Name: ____________________________
Net ID ________________

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.

Section 1: (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:

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

def turtle_line_up():
    import turtle
    global my_turtle1
    global my_turtle2
    global my_screen
    my_screen = turtle.Screen()
    my_turtle1 = turtle.Turtle()
    my_turtle2 = turtle.Turtle()
    my_turtle1.pu()
    my_turtle2.pu()
    my_turtle1.setposition(0,100)
    my_turtle2.setposition(0,-100)
    my_turtle1.pd()
    my_turtle2.pd()
    my_turtle1.setposition(0,0)
    my_turtle2.setposition(0,0)

turtle_line_up()

Question: Draw (approximately) the configuration that would be drawn by the turtle program.
def word_combo (word1,word2):
    return(word1+'_'+word2)

def word_list_combo(word_list1,word_list2):
    output = []
    for word1 in word_list1:
        for word2 in word_list2:
            output.append(word_combo(word1,word2))
    return(output)

def main():
    list1 = ['cat','horse','chicken','lettuce']
    list2 = ['bus','head','bed','balloon']
    output = word_list_combo(list1,list2)
    for word in output:
        print(word)

main()

Question: Indicate what would be printed out by the code above.
Question 3

def decode_number_list(number_list):
    output = ''
    offset = ord('a')-1
    # this number + 1 will equal the
    # unicode value of 'a'
    for number in number_list:
        if number == 0:
            output = output + ' '
        else:
            character = chr(offset+number)
            # adding the offset 96 to 1,
            # will result in 97, the value for 'a'
            output = output + character
    return(output)

input_list = [20, 23, 15, 0, 23, 15, 18, 4, 19]
output_string = decode_number_list(input_list)
print(output_string)

Question: Indicate what would be printed out by the code above.
def count_non_vowels(word):
    total = 0
    for char in word:
        if char in 'aeiou':
            # pass doesn't do anything
        elif char == 'y':
            # count y as half a consonant
            total = total + .5
        else:
            total = total + 1
    return(total)

def non_vowel_count_sort(inlist):
    pairs = []
    for word in inlist:
        pairs.append([count_non_vowels(word),word])
    pairs.sort()
    # print(pairs)
    output = []
    for pair in pairs:
        output.append(pair[1])
    return(output)

word_list = ['first','hymn','compromise','polynomial','roadway','yak']
output = non_vowel_count_sort(word_list)
for word in output:
    print(word,end=' ')
print()
Question 5: Write a program that returns a list of ingredients for making a vinaigrette salad dressing. We will define Vinaigrette salad dressing as a dressing containing oil, vinegar, salt, pepper and other ingredients—the program will determine these other ingredients randomly. You can assume that your program has access to the following variables (you do not have to define them in your program):

```python
salad_ingredients = ['lemon', 'mustard', 'raspberry', 'sugar',
                     'honey', 'ginger', 'dill', 'oregano', 'garlic']
non_vegan = ['bacon', 'sour cream', 'grated cheese']
```

The output of the program should be a list of lists. Each sublist should contain a number (indicating an amount) and the name of an ingredient. Sample output of the program could be:

```python
 [2, 'raspberry'], [2, 'sugar'], [1, 'ginger'], [1, 'dill'], [2, 'sour cream']]
```

The program should have the following structure:

1. Initialize a variable `output` as the empty list
2. Ask the user if they want more vinegar than oil and then append pairs representing the amounts of oil and vinegar that are added to `output`. If they want more vinegar, `[3, vinegar]` and `[2, oil]` should be added. Otherwise, `[2, vinegar]` and `[3, oil]` should be added.
3. Add `[1, 'pepper']` to `output`.
4. Add a list consisting of a number and the word 'salt' to the output. The number should be 1 if the user wants the soup to be low salt and 2 otherwise (query the user).
5. Loop through the list of `salad_ingredients`. For each ingredient, there is a 40% chance the ingredient should be added. Randomly generate a number to determine this, e.g., use `random.random()` to generate a number from 0 to 1 and only add the ingredient if the result is less than or equal to .4. If the ingredient gets added, randomly choose whether to add 1 or 2 units of that ingredient.
6. Give the user the option of making the dressing vegan. If they choose to make the dressing non-vegan, the program should randomly choose one item from this list bound to the variable `non_vegan`. Randomly choose to add either 1 or 2 units of this ingredient to `output`.
7. Return or print out the resulting list of ingredients (`output`) for the salad dressing.

