Name: 

Email id: 

Pledge: 

**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 a test score of 0.
- 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.
- With regard to your functions:
  - Comments including header identifying comments are not necessary.
  - You should follow other class style practices; e.g., whitespace, identifier naming, etc.
  - Only do what is requested.
  - None of the functions should get input or produce output.
  - Functions should not modify their parameters in any way.
  - Whether a function is testable is important.
- Any form of cheating on a test can result in failing the class and the incident being referred to the Honor Committee.
Part 1: Program implementation

1. Implement a program `hone.py`. The program prints `yes` if you have either asked a question of the instructor during class or answered a question of the instructor during class; otherwise, the program prints `maybe`. There should be no other output. FYI: some points will be awarded regardless of your output.

2. Implement a program `cab.py`. The program separately prompts and reads four values.
   - The integer number of taxis \( t \).
   - The integer number of days \( d \).
   - The decimal number of the expected number of rides per day \( r \).
   - The decimal number of the expected number of miles per ride \( m \).

   The program computes and prints the decimal number of miles driven by the \( t \) taxis over \( d \) days with \( r \) rides per day and with \( m \) miles per ride.

   Two sample program runs are given below.

   Enter number of taxis: 4
   Enter number of days: 12
   Enter rides per day: 74.5
   Enter miles per ride: 2.6
   9297.6

   Enter number of taxis: 1
   Enter number of days: 7
   Enter rides per day: 150.0
   Enter miles per ride: 3.8
   3990.0

3. Implement a program `ding.py`. The program prompts for a line of text. The program computes and prints the integer average word length of the text and the number of words having that average word length.

   Two sample program runs are given below.

   Enter text: the yellow car jumped over the moon
   4 2

   Enter text: one hundred
   5 0

4. Implement a program `esr.py`. The program prompts for a line of text. The program prints the reverse of the input.

   Two sample program runs are given below.

   Enter text: the yellow car jumped over the moon
   noom eht revo depmuj rac wolley eht

   Enter text: one hundred
derdnuh eno
Part II. Function implementation

5. Implement a module *randy.py* that defines a function `p()` with parameters `n`, `b`, and `s`. Parameters `n` and `b` are integers; parameter `s` can be anything. The function first uses `s` to set the seed for the random number generator. The function then computes and returns a list of `n` base `b` numbers (i.e., numbers in the range `0` through `b-1`). The module has a built-in tester. The output of its testing is below.

```
p( 12, 2, 'randy' ): [0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1]
p( 5, 10, 15 ): [3, 0, 8, 0, 2]
p( 3, 8, 38 ): [6, 6, 1]
p( 0, 8, 11 ): []
```

6. Implement a module *soda.py* that defines a function `pop()`. Function `pop()` has three integer parameters `a`, `b`, and `c`. The function returns 'x' if `b` is greater than `a`; the function returns 'y' if `c` is greater than `a`; and returns 'xy' if both `b` and `c` are greater than `a`. Otherwise, the function returns None. The module has a built-in tester. The output of the tester should be

```
pop( 3, 4, 1 ): x
pop( 3, 1, 5 ): y
pop( 3, 7, 5 ): xy
pop( 3, 2, 1 ): None
```

7. Implement a module *parse.py* that defines a function `dec()`. Function `dec()` has one string parameter `s`. The function returns whether `s` is a decimal string. The module has a built-in tester.

For our purposes, a decimal string is a nonempty string containing exactly one decimal point, and where the parts of `s` both before and after the decimal point are composed of one or more digits.

The output of the tester should be

```
dec( '3.14' ): True
dec( '.14' ): False
dec( '14.' ): False
dec( '.' ): False
dec( '3.1.4' ): False
dec( '' ): False
dec( 'x' ): False
```

8. Implement a module *condo.py* that defines a function `sider()`. Function `sider()` has two parameters `s` and `x`, where `s` is a string and `x` is a list of strings. The function returns the number of strings in `x` that equal `s` if both capitalization, and leading and trailing whitespace is ignored when comparing. The module has a built-in tester. The output of the tester should be

```
sider( 'cat', ['cat', 'dog'] ): 1
sider( 'CAT', ['cat', 'dog', 'CAT'] ): 2
sider( 'cat', [' cat', 'cat', 'cat', ' cat', 'dog'] ): 3
sider( 'cat', [' cat', 'cAt', ' caT', 'dog'] ): 3
sider( ' cat', ['dog', ' cat', 'cAt', ' caT', 'dog'] ): 3
```
9. Implement a module `dis.py` that defines a function `anti()`. The function has two list parameters `x` and `y`. The function returns a new list consisting of the elements of `x` that are not part of `y`, followed by the elements of `y` that are not part of `x`. The function does not modify the contents of `x` and `y`. The module has a built-in tester. The output of the tester should be

```python
anti( [3, 1, 4], [4, 3, 1] ): []
anti( ['a', 1], [4, 'a', '1'] ): [1, 4, '1']
anti( ['a', 'b', 'c', 'd'], ['e', 'c', 'a'] ): ['b', 'd', 'e']
```

10. Implement a module `flat.py` that defines a function `ten()`. The function has one dataset parameter `d`; that is, it is a list of row data. The function does not modify the contents of `d`. The module has a built-in tester.

Function `ten()` returns a new list that is a flattened version of `d`; that is, the new list consists of all of the data cells in `d` in row-major order; that is, the elements of the first row occur first, followed by the elements of the next row, and so on.

