Name: _____ SOLUTION _____

Use this quiz to help you prepare for the Paper-and-Pencil portion of Test 1. Complete it electronically or print it and complete it by hand, your choice. ***Answer all questions.*** Make additional notes as desired. ***Not sure of an answer?*** Ask your instructor to explain in class and revise as needed then.

Throughout, where you are asked to "circle your choice", you can circle or underline it (whichever you prefer).

1. Consider the ***secret*** function defined to the right. What are the values of:

```
def secret(x):
    y = (x + 1) ** 2
    return y
```

   a. **secret(2)** _____ **9** _____

   b. **secret(secret(2))** ____ **100** _____

2. Consider the ***mystery*** function defined to the right. What are the values of:

```
def mystery(x, y):
    result = x + (3 * y)
    return result
```

   a. **mystery(5, 2)** ____**11** ____

   b. **mystery(2, 5)** ____**17**_____

3. Consider the code snippets defined below. They are contrived examples with poor style but will run. For each, what does it print when *main* runs? (Each is an independent problem.)

```
def main():
    x = 5
    foo(x)
    print(x)

def foo(x):
    print(x)
    return x ** 3
```

Prints: _____ **5** _____

_____ **5**_____

```
def main():
    x = 5
    y = foo(x)
    print(y)

def foo(x):
    x = 10
    print(x)
    return x ** 3
```

_____**10**___

_____ **1000** _

```
def main():
    x = 5
    x = foo(x)
    print(x)

def foo(x):
    print(x)
    return x ** 3
```

___ **5** _____

____ **125** _____

4. What is the value of each of the following expressions?

`7 // 4`	`1`
`14 % 3`	`2` Hint: This is the REMAINDER from 14 // 3.
`3 / 4`	`0.75`
`7 % 2`	`1` Note: If x % 2 == 0, then x is EVEN. If x % 2 == 1, then x is ODD.
`7 ** 2`	`49`
`'fun' + 'ny'`	`'funny'`
`'hot' * 5`	`'hothothothothot'`

5. For each of the following code snippets, what does it print?
(Write each answer directly below its code snippet.)

```
for j in range(3):
    print((j * 2) + 1)
```

```
1
3
5
```

```
a = 10
for k in range(8):
    if k % 2 == 0:
        a = a + k
        print(k, a)
print(a)
```

```
0  10
2  12
4  16
6  22
22
```

```
b = 0
for k in range(8):
    if (k + 4) % 3 == 2:
        b = b + 1
        print(k, b)
print(b)
```

```
1  1
4  2
7  3
3
```

6. For each of the following Boolean expressions, indicate whether it evaluates to ***True*** or ***False*** (circle your choice):

True	(***False***)	`not (5 < 7)`
True	(***False***)	`(7 < 5) or not (5 < 7)`
(***True***)	***False***	`(x < 0) or (x >= 0)`
(***True***)	***False***	`(6 <= 6) or (3 == 2)`
True	(***False***)	`(6 <= 6) and (3 == 2)`
True	(***False***)	`not not False`

7. What gets printed when ***main*** is called in the program shown to the right? (Pay close attention to the order in which the statements are executed. ***Write the output in a column to the left of the program.***)

Output

2 3

2 3

2 3

```
def main():
    a = 2
    b = 3

    foo1()
    print(a, b)

    foo2(a, b)
    print(a, b)

    foo3(a, b)
    print(a, b)

def foo1():
    a = 88
    b = 99

def foo2(a, b):
    a = 400
    b = 500

def foo3(x, y):
    x = 44
    y = 55
```

8. True or False: As a ***user*** of a function (that is, as someone who will ***call*** the function), you *don't need to know how the function is* ***implemented***; you just need to know the ***specification*** of the function. **True** **False** (circle your choice)

9. List ***two*** reasons why functions are useful and important.

Reason 1: **They help organize the code, which makes it easier to get correct and maintain.**

Reason 2: **They allow for code re-use, by allowing the function to be called multiple times with different values for the parameters.**

10. ***float*** versus ***int***:

a. Write two Python constants – one an integer (**`int`**) and one a floating point number (**`float`**) – that clearly shows the difference between the **`int`** and **`float`** types.

