board_right()
    elif direction == 'z':
        print('... DOWN ...')
        move_board_down()
    else:
        print('... INVALID MOVE ...')

    made_move = has_shift or has_merge
    return (game, made_move)


# Prints the board.
def print_board(board):
    for row in board:
        for col in row:
            print('{0} '.format(col), end=' ')
        print()
    print()


# Initializes board.
def init_board(board):
    add_two(board)
    add_two(board)
    print_board(board)


# Makes move on the board.
def make_move(board):
    add_two(board)

    # Prints the board.
    print_board(board)

    # Set variables for next loop.
    return get_game_state(board)


# Prints result of the game.
def print_result(game_state):
    if game_state == 1:
        print('YOU WON!')
    else:
        print('YOU LOST')


def main():
    # Initialize board.
    board = new_game(size=4)
    init_board(board)

    # Loop until game state ended.
    game_state = 0

    while game_state == 0:
        direction = input('Enter direction to move (w = UP, a = LEFT, s = RIGHT, z = DOWN): ')
        board, made_move = move_board(board, direction)
        if made_move:
            game_state = make_move(board)
    print_result(game_state)

if __name__ == '__main__':
    main()
```

57. **Do you believe this code should be decomposed further?** *

In other words, are there areas of the code that should come up as suggestions but did not?
Mark only one oval.

- Yes
- No

58. **If you answered yes to the previous question, what line numbers in the revised code would you make into a separate function.**

59. **What would you change to make the text of the suggestions more useful?**

60. **Any other thoughts/comments on this code sample?**

Coding Sample 3: Crossword Puzzle

This program is a crossword puzzle solver. The two functions provided search for words in the rows (forwards or backwards) and for words in the columns (up or down).

```
11 def check_rows(puzzle, word):
12     newword = []
13     j = len(word)
14     while j > 0:
15         newword.append(word[j-1])
16         j -= 1
17     backWord = ''.join(newword)
18     for row in range(len(puzzle)):
19         if word in puzzle[row]:
20             length = 0
21             col = 0
22             for char in puzzle[row]:
23                 letter = word[length]
24                 if char != letter:
25                     length = 0
26                     letter = word[length]
27                 if char == letter:
28                         length += 1
29                         if len(word) == length:
30                             col -= (length-1)
```

```
                    place = ['(FORWARD)', row, col]
                    return place
            col += 1
      if backWord in puzzle[row]:
         length = 0
         col = 0
         for char in puzzle[row]:
            letter = backWord[length]
            if char != letter:
               length = 0
               letter = backWord[length]
            if char == letter:
               length += 1
               if len(backWord) == length:
                  place = ['(BACKWARD)', row, col]
                  return place
            col += 1

def check_cols(puzzle, word):
   newword = []
   j = len(word)
   while j > 0:
      newword.append(word[j-1])
      j -= 1
   backWord = ''.join(newword)
   newpuzzle = []
   for col in range(len(puzzle)):
      new = []
      for row in range(len(puzzle)):
         new.append(puzzle[row][col])
      newpuzz = ''.join(new)
      newpuzzle.append(newpuzz)
   for col in range(len(newpuzzle)):
      if word in newpuzzle[col]:
         length = 0
         row = 0
         for char in newpuzzle[col]:
            letter = word[length]
            if char != letter:
               length = 0
               letter = word[length]
            if char == letter:
               length += 1
               if len(word) == length:
                  row -= (length-1)
                  place = ['(DOWN)', row, col]
                  return place
            row += 1
      if backWord in newpuzzle[col]:
         length = 0
         row = 0
         for char in newpuzzle[col]:
            letter = backWord[length]
            if char != letter:
               length = 0
               letter = backWord[length]
            if char == letter:
               length += 1
               if len(backWord) == length:
                  place = ['(UP)', row, col]
                  return place
            row += 1
```

Suggestions

Each suggestion below indicates how you may be able to refactor the problematic code into a new function, with the function's parameters and return value provided.

61. **Do you believe the code decomposition of the code picture above should change?** *
Mark only one oval.

- ◯ Yes
- ◯ No

Suggestion #1

line 23-26 (check_rows):
parameters: char, length, word
returns: length, letter
reason: Multiple variables defined prior to these instructions are not used in these line numbers.

