Notes about Length and Level of Difficulty

1. The sample test is longer than the actual test will be.

2. Section 1
   - Section 1 of the sample contains 2 easy section 1 questions, 1 medium question, and 2 difficult questions.
   - Section 1 of the actual midterm will contain 2 easy questions, 1 medium question, 1 difficult question.

3. Section 2
   - Section 2 of the sample contains 3 questions: 1 easy, 1 medium, 1 hard
   - Section 2 of the actual midterm will contain 2 questions: 1 easy, 1 medium-to-hard

There are two sections, each worth 50 points. One of the questions in section 2 has an extra credit component worth up to 5 points. Partial credit is possible for each question. The maximum score for the test is 100 points (or 105 including the extra credit).

Note: there will be extra credit on the actual midterm, but not on this sample test.

It is essential that you PUT YOUR NAME ON ALL TEST MATERIALS. It can be difficult to identify the author of an unsigned test and it would be better to avoid this problem.

Section 1: Below you will find several Pieces of Code followed by a question and a place to fill in an answer. Assume that there are no bugs in the code that will make the system crash, although some code may not solve a particular problem perfectly. If you find anything that you think is a bug, there is either a typo (and I should fix it for everyone) or you are mistaken.

Sample Question A

```
output = '1'+'1'
```

Question: What does output equal?
Answer: '11'

Note: Attention to detail is important. The quotes indicate that it is a string. Partial credit is possible. For example, leaving out the quotes would have lost just a little bit, but answering 2, would have resulted in an incorrect answer.
**Question 1**

def remove_middle(string):
    if len(string) < 2:
        return(string)
    else:
        return(string[0]+string[-1])

def question1():
    for item in ['01234567890','0123450','00']:
        print(remove_middle(item))

question1()

Question: What would the above function call print out?

Answer:

**Question 2**

def diagonal_asterisks(length):
    for spaces in range(length):
        print(' ' * spaces,' * ',sep='')

def question2():
    diagonal_asterisks(7)

question2()

Question: What would the above function call print out?

Answer:
Question 3:

```python
def convert_roman_character(character):
    if (character == 'I'):
        return(1)
    elif(character == 'V'):
        return(5)
    elif(character == 'X'):
        return(10)
    elif(character == 'L'):
        return(50)
    elif(character == 'C'):
        return(100)
    elif(character == 'D'):
        return(500)
    elif(character == 'M'):
        return(1000)

def roman_numeral_convert(string):
    total = 0
    current = 0
    last = 0
    ## roman numerals are either added or subtracted
    ## depending on the next character
    ## thus we must usually wait one character before
    ## determining what to do with it
    for character in string:
        current = convert_roman_character(character)
        if (last == 0):
            last = current
            ## if we are not storing anything, we store current
        elif last < current:
            total = total + current - last
            last = 0
            ## in case of subtraction, we can add the difference immediately
            ## but there is nothing to store for next time
        else:
            total = total + last
            last = current
            ## usually, we just add the last one when we see the current one
    return(total)

roman_numeral_convert('LCXM')
```

Question: What will be returned from the above function call? The program is designed to calculate the numeric equivalent of well-formed roman numerals, but is not designed to distinguish well-formed Roman Numerals from ill-formed ones. Thus, although 'LCXM' is not a legal Roman Numeral, the program will still calculate a value.

Answer:
**Question 4:** The following program prints out strings consisting of spaces, asterisks, and percent signs. Random numbers are used to determine the following: (1) how many non-space characters to print; (2) how many spaces to print; (3) whether to print an asterisk or percent; and (4) how long to pause (sleep) between lines of print. This program prints to the screen forever (until interrupted by a control-C). Think of it like a screen-saver.

```python
import random
import math
import time

def random_1_to_10():
    return(math.ceil(random.random() * 10))

def random_1_or_2():
    return(math.ceil(random.random() * 2))

def snow_flakes():
    number = 0
    spaces = 0
    character = ''
    while(True):
        number = random_1_to_10()
        spaces = random_1_to_10()
        for number in range(number):
            if random_1_or_2() == 1:
                character = '*'
            else:
                character = '%'
            print(' ' * spaces + character, end='')
        print()
        time.sleep(random_1_to_10() * .03)
```

Approximately, write out 3 consecutive lines of what would be printed assuming that: random_1_to_10 generated 5, 7, 4, 5, 2, 10, 2, 2, 7, 3, 7, 5, 7 and 9 (only use the values needed); and random_1_or_2 generates: 2, 1, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1 and 2 (only use the values needed).

Note that your answer need only be approximate and that there may be random numbers listed that you don’t end up using.

