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

Note: These practice problems relate very directly to what you can expect to see on the Paper-and-Pencil portion of Test 3. **Do these problems carefully** and **check your answers via the online Solution to these problems**.

Many of these problems are similar to problems that you saw on the Paper-and-Pencil portion of Test *2*.

Each problem has a *time estimate* that indicates how long the problem might take (at most) for a well-prepared student who had no prior experience in software development before this course. *If you are taking much longer than the time estimates, work with your instructo*r to learn techniques to solve them more quickly.

A well-prepared student should be able to complete **problems 1 through 22** in well **under 2 hours**.

Be forewarned that the Paper-and-Pencil portion of Test 3 may not be as generous regarding partial credit as was Test 2. *If you are not 100% clear on ALL parts of ALL of these problems, meet with your instructor,* a course assistant, or another equally qualified person to become solid in your understanding of the concepts that these problems assess:

- Problem 1: Scope of variables, flow of control through function calls and object construction, arguments and parameters.
- Problem 2: Mutation via function calls (when the arguments are mutable), box-and-pointer diagrams, references, mutation versus reassignment.
- Problem 3: Object construction.
- Problem 4: Tracing function calls by hand, returning values, order of operations, `return` really *leaves* the function.
- Problem 5: Object construction, aliases (two names for the same object).
- Problem 6: Mutation via assignment of the insides of container objects, box-and-pointer diagrams, references, aliases, mutation versus reassignment.
- Problem 7: Mutation via function calls (when the arguments are mutable), box-and-pointer diagrams, references, mutation versus reassignment.
- Problems 8, 9 and 10: Tracing loops within loops by hand, with #10 featuring sequences inside sequences.
- Problem 11: Tracing `while` loops by hand.
- Problem 12: Implementing by hand a function that requires loops within loops.
- Problem 13: Implementing a simple class by hand.
- Problems 14 through 22: References and mutation.

1. [A well-prepared student should not require more than about **10** **minutes** to complete this problem.]

   Consider the code on the next page. Arrange so that you can see both this page and the next page at the same time. (Un-staple as needed.) On the test, you will receive the code for any problem like this one on a separate page.

   The code 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:*

   1. *CIRCLE each variable* that is *defined* at that location.

   2. *WRITE the VALUE* of each variable that you *circled* directly *BELOW* the circle.

   For example, the run starts in *main*, as usual. The first of the seven locations **to be encountered** is **Location 6**. At Location 6, the only variables defined are *a* and *z*, with values **5** and **3** at that point of the program's run. So, on the row for Location 6, I have circled *a* and *z* and written their values at Location 6 directly below them.

   Note that you fill out the table in the order that the locations are encountered, *NOT from top to bottom. ASK FOR HELP IF YOU DO NOT UNDERSTAND WHAT THIS PROBLEM ASKS YOU TO DO.*

Location 1 the 1st time that it is encountered	(a) 5	(w) 7	z	self.w	cat.w	dog.w	cat.a
Location 1 the 2nd time that it is encountered	(a) 1	(w) 9	z	self.w	cat.w	dog.w	cat.a
Location 2 the 1st time that it is encountered	(a) 10	(w) 7	z	(self.w) 70	cat.w	dog.w	cat.a
Location 2 the 2nd time that it is encountered	(a) 10	(w) 9	z	(self.w) 90	cat.w	dog.w	cat.a
Location 3	(a) 3	(w) 4	(z) 5	self.w	cat.w	dog.w	cat.a
Location 4	(a) 3	(w) 4	(z) 5	self.w	cat.w	(dog.w) 70	cat.a
Location 5	(a) 3	(w) 4	(z) 5	self.w	(cat.w) 90	(dog.w) 80	cat.a
Location 6	(a) 5	w	(z) 3	self.w	cat.w	dog.w	cat.a
Location 7	(a) 5	w	(z) 3	self.w	cat.w	(dog.w) 90	cat.a

Code for Problem 1:

Arrange so that you can see this code and the problem itself at the same time. (Un-staple as needed.)

