Exam 2 – Paper and Pencil part (Fall, 2020-21)

Name: _____ SOLUTION and RUBRIC _____ Section: _____

Rules and Expectations *Sections 1 and 2 for Mutchler. Section 3 for Noureddine. Section 4 for Ebrahimi.*

At the beginning of this exam, you will receive the **Expectations about Academic Integrity** for this exam -- it is the same as what you were given to read previously. Re-read that document as needed. ***Sign it and turn it in when you finish this exam (both parts).***

Two parts (this is Part 1, Paper-and-Pencil)

For this part, the ONLY external resource you may use is *two* 8½ by 11-inch sheets of paper (or ***one double-sided sheet***), with whatever you want on it, typed or handwritten or a combination of the two. You must have prepared the sheets *before* beginning this exam. You may also use a calculator if you like (but only for calculating).

Problem	Points Possible	Points Earned	Comments
1	7		
2	3		
3	10		
4	10		
5	8		
6	12		
Total (of 100 on the exam)	50		

Communication

For both parts of the exam, ***you must not communicate with anyone*** except your instructors and their assistants, if any. In particular:

- You must not talk with anyone else or exchange information with them during this exam.
- ***After*** this exam, **you must not talk about the exam with anyone who has not yet taken it.**

Do NOT use email, *chat* or the like during this exam. **Close any such applications now.**

1. Consider a function named ***do_it*** that takes a ***list of integers*** as its sole argument.
   For each of the following possible specifications for what ***do_it*** returns:

   Circle ***Yes*** if the code for ***do_it*** would require a loop.

   Circle ***No*** if the code for ***do_it*** would NOT require a loop.

   Rubric: 1 point each.

   If ***do_it*** returns:

   a. The number of odd integers in the list. ***Yes*** **No**

   b. The average of the integers in the list. ***Yes*** **No**

   c. The number of integers in the list. **Yes** ***No***

   d. ***True*** if the first integer in the list is even, else ***False***. **Yes** ***No***

   e. ***True*** if the list contains no even integers, else ***False***. ***Yes*** **No**

   f. The second smallest integer in the list. ***Yes*** **No**

   g. The last integer in the list. **Yes** ***No***

2. Consider a function named ***middle*** that takes a single argument:
   a sequence of numbers, where the length of the sequence is *guaranteed* to be odd.

   The function returns the number at the middle of the sequence. For example:

   - `middle( [90, 10, 45] )` returns **10**
   - `middle( [4, 1, 6, 18, 10, 12, 21] )` returns **18**
   - `middle( [33] )` returns **33**

   Write (in the box below) a complete implementation of the above ***middle*** function.

```
def middle(numbers):
    return numbers[len(numbers) // 2]
```

Rubric (3 points):

- ***OK*** if they use `/` instead of `//`
- OK if they use a more complicated formula for the index as long as it is correct for sequences whose length is odd.

Subtract points as shown to the right, but do not subtract more than 3 points (i.e., no negative scores).

For the ***index***, subtract:

- 1 point if the index is close but wrong
- 2 points if the index is not even close

For ***numbers[...]***, subtract:

- 1 point if parentheses instead of square brackets
- 2 points if the use of *numbers* is even more wrong

Also:

- Subtract ½ point if ***return*** is missing.
- Subtract 1 point if there is a loop, even if the code is correct.

3. Consider the code snippet below. It is a contrived example with poor style, but it will run without errors. What does it print when it runs? Write your answer in the box to the right of the code. Show your work by making notations in the code or by using the empty space below or on another sheet of paper, as desired.

```
def home():
    x = bear(3, 8)
    print("Home A:", x)
    y = bear(ant(4), ant(1))
    print("Home B:", y)


def ant(x):
    print("Ant:", x)
    x = x + 5
    return x


def bear(x, y):
    print("Bear A:", x, y)
    r = ant(x + 4)
    print("Bear B:", r)
    return ant(r + y)
    print(ant(100))
    return 33


print("OK:", ant(101) % 100)
home()
```

In grading this, try to give full or almost full credit if there is a slip somewhere but subsequent parts of the problem make it clear that the student understands the concept being assessed. For example, if the student gets `Home A: 25` correct, the student has earned most of the ...

