15-112
Fall 2018 Midterm 1
October 11, 2018

Name:

Andrew ID:

Recitation Section:

- You may not use any books, notes, extra paper, or electronic devices during this exam. There should be nothing on your desk or chair aside from this exam and any writing implements.
- You may not ask questions about the exam except for language clarifications.
- Show your work on the exam to receive credit.
- You may use the backs of pages as scratch paper. Nothing written on the back of any pages will be graded.
- You may complete the problems in any order you'd like; you may wish to start with the free response problems, which are worth most of the credit.
- All code samples run without crashing unless we state otherwise.
- You may assume that math, string, tkinter, and copy are imported; do not import any other modules.
- Do not use these concepts: time-based animation, dictionaries, sets, and recursion.

Don't write anything in the table below.

Question	Points	Score
1	10	
2	12	
3	10	
4	18	
5	25	
6	25	
Total:	100	

1. **Short Answer**

Answer each of the following *very briefly*.

(a) (3 points) The following function does some mathematical computations. The function works fine but has many style violations. List three **different** style rules from the 15-112 style guide that this code violates, and clearly indicate where each is violated in the code.

```
def homeworkProblem3(x, y):
    if x%y==0:
        z=x // y
    if not x%y==0:
        return False
    else:
        result=0
        while z>0:
            result=result + z%10
            z       =       z//10
        return result==y
    return False
```

(b) (2 points) Soda from the vending machine costs $1.80. This function should return `True` if the sum of elements in the list (representing dollar and coin values) is exactly 1.80. Why might this assert statement fail, given the input list? Briefly and in general, how might you avoid this problem when writing code?

```
def exactChange(coins):
    dollars = 0
    for i in range(len(coins)):
        dollars += coins[i]
    return dollars == 1.80

assert(exactChange([1, 0.25, 0.25, 0.1, 0.1, 0.1]))
```

(c) (3 points) The function `findAndRemove(lst, myItem)`, which removes all instances of `myItem` from `lst`, is written three different ways below. Write whether each implementation produces a syntax, runtime, or logical error in the box below it, and **briefly** explain what causes the error. If there is no error, just write ”no error.”

```
def findAndRemove(lst, myItem):
    for i in range(len(lst)):
        if lst[i] == myItem:
            lst.pop(i)
lst = ["a", 3, "snarf", 3, 42]
findAndRemove(lst, 3)
assert(lst == ["a", "snarf", 42])
```

```
def findAndRemove(lst, myItem):
    for i in range(len(lst) - 1, -1, -1):
        if lst[i] == myItem:
            lst.pop(i)
lst = ["a", 3, [3, 4], "snarf", 3, 42]
findAndRemove(lst, "a")
assert(lst == [3, [3, 4], "snarf", 3, 42])
```

```
def findAndRemove(lst, myItem):
    while i < len(lst):
        if lst[i] == myItem:
            lst.pop(i)
        else:
            i += 1
lst = ["a", 3, [3, 4], "snarf", 3, 42]
findAndRemove(lst, "snarf")
assert(lst == ["a", 3, [3, 4], 3, 42])
```

(d) (2 points) For the following components of the word search animation, mark whether these components should be considered as part of the Model, View, or Controller.

1. The size of each cell is stored as an integer

   ◯ Model ◯ View ◯ Controller

2. Mouse click handler adds a cell to selection

   ◯ Model ◯ View ◯ Controller

3. Selected cells are drawn in yellow

   ◯ Model ◯ View ◯ Controller

4. Lines are drawn through found words

   ◯ Model ◯ View ◯ Controller

5. A 2D list specifies the letters for each cell

   ◯ Model ◯ View ◯ Controller

6. Pressing the ”s” key searches for remaining words

   ◯ Model ◯ View ◯ Controller

2. **Code Tracing**

   Indicate what each piece of code will print. Place your answer (and nothing else) in the box below each piece of code.

   (a) (6 points) CT 1

```
import copy
def ct1(a):
    b = copy.copy(a)
    c = copy.deepcopy(a)
    a[1] = "hack"
    b[0][1] = 20
    c[1][0] = 32
    c.append(b[0].pop(0))
    a.extend([a[0]])
    print("a:", a)
    print("b:", b)
    print("c:", c)

z = [ [ "cat", 4 ], [ "dog", 12 ] ]
ct1(z)
print("z:", z)
```

(b) (6 points) CT 2

```
def a(s):
    print("a:", s)
    s = s[1:-1]
    b(s)
    print(s)

def b(s):
    print("b:", s)
    s = s + s
    return s

def c(s):
    print("c:", s)
    t = ""
    for x in s:
        if x not in t:
            t += x
    return t

s = "mr_roboto"
print("out:", a(c(b(s))))
```

3. **Reasoning Over Code**

For each function, find parameter values that will make the function return True. Place your answer (and nothing else) in the box below each block of code.

