Your name:______________________________________

If you don't know the answer to a question, *ask your instructor for help*.

1.

<table>
<tr>
<td>

1. Consider the code shown to the right. The code will produce an error message ("crash") when it runs.

Why does the program crash?

Will PyCharm identify the error even before the code runs?

Is this a ***syntax*** or ***semantic*** error? (Circle your choice.)

</td>
<td>

```
def main():
    cat(4, 10)
    print(c)


def cat(a, b):
    c = a + b


main()
```

</td>
</tr>
</table>

<table>
<tr>
<td>

2. Consider the code shown to the right. It is a contrived example with poor style, but it will run without errors.

What does it print when the function named ***one*** runs?

Write your answer in the box to the right of the code.

</td>
<td>

```
def one():
    a = 4
    b = 10
    c = two(a, b)
    print(a, b, c)


def two(b, a):
    print(a, b)
    a = 100
    b = 200
    return a + b
```

</td>
<td>Output:</td>
</tr>
</table>

3. The specification of a function tells which things?
   Mark all that apply.

_____ Any side effects of the function   _____ What goes in

_____ How the function works   _____ What comes out

4.

<table>
<tr>
<td>Consider the code in the next column.

In the third column, show what the code prints when it runs.

Your instructor will show you how to use the 4th column.</td>
<td><pre>
size = 10
for j in range(3):
    size = size + 5
    print(j, size)
    size = size - j
    print(size)
</pre></td>
<td>Output:</td>
<td><pre>
j   size
    10
0   15
    15
</pre></td>
</tr>
</table>

5. How many integers are there from ***3*** to ***8***, inclusive
(that is, including both the ***3*** and the ***8***)?

6. How many integers are there from ***3*** to ***b***, inclusive
(that is, including both the ***3*** and the ***b***?

7. How many integers are there from ***a*** to ***b***, inclusive
(that is, including both the ***a*** and the ***b***?

8. Fill in the blanks below to complete the Accumulator pattern that implements the function ***sum_many*** that takes two arguments, ***m*** and ***n*** (with ***m <= n***), and returns the sum of the squares of the integers from ***m*** to ***n***, inclusive. For example,

`sum_many(3, 6) returns (3 * 3) + (4 * 4) + (5 * 5) + (6 * 6),` which is `86.`

In this and ALL problems through Exam 1, you are forbidden from using the multiple-argument form of the RANGE expression. That is, `range(a)` is OK but NOT `range(a, b)` or `range(a, b, c)`.

```
def sum_many(m, n):

    total = _______________

    for k in range(_________________________________________):

        total = total + _____________________________________

    ___________________________
```

9. [**Do this problem with your instructor. Don't do the remaining problems until you have done this one.**]

   Suppose that your module contains a function, ***sum_of_digits(number)***, described below. Assume that it has been implemented correctly (per the specification in its doc-string):

```
def sum_of_digits(number):
    """
    What comes in : A non-negative integer.
    What goes out: Returns the sum of the digits in the given integer.
    Example: If the integer is 83135, this function returns
             (8 + 3 + 1 + 3 + 5), which is 20.
    """
    <code hidden>
```

In the box below, implement a second function, ***product_of_sums_of_digits(x, y)***, per the specification in its doc-string. **Hint**: ***reuse*** ***sum_of_digits*** by ***calling*** it in your answer.

In general: *reuse* functions you or someone else wrote by ***calling*** them.

```
def product_of_sums_of_digits(x, y):
    """
    What comes in : Non-negative integers x and y.
    What goes out: Returns (the sum of the digits of x) times
                           (the sum of the digits of y).
    Example: If x is 12 and y is 501, this function returns 3 * 6,
             which is 18.
    """
```

10. Fill in the blanks below to complete the Accumulator pattern that implements the function ***sum_many_digits*** that takes a non-negative integer ***upper_bound*** and returns the sum of the sum-of-digits of the integers from ***0*** to ***upper_bound***, inclusive. For example,

    `sum_many_digits(12)` returns
    `0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 1 + 2 + 3,` which is `51.`

    Hint: ***Reuse the sum_of_digits function from the previous problem!*** That is, ***call*** ***`sum_of_digits`*** as part of your solution to this ***sum_many_digits*** problem.

    In this and ALL problems through Exam 1, you are forbidden from using the multiple-argument form of the RANGE expression. That is, `range(a)` is OK but NOT `range(a, b)` or `range(a, b, c)`.

```
def sum_many_digits(upper_bound):

    total = _______________

    for k in range(_________________________________________):

        total = total + _____________________________________

    ___________________________
```

11. Finally, implement a function ***more_sum_many_digits*** that takes two non-negative integers ***lowerr_bound*** and ***upper_bound*** and returns the sum of the sum-of-digits of the integers from ***lower_bound*** to ***upper_bound***, inclusive.

    Hint: ***Reuse the function from the previous problem!*** This problem is SHORT and EASY, once you see the idea. It can be done with a SINGLE line of code!

```
def more_sum_many_digits(lower_bound, upper_bound):
```