Name: 
Email id: 

Notices

• Based on your past educational achievements, I expect you to do well on this test.
• Answer the questions in any order that you want.
• Hand in both parts of the test.

Test rules

• Check before you leave the room, that you uploaded all of your solutions. Do not ask afterwards whether you can submit a forgotten solution.
• This pledged exam is closed notes. The only device you may access during the test is your laptop.
• Uploading after you leave the room means you are withdrawing from the class.
• Do not access class examples, web solutions, 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 windows to be open on your computer are PyCharm and a single browser with tabs reachable from the class website.
• PyCharm can be used for developing the modules to be submitted. It cannot be used for the short answer questions.
• With regard to your functions:
  o Comments including header identifying comments are not necessary.
  o You should follow other class style practices; e.g., whitespace, identifier naming, etc.
  o Only do what is requested.
  o None of the functions should get input or produce output.
  o Functions should not modify their parameters in any way.
  o Whether a function is testable is important.
• Any form of cheating on a test can result in expulsion from the class and the incident being referred to the Honor Committee.
THIS PAGE IS ALMOST BLANK
Part I. Function implementation

1. (13 points) Module `luna.py` defines a function `h()`. The function has a single numeric parameter `x`. The function returns the number of hours it would take to get the moon while traveling at `x` miles per hour. A simple tester `ltest.py` is available. For your information: \( distance = speed \times elapsed \ time. \)
   - The module also defines the constant
     \[ \text{DISTANCE-IN-MILES-TO-MOON} = 238900.0 \]
   - The output of the tester should be
     \[
     \begin{align*}
     h(119.45) &= 2000.0 \\
     h(597.25) &= 400.0 
     \end{align*}
     \]

2. (13 points) Module `calc.py` defines a function `e()`. The function has three parameters `x`, `y`, and `s`. Parameters `x` and `y` are decimals; parameter `s` is a string. A simple tester `ctest.py` is available.
   - If `s` is either `'+', '-', '*'`, or `'/'`, then the function returns respectively `x + y`, `x - y`, `x * y`, or `x / y`. Otherwise, the function `None`. The output of the tester should be
     \[
     \begin{align*}
     19.5 + 5.25 &= 24.75 \\
     12.5 - 6.5 &= 6.0 \\
     12.5 \times 4.5 &= 56.25 \\
     10.0 / 2.25 &= 4.444444444444445 \\
     1.0 \oplus 5.0 &= \text{None}
     \end{align*}
     \]

3. (13 points) Module `eval.py` defines a function `f()`. Function `f()` has two list parameters `x` and `y`. The function returns a new list whose elements are the elements of `x` followed by the elements of `y`. The function does not change its list parameters. A simple tester `etest.py` is available. The tester makes use of the following lists.
   - The output of the tester should be
     \[
     \begin{align*}
     x1 &= [ ]; \\
     x2 &= [3, 1, 4 ]; \\
     x3 &= [ ]; \\
     x4 &= [3, 1, 4 ]; \\
     y1 &= [ ]; \\
     y2 &= [ ]; \\
     y3 &= [2, 7, 8] \\
     y4 &= [1, 5, 1, 9]
     \end{align*}
     \]
     \[
     \begin{align*}
     f( x1, y1 ) &= [ ] \\
     f( x2, y2 ) &= [3, 1, 4] \\
     f( x3, y3 ) &= [2, 7, 8] \\
     f( x4, y4 ) &= [3, 1, 4, 1, 5, 1, 9]
     \end{align*}
     \]
4. (13 points) Module update.py defines a function \( g() \). Function \( g() \) has one list parameter \( x \). The function returns a new list whose elements are the element values of \( x \) without duplication. The function does not change its list parameter. A simple tester utest.py is available. The tester makes use of the following lists.

\[
\begin{align*}
\text{x1} &= [0, 1, 2] \\
\text{x2} &= [0, 4, 1, 2, 2, 1, 3, 6, 3, 3, 4] \\
\text{x3} &= [
\end{align*}
\]

- The output of the tester should be

\[
\begin{align*}
g(\text{x1}) &= [0, 1, 2] \\
g(\text{x2}) &= [0, 4, 1, 2, 3, 6] \\
g(\text{x3}) &= [
\end{align*}
\]

5. (13 points) Module sigma.py defines a function \( s() \). The function has one parameter \( d \). Parameter \( d \) is an already initialized integer dataset; that is, it is a list of integer lists. The function returns the sum of the dataset values. The function does not change its list parameter. A simple tester dtest.py is available. The tester makes use of the following datasets.