The arrows are there to help you see where the seven Locations are in the code.

```python
class Animal(object):
    def __init__(self, w, a):
        #### Location 1          ⬅


        a = 10
        self.w = a * w    # MULTIPLY, not add
        #### Location 2          ⬅


    def eat(self, w):
        self.w = self.w + (2 * w)


def make_animals(z, a, w):
    #### Location 3              ⬅


    dog = Animal(w + a, z)
    #### Location 4              ⬅


    cat = Animal(w + z, 1)
    dog.eat(5)
    #### Location 5              ⬅


    return cat


def main():
    a = 5
    z = 3
    #### Location 6              ⬅


    dog = make_animals(a, z, z + 1)
    #### Location 7              ⬅
```

2. [A well-prepared student should not require more than about **15 minutes** to complete this problem.]

Consider the code on the next page. Arrange so that you can see both this page and the next page at the same time. (Un-staple as needed.)   On the test, you will receive the code for any problem like this one on a separate page.

The code is a contrived example with poor style but will run without errors. Trace the code's execution and **draw a box-and-pointer diagram on a separate sheet of paper** as you trace the code's execution.

As you draw the box-and-pointer diagram, write (in the box below) what gets printed when *main* runs. Write Point objects as in this example: **(100, 150)**.

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

Beta 1:   33

Beta 2:   100

Beta 3:   (600, 99)

Beta 4:   [100, 200, 300]

Main 1:   70

Main 2:   (678, 150)

Main 3:   (678, 150)

Main 4:   [444, 100, (678, 150), 222]

Main 5:   300

Reminder:

