All written answers are limited to their question boxes. Make sure *all answers are easily legible.*

1. (1 point) Print your name and email id.

2. (2 points) What makes functions so important?

   Ability to reuse code in a parameterized way

3. (2 points) What is the role of parameters in a function definition?

   Receive information into a function so that it's execution can be tailored appropriately

4. (2 points) What is the role of arguments in a function invocation?

   Send information into a function so that it's execution can be tailored appropriately

5. (2 points) What is the role of a return statement in a function definition?

   Return information from on what it accomplished

6. (2 points) What is the role of local variables in a function definition?

   Assist with the function's computation. BTW: they only exist during the execution and there are new ones every invocation

7. (2 points) Why or why not does every Python function invocation produce a return value?

There is a default return value of None

8. (2 points) Why is a function unable to affect the values of its arguments?

A copy of the values of the arguments are passed so the function manipulates the copy rather than arguments themselves

9. (2 points) Why is a function able to affect the contents of the list objects that its invocation arguments refer?

Although a parameter is a copy it points to the actual list and can do all list operations on it.

10. (2 points) What should the comment be to describe the action of function f() with dict parameter d, and string parameters k and v?

```
def f( d, k, v ) :
    if ( k not in d.keys() ) :
        d[ k ] = v
```

Adds the mapping k → v if there is not already a mapping from k in the dict

11. (2 points) Consider function f( ) where its parameter is a list. Suppose t equals [1, 4, 9]. What is the value of t after the invocation f(t) completes?

```
def f( x ) :
    y = x
    y.append( 1 )
```

[1, 4, 9, 1]

12. (2 points) What should the comment be describing function f() with its nonempty list parameter x?

```
def f( x ) :
    m1 = min( x )
    m2 = max( x )
    if ( m1 == m2 ) :
        return True
    else :
        return False
```

Returns whether the elements of the x are identical.

13. (11 points) Develop module *m1.py*. The module defines function t(). Function `t()` has no parameters and returns the logical (**`bool`**) value **`True`**. A run of tester *tester1.py* should produce output

```
t() = True
t() = True
t() = True
```

```
def t() :
    return True
```

14. (11 points) Develop module *m2.py*. The module defines function c(). Function `c()` takes one numeric parameter *r*. The function returns the circumference of a circle with radius *r*; that is, $2 \pi r$. A run of tester *tester2.py* should produce output

```
c( 3 ) = 18.84955592153876
c( 1 ) = 6.283185307179586
c( 4 ) = 25.132741228718345
```

```
import math
def c( r ) :
    return 2 * math.pi * c
```

15. (11 points) Develop module *m3.py*. The module defines a function `eval()`. Function `eval()` takes three parameters *a*, *op*, and *b*. If *op* equals the string '+', the function returns *a* + *b*. If instead *op* equals the string '-', the function returns *a* - *b*. If instead *op* equals the string '*', the function returns *a* * *b*. If instead *op* equals the string '/', the function returns *a* / *b*. Otherwise, the function returns None. A run of tester *tester3.py* should produce output

```
eval( 3, '+', 4 ) = 7
eval ( 4, '-', 5 ) = -1
eval ( 7, '*', 9 ) = 63
eval ( 5, '/', 2 ) = 2.5
eval ( 3, '?', 4 ) = None
```

```
def eval( a, op, b ) :

    if ( op == '+' ) :
        result = a + b
    elif ( op == '-' ) :
        result = a - b
    elif ( op == '*' ) :
        result = a * b
    elif ( op == '/' ) :
        result = a / b
    else :
        result = None

    return result
```

16. (11 points) Develop module *m4.py*. The module defines function *odds*(). Function *odds*() takes one integer parameter *n*. The function returns a new list whose elements are the first *n* odd integers. You can assume *n* is not negative. A run of tester *tester4.py* should produce output

```
odds( 3 ) = [ 1, 3, 5 ]
odds( 0 ) = []
odds( 9 ) = [ 1, 3, 5, 7, 9, 11, 13, 15, 17 ]
```

```
def odds( n ) :

    result = []
    for i in range( 0, n ) :
        odd_i = 2*i + 1 )

        result.append( odd_i )

    return result
```

