Test 3 – *SOLUTION* to Practice Problems for the Paper-and-Pencil portion

1. In the space below, write an implementation for the function whose specification is shown in the following box. Do NOT use your computer for this (or for any other of these paper-and-pencil problems).

```
def shape(r):
    """
    Prints shapes per the following examples:
    When r = 5:                             When r = 3
        *****5                              ***3
        ****54                              **32
        ***543                              *321
        **5432
        *54321
    Precondition:  r is a non-negative integer.
    For purposes of "lining up", assume r is a single digit.
    """
```

```
One answer:

    for k in range(r):
        for j in range(r - k):
            print('*', end='')
        for j in range(k + 1):
            print(r - j, end='')
        print()
```

2. Consider the code snippet below. It is a contrived example with poor style, but it will run without errors. What does it print when ***main*** runs?

   Write your answer in the box to the right.

```
def main():
    for j in range(5):
        for k in range(j):
            print(j, k)
```

Output:
(I have put extra blank lines in this solution to make it more readable.)

```
1 0

2 0
2 1

3 0
3 1
3 2

4 0
4 1
4 2
4 3
```

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 ***main*** runs?

   Write your answer in the box to the left.

```
def main():
    for j in range(5):
        print('here')
        for k in range(1, j - 1):
            print(j, k)

        print('there')
        for k in range(2, j + 1):
            print(j, k)
```

Output: (I have put extra blank lines in this solution to make it more readable.)

```
here
there

here
there

here
there
2 2

here
3 1
there
3 2
3 3

here
4 1
4 2
there
4 2
4 3
4 4
```

4. Consider the code snippet in the box below. It is a contrived example with poor style, but it will run without errors. What does it print when ***main*** runs?

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

```
def main():
    seq = [('one', 'two', 'three', 'four'),
           ('five', 'six', 'seven'),
           ('eight', 'nine', 'ten'),
           ['is this ok?'],
           (),
           ('123456', '1234')]

    for k in range(len(seq)):
        for j in range(len(seq[k])):
            print(j, k)
            if len(seq[k][j]) > 3:
                print(seq[k][j], len(seq[k][j]))
```

Output:

(I have put extra blank spaces and lines in this solution to make it more readable.)