You must *draw* (on a separate page) a

** *box-and-pointer* ** *diagram* for this problem.

NOTE: The solution for the box-and-pointer diagram appears two pages after this page.

```python
def main():
    radius = 70
    center = Point(100, 150)
    p = center
    seq = [radius, center.x, p, 10]

    ans = beta(radius, center.x, p, seq)

    print('Main 1:', radius)
    print('Main 2:', center)
    print('Main 3:', p)
    print('Main 4:', seq)
    print('Main 5:', ans)


def beta(radius, x, p, seq):
    radius = 33
    p.x = 678
    seq[0] = 444
    seq[3] = 222
    seq = [100, 200, 300]
    p = Point(600, 700)
    p.y = 99

    print('Beta 1:', radius)
    print('Beta 2:', x)
    print('Beta 3:', p)
    print('Beta 4:', seq)
    return seq[2]


main()
```
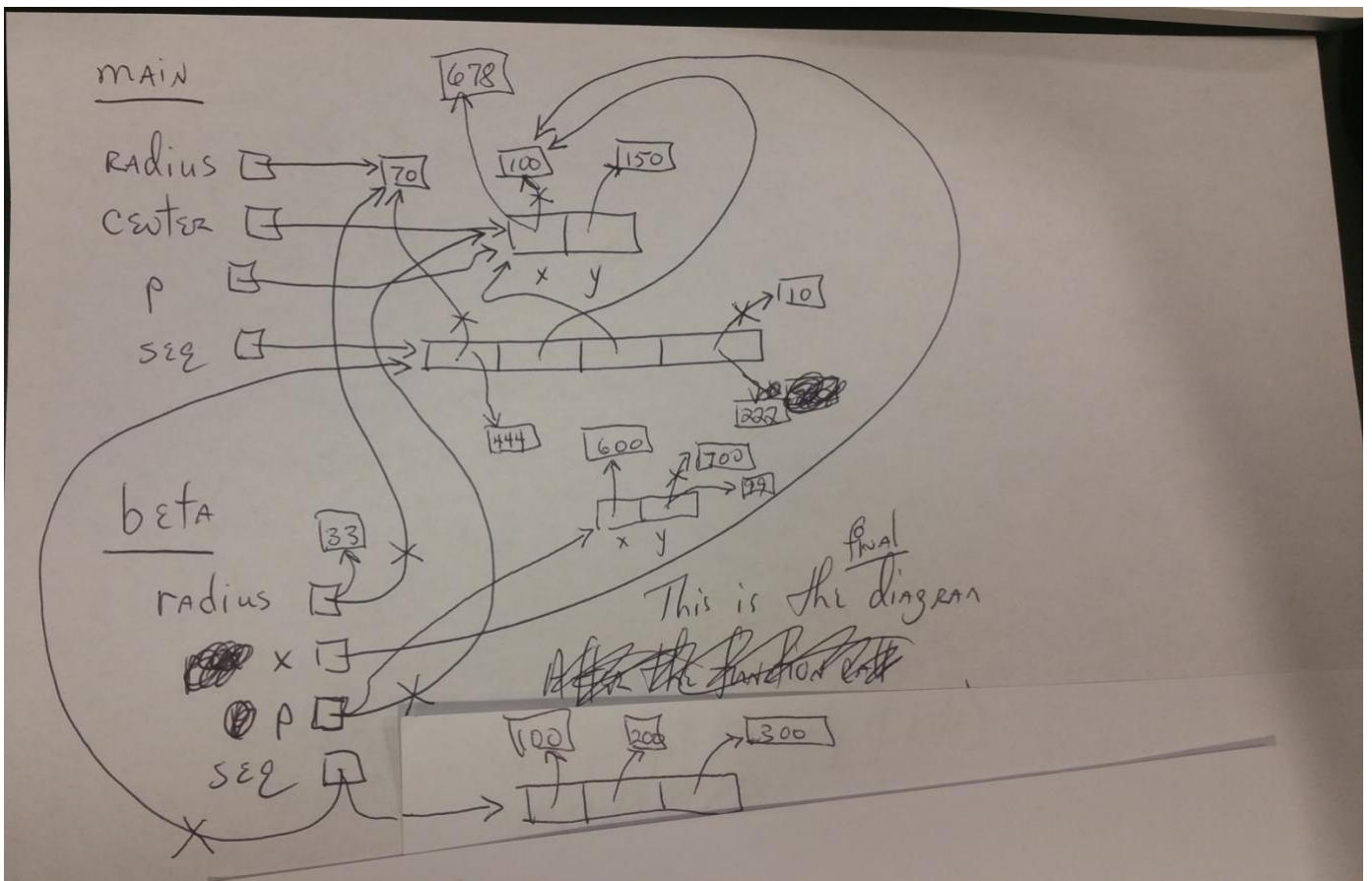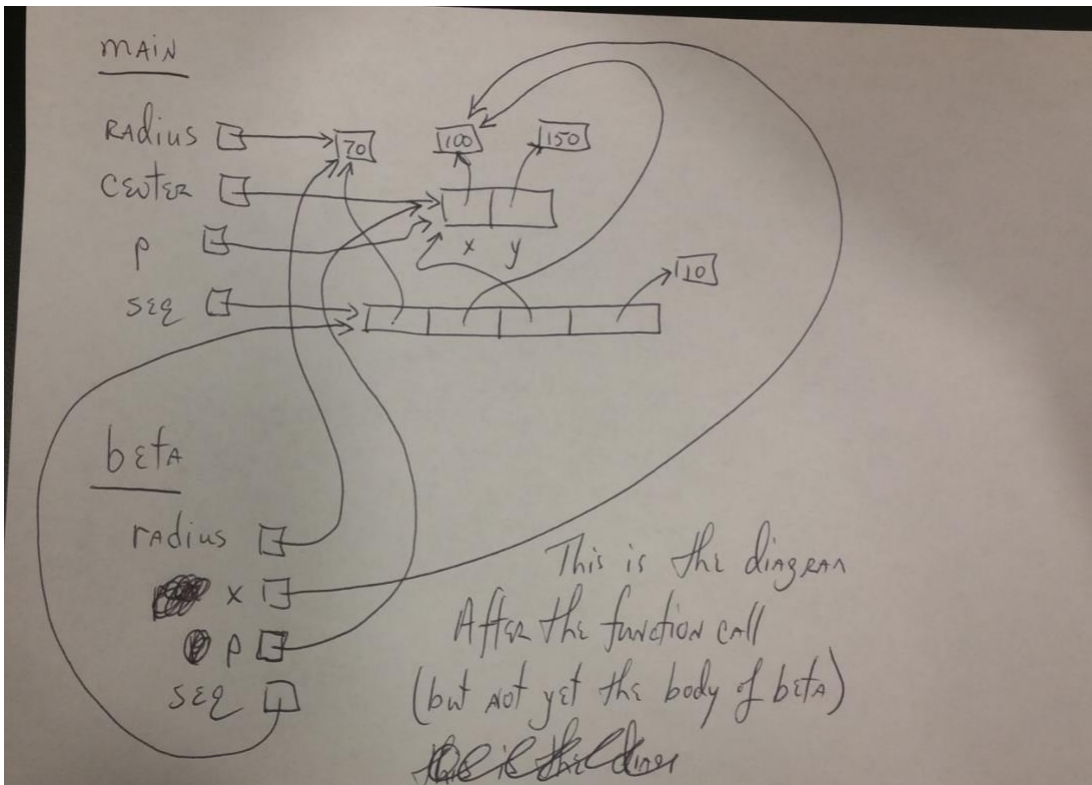
MAIN

