Below are sample questions to help you prepare for Exam #2. Make sure you can solve all of these problems by hand. For most of the questions, you can check your answers by typing in the programs and seeing what happens on the computer.

1. Consider the following program.

Listing 1: functions.py

```python
n = 10

def foo(x):
    n = 3 + x
    x = n * 2
    return n, x

def bar(m):
    n = 4 + m
    a, b = foo(n)
    print 'The result in bar is %d.' % (a + b)

bar(10)
print 'Goodbye!'
```

a) List all of the keywords in the program.
b) Write the order in which the lines in the program are executed?

c) What is the output?

d) For each function, list each of its local variables and the line number(s) in which the local variables appear.

e) List all of the global variables that appear in the program and the line numbers in which they appear.
2. Consider the following Python program.

Listing 2: sets.py

```python
def f1(s1, s2, op):
    if op == 'i':
        result = s1.intersection(s2)
    elif op == 'u':
        result = s1.union(s2)
    elif op == 'd':
        result = s1.difference(s2)
    else:
        result = s1.issubset(s2)
    return result

def main():
    set1 = set(['apple', 5, 'banana', 7])
    set2 = set(['banana', 5])
    print len(set2)  # Line A
    print f1(set1, set2, 'i')  # Line B
    print f1(set2, set1, 'i')  # Line C
    print f1(set1, set2, 'u')  # Line D
    print f1(set2, set1, 'u')  # Line E
    print f1(set1, set2, 'd')  # Line F
    print f1(set2, set1, 'd')  # Line G
    print f1(set1, set2, 'p')  # Line H
    print f1(set2, set1, 'p')  # Line I

main()
```

a) What output does Line A produce?
b) What output does Line B produce?
c) What output does Line C produce?
d) What output does Line D produce?
e) What output does Line E produce?
f) What output does Line F produce?
g) What output does Line G produce?
h) What output does Line H produce?
i) What output does Line I produce?
3. Consider the following Python program.

Listing 3: myFun.py

```python
def myFun(param1, param2):
    result = ""
    inc = param1
    while param1 < len(param2):
        result = result + param2[param1]
        param1 += inc
    return result

def main():
    raw_inputs = raw_input("Give me an int and a string: ")
    inputs = raw_inputs.strip()
    int_arg, str_arg = inputs.split(‘ ‘)
    int_arg = int(int_arg)
    print int_arg
    print str_arg
    print myFun(int_arg, str_arg)
main()
```

a) What is the output by Line C if the user inputs “1 Aggies” (without the quotes) when prompted?
b) What is the output by Line D if the user inputs “1 Aggies” (without the quotes) when prompted?
c) What is the output by Line E if the user inputs “1 Aggies” (without the quotes) when prompted?
d) What is the output by Line E if the user inputs “4 Aggies” (without the quotes) when prompted?
e) What is the output by Line E if the user inputs “0 Aggies” (without the quotes) when prompted?
f) If the user inputs “0 Aggies” (without the quotes) when prompted, what is the effect on the program if the < symbol in Line A is changed to <=? Explain.
g) If the user inputs “0 Aggies” (without the quotes) when prompted, what is the resulting output if Line B is changed to the following?

```python
result = result + param2[param1]
```
h) Give a brief description of what the above program does?
4. Consider the following Python program.

Listing 4: list-comprehension.py

```python
fruits = ['cherry', 'pecan', 'apple']
print [fruit + ' pie' for fruit in fruits]  # Line A
vec = [2, 4, 6]
print [3*x for x in vec]  # Line B
print [3*x for x in vec if x > 3]  # Line C
print [[x,x**2] for x in vec]  # Line D
vec1 = [2, 4, 6]
vec2 = [4, 3, -9]
print [x*y for x in vec1 for y in vec2]  # Line E
print [vec1[i]*vec2[i] for i in range(len(vec1))]  # Line F
```

a) What is the output of Line A?

b) What is the output of Line B?

c) What is the output of Line C?

d) What is the output of Line D?

e) What is the output of Line E?

f) What is the output of Line F?

