Test 2 – Practice Problems for the Paper-and-Pencil portion

Solution

1. 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():
    b = [44]
    a = (50, 30, 60, 77)
    x = 3

    for k in range(len(a)):
        b.append(a[x - k])
        print(k, b)

    print(a)
    print(b)
```

Output:

```
0   [44, 77]
1   [44, 77, 60]
2   [44, 77, 60, 30]
3   [44, 77, 60, 30, 50]
(50, 30, 60, 77)
[44, 77, 60, 30, 50]
```

2. Consider the following two candidate function definitions:

```
def foo():
    print('hello')
```

```
def foo(x):
    print(x)
```

(circled: `def foo(x):`)

   a. Which is “better”? Circle the better function.

   b. Explain why you circled the one you did.

   The second form allows the caller of the function to print ANYTHING, while the first is useful only for printing `'hello'`.

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

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

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

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

   Here are several examples: a *list*, a *tuple*, a *string*, a rosegraphics' *circle*, *window*, etc.

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

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

8. 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** (circled)
   (circle your choice)

9. Short answer:

   a. What is the difference between a ***class*** and an ***instance of a class*** (in other words, the difference between a ***class*** and an ***object***)?

      A *class* defines a *kind of thing*: What those things can do and what data they hold/know. An *object* (or *instance of a class*) is a *particular one* of that kind of thing, with its *particular values for its data*.

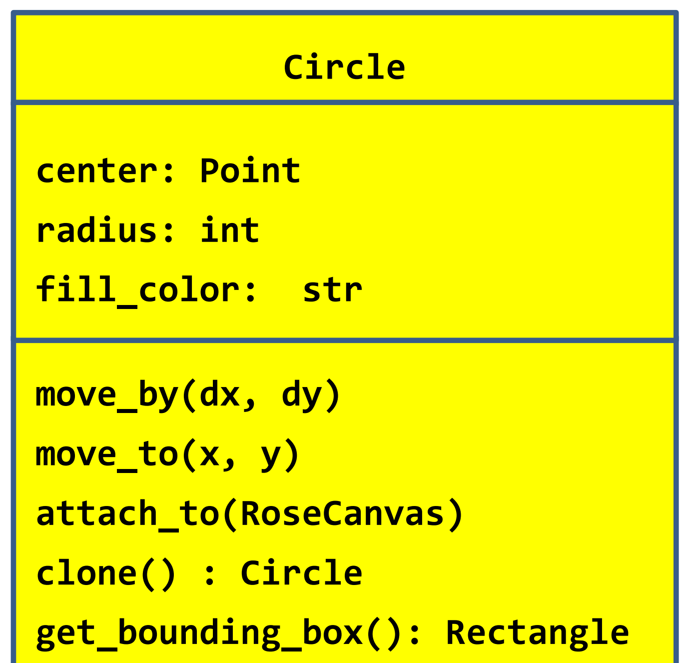
   b. Write a line or two of code that contains an example of each, clearly identifying the ***class*** and the ***object***.

```
point1 = zg.Point(100, 44)
point2 = zg.Point(33, 1900)
```

`point1` and `point2` are *objects*

They are both instances of the `zg.Point` *class*.

10. Draw a portion of the UML class diagram for Rosegraphics' ***`Circle`*** class. You don't have to get the details right (in fact, you can invent details as you wish), nor to show the entire UML class diagram – all you have to do is show that you know what it means to draw a UML class diagram.



11. Consider the following statements:

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

At this point, how many ***zg.Circle*** objects have been constructed? (circle your choice) **1** (circled) **2**

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

```
c1.radius = 77
```

After the above statement executes, the variable ***c1*** refers to the same object to which it referred prior to this statement. (circle your choice) **True** (circled) **False**

13. Continuing the previous problems:

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

14. Which of the following two statements mutates an object? (Circle your choice.)

