There are 2 sections, the first section is worth 50 points and the second section is worth 50 points, for a total of 100. Answer all 4 questions in Section 1. Answer 2 out of the 3 questions in Section 2.

It is essential that you **PUT YOUR NAME AND NET ID 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. There is a GLOSSARY OF TERMS at the end of the test. Please feel free to look up some of the basics in this glossary. I will also answer any reasonable look-up style questions as I am more interested in your ability to reason than your ability to memorize.

1 Answer questions about code.

Worth 50 points
Each example consists of Python code, followed by questions and places to fill in answers. Please read all questions carefully and answer them as precisely as possible. Assume that there are no bugs in the code that will make the system crash, although the code may not solve a 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:

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

Question: What is the value of the variable `output`?

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**

```python
def multi_level_sharing(total,levels):
    ## computes shares of profit for multi-level
    ## marketing, ponzi schemes, or chain letters
    ## assumes fixed number of levels and division by 2
    new_total = total
    for level in range(1,levels+1):
        new_total = new_total/2
        print(level,new_total)

multi_level_sharing(64,5)
```

Question: write out each line that would be printed as a result of executing the code above.
import math
import turtle

def turtle_setup():
    global my_screen
    global my_turtle
    my_screen = turtle.Screen()
    my_turtle = turtle.Turtle()

def spikey_star(number_of_spikes,length):
    turtle_setup()
    inside_angle = 360/number_of_spikes
    for num in range(number_of_spikes):
        my_turtle.pd()
        my_turtle.fd(length)
        my_turtle.pu()
        my_turtle.left(180)
        my_turtle.fd(length)
        my_turtle.left(180+inside_angle)

spikey_star(5,50)

**Question:** draw (approximately) whatever would be drawn by the turtle program above.
def choose_next_line(number, other_number, eye_number=False):
    if eye_number == 1:
        eyeball = 'O'
    elif eye_number == 2:
        eyeball = '*'
    elif eye_number == 3:
        eyeball = '+'
    else:
        eyeball = '#'
    if number == 1:
        return('˜'*other_number)
    elif number == 2:
        return('o'*other_number)
    elif number == 3:
        return((' ')*((other_number//2)-2)+'[--]')
    elif number == 4:
        return((((other_number//2)-4)*' ')+'<'+eyeball+'>'+3*' '+'<'+eyeball+'>')
    elif number == 5:
        return((' '*(other_number//4))+('ˆ'*(other_number//2)))
    elif number == 6:
        return((((other_number//2)-3)*' ')+eyeball+5*' '+eyeball)
    elif number == 7:
        return((' '*(other_number//4))+('v'*(other_number//2)))
    elif number == 8:
        spaces = math.floor(other_number * 3/8)
        return((' ')*spaces+("o"*spaces))

def draw_that_face():
    print(choose_next_line(2, 20))
    print(choose_next_line(4, 20, eye_number=2))
    print(choose_next_line(3, 20))
    print()
    print(choose_next_line(5, 20))

draw_that_face()

Question: Write out (approximately) what would be printed out by the program above.
def convert_roman_char(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_to_number(input_string):
    input_string = input_string.upper()
    output1 = []
    for character in input_string:
        output1.append(convert_roman_char(character))
    previous_number = 'START'
    total = 0
    for number in output1:
        if previous_number == 'START':
            previous_number = number
        elif previous_number < number:
            increment = number-previous_number
            total = total+increment
            previous_number = 'START'
        else:
            total = total+previous_number
            previous_number = number
    if previous_number !='START':
        total = total+previous_number
    return(total)

for numeral in ['XLV','MMXVI','MMMMMMMCILXVIV','CICVI',]:
    print(numeral,roman_numeral_to_number(numeral))

Question: write out each line that would be printed as a result of executing the code above.
2 Write Code

Worth 50 points
Answer 2 of the 3 questions in section 2 (Questions 5, 6 and 7). For each question that you answer, write a Python program as specified. If you choose to answer all 3 questions, please indicate which ones you would like to count for the test.

**Question 5:** Using the turtle module, write a program that draws a balloon tree, a combination of a line and some circles, like the one in Figure 1. The program should take a number of circles and the size of the radius as parameters. For example, the command `draw_balloon_tree(5,15)` produced 5 circles with radius 15 to produce the picture in figure 1. The program should draw a line and circles that touch the line. The circles should alternate between being on one side of the line and on the other side. The circles should be next to each other, e.g., the second circle starts where the first one ends. Note the following:

- A turtle draws circles above a line when it faces right (→) and below the line when it faces left (←).
- A turtle starts and ends in the same place when it draws a circle.
Figure 2: make_branching_tree(7,15)