\[
\begin{align*}
\text{d1} &= [[0], [1, 2], [1, 2, 3], [0]] \\
\text{d2} &= [[1, 0, 1, 2, 2], [3, 0, 1, 1, 0], [2], [0, 0, 1]] \\
\text{d3} &= [[3, 0, 3], [3, 0, 3, 0, 1], [1, 0, 2]] \\
\text{d4} &= [\n\end{align*}
\]

- The output of the tester should be

\[
\begin{align*}
s(\text{d1}) &= 9 \\
s(\text{d2}) &= 15 \\
s(\text{d3}) &= 16 \\
s(\text{d4}) &= 0
\end{align*}
\]

6. (13 points) Module trio.py defines a function \( t() \). The function has one list parameter \( x \) of numeric values. The function does not change its list parameter. The function returns a three-element list whose values are respectively the number of negative, zero, and positive values in \( x \). A simple tester ttest.py is available. The tester makes use of the following lists.

\[
\begin{align*}
\text{x1} &= [0, -3, 0, -4, -2] \\
\text{x2} &= [-3, 1, -2, 1, -3, -3, -2, -4, -1, -4] \\
\text{x3} &= [2, -1, 0, 3, 0, 3, -2, -2, -1, -4, 3, -4, 3, -1, 3] \\
\text{x4} &= [\n\end{align*}
\]

- The output of the tester should be

\[
\begin{align*}
t(\text{x1}) &= [3, 2, 0] \\
t(\text{x2}) &= [8, 0, 2] \\
t(\text{x3}) &= [7, 2, 6] \\
t(\text{x4}) &= [0, 0, 0]
\end{align*}
\]
Part II. Short answer questions

1. TRUE   FALSE   Python function parameters are named in the function definition.

2. TRUE   FALSE   Python function parameters are named in a function invocation.

3. TRUE   FALSE   A Python function parameter can also act as a function argument.

4. TRUE   FALSE   Python function arguments are given in a function invocation.

5. TRUE   FALSE   All Python function invocations require the use of parentheses.

6. TRUE   FALSE   All Python function invocations have a return value.

7. TRUE   FALSE   All Python function definitions must explicitly have a return statement.
8. TRUE   FALSE  A function can use a print() statement to return a value.

9. TRUE   FALSE  A function invocation that increments its parameter by one, updates the argument used to initialize the parameter.

10. TRUE   FALSE  A function invocation that assigns a new value to a parameter, updates the argument used to initialize the parameter.

11. TRUE   FALSE  A function invocation that appends a new value to its list parameter, updates the list of the argument used to initialize the parameter.

12. TRUE   FALSE  Consider function f().
    
    ```python
    def f( x, y ) :
        remember = x
        x = y
        y = remember
    
    The below statement correctly swaps the values of a and b.
    a, b = f( a, b )
    ```

13. TRUE   FALSE  Consider function f().
    
    ```python
    def f( x, y ) :
        return y, x
    
    The below statement correctly swaps the values of a and b.
    a, b = f( a, b )
    ```

14. TRUE   FALSE  Although local variables only exist during the execution of their function, their values survive from invocation to invocation.

15. TRUE   FALSE  The parameters for a function must have different names than the argument names.
16. TRUE FALSE If a Python function invocation does not supply enough values for the function, Python supplies None for the missing values.

17. TRUE FALSE A function invocation must supply at least one argument value.

18. TRUE FALSE A function definition can contain a function invocation.

19. TRUE FALSE Suppose \( d = \{ \emptyset, \{ 1, 2 \}, \{ 1, 2, 3 \} \} \). The below invocation of built-in function \( \text{sum()} \) correctly totals dataset \( d \).

\[
\text{total} = \text{sum}(d)
\]

20. TRUE FALSE Functions with integer parameters always return an integer value.

21. TRUE FALSE The following function definition correctly determines whether \( x \) is equal to the minimum of strings \( x \), \( y \), and \( z \).

```python
def f( x, y, z ):
    if ( x <= min( y, z )):
        return True
    else:
        return False
```

22. TRUE FALSE The following function definition correctly determines whether \( x \) is equal to the minimum of integers \( x \), \( y \), and \( z \).

```python
def f( x, y, z ):
    if ( x <= min( y, z )):
        return True
    else:
        return False```