Signature:

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

Test rules

- You may use a single page of notes. The only device you may access during the test is your laptop.
- Do not access class examples 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 computer windows to be open are PyCharm and a browser that only accesses class website links.
- Code should demonstrate follow style rules; e.g., header comments, whitespace, identifier naming, etc.
- Whether a function is runnable is important.
- Unless indicated, functions should not modify their parameters in any way.

---

1. (9 points) Develop a program `q01.py`. The program prints the string `'I pledge to do my honest best on this test.'` It prints no other output, whatsoever. Below is a sample run of the program

```
I pledge to do my honest best on this test.
```

---

2. (9 points) Develop a program `q02.py`. The program separately prompts the user (i.e., it uses two `input()` invocations) for two strings and prints their combined length and nothing else (e.g., no labelling or blank lines). Below are two sample runs of the program.

```
Enter a string: Can anyone hear me?
Enter a string: What did you say?
36
```

```
Enter a string: What time is it please?
Enter a string: The time is now.
39
```

---

3. (10 points) Develop a program `q03.py`. The program prompts the user for two integers `m` and `n` (i.e., it uses a single `input()` invocation) and prints a single line containing the quotient of `m` divided by `n`, and the remainder of `m` divided by `n`, and nothing else. Below are two sample runs of the program.

```
Enter two numbers: 55 16
3 7
```

```
Enter two numbers: 1776 1066
1 710
```

4. (10 points) Develop a module `q04.py`. The module defines a function `bmi()` that takes two parameters `w` and `h`, which are respectively in pounds and inches.

   - `bmi( w, h )`
     The function returns the body mass index for a person of weight `w` and height `h`. The formula for body mass index is 703 w / h^2.

   Program `test04.py` makes three invocations of function `bmi()` and produces output

```
25.780
24.410
21.698
```

5. (10 points) Develop a module q05.py. The module defines a function `mid()` that takes a string parameter `s`.

   - `mid( s )`
     If length of *s* is odd, the function returns the middle character of *s*; otherwise, the function returns the two middle characters of *s*. For example, `mid( 'abcdef' )` evaluates to `'cd'` and `mid( 'abcde' )` evaluates to `'c'`. Program `test05.py` makes two invocations of function `mid()` and produces output

```
cd
c
```

6. (11 points) Develop a module q06.py. The module defines a function `loners()` that takes a single list parameter x.

   - `loners( s )`
     The function returns a new list composed of all the items appearing in *x* exactly once. For example, `loners( [ 8, 9, 2, 1, 1, 0, 6, 2 ] )` evaluates to `[ 8, 9, 0, 6 ]`. Program `test06.py` makes three invocations of function `loners()` and produces output

```
[0, 6]
[8, 9, 0, 6]
['e', 'd']
```

7. (12 points) Develop a module `q07.py`. The module defines a function hangman() that takes two parameters, a string s and a list of individual characters c.

- hangman(s, c)

  The function returns a new string whose value is related to s. The new string leaves all copies of the characters in c alone, and replaces all of the other characters with underscores. For example, hangman('hello', ['l', 'a', 'h',]) evaluates to 'h_ll_'. Program `test07.py` makes three invocations of function hangman() and produces output

```
h_ll_
_ig___
______
```

8. (10 points) Develop a module q08.py. The module defines a function `build()` that takes a single list parameter *x*.

- build(x)

  The function returns a new `dict`. The keys for the new mappings are the item values in x, the value of a key is the number of times it occurs in x. For example, counts([8, 9, 2, 1, 1, 0, 6, 2]) evaluates to {0: 1, 1: 2, 2: 2, 6: 1, 8: 1, 9: 1}. Program test08.py makes three invocations of function build() and produces the below output. Be aware that your output may list the mappings in a different order.

```
{0: 1, 2: 3, 3: 5, 4: 2, 6: 1, 9: 3}
{0: 1, 1: 2, 2: 2, 6: 1, 8: 1, 9: 1}
{'d': 1, 'b': 2, 'a': 3, 'e': 1, 'c': 2}
```

9. (10 points) Develop a module q09.py that defines two functions `scrub()` and `decode()`. The functions support a very simple version of steganography (i.e., hiding a message in something non-secret (e.g., a picture). Both functions have three parameters `original`, c, and d, where `original` is an `Image`, and c and d are RGB colors (i.e., 3-tuples).

- scrub(original, c, d)

  Returns a new copy of *original* where all occurrences of *c* in *original* are replaced with *d* in the copy.