RADIUS → 70   100   150

CENTER

P

SEQ → 10

x y

beta

radius

x

P

SEQ

This is the diagram
After the function call
(but not yet the body of beta)

MAIN   678

RADIUS → 70   100   150

CENTER

P

SEQ → 10

x y

444   600   700   77

beta

33

radius   222

x

P   final

SEQ   This is the diagram

100   200   300

3. [A well-prepared student should not require more than about **1** **minute** to complete this problem.]

   When the code in the previous problem runs: (circle your choice for each of the following)

   a. How many *Point* objects are constructed in *main*?   0   **①**   2   3   4

   b. How many *Point* objects are constructed in *beta*?   0   **①**   2   3   4

4. [A well-prepared student should not require more than about **7 minutes** to complete this problem.]

   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.

```
def main():
    print('Main:', three())


def three():
    print('Three 1:', one(7))
    return two(10) + (100 * one(5))
    print('Three 2:')


def two(y):
    answer = one(2 * y)
    print('Two:', y, answer)
    return (5 + answer)


def one(x):
    print('One:', x)
    return (3 * x)


main()
```

Output:

One: 7

Three 1: 21

One: 20

Two: 10 60

One: 5

Main: 1565

5. [A well-prepared student should not require more than about **1** **minute** to complete this problem.]

   Consider the following four statements:

   ```
   p1 = rg.Point(4, 5)
   p2 = rg.Point(p1.x, p1.y)
   p3 = p1
   p4 = p2
   ```

   At this point, how many **rg.Point** objects have been constructed?   **1**   **(2)**   **3**   **4**
        (circle your choice)

6. [A well-prepared student should not require more than about **7 minutes** to complete this problem.]
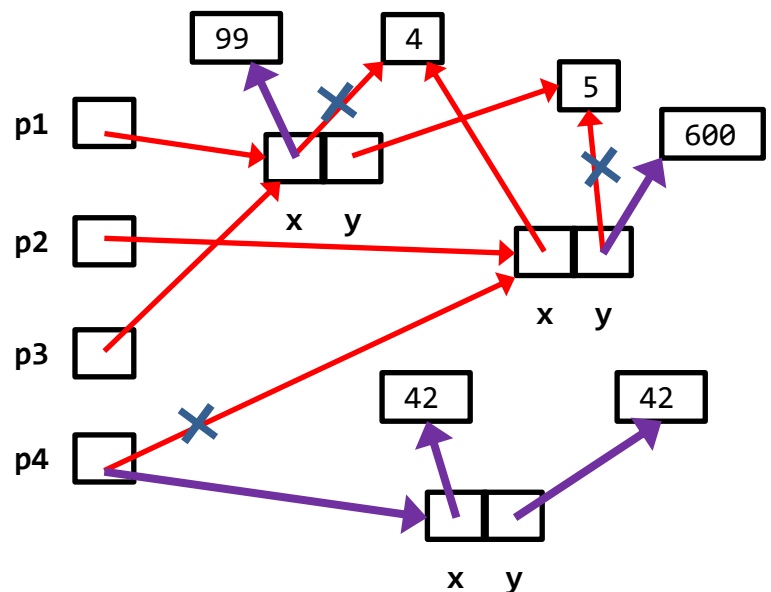
   Consider the code snippet to the right. Trace the execution of the code snippet and **draw a box-and-pointer diagram on a separate sheet of paper** as you trace the code's execution.