82 **13.793** **[the *float* has a decimal point]**

b. A Python **`int`** can have an arbitrarily large number of digits. ***True*** ***False***
(circle your choice)

c. A Python **`float`** can represent an arbitrarily large number. ***True*** ***False***
(circle your choice)

d. There is a limit to the number of significant digits a Python **`float`** can have. ***True*** ***False***
(circle your choice)

11. ***int*** versus ***str***: What does each of the following code snippets print or cause to happen?
(Write each answer to the side of its code snippet.)

```
x = '5'
print(x * 3)
```

555

```
y = int('5')
print(y * 3)
```

15

Hint for the above: The int function converts a string argument to the equivalent integer. For example, int(8) evaluates to the INTEGER 8.

```
z = '5'
print(z / 3)
```

Raises an exception (error). That is, it causes the program to break at the attempt to do division on a string.

12. Does the following function meet its specification? If not, why not?

```
def get_number(x):
    """
    Returns x squared plus x cubed, for the given x.
    For example, if x is 5, returns (5 ** 2) + (5 ** 3), which is 150.
    """
    answer = (x ** 2) + (x ** 3)
    print(answer)
```

No – it does NOT meet its specification.

Its specification says to RETURN the answer, *not* PRINT it.

13. Does the following function meet its specification? If not, why not?

```
def get_number(x):
    """
    Returns x squared plus x cubed, for the given x.
    For example, if x is 5, returns (5 ** 2) + (5 ** 3), which is 150.
    """
    answer = (x ** 2) + (x ** 3)
    print(answer)
    return answer
```

No – it does NOT meet its specification.

Its specification does NOT say to PRINT anything, so doing so violates the specification. Printing is a SIDE-EFFECT – a function must have no side-effects beyond what the specification specifies.

(Continued on the next page)

14. Does the following function meet its specification? If not, why not?

```
def test_get_number(x):
    """ Tests the   get_number   function. """
    answer1 = get_number(5)
    answer2 = get_number(1)
    answer3 = get_number(2)
```

No – it does NOT meet its specification.

Its specification says to TEST the function. The code CALLS the function (good!), but does nothing with the returned value. As such, it does not TEST whether the returned value is correct.

Testing the returned value requires either printing it (so that the human user can check whether or not the returned value is correct) or otherwise checking the returned value (e.g., by comparing the returned value to the correct answer and printing an appropriate message as a result).

Furthermore, we will also require that you print the EXPECTED value to be returned, so that you can demonstrate that you really did have something to check the answer against.

(Continued on the next page)

15. Consider a function whose name is ***print_string*** that takes two arguments as in this example:

```
print_string('Robots rule!', 4)
```

The function should print the given string the given number of times. So, the above function call should produce this output:

```
Robots rule!
Robots rule!
Robots rule!
Robots rule!
```

Write (in the space to the right) a complete implementation, ***including the header (def) line,*** of the above ***print_string*** function.

```
def print_string(s, n):
    for k in range(n):
        print(s)
```

A better answer might choose better names for `s` and `n` (e.g. `string_to_print` and `times_to_print`), but the answer above is acceptable in this context.

16. Each of the 5 items listed below has one or more examples of it in the code shown to the right.

```
1 import rosegraphics as rg
2 import math
3 window = rg.RoseWindow(500, 250)
4 radius = math.sin(2)
5 color = 'blue'
6 point = rg.Point(1, 2)
7 circle = rg.Circle(point, radius)
8 circle.fill_color = color
9 circle.attach_to(window)
```

For each of the 5 items listed below, find ***ONE*** example of that item in the code. Then write ***the line number*** on which your example of that item appears, and ***circle on that line*** your example of that item. (It is possible that one line has examples of multiple items, so you might have more than one circle on a line --- that is OK.)

a. **Object.** Appears on line number __ **3** __ **(also on lines 6, 7 and elsewhere)**

b. **Method.** Appears on line number __ **9** __

c. **Instance variable** (aka **field**). Appears on line number __ **8** __

d. **Function.** Appears on line number __ **4** __

e. **Variable** (aka **local variable**). Appears on line number __ **5** __ **(and other places too).**