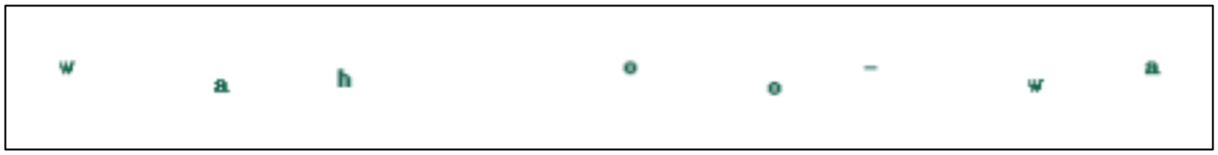
- decode(original, c, d)

  Returns a new copy of *original* where all pixels in *original* not equal to *c* are replaced with *d* in the copy.

Program test09.py tests function `scrub()` and decode() and should produce output

```
scrub() passed test
decode() passed test
```

A successful decode() should also produce a decoding of the rotunda that contains the following sub-image



---

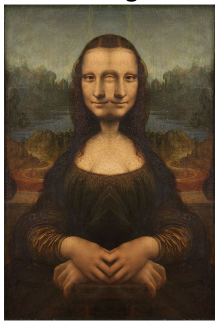
10. (10 points) Develop a module q10.py that defines an image producing function `symmetric()`.

- `symmetric( original )`
  Returns a new copy of *original* where the left-half matches original and the right-half is a reverse of the left-half.

Program test10.py tests function `symmetric ()` and should produce output

```
symmetric() passed test
```

A successful `symmetric()` should also produce the image



---

11. (12 points) Develop a module q11.py that implements a version of the game *marbles*. This two-person game starts off with n marbles being available. The players alternate turns. On a turn, a player must remove at least one marble, but no more than half of the remaining marbles. The player who removes the last marble loses. Our version of the game will have the "*computer*" play against a person. To support the game the following functions are to be implemented.

- `over( n )`
  Returns True if n equals 1; otherwise, returns False.
- `computer( n )`
  Returns the number of marbles the computer takes for its move. The amount is randomly chosen from the *inclusive* range 1 through n / / 2.
- `legal( n, m )`
  If `m` is a legal amount to take, the function returns `True`; otherwise, the function returns `False`.
- `player( n, m )`
  If `m` is a legal amount to take, the function returns `m`; otherwise, the function returns 0.

Program `test11.py` makes use of a function `run()` that plays the game. The following is a sample program interaction (since the computer plays randomly, your game may different).

```
There are 16 marbles
How many marbles should be removed: 0
Invalid move, try again: 9
Invalid move, try again: 8
There are 8 marbles left
I take 3 marbles
There are 5 marbles left
How many marbles should be removed: 2
There are 3 marbles left
I take 1 marbles
There are 2 marbles left
How many marbles should be removed: 1
There are 1 marbles left
You win
```

---

12. (9 points) Develop a module `q12.py`. The module has three functions `distance` and `adjacent()`. Both functions take four integer parameters x1, y1, x2, and y2.

- `distance( x1, y1, x2, y2 )`
  Returns the Euclidian distance between `(x1, y1)` and `(x2, y2)`. The Euclidean distance formula is

  $$\sqrt{(x1-x2)^2+(y1-y2)^2}$$

- `adjacent( x1, y1, x2, y2 )`
  Returns whether the distance between `(x1, y1)` and `(x2, y2)` equals 1.

Program test12.py invokes each of the three functions on the locations (3, 1) and (4, 1), (1, 5) and (6, 9), and (5, 5) and (5, 5). The program produces the below output.

```
1.0
2.8284271247461903
3.605551275463989

True
False
False
```

13. (12 points) Develop a module `q13.py` that accesses a datasheet. The module defines two functions `frequent()` and `indices()`.

- `frequent( d, c )`
  Returns the most frequently occurring value in column `c` of datasheet `d`.

- `indices( d, c, v )`

  Return the indices of the rows in the datasheet d whose column c values equal v.

Program test11.py invokes both of the functions and produces output.

```
Quitman
[11, 19, 27, 30, 35, 46]
```

FYI, program `test13.py` uses the functions to analyze a dataset of USA locations starting with the letter Q to determine the town name starting with Q that occurs most frequently across the USA (i.e., Quincy). The tester gets the datasheet from the CSV file

`www.cs.virginia.edu/~cs1112/datasets/csv/q.csv`

The first six rows in the places datasheet are given below (a different dataset will be used during grading).

```
['Quimby', 'IA']
['Quinebaug', 'CT']
['Quinton', 'NJ']
['Quebradillas', 'PR']
['Quinault', 'WA']
['Quantico', 'MD']
```