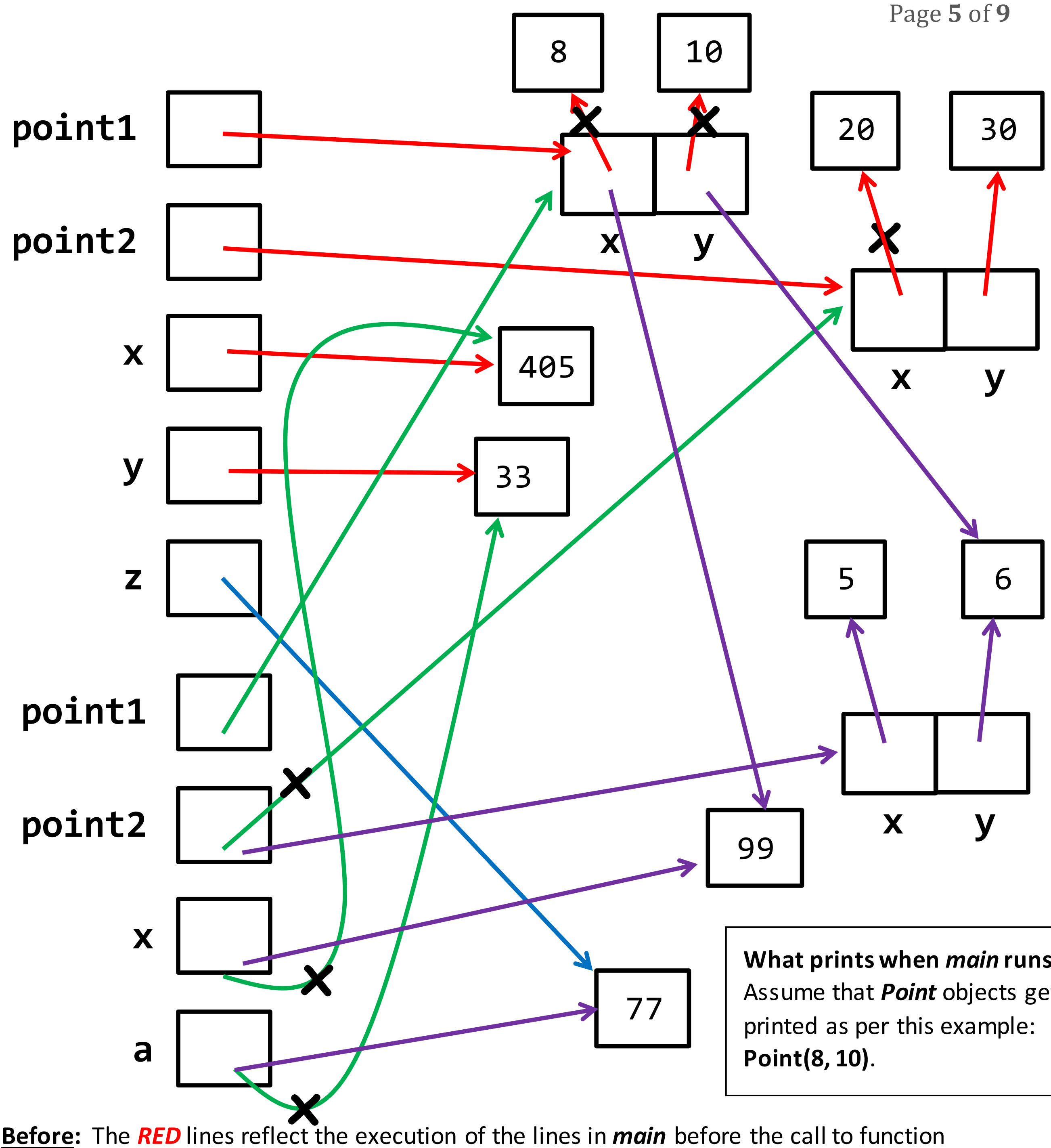
```
0   0
1   0
2   0
three   5
3   0
four   4

0   1
five   4
1   1
2   1
seven   5

0   2
eight   5
1   2
nine   4
2   2

0   3
is this ok?   11

0   5
123456   6
1   5
1234   4
```

5. In Session 9, you implemented a ***Point*** class. Recall that a ***Point*** object has instance variables ***x*** and ***y*** for its x and y coordinates.

   Consider the code in the box below. On the **next** page, draw the ***box-and-pointer diagram*** for what happens when ***main*** runs. Also on the next page, show what the code would ***print*** when ***main*** runs.

```python
def main():
    point1 = Point(8, 10)
    point2 = Point(20, 30)
    x = 405
    y = 33

    print('Before:', point1, point2, x, y)

    z = change(point1, point2, x, y)

    print('After:', point1, point2, x, y, z)


def change(point1, point2, x, a):
    print('Within 1:', point1, point2, x, a)
    point2.x = point1.x
    point2 = Point(5, 6)
    point1.y = point2.y
    x = 99
    point1.x = x
    a = 77

    print('Within 2:', point1, point2, x, a)

    return a
```

<u>Draw your box-and-pointer diagram on the next page</u>:



What prints when *main* runs?
Assume that ***Point*** objects get printed as per this example:
Point(8, 10).

Before: The ***RED*** lines reflect the execution of the lines in ***main*** before the call to function ***change***. Therefore, what gets printed BEFORE the call to ***change*** is:

```
Point(8, 10)   Point(20, 30)   405   33
```

Within: The ***GREEN*** lines reflect the execution of the call to function ***change***. Thus what gets printed at ***Within 1:*** is `Point(8, 10)   Point(20, 30)   405   33`

The ***PURPLE*** lines reflect the execution of the lines in ***change***. Therefore, what gets printed WITHIN the call to ***change*** (at the end of that function, i.e., when ***Within 2:*** is printed) is:

```
Point(99, 6)   Point(5, 6)   99   77
```

After: The ***BLUE*** line reflects the execution of the return from ***change*** and the assignment to ***z*** in function ***main***. Therefore, what gets printed AFTER the call to ***change*** is:

```
Point(99, 6)   Point(8, 30)   405   33   77
```

From the picture on the previous page, we see that:

What prints when *main* runs?

