

Name: _____

Use this quiz to help you prepare for the Paper-and-Pencil portion of Exam 1. Print it and write your answers directly on the printed copy, or read the electronic copy and write your answers on your own document – 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)` _____
- b. `secret(secret(2))` _____

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)` _____
- b. `mystery(2, 5)` _____
- c. `x = 2`
`y = 5`
`mystery(x, y)` _____
- d. `x = 2`
`y = 5`
`mystery(y, x)` _____

3. Consider the *secret* and *mystery* functions defined above. What are the values of:

- a. `x = 2`
`y = 5`
`secret(3) + mystery(x, y)` _____
- b. `secret(mystery(2, 1))` _____
- c. `x = 2`
`y = 1`
`mystery(secret(x), secret(y))` _____

4. 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
    
```

5. 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)
    
```

6. 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
    foo(x)
    print(x)

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

```

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

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

```

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

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

Prints: _____

Prints: _____

Prints: _____

```
def main():
    a = 2
    b = 3
    c = 44
    d = 55

    foo1(a, b)
    ##### Location 1

    foo2(c, d)
    ##### Location 2

    r = 25
    s = 35
    r = foo3(r, s)
    ##### Location 3

def foo1(a, b):
    ##### Location 4
    a = 88
    b = 99
    ##### Location 5

def foo2(x, y):
    ##### Location 6
    a = 400
    b = 500
    y = 600
    ##### Location 7

def foo3(s, r):
    ##### Location 8
    return r + s

main()
##### Location 9
```

7. Consider the code to the left. It is a contrived example with poor style but will run without errors. In this problem, you will trace the execution of the code. **As each location is encountered during the run:**

1. **CIRCLE** each variable that is *defined* at that location.
2. **WRITE** the **VALUE** of each variable that you *circled* directly **BELOW** the circle.

For example, the run defines the functions and then calls *main*, as usual. The first of the nine locations **to be encountered** is **Location 4**. At Location 4, the only variables defined are *a* and *b*, with values **2** and **3** at that point of the program’s run. So, on the row for Location 4, I have circled *a* and *b* and written their values at Location 4 directly below them.

Note that you fill out the table in the order that the locations are encountered, **NOT from top to bottom**. **ASK FOR HELP IF YOU DO NOT UNDERSTAND WHAT THIS PROBLEM ASKS YOU TO DO.**

Location 1	a	b	c	d	r	s	x	y
Location 2	a	b	c	d	r	s	x	y
Location 3	a	b	c	d	r	s	x	y
Location 4	a	b	c	d	r	s	x	y
	2	3						
Location 5	a	b	c	d	r	s	x	y
Location 6	a	b	c	d	r	s	x	y
Location 7	a	b	c	d	r	s	x	y
Location 8	a	b	c	d	r	s	x	y
Location 9	a	b	c	d	r	s	x	y

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

$17 // 4$ _____

Hint: This is a **WHOLE** number (i.e., integer).

$17 \% 4$ _____

Hint: This is the **REMAINDER** from $17 // 4$.

$3 / 4$ _____

$7 \% 2$ _____

Aside: If $x \% 2 == 0$, then x is **EVEN**.
If $x \% 2 == 1$, then x is **ODD**.

$7 ** 2$ _____

'fun' + 'ny' _____

'hot' * 5 _____

'fun' + 3 _____

Hint: This is a trick question.

$10 ^ 2$ _____

Hint: This is a trick question.

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

Reason 1: _____

Reason 2: _____

10. Assume that you have a variable (name) **x** that is a positive integer. Write a snippet of code that prints **30** if **x** is odd.

11. Continuing the previous problem, write a snippet of code that prints **30** if **x** is odd and prints **22** if **x** is even.

12. Continuing the previous problem, write a snippet of code that prints **30** if **x** is odd and prints **'hello'** if **x** is greater than or equal to **100**. Note that when **x** is (for example) **151** this snippet would print both **30** and **'hello'**.

13. Write a snippet of code that would construct an **rg.Point** object at **(300, 444)** and give the **rg.Point** object a name. You can choose any reasonable name that you like.

14. Assume that you have a variable (name) **p2** that is an **rg.Point** object. Write a snippet of code that would triple the y-coordinate of **p2**.

