Very legibly *print* your name:

Very legibly *print* your email id:

Pledge:

Important

- You are responsible for reading this page in its entirety.
- Based on your many past educational achievements, I expect you to do well on this test.
- Read each question both thoroughly and mindfully of what is being asked.
- To receive points, you must turn in your test.

Terminology

- *Increment* means add.
- *Decrement* means subtract.
- The *additive inverse* of a number *x*, is –*x*.
- The value of x / y is called a *quotient*.
- A number *x* is *evenly divisible* by a number *y*, if the remainder of *x* divided by *y* is zero. The % is the Python remainder operator.
- The terms *dataset* and *table* are synonymous. They are lists of lists. The standard view is they are lists of *rows*, where each row is a list of *column* values. The individual column values are called *cells*.

Test rules of conduct

- This pledged exam is closed notes. The only device you may access during the test is your laptop.
- 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 linked from the class website.
- Functions should demonstrate follow style rules; e.g., whitespace, identifier naming, commenting, etc.
- Whether a function is testable is important. To be testable, a function must have at least one statement. So, if you comment out all of a function's code, add an uncommented `pass` statement.
- With respect to output, only do what is requested. Spurious output can cause loss of points.
- Your functions should neither get input nor produce output.
- Your functions should not modify their parameters in any way.
- Any form of cheating on a test can result 6 failure and the incident being referred to the Honor Committee.

1. Modify the definition of variable *status* in program *q01_output.py*. It should be set to 'yes' or 'no' depending whether you asked a question to the instructor or gave an answer in reply to a question by the instructor (full points will be awarded regardless of whether you answer 'yes' or 'no'. There should be no other output.

   The correct program output is either

   ```
   yes
   ```

   or

   ```
   no
   ```

2. Implement program *q02_input.py*. The program uses a single prompt to cause the user to supply three words. The words are printed out with the first word in lowercase, the second word in uppercase, and with the third word capitalized. There should be no other output.

   A possible run of the program is depicted below.

   ```
   Enter three words: AlphA   betA  dELTa
   alpha BETA Delta
   ```

3. Implement program *q03_decisions.py*. The program uses a single prompt to cause the user to supply two non-zero integers. If the numbers are both positive, their difference is printed. If instead the numbers are both negative, their sum is printed. If instead the first number is negative and the second is positive, their decimal quotient is printed; otherwise their product is printed. There should be no other output.

   Four *different* possible runs are given below.

   ```
   Enter two non-zero integers: 3 4
   -1
   ```

   ```
   Enter two non-zero integers: -2 -5
   -7
   ```

   ```
   Enter two non-zero integers: -4 8
   -0.5
   ```

   ```
   Enter two non-zero integers: 9 -3
   -27
   ```

4. Implement program *q04_list_input.py*. The program uses a single prompt to cause the user to supply integers. The values are first printed out as a list of strings. The values are then printed out as a list of integers. The elements of that list are then replaced with their additive inverses and printed out.

   A possible run of the program is depicted below.

   ```
   Enter integers: 3 -1 4 1 -5 -9
   ['3', '-1', '4', '1', '-5', '-9']
   [3, -1, 4, 1, -5, -9]
   [-3, 1, -4, -1, 5, 9]
   ```

Another possible run of the program is depicted below.

```
Enter integers: -2 -4 6
['-2', '-4', '6']
[-2, -4, 6]
[2, 4, -6]
```

5. Produce a module *q05_randomness.py*. The module defines two functions `toss()` and `streak()`.

   `toss()`

   - Simulates a fair coin toss by randomly returning `'heads'` or `'tails'`.

   `streak( n )`

   - Invokes the `toss()` function *n* times. Returns a two-element list. The first element is the number of times `toss()` returns `'heads'`; and the second element is the number of times the `toss()` returns `'tails'`.

   The module includes a built-in tester. A run of the tester should produce the below output.

```
heads tails heads tails
[2482, 2518]
[24929, 25071]
```

6. Produce a module *q06_functions.py*. The module defines three functions `leap_year()`, `months()`, and `ytd()`.

   `leap_year( y )`

   - Returns `True` or `False` depending whether *y* is a leap year. A year is a leap year, if it is evenly divisible by 4 and not evenly divisible by 100. Also, a year is a leap year, if it is evenly divisible by 400. All other years are not leap years.

   `months( y )`

   - Returns a 12-element list giving the numbers of days in each month for year *y*.

