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.
On the actual midterm, there will only be 4 questions in this section. There are 5 questions on this sample test to enable
more practice.
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 resulted in a small deduction, but answering '2', would have resulted in an incorrect
answer.

**Question 1**

```python
def print_5s():
    print(5,5,5,sep='*',end='**')
    print(5,5,5)

print_5s()
```

**Question:** What prints out by executing the code above (approximately)?

**Answer:**

**Question 2**

```python
def nine_complement(digit):
    return(9 - digit)

def nine_complement_string(number_string):
    output = ''
    for digit in number_string:
        output = output+str(nine_complement(int(digit)))
    return(output)

output = nine_complement_string('0123456789')
```

**Question:** What is the value of the global variable `output`, after the code above executes?

**Answer:**
def guess_a_letter(guessed_letters):
    letter = input('Guess a letter: ')
    letter = letter.lower()
    while (letter in guessed_letters) or (not (letter in 'abcdefghijklmnopqrstuvwxyz')):
        print(letter, 'is not a valid letter.', 'Please try again.')
        letter = input('Guess a letter: ')
    letter = letter.lower()
    return(letter)

def print_body_part_message (number):
    ## full list [Head, left eye, right_eye, mouth, neck, left arm, right arm, torso,
    if number == 1:
        print('Head')
    elif number == 2:
        print('Head, left eye')
    elif number == 3:
        print('Head, left eye, right_eye')
    elif number == 4:
        print('Head, left eye, right_eye, mouth')
    elif number == 5:
        print('Head, left eye, right_eye, mouth, neck')
    elif number == 6:
        print('Head, left eye, right_eye, mouth, neck, left arm')
    elif number == 7:
        print('Head, left eye, right_eye, mouth, neck, left arm, right arm')
    elif number == 8:
        print('Head, left eye, right_eye, mouth, neck, left arm, right arm, torso')
    elif number == 9:
        print('Head, left eye, right_eye, mouth, neck, left arm, right arm, torso, left')
    elif number == 10:
        print('Head, left eye, right_eye, mouth, neck, left arm, right arm, torso, left

def hang_man(word):
    word = word.lower()
    guessed_letters = ''
    incomplete_word = len(word)*'_'
    unguessed_letters = 26
    incorrect_letters = 0
    while incorrect_letters<10:
        letter = guess_a_letter(guessed_letters)
        guessed_letters = guessed_letters+letter
        if letter in word:
            replacement_incomplete_word = ''
            for num in range(len(word)):
                if word[num] == letter:
                    replacement_incomplete_word=replacement_incomplete_word+letter
                else:
                    replacement_incomplete_word=replacement_incomplete_word\
                    +incomplete_word[num]
            incomplete_word = replacement_incomplete_word
        print('Your letter is correct. The word so far is: ',incomplete_word)
else:
    incorrect_letters = incorrect_letters+1
if incomplete_word == word:
    print('You Win')
    return('Hurray')
else:
    print_body_part_message(incorrect_letters)
    print('You Lose')

hang_man('chicken')

Question 3a: What does the function guess_a_letter do with the variable guessed_letters?

Question 3b: What are the 2 possible conditions in which the game ends?

Question 3c: Write down the user input and what the system prints out for a possible game in which the word being guessed is “chicken” and the user guesses at least 3 incorrect letters and at least 3 correct letters.

Question 4

output = 50
    for number in range(6):
        output = output - number

Question: What does output equal after the loop executes?
Answer:
Question 5

def doub_dig(number):
    number_string = str(int(number))
    if len(number_string)<2:
        return('0'+number_string)
    else:
        return(number_string)

def decimal_clock ():
    import time
    ## There are 86400 old fashioned seconds in a day
    ## new_seconds should divide days by a power of 10
    new_second = 86400/100000
    minute = 100  ## a new minute is a hecto-second
    hour = 1000  ## a new hour is a kilo-second
    day = 100000  ## a hecto-hour is a day
    ## lets assume exactly 10 new_months per calendar year
    ## i.e., year is a deca-month
    new_month = (365 * day)/10  ## i.e., 36.5 days
    total_seconds = 0
    current_new_month = 0
    current_new_day = 0
    current_new_hour = 0
    current_new_minute = 0
    current_new_second = 0
    while True:
        print_string=doub_dig(current_new_month)+':'+doub_dig(current_new_day)+':'
        +doub_dig(current_new_hour)+':'+doub_dig(current_new_minute)+':'
        +doub_dig(current_new_second)
        print(print_string)
        time.sleep(new_second)
        total_seconds = total_seconds + 1
        current_new_month = total_seconds//new_month
        current_new_day = total_seconds//day
        current_new_hour = total_seconds//hour
        current_new_minute = total_seconds//minute
        current_new_second = total_seconds%minute
        if new_month == 11:
            new_month = 1  ## reset new_month to 1 when near year starts

decimal_clock()

Question 5a: What does the function doub_dig do?

Question 5b: Once started, how long will decimal_clock run? What will cause it to halt?

Question 5c: Write down the first 10 lines that will print out.

Question 5d: How many days should it take for the following line to print out?
01:00:00:00:00
Section 2: Write Programs as specified. On the actual midterm, you will only be expected to write 2 programs out of 3 possibilities. It is a bad idea to do all the programs. It can make it difficult to finish the test.