```
23              letter = word[length]
24              if char != letter:
25                 length = 0
26                 letter = word[length]
```

62. **Rate the suggestion's usefulness.** *
Usefulness can be defined as how much this code decomposition improves overall reusability, readability, and testability.
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

63. **How helpful is the text of the suggestion?** *
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

64. **If this change were made, would it be easier to write meaningful tests for each task that the program performs?** *
In other words, would making this change make it easier to develop unit tests (tests testing each functional unit) for the program.
Mark only one oval.

- ◯ Yes
- ◯ No

Suggestion #2

line 27-32 (check_rows):
parameters: char, col, length, letter, row, word
returns: col, length, place
reason: Removing these instructions decreases number of paths of execution in the function - making the code more readable and testable.

```
27              if char == letter:
28                    length += 1
29                    if len(word) == length:
30                       col -= (length-1)
31                       place = ['(FORWARD)', row, col]
32                       return place
```

65. **Rate the suggestion's usefulness.** *

Usefulness can be defined as how much this code decomposition improves overall reusability, readability, and testability.
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

66. **How helpful is the text of the suggestion?** *

Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

67. **If this change were made, would it be easier to write meaningful tests for each task that the program performs?** *

In other words, would making this change make it easier to develop unit tests (tests testing each functional unit) for the program.
Mark only one oval.

- ◯ Yes
- ◯ No

Suggestion #3

line 37-47 (check_rows):
parameters: backWord, col, length, puzzle, row
returns: letter, place
reason: Removing these instructions decreases number of paths of execution in the function - making the code more readable and testable.

```
37          for char in puzzle[row]:
38             letter = backWord[length]
39             if char != letter:
40                length = 0
41                letter = backWord[length]
42             if char == letter:
43                length += 1
44                if len(backWord) == length:
45                   place = ['(BACKWARD)', row, col]
46                   return place
47             col += 1
```

68. **Rate the suggestion's usefulness.** *

Usefulness can be defined as how much this code decomposition improves overall reusability, readability, and testability.
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

69. **How helpful is the text of the suggestion?** *

Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

70. **If this change were made, would it be easier to write meaningful tests for each task that the program performs?** *

In other words, would making this change make it easier to develop unit tests (tests testing each functional unit) for the program.
Mark only one oval.

- ◯ Yes
- ◯ No

Suggestion #4

line 42-46 (check_rows):
parameters: backWord, char, col, length, letter, row
returns: length, place
reason: Removing these instructions decreases number of paths of execution in the function - making the code more readable and testable.

```
42              if char == letter:
43                  length += 1
44                  if len(backWord) == length:
45                      place = ['(BACKWARD)', row, col]
46                      return place
```

71. **Rate the suggestion's usefulness.** *

Usefulness can be defined as how much this code decomposition improves overall reusability, readability, and testability.
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

72. **How helpful is the text of the suggestion?** *
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

73. **If this change were made, would it be easier to write meaningful tests for each task that the program performs?** *
In other words, would making this change make it easier to develop unit tests (tests testing each functional unit) for the program.
Mark only one oval.

◯ Yes
◯ No

Suggestion #5

line 63-92 (check_cols):
parameters: backWord, newpuzzle, word
returns: place
reason: Removing these instructions decreases number of paths of execution in the function - making the code more readable and testable.

```
63    for col in range(len(newpuzzle)):
64       if word in newpuzzle[col]:
65          length = 0
66          row = 0
67          for char in newpuzzle[col]:
68             letter = word[length]
69             if char != letter:
70                length = 0
71                letter = word[length]
72             if char == letter:
73                length += 1
74                if len(word) == length:
75                   row -= (length-1)
76                   place = ['(DOWN)', row, col]
77                   return place
78             row += 1
79       if backWord in newpuzzle[col]:
80          length = 0
81          row = 0
82          for char in newpuzzle[col]:
83             letter = backWord[length]
84             if char != letter:
85                length = 0
86                letter = backWord[length]
87             if char == letter:
88                length += 1
89                if len(backWord) == length:
90                   place = ['(UP)', row, col]
91                   return place
92             row += 1
```

74. **Rate the suggestion's usefulness.** *
Usefulness can be defined as how much this code decomposition improves overall reusability, readability, and testability.
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

75. **How helpful is the text of the suggestion?** *
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

76. **If this change were made, would it be easier to write meaningful tests for each task that the program performs?** *
In other words, would making this change make it easier to develop unit tests (tests testing each functional unit) for the program.
Mark only one oval.

- ◯ Yes
- ◯ No

Suggestion #6

line 68-71 (check_cols):
parameters: char, length, word
returns: length, letter
reason: Multiple variables defined prior to these instructions are not used in these line numbers.