The testing makes use of the following datasets.

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

The output of its testing should be

```python
ten( d1 ): [0, 1, 2, 1, 2, 3, 0]
ten( d2 ): [1, 0, 1, 2, 2, 3, 0, 1, 1, 0, 2, 0, 0, 1]
ten( d3 ): [3, 0, 3, 3, 0, 3, 0, 1, 1, 0, 2]
ten( d4 ): []
```

11. Implement a module `game.py` that defines one function `encode()` with two parameters `s` and `x`, where `s` is a string and `x` is a list of characters. The module has a built-in tester.

The function turns a new string whose value is related to `s`. The new string leaves all copies of the characters in `x` alone and replaces all of the other characters with underscores. For example, `encode( 'hello', [ 'l', 'a', 'h' ] )` evaluates to `'^lll'`.

The output of the tester should be

```python
    _ee_ [ 'e' ]
    _ee_ [ 'e', 'b' ]
    _ee_ [ 'e', 'b', 'h' ]
    _ee_ [ 'e', 'b', 'h', 'l' ]
    _ee_ [ 'e', 'b', 'h', 'l', 'j' ]
    peep [ 'e', 'b', 'h', 'l', 'j', 'p' ]
success
```
12. Implement a module `data.py` that defines three functions `row_sum()`, `col_sum()`, and `d_sum()`. Functions `row_sum()` and `col_sum()` both have two parameters `d` and `k`. Functions `d_sum()` has one parameter `d`. For all three functions, `d` is a dataset; that is, it is a list of row data. For `row_sum()` and `col_sum()`, parameter `k` is an integer index. The functions do not modify the contents of `d`. The module has a built-in tester. The tester makes use of two datasets `d1` and `d2`. The rows of the datasets are taken respectively from web datasets:

- www.cs.virginia.edu/~cs1112/datasets/csv/trex.csv
- www.cs.virginia.edu/~cs1112/datasets/csv/rotunda.csv

Function `row_sum( d, k )`
- Returns the sum of the `k`\textsuperscript{th} row in `d`.

Function `col_sum( d, k )`
- Returns the sum of the `k`\textsuperscript{th} column in `d`.

Function `d_sum( d )`
- Returns the sum of all of the elements in `d`.

The output of the tester should be

\[
\begin{align*}
\text{row}_\text{sum}( \text{d1}, 3 ) & : 317 \\
\text{row}_\text{sum}( \text{d2}, 3 ) & : 158 \\
\text{col}_\text{sum}( \text{d1}, 0 ) & : 343 \\
\text{col}_\text{sum}( \text{d2}, 2 ) & : 1239 \\
\text{d}_\text{sum}( \text{d1} ) & : 1104 \\
\text{d}_\text{sum}( \text{d2} ) & : 4354
\end{align*}
\]

Note: in my implementation of the module, none of the functions made use of the other functions in the module.

13. Implement a module `gen.py` that defines two functions `neg_pixel()` and `neg_image()`. Function `neg_pixel()` has one parameter `p`, which is a pixel. Function `neg_image()` has one image parameter `original`. The module has a built-in tester.

Function `neg_pixel( p )`
- Returns a new color-negative version of pixel `p`, that is, it returns \((255 - r, 255 - g, 255 - b)\), where `r`, `g`, and `b` are the RGB levels of `p`.

Function `neg_image( original )`
- Returns a new color-negative version of the `original` image, where for an `original` image pixel equal to \((r, g, b)\), the new image has pixel \((255 - r, 255 - g, 255 - b)\). The function does not modify `original`.

The output of its testing is below. A color version of its imagery is available on the class website.

\[
\text{neg}_\text{pixel}( 50, 100, 200 ) : ( 205, 155, 55 )
\]
14. Implement a module `trans.py` that defines three functions `factor()`, `analyze()`, and `mesh()`. The module has a built-in tester.

Function `factor(a, b)`

- The function returns `True` or `False` depending whether `b` evenly divides `a` or not.

Function `analyze(spot, k, c1, c2)`

If `k` evenly divides either the `x` or `y` components of `spot`, then the function returns `c1`; otherwise, the function returns `c2`.

Function `mesh(original, k, c)`

- Returns a new copy of the original image except for its pixel locations where `k` evenly divides either the `x` or `y` location components. For those pixels, color `c` is used. The function does not modify `original`.

The output of its testing is below. A color version of its imagery is available on the class website.

```
factor(11, 4):  False
factor(12, 3):  True
(255, 255, 255)
(255, 255, 255)
(0, 0, 0)
```
15. Implement a module `nary.py` that defines one function `sym()`. The function has one `dict` parameter `d`. The function returns `True` or `False` depending whether `d` is symmetric. The module has a built-in tester.

A dictionary is symmetric if for any mapping from `k` to `v` in the dictionary, then there is also a mapping from `v` to `k` in the dictionary.

The testing makes use of the following dictionaries.

\[
\begin{align*}
\text{d1} & = \{ 'a': 1, 'b': 2, 2: 'b', 1: 'a' \} \\
\text{d2} & = \{ 'a': 1, 'b': 2, 2: 'b', 1: 2 \} \\
\text{d3} & = \{ 'a': 1, 'b': 2, 2: 'b' \}
\end{align*}
\]

The output of its testing is below.

```
sym( d1 ): True
sym( d2 ): False
sym( d3 ): False
```