17. (11 points) Develop module *m5.py*. The module defines function *counts*(). Function *counts*() takes one list parameter *x*. The function returns a new list with three elements, where the first element is the number of negative numbers in *x*, the second element is the number of 0's in *x*, and the third element is the number of positive numbers in *x*. A run of tester *tester5.py* should produce output

```
counts( [ -3, -1, 5, -5, -8, 3, 4 ] ) = [4, 0, 3]
counts( [ 8, -9, 0, 4, -3, 1, -8, 0, 2 ] ) = [3, 2, 4]
counts( [ 2, -1, -4, -2, 1, 6, 0, 4, 2 ] ) = [3, 1, 5]
```

```
def counts( x ) :

    nn = 0
    nz = 0
    np = 0

    for v in x :
        if ( v < 0 ) :
            nn = nn + 1
        elif ( v > 0 ) :
            np = np + 1
        else :
            nz = nz + +1

    return [ nn, nz, np ]
```

18. (11 points) Develop modue *m6.py*. The module defines function *gt*(). Function *gt*() takes two list parameters *x* and *y*. You can assume the lists are of the same length. The function returns a new list of that same length, where the first element of the new list is the max of the first elements of *x* and *y*, the second element of the new list is the max of the second elements of *x* and *y*, and so on. A run of tester *tester6.py* should produce output

```
x1 = [ -3, -1,  5, -5, -8,  3,  4 ]
y1 = [  8, -9,  0,  4, -3,  1, -8 ]

x2 = [ 'a', 'd',   'b', 'x' ]
y2 = [ 'b', 'abc', 'b', 'v' ]

x3 = [ -3, -1,  5, -5, -8,  3,  4, 'a', 'd',   'b', 'x' ]
y3 = [  8, -9,  0,  4, -3,  1, -8, 'b', 'abc', 'b', 'v' ]

gt( x1, y1 ) = [ 8, -1, 5, 4, -3, 3, 4 ]
gt( x2, y2 ) = [ 'b', 'd', 'b', 'x' ]
gt( x3, y3 ) = [ 8, -1, 5, 4, -3, 3, 4, 'b', 'd', 'b', 'x' ]
```

```
def gt( x, y ) :

    n = len( x )

    result = []
    for i in range( 0, n ) :
        vx = x[ i ]
        vy = y[ i ]

        vr  = max( vx, vy )

        result.append( vr )

    return result
```

19. (11 points) Develop module *m7.py*. The module defines function *trans*(). Function *trans*( ) takes two parameters, *d* and *x*, where *d* is a dict and *x* is a list. The function returns a new list *y*, where if *x* has an item *v*, the corresponding item in *y* has value *d*[*v*]. A run of tester *tester7.py* should produce output

```
d1 = { 1 : 'a', 2 : 'b', 3 : 'c', 4 : 'd' }
d2 = { 'zero' : 0, 'one' : 1 }
d3 = { 1 : 1, 2 : 4, 3 : 9, 4 : 16, 5 : 25 }

x1 = [ 2, 2, 3, 1, 2, 4, 2 ]
x2 = [ 'zero', 'zero', 'one', 'one', 'zero', 'zero', 'zero' ]
x3 = [ 4, 5, 5, 1, 3 ]

trans( d1, x1 ) =  [ 'b', 'b', 'c', 'a', 'b', 'd', 'b' ]
trans( d2, x2 ) =  [ 0, 0, 1, 1, 0, 0, 0 ]
trans( d3, x3 ) =  [ 16, 25, 25, 1, 9 ]
```

```
def trans( d, x ) :

    result = []

    for xv in x :
        rv = d[ xv ]

        result.append( rv )

    return result
```

Signature:

Pledge: On my honor, I pledge that I have neither given nor received help on this test.

Test rules

- You may use a single page of notes. The only device you may access during the test is your laptop.
- Do not access class examples or your own past assignments during the test; that is, the only code you may access or view are ones that you develop for this test.
- The only computer windows to be open are PyCharm and a browser that only accesses class website links.
- PyCharm ***cannot be used*** for the short answer questions.
- Code should demonstrate follow style rules; e.g., header comments, whitespace, identifier naming, etc.
- Whether a function is runnable is important.
- Unless indicated, functions should not modify their parameters in any way.