```
68              letter = word[length]
69              if char != letter:
70                 length = 0
71                 letter = word[length]
```

77. **Rate the suggestion's usefulness.** *
Usefulness can be defined as how much this code decomposition improves overall reusability, readability, and testability.
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

78. **How helpful is the text of the suggestion?** *
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

79. **If this change were made, would it be easier to write meaningful tests for each task that the program performs?** *

In other words, would making this change make it easier to develop unit tests (tests testing each functional unit) for the program.
Mark only one oval.

- ◯ Yes
- ◯ No

Suggestion #7

line 72-77 (check_cols):
parameters: char, col, length, letter, row, word
returns: length, place, row
reason: Removing these instructions decreases number of paths of execution in the function - making the code more readable and testable.

```
72              if char == letter:
73                  length += 1
74                  if len(word) == length:
75                      row -= (length-1)
76                      place = ['(DOWN)', row, col]
77                      return place
```

80. **Rate the suggestion's usefulness.** *

Usefulness can be defined as how much this code decomposition improves overall reusability, readability, and testability.
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

81. **How helpful is the text of the suggestion?** *

Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

82. **If this change were made, would it be easier to write meaningful tests for each task that the program performs?** *

In other words, would making this change make it easier to develop unit tests (tests testing each functional unit) for the program.
Mark only one oval.

- ◯ Yes
- ◯ No

Suggestion #8

line 79-92 (check_cols):
parameters: backWord, col, newpuzzle

returns: place
reason: Removing these instructions decreases number of paths of execution in the function - making the code more readable and testable.

```
79        if backWord in newpuzzle[col]:
80           length = 0
81           row = 0
82           for char in newpuzzle[col]:
83              letter = backWord[length]
84              if char != letter:
85                 length = 0
86                 letter = backWord[length]
87              if char == letter:
88                 length += 1
89                 if len(backWord) == length:
90                    place = ['(UP)', row, col]
91                    return place
92              row += 1
```

83. **Rate the suggestion's usefulness.** *
Usefulness can be defined as how much this code decomposition improves overall reusability, readability, and testability.
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

84. **How helpful is the text of the suggestion?** *
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

85. **If this change were made, would it be easier to write meaningful tests for each task that the program performs?** *
In other words, would making this change make it easier to develop unit tests (tests testing each functional unit) for the program.
Mark only one oval.

- ◯ Yes
- ◯ No

Suggestion #9

line 83-86 (check_cols):
parameters: backWord, char, length
returns: length, letter
reason: Multiple variables defined prior to these instructions are not used in these line numbers.

```
83              letter = backWord[length]
84              if char != letter:
85                 length = 0
86                 letter = backWord[length]
```

86. **Rate the suggestion's usefulness.** *

Usefulness can be defined as how much this code decomposition improves overall reusability, readability, and testability.
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

87. **How helpful is the text of the suggestion?** *

Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

88. **If this change were made, would it be easier to write meaningful tests for each task that the program performs?** *

In other words, would making this change make it easier to develop unit tests (tests testing each functional unit) for the program.
Mark only one oval.

- ◯ Yes
- ◯ No

Suggestion #10

line 87-92 (check_cols):
parameters: backWord, char, col, length, letter, row
returns: length, place
reason: Removing these instructions decreases number of paths of execution in the function - making the code more readable and testable.