Question 6: Write a function called *word_repeater_and_counter*. It should take no parameter arguments. The function should solicit the user to input one word at a time or indicate they are done typing in words. The program should form a new string by concatenation of the input strings, with spaces in between. The program should also count the number of words. When the user is done, the program should print out the sequence of words they typed in and then return the number of words. A sample interaction with such a function is provided below. In this example, an input function uses 'Type next word or "XXX" to indicate you are done.' as a prompt. The user enters the word 'blah' 3 times and then enters XXX to indicate they are finished. The function returns the total number of words (not 'XXX') that the user typed in.

```python
>>> word_repeater_and_counter()
Type next word or "XXX" to indicate you are done. blah
Type next word or "XXX" to indicate you are done. blah
Type next word or "XXX" to indicate you are done. blah
Type next word or "XXX" to indicate you are done. XXX
blah blah blah
3
>>> 
```
Question 7: Write a function called `striped_rectangle` that takes four parameters as arguments: `character1`, `character2`, `width` and `height`. It should draw a striped rectangle consisting entirely of characters 1 and 2. The parameters `width` and `height` should determine the width and height of the rectangle, where the width is measured in characters and the height is measured in lines. The lines should alternate between being entirely of `character1` or entirely of `character2`. Thus if `height` is an odd number there will be one more `character1`-based line than there are `character2`-based lines. The following is an example of how the intended function should behave:

```python
>>> striped_rectangle(‘=’,’*’,15,6)
===============
***************
===============
***************
===============
***************
===============
```

```
Question 8: Write a program that will ask a user several questions about how nice they are. The program will then calculate how nice they are in Standard Niceness Units (SNUs), according to guidelines devised by the World Council on Niceness (WCN). These guidelines are represented in the flowchart below. The flowchart indicates when questions should be asked and how many SNUs are added or subtracted to the total SNU score based on the answers. The questions are given as global variables. You should use these variables instead of writing out all the questions (it will save a lot of time). At your option, you may also use (without writing it yourself), the function yes_or_no. You can vary from the flowchart if it doesn’t matter logically. For example, you could ask the user questions 1, 2, 3, 4, 5 and 6 and use the answers (stored as variables) in the if/else statements of your decision tree if this seems easier to you.

The yes_or_no function and global variables:

```python
def yes_or_no(question):
    answer = ''
    while not (answer in ['Yes','yes','No','no']):
        print('Please answer the following question "yes" or "no". ')
        answer = input(question)
        if answer in ['Yes','yes']:
            return(True)
        else:
            return(False)

question1 = 'Do you always share your candy? ' 
question2 = 'Do you clean up after yourself most of the time? ' 
question3 = 'Do you give money to charity? ' 
question4 = 'Do you spend some of your time doing charitable work? ' 
question5 = 'Do you care about non-human animals? ' 
question6 = 'Would you commit a crime against another person for personal gain? ' 
question1a = 'Do you always share your food? ' 
question2a = 'Do you clean up after other people? ' 
question3a = 'Would you give up all your money to help a friend in serious need? ' 
question3a1 = 'Would you do the same for a stranger? ' 
question4a = 'Would you dedicate your life to a charitable cause and be poor yourself? ' 
question5a = 'Do you believe in eating all non-human animals? ' 
question5a1 = 'Do you believe in eating some non-human animals? ' 
question6a = 'Would you steal so much that your victim becomes poor? ' 
question6b = 'Would you murder another person for profit? '
```

Some hints for following the flow chart: questions 1,2,3,4,5,6 are mandatory. Regardless of whether question 1 is answered with 'yes' or 'no', question 2 must be asked. Regardless of whether question 2 is answered 'yes' or 'no', question 3 must be asked, etc. However, other questions are only asked under certain circumstances. In most cases a positive answer yields more questions, but not always. Usually, positive answers cause points to be added to the SNU (total Standard Niceness Units), but occasionally, positive answers cause points to be subtracted. The flow chart is on the following page.

For extra credit, you can complicate the decision tree further by adding five additional questions. Only add questions that would be asked if previous questions are answered No.
Initialize SNU to 0

Start

Ask Question1

Yes

Add 10 to SNU

Ask Question 1a

Yes

Ask Question2

Yes

Add 10 to SNU

Ask Question 2a

Yes

Ask Question3

Yes

Add 10 to SNU

Ask Question 3a

Yes

Add 10 to SNU

Ask Question 3a1

Yes

No

Add 10 to SNU

Ask Question 3a

No

Ask Question4

Yes

Add 10 to SNU

Ask Question 4a

Yes

Yes

Add 10 to SNU

Ask Question 4a

No

Add 10 to SNU

Ask Question 5a

Yes

Yes

Subtract 3 from SNU

Ask Question 5a

No

Yes

Subtract 30 from SNU

Ask Question 5a

No

Ask Question5

Yes

Add 10 to SNU

Ask Question 6

No

Ask Question6

Yes

Subtract 30 from SNU

Ask Question 6a

Yes

Subtract 50 from SNU

Ask Question 6b

Yes

Print Results

Return Total Score
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 multiple two numbers or repeat 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: _0d_1_u_2c_3_d_4_. 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* – this also exits 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