Name: ____SOLUTION_________ **Section:** 1 2 3 4 5

PRINT this document and handwrite your answers on it. Bring your solution to class.

Throughout these problems:

- **Use the boxes we supplied; just add labels and arrows for variables and data for non-container objects.**
- **Assume the existence of a *Point* class with just two instance variables (x and y).**
- **Assume the existence of a *Circle* class with just two instance variables (center and radius, where center is a Point object). Assume that a Circle object stores the Point object that it is given as the center, not a copy of that Point.**

As a reminder, here are the four rules for drawing box-and-pointer diagrams, followed by an example from the video.

Rule 1: Draw a ***NON-container object*** by putting its value inside a box.

Rule 2: Draw a ***variable*** using a box labeled with the variable's name and with arrows from the box to the object to which the variable currently refers.

Rule 3: Draw a ***CONTAINER object*** by making a box for it, and then creating sub-boxes that are drawn as if they were variables, but with names for the instance variables of an object and indices for items of a sequence.

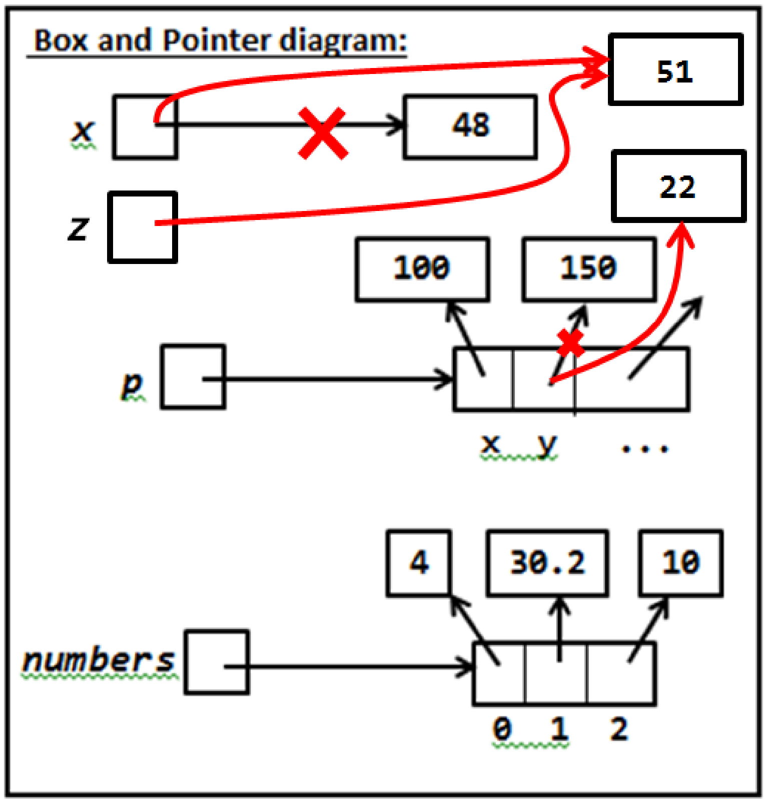
Rule 4: When code RE-assigns a variable, as in `x = blah`:

- Evaluate the expression on the right-hand-side. If it is a new object, draw a box for it.
- Cross through the existing arrow from the variable.
- Draw a NEW arrow from the variable to the object to which the right-hand-side evaluated.

Arrows ALWAYS go FROM a ***variable's*** box TO an ***object's*** box.