   After the code snippet is executed, what are the values of the variables? (Write your answer in the spaces provided below.)

   p1.x = _____   **99**

   p1.y = _____   **5**

   p2.x = _____   **4**

   p2.y = _____   **600**

   p3.x = _____   **99**

   p3.y = _____   **5**

   p4.x = _____   **42**

   p4.y = _____   **42**

   ```
   p1 = rg.Point(4, 5)
   p2 = rg.Point(p1.x, p1.y)
   p3 = p1
   p4 = p2


   p3.x = 99
   p4.y = 600
   p4 = rg.Point(42, 42)
   ```

   In the **box-and-pointer diagram** to the right, arrows in **purple** show the effects of the last 3 lines of code.

7. [A well-prepared student should not require more than about **12 minutes** to complete this problem.]

    Recall that our Point class 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 code snippet:

    1. Trace the code snippet's execution when *main* runs and **draw a box-and-pointer diagram on a separate sheet of paper** as you trace the code snippet's execution.

    2. Write below what the code snippet prints.

    (Each code snippet 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]

Reminder: You must *draw* (on a separate page) *TWO* ** *box-and-pointer diagrams* ** for this problem. ***See the next page for these.***

(I will put the box and pointer diagrams here later.)

8. [A well-prepared student should not require more than about **10 minutes** to complete this and the next problem, combined.]

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.

```python
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

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

9. 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.

```python
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)
```

10. 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.

```python
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
```

11. [A well-prepared student should not require more than about **5 minutes** to complete this and the next problem, combined.]

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.

Note: The expression:

```
while BLAH:
    ...
```

makes the body of the **while** expression keep running "while" BLAH is true. For example:

```
while x > 20:
    ...
```

makes the body (that is, indented part) of the **while** expression keep running while **x** is greater than 20. (And presumably **x** is decreasing inside the body of the loop, so eventually **x** is less than or equal to **20** and the loop then exits.)

```
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

12. 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).

```python
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:

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

13. [A well-prepared student should not require more than about **8 minutes** to complete this problem.]

    Consider the Blah class shown below. Implement the three methods per their specifications.

```python
class Blah(object):

    def __init__(self, a, b):
        """ Sets instance variables names  color  and  size
            to the given arguments. """
        self.color = a
        self.size = b


    def multiply_me(self):
        """ Sets this Blah object's   size   to 10 times
            what its value is when this method is called. """
        self.size = self.size * 10