**Extra Credit for Question 5:** Only do this part if you have time. Write a function that calls your balloon tree program twice, upward and at 90 degree angles from each other, i.e., by turning the turtle in the appropriate directions before drawing (and starting both trees from the same screen position). Figure 2 is an example of what such a program would draw given the command `make_branching_tree(7,15)`. The program would call `draw_balloon_tree(7,15)`, 2 times, producing branches in different directions.
Question 6: A series of 5 psychological experiments test a subject’s reaction times to 5 different stimuli. For each subject, reaction times for tests 1, 2, 3, 4, and 5 are stored in a list. The following global variable `react_times` stores the results (in seconds) from 4 subjects:

```python
react_times = [[1, 2, 3, 4, 5], [5, 2, 3, 2, 2], [1, 1, 1, .5, 2], [3, 3, 3, 1, 1]]
```

Create a program that will loop through this sort of data and compute an average reaction time for each of the tests. For example, the command `average_reaction_times(react_times)` would return a list of average times: [2.5, 2.0, 2.5, 1.875, 2.5], such that 2.5 (seconds) is the average reaction time for test 1, 2.0 is the average reaction time for test 2, etc. Your program should work for any number of subjects and any number of tests, such that the results are recorded in this format.
**Question 7:** Wikipedia (https://en.wikipedia.org/wiki/Pig_dice_game) describes the dice game pig as follows:

Each turn, a player repeatedly rolls a die until either a 1 is rolled or the player decides to "hold": If the player rolls a 1, they score nothing and it becomes the next player’s turn. If the player rolls any other number, it is added to their turn total and the player's turn continues. If a player chooses to "hold", their turn total is added to their score, and it becomes the next player’s turn. The first player to score 100 or more points wins.

Implement a version of this dice game in which the user plays against the computer. The computer will always “hold” if it rolls higher than 3. The user can choose when they want to “hold” or continue rolling. You can call the following 2 functions without defining them:

```python
def roll_die():
    import random
    return(random.randint(1,6))

def yes_or_no():
    answer = 'blah'
    while not answer in ('y','n'):
        answer = input('Answer Y for yes or N for no: ')
        answer = answer.lower()
    if answer == 'y':
        return(True)
    else:
        return(False)
```

A sample game follows:

```python
>>> pig()
Computer 0 Player 0
Your Turn
You rolled 1
You scored nothing this round
Computer rolled 2
Computer rolled 1
Computer scored nothing this round
Computer 0 Player 0
Your Turn
You rolled 6
Do You Want to Hold?
Answer Y for yes or N for no: n
You rolled 4
Do You Want to Hold?
Answer Y for yes or N for no: y
Computer rolled 4
Computer 10 Player 26
Your Turn
You rolled 1
You scored nothing this round
Computer rolled 5
Computer 15 Player 26
Your Turn
You rolled 1
You scored nothing this round
Computer rolled 5
Computer 20 Player 26
Your Turn
You rolled 4
Do You Want to Hold?
Answer Y for yes or N for no: n
You rolled 3
Do You Want to Hold?
Answer Y for yes or N for no: y
Computer rolled 6
Computer 6 Player 16
Your Turn
You rolled 2
Do You Want to Hold?
Answer Y for yes or N for no: n
You rolled 6
Do You Want to Hold?
Answer Y for yes or N for no: n
You rolled 4
Do You Want to Hold?
```
Do You Want to Hold?
Answer Y for yes or N for no: y

Computer rolled 1
Computer scored nothing this round
Computer 20 Player 41
Your Turn
You rolled 3
Do You Want to Hold?
Answer Y for yes or N for no: n
You rolled 4
Do You Want to Hold?
Answer Y for yes or N for no: n
You rolled 6
Do You Want to Hold?
Answer Y for yes or N for no: n
You rolled 4
Do You Want to Hold?
Answer Y for yes or N for no: y

Computer rolled 4
Computer 26 Player 58
Your Turn
You rolled 4
Do You Want to Hold?
Answer Y for yes or N for no: n
You rolled 5
Do You Want to Hold?
Answer Y for yes or N for no: n
You rolled 2
Do You Want to Hold?
Answer Y for yes or N for no: n
You rolled 6
Do You Want to Hold?
Answer Y for yes or N for no: n
You rolled 6
Do You Want to Hold?
Answer Y for yes or N for no: n
You rolled 5
Do You Want to Hold?
Answer Y for yes or N for no: y

Computer rolled 2
Computer rolled 3
Computer rolled 5
Computer 42 Player 92
Your Turn
You rolled 1
You scored nothing this round
Computer rolled 2
Computer rolled 3
Computer rolled 5
Computer 55 Player 92
Your Turn
You rolled 6
Do You Want to Hold?
Answer Y for yes or N for no: n
You rolled 2
Do You Want to Hold?
Answer Y for yes or N for no: y
Computer rolled 3
Computer rolled 3
Computer rolled 6
You Win

>>>
Basic Stuff to Look Up for the Test

1. Some Basics

- **return(X)** causes the current function to exit and cause the expression represented by the function call to evaluate as X. For example given the following steps, the value of output would be 5:

  ```python
  def add(num1,num2):
      return(num1+num2)
  output = add(2,3)
  ```

- **print(X)** prints X to the screen. This is only for the benefit of the user. It is not useful for having programs interact.

