Name:

E-mail ID:

On my honor, I pledge that I have neither given nor received help on this test.

Signature:

Testing

- Print your name, id, and pledge as requested.
- This pledged exam is closed textbook. The only device you may access during the test is your own laptop.
- You are not allowed to access class examples or your own past assignments during the test; i.e., the only Python code you may access or view are ones that you develop for this test.
- The only windows that can be open on your computer are PyCharm and a single browser with tabs only open to the class website.
- Only problems 1 and 6 should produce output.
- *No submissions will be accepted after the test is over. Eleven points will be awarded for each of the requested program or functions to be implemented. It is incumbent on you to make the submissions. You are to make submissions as you complete problems.*
- The expected grading rubric for the Problem 1 program is
    - Two points for the submission;
    - Nine points for correctness, where they are awarded as follows:
        - One point for a single line of output;
        - One point for no leading or trailing spaces in the output;
        - Three points for printing 711;
        - One point for a printing a single space after the 711;
        - Three points for printing pounds in lower case after the space.
- The expected grading rubric for a function is
    - Two points for attempting the function;
    - One point for following instructions (e.g., no output for function *q02.rate()*;
    - Two points for always returning a value of the proper return type.
    - Six points for correctness, where except for functions *q03.one*(), *q06.generate*(), *q07.f*(), *q09.friendly*(), and *q09.mutual*() they are awarded as follows.
        - One point for getting one test case correct;
        - Two points for getting two test cases correct;
        - Three points for getting three test cases correct; and
        - Six points for getting all test cases correct.

- For function *q03.one*(), *q09.friendly*(), and *q09.mutual*() the correctness points are awarded as follows.
  - Two points for getting a test case correct where a true return value is wanted and also getting a test case correct where a false return value is wanted.
  - Six points for getting all test cases correct.
- For function *q06.generate*() the correctness points are awarded as follows.
  - Four points for printing all of the requested tuples;
  - One point for printing the tuples one per line;
  - One point for no output other than the tuples (e.g., no blank lines).
- For function *q07.f*() the correctness points are awarded as follows.
  - Two points for having a recursive definition of *f*().
  - Two points for getting a test case correct where a true return value is wanted and also getting a test case correct where a false return value is wanted.
  - Four points for getting all test cases correct

- Our testing of the functions will involve different test cases than the ones used for elaborative purposes in the problem descriptions.

1. (11 points) Develop program q01.py

- The program prints the string `'711 pounds'`. For your information that is the answer to the riddle how much wood could a woodchuck chuck if a woodchuck could chuck wood (on a good day with the wind too its back.
- A run of program `q01.py` should produce the following output.

```
711 pounds
```

2. (11 points) Develop module q02.py. The module defines a function *rate*().

- Function *rate*() has one integer parameter *kph*. The function returns its wind force category according to an abbreviated version of the Beaufort scale.
    - 0 kph is Calm
    - 1 – 49 kph is a Breeze
    - 50 – 117 kph is a Storm
    - 118 or greater is a Hurricane
- Program `prog02-rate.py` invokes function *rate*() and should produce the following output.

```
rate(   1 ) = Breeze
rate(   0 ) = Calm
rate( 121 ) = Hurricane
rate( 117 ) = Storm
```

3. (11 points) Develop module q03.py. The module defines a function *one*().

- Function *one*() has four Boolean parameters *w*, *x*, *y*, and *z*. The function returns whether exactly one of *w*, *x*, *y*, and *z* is `True`.

  Program `prog03-one.py` invokes function *one* () and produces 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. (11 points) Develop module q04.py. The module defines function convert() .

- Function *convert*() has one string parameter *csv*. Parameter csv is a string of comma-separated integers. The function returns a list of integer values corresponding to the number strings in *csv*.

  Program `prog04-convert.py` invokes function *convert*() and produces the following output.

```
convert( '31,41,59\n26,53,58\n97,93,23\n84,62,64')
     = [[31, 41, 59], [26, 53, 58], [97, 93, 23]]
convert( '84 62 64\n33,83 27\n95, 2,88\n41,97,16\n93,99,37' )
     = [[84, 62, 64], [33, 83, 27], [95, 2, 88], [41, 97, 16], [93, 99, 37]]
```

5. (11 points) Develop module q05.py. The module defines function calc().

