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:**

```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 cylinder(radius, height):
    turtle_setup()
    for half in range(2):
        my_turtle.pd()
        my_turtle.circle(radius)
        my_turtle.pu()
        ## move parallel to middle of circle before turning
        my_turtle.fd(radius)
        my_turtle.left(90)
        my_turtle.fd(radius)
        my_turtle.left(180)
        my_turtle.pd()
        my_turtle.fd(height)
        my_turtle.pu()
        ## move parallel to middle of circle before turning
        my_turtle.left(180)
        my_turtle.fd(radius)
        my_turtle.left(90)
        my_turtle.fd(radius)

cylinder(50, 200)
```

**Question:** draw (approximately) whatever would be drawn by the turtle program above.
Question 2

def average_list(list1, list2):
    index = 0
    max_length = max(len(list1), len(list2))
    output = []
    while index < max_length:
        if index < len(list1):
            value1 = list1[index]
        else:
            value1 = 0
        if index < len(list2):
            value2 = list2[index]
        else:
            value2 = 0
        output.append((value1 + value2) / 2)
        index = index + 1
    return(output)

answer = average_list([1, 2, 3, 4], [3, 3, 3])

Question: what is the value of the global variable answer after the above code executes.
Question 3

def inside_out(word):
    middle = len(word)//2
    if len(word)%2==0:
        return(word[middle:]+word[:middle])
    else:
        return(word[-1*middle:]+word[middle]+word[:middle])

def main():
    for word in ['cat','chicken','rabbit','goat','monkey','kangaroo','octopus']:
        print(word,inside_out(word))

main()

**Question:** List each line that prints out when the function `main` is executed?
Question 4

movie_list = [['Batman v Superman', 873],
              ['Captain America', 1153],
              ['Finding Dory', 1024],
              ['The Jungle Book', 966],
              ['The Secret Life of Pets', 871],
              ['Zootopia', 1023]]

## top 6 2016 movies with gross receipts in millions

def sort_movie_pairs(list_of_pairs):
    output = []
    for item in list_of_pairs:
        item.reverse()
        output.append(item)
    output.sort()
    output.reverse()
    return(output)

def main():
    answer = sort_movie_pairs(movie_list)
    for item in answer:
        print(item)
main()

Question: List each line that prints out when the function main is executed?
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: Write a function that takes as input 2 lists:

- A list of lists: Each sublist is a student record and it consists of 6 items, ordered as follows:
  1. An id string (‘id1’, ‘id2’, etc.)
  2. A quiz grade from 0 to 4
  3. A homework grade from 0 to 4
  4. A midterm1 grade from 0 to 4
  5. A midterm2 grade from 0 to 4
  6. A final grade from 0 to 4

- A list of weights: This is a list of floats between 0 and 1, ordered as follows:
  1. A quiz weight
  2. A homework weight
  3. A midterm1 weight
  4. A midterm2 weight
  5. A final-grade weight

Write a program that prints out ids followed by class grades. The class grade is calculated by looping through each student record (after the id), multiplying the grade by the corresponding weight, and summing up the total. The following is a sample run of such a program.

```python
>>> grades = [['id1',3,4,2,4,3.7],['id2',3.7,4,2.7,4,4],['id3',3.3,4,2.7,4,3.3],['id4',3.7,4,2.3,4,3.7]]
>>> weights = [.05,.2,.2,.2,.35]
>>> combine_scores(grades,weights)
id1 3.45
id2 3.73
id3 3.46
id4 3.54
```

In this example, grades is a list of student records. The first instance is [‘id1’,3,4,2,4,3.7]. The score for this one is equal to (.05*3)+(.2*4)+(.2*2)+(.2*4)+(.35*3.7) = 3.45.
Question 6: Write a program using the turtle library that draws a square with four circles around it, each circle touching the middle of one of the sides, as in the image above. The diameter of each circle, should be the same as the side of each square. Note that if the turtle is facing the right of the screen, my_turtle.circle(X) will draw a circle of radius X above the line. If it is facing left, it will draw that circle below the line, etc. (Note that 2 times the radius is the diameter.)
Question 7: Write a program for playing a dice game called *smush*. The object of the game is to get the highest score. First, the player and the computer each roll 7 dice. Then the player has the option of re-rolling 1, 2 or 3 dice. The player’s score is the sum of the 7 dice after the re-rolls. Next, the computer re-rolls 1, 2 or 3 dice. The computer will choose to re-roll the lowest scoring die, and the second lowest scoring die if it has a value of 3 or lower and the third lowest scoring die, if it has a value of 3 or lower. Here is an example game being played:

```python
>>> smush()
Your initial hand is [6, 6, 6, 5, 5, 4, 3] for a total score of 35
The computer’s initial hand is [6, 6, 6, 4, 3, 2, 1] for a total score of 28
Your current hand is [6, 6, 6, 5, 5, 4, 3]
Permitted rerolls is:  3
How many rerolls? 1
Your hand is [6, 6, 6, 5, 5, 4, 4] for a total score of 36
The computer’s hand is [6, 6, 6, 4, 6, 3, 2] for a total score of 33
You won this round.
```

**Extra Credit for Question 7:** Only do this part if you have time. The question 7 program represents a single round of *smush*. Modify the previous program so each round returns 'win', 'lose' or 'tie'. Write a program that uses these rounds as part of a betting game. The player starts out with $1000 and can bet any amount per round. In the case of a tie, no money is won or lost. A 'lose' results in losing the amount bet and a 'win' results in gaining the amount bet. The player must stop playing if he/she runs out of money, but can choose to stop playing at any time. If the player either runs out of money or stops with less than $500, print that they 'lost bigtime'. If the player stops with less than $1000, print that they 'lost some money'. If they stop with $1500 or more, print that they 'won big time'. If they stop with more than $1000 and less than $1500, print that they 'are a little better off'. If they have exactly $1000 when the game ends then print that they 'broke even'.
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: ’0’T1h2e3 4b5o6o7k8. Similarly, the indices in [0 ’The’,1 ’book’,2 is,3 ’there’] 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', '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)