Assume that ***Point*** objects get printed as per this example: **Point(8, 10)**.

Before: `Point(8, 10)    Point(20, 30)    405    33`

Within 1: `Point(8, 10)    Point(20, 30)    405    33`

Within 2: `Point(99, 6)    Point(5, 6)    99    77`

After: `Point(99, 6)    Point(8, 30)    405    33    77`

6. 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.

```
x = 2
while x < 9:
    print(x)
    x = x + 3
print('One', x)

y = 2
while True:
    print(y)
    if y > 9:
        break
    y = y + 3

print('Two', y)
```

Output:

(I have put extra blank spaces and lines in this solution to make it more readable.)

```
2

5

8

One 11


2

5

8

11

Two 11
```

7. True or false: ***Variables are REFERENCES to objects.*** [✓ **True**] **False** (circle your choice)

8. True or false: ***Assignment*** (e.g. **`x = 100`**) causes a variable to refer to an object. [✓ **True**] **False** (circle your choice)

9. True or false: ***Function calls*** (e.g. **`foo(54, x)`**) also cause variables to refer to objects. [✓ **True**] **False** (circle your choice)

10. Give one example of an object that is a ***container*** object:

    Here are several examples: a *list*, a *tuple*, a rg.Circle, a Point, an rg.*window*

11. Give one example of an object that is ***NOT*** a ***container*** object:

    Here are several examples: an *integer*, a *float*, None, `True`, `False`.

12. True or false: *When an object is mutated, it no longer refers to the same object to which it referred prior to the mutating.* **True** [✓ **False**]
    (circle your choice)

13. Consider the following statements:

    ```
    c1 = rg.Circle(rg.Point(200, 200), 25)
    c2 = c1
    ```

    At this point, how many ***`rg.Circle`*** objects have been constructed? [✓ **1**] **2**
    (circle your choice)

14. Continuing the previous problem, consider an additional statement that follows the preceding two statements:

    ```
    c1.radius = 77
    ```

    True or False: After the above statement executes, the variable ***`c1`*** refers to the same object to which it referred prior to this statement. [✓ **True**] **False**
    (circle your choice)

15. Continuing the previous problems:

    - What is the value of ***`c1`***'s radius after the statement in the previous problem executes? **25** [✓ **77**] (circle your choice)
    - What is the value of ***`c2`***'s radius after the statement in the previous problem executes? **25** [✓ **77**] (circle your choice)

16. In Session 9, you implemented a ***Point*** class. Recall that a Point object has instance variables **x** and **y** for its x and y coordinates

Consider the code snippets below. They are contrived examples with poor style but will run without errors. For each, what does it print when *main* runs?

(Each is an independent problem.)

```
def main():
    p1 = Point(11, 12)
    p2 = Point(77, 88)
    p3 = foo(p1, p2)
    print(p1.x, p1.y)
    print(p2.x, p2.y)
    print(p3.x, p3.y)

def foo(p1, p2):
    p1 = Point(0, 0)
    p1.x = 100
    p2.y = 200
    p3 = Point(p2.x, p1.y)
    return p3
```

```
def main():
    a = [1, 2, 3]
    b = [100, 200, 300]
    c = foofoo(a, b)
    print(a)
    print(b)
    print(c)

def foofoo(a, b):
    a = [11, 22, 33]
    a[0] = 777
    b[0] = 888
    x = [a[1], b[1]]
    return x
```

Prints:

```
11   12

77  200

77    0
```

Prints:

```
[1, 2, 3]

[888, 200, 300]

[22, 200]
```