Also indicate how much time would occur between the first and second line; and how much time would occur between the second and third line.
**Question 5:** Do your best to understand how the functions listed on the next page: `turtle_zig_zag` and `turtle_scribble` work. Then answer 5a, 5b and 5c:

5a Assuming the turtle begins at the center of the screen facing right (along the X axis), (approximately) draw the line that would result from:

\[ \text{turtle}_\text{zig}_\text{zag}(3) \]

5b Assuming the turtle begins at the center of the screen facing right (along the X axis), (approximately) draw the line that would result from:

\[ \text{turtle}_\text{scribble}(3) \]

Assume that the function random_output generated these numbers in the order indicated: 193, 49, 179

5c For the same function call \( (\text{turtle}_\text{scribble}(3)) \), how many times would the color change?
import turtle
import random
import math

my_screen = turtle.Screen()
my_turtle = turtle.Turtle()

my_screen.colormode(255)
## makes it so the colors are divided up into
## 3 values from 0 to 255
## one value represents red, another green and the final one blue

def turtle_zig():
    ## This draws a leftward sixty degree angle
    my_turtle.pd()
    my_turtle.fd(20)
    my_turtle.left(60)
    my_turtle.fd(20)
    my_turtle.pu()

def turtle_zag():
    ## This draws a rightward sixty degree angle
    my_turtle.pd()
    my_turtle.fd(20)
    my_turtle.right(60)
    my_turtle.fd(20)
    my_turtle.pu()

def random_direction():
    return(math.ceil(random.random() * 360))

def random_0_to_255():
    return(math.floor(random.random() * 255))

def random_pen_color():
    red = random_0_to_255()
    green = random_0_to_255()
    blue = random_0_to_255()
    my_turtle.pencolor(red, green, blue)

def turtle_zig_zag(repetitions):
    for number in range(repetitions):
        turtle_zig()
        turtle_zag()

def turtle_scribble(repetitions):
    for number in range(repetitions):
        if (number%2) == 0:
            random_pen_color()
        turtle_zig()
        turtle_zag()
        my_turtle.left(random_direction())
Section 2: Write Functions as specified.

Question 6: Write a program that a vending machine could use to give change. Assume that the person will always insert one to five dollar bills and that the cost of the item can be any amount less than five dollars. The machine will dispense change in some combination of dollar coins, quarters, dimes, nickels and pennies.

Print out Not enough Money. if the cost of the item exceeds the money put into the machine. For all other situations, determine what change to give. Assume that it is always better to use higher denominations than lower ones. The program should try to use as many dollar coins as possible and only use quarters if necessary. Then it must use as many quarters as possible and only use dimes if necessary; then as many dimes as possible before using nickels; and as many nickels as possible before using any pennies.
Question 7: Write a function called \textit{mirror\_image\_string} that takes a string as an argument and returns the string concatenated with the string backwards. For example,

\begin{verbatim}
mirror_image_string('red')
\end{verbatim}

would return a value of 'redder' The backwards portion should be created one character at a time using a loop.
Note about Question 8: This was probably the hardest question last term on any test over all three classes. Two people out of 50 got full credit on it. For this sample test, consider this a special challenge question.

**Question 8:** Write a function called `negotiator` that will negotiate a price on your behalf. Assume you are buying some item and the program is negotiating with the seller. (Optionally you can add the name of the item as an additional parameter and use it in some of your print statements. However, this is not necessary for the test.)

Given an initial low price and an initial high price, the negotiator will use your low price as its first bid and then negotiate with the seller to purchase the item at your high price or lower. The negotiator program should implement the algorithm in the figure on the next page.

You should use the functions listed below in your code as necessary. In order to simplify the problem for a testing environment, `set_offer` and `get_most_reasonable_offer` are specifically mentioned in the flowchart.

The function should be called with your initial high and low prices as parameters (arguments). Note that the function `get_most_reasonable_offer` uses the `input` function for getting input from the seller. You can either get all your bids from the seller using this function, or use the `input` function in your own way to get bids from the seller.

```python
def yes_or_no(input_string):
    output = ''
    output_string = ''
    while (output == ''):
        print ('Please answer yes or no to the following question.
        output_string = input(input_string)
        if (output_string == 'yes' or output_string == 'Yes' 
            or output_string == 'YES'):
            output = True
        elif (output_string == 'no' or output_string == 'No' 
            or output_string == 'NO'):
            output = False
        else:
            print('I don\'t understand your answer.\n        return(output)

def set_offer (high, low):
    ratio = high/low
    if ratio >= 3:
        return(2*low)
    elif ratio >= 1.2:
        return(1.25*low)
    elif ratio <= 1.1:
        return(high)
    else:
        return ((high + low)/2)

def get_most_reasonable_offer(high):
    offer = float(input('How much do you want for it?'))
    rejected=False
    question = 'I cannot accept your current offer. Do you still want to do business?: '
    new_offer = 99999999
    while(not rejected) and (offer > high) and yes_or_no(question):
        new_offer = float(input('How much do you want for it?'))
        if (new_offer > offer):
            rejected = True
        else:
            offer = new_offer
    return(offer)
```

Figure 1: The Negotiator Algorithm

1. **Start**
2. Set current high and current low price at Buyer’s discretion.
3. Offer to purchase for current low.
4. Obtain an offer from the seller using `get_most_reasonable_offer(current_high)`.
5. If offer accepted?
   - Yes: Print 'We have a deal!' and return True.
   - No: Set current high to seller’s price. Set current_low to output of `set_offer(current_high,current_low)`.
6. Does the seller want to sell the item?
   - Yes: End.
   - No: Is the price less than or equal to current_low?
     - Yes: Offer to purchase for current low.
     - No: Print 'I hope we can do business in the future' and return False.
7. Is the price less than or equal to current_high?
   - Yes: Print 'I am sorry, but I cannot afford to make this purchase.' and return False.
   - No: Go back to step 4.

**Diagram Notes:**
- `get_most_reasonable_offer` and `set_offer` functions are not defined in the text.