The following is a sample interaction with the program, followed by printing each of the ingredients on a separate line.

```python
>>> generate_salad_dressing()
Welcome to the vinaigrette salad dressing generator.
Do you like more vinegar than oil?
Yes or No? no
Do you want low salt dressing?
Yes or No? no
Adding random ingredients.
Adding lots of lemon
Adding lots of mustard
Adding lots of raspberry
Adding lots of sugar
Adding a little ginger
Adding a little dill
Would you like this to be a vegan salad dressing?
```
Yes or No? no
Adding lots of sour cream
[1, ‘pepper’]
[2, ‘vinegar’]
[3, ‘oil’]
[2, ‘salt’]
[2, ‘lemon’]
[2, ‘mustard’]
[2, ‘raspberry’]
[2, ‘sugar’]
[1, ‘ginger’]
[1, ‘dill’]
[2, ‘sour cream’]

>>>
Question 6: Write a turtle program called `turtle_pie` which draws a circle (of radius 100) divided into equal sections as in the example below. You should use a parameter to determine the number of sections (an integer greater than 1) to divide the circle into. The example below was generated by the command `turtle_pie(3)` and it divides the circle into 3 sections. Your program should first draw the circle, then it should keep going to the center of the circle and drawing lines from the center to the edge (each line is a radius of the circle and therefore is 100 long). The angle between lines can be determined based on the fact that a circle is 360 degrees and the lines should divide the circle into equal parts. Remember, you can change the angle the turtle is facing by having the turtle turn left (`my_turtle.left(degrees)`), having the turtle turn right (`my_turtle.right(degrees)`), or setting its heading (`my_turtle.setheading(degrees)`).
**Question 7:** Write a game program called `dice_poker` in which a player (or user) plays a game against the computer (aka, the dealer). This is a simplified version of poker using dice instead of cards. The object of this game is to maximize the number of dice with the same value as possible. This is reflected in the scoring system (discussed below).

In the first round, the player and the dealer each roll 5 virtual dice. (a number from 1 to 6 is returned by `random.randint(1,6)`). Then the player is given the opportunity to reroll up to three of these dice. They are queried one die at a time if they want to reroll that die, but are not permitted to reroll more than 3. The dealer always rerolls the 3 lowest scoring dice in its hand.

The player and dealer’s dice are then compared to determine the winner. Each hand has two scores: 1) the number of duplicates of any one number, e.g., [2,3,3,4,3] has 3 of the same number, but [2,1,1,4] only has 2 of the same number; and 2) the sum of the dice in the hand. The player has the highest number of duplicates wins the round. If both players have the same number of duplicates, the player with the highest sum wins. If both players have the same number of duplicates and the same sums, the dealer wins.