```
87              if char == letter:
88                  length += 1
89                  if len(backWord) == length:
90                      place = ['(UP)', row, col]
91                      return place
92              row += 1
```

89. **Rate the suggestion's usefulness.** *

Usefulness can be defined as how much this code decomposition improves overall reusability, readability, and testability.
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

90. **How helpful is the text of the suggestion?** *
Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

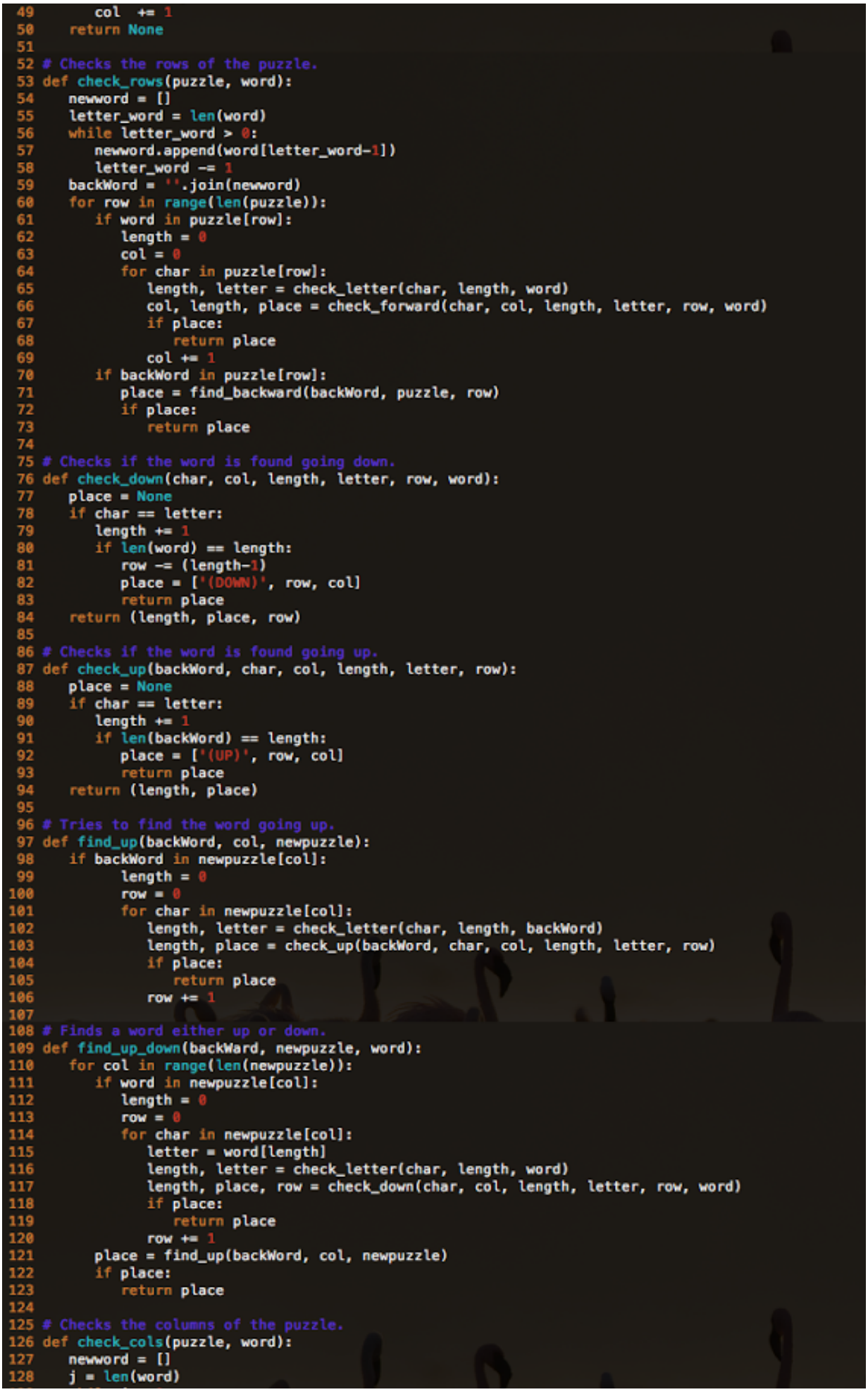
91. **If this change were made, would it be easier to write meaningful tests for each task that the program performs?** *
In other words, would making this change make it easier to develop unit tests (tests testing each functional unit) for the program.
Mark only one oval.

- ◯ Yes
- ◯ No

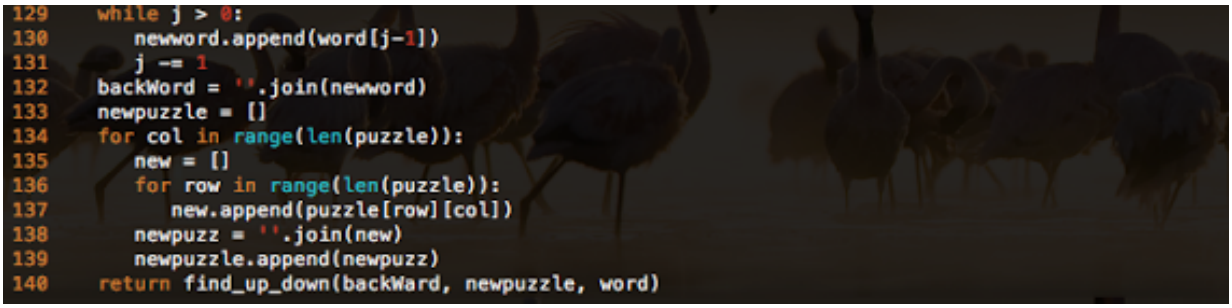
General Questions

Below is the revised code decomposition after making the suggestions listed above.

