Name: ________________________________
Net ID ________________

There are 2 sections. Each section is worth 50 points, for a total of 100.
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 resulted in a small deduction, but answering 2, would have resulted in an incorrect
answer.

Question 1: What prints out after the following code executes?

```python
def square_word(word, length):
    for num in range(length):
        print(word*length)

square_word('Blah', 5)
```
Question 2: What prints out after the following code executes? A simple calculator is suggested to figure out some of the details.

```python
def metrocard_calculator(money):
    ## money entered as a float
    if money>=5.5:
        total_money = money*1.05
    else:
        total_money = money
    ## add bonus for buying at least 2 rides
    pennies = round(total_money*100)
    ## convert total to number of pennies
    ## round makes sure this is an integer
    rides = pennies//275
    remainder = pennies%275
    dollars = remainder//100
    cents = remainder%100
    print('Good for',rides,'rides with',dollars,'dollars and',cents,'cents left over')

metrocard_calculator(20)
```
Question 3: What prints out, approximately, after the following code executes? To simplify this, a 10 X 10 grid is provided so you can fill in one character per square.

def diamond_face():
    print(' '*4+'*')
    print(' '*3+'*'*3)
    print(' '*2+'O'+'O'*2)
    print(' '+*2+'|'*3+'*'*2)
    print('*'*3+'='*3+'*'*3)
    print(' '+'\'+'V'*5+'/')
    print(' '*2+'*'*5)
    print(' '+*3+'*'*3)
    print(' '*4+'*')

diamond_face()

10 X 10 Grid

____ ____ ____ ____ ____ ____ ____ ____ ____ ____
| | | | | | | | | | |
|____|____|____|____|____|____|____|____|____|____|
| | | | | | | | | | |
|____|____|____|____|____|____|____|____|____|____|
| | | | | | | | | | |
|____|____|____|____|____|____|____|____|____|____|
| | | | | | | | | | |
|____|____|____|____|____|____|____|____|____|____|
| | | | | | | | | | |
|____|____|____|____|____|____|____|____|____|____|
| | | | | | | | | | |
|____|____|____|____|____|____|____|____|____|____|
| | | | | | | | | | |
|____|____|____|____|____|____|____|____|____|____|
| | | | | | | | | | |
|____|____|____|____|____|____|____|____|____|____|
| | | | | | | | | | |
|____|____|____|____|____|____|____|____|____|____|
| | | | | | | | | | |
|____|____|____|____|____|____|____|____|____|____|
| | | | | | | | | | |
|____|____|____|____|____|____|____|____|____|____|
| | | | | | | | | | |
|____|____|____|____|____|____|____|____|____|____|
| | | | | | | | | | |
|____|____|____|____|____|____|____|____|____|____|
| | | | | | | | | | |
|____|____|____|____|____|____|____|____|____|____|
| | | | | | | | | | |
|____|____|____|____|____|____|____|____|____|____|
| | | | | | | | | | |
|____|____|____|____|____|____|____|____|____|____|
| | | | | | | | | | |
|____|____|____|____|____|____|____|____|____|____|
| | | | | | | | | | |
|____|____|____|____|____|____|____|____|____|____|
| | | | | | | | | | |
|____|____|____|____|____|____|____|____|____|____|
| | | | | | | | | | |
|____|____|____|____|____|____|____|____|____|____|
Question 4: On this page and continuing on the following page is a Python program. Following the program, there is a table with user answers to the questions asked by the program. For each set of answers, indicate which pet the `classify_your_pet` program returns.

def get_integer(question):
    answer = input(question+'' ')
    if answer.isnumeric():
        return(int(answer))
    elif (answer == 'No') or (answer == 'no') \
         or (answer == 'none') or (answer == 'None'):
        return(0)
    else:
        return(100)

def yes_or_no(question):
    answer = input(question+'' ')
    answer = answer.lower()
    while (answer != 'yes') and (answer != 'no'):
        print('Your response', '('+answer+')', \
             'is invalid. Please only answer yes or no.')
        answer = input(question+'' ')
    if answer == 'yes':
        return(True)
    else:
        return(False)
```python
def classify_your_pet():
    question1 = "How many legs?"
    answer1 = get_integer(question1)
    question2 = "Does it breath in water?"
    answer2 = yes_or_no(question2)
    question3 = "Is it warm blooded?"
    answer3 = yes_or_no(question3)
    question4 = "Does it weigh more than 90 kilograms or 200 pounds?"
    answer4 = yes_or_no(question4)
    question5 = "Does it weigh more than 18 kilograms or 40 pounds?"
    answer5 = yes_or_no(question5)
    question6 = "Can it fly?"
    answer6 = yes_or_no(question6)
    if answer1 >= 10:
        return('centipede')
    elif answer1 == 8:
        if answer2:
            return('octopus')
        else:
            return('spider')
    elif answer1 == 6:
        return('Cockroach')
    elif answer1 == 4:
        if answer3:
            if answer4:
                return('Cow')
            elif answer5:
                return('Dog')
            else:
                return('Rat')
        else:
            return('Lizard')
    elif answer1 == 2:
        if answer6:
            return('Bird')
        else:
            return('Monkey')
    elif answer1 == 0:
        if answer2:
            return('Fish')
        else:
            return('Snake')
    else:
        return('Cat')

classify_your_pet()
```

<table>
<thead>
<tr>
<th>Question 4a</th>
<th>answer1</th>
<th>answer2</th>
<th>answer3</th>
<th>answer4</th>
<th>answer5</th>
<th>answer6</th>
<th>Pet Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>Question 4b</td>
<td>8</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>Question 4c</td>
<td>0</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>Question 4d</td>
<td>2</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>Question 4e</td>
<td>4</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>Question 4f</td>
<td>100</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td></td>
</tr>
</tbody>
</table>
Section 2 (50 points): Answer 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 (only do this if you really have enough time), please indicate which ones you would like to count for the test.

