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 exam is *pledged*. The *only* device you may access during the test is your own laptop.
- The last test page is blank. You can use it as scratch paper.
- 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 open *only* to the class website.
- PyCharm can *only* be used in the module implementation section.
- Functions you implement should *neither* produce output *nor* read input.
- Submissions will *not* be accepted after the test is over. It is incumbent on you to make the submissions. We *strongly* suggest you make submissions as you complete the problems.
- Grading of the functions will involve *different* test cases than the ones used for elaborative purposes in the problem descriptions and supplied testers.

```
Page 1   _______  /  14 points

Page 2   _______  /  12 points

Page 3   _______  /   8 points


Subtotal _______ /  34 points
```

Short answer

1. (14 points) Consider the following definitions for the questions below on this page.

```
 1: def f( x ) :
 2:     x = 100
 3:     return x
 5:
 6: def g( x ) :
 7:     y = 100
 9:     x = y
10:
11: def h( x ) :
12:     print( x )
```

- What best describes x in function f()?
  a) Argument
  b) Argument and parameter
  c) Local variable
  d) Parameter

- What best describes y in function g()?
  a) Argument
  b) Argument and parameter
  c) Local variable
  d) Parameter

- What best describes x in function h()?
  a) Argument
  b) Argument and parameter
  c) Local variable
  d) Parameter

- What does the following code segment output?

```
x = 10
f( x )
print( x )
```

- What does the following code segment output?

```
x = 10
g( x )
print( x )
```

- What does the following code segment output?

```
x = 10
x = f( x )
print( x )
```

- What does the following code segment output?

```
x = 10
x = g( x )
print( x )
```

2. (12 points) Consider the following definitions for the questions below on this page.

```
 1: def e( x ) :
 2:     x.append( 100 )
 3:
 4: def f( x ) :
 5:     y = x
 6:     y.append( 100 )
 7:
 8: def g( x ) :
 9:     x.append( 100 )
10:     return x
```

- What does the following code segment output?

```
a = []
e( a )
print( a )
```

- What does the following code segment output?

```
a = []
f( a )
print( a )
```

- What best describes the relationship of `x` and `y` in function `f()`?
  a) They reference the same list.
  b) They reference different lists whose elements are identical.

- What does the following code segment print?

```
a = []
g( a )
print( a )
```

- What does the following code segment print?

```
a = []
a = e( a )
print( a )
```

- What does the following code segment print?

```
a = []
a = g( a )
print( a )
```

3. (2 points) What should the comment be to describe the action of function `e()` with `dict` parameter `d`?

```
def e( d ) :
   n = len( d.keys() )
   return n
```

4. (2 points) What should the comment be to describe the action of function `f()` with `dict` parameter `d`?

```
def f( d ) :
   accumulator = 0
   for k in d.keys() :
      if ( d[ k ] == k ) :
         accumulator = accumulator + 1
   return accumulator
```

5. (2 points) What should the comment be to describe the action of function `g()` with `dict` parameter `d` and string parameter `v`?

```
def g( d, v ) :
   accumulator = []
   for k in d.keys() :
      if ( d[ k ] == v ) :
         accumulator.append( k )
   return accumulator
```

6. (2 points) What should the comment be to describe the action of function `h()` with `dict` parameter `d`, and string parameters `k` and `v`?

```
def h( d, k, v ) :
   if ( k not in d.keys() ) :
      d[ k ] = v
```

Implementation

7. (11 points) Implement module `m01.py`. The module defines a single function *snake*(). A simple tester `test-snake.py` is available.

   - Function *snake*() takes no parameters. The function returns the string *python* in all lower case. For example, the invocation `m01.snake()` evaluates to `'python'`.

   A run of tester `test-snake.py` should produce output

   ```
   python python
   ```

8. (11 points) Implement module `m02.py`. The module defines a single function *year*(). A simple tester `test-year.py` is available.

   - Function *year*() takes a single *int* parameter *y*. The function returns lower case string '*yes*', if parameter *y* is a USA presidential election year, and returns lower case string '*no*', otherwise. A year is a USA presidential year, if the year occurs after 1787 and the year is a multiple of four (i.e., the remainder of *y* divided by 4 is 0). For example, the invocation `m02.year( 1776 )` evaluates to `'no'` and the invocation `m02.year( 2020 )` evaluates to `'yes'`.

   A run of tester `test-year.py` should produce output

   ```
   no yes no yes
   ```

9. (11 points) Develop module `m03.py`. The module defines a function *series*(). A simple tester `test-series.py` is available.

   - Function *series*() takes a single *int* parameter *n*. The function returns the sum of the series

     $1/2^0 + 1/2^1 + 1/2^2 + 1/2^3 + \ldots + 1/2^n$

   For example, the invocation `m03.series( 25 )` evaluates to `1.9999999701976776`.

   A run of tester `test-series.py` should produce output

   ```
   1.0
   1.5
   1.9999999701976776
   1.9999999999999991
   2.0
   ```

   Observations: There are $n+1$ values in the series of numbers to be summed. Operator** is the Python exponentiation operator.

10. (11 points) Develop module `m04.py`. The module defines a function *join*(). A simple tester `test-join.py` is available.

    - Function *join*() takes two list parameters *s* and *t*. The function returns a new list. The elements of the new list are the elements of *s* followed by the elements of *t*. The function does not print anything and does not read any input. The function does not modify the contents of lists *s* and *t*. For example, the invocation `m04.join([ 