    def make_child(self, another_blah):
        """ Returns a new Blah object whose  color  is
            the same as this Blah's  color  and whose  size is
            the same as the given argument's  size. """
        return Blah(self.color, another_blah.size)
```

[A well-prepared student should not require more than about **5 minutes** to complete all of the problems on this page.]

14. True or false: ***Variables are REFERENCES to objects***. **(True)** **False** (circle your choice)

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

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

17. 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.RoseWindow`.**

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

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

19. 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)

20. 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)

21. 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)

22. 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)

The remaining problem is OPTIONAL. Do it if you want to reinforce further the concepts of *references, box-and-pointer diagrams,* and *mutation.*

23. 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
```
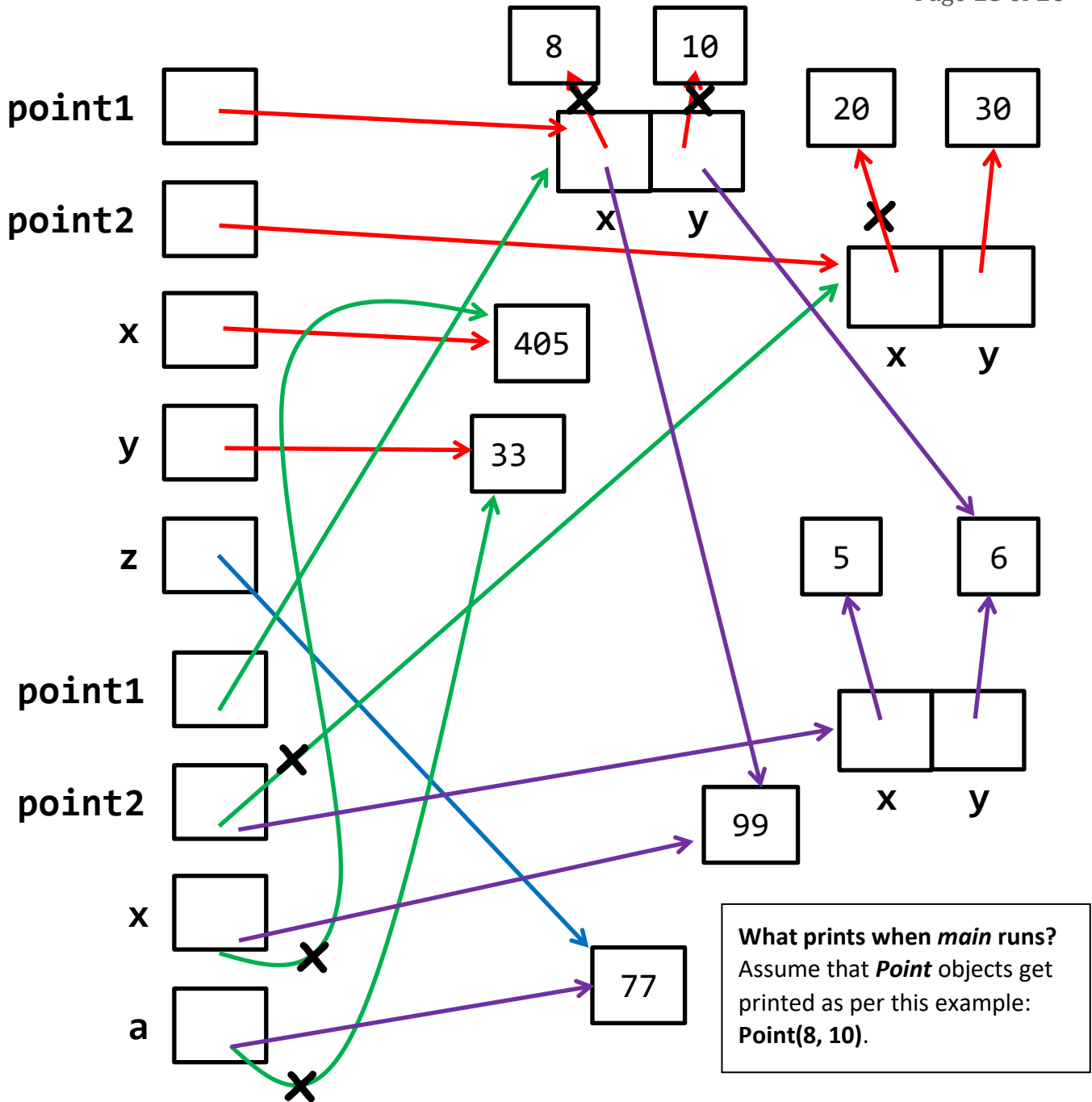
Draw your box-and-pointer diagram on the next page:

point1

point2

x

y

z

point1

point2

x

a

8   10   20   30

405

33

x   y

x   y

5   6

x   y

99

77

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**