

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

Note: the first 3 problems review important concepts from Test 1 about *scope* and *lifetime*.

1. Consider the code snippets defined 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. Pay close attention to the order in which the statements are executed.)

```
def main():
    x = 5
    y = 3
    print('main 1', x, y)
    foo(x, y)
    print('main 2', x, y)
```

```
def foo(a, b):
    print('foo 1', a, b)
    a = 66
    b = 77
    x = 88
    y = 99
    print('foo 2', a, b,
          x, y)
```



Prints: _____

```
def main():
    x = 5
    y = 3
    print('main 1', x, y)
    foo(x, y)
    print('main 2', x, y)
```

```
def foo(x, y):
    print('foo 1', x, y)
    a = 66
    b = 77
    x = 88
    y = 99
    print('foo 2', a, b,
          x, y)
```



Prints: _____

```
def main():
    x = 5
    y = 3
    print('main 1', x, y)
    foo(y, x)
    print('main 2', x, y)
```

```
def foo(x, y):
    print('foo 1', x, y)
    a = 66
    b = 77
    x = 88
    y = 99
    print('foo 2', a, b,
          x, y)
```



Prints: _____

2. Consider the code snippet to the right. Both **print** statements are wrong.

- Explain why the first **print** statement (in **main**) is wrong.
- Explain why the second **print** statement (in **foo**) is wrong.

```
def main():  
    x = 5  
    foo(x)  
    print(z)  
  
def foo(a):  
    print(x)  
    z = 100  
    return z
```

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 it runs?

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

```
def main():
    a = alpha()

    print()
    b = beta()

    print()
    g = gamma()

    print()
    print("main!", a, b, g)

def alpha():
    print("Alpha!")
    return 7

def beta():
    print("Beta!")
    return 15 + alpha()

def gamma():
    print("Gamma!", alpha(), beta())
    return alpha() + beta() + alpha()

main()
```

Output:

4. 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.

```
b = [44]
a = (50, 30, 60, 77)
x = 3

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

print('A.', a)
print('B.', b)
print('X.', x)
```

Output:

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.

SKIP THIS PROBLEM.
It covers material for Test 3 (not 2).

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

print()

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

print('Two', y)
```

Output:

6. Consider the following two candidate function definitions:

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

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

- a. Which is “better”? Circle the better function.
- b. Briefly explain why you circled the one you did.

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:

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

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
```

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

```
numbers1 = numbers2
```

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

17. Mutable objects are good because:

18. Explain briefly why mutable objects are dangerous.

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

```
numbers[3]      numbers = [3]
```

20. 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: _____

Prints: _____

21. 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.

Here, you will implement a portion of a class called **TwoPoints**, described as follows:

- The **TwoPoints** constructor takes 2 arguments, each a **Point** object.
- The **TwoPoints** class has a method called **swap()**. It swaps the two points that a **TwoPoints** object has.
- The **TwoPoints** class has a method called **number_of_swaps()** that returns the number of times the TwoPoints object has called its **swaps()** method.

In this column, write code that would TEST the TwoPoints class.

In this column, write the IMPLEMENTATION of the TwoPoints class.

22. 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.

```
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 here:

What prints when *main* runs?

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

Before: _____

Within 1: _____

Within 2: _____

After: _____