Output:

```
Ant: 101

OK: 6

Bear A: 3 8
Ant: 7

Bear B: 12

Ant: 20

Home A: 25

Ant: 4
Ant: 1

Bear A: 9 6
Ant: 13

Bear B: 18
Ant: 24

Home B: 29
```

… first 6 points from that alone.

Rubric (10 points):

Each "clump" to the left is 1 point.

- Ignore wrong colons, misspellings, spaces, etc.
- If they omitted all or part of the leading phrase (e.g. "Bear" instead of "Bear A"), try to give full credit if the number(s) suggest that they just had a "slip of the pen.".

Within a clump, if the lines are in the wrong order, subtract ½ point for each such error.

For whole clumps in the wrong order, generally subtract:

- 1 point for 1 error
- 2 points for 2 errors
- 3 points for 3 errors
- 4 points for 4 or more errors.

For extra lines, subtract ½ point for one extra line, 1 point for 2 extra lines, 2 points for 3 or 4 extra lines, and 3 points for more than four extra lines.

Don't subtract more than 10 points (no negative scores). If this rubric is unduly harsh for their answer, assign a "holistic" score as you see fit.

4. Consider the code on the next page. It is a contrived example with poor style but will run without errors. In this problem, you will trace the execution of the code. As each location is encountered during the run: *[See next page for grading rubric]*

   - ***CIRCLE* each variable** that is ***defined*** at that location.
   - ***WRITE*** the ***VALUE*** of each variable that you ***circled*** directly ***BELOW*** the circle.

	a	b	x	self.a	self.b	bear.a	bear.b	cat.a	cat.b	dog.a	dog.b
Location 1 *(1st time)*	31		8	14	80						
Location 1 *(2nd time)*	31		6	5	60						
Location 2 *(1st time)*	1			14	80	5	60				
Location 2 *(2nd time)*	2			5	63	5	63				
Location 3 *(1st time)*	2			114	80	5	63				
Location 3 *(2nd time)*	3			105	66	105	66				
Location 4		1									
Location 5	10	8						14	80		
Location 6	10	8						14	80	5	60
Location 7	10	8	85					114	80	5	63
Location 8	171	8	85					114	80	105	66

```
class Animal():
    def __init__(self, a, x):
        self.a = a + 4
        self.b = x * 10
        a = 31
        #### ---  Location 1  ---

    def zoo(self, a, bear):
        #### ---  Location 2  ---
        a = a + 1
        self.a = self.a + 100
        bear.b = bear.b + 3
        #### ---  Location 3  ---
        return self.b + bear.a


def beach(b):
    #### ---  Location 4  ---
    return Animal(b, b + 5)


def main():
    a = 10
    b = 8
    cat = Animal(a, b)
    #### ---  Location 5  ---
    dog = beach(1)
    #### ---  Location 6  ---
    x = cat.zoo(1, dog)
    #### ---  Location 7  ---
    a = dog.zoo(2, dog)
    #### ---  Location 8  ---

main()
```

Rubric (10 points):

There are 11 x 11 (i.e., 121) cells in the table on the previous page.

Subtract ¼ point for each cell that is wrong, where a cell is wrong if it has something when it should not, or is empty when it should have something, or has the wrong number.

Round up to the nearest half-point, e.g., 3 cells wrong is 9.5.

Don't subtract more than 10 points (no negative scores). If this rubric is unduly harsh for their answer, assign a "holistic" score as you see fit.

5. Consider the code snippet below. It is a contrived example with poor style, but it will run without errors.

   What does it print when it runs? Write your answer in the box below.

