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).

Throughout, assume that there are no global variables (if you happen to know what they are).

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** ____

   c.
```
x = 2
y = 5
mystery(x, y)
```
   _____ **17** ____

   d.
```
x = 2
y = 5
mystery(y, x)
```
   _____ **11** ____

3. Consider the code snippet to the right. Explain briefly why there is a ***red X*** beside ***Line 3***.

```
1  def main():
2      foo(10)
3      print(n)
4
5
6  def foo(m):
7      n = m + 20
8      return n
```

The variable *n* in *main* is undefined. That is, it has no value when Line 3 executes.

The presence of a variable *n* in *foo* is irrelevant. Names defined inside one function are *independent* of names defined in other functions.

4. Consider the code snippet to the right. Explain briefly why there is a ***red X*** beside ***Line 7***.

```
1  def main():
2      a = 3
3      foo()
4
5
6  def foo():
7      b = a + 7
8      print(7)
```

The variable *a* in *foo* is undefined. That is, it has no value when Line 7 executes.

The presence of a variable *a* in *main* is irrelevant. Names defined inside one function are *independent* of names defined in other functions.

5. 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. Pay close attention to the order in which the statements are executed.)

```
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

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

`17 // 4` = 4 — Hint: This is a WHOLE number (i.e., integer).

`17 % 4` = 1 — Hint: This is the REMAINDER from `17 // 4`.

`3 / 4` = 0.75

`7 % 2` = 1 — Aside: 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'`

`'fun' + 3` — This is not a legal expression. It breaks when it runs.

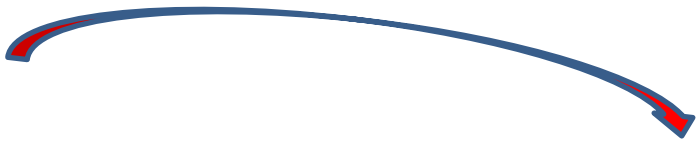
`10 ^ 2` — This does NOT evaluate to 100. The `^` (caret) symbol does NOT mean exponentiation (raising to a power) in Python. It has an entirely different meaning that is not important to our current work.[1] We won't ask you what `^` means on the test, but it is important to know that `^` is NOT exponentiation.

[1] But just in case you are curious, here is what it does mean: bitwise exclusive-OR. Since `10` is `0110` in binary and `2` is `0010` in binary and `0110` bitwise exclusive-OR'ed with `0010` is `0100`, which is `8` in decimal, `10 ^ 2` evaluates to `8`.

7. For each of the 3 code snippets below, what does it print?
(Write each answer directly below its code snippet.)

Hint: Solve problems like this by make a ***table with the variables, showing the places where their values change.*** Here is an example of a table appropriate for the **3rd (rightmost)** problem. It was made by tracing the code by hand, starting from line 1 of the table (which came from the statement **`b = 0`**) and continuing downward from there as the by-hand trace continues.

k	b
	0
0	
1	1
2	
3	
4	2



```
for j in range(4):
    print((j * 2) + 1)
```

```
1
3
5
7
```

```
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(5):
    if (k + 4) % 3 == 2:
        b = b + 1
        print('b is:', b)
    print(k, b)
print(b)
```

```
0  0
b  is: 1
1  1
2  1
3  1
b  is: 2
4  2
2
```

8. 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
```

9. 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)

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

Reason 1: **They help *organize* the code, which makes it easier to get the code correct when writing it and to maintain that code's correctness as changes are made later in the lifetime of the software.**

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

11. 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?

```
def main():
    one()
    two()
    three()


def one():
    print('One!')
    return 1 + two()


def two():
    print('Two!')
    return 1
    print('Done!')


def three():
    print('Three!', two(), one())


main()
```

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

Output:

```
One!
Two!
Two!
Two!
One!
Two!
Three! 1 2
```

Here is an explanation of what happens in the above:

1. The definitions are all read, then ***main*** is called at the bottom of the code.
2. ***main*** calls ***one***.
3. ***one*** prints `One!` and then calls ***two***.
4. ***two*** prints `Two!` and then returns **1**. (The `print('Done!')` statement is never reached, since a ***return*** statement really ***leaves the function***, returning to its caller.)
5. Control returns to ***one***, where the returned **1** is added to the **1** in `return 1 + two()`, yielding **2**, so **2** is returned from the ***one*** function, back to ***main***.
6. ***main*** ignores the returned value from ***one*** and calls ***two***.
7. ***two*** prints `Two!` and returns **1**. (Again, the `print('Done')` is never reached.)
8. ***main*** ignores the returned value from ***two*** and calls ***three***.
9. ***three*** calls ***two***. two prints `Two!` and returns **1** back to ***three***.
10. ***three*** calls ***one***. ***one*** prints `One!`, calls ***two*** which prints `Two!` and returns **1** to ***one***, ***one*** adds the returned **1** to **1** and returns **2** to ***three***.
11. ***three*** has now computed the values of the 3 arguments to its ***print*** statement and prints them: `Three! 1 2`.

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.

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. (This explanation continues on the next page)

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.

Finally, if you simply *RUN* your function and *THEN* provide the "expected value" as the value that your function produces, that is NOT A TEST and you will get NO CREDIT for doing so.

You MUST have tests that are either *GIVEN* to you by us (possibly as an example in the specification, possibly in the testing code) or *COMPUTED BY HAND* by you.

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.

Answer:

```
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.