- Function *calc*() has one list parameter *data*. Each element of *data* is a list of three integer values. The function returns a list of three values. The first value in the return is the integer average of the first elements in the *data* lists. The second value in the return is the integer average of the second elements in the *data* lists. The third value in the return is the integer average of the third elements in the *data* lists.

  Program `prog05-calc.py` invokes function *calc* () and produces the following output.

```
calc( [[31, 41, 59], [26, 53, 58], [97, 93, 23], [84, 62, 64]] )
     = [59, 62, 51]
calc( [[33, 83, 27], [95,  2, 88], [41, 97, 16]]] )
     = [56, 60, 43]
```

6. (11 points) Develop module `q06.py`. The module defines a function generate() .

- Function *generate*() has one parameter *n* and does not return a value. The program prints all three-tuples where each value in a tuple comes from the range(1, *n*+1).

  Program `prog06-generate.py` invokes function *generate*() and produces the following output.

```
(1, 1, 1)

(1, 1, 1)
(1, 1, 2)
(1, 2, 1)
(1, 2, 2)
(2, 1, 1)
(2, 1, 2)
(2, 2, 1)
(2, 2, 2)
```

  Because function *generate*() can be implemented in different ways, your listing of tuples can be different. However, all the tuples listed in the box, and no others, must make up your output.

7. (11 points) Develop module `q07.py`. The module defines a recursive function *f*().

- Function *f* () must be recursive. The function has one string parameter *s* and returns `True` or `False` according to the following specification. In the specification *n* is the length of string *s*:

$$f(n) = \begin{cases} \text{True} & n = 0 \\ \text{True} & n = 1 \\ (s[\,0\,] \text{ equals } s[\,n-1\,]) \text{ and } f(\,s[\,1{:}n-1\,]\,) & n > 1 \end{cases}$$

Program `prog07-f.py` uses function f() and produces the following output.

```
f( 'A' ) =  True
f( 'AB' ) =  False
f( 'ABA' ) =  True
f( 'ABAB' ) =  False
f( 'ABBA' ) =  True
f( 'ABABA' ) =  True
f( 'ABBABB' ) =  False
```

8. (11 points) Develop module `q08.py`. The module defines a function tri() .

- Function *tri*() has one parameter *drawing* of type *Image*. The function returns a new *Image* with the same dimensions as *drawing*. The color of a pixel in the new *Image* depends upon the (*r*, *g*, *b*) values of the corresponding pixel in *drawing*.
  - The pixel color is (*r*, 0, 0), if *r* is the maximum of *r*, *g*, and *b*;
  - Otherwise, the pixel color is (0, *g*, 0), if *g* is the maximum of *g* and *b*;
  - Otherwise, the pixel color is (0, 0, *b*).

The below images are grey scale versions of the images program `prog08-tri.py` manipulates with its use of function *tri*(). Color versions of the pictures are available on the class website.

9. (44 points) Develop module `q09.py`. The module defines functions *friendly*(), *mutual*(), *join*(), and *bff*().

- Several of the module functions have a `dict` parameter *liking*. A *liking* key is the name of a person; the value of a *liking* key is the list of people to whom the key person is friendly (i.e., the key person's friends). The following code segment defines *likes1* and *likes2* for liking examples.

```
likes1 = { 'Noah':    [ 'Sophia', 'Mason', 'Liam' ],
           'Emma':    [ 'Sophia', 'Liam', 'Olivia' ],
           'Sophia':  [ 'Mason' ],
           'Olivia':  [ 'Emma', 'Liam',  ],
           'Mason':   [ 'Olivia', 'Sophia' ],
           'Liam':    [ 'Emma', 'Noah', 'Olivia', 'Sophia' ] }

likes2 = { 'Bella':   [ 'Max', 'Lucy', 'Charlie', 'Bailey', 'Daisy', 'Molly' ],
           'Max':     [ ],
           'Charlie': [ 'Bella', 'Sadie', 'Bailey', 'Max' ],
           'Molly':   [ 'Bailey', 'Bella' ],
           'Lucy':    [ 'Max', 'Bailey', 'Bella' ],
           'Bailey':  [ 'Lucy', 'Sadie', 'Daisy' ],
           'Sadie':   [ 'Lucy', 'Daisy' ],
           'Daisy':   [ 'Bailey', 'Lucy', 'Molly', 'Charlie', 'Bella' ] }
```

