Exam 2 – Paper and Pencil part (Winter, 2018-19)

Name: ______SOLUTION_____ Section: _______

<u>Rules and Expectations</u>

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 <u>finish</u> this exam (both parts).***

<u>Two parts (this is Part 1, Paper-and-Pencil)</u>

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

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

<u>Communication:</u> For both parts of the exam, ***you must not communicate with anyone*** except your instructors and her 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.

You must NOT use email, *chat* or the like during this exam. **Close all such applications before you start the exam.**

1. (10 points.) Consider a function named ***blah*** that takes a ***list of numbers*** as its sole argument. For each of the following possible specifications for what ***blah*** returns:

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

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

1 pt each

If ***blah*** returns:

a. The smallest number in the list.	(Yes)	No
b. The second smallest number in the list.	(Yes)	No
c. The second number in the list.	Yes	(No)
d. The first positive number in the list, or -1 if there is no positive number in the list.	(Yes)	No
e. ***True*** if the first number in the list is positive, else ***False***.	Yes	(No)
f. ***True*** if the list contains no positive numbers, else ***False***.	(Yes)	No
g. The average of the positive numbers in the list.	(Yes)	No
h. The number of numbers in the list.	Yes	(No)
i. The number of positive numbers in the list.	(Yes)	No
j. The number in the middle of the list.	Yes	(No)

2. (2 points) Consider the code shown to the right. As always for quiz questions or exams, assume **rg.Rectangle** ***uses*** Point objects and **does not make clones**.

```
p1 = rg.Point(50, 25)
p2 = rg.Point(8, 88)
p3 = p2
p4 = rg.Point(p1.x + 100, p2.y + 40)
r = rg.Rectangle(p3, rg.Point(9, 321))
```

How many **rg.Point** objects have been constructed?

1   2   3   (4)   5   6   7   8   *more than 8* (circle your choice)

3. (10 points). 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?

```
def main():
    foo([33, 1, 3, 5, 3, 99])
    foo([100])
    foo([1, 2, 1, 3, 1])


def foo(seq):
    print(seq)
    for k in range(len(seq) - 1):
        if seq[k] > k:
            print(k, seq[k])
        else:
            print(k)


main()
```

Output:
[33,1,3,5,3,99]
0 33
1
2 3
3 5
4

[100]

[1, 2, 1, 3, 1]
0 1
1 2
2
3

2 pts for showing lists
3 pts for ignoring last element of sequence
2 pts for k's
3 pts for correct seq[k]s printed (start with 3, -1 for each extra/omitted, but worth max of 3)

4. Consider the code on the page to the right of this 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, in the table below:
   - ***CIRCLE* each variable** that is ***defined*** at that location.
     WRITE the ***VALUE*** of each variable that you ***circled*** directly ***BELOW*** the circle.

1 pt each perfect row, considered perfect if transforms earlier wrong one correctly.

Location 1	a	b	d 60	n	donut .radius	donut .calories	glazed .radius	glazed .calories	self .radius	self .calories
Location 2	a	b	d 60	n	donut .radius	donut .calories	glazed .radius	glazed .calories	self .radius 30	self .calories 600
Location 3	a	b	d	n 20	donut .radius	donut .calories	glazed .radius	glazed .calories	self .radius 30	self .calories 600
Location 4	a	b	d	n 20	donut .radius	donut .calories	glazed .radius	glazed .calories	self .radius 10	self .calories 600
Location 5	a 4	b 123	d	n	donut .radius	donut .calories	glazed .radius	glazed .calories	self .radius	self .calories
Location 6	a 10	b	d	n	donut .radius 30	donut .calories 600	glazed .radius	glazed .calories	self .radius	self .calories
Location 7	a 10	b	d	n	donut .radius 10	donut .calories 80	glazed .radius	glazed .calories	self .radius	self .calories
Location 8	a 10	b 40	d	n	donut .radius	donut .calories	glazed .radius	glazed .calories	self .radius	self .calories
Location 9	a 10	b 40	d	n	donut .radius 30	donut .calories 600	glazed .radius 30	glazed .calories 600	self .radius	self .calories
Location 10	a 70	b 40	d	n	donut .radius 10	donut .calories 80	glazed .radius 10	glazed .calories 80	self .radius	self .calories

Showing your work (by marking up the code, drawing a box-and-pointer diagram, or any other way you wish) is the best way to allow for partial credit.
Feel free to use a separate blank sheet of paper (and attach it to this exam) if you like.