Arrows NEVER go from a ***variable's*** box to ***another variable's*** box.

```
x = 48
p = rg.Point(100, 150)
numbers = [4, 30.2, 10]

x = x + 3
p.y = 22
z = x
```



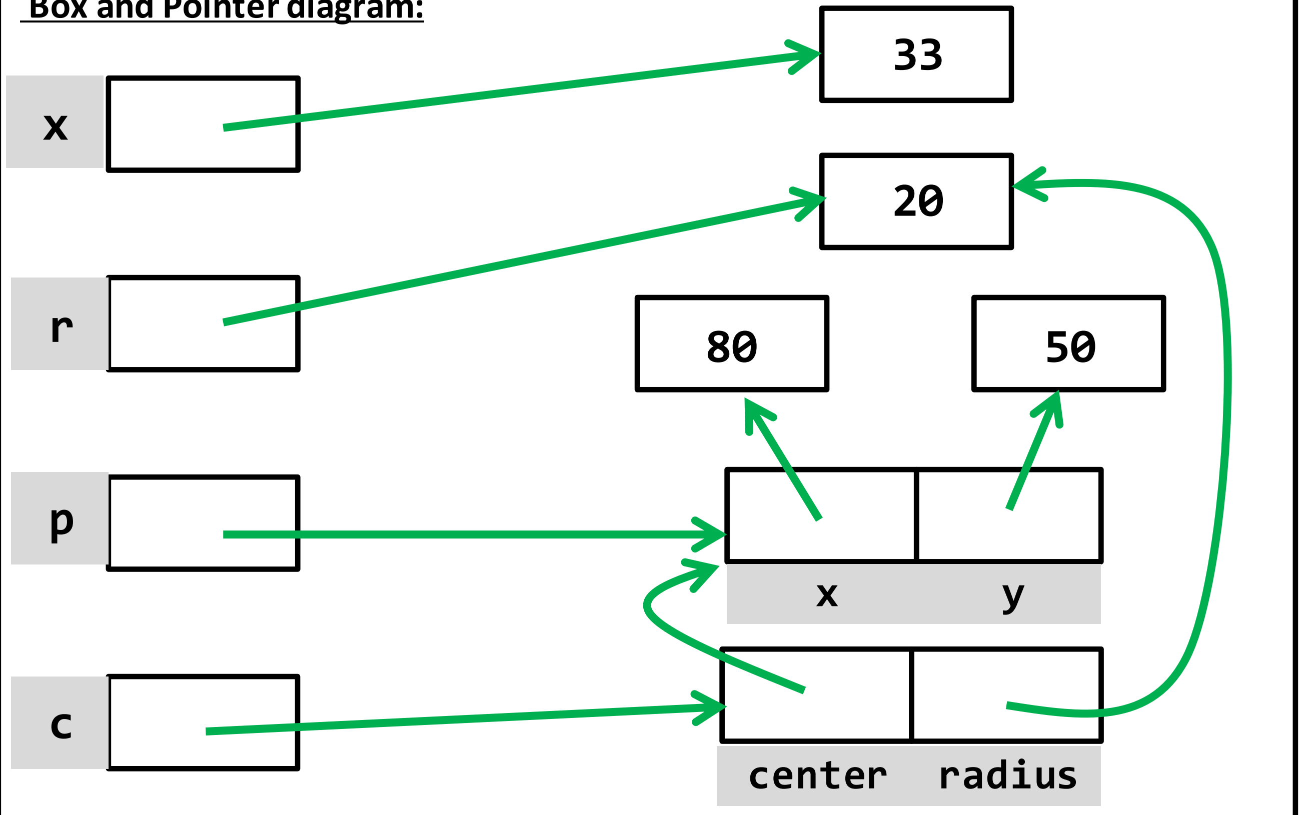
1. Using the diagram at the bottom of this page, draw a Box-and-Pointer diagram that shows what happens when the following statements execute. Also indicate what output is printed. Note that we already supplied the boxes for the diagram; you label them and draw arrows.

```
x = 33
r = 20
p = Point(80, 50)
c = Circle(p, r)

print('x:',   x)
print('r:',   r)
print('p.x:', p.x)
print('p.y:', p.y)
print('c.center.x:', c.center.x)
print('c.center.y:', c.center.y)
print('c.radius:',   c.radius)
```