5. Consider the following Python program.

Listing 5: advanced-functions.py

```python
def alpha(x,y):
    return x + beta(y,x)
def beta(x,y):
    return y - x
print alpha(10,4)  # Line A
print alpha(beta(3,8), alpha(3,1))  # Line B
```

a) What output does Line A produce?

b) What output does Line B produce?
6. Consider the following program that uses the matplotlib module.

```python
import matplotlib.pyplot as plot

# Line A
def draw_plot(x_axis, y_axis, x_ticks, y_ticks):
    plot.plot(x_axis, y_axis, marker='o', markersize=7.0,
              label='a simple line')
    plot.xlabel('x axis')
    plot.ylabel('y axis')
    plot.title('a simple plot')
    plot.xticks(x_ticks)
    plot.yticks(y_ticks)
    plot.legend()
    plot.grid(True)
    plot.show()

def main():
    step = 1
    # Line B
    x_list = range(5)
    y_list = [3, 2, 5, 0, 2]
    xticks_list = [0, 1, 2, 3, 4]
    yticks_list = range(min(y_list), max(y_list) + 2, step)
    print yticks_list
    draw_plot(x_list, y_list, xticks_list, yticks_list)
    # Line C

main()
```

a) What output does Line B produce?

b) What output does Line C produce? Yep, I want you to be able to draw the plot by hand.

c) What is the effect on the program if Line A is changed to `step = 2`. Be sure to show the resulting output.

7. Write a program to check whether a given solution is a valid $n \times n$ magic square. A magic square of order $n$ is an arrangement of $n^2$ numbers, usually distinct integers, in a square, such that the $n$ numbers in all rows, all columns, and both diagonals sum to the same constant. A normal magic square contains the integers from 1 to $n^2$.

Below, are a few examples of magic squares.

- **Example #1.** A magic square of order 3 where all rows, columns, and both diagonals sum to 15.

  \[
  \begin{array}{ccc}
  2 & 7 & 6 \\
  9 & 5 & 1 \\
  4 & 3 & 8 \\
  \end{array}
  \]

- **Example #2.** A magic square of order 4 where all rows, columns, and both diagonals sum to 34.

  \[
  \begin{array}{cccc}
  4 & 14 & 15 & 1 \\
  9 & 7 & 6 & 12 \\
  5 & 11 & 10 & 8 \\
  16 & 2 & 3 & 13 \\
  \end{array}
  \]
Sample Output #1.

Order of magic square: 3
Enter numbers for Row 0 (separated by spaces): 2 7 6
Enter numbers for Row 1 (separated by spaces): 9 5 1
Enter numbers for Row 2 (separated by spaces): 4 3 8
Valid solution.

Sample Output #2.

Order of magic square: 4
Enter numbers for Row 0 (separated by spaces): 7 12 1 14
Enter numbers for Row 1 (separated by spaces): 2 8 13 11
Enter numbers for Row 2 (separated by spaces): 16 3 10 5
Enter numbers for Row 3 (separated by spaces): 9 6 15 4
Invalid solution.

I’ve given you the main(), check_solution(), check_digits(), and split_on_whitespace() functions. You are to complete the following functions: check_rows(), check_cols(), and check_diagonals(). There is no code in these functions except the pass statement, which is used as a placeholder. If you need more functions to complete your program, that’s fine. But, at a minimum, you need to write the three functions specified.

def check_solution(puzzle, magic_sum, order):
    ''' Check to make sure the digits in the puzzle are unique. '''
    if check_digits(puzzle, order) == False:
        return False
    ''' Check if rows are valid. '''
    if check_rows(puzzle, magic_sum, order) == False:
        return False
    ''' Check if columns are valid. '''
    if check_cols(puzzle, magic_sum, order) == False:
        return False
    ''' Finally, check if diagonals are valid. '''
    if check_diagonals(puzzle, magic_sum, order) == False:
        return False
    ''' Since all 3 checks report the puzzle is valid, then 
    the magic square solution is valid. '''
    return True

def check_digits(puzzle, order):
    digits_set = set()
    for row in puzzle:
        for digit in row:
            digits_set.add(digit)
    if len(digits_set) == order ** 2:
        return True
    else:
        return False
def check_rows(puzzle, magic_sum, order):
    ''' Checks each row in the puzzle to make sure it is valid.'''
    pass

def check_cols(puzzle, magic_sum, order):
    ''' Checks each col in the puzzle to make sure it is valid.'''
    pass

def check_diagonals(puzzle, magic_sum, order):
    ''' Check the two diagonals.'''
    pass

def split_on_whitespace(string_data):
    integers_list = []
    number = ''
    for digit in string_data:
        if digit != ' ':
            number += digit
        else:
            integers_list += [int(number)]
            number = ''
    if digit != ' ':
        integers_list += [int(number)]
    return integers_list

def main():
    ''' The magic square will be stored as a list of lists.'''
    magic_square = []
    order = int(raw_input('Size of magic square: '))
    for i in range(order):
        prompt = 'Enter numbers for Row ' + str(i) + ' (separated by spaces): '
        magic_square += [split_on_whitespace(raw_input(prompt))]
        magic_sum = sum(magic_square[0])
    if check_solution(magic_square, magic_sum, order) == True:
        print 'Valid solution.'
    else:
        print 'Invalid solution.'
main()