CSc 110 Sample Midterm Exam #2

1. **Expressions**
   For each expression in the left-hand column, indicate its value in the right-hand column.
   Be sure to list a constant of appropriate type (e.g., `7.0` rather than `7` for a `float`, `String`s in quotes).

Expression	Value
`1 + 2 * 3 - 4 * 5`	______________________________
`5 // 2 + 9.0 / 2.0 - 2 * 1.25`	______________________________
`29 % 2 % 5 + 34 % 3`	______________________________
`8 + 6 * -2 + 4 + (2 + 5) > 5`	______________________________
`31 // 2 / 10.0 + 10 / (5 / 2.0)`	______________________________
`(1 != 2) != (2 != 3)`	______________________________

2. **Parameter Mystery**
At the bottom of the page, write the output produced by the following program.

```
def main():
    a = "felt"
    b = "saw"
    c = "drew"
    saw = "sue"
    drew = "b"

    mystery(a, b, c)
    mystery(b, a, saw)
    mystery(drew, c, saw)
    mystery("a", saw, drew)
    mystery(a, a, "drew")

def mystery(b, a, c):
    print(c + " " + a + " the " + b)

main()
```

3. **If/Else Simulation**

For each call of the function below, write the output that is produced:

```
def mystery(x, y):
    if (x > y):
        x = x - 5
        y = y + 5
    if (x < y):
        x += 1
        y -= 1
    else:
        x = y * 2
    print(str(x) + " " + str(y))
```

Function Call	Output
`mystery(4, 7)`	____
`mystery(3, 3)`	____
`mystery(10, 5)`	____
`mystery(20, 4)`	____
`mystery(1, 1)`	____

4. **Programming**

Write a static method named `enough_time_for_lunch` that accepts four integers *hour1*, *minute1*, *hour2*, and *minute2* as parameters. Each pair of parameters represents a time on the 24-hour clock (for example, 1:36 PM would be represented as 13 and 36). The function should return `True` if the gap between the two times is long enough to eat lunch: that is, if the second time is at least 45 minutes after the first time. Otherwise the method should return `False`.

You may assume that all parameter values are valid: the hours are both between 0 and 23, and the minute parameters are between 0 and 59. You may also assume that both times represent times in the same day, e.g. the first time won't represent a time today while the second time represents a time tomorrow. Note that the second time might be earlier than the first time; in such a case, your function should return `False`.

Here are some example calls to your function and their expected return results:

Call	**Value Returned**
`enough_time_for_lunch (11, 00, 11, 59)`	`True`
`enough_time_for_lunch (12, 30, 13, 00)`	`False`
`enough_time_for_lunch (12, 30, 13, 15)`	`True`
`enough_time_for_lunch (14, 20, 17, 02)`	`True`
`enough_time_for_lunch (12, 30, 9, 30)`	`False`
`enough_time_for_lunch (12, 00, 11, 55)`	`False`

5. **Programming**

Write a function named **`cheerleader`** that accepts two integer parameters *lines* and *cheers* and prints a series of "cheer" lines at increasing levels of indentation. The first parameter represents the number of lines of output to print, and the second represents the number of "cheers" per line. For example, the call of `cheerleader(2, 4)` means that you should print 2 lines of output, each containing 4 "cheers." A "cheer" is an occurrence of the word "Go" in the output. Neighboring cheers are separated by the word "Team", so 1 cheer is printed as "Go", 2 cheers as "Go Team Go", 3 cheers are printed as "Go Team Go Team Go", and so on.

The lines you print should be displayed at increasing levels of indentation. The first line displayed should have no indentation, but each following line should be intended by 3 spaces more than the one before it. In other words, the 2nd line of output should be indented by 3 spaces, the 3rd line by 6 spaces, and so on.

You may assume that both parameters passed your function will have values of at least 1.