```
numbers1 = numbers2
numbers1[0] = numbers2[0]
```

(circled: `numbers1[0] = numbers2[0]`)

15. Mutable objects are good because: **They allow for efficient use of space and hence time – passing a mutable object to a function allows the function to change the "insides" of the object without having to take the space and time to make a copy of the object. As such, it is an efficient way to send information back to the caller.**

16. Explain briefly why mutable objects are dangerous. **When the caller sends an object to a function, the caller may not expect the function to modify the object in any way. If the function does an unexpected mutation, that may cause the caller to fail. If the object is immutable, no such danger exists – the caller can be certain that the object is unchanged when the function returns control to the caller.**

17. What is the difference between the following two expressions?

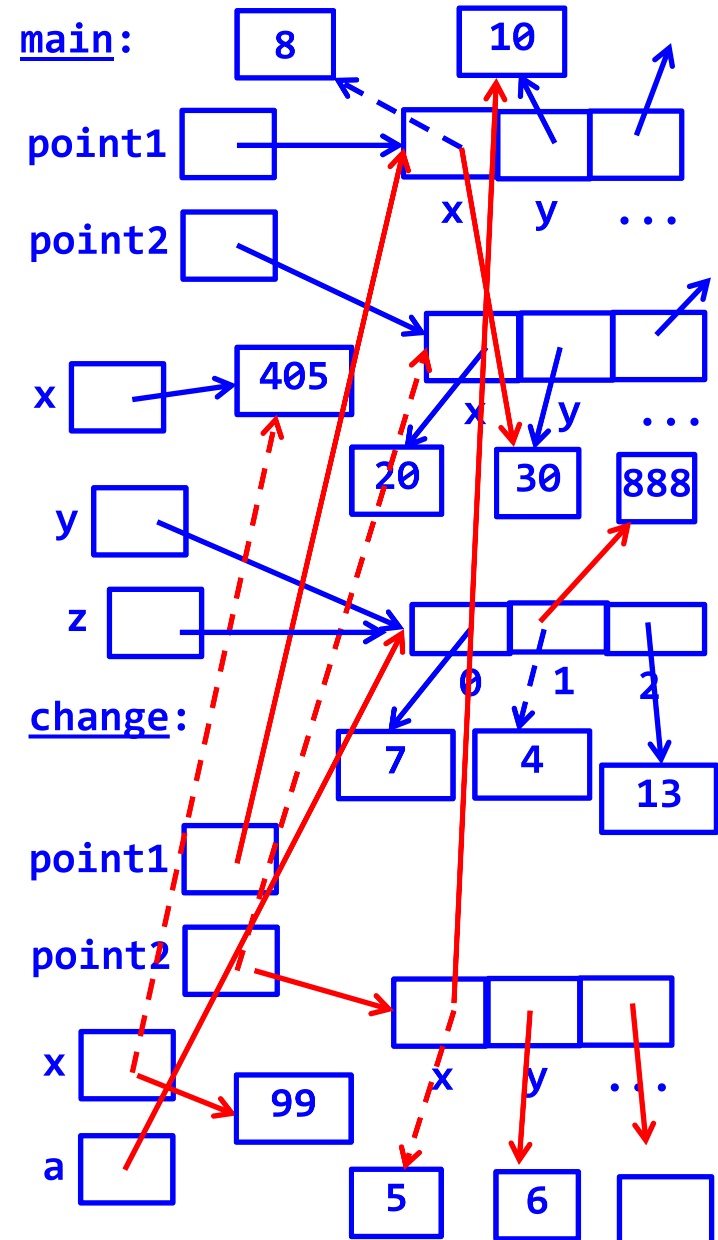
```
numbers[3]                          numbers = [3]
```

The expression on the left refers to the index 3 item in the sequence called *numbers*. It refers to that item but changes nothing (of itself). The statement on the right sets the variable called *numbers* to a list containing a single item (the number 3).

18. Consider the code in the below. To the right of the box of code, draw the ***box-and-pointer diagram*** for what happens when ***main*** runs. In the space below, show what the code would ***print*** when ***main*** runs.

```
import zellegraphics as zg

def main():
    point1 = zg.Point(8, 10)
    point2 = zg.Point(20, 30)
    x = 405
    y = [7, 4, 13]


    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):
    point1.x = point2.y
    point2 = zg.Point(5, 6)
    point2.x = point1.y
    x = 99
    a[1] = 888
    print('Within:',
          point1, point2, x, a)

    return a
```

Draw box-and-pointer diagram below here



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

```
Before:  Point(8, 10)   Point(20, 30)   405   [7, 4, 13]
Within:  Point(30, 10)  Point(10, 6)     99   [7, 888, 13]
After:  Point(30, 10)  Point(20, 30)  405  [7, 888, 13]  [7, 888, 13]
```