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.

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:

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 take_em_off (inlist, total):
    output = total
    for number in inlist:
        output = output - number
    return(output)

answer = take_em_off([50,15,10,7],100)

Question: What is the value of the variable answer, after the code above executes.
Question 2

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

def grapes_on_a_stick(number, radius):
    turtle_setup()
    my_turtle.pd()
    my_turtle.left(90)
    my_turtle.fd(radius)  ## move before first grape
    my_turtle.right(90)
    for num in range(number):
        my_turtle.circle(radius)
        my_turtle.left(90)
        my_turtle.fd(radius*2.25)  ## diameter + extra space
        my_turtle.right(90)
    my_turtle.right(90)

grapes_on_a_stick(5, 25)

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))+(u'ˆ'*(other_number//2))
    elif number == 6:
        return (((other_number//2)-3)*' ')+ eyeball+5*' '+ eyeball
    elif number == 7:
        return (u'v'*(other_number//2))
    elif number == 8:
        spaces = math.floor(other_number * 3/8)
        return (' ')*spaces+(u'o'*spaces)

def draw_that_face():
    print(choose_next_line(8, 20))
    print()
    print(choose_next_line(4, 20, eye_number=4))
    print(choose_next_line(3, 20))
    print()
    print(choose_next_line(5, 20))
    print(choose_next_line(7, 20))

draw_that_face()
**Question 4**

```python
ity_words = ['able','brief','complex','human','magnanimous',
             'mediocre','pompous','probable','rare','sublime']

def add_ity(word):
    if len(word)<3:
        return(word)
    elif word[-3:]=='ous':
        return(word[:-3]+'ity')
    elif word[-1] == 'e':
        return(word[:-1]+'ity')
    elif word[-1] == 'f':
        return(word[:-1]+'vity')
    else:
        return(word+'ity')

for word in ity_words:
    print(word,add_ity(word))
```

**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 creates zigzagging lines such as the one in figure 1. The main function of the program should take 3 parameters, the number of zigzags, the angle between lines in the zigzag and the length of each line. For example, figure 1 could be generated by the command `draw_zig_zags(5,45,25)`, where the first parameter indicates 5 pairs of lines, the second parameter indicates a 45 degree angle between each pair of connected lines in the zigzag, and the third parameter indicates a 25 pixel length for each line. Each pair of lines forms a zigzag, thus the example contains 5 zigzags or 10 lines in total. The angle between each pair of lines is equal to angle (in this case 45 degrees) and the angles alternate directions (left, than right or right than left).
Question 6: The following list of lists represents a simple street map grid (e.g., for a game). Each list represents a street and each position in a list represents an avenue. Thus house5 is on street 3, avenue 3, if we are counting from 1 to 5. In other words, house5 is equal to street_map[2][2] – we add 1 to these positions because it is appropriate to start with 1 when naming a street or an avenue, i.e., there is typically no such thing as an avenue zero.

street_map = [[' ','toys','food','house1','house2'],
       ['house3','zoo','haunted_house','house4',' '],
       ['furniture','castle','house5','armory','ducks'],
       ['house6',' ','park',' ','house7','house8'],
       ['laundry','school','house9','sheriff','farm']]

Write a function that takes two parameters: the name of a place and a map (list of lists like street_map). The function should print out or return the street and avenue of the place. There should also be some message indicating places that are not found. For example, the function could work as follows:

```python
>>> name_place_on_map('castle',street_map)
castle is at 3 street 2 avenue
>>> name_place_on_map('farm',street_map)
farm is at 5 street 5 avenue
>>> name_place_on_map('sugar mountain',street_map)
sugar mountain is not on the map.
```  

Extra Credit for Question 6: Only do this part if you have time. Write a function that gives directions based on two places and a map of the same form. For purposes of the program, you can assume that people know how to walk in straight lines between streets in a standard grid and that they also know how to walk in straight lines between avenues. For example, the program could work as follows.

```python
>>> give_directions('zoo','farm',street_map)
You are at 2 street 2 avenue
You are going to 5 street and 5 avenue
Walk from 2 street to 5 street.
Then walk from 2 avenue to 5 avenue.
```
Question 7: Write a program for playing a game dice war in which a user plays against the computer. The player and the computer each start with a list of five random rolls between 1 and 6. We will refer to these lists as the player’s hand and the computer’s hand. Each round consists of the following steps:

- Pop off the first die from each hand and compare them. Remember using my_variable = my_list.pop(index) removes the item at that index from the list and returns it, thus setting my_variable to that item.

- If the 2 die are the same, store both die in a list (called winnings).

- If the player’s die is greater than the computer’s die, add both die to the end of the player’s hand (using append or extend). If there are any items from previous rounds stored in winnings, add these all to the end of the player’s hand (using extend).

- If the computer’s die is higher than the player’s, add both die to the end of the computer’s hand (using append or extend). If there are any items from previous rounds stored in winnings, add these all to the end of the computer’s hand (using extend).

