There are 2 sections, the first section is worth 50 points and the second 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 lost just a little bit, but answering 2, would have resulted in an incorrect answer.

Question 1

```
def print_stars_and_stripes(stars,stripes):
    for num in range(5):
        print((stars*' '*), (stripes*'|'))

print_stars_and_stripes(5,5)
```

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

Answer:
Question 2

def multiply_digits_in_string (input_string):
    output = 1
    for digit in input_string:
        output = output * int(digit)
    return(output)

def add_digits_in_string (input_string):
    output = 0
    for digit in input_string:
        output = output + int(digit)
    return(output)

secret_code = ’010010’

product = multiply_digits_in_string(secret_code)
sum = add_digits_in_string(secret_code)
answer = sum * product

Question: After the above code executes, what are the values of the three global variables: product, sum and answer?
Answer:
Question 3

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

decimal_clock()

Question 3a: What does the function doub_dig do?

Question 3b: Once started, how long will decimal_clock run? What will cause it to halt?

Question 3c: Write down the first 10 lines that will print out.

Question 3d: How many days should it take for the following line to print out?
01:00:00:00:00
Question 4

def print_picture2():
    print(8*' '+'(')
    print(6*' '+'4*' o')
    print(5*' '+'6*' o')
    print(4*' '+'<@>'+2*' '='+'<@>')
    print(3*' '+4*' o'+' /\ '+4*' o')
    print(6*' '+'o'+' ^'+4*' o')
    print(5*' '+'6*' o')
    print(6*' '+'VVVV')

print_picture2()

Question: What prints out by executing the code above (approximately)?
def integer_check (input_string):
    if input_string == '':
        return(False)
    else:
        for char in input_string:
            if not ((char == '0') or (char == '1') or (char == '2') or (char == '3')
                or (char == '4') or (char == '5') or (char == '6') or (char == '7')
                or (char == '8') or (char == '9')):
                return(False)
        return(True)

def get_integer_input():
    query_string = 'Please enter an integer. '
    correction_string = 'is not an integer. Integers consist only of digits. Try again.'
    answer = input(query_string)
    while not integer_check(answer):
        print(answer,correction_string)
        answer = input(query_string)
    return(int(answer))

def yes_or_no():
    yes_or_no_string = 'Please answer yes or no. '
    correction_string = 'is not "yes" and it is not "no". Try again.'
    answer = input(yes_or_no_string)
    while (answer != 'yes') and (answer != 'no'):
        print(answer,correction_string)
        answer = input(yes_or_no_string)
    if answer == 'yes':
        return(True)
    else:
        return(False)

def sorting_hat():
    print('What is your shoe size? ')
    shoe_size = get_integer_input()
    print('Do you wear glasses?')
    wear_glasses = yes_or_no()
    print('Do you like Bologna sandwiches?')
    bologna_affinity = yes_or_no()
    if wear_glasses:
        return('Gryffindor')
    elif bologna_affinity:
        return('Slytherin')
    elif shoe_size >= 9:
        return('Ravenclaw')
    else:
        return('Hufflepuff')

print('Sorting will now begin. Please answer some questions.')
Final_Sorting = sorting_hat()
print('You are sorted into:',Final_Sorting)
Question 5 continued

Question: Walk through an execution of the above program in which you are the user. Indicate everything that prints to the screen, including your answers to questions. Purposely make 2 mistakes in your initial input so that the functions get_integer_input and yes_or_no each print out their instances of the local variable named correction_string. For clarity, you can number the output statements (but that is not required).
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 6:

Write a Deluxe Marquis Sentence Display program, that creates a boundary around a sentence in asterisks like this:

```
*************************************************** *************
*************************************************** *************
*************** This is a sentence ***************
*************************************************** *************
*************************************************** *************
```

If the input sentence is longer than 30 characters, the program should print out the message: “Your input sentence is too long for the Deluxe Marquis Sentence Display program” and the program should halt. Otherwise, the program should proceed with the following steps:

1. The program should print 2 lines consisting of 60 asterisks.

2. The next line should be 60 characters long including: 15 asterisks at the beginning; 30 characters consisting of the input sentence, centered between spaces, e.g., if the sentence is just the string ’XX’, then the line should consist of 15 asterisks, followed by 14 spaces, followed by ’XX’, followed by 14 spaces, followed by 15 asterisks. For odd length strings, the number of spaces preceding and following the string should be one different from each other, e.g., for ’XXX’, there can be 13 spaces before and 14 spaces after.

