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.
- None of your functions should modify any list or dataset parameters.
- None of your functions should print or get input.
- Class personnel cannot help you debug your answers.
- 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 solutions are all 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.

Problems

1. Implement module *onds.py*. The module defines a function `sec()`. The function has two integer parameters `h` and `m`.

   The function returns the total number of seconds in a time period of `h` hours and `m` minutes. Remember there are 60 minutes in an hour and 60 seconds in a minute.

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

   ```
   sec( 1, 0 ): 3600
   sec( 0, 1 ): 60
   sec( 23, 60 ): 86400
   ```

2. Implement module *how.py*. The module defines a function `many()`. The function has two string parameters `s` and `t`.

   The function returns the number of words in string `s` that contain string `t`.

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

   ```
   many( 'aaa bbb ccc ddd', 'bb' ): 1
   many( 'able was i ere i saw elba', 'a' ): 4
   many( 'bookkeeper', 'eee' ): 0
   ```

3. Implement module *twelve.py*. The module defines a function `count()`. The function has one integer parameter `n`.

   The function should have a loop that repeats `n` times. Each time the loop repeats, it simulates the rolling of two normal six-sided dice by generating two numbers from the interval 1 through 6. The function returns the number of times, the two dice rolls sum to 12.

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

   ```
   count( 36 ): 1
   count( 3600 ): 88
   ```

4. Implement module *two.py*. The module defines a function `add()`. The function has two integer list parameters `x` and `y`. The lists have the same length.

   The function returns a new list whose first element is the total of the first elements of `x` and `y`, the second element is the total of the second elements of `x` and `y`, and so on.

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

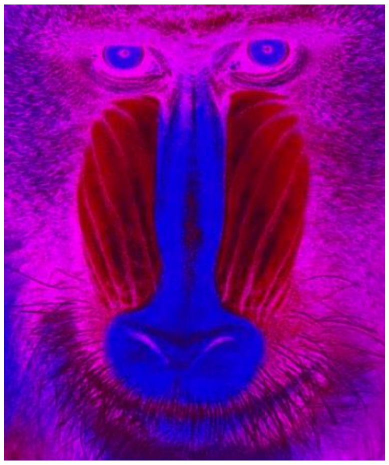
```
add( [1, 2], [3, 4] ): [4, 6]
add( [1, 3, 5, 7], [9, 5, 1, -3] ): [10, 8, 6, 4]
add( [3, 1, 4], [2, 7, 1] ): [5, 8, 5]
```

5. Implement module *tweak.py*. The module defines a function `cynanic().` The function has one RGB pixel parameter `opixel`.

   The function returns a new pixel, where

   - The red component of the new pixel equals 255 minus the red component of `opixel`.
   - The green component of the new pixel equals 0.
   - The blue component of the new pixel equals 255 minus the blue component of `opixel`.

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



6. Implement module *dd.py*. The module defines a single function `iso().` The function has two dictionary parameters `d1` and `d2`.

   The function returns whether the two dictionaries have the same keys.

   A possible algorithm

   - Check whether each of the keys in `d1` is also in `d2`. If any are not, the function returns `False`.
   - Check whether each of the keys in `d2` is also in `d1`. If any are not, the function returns `False`.
   - If no missing keys are found, the function returns `True`.

   The built-in tester uses the following dictionaries.

```
e1 = { 1: '1', '2': 200, 'c': 'iii' }
e2 = { 1: 100, 'b': '2', 'c': 3 }
```

```
f1 = { 1: '1', '2': 2, 3: 'iii' }
f2 = { 1: '1', '2': 2, 3: 'iii', 4: 'four' }

g1 = { 0: 0, 1: 1, 2: 10, 3: 11, 'b': 2 }
g2 = { 1: 1, 0: 0, 3:  3, 2:  4, 'b': 2 }
```

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

```
iso( e1, e2 ):  False

iso( f1, f2 ):  False

iso( g1, g2 ):  True
```

7. Implement module *ma.py*. The module defines a single function `sig()`. The function has one dataset parameter `d`.

   The function returns a list of three values: the number of negative values, the number of zero values and the number of positive values.

   The built-in tester uses the following datasets.

```
d1 = [ [ 0 ], [ -3, 0 ], [ -4, -2 ] ]
d2 = [ [ -3, 1, -2, 1 ], [ -3, -3, -2, -4, -1, -4 ] ]
d3 = [ [ 2, -1, 0 ], [ 3, 0, 3, -2, -2 ], [ -1, -4, 3 ], [ -4, 3, -1, 3 ] ]
d4 = [ ]
```

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

```
sig( d1 ): [3, 2, 0]

sig( d2 ): [8, 0, 2]

sig( d3 ): [7, 2, 6]

sig( d4 ): [0, 0, 0]
```

8. Implement module *ds.py*. The module defines a single function `create()`. The function has three integer parameters `nr`, `nc`, and `k`.

   The function returns a new dataset with `nr` rows with all rows having `nc` columns. Each cell value is a random number from the `range( 0, k )`.

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

```
create( 2, 4, 5 ): [[3, 4, 3, 3], [4, 4, 1, 1]]

create( 3, 2, 9 ): [[2, 6], [6, 4], [7, 3]]

create( 3, 3, 4 ): [[2, 0, 2], [0, 3, 0], [2, 1, 2]]
```