-

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.
- None of the functions you write should produce any output.
- *Ten points will be awarded for making all 14 submissions. No submissions will be accepted after the test is over. It is incumbent on you to make the submissions. You are to make submissions as you complete questions.*
- Each function you write is worth ten points. The expected grading rubric is
    - Two points for attempting the function;
    - One point for following instructions (e.g., no output or *q06.f*() must be recursive);
    - Two points for returning a value of the proper return type.
    - Five points for correctness, where except for modules `q03.py` and `q09.py` 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
        - Five points for getting all test cases correct.

      For function `q03.py` the correctness points are 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.
        - Five points for getting all test cases correct.

      For function `q09.py` the correctness points are as follows.
        - Two points for getting a test case correct where None is wanted and also getting a test case correct where None is not wanted.
        - Five 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. Develop module `q01.py`. The module defines function *handback*().

Function *handback*() has no parameters. The function returns the string `'success'`.

Program `q01-tester.py` for module `q01.py` should produce the following output.

```
handback() = success
handback() = success
handback() = success
```

2. **Develop module `q02.py`. The module defines function *line*().**

Function *line*() has three parameters *m*, *x*, and *b*. The function returns the value of $m \cdot x + b$.

Program `q02-tester.py` for module `q02.py` should produce the following output.

```
line(3, 1, 4) = 7
line(5, 9, 2) = 47
line(3, 8, 9  = 33
line(7, 9, 3) = 66
line(8, 4, 6) = 38
```

3. Develop module `q03.py`. The module define function *same*().

Function *same*() has two parameters *u* and *v*. The function returns whether *u* and *v* have the same type.

Program `q03-tester.py` for module `q03.py` should produce the following output.

```
same( 1, 10 ) = True
same( 1.0, 1 ) = False
same( [1], [2, 3] ) = True
same( 31, abc ) = False
same( (1, 2), [1, 2] ) = False
```

4. **Develop module `q04.py`. The module defines a function *common*()**.

Function *common*() has integer parameters *u* and *v*, where $u \le v$. The function returns a list of proper factors common to both *u* and *v*.

The *factors* of a number *n* are those integers that when dividing *n* have a remainder of zero (e.g., factors of 10 are 1, 2, 5, and 10). A factor of *n* is a *proper factor* if it is neither 1 nor *n* (e.g., proper factors of 10 are 2, and 5).

Program `q04-tester.py` for module `q04.py` should produce the following output.

```
common( 2 , 4 ) = []
common( 4 , 4 ) = [2]
common( 6 , 18 ) = [2, 3]
common( 8 , 16 ) = [2, 4]
common( 40 , 80 ) = [2, 4, 5, 8, 10, 20]
common( 90 , 100 ) = [2, 5, 10]
```

5. **Develop module `q05.py`. The module defines functions *shortest*().**

Function *shortest*() has one parameter *data*. Parameter *data* is a list of strings. The function returns the length of the shortest string in *data*, where *data* is nonempty.

Program `q05-tester.py` for module `q05.py` should produce the following output.

```
shortest( ['doing', 'assign', 'normally', 'each', 'part'] ) =  4
shortest( ['of', 'an', 'assign', 'corresponds'] ) =  2
shortest( ['works', 'to', 'the', 'operator'] ) =  2
shortest( ['performs', 'addition', 'when', 'operands'] ) =  4
shortest( ['style', 'rule', 'convene', 'for',', 'code'] ) =  3
shortest( ['ab', '', 'r',] ) =  0
```

6. Develop module `q06.py`. The module defines a recursive function *f*().

Function $f()$ must be recursive. The function has one integer parameter n and returns an integer according to the following specification:

$$f(n) = \begin{cases} 1 & n \leq 0 \\ 2 & n = 1 \\ f(n-1) \cdot f(n-2) & n \geq 2 \end{cases}$$

Program `q06-tester.py` for module `q06.py` should produce the following output.

```
f( -1 ) = 1
f( 0 ) = 1
f( 1 ) = 2
f( 2 ) = 2
f( 4 ) = 8
f( 8 ) = 2097152
f( 10 ) = 36028797018963968
```

7. Develop module `q07.py`. The module defines a function *grab*().

Function *grab*() has two parameters *d* and *data*, where *d* is a *dict* and *data* is a list of possible key values for *d*. The function returns a list whose elements are gotten by querying *d* with the elements of *data* as keys. The order of elements in the list to be returned should match the order of the corresponding keys in *data*.

Program `q07-tester.py` for module `q07.py` should produce the following output.

```
grab( {1: 'I', 10: 'X', 5: 'V'}, [5, 10] ) = ['V', 'X']
grab( {'B': 'b', 'C': 'c', 'D': 'd', 'A': 'a'}, ['B', 'B', 'D'] ) = ['b', 'b', 'd']
grab( {9: 2, 3: 1, 4: 1, 5: 1, 6: 5}, [3, 5] ) = [1, 1]
```

8. Develop module `q08.py`. The module defines a function *search*().

Function *search*() has two parameters *data* and *w*, where *data* is a list and *w* is a string. The function returns the number of occurrences of *w* in *d*. In determining the number of occurrences, case does not matter.