   Show your work in any way that you think would be helpful.

```
s = [5,  3,  1,  8,  4,
     9,  7,  6,  2,  10,  20]
a = 3
b = 0
print(len(s))
for k in range(1, 10, 3):
    a = a + s[k]
    b = b + s[len(s) - k - 1]
    print("A.", k, len(s) - k - 1)
    print("B.", s[k], s[len(s) - k - 1])
    print("C.", a, b)
    print()
print("Now:", a, b)
```

Rubric (8 points):

Subtract 1 point for each line that is wrong, except:

- Full credit if they forgot to write the ***11***.
- If the ***11*** is wrong, subtract 1 point and try to grade the problem equitably given that they may have miscounted the number of items in the list.
- Ignore wrong periods, misspellings, spaces, etc.
- If a single number is wrong and they appear to have made only a clerical mistake, assign full credit.
- If the first number in the first column of numbers is wrong but the subsequent numbers are consistent with it, subtract only 1 point (once).
- Likewise for the second column of numbers.

Output:

```
11
A. 1 9
B. 3 10
C. 6 10

A. 4 6
B. 4 7
C. 10 17

A. 7 3
B. 6 8
C. 16 25

Now: 16 25
```

In addition to the points to the left: If there are 4 chunks of ABC items instead of 3 (an "off by one" error in the range), subtract 1 point. If there are more than 4 such chunks, subtract 2 points.

Don't subtract more than 8 points (no negative scores). If this rubric is unduly harsh for their answer, assign a "holistic" score as you see fit.

6. Consider a function named ***average*** that takes a single argument:
a sequence of numbers, where the length of the sequence is *guaranteed* to be odd.

The function returns the average of the numbers in the sequence that are bigger than or equal to the number at the middle of the sequence. For example:

- `average( [1, 7, 6, 5, 10, 3, 9] )` returns **7.4**
since the number at the middle of the sequence is **5**
and the numbers in the list bigger than or equal to **5** are **7, 6, 5, 10** and **9**
and those **5** numbers add up to **7 + 6 + 5 + 10 + 9**, which is **37,**
and the average of **5** numbers that sum to **37** is **37 / 5**, which is **7.4**.

- `average( [2, 1, 4, 4, 7] )` returns **5.0** since the number at the middle of the sequence is **4,** and the numbers in the list bigger than or equal to **4** are **4, 4** and **7**, and the sum of those **3** numbers is **15**, so the average of them is **15/3**, which is **5.0**.

- `average( [9, 4, 7, 6, 2] )` returns **8.0** since the number at the middle of the sequence is **7,** and the numbers in the list bigger than or equal to **7** are **9** and **7**, and the sum of those **2** numbers is **16**, so the average of them is **16/2**, which is **8.0**.

Write (in the box below) a complete implementation of the above ***average*** function.

```
def average(numbers):
    count = 0
    total = 0
    middle_number = numbers[len(numbers) // 2]
    for k in range(len(numbers)):
        if numbers[k] > middle_number:
            count = count + 1
            total = total + numbers[k]
    return total / count
```

See the next page for the rubric.

Rubric (12 points):

- 1 BONUS point if they indicate in any way that they would have called their answer to Problem 2 to compute the middle number.
- Ignore any of the following mistakes: missing colon, `/` instead of `//`, mismatched parentheses, any typographical mistake like that, or misspellings.
- A correct answer should have the following components. Subtract points for each as indicated, but subtracting at most 12 points (no negative scores).
    - Computest the middle number in the list correctly. **3 points**, but if they made the SAME error as they did in Problem 2, then subtract only 1 point for the error.
    - A count that increments by 1 conditionally inside the loop. **2 points**. Subtract ½ point if they omit the `count = 0`.
    - A sum that increments by some number conditionally inside the loop (regardless of the condition). **2 points**. (So any statement of the form `total = total + ...` inside any IF statement earns these points, except subtract ½ point if they omit the `total = 0`.)
    - A sum that increments by `numbers[k]`. **2 points**. Subtract ½ point if they write `numbers(k)` instead of `numbers[k]`.
    - A loop with the correct RANGE statement. **2 points** (but subtract only 1 point for an off-by-one error, or for omission of the ***`Len`***, or anything close like that).
    - An IF statement that references `numbers[k]`. **2 points**.
    - An IF statement that correctly compares to the middle number. **2 points**.
    - Divides the total by the count. **2 points** (but subtract only 1 point if they compute it INSIDE the loop, where it might cause a divide-by-zero error).
    - Returns the total divided by the count. **1 point**.

This is the back page of this exam. Use it for scratch work if you like.