(a) (5 points) ROC 1

```
def helper(num):
    for i in range(1, num):
        if num % i == 0:
            d = num % 5
            c = i
    return (c, d)

def roc1(num):
    assert(100 < num < 200)
    a, b = helper(num)

    try:
        if a // 10 == 4 and a // b == 0:
            return False
    except:
        return True
    return False
```

(b) (5 points) ROC 2

```
def roc2(s):
    for i in range(1, len(s) - 1, 2):
        s = s[:i] + chr(ord(s[i]) - 1) + s[i + 1:]

    assert("r" not in s)
    s = s.replace("b", "r")
    s = s.replace("y", "b")
    return s == "october"
```

4. (18 points) **Free Response: nthCircularPrime**

   Write the function `nthCircularPrime` that takes a non-negative int `n` and returns the nth circular prime, which is a prime number that does not contain any 0's and such that all the numbers resulting from rotating its digits are also prime. One example of a circular prime is 197; note that this number is a circular prime because 197 is prime, as is 971 (rotated left), as is 719 (rotated left again). **Note that you do not need to write `isPrime(n)`; assume it has already been provided.**

   nthCircularPrime(0) is 2, which is followed by 3, 5, 7, 11, 13, 17, 31, 37, 71, 73, 79, 97, 113, 131, 197...

   Note: you may not use strings at all in this problem.

   Hint: consider writing one or more helper functions!

Additional Space for Answer to Question 4

5. (25 points) **Free Response: Regular polygon animation**

A polygon is **regular** if every angle is equal and every side length is equal. Assuming the `run()` function is already written for you, write `init(d)`, `keyPressed(e, d)`, `mousePressed(e, d)`, and `redrawAll(c, d)` so that when the animation is first run:

1. A circle with a red outline and no fill is displayed in the center of the window. Its diameter should be equal to the window's height or width, whichever is smaller.
2. An equilateral triangle (a regular 3-sided polygon) is drawn such that its three points touch the edge of the circle. The triangle is blue and filled in.

The animation proceeds as follows:

1. If the user presses the up arrow key, the number of points of the inner polygon increases by one. Every point of the polygon should still touch the circle, and the polygon should remain regular. For example, if the up key is pressed once, the triangle becomes a square. For subsequent up key presses, it becomes a pentagon (5 sides), a hexagon (6 sides), a heptagon (7 sides) and so forth.
2. If the user presses the down arrow key and the current polygon has more than three points, the number of points in the polygon decreases by one. As before, every point of the polygon should still touch the circle, and the polygon should remain regular. If the user presses the down arrow key when only three points remain, nothing happens.
3. If the user clicks the mouse outside of the circle, the number of points in the polygon should reset to 3.

Make reasonable assumptions for anything not specified here. Do not hardcode values for data.width or data.height. We recommend that, to save time writing, you abbreviate canvas, event, and data: use c, e and d, respectively. You should also use short variable names.

Additional Space for Answer to Question 5

Additional Space for Answer to Question 5

6. (25 points) **Free Response: longestPalindrome**

   A palindrome is a word that reads the same forwards and backwards. A few example palindromes are "level", "racecar", and "wow".

   Write the function `longestPalindrome(board)` which takes a 2D list of single-character strings as a parameter and returns the longest palindrome that can be found in any row or column of the board. For example, given the following board:

```
board = [ [ "w", "n", "b", "a", "a", "j", "t", "q" ],
          [ "o", "r", "a", "c", "e", "c", "a", "r" ],
          [ "w", "x", "l", "e", "v", "e", "l", "z" ] ]
```

   we can construct four palindromes: "wow" in column 0, "aa" in row 0, "racecar" in row 1, and "level" in row 2. (Note that all single-letter characters are also palindromes by definition). In this board, "racecar" would be returned as the longest palindrome.

   You are guaranteed that the provided value will be a 2D list containing only one-character strings, and that the list will be rectangular. If there are multiple palindromes with the longest length, you may return any one of them. If there are no palindromes (i.e., if the board is empty), you should return an empty string.

   Note: you should not look for palindromes in diagonals; only search horizontally and vertically.

   Hint: consider how you might find the longest palindrome in a 1D list of letters. A very similar approach can be used here...

Additional Space for Answer to Question 6

Additional Space for Answer to Question 6