   `ytd( m, d, y )`

   - Returns the day of the year, where the year is *y*, the month is *m*, and the day of the month is *d*. For example, 1/1/2017 is the 1st day of the year, 12/7/2017 is the 341st of the year, and 3/1/2020 is the 61st day of the year.

   The module includes a built-in tester. A run of the tester should produce the below output.

```
False True False True
[31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
[31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
1 341 61 365
```

7. Produce a module *q07_dictionary.py*. The module defines functions `make()` and `surjection()`.

   `make( k, v )`

   - If the lengths of *k* and *v* are not equal, the function returns `None`. Otherwise, the function returns a new dictionary. In this dictionary, the i^{th} element of *k* maps to the i^{th} element of *v*.

`surjection( d )`

- Returns `True` or `False` depending there are *no* duplicates in *`d.values()`*.

The module includes a built-in tester. A run of the tester should produce the below output.

```
None
{1: 'a', 2: 'b', 3: 'c', 4: 'd'}
True, False
```

8. Produce a module *`q08_bw.py`*. The module defines two functions `bicolor()` and `bw()`.

`bicolor( pixel )`

- Returns (0, 0, 0) if the sum of the RGB values for *`pixel`* is less than 382.5 (i.e., 3•255/2); otherwise, returns (255, 255, 255).

`bw( original )`

- Returns a new image that is based on *`original`*, where a *`pixel`* in the new image is obtained by using `bicolor()` on the corresponding *`pixel`* in *`original`*.

The module includes a built-in tester. A run of the tester should produce the below output and the image to the right.

```
(255, 255, 255)
(0, 0, 0)
```

9. Produce a module *`q09_shift.py`*. The module defines three functions `legal()`, `offset()`, and `shift()`.

`legal( v )`

- Returns an integer based on integer *v*. If *v* is negative, the function returns 0. If instead *v* exceeds 255, the function returns 255. Otherwise, the function returns *v*. Thus, the return value is always a legal value for an individual RGB level.

`offset( pixel, x, y, z )`

- Returns the new color (`legal(` *nr*), `legal(` *ng*), `legal(` *nb*)), where *nr*, *ng*, and *nb* are the values gotten by decrementing the RGB values of *`pixel`* respectively by the amounts *x*, *y*, and *z*.

`shift( original, x, y, z )`

- Returns a new image that is based on `original`, where a pixel in the new image is obtained by using `offset()` on the corresponding pixel in `original` with offset amounts *x*, *y*, and *z*.

The module includes a built-in tester. A run of the tester should produce the below output and a greenish version of the image to the right.

```
0 125 255
(0, 50, 125)
```

10. Produce a module *q10_datasets.py*. The module defines one function `add()`.

`add( d1, d2 )`

- Parameters *d1* and *d2* are both integer datasets.

You can assume *d1* and *d2* have the same number of rows. You can also assume all rows of *d1* and *d2* have the same number of columns.

- The function returns a new dataset with the same number of rows and columns as does *d1* and *d2*. A cell in the new table is the sum of the corresponding cells in *d1* and *d2*.

The module includes a built-in tester. A run of the tester should produce the below output.

```
     [[1, 2, 2, 0], [0, 4, 2, 4], [2, 4, 1, 1]]
     [[4, 4, 0, 1], [4, 4, 4, 1], [1, 2, 0, 4]]
sum: [[5, 6, 2, 1], [4, 8, 6, 5], [3, 6, 1, 5]]

     [[1, 2, 2], [2, 3, 2], [4, 2, 1], [0, 0, 2], [2, 4, 4], [4, 4, 1]]
     [[1, 2, 0], [4, 0, 0], [2, 2, 1], [4, 1, 0], [1, 0, 3], [1, 4, 3]]
sum: [[2, 4, 2], [6, 3, 2], [6, 4, 2], [4, 1, 2], [3, 4, 7], [5, 8, 4]]
```

11. Produce a module *q11_strings.py*. The module defines three functions `reverse()`, `cleanup()`, and `viceversa()`.

`reverse( s )`

- Returns the reverse of string *s*.

`cleanup( s )`

- Returns a version of *s* with all blanks removed.

`viceversa( s )`

- Returns `True` or `False` depending whether a `cleanup()` version of *s* is the same forward as backward.

The module includes a built-in tester for functions `reverse()`, `cleanup()`, and `viceversa()`. A run of the tester should produce the below output.

```
' how high is high '
'   able was i ere i saw elba'

[' how', ' high is h', 'igh ']
['   able', ' was i ere i s', 'aw elba']

'howhighishigh'
'ablewasiereisawelba'

' hgih si hgih woh '
'able was i ere i saw elba   '

False
True
```