Output:

```
x:            33
r:            20
p.x:          80
p.y:          50
c.center.x:   80
c.center.y:   50
c.radius:     20
```

Box and Pointer diagram:



2. This problem continues the previous one. We have drawn a ***SOLUTION*** to the previous problem below. Use it to check your answer to the previous problem. Then augment the box-and-pointer diagram below to include the new statements in the code below. Also indicate what output is printed by the *print* statements that follow that new code.

```
x = 33
r = 20
p = Point(80, 50)
c = Circle(p, r)
<same print statements as in
problem 1>

r = 77
p.x = 44

<same print statements as in
problem 1, repeated here>
```

Previous problem printed these numbers.

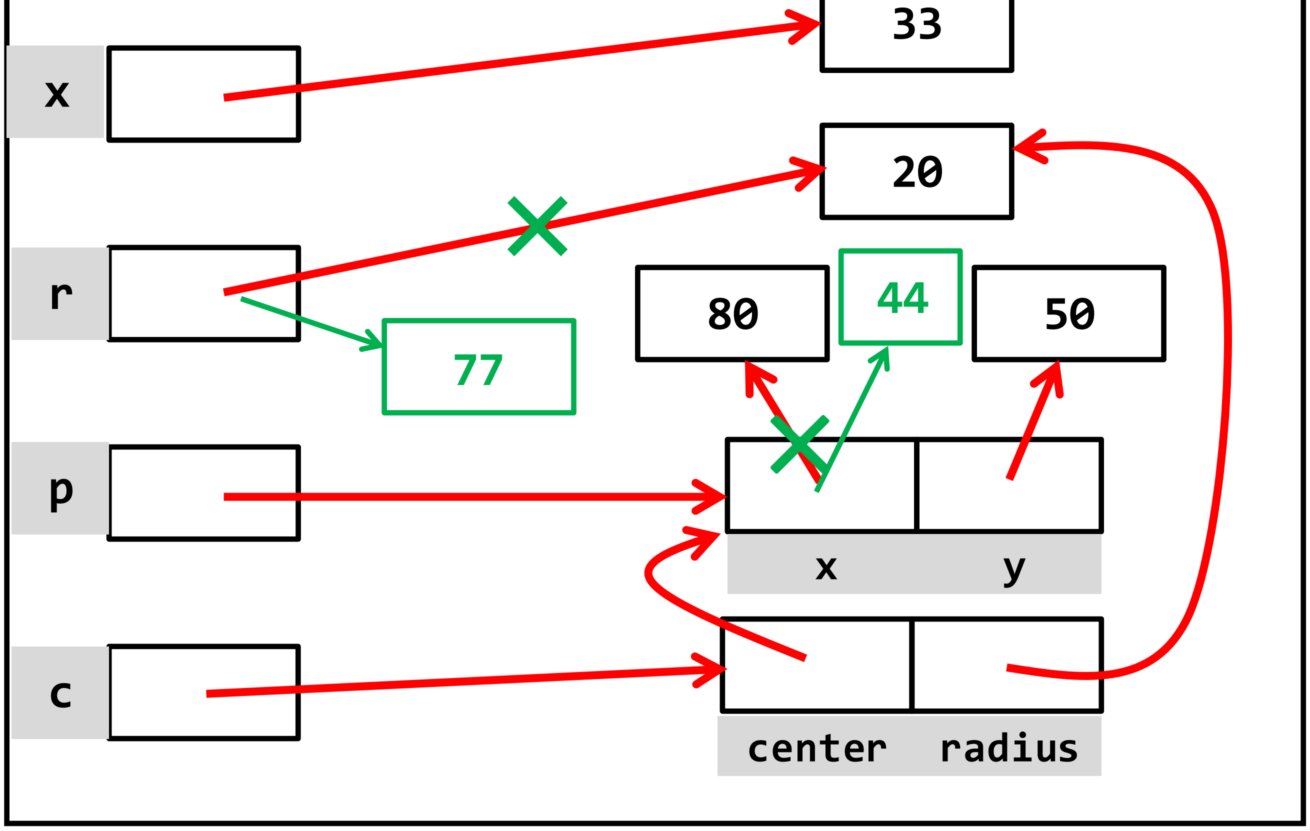
New code is here

Output from 2nd set of print statements:

Especially check the circled ones!