15. Assume that you have a variable (name) **p2** that is an **rg.Point** object and a variable named **x** that is a floating point number. Write a snippet of code that would increase the x-coordinate of **p2** by **x**. (Note: using **x** as a variable name here is a poor choice, but solve the problem as written anyhow.)

16. Assume that you have a variable (name) **circle9** that is an **rg.Circle** object. Write a snippet of code that would make the radius of **circle9** decrease by **30**.

17. Continuing the previous problem, write a snippet of code that would make the radius of **circle9** decrease by **30** unless doing so would make that radius less than **1**, in which case the code should make the radius be **1**.

18. Assume that you have a variable (name) **rectangle** that is an **rg.Rectangle** object. Write a snippet of code that would use its **get_upper_left_corner** method to print the rectangle's upper-left corner.
19. Assume that you have a name (variable) **findo** that refers to a **Dog** object. Assume further that **Dog** objects have a **bark** method that takes as an argument the number of times to bark. Write a statement that would make **findo** bark 5 times.
20. Write a snippet of code that would print the following numbers (but each on its own line):

5 8 11 14 17 20 23 26 29 32 35

Note: No credit for just 11 **print** statements, that is, for
`print(5) print(8) print(11) ... print(35).`

21. Assume that you have a name (variable) **win3** that is an **rg.RoseWindow** object, and also names (variables) **r** and **s** that are positive, even integers, with **s > r**. Write a snippet of code that would construct **rg.Circle** objects, with each centered at **(300, 200)**, but with radii that are:

r r + 2 r + 4 r + 6 ... s

For example, if **r** is **8** and **s** is **14**, your code must construct **4** **rg.Circle** objects, with radii **8, 10, 12, and 14**, respectively.

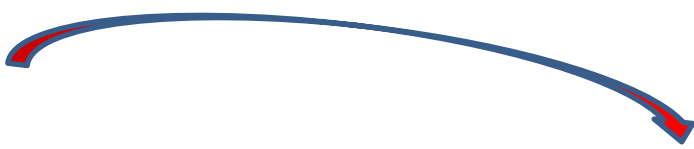
Your code must also attach each **rg.Circle** that it constructs to **win3**.

Write code for the generic case for **r** and **s**. That is, use the names (variables) **r** and **s** in your code. You should use a **range** statement in your solution. You may NOT use the multiple-argument form of **range** in this problem. That is, the **range** expression in your solution must have only a **single** argument.

22. 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.

<u>k</u>	<u>b</u>
	0
0	
1	1
2	
3	
4	2



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

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

```
b = 0  
for k in range(5):  
    if (k + 4) % 3 == 2:  
        b = b + 1  
        print('b is:', b)  
    print(k, b)  
print(b)
```


23. What gets printed when *main* is called in the program shown to the right?

Write the output in the provided box.

Output

```
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
```

24. Consider the snippet of code shown to the right. Circle whichever of the following is correct, assuming that **foo_11** and **foo_12** were written as they appear in the snippet intentionally.

- Line 44 is sensible but Line 47 is not.
- Line 47 is sensible but Line 44 is not.
- Both Line 44 and Line 47 are sensible.
- Neither Line 44 nor Line 47 is sensible.

```
43 def blah():  
44     x = foo_11(3, 7)  
45     print(x)  
46  
47     y = foo_12(3, 7)  
48     print(y)  
49  
50  
51 def foo_11(a, b):  
52     print(a * b)  
53  
54  
55 def foo_12(a, b):  
56     return (a * b)  
57
```

25. 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? (Consider using **post-it notes** as was done in one of the videos you watched on function calls.)

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

```
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()
```

Output:

26. 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)

27. Does the function definition shown to the right 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)
```

28. Does the function definition shown to the right 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
```

29. Does the function definition shown to the right 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)
```

30. 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.

31. Assume that you have a function `is_perfect` whose specification is as shown to the right.

```
def is_perfect(m):  
    """  
    What comes in: an integer m.  
    What goes out: Returns True if the argument m  
        is "perfect". Returns False otherwise.  
    """
```

Consider a function whose name is `add_them` that takes two integer arguments m and n (with $m \leq n$) and returns the sum of the integers from m to n , inclusive, that are NOT perfect. Write (in the space below) a complete implementation, *including the header (def) line*, of the `add_them` function. Note that you do not need to know what makes a number “perfect” to solve this problem.