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:

```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
import turtle

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

def simple_house():
    my_turtle.pd()
    for num in range(4):
        my_turtle.left(90)
        my_turtle.forward(100)
    my_turtle.pu()
    my_turtle.left(90)
    my_turtle.forward(100)
    my_turtle.pd()
    my_turtle.left(30) ## facing up, not facing right
    my_turtle.fd(100)
    my_turtle.left(120)
    my_turtle.fd(100)

def main():
    turtle_setup()
    simple_house()

main()
```

**Question:** Draw (approximately) the configuration that would be drawn by the turtle program.
def convert_instruct(instruc):
    if instruc == 'eye':
        return('<O>')  ## less than, O, greater than (length 3)
    elif instruc == 'eye_brow':
        return('*****')  ## 5 asterisks (length 5)
    elif instruc == 'nose_part_1':
        return('| |')## vertical line, space, vertical line (length 3)
    elif instruc == 'nose_part_2':
        return('<- ->')  ## less than, dash, space, dash, greater than (length 5)
    elif instruc == 'mouth1':
        return('\___________/') ## this mouth uses underscores
    elif instruc == 'mouth2':
        return('/-----------\') ## this mouth uses dashes
    elif type(instruc) == int:
        return(' ' * instruc) ## return instruc * number of spaces

def make_face_a():
    for line in [[3,'eye_brow',3,'eye_brow',3,'eye_brow'],
                 [4,'eye',5,'eye',5,'eye'],[12,'nose_part_1'],
                 [11,'nose_part_2'],[7,'mouth1']] :
        out_line = ''
        for instruc in line:
            out_line = out_line + convert_instruct(instruc)
        print(out_line)

make_face_a()

Question: After the above code executes, what would be printed out?
def inflate(start_number, inflation_list):
    total = start_number
    for inflation_rate in inflation_list:
        total = total * (inflation_rate + 1)
    return total

result = inflate(100, [.5, .05, .05])

Question: What is the value of the global variable result after the code executes?
Answer:
def add_ing (word):
    if len(word)<=1:
        print('error:',word,'cannot have an -ing ending')
        return(False)
    if word[-2:] in ['de','ge','ke','me','ne','pe','re','se',
        'te','ve','ze']:
        return(word[:-1]+'ing')
    elif (word[-1] in 'dgklmnprstz') and (word[-2] in 'aeiou'):
        return(word+word[-1]+'ing')
    else:
        return(word+'ing')

def main():
    output = []
    for word in ['abbreviate','abolish','sit','balk','ball oon']:
        output.append(add_ing(word))
    print(output)

main()

Question: What is the value of the local variable output when it is printed out during the execution of the main function?

Answer:
Section 2 (50 points): Write 2 of the 3 questions in this section. For each question, you do, 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:

```python
# Sample Output

average_price_print([['Milk', 2.7, 3.3, 4, 2.7, 3],
                     ['Wheat', 3.3, 4, 2, 3.7, 4, 4, 3.7],
                     ['Chicken', 4, 2.7, 3.3, 4]])

Milk: 3.14
Wheat: 3.59
Chicken: 3.5
```

Write a program that prints out names of commodities and their average prices, as in the example above. The function should take one input parameter: a list of lists. Each list in the parameter has the name of a commodity as the first item and the remaining items are recent prices. The program should be able to average any number of prices.

Extra Credit: Include an option, in which the lowest price and the highest price are dropped before computing the averaging.
Question 6: Write a program that uses the turtle module to draw three circles that are next to each other, so that their edges are just touching, as in the picture below. Keep in mind, that the turtles beginning position is in the center of the screen (0,0), facing rightward (→).
**Question 7:** Write a program in which the user plays a card game (War) against the program. Assume that the following list represents a deck of cards:

```python
deck_of_cards = [
    ['2', 'C'], ['2', 'D'], ['2', 'H'], ['2', 'S'], 
    ['3', 'C'], ['3', 'D'], ['3', 'H'], ['3', 'S'], 
    ['4', 'C'], ['4', 'D'], ['4', 'H'], ['4', 'S'], 
    ['5', 'C'], ['5', 'D'], ['5', 'H'], ['5', 'S'], 
    ['6', 'C'], ['6', 'D'], ['6', 'H'], ['6', 'S'], 
    ['7', 'C'], ['7', 'D'], ['7', 'H'], ['7', 'S'], 
    ['8', 'C'], ['8', 'D'], ['8', 'H'], ['8', 'S'], 
    ['9', 'C'], ['9', 'D'], ['9', 'H'], ['9', 'S'], 
    ['10', 'C'], ['10', 'D'], ['10', 'H'], ['10', 'S'], 
    ['J', 'C'], ['J', 'D'], ['J', 'H'], ['J', 'S'], 
    ['Q', 'C'], ['Q', 'D'], ['Q', 'H'], ['Q', 'S'], 
    ['K', 'C'], ['K', 'D'], ['K', 'H'], ['K', 'S'], 
    ['A', 'C'], ['A', 'D'], ['A', 'H'], ['A', 'S']
]
```

First the player and the computer are randomly given 7 cards. Your program should make sure that the same card is not picked twice. Suggestion: use the method `.pop` with a random number that is less than the length of the list, e.g.,

```python
new_card = deck_of_cards.pop(random.randint(0, len(deck_of_cards) - 1))
```

would assign `['5', 'D']` to the variable `new_card` and remove the card from the deck. Use `random.randint` to select the random number.

There are then seven rounds. For each turn, the program randomly selects one of its cards to play and the user also selects one of his/her cards (his/her choice). The cards are compared and the better card wins the round. Whoever wins at least 4 of the 7 rounds wins the game.

Each card consists of a face value (the first item in the list) and a suit (the second item). Possible face values include: '2', '3', '4', '5', '6', '7', '8', '9', 'J', 'Q', 'K', 'A'. Possible suits include: 'C', 'D', 'H' or 'S'. The value of the card is mainly determined by its face value, the first item in the card. The value can be determined based on its position in the list:

```python
face_card_order = ['2', '3', '4', '5', '6', '7', '8', '9', 'J', 'Q', 'K', 'A']
```

which can be derived by the method `.index`, e.g., `face_card_order.index('2')` equals 0, whereas `face_card_order.index('J')` equals 8. Therefore a 'J' is more valuable than a '2'. If the two cards have the same face values, compare the suits of the cards. The value of the suits can be determined by the greater than (`>`) operator (the suits are in alphabetic order and in all caps), e.g., 'D' > 'C' because the unicode value of 'D' is greater than the unicode value of 'C'.

**Extra Credit:** Modify the rules to make it more complicated. Do comparisons based on face value of the cards only, ignoring the suit. If there are ties, the winner of the next winning round wins the tying rounds as well. For example, if the third round is a tie, it will count as if the winner of the fourth round won the third round as well. If both the third and fourth round are ties, than the winner of the sixth round will win the third and fourth rounds as well. If the seventh round is a tie, players each draws one additional card and play an eighth round to settle the game. If the eighth round is a tie, a further card must be drawn and so on. In the unlikely event that the twenty-sixth round is a tie, the game ends in a tie.
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*: ’0T1h2e3b4o5o6r7k8. Similarly, the indices in [’The’, ’book’, ’is’, ’here’] 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’.

  - 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'].

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)