33	x:	33
20	r:	77
80	p.x:	44
50	p.y:	50
80	c.center.x:	44
50	c.center.y:	50
20	c.radius:	20

Box and Pointer diagram:



Your next problems will involve ***function calls***.

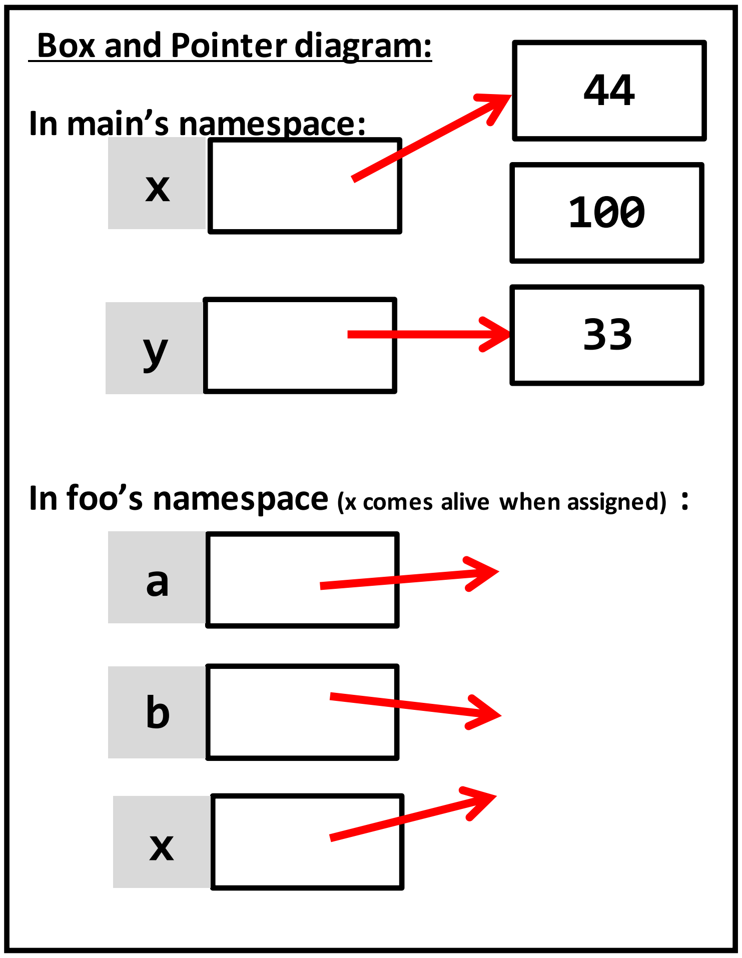
A ***function call*** creates a new ***namespace*** in which the function will run. The function's ***parameters*** are variables in that namespace, as are all variables assigned values with assignment in that function.[1]