```
def make_puzzle(Puzzle):
    puzzle = []
    for i in range(0,10):
        puzzle.append(Puzzle[i*10:i*10+10])
    return puzzle

def make_words(Words):
    return Words.split()

# Checks if the given letter is equal.
def check_letter(char, length, word):
    letter = word[length]
    if char != letter:
        length = 0
        letter = word[length]
    return (length, letter)

# Checks if the word is found going forward.
def check_forward(char, col, length, letter, row, word):
    place = None
    if char == letter:
        length += 1
        if len(word) == length:
            col -= (length-1)
            place = ['(FORWARD)', row, col]
    return (col, length, place)

# Checks if the word is found going backward.
def check_backword(backWord, char, col, length, letter, row):
    place = None
    if char == letter:
        length += 1
        if len(backWord) == length:
            place = ['(BACKWARD)', row, col]
    return (length, place)

# Tries to find the word going backwards.
def find_backward(backWord, puzzle, row):
    length = 0
    col = 0
    for char in puzzle[row]:
        letter = backWord[length]
        if char != letter:
            length = 0
            letter = backWord[length]
        length, place = check_backword(backWord, char, col, length, letter, row)
        if place:
            return place
```



```
        col  += 1
    return None

# Checks the rows of the puzzle.
def check_rows(puzzle, word):
    newword = []
    letter_word = len(word)
    while letter_word > 0:
        newword.append(word[letter_word-1])
        letter_word -= 1
    backWord = ''.join(newword)
    for row in range(len(puzzle)):
        if word in puzzle[row]:
            length = 0
            col = 0
            for char in puzzle[row]:
                length, letter = check_letter(char, length, word)
                col, length, place = check_forward(char, col, length, letter, row, word)
                if place:
                    return place
                col += 1
        if backWord in puzzle[row]:
            place = find_backward(backWord, puzzle, row)
            if place:
                return place

# Checks if the word is found going down.
def check_down(char, col, length, letter, row, word):
    place = None
    if char == letter:
        length += 1
        if len(word) == length:
            row -= (length-1)
            place = ['(DOWN)', row, col]
            return place
    return (length, place, row)

# Checks if the word is found going up.
def check_up(backWord, char, col, length, letter, row):
    place = None
    if char == letter:
        length += 1
        if len(backWord) == length:
            place = ['(UP)', row, col]
            return place
    return (length, place)

# Tries to find the word going up.
def find_up(backWord, col, newpuzzle):
    if backWord in newpuzzle[col]:
            length = 0
            row = 0
            for char in newpuzzle[col]:
                length, letter = check_letter(char, length, backWord)
                length, place = check_up(backWord, char, col, length, letter, row)
                if place:
                    return place
                row += 1

# Finds a word either up or down.
def find_up_down(backWard, newpuzzle, word):
    for col in range(len(newpuzzle)):
        if word in newpuzzle[col]:
            length = 0
            row = 0
            for char in newpuzzle[col]:
                letter = word[length]
                length, letter = check_letter(char, length, word)
                length, place, row = check_down(char, col, length, letter, row, word)
                if place:
                    return place
                row += 1
        place = find_up(backWord, col, newpuzzle)
        if place:
            return place