3. The next 2 lines should each consist of 60 asterisks.
Question 7:

Many people estimate that one year of a dog’s life is equivalent to seven years of a human’s life, e.g., a seven year old dog is like a one year old human and a 10 year old dog is like a 65 year old human. However, there are some problems with this estimate. For example, a one year old dog is of breeding age, but a 7 year old child is not. Furthermore, dogs have lived to almost 30 years old, whereas only a handful of human beings have ever lived past the age of 114 (not 210).

Write a program that estimates a dog’s age by accumulating different numbers of years for each year of a dog’s life, based on the table below. In the table, years of a dogs life are associated with value in equivalent human years, e.g., the first year of a dog’s life is equivalent to 14 human years; each year from the second through the fifth year is equivalent to 6.5 human years, etc.

<table>
<thead>
<tr>
<th>Chronological Years</th>
<th>Human Year Equivalence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>2–5</td>
<td>6.5</td>
</tr>
<tr>
<td>6–10</td>
<td>5</td>
</tr>
<tr>
<td>11–20</td>
<td>4</td>
</tr>
<tr>
<td>21–30</td>
<td>1</td>
</tr>
</tbody>
</table>
Question 8: Write a game program that tries to guess a number from 1 to 1000 picked by the player in less than 10 tries. Use a for loop to ensure that the computer has at most 10 tries. If the computer guesses correctly in less than 10 tries, a return will exit the for loop.

The player is instructed to pick a number from 1 to 1000. The computer will guess numbers between 1 and 1000 and the player will answer either: High if the computer’s guess is too high; Low if the computer’s guess is too low or Correct if the computer’s guess is correct. The game ends when either: (1) the for loop ends and the computer has not guessed the right answer; or (2) the computer guesses correctly.

The program should always guess the midpoint between the highest and lowest possible numbers (rounded to the nearest whole number). Thus the computer’s first guess will always be 500 or 501 (depending on how you are rounding). When the player tells the program that a guess is high, the program should reset its high to one less than its current guess. When the player tells the program that a guess is low, the program should reset its low to one more than the current guess. Thus guessing the average between high and low will always yield a new guess in the right direction.

Below is a sample game output in which the program won in 10 rounds.

```python
>>> high_low()
I will guess 500. Please answer ‘correct’, ‘high’ or ‘low’ high
I will guess 250. Please answer ‘correct’, ‘high’ or ‘low’ high
I will guess 125. Please answer ‘correct’, ‘high’ or ‘low’ high
I will guess 62. Please answer ‘correct’, ‘high’ or ‘low’ high
I will guess 31. Please answer ‘correct’, ‘high’ or ‘low’ low
I will guess 46. Please answer ‘correct’, ‘high’ or ‘low’ low
I will guess 54. Please answer ‘correct’, ‘high’ or ‘low’ low
I will guess 58. Please answer ‘correct’, ‘high’ or ‘low’ high
I will guess 56. Please answer ‘correct’, ‘high’ or ‘low’ low
I won in 10 turns.
```

Bonus part of question (5 points). Write a new version of this game where:

(1) The computer gets only 7 tries.

(2) The computer guesses two numbers which divide the possibilities in thirds (rounded off). For example, the first guess will be 333 and 667. We will call the lower guess low number (333) and the higher guess high number: (667). Subsequent guesses will similarly divide the set of possible numbers into 3 parts (before, after and between).

(3) The player can answer: (A) higher than high number; (B) lower than low number; (C) between the two numbers; (D) low number is correct; (E) high number is correct. Note that it is possible for high number and low number to be equal.

(4) The game ends when either one of the two numbers is correct (the computer wins) or the computer does not guess the right answer in the allowed 7 tries.
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)

- 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', 'l', '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.