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:

\[
\text{output} = '1'+'1'
\]

**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
def camouflage_addition(number1, number2, number3):
    sum_of_number = number1 + number2
    hidden_answer = str(number3) + str(sum_of_number) + str(number3)
    print('The sum of', number1, 'and', number2, 'is hidden inside', hidden_answer)
    return(sum_of_number)

answer = camouflage_addition(20, 25, 999999)
```

**Question 1a:** What prints out when the above code is executed?

**Question 1b:** What is the value of the global variable `answer` after the above code is executed?
Question 2

def print_the_two_thing():
    print('2'*9)
    print((' '*8)+'22')
    print((' '*8)+'22')
    print((' '*8)+'22')
    print((' '*8)+'22')
    print((' '*7)+'22')
    print((' '*6)+'22')
    print((' '*5)+'22')
    print((' '*4)+'22')
    print((' '*3)+'22')
    print((' '*2)+'22')
    print(' '+'22')
    print('22')
    print('2'*10)

print_the_two_thing()

* * * * * * * * * * * * * * * * * * * * * * * * * *
|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|

* * * * * * * * * * * * * * * * * * * * * * * * * *

Draw approximately what the above code causes to print out. A 14 by 14 grid is provided, to make this easier.
def flock_of_zs(height):
    import math
    half_height = math.ceil(height/2)
    for num in range(height):
        if num < half_height:
            space_num = num
            num_of_zs = num+1
        else:
            num_of_zs = height-num
            space_num = height-(num+1)
        print((space_num*' ')+(num_of_zs*'z'))

flock_of_zs(7)

Draw approximately what the above code causes to print out. A 14 by 14 grid is provided, to make this easier.
Question 4

def ask_question(question):
    answer = input(question+' ')
    while (not ((answer == 'Yes') or (answer == 'yes') or 
        (answer == 'y') or (answer == 'Y') or
        (answer == 'No') or (answer == 'no') or
        (answer == 'n') or (answer == 'N'))):
        answer = input("Please answer: yes or no")
    if (answer == 'Yes') or (answer == 'yes') or 
       (answer == 'y') or (answer == 'Y'):
        return(True)
    else:
        return(False)

def guess_the_animal():
    question1 = "Does it have fur?"
    question2 = "Does it have a curly tail?"
    question3 = "Does it have hooves?"
    question4 = "Does it have fins?"
    answer1 = ask_question(question1)
    answer2 = ask_question(question2)
    answer3 = ask_question(question3)
    answer4 = ask_question(question4)
    if answer1:
        if answer2:
            if answer3:
                print(‘Pig’)
            else:
                print(‘Dog’)
        elif answer4:
            print(‘Whale’)
        else:
            print(‘Sea Horse’)
    elif answer2:
        print(‘northern curly-tailed lizard’)
    elif answer3:
        print(‘Not possible’)
    elif answer4:
        print(‘Fish’)
    else:
        print(‘Worm’)

def question_4():
    guess_the_animal()

Question 4a: What will the program print out if the user answers 'yes' to questions 1, 3 and 4 and answers 'no' to question 2 (the animal supposedly has fur, hooves and fins and does not have a curly tail)?

Question 4b: What will print out if the user answers 'yes' to questions 2 and 3, but 'no' to questions 1 and 4 (the animal has a curly tail and hooves, but no fur or fins)?

Question 4c: What will print out if the user answers 'yes' to questions 1, 2 and 3, but 'no' to question 4 (the animal has fur, a curly tail and hooves, but no fins)?
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 adds up numbers entered by a user. The function should be called `add_them_up` and should not take any parameter arguments. Rather it should solicit numbers from a user one at a time using the `input` function inside of a `while` loop. The `input` statement should solicit either a number or the word “Done”. If the user enters “Done”, the loop should exit, whereas if the user enters anything else, the program should convert that input into a number and add that number to an accumulator variable called `total`, which you should initialize before the loop. The function should return `total`. A sample interaction with such a function is provided below. In this example, an input function uses ‘Give me a number or type “Done” ’ as a prompt. The user enters 3 different numbers before typing “Done” to indicate that they are finished. The function returns the sum of the numbers entered.

```python
>>> add_them_up()
Give me a number or type "Done" 7
Give me a number or type "Done" 10
Give me a number or type "Done" 11.5
Give me a number or type "Done" Done
28.5
>>> 
```
Question 6: Write a decision-tree-based interactive fiction program, based on the flowchart in Figure 1. You do not have to type in the various strings into your program. Assume that your program has access to the variables defined below.

```
duck1 = 'The friendly duck was swimming along'
duck2 = 'He is getting hungry'
duck3 = '''The duck goes off to take a CS midterm. He can choose 2 out of 3 questions on Part 2.'''
duck4 = 'He finishes the exam and gets an A−'
duck5 = '''He realizes that the test is really about him. It all makes sense now. He gets an A on the test.'''
duck6 = 'He faints and wakes up the next day'

duck_question1 = 'Should he dive to eat a fish?'
duck_question2 = '''Should he find a clam break its shell on a rock and eat it?'''
duck_question3 = '''Should he fly over to the path by the lake and eat potato chips off the ground?'''
duck_question4 = '''Should he choose the question about the duck?'''
```

Figure 1: Flowchart for Duck Story
**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 these *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 consist of alternations between *character1* and *character2*. Each line should begin with *character1*, i.e., odd-number-length lines will have one more instance of *character1* than *character2*. The following is an example of how the intended function should behave:

```python
>>> striped_rectangle(’=’,’*’,15,7)
=*=*=*=*=*=*=*=*=*=*=*=*=*
=*=*=*=*=*=*=*=*=*=*=*=*=*
=*=*=*=*=*=*=*=*=*=*=*=*=*
=*=*=*=*=*=*=*=*=*=*=*=*=*
=*=*=*=*=*=*=*=*=*=*=*=*=*
=*=*=*=*=*=*=*=*=*=*=*=*=*
=*=*=*=*=*=*=*=*=*=*=*=*=*
>>> 
```
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)
- 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 ”

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.