Name: _____ **Section:** 2 (7th-8th) or 3 (9th-10th)

Use this quiz as a way to follow the lecture. ***Answer all questions.*** Make additional notes as desired. ***Not sure of an answer?*** Ask your instructor to explain further ***right then***.

Review: Functions and namespaces.

1. Functions are powerful for two reasons (fill in the blanks – you can be brief):

   a. _____

   b. _____

2. The code in the box to the right has illegal syntax – Python displays a Red X error message. Why?

```
def main():
    foo(x)

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

Review: Numbers.

3. Circle the right answers below
   (for Python – most other languages do NOT treat integers as Python does):

   a. ***Integers*** (in Python) have: **unlimited precision** **limited precision**

   b. ***Floating point numbers*** (“***floats***”) have: **unlimited precision** **limited precision**

4. Circle the right answers below:

   a. `1 + 1` evaluates to exactly the same number (in Python) as `2` ***True*** **or** ***False***

   b. `1.1 + 2.2` evaluates to exactly the same number (in Python) as `3.3`

      True **or** ***False***

5. Write the expressions that:

   a. Convert a string `s` to its equivalent ***float***
      (assuming `s` has the form of a floating point number).

   b. Convert a string `s` to its equivalent ***int***
      (assuming `s` has the form of an integer).

   c. Convert a float `f` to its representation as a string.

   d. Convert an integer `i` to its representation as a string.

6. In Python:

   a. Integer arithmetic works just like ordinary arithmetic with whole numbers, like we are accustomed to. ***True*** **or** ***False*** **(circle your choice)**

   b. Floating-point arithmetic works just like ordinary arithmetic with real numbers, like we are accustomed to. ***True*** **or** ***False*** **(circle your choice)**

7. What are some examples of floating-point arithmetic that should cause you to be particularly cautious?

8. Write expressions to:

   a. Divide the integer 7 by 2, yielding 3.5

   b. Divide the integer 7 by 2, yielding 3 (integer division)

   c. Divide the integer 17 by 4, yielding the remainder 1

   d. Divide the floats 7.0 by 2.0, yielding approximately 3.5

   e. Divide the floats 7.0 by 2.0, yielding approximately 3.0 (integer division)

Review of *Loops: Counted loops*.

9. Loops:

   a. Write a FOR statement for a ***counted loop*** that runs 45 times.

   b. Write a FOR statement for a ***counted loop*** whose index variable goes **`4, 7, 10, 13, ... 25`**.

Conditionals.

10. Complete the implementation of the following function:

```
def sq(x):
    """
    If x is non-negative, prints the square root of x.
    Otherwise, prints the square root of -x.
    """
```

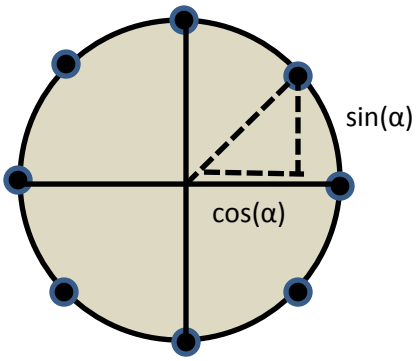
Output

Accumulators.

11. ***Summing***: Show what gets printed by the function to the right when it runs.

```
def blah():
    total = 0
    for k in range(6):
        total = total + k
        print(total)
```

12. ***Counting***: Show what gets printed by the function to the right when it runs. (The picture to the left may help.)




```
def more_blah():
    count = 0
    for k in range(11):
        x = Math.pi * k / 4
        if Math.cos(x) > 0.3:
            count = count + 1

    print(count)
```

Output

13. Your goal in this exercise is to produce a loop that produces the following table:

0	5	0	17	9000	1	101
1	6	8	217	8960	5	103
2	7	16	417	8920	25	109
3	8	24	617	8880	125	127
4	9	32	817	8840	625	181
5	10	40	1017	8800	3125	343
6	11	48	1217	8760	15625	829
7	12	56	1417	8720	78125	2287
8	13	64	1617	8680	390625	6661
9	14	72	1817	8640	1953125	19783

You will work through this problem column by column, and you will do it twice: once (in this problem) using the ***index variable*** of a loop, and again (in the next problem) using ***accumulator*** variables.

a. Code to produce the **1st** column:

b. To produce the **2nd** column:

```
for k in range(10):
    a = _____________
    print(a)
```

c. To produce the **3rd** column:

```
for k in range(10):
    b = _____________
    print(b)
```

d. To produce the **4th** column:

```
for k in range(10):
    c = _____________
    print(c)
```

e. To produce the **5th** column:

```
for k in range(10):
    d = _____________
    print(d)
```

f. To produce the **6th** column:

```
for k in range(10):
    e = _____________
    print(e)
```

g. To produce the **7th** column (too hard?):

```
for k in range(10):
    f = _____________
    print(f)
```

14. Your goal in this exercise is to produce a loop that produces the following table (same table as in the previous problem).

0	5	0	17	9000	1	101
1	6	8	217	8960	5	103
2	7	16	417	8920	25	109
3	8	24	617	8880	125	127
4	9	32	817	8840	625	181
5	10	40	1017	8800	3125	343
6	11	48	1217	8760	15625	829
7	12	56	1417	8720	78125	2287
8	13	64	1617	8680	390625	6661
9	14	72	1817	8640	1953125	19783

In the previous problem, you used the ***index variable*** of a loop. Now use ***accumulator*** variables instead of the index variable.

a. To produce the **2nd** column:

```
a = _____________
for k in range(10):
    print(a)
    a = _____________
```

b. To produce the **3rd** column:

```
b = _____________
for k in range(10):
    print(b)
    b = _____________
```

c. To produce the **4th** column:

```
c = _____________
for k in range(10):
    print(c)
    c = _____________
```

d. To produce the **5th** column:

```
d = _____________
for k in range(10):
    print(d)
    d = _____________
```

e. To produce the **6th** column:

```
e = _____________
for k in range(10):
    print(e)
    e = _____________
```

f. To produce the **7th** column:

```
total = _____________
f = _____________
for k in range(10):
    print(total)
    f = _____________
    total = _____________
```