Two Sample games follow:

```python
>>> dice_poker()
Your hand is [4, 4, 2, 1, 3]
You can reroll up to 3 of the items in your hand.
Would you like to reroll dice number 1 which is a 4 ?
Yes or No? no
Would you like to reroll dice number 2 which is a 4 ?
Yes or No? no
Would you like to reroll dice number 3 which is a 2 ?
Yes or No? yes
Your hand is now [4, 4, 3, 1, 3]
You can reroll 2 more dice
Would you like to reroll dice number 4 which is a 1 ?
Yes or No? yes
Your hand is now [4, 4, 3, 3, 3]
You can reroll 1 more dice
Would you like to reroll dice number 5 which is a 3 ?
Yes or No? no
Your sorted hand is [3, 3, 3, 4, 4]
You win
Your hand was: [3, 3, 3, 4, 4] with a count of 3 and a score of 17
The dealer’s hand was: [3, 4, 5, 6, 6] with a count of 2 and a score of 24
```

```python
>>> dice_poker()
Your hand is [2, 1, 5, 4, 1]
You can reroll up to 3 of the items in your hand.
Would you like to reroll dice number 1 which is a 2 ?
Yes or No? yes
Your hand is now [2, 1, 5, 4, 1]
You can reroll 2 more dice
Would you like to reroll dice number 2 which is a 1 ?
Yes or No? no
Would you like to reroll dice number 3 which is a 5 ?
Yes or No? yes
Your hand is now [2, 1, 6, 4, 1]
You can reroll 1 more dice
Would you like to reroll dice number 4 which is a 4 ?
Yes or No? yes
Your hand is now [2, 1, 6, 4, 1]
You can reroll 0 more dice
Your sorted hand is [1, 1, 2, 4, 6]
You lose
Your hand was: [1, 1, 2, 4, 6] with a count of 2 and a score of 14
The dealer’s hand was: [1, 2, 4, 4, 5] with a count of 2 and a score of 16
```
```
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:

```
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.
- The operator ** will represent exponents, e.g., 5**2 == 25.

2. Division and Modulus

- 5 // 2 == 2
- 5/2 == 2.5
- 5%2 == 1

3. More Math

- `round(4.5) == 4, round(4.6) == 5, round(-4.5) == -4, round(-4.6) == -5`
- `round(2/3,2) == .67` ## The second argument of round indicates number of decimal places. The default is to round to the nearest whole number.
- `math.floor(4.9) == 4, math.floor(-4.9) == 5`
- `math.ceil(4.9) == 5, math.ceil(-4.9) == -4`
- `math.trunc(4.9) == 4, math.trunc(-4.9) == -4`
- `math.pi == 3.141592653589793` – a variable for the value of pi

4. 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 subcribed numbers indicate indices for the string *The book*: '0T1h2e3a4b5o6t7k8. Similarly, the indices in 
```'
'The', 'book', 'is', 'there'
```
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'.

sequence[num1:num2] indicates a subsequence beginning at position num1 and ending at num2,e.g., 'The book'[4:6] evaluates to 'bo'; ['The', 'book', 'is','here'][0:2] evaluates to ['The', 'book'].

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', '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'].
  – You can pop an item off the end of a list with the pop method. pop will return the removed item. For example, if my_list is set to ['a', 'b', 'c', 'd'], then my_list.pop() will return 'd' and shorten the list to ['a', 'b', 'c'].
  – You can combine two lists together using extend. If my_list is set to ['a', 'b', 'c'], then the command my_list.extend(['d', 'e', 'f']) will reset my_list to ['a', 'b', 'c', 'd', 'e', 'f'].
  – The method sort will put a list in order based on how < applies to pairs of elements in the list. This will only work if the members of the list are comparable with each other, e.g., you cannot compare an integer to a string.

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

6. 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.

7. **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.

8. **logical operators**

- `X and Y` returns True only if both X and Y are True
- `X or Y` returns True only if X is True, Y is True or both are True
- `X in Sequence` returns True if X is in a member of a sequence, e.g., `'a' in 'abcdefg'` would return True
- `X == Y` returns True if X and Y are the same
- `X != Y` returns True if X and Y are the different
- `X < Y` returns True if X is less than Y
- `X > Y` returns True if X is greater than Y
- `X <= Y` returns True if X is less than or equal to Y
- `X >= Y` returns True if X is greater than or equal to
- `Not X` returns True if X is False
- The operators `<`, `>`, `<=`, `>=` can apply to non-numeric objects. Characters are compared based on their unicode values (so `'a' < 'b'`). True is assumed to be greater than False. Sequences are compared lexicographically – the first item in pairs of sequences are compared first, but if they are equal the second items are compared, and so on.

9. **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.

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

11. **random** – the random module

- `random.random()` returns a number between 0 and 1
- `random.randint(num1,num2)` returns a number between num1 and num2 (inclusive).
- `random.choice(sequence)` returns member of `sequence`. 