The following calls demonstrate your function's behavior. Your function should match this output format exactly:

```
Call     cheerleader(2, 1)    cheerleader(4, 3)              cheerleader(2, 4)
Output   Go                   Go Team Go Team Go             Go Team Go Team Go Team Go
            Go                   Go Team Go Team Go             Go Team Go Team Go Team
                                    Go Team Go Team Go
                                       Go Team Go Team Go
```

8. **Programming**
Write a function named **`random_rects`** that asks a user how many rectangles they want and then prompts them for a width and height for each rectangle. It then outputs all of the rectangles made of ascii stars and their combined area. The user will be guaranteed to input valid positive integers for each value when prompted.

The following calls demonstrate your function's behavior. Bold and underlined text is user input. Your function should match this output format exactly:

Call

```
random_rects()
```

```
random_rects()
```

Output

```
How many rectangles? 3
Width 1? 2
Height 1? 3
**
**
**
Width 2? 3
Height 2? 2
***
***
Width 3? 10
Height 3? 1
**********
Total area: 22
```

```
How many rectangles? 4
Width 1? 5
Height 1? 2
*****
*****
Width 2? 4
Height 2? 2
****
****
Width 3? 3
Height 3? 2
***
***
Width 4? 2
Height 4? 2
**
**
Total area: 28
```

Midterm Exam 1, Sample 2 Solutions

1. **Expressions**

Expression	Value
`1 + 2 * 3 - 4 * 5`	-13
`5 // 2 + 9.0 / 2.0 - 2 * 1.25`	4.0
`29 % 2 % 5 + 34 % 3`	2

`8 + 6 * -2 + 4 + (2 + 5) > 5`	`True`
`31 // 2 / 10.0 + 10 / (5 / 2.0)`	`5.5`
`(1 != 2) != (2 != 3)`	`False`

2. Parameter Mystery

```
drew saw the felt
sue felt the saw
sue drew the b
b sue the a
drew felt the felt
```

3. If/Else Simulation

Method Call	Output
`mystery(4, 7)`	`5 6`
`mystery(3, 3)`	`6 3`
`mystery(10, 5)`	`6 9`
`mystery(20, 4)`	`18 9`
`mystery(1, 1)`	`2 1`

6. Programming (five solutions shown)

```
def enough_time_for_lunch(h1, m1, h2, m2):
    if (h1 > h2):
        return False
    elif (h1 == h2):
        return m2 - m1 >= 45
    elif (h1 == h2 - 1):
        return 60 + m2 - m1 >= 45
    else:
        return True

def enough_time_for_lunch(h1, m1, h2, m2):
    if (h2 > h1 + 1):
        return True
    else if (h2 == h1 and m1 + 45 <= m2):
        return True
    else if (h2 == h1 + 1 and m1 - 15 <= m2):
        return True
    else:
       return False


def enough_time_for_lunch(h1, m1, h2, m2):
    if ((h1 == h2 and m1 + 45 <= m2) or
        (h2 == h1 + 1 and m1 - 15 <= m2) or (h1 < h2 - 1)):
        return True
    else:
        return False


def enough_time_for_lunch(h1, m1, h2, m2):
    return 60 * h1 + m1 + 45 <= 60 * h2 + m2
```

5. Programming (three solutions shown)

```
def cheerleader(lines, cheers):
    for line in range(0, lines):
        for space in range(1, line * 3 + 1):
            print(" ", end=’’)
        print("Go", end=’’)
        for cheer in range(2, cheers + 1):
            print(" Team Go", end=’’)
        print()

def cheerleader(lines, cheers):
    indent = ""
    for line in range(1, lines + 1):
        print(indent, end=’’)
        for cheer in range(1, cheers):
            print("Go Team ", end=’’)
        print("Go")
        indent += "   "


def cheerleader(lines, cheers):
    for line in range(1, lines + 1):
        for space in range(1, line):
            print("   ", end=’’)
        for cheer in range(1, cheers):
            print("Go Team ", end=’’)
        print("Go")
```

6. **Programming**

```
def random_rects():
    area = 0
    count = int(input("How many rectangles? "))
    for i in range(1, count + 1):
        width = int(input("Width " + str(i) + " ? "))
        height = int(input("Height " + str(i) + " ? "))

        for j in range(1, height + 1):
            for k in range(1, width + 1):
                print("*", end='')
            print()

        area += (width * height)

    print("Total area: " + str(area))
```