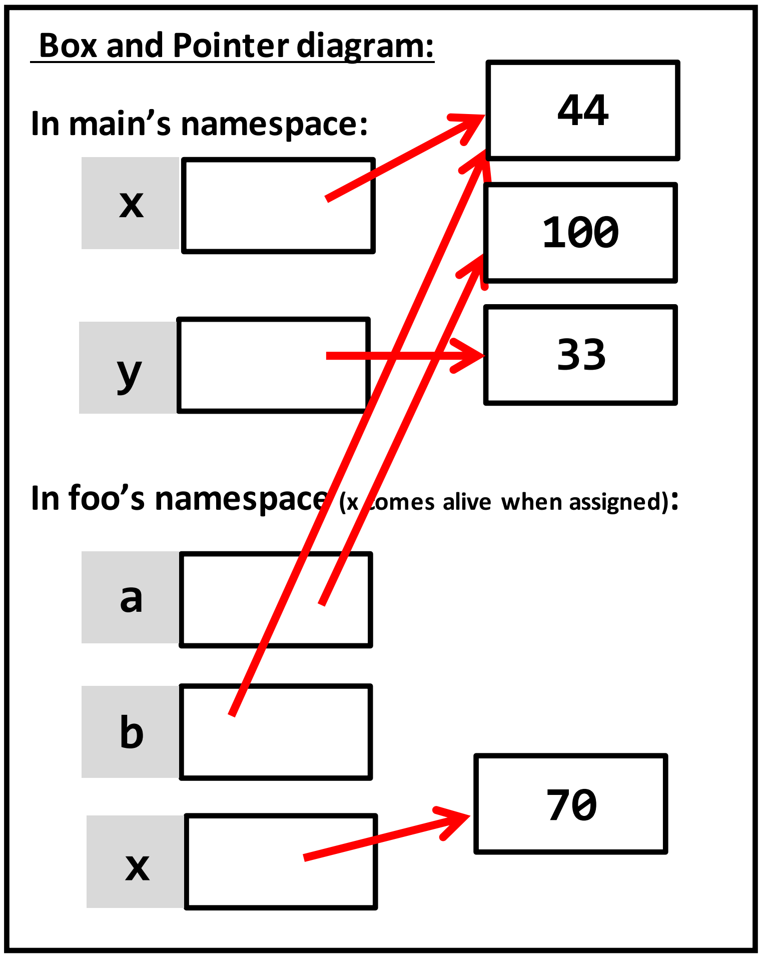
```
def main():
    x = 44
    y = 33
    foo(100, x)

def foo(a, b):
    ...
    x = 70
```

So for example, in the code snippet to the right, when function ***foo*** is called, the box-and-pointer diagram will gain 3 new variables labeled ***a***, ***b*** and ***x***, respectively. These are ***in addition to*** any variables by the same name that are in ***main***'s namespace. That is, after **foo** is called in the snippet to the right, the box-and-diagram will look like the one shown below on the LEFT, in part.

Furthermore, when a function is called, each parameter is assigned the ***value*** of the corresponding actual argument.

For example, in the code snippet above, when `foo(100, x)` executes, the parameter **a** is assigned the value **100**, just as if the statement `a = 100` were executed, and the parameter **b** is assigned the value of the variable **x**, just as if the statement `b = x` were executed. The diagram on the RIGHT shows the effects of those assignments. Study that picture carefully!





[1] This is not the entire truth, but it will do. For example, there is an exception to this regarding global variables, but we won't be using global variables, as their use is a practice that does NOT scale up to real-sized programs.

3. Draw a Box-and-Pointer diagram that shows what happens when ***main*** executes. Also indicate what output is printed, assuming appropriate ***print*** statements.

Output:

a:	44
b:	33
z:	22
p1.x:	1
p1.y:	200

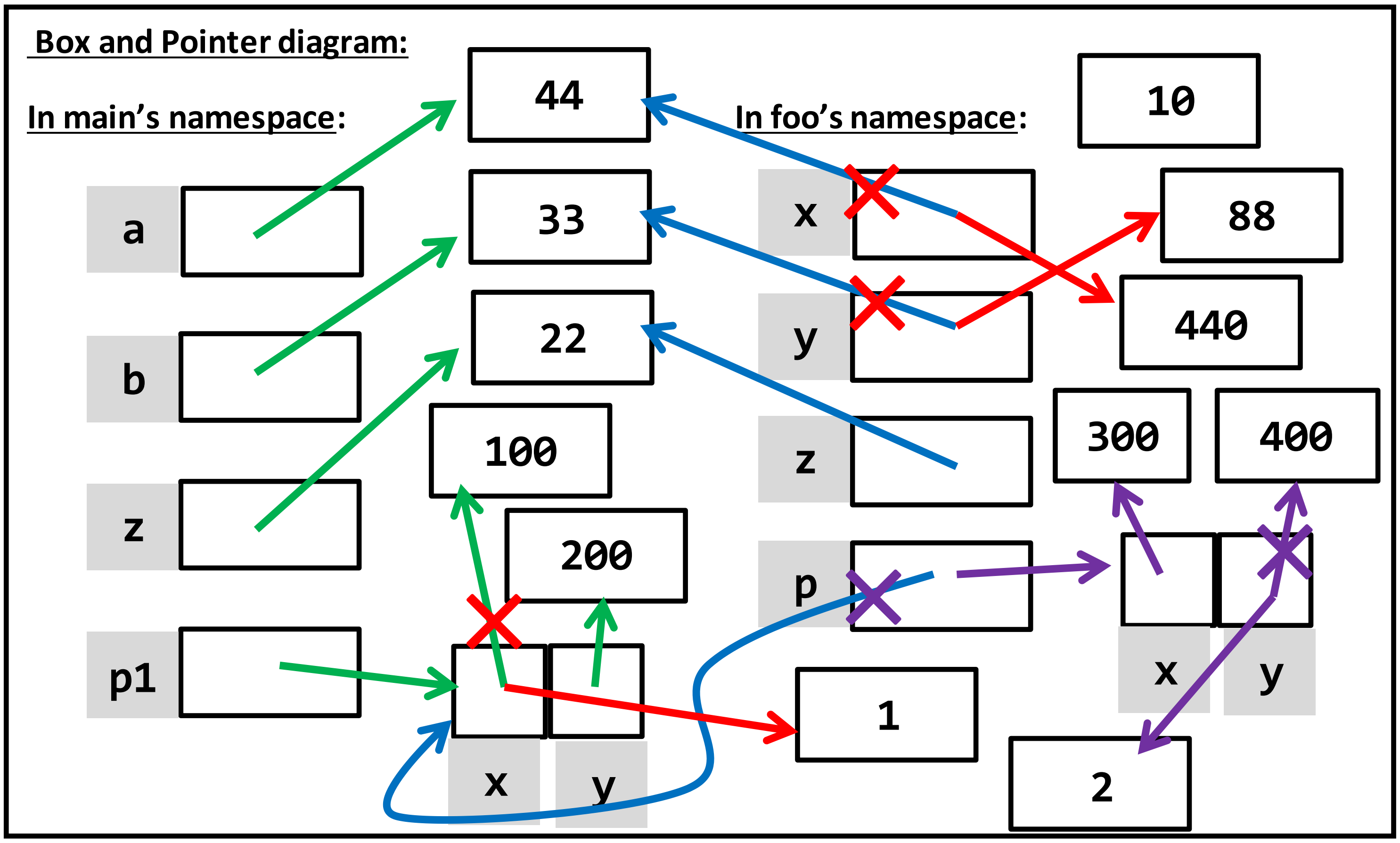
Check this problem in color-coded order: ***green*** marks, then ***blue***, then ***red***, then ***purple***. Check the output last.

As soon as there is an error, stop there and help the student walk through the code to that point. Ask the student to try again on the error, helping as needed. Let the student go forth from there on her own, re-doing the rest of the problem.

```
def main():
    a = 44
    b = 33
    z = 22
    p1 = Point(100, 200)

    foo(a, b, z, p1)

    <print statements here>

def foo(x, y, z, p):
    x = 10 * x
    y = 88
    p.x = 1
    p = Point(300, 400)
    p.y = 2
```

We have already drawn all the boxes that you need Just draw arrows (and eventually X's).

Box and Pointer diagram:


In main's namespace:
a
b
z
p1
44
33
22
100
200
x
y
In foo's namespace:
x
y
z
p
10
88
440
300
400
x
y
1
2