Program `q08-tester.py` for module `q08.py` should produce the following output.

```
search( ['1', 'I', '5', 'V', '10', 'X', '5', '10'], '0' ) = 0
search( ['1', 'I', '5', 'V', '10', 'X', '5', '10'], '5' ) = 2
search( ['A', 'a', 'B', 'b', 'C', 'c', 'D', 'd', 'B', 'B', 'D'], 'B' ) = 4
search( ['A', 'a', 'B', 'b', 'C', 'c', 'D', 'd', 'B', 'B', 'D'], 'b' ) = 4
search( ['A', 'a', 'B', 'b', 'C', 'c', 'D', 'd', 'B', 'B', 'D'], 'D' ) = 3
```

9. Develop module `q09.py`. The module defines a function *solid*().

Function *solid*() has one parameter *drawing*. Parameter *drawing* is an *Image*. If all pixels in *drawing* are the same color, the function returns that color. If the pixels are not all the same, the function returns `None`.

Program `q09-tester.py` for module `q09.py` should produce the following output.

```
solid( <PIL.Image.Image image mode=RGB size=31x41 at 0x101E6DC88> ) = None
solid( <PIL.Image.Image image mode=RGB size=59x26 at 0x101E6DCF8> ) = (128, 0, 128)
solid( <PIL.Image.Image image mode=RGB size=45x45 at 0x101E6DD68> ) = None
```

10. Develop module `q10.py`. The module defines functions *generate*(), *complement*(), *slice*(), *snip*(), and *inverse* ().

There are four kinds of nucleotide modules – adenine, cytosine, guanine, and thymine. They are commonly referred to respectively as A, C, G, and T. Nucleotides A and T are *complementary* molecules, C and G are also *complementary* molecules.

Functions *generate*(), *complement*(), *slice*(), *snip*(), and *inverse* (), all deal with nucleotides, where nucleotides are represented in uppercase string format (i.e., `'A'`, `'C'`, `'G'`, or `'T'`).

Function *generate*() has no parameters and returns a random nucleotide. Each of the possible nucleotides should be equally likely to be the return value.

Function *complement*() has one parameter *n* representing a single nucleotide. The function returns the complement of the nucleotide. If *n* does not represent a single nucleotide, the function returns `None`.

Function *slice*() has three parameters *s*, *a*, and *b*, where *s* is a string of nucleotides, and *a* and *b* are indices into *s*. The function returns a new string of nucleotides that is equal the nucleotides of *s* from indices *a* through *b*–1. You may assume that $a \le b$.

Function *snip*() has three parameters *s*, *a*, and *b*, where *s* is a string of nucleotides, and *a* and *b* are indices into *s*. The function returns a new string of nucleotides that is equal to *s* without its nucleotides at indices *a* through *b*–1 inclusively. You may assume that $a \le b$.

Function *inverse*() has a parameter *s*, where *s* is a string of nucleotides. The function returns a string of nucleotides, each of whose nucleotides is the complement of the corresponding *s* nucleotide.

Program `q10-tester.py` for module `q10.py` should produce the following output.†

```
Test generate() yes or no? yes

generate() = G
generate() = G
generate() = A
generate() = T
generate() = C

Test complement() yes or no? yes

complement( 'A' ) = T
complement( 'C' ) = G
complement( 'G' ) = C
complement( 'T' ) = A
complement( 'other' ) = None

Test slice() yes or no? yes

slice( 'ACAGTCT', 0, 3 ) = ACA
slice( 'ACCAACCCCGG', 1, 4 ) = CCA
slice( 'TGAGTCCGAGGAGA', 5, 9 ) = CCGA
slice( 'GGGTGCTTCAGAG', 2, 8 ) = GTGCTT

Test snip() yes or no? yes

snip( 'ACAGTCT', 0, 3 ) = GTCT
snip( 'ACCAACCCCGG', 1, 4 ) = AACCCCGG
snip( 'TGAGTCCGAGGAGA', 5, 9 ) = TGAGTGGAGA
snip( 'GGGTGCTTCAGAG', 2, 8 ) = GGCAGAG

Test inverse() yes or no? yes

inverse( 'ACAGTCT' ) = TGTCAGA
inverse( 'ACCAACCCCGG' ) = TGGTTGGGGCC
inverse( 'TGAGTCCGAGGAGA' ) = ACTCAGGCTCCTCT
inverse( 'GGGTGCTTCAGAG' ) = CCCACGAAGTCTC
```

†. Function *generate*() can be implemented in different ways, so your output could be different. However, in your output you should see all four possible nucleotides occurring and with the first two nucleotides generated being the same.