# Checks the columns of the puzzle.
def check_cols(puzzle, word):
    newword = []
    j = len(word)
```



```
    while j > 0:
        newword.append(word[j-1])
        j -= 1
    backWord = ''.join(newword)
    newpuzzle = []
    for col in range(len(puzzle)):
        new = []
        for row in range(len(puzzle)):
            new.append(puzzle[row][col])
        newpuzz = ''.join(new)
        newpuzzle.append(newpuzz)
    return find_up_down(backWard, newpuzzle, word)
```

92. **Do you believe this code should be decomposed further?** *

Mark only one oval.

- ◯ Yes
- ◯ No

93. **If you answered yes to the previous question, what line numbers in the revised code would you make into a separate function.**

94. **What would you change to make the text of the suggestions more useful?**

95. **Any other thoughts/comments on this code sample?**

General

96. **How important do you believe code decomposition is to learn in school?** *

Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

97. **How important do you believe code decomposition is to learn early in computer science education?** *

Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

98. **How useful do you believe code decomposition is as a skill for industry?** *

Mark only one oval.

	1	2	3	4	5	6	7	
Not at all	◯	◯	◯	◯	◯	◯	◯	Very useful

99. **If you studied computer science in college (or related major), how early in your curriculum was the emphasis on code decomposition?** *

Mark only one oval.

- ◯ Did not study computer science (or related major)
- ◯ Year 1
- ◯ Year 2
- ◯ Year 3+
- ◯ Not part of curriculum

Demographic Questions

100. **Do you attend or have you attended Cal Poly for computer science?** *

Select yes if you were a SE, CSC, or CPE major or CSC minor.
Mark only one oval.

- ◯ Yes
- ◯ No

101. **What universities have you taken computer science courses at in person?** *

Write N/A if you have not taken any in person computer science courses at any university.

_____

102. **What universities have you taken computer science courses at online?** *

Write N/A if you have not taken any online computer science courses at any university.

_____

103. **How many years of experience do you have in computer science?** *

Use your best estimation for when you believe you started computer science - either formally or informally.

_____

104. **What year of college are you in?** *

Mark only one oval.

- ◯ Did not attend college
- ◯ Graduated
- ◯ Freshman
- ◯ Sophomore
- ◯ Junior
- ◯ Senior
- ◯ Masters student
- ◯ PhD student

105. **If you have an bachelors degree, what is your bachelors degree in?** *

If you are currently in college for your bachelors degree, then indicate what you intend to major in. If you minored in computer science indicate that in "Other" section.
Mark only one oval.

- ◯ Do not have an bachelors degree.
- ◯ Computer Science
- ◯ Computer Engineering (or Computer Science & Electrical Engineering dual major)
- ◯ Software Engineering
- ◯ Information Technology
- ◯ Other: _____

106. **If you have a masters degree, what is your masters degree in?** *

If you are currently in college for your masters degree, then indicate what you intend to major in.
Mark only one oval.

- ◯ Do not have a masters degree
- ◯ Computer Science
- ◯ Computer Engineering (or Computer Science & Electrical Engineering dual major)
- ◯ Software Engineering
- ◯ Information technology
- ◯ Other: _____

107. **If you have a PhD, what is your PhD in?**

Please use a very high level few word description of your PhD.

_____

108. **Have you previously tutored students in computer science?** *
Mark only one oval.

- ◯ 0-2 months of experience
- ◯ 2-6 months of experience
- ◯ 6-12 months of experience
- ◯ 12+ months of experience
- ◯ I have previously taught an introductory level course
- ◯ I have previously taught an advanced level course

109. **What is your experience with internships in computer science?** *
Mark only one oval.

- ◯ 0 internships
- ◯ 1 internship
- ◯ 2-3 internships
- ◯ 4+ internships

110. **What is your experience with full time employment in computer science?** *
Mark only one oval.

- ◯ < 6 months of experience
- ◯ 6-12 months of experience
- ◯ 1-2 years of full time experience
- ◯ 3+ years of full time experience

111. **What has been your primary title in computer science related jobs?** *
Note: If you were an intern then mark the closest title without the Intern (ex. if you were a "Software Engineering Intern" then mark "Software Engineer")
Mark only one oval.

- ◯ I have not had an industry job
- ◯ Software Engineer
- ◯ Site Reliability Engineer
- ◯ Quality Assurance Engineer
- ◯ Other: _____