- The parameters of a function are the local variables inside of the parentheses in the function definition. They are useful when you have functions call functions.

- **input(prompt)** is used to ask a human being a question so that a program can interact with a human being. This is useful when you want a human being to enter information interactively. *input* statements should be used only when human interaction is appropriate. *input* statements return a string corresponding to what the user typed in. It may be necessary to convert this string to some other data type, e.g., an integer (with *int*) or a float (with *float*).

- The operator + will add two numbers or concatenate two strings

- The operator * will multiple two numbers or print a string some number of times.

2. sequences

- object made up of other objects in an order

- the function len(sequence) returns the number of items in the sequence

- the operator in tests for membership in sequence, e.g., (’a’ in ’abc’) would have the value True.

- sequences are used in for loops (see below)

- indices and slices
  - Indices in a sequence are numbers from zero to the length of the string. Zero refers to the position before the first item in the string and the length of the string refers to the position following the last item. Thus each item in the sequence are between two consecutive indices. For example, the subscripted numbers indicate indices for the string The book: ’0T1h2e3 4b5o6o7k8. Similarly, the indices in [’0 ’The’,1 ’book’,2 is,3 ’there’,4] indicate positions in the list [’The’, ’book’, ’is’,’here’].
  - sequence[num] indicates an element in a sequence beginning at num (a number from zero to one less than the length of the string), e.g., ’The book’[4] evaluates to ’b’; [’The’, ’book’, ’is’,’here’][0] evaluates to ’The’.
  - Leaving out the number before the colon suggests that a subsequence begins at the beginning of the sequence and leaving out the number after the colon suggests that the subsequence ends at the end of the list. Thus ’The book’[:3] evaluates as ’The’ and [’The’, ’book’, ’is’,’here’][2:] evaluates as [’is’,’here’].

- ranges
  - range(5) is approximately equivalent to [0,1,2,3,4]
  - range(1,5) is approximately equivalent to [1,2,3,4]

- Strings
  - an empty string has zero characters”
– strings are sequences of characters, e.g., 'Hello World!' consists of the items ['H', 'e', 'l', 'o', 'W', 'o', 'r', 'l', 'd', '!']
– string1.strip(string2) – removes instances of string2 from beginning and end of string. For example, '***Hello World***'.strip('*') will return 'Hello World'.
– string.split(character) – creates a list by dividing a string at each instances of character. For example, 'Hello World'.split(' ') will return the list ['Hello', 'World'].
– string.lower() converts string to lower case

• Lists
  – A list is represented by square brackets surrounding a list of objects, divided by commas, e.g., ['A', 'List', 'of', 'Strings']
  – Lists are mutable.
  – You can add an object to the end of a list using the append method. For example, suppose my_list = ['a', 'b', 'c']. Then my_list.append('d') will add 'd' to the end of my_list, setting it to ['a', 'b', 'c', 'd'].

3. Division and Modulus

• \( 5 \div 2 = \) 2
• \( 5/2 = 2.5 \)
• \( 5\%2 = 1 \)

4. print

• sep – separator between items
• end – printed at the end of print statement

5. for loops

• First Line: for VARIABLE in SEQUENCE:
• VARIABLE is set to each item in the sequence one at a time
• The Indented body repeats once for each item in sequence (for each setting of VARIABLE).
• It is common to exit a loop of any kind by using a return to exit the function.
• It is common to initialize a variable outside a loop (called an accumulator) that then gets incremented inside the loop.

6. if statements

• the first line of an if statement consists of if BOOLEAN-EXPRESSION:
• the body of text indented under the first line is executed if the BOOLEAN-EXPRESSION evaluates to True
• the if statement can be followed by optional elif statements of the same form, except that the first line begins with elif. Each elif statement is only evaluated if the BOOLEAN expressions in the if and elif statements leading up to this one are False.
• The block of if and optional elif statements can end with an optional else statement. The first line is simply else:. The body of text under else executes if the Boolean expressions for all previous if and elif statements in the sequence evaluate to False.

7. Turtles

• Screen and Turtle objects are created using the commands turtle.Screen() and turtle.Turtle().
• The turtle is initially in the center of the screen facing rightward.
• my_turtle.left(degrees) – rotates the my_turtle degrees left (from its perspective).
• `my_turtle.fd(distance)` – moves the `my_turtle` distance units forward.
• `my_turtle.pu()` – picks the pen up
• `my_turtle.pd()` – puts the pen down (ready to write)
• `my_turtle.circle(radius)` – creates a circle with radius `radius`. The circle will be above the direction the turtle was facing when it started drawing. The turtle will move left and up in a circle and end up in the same place as before.

8. `time.sleep(sec)` – pauses for sec seconds (requires the module sleep to be imported)