ASK FOR HELP IF YOU DO NOT UNDERSTAND WHAT THIS PROBLEM ASKS YOU TO DO.

5.

```
class Donut(object):
   def __init__(self, d):
       #### ---  Location 1  ---
       self.radius = d / 2
       self.calories = d * 10
       #### ---  Location 2  ---

   def bite(self, n):
       #### ---  Location 3  ---
       self.radius = self.radius - n
       if self.radius < 0:
           self.calories = 0
       #### ---  Location 4  ---


def foo1(a, b):
    b = 123
    #### ---  Location 5  ---
    return 10 * a


def foo2(donut, a):
    #### ---  Location 6  ---
    donut.bite(a * 2)
    donut.calories = 80
    #### ---  Location 7  ---
    return donut.calories - 10


def main():
    a = 10
    b = 20
    b = foo1(4, a)
    #### ---  Location 8  ---
    glazed = Donut(60)
    donut = glazed
    #### ---  Location 9  ---
    a = foo2(glazed, a)
    #### ---  Location 10  ---


main()
```

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 to the right of the code.

   Show your work by marking up the code to show intermediate values.

```
def main():
    x = pizza(donuts(12, 80))
    print('Main: ', x)


def donuts(x, y):
    z = pizza(y) + pretzel(x)
    print('Donuts: ', x, y, z)
    return z + y


def pizza(a):
    print('Pizza: ', a)
    return a / 4
    print('Z2: ', 10 * a)


def pretzel(a):
    b = pizza(10 * a)
    print('Pretzel: ', a, b)
    return b


main()
```

Pizza: 80 (2 pts)

Pizza: 120 (2 pts)

Pretzel: 12 30 (2 pts)

Donuts: 12 80 50 (2 pts)

Pizza: 130 (1 pt)

Main: 32.5 (1 if ¼ of last Pizza)

-1 for each out of order, up to -3

(But try to give credit for the parts that they do right based on wrong inputs)

6. (4 points) Assume that there is a class named ***Elevator*** whose constructor:
   - requires two arguments: the number of floors in the Elevator's building, and the current floor at which the Elevator resides.
   - and stores those arguments in instance variables named ***num_floors*** and ***current_floor***, respectively.

   Assume further that ***Elevator*** objects have a ***go_to_floor*** method that takes an positive argument N that is the floor to which the Elevator should move, and moves the Elevator to floor N unless N is greater than the Elevator's number of floors (in which case the Elevator remains at its current floor).

   a. Write code that would construct an **Elevator** object for a building with **12** floors, with the Elevator starting at floor **5**.

      elevator = Elevator(12, 5) **2 pts**

   b. Write code that would use the ***go_to_floor*** method to move the Elevator from part (a) to the floor that is 3 floors higher than its current floor.

      elevator.go_to_floor(elevator.current_floor + 3) **2 pts**

7. (4 points) Continuing the previous problem, implement the methods in the Elevator class by filling in the blanks in the following:

```
class Elevator(object):

    def __init__(self, num_floors, current_floor):
        # Write code here that implements the __init__ method.
        self.num_floors = num_floors
        self.current_floor = current_floor
                                        2 pts

    def go_to_floor(self, N):
        # Write code here that implements the go_to_floor method.
        if N <= self.num_floors:
            self.current_floor = N
                                        2 pts
```