The game ends either when a maximum of 25 rounds have been played, when the player has no more dice left, or when the computer has no more dice left. If the player has more dice left at the end than the computer, the player wins. If the computer has more dice, the computer wins. Otherwise, it is a tie. A sample game follows.

```python
>>> dice_war(5,25)
Round 1
Your Current Hand: [2, 3, 5, 1, 5]
Computer’s Hand: [4, 5, 1, 3]
player 2 computer 4
You lose this round
Hit return when ready for next round.
Round 2
Your Current Hand: [3, 5, 1, 5]
Computer’s Hand: [5, 5, 1, 3, 2, 4]
player 3 computer 5
You lose this round
Hit return when ready for next round.
Round 3
Your Current Hand: [5, 1, 5]
Computer’s Hand: [5, 1, 3, 2, 4, 3, 5]
player 5 computer 5
Hit return when ready for next round.
Round 4
Your Current Hand: [1, 5]
Computer’s Hand: [1, 3, 2, 4, 3, 5]
player 1 computer 1
Hit return when ready for next round.
Round 5
Your Current Hand: [5]
Computer’s Hand: [3, 2, 4, 3, 5]
player 5 computer 3
You win this round
You get additional items [5, 5, 1, 1]
Hit return when ready for next round.
Round 6
Your Current Hand: [5, 3, 5, 5, 1, 1]
Computer’s Hand: [2, 4, 3, 5]
player 5 computer 2
You win this round
Hit return when ready for next round.
Round 7
Your Current Hand: [3, 5, 5, 1, 1, 5, 2]
Computer’s Hand: [4, 3, 5]
player 3 computer 4
You lose this round
Hit return when ready for next round.
Round 8
Your Current Hand: [5, 5, 1, 1, 5, 2]
Computer’s Hand: [3, 5, 3, 4]
player 5 computer 3
You win this round
Hit return when ready for next round.
Round 9
Your Current Hand: [5, 1, 1, 5, 2, 5, 3]
Computer’s Hand: [5, 3, 4]
player 1 computer 3
You lose this round
Computer gets additional items [5, 5]
Hit return when ready for next round.
Round 10
Your Current Hand: [1, 1, 5, 2, 5, 3]
Computer’s Hand: [3, 4]
player 1 computer 1
You lose this round
Hit return when ready for next round.
Round 11
Your Current Hand: [5, 2, 5, 3]
Computer’s Hand: [4, 1, 3, 5, 5]
player 1 computer 4
You lose this round
Hit return when ready for next round.
Round 12
Your Current Hand: [5, 2, 5, 3]
Computer’s Hand: [1, 3, 5, 5, 1, 4]
player 5 computer 1
```
You win this round
Hit return when ready for next round.

Round 13
Your Current Hand: [2, 5, 3, 5, 1]
Computer’s Hand: [3, 5, 5, 1, 4]
player 2 computer 3
You lose this round
Hit return when ready for next round.

Round 14
Your Current Hand: [5, 3, 5, 1]
Computer’s Hand: [5, 5, 1, 4, 2, 3]
player 5 computer 5
Hit return when ready for next round.

Round 15
Your Current Hand: [3, 5, 1]
Computer’s Hand: [5, 1, 4, 2, 3]
player 3 computer 5
Computer gets additional items [5, 5]
Hit return when ready for next round.

Round 16
Your Current Hand: [5, 1]
Computer’s Hand: [1, 4, 2, 3, 3, 5, 5]
player 5 computer 1
You win this round
Hit return when ready for next round.

Round 17
Your Current Hand: [1, 5, 1]
Computer’s Hand: [4, 2, 3, 3, 5, 5]
player 1 computer 4
You lose this round
Hit return when ready for next round.

Round 18
Your Current Hand: [5, 1]
Computer’s Hand: [2, 3, 5, 5, 1, 4]
player 5 computer 2
You win this round
Hit return when ready for next round.

Round 19
Your Current Hand: [1, 5, 2]
Computer’s Hand: [3, 3, 5, 5, 1, 4]
player 1 computer 3
You lose this round
Hit return when ready for next round.

Round 20
Your Current Hand: [5, 2]
Computer’s Hand: [3, 5, 5, 1, 4, 1, 3]
player 5 computer 3
You win this round
Hit return when ready for next round.

Round 21
Your Current Hand: [2, 5, 3]
Computer’s Hand: [5, 5, 1, 4, 1, 3]
player 2 computer 5
You lose this round
Hit return when ready for next round.

Round 22
Your Current Hand: [5, 3]
Computer’s Hand: [5, 5, 1, 4, 1, 3, 2, 5]
player 5 computer 5
Hit return when ready for next round.

Round 23
Your Current Hand: [3]
Computer’s Hand: [5, 1, 4, 1, 3, 2, 5]
player 3 computer 5
You lose this round
Computer gets additional items [5, 5]
Hit return when ready for next round.

Final Results
Player’s Final Hand: []
Computer’s Final Hand: [1, 4, 1, 3, 2, 5]
You Lose
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', '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 // 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)