READ THIS ENTIRE PAGE. YOU ARE RESPONSIBLE KNOWING WHAT IT SAYS.

HONOR

- By submitting solutions for this test, you are agreeing that
  - You neither given nor received help directly or indirectly to or from anyone else;
  - You did not directly or indirectly use materials from non-allowed sources.

IMPORTANT

- **You must use our files when coding.**
- The `WWHAD` strategy (what would a human do strategy) should serve you well.
- During the test you may not access past code or algorithms (yours, ours, or anyone else's).
- During the test you may not access class notes, epistles, examples, artifacts, solutions on the web, or your own past assignments during the test.
- Class personnel cannot help you debug your answers.
- All functions make use of tester module *run.py*.
- None of your functions should modify any list or dataset parameters.
- None of your functions should print or get input.
- Comment out or delete all debugging `print()` statements before submitting.
- Whether code is testable is important. Every function needs to have at least one uncommented statement.
- None of the testing code should be modified.
- The only device you may access during the exam is your laptop. The only open windows allowed are PyCharm and a browser with tabs linked from the class website.
- During the test you can access the course module descriptions and the course Python information sheet.
- You are responsible for submitting for your work, so check before exiting the testing. Late submissions will not be graded, so do not submit once your testing time is up.
- Code should follow class programming practices; e.g., whitespace, identifier naming, etc.
- Because the problems are short, commenting is not necessary.
- You might add comments if you were unable to complete a problem and want to explain what you were attempting to do.

QUESTIONS

1. Implement module *line.py*. The module defines a single function `y()`. The function has three numeric parameters, `m`, `x`, and `b`.

   The function returns the value of the expression `m` · `x` + `b`.

   The built-in tester for the module should produce the following output.

```
y( 3, 5, 7): 22
y( 5, 7, 3): 38
y( 7, 3, 5): 26
```

2. Implement module *tex.py*. The module defines a single function `words()`. The function has a single string parameter `s`.

   The function returns the number of words in `s`.

   The built-in tester for the module should produce the following output.

```
avg( "The cow mooed and mooed" ): 5
avg( "All things must pass" ): 4
avg( "I have a dream that one day" ): 7
```

3. Implement module *just.py*. The module defines a single function `one()`. The function has four logical (`True` / `False`) parameters `w`, `x`, `y`, and `z`.

   The function returns whether exactly one of parameters `w`, `x`, `y`, and `z` is equal to `True`.

   A list whose elements are the values of `w`, `x`, `y`, and `z` could prove helpful.

   The built-in tester for the module should produce the following output.

```
one( True, False, False, False): True
one( False, False, True, True): False
one( False, False, True, False): True
one( False, True, False, False): True
one( False, False, False, False): False
```

4. Implement module *check.py*. The module defines a single function `in_order()`. The function has a list of integers parameter `x`.

   The function returns whether the values in `x` are arranged in numeric order.

   The built-in tester for the module should produce the following output.

```
in_order( [1] ): True
in_order( [2, 5, 4] ): False
in_order( [5, 6, 8, 8] ): True
in_order( [7, 7, 1, 7, 9] ): False
```

5. Implement module *inv.py*. The module defines a single function `erse()`. The function has one dataset parameter d. The cell values in d are all numeric.

   The function returns a *new* dataset. The values in the new dataset are the additive inverses of the values in d; that is if an individual cell in d has value v, then the corresponding cell in the new dataset has value −v.

   The built-in tester makes uses of datasets d1 and d2.

   `d1 = [ [ 3, 1, −4 ], [ 1, 5 ], [ −9, −2], [ −6 ] ]`

   `d2 = [ [ 1 ], [ 0 ], [−1 ] ]`

   The built-in tester for the module should produce the following output.

```
erse( d1 ): [[−3, −1, 4], [−1, −5], [9, 2], [6]]
erse( d2 ): [[−1], [0], [1]]
```

6. Implement module *bit.py*. The module defines a single function `ter()`. The function has two integers parameter n and k.

   The function returns a *new* list with n elements. Each element is a random value between 0 and k−1.

   Your code *may not make use* of the `random` module `seed()` function.

   The built-in tester for the module should produce the following output.

```
ter( 5, 2 ): [1, 0, 1, 0, 1]
ter( 8, 10 ): [4, 1, 4, 8, 0, 6, 1, 9]
```

7. Implement module *sim.py*. The module defines a single function `metric()`. The function has one dictionary parameter d.

   The function returns `True` or `False` depending whether d is a *symmetric* dictionary.

   A dictionary is *symmetric* if for every mapping *k* to *v* in the dictionary, then there is also a mapping of *v* to *k* in the dictionary.

   Recommendation: Loop on the keys in d. Suppose key *k* maps to value *v*.

   - If *v* is not `d.keys()`, then there is a missing mapping for dictionary d to be symmetric.
   - If instead d[*v*] is not equal to *k*, then there is a missing mapping for dictionary d to be symmetric.
   - If there are no missing mappings, d is symmetric.

   The built-in tester for the module should produce the following output.

```
metric( {1: 2, 2: 3, 3: 1} ): False
metric( {1: 2, 2: 1, 3: 4, 4: 3} ): True
metric( {1: 2, 2: 5} ): False
```

8. Implement module *exc.py*. The module defines a single function `lusive()`. The function has two list parameter x and y.

   The function returns a new list whose elements are all of the elements of x that are not in y, *followed by* all of the elements of y that are not in x.

   The built-in tester for the module should produce the following output.

```
lusive( [3, 5, 9], [2, 5, 3, 5, 8, 8] ): [9, 2, 8, 8]
lusive( [9, 7, 3, 2], [2, 3, 7] ): [9]
lusive( [3], [1, 4] ): [3, 1, 4]
lusive( [], [1, 2, 3] ): [1, 2, 3]
lusive( [1, 2], [1, 2] ): []
```