Question 5: Write a function that takes an integer as input. For any integer from 1 to 99999999, it should print out the number of millions, hundred thousands, thousands, hundreds and the remaining amount under one hundred. For example, such a function could have the print out indicated below for the parameter values indicated. It is expected that you will need to use integer division and modulus to figure out the amounts of each increment.

```python
>>> print_number_in_style(999999999)
Too Big
>>> print_number_in_style(99999999)
99 million 9 hundred 99 thousand 9 hundred and 99
>>> print_number_in_style(52002001)
52 million 0 hundred 2 thousand 0 hundred and 1
>>> print_number_in_style(1090929)
1 million 0 hundred 90 thousand 9 hundred and 29
>>> print_number_in_style(1929)
0 million 0 hundred 1 thousand 9 hundred and 29
>>> print_number_in_style(0)
Too Small
```
**Question 6:** Write a function that guesses a user number from 1 to 10. The program should use a while loop that exits when it gets the correct number. The program should use the following strategy:

1. It should initialize a `maximum` value to 10 and a `minimum` value to 1.

2. It should enter a while loop in which it guesses the user’s number and the user is expected to respond as to whether its guesses are 'low', 'high' or 'correct'.

3. Each guess should be the rounded average of the `maximum` and the `minimum`. Note that the function `round` will round a float to the nearest integer (.5 is rounded up).

4. A correct guess, should cause the loop to end and the total number of guesses to be printed out.

5. A 'low' guess should result in the `minimum` to be reset to the current guess + 1.

6. A 'high' guess should result in the `maximum` to be reset to the current guess -1.

A sample run of the program follows. The statements of the form *My guess is X* are the program “guessing” a number. The program repeatedly “asks” *Is this guess correct, high or low?*. The sample user responses immediately follow the questions. In the example, the user’s number is 9 and the program guesses 6, 8, 10 and then finally 9, as directed by the user’s responses and the strategy described above.

```python
>>> guess_number_one_to_ten()
My guess is 6
Is this guess correct, high or low? low
My guess is 8
Is this guess correct, high or low? low
My guess is 10
Is this guess correct, high or low? high
My guess is 9
Is this guess correct, high or low? correct
Your number, is 9
It took me 4 guesses.
```
**Question 7:** Write a function that draws a striped triangle. The function should take three parameters: \( \text{height} \), \( \text{character1} \) and \( \text{character2} \). A sample execution of this program follows. Observe that the first line consists of 9 spaces, followed by one (non-space) character, followed by one space. The second line consists of 8 spaces, followed by (non-space) character, space, (non-space) character, space, and so on until the last line consists of zero spaces, followed by 10 (non-space) characters, with spaces in between. The (non-space) characters on the lines alternate between ‘0’ and ‘@’.

The first line of print out should be made up of instances of \( \text{character1} \) (and spaces); the second line is made up of instances of \( \text{character2} \) (and spaces); the third line uses \( \text{character1} \) again, and so on.

The \( \text{height} \) parameter indicates both the number of lines in the triangle and the number of non-space characters in the base (the last line). The first line of the triangle should have 1 centered non-space character, the second line should have 2 non-space characters, and so on, until the final line which has the maximum number of non-space characters (equal to the \( \text{height} \) parameter).

Each line should consist of a certain number of spaces followed by (non-space) characters with spaces in between them. The number of initial spaces should equal the total height minus the number of (non-space) characters to be printed on that line. One space should be printed after each (non-space) character.

```python
>>> print_striped_triangle(10, 'O', '@')
  0
    @ @
  0 0 0
    @ @ @
  0 0 0 0
    @ @ @ @
  0 0 0 0 0
    @ @ @ @ @
  0 0 0 0 0 0
    @ @ @ @ @ @
  0 0 0 0 0 0 0
    @ @ @ @ @ @ @
  0 0 0 0 0 0 0 0
    @ @ @ @ @ @ @ @
  0 0 0 0 0 0 0 0 0
    @ @ @ @ @ @ @ @ @
```
Glossary for Python 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 multiply 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.
- Indices: an index from 0 to (sequence length + 1) represent positions before and after items in a sequence. For example, in the string duck, the subscripts indicate indexed positions: `0d1u2c3d4`. In square brackets, and index can be used to identify the character that follows that idnex, e.g., the expression `duck[2]` would evaluate to be c, the character after index position 2,
- sequences are used in `for` loops (see below)
- 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', '!', ' ']
  – string.lower() will return an all-lowercase version of a string, e.g., 'CaT'.lower() is equivalent to 'cat'. In the same way, string.upper() converts to uppercase, e.g., 'CaT'.upper() is equivalent to CAT.

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 break. After a break, the statement after the loop executes.
  • 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. while loops
• First line While (BOOLEAN-EXPRESSION):
  • The loop keeps executing the indented body until BOOLEAN-EXPRESSION evaluates as False.
  • If BOOLEAN-EXPRESSION is always True, the loop is endless.
  • Typically BOOLEAN-EXPRESSION contains one or more variable(s), such that some values of these variables (or this variable) cause BOOLEAN-EXPRESSION to evaluate as True and other values cause it to evaluate as False.
  • The body of the loop can change these variables. The loop starts when BOOLEAN-EXPRESSION is True. Then, under most circumstances, BOOLEAN-EXPRESSION eventually evaluates as False and the loop halts. For example, if the BOOLEAN-EXPRESSION is stop == False, setting stop to True inside the loop, will cause the loop to finish.
  • It is common to use accumulator variables in a similar manner as with for loops.
  • break and return behave the same way for while loops as they do with for loops.

8. 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.

9. 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