  In *likes1*, `'Mason'` is friendly with `'Olivia'` and `'Sophia'`. In *likes2*, `'Lucy'` is friendly with `'Max'`, `'Bailey'`, and `'Bella'`

- Function *friendly*() has parameters *liking*, *name1*, and *name2*. Parameter *liking* is a `dict`; parameters are *name1* and *name2* are strings. The function returns whether according to the *liking* `dict`, *name1* has *name2* as a friend.

  For example, suppose the *liking* parameter *list1* `dict` given above. If *name1* and *name2* were respectively `'Mason'` and `'Noah'`, then the function returns `False` (`'Mason'` does not consider `'Noah'` a friend). If instead *name1* and *name2* were respectively `'Noah'` and `'Mason'`, then the function returns `True` (`'Noah'` considers `'Mason'` a friend).

  Program `prog09-friendly.py` uses function *friendly*() and produces the following output.

```
friendly( like1, 'Mason',   'Sophia' ) = True
friendly( like2, 'Charlie', 'Molly'  ) = False
```

- Function *mutual*() has parameters *liking*, *name1*, and *name2*. Parameter *liking* is a `dict`; parameters are *name1* and *name2* are strings. The function returns whether according to the *liking* `dict`, *name1* has *name2*, and *name2* as a friend.

  For example, suppose the *liking* parameter *list1* `dict` given above. If *name1* and *name2* were respectively `'Mason'` and `'Noah'`, then the function returns `False` (`'Mason'` does not consider `'Noah'` a friend). If instead *name1* and *name2* were respectively `'Mason'` and `'Sophia'`, then the function returns `True` (`'Mason'` considers `'Sophia'` a friend, as does `'Sophia'` with `'Mason'`).

Program `prog09-mutual.py` uses function *mutual*() and produces the following output.

```
mutual( like1, 'Liam', 'Noah' ) = True
mutual( like2, 'Lucy', 'Charlie' ) = False
```

- Function *join*() has two parameters *friends1* and *friends2*. The function returns a new list whose elements are those of *friends1* followed by those of *friends2* with duplicates not being removed.

  For example, suppose *friends1* is the list `['Sophia', 'Mason', 'Liam']`; and *friends2* is the list `['Olivia', 'Sophia']`, the return value is the list `['Sophia', 'Mason', 'Liam', 'Olivia', 'Sophia']`.

  Program `prog09-join.py` uses function *join*() and produces the following output.

```
join( ['Sophia', 'Mason', 'Liam'], ['Liam', 'Olivia', 'Sophia', 'Mason'] )
    = ['Sophia', 'Mason', 'Liam', 'Liam', 'Olivia', 'Sophia', 'Mason']
join( ['Olivia', 'Emma', 'Liam', 'Mason', 'Olivia'], ['Sophia', 'Liam', 'Emma'] )
    = ['Olivia', 'Emma', 'Liam', 'Mason', 'Olivia', 'Sophia', 'Liam', 'Emma']
```

- Function *bff*() has one `dict` parameter *liking*. The function returns the person who is a friend of the largest number of people (i.e., the person who appears in the most friends lists). If there is tie for most frequent, then return any one of the most frequent.

  For example, suppose the *liking* parameter *list1* `dict` given above. Then the return value is `'Sophia'` because that name occurs the most among the lists of friends. Your implementation cannot import any statistics modules.

  Program `prog09-bff.py` uses function *bff*() and produces the following output.

```
bff( likes1 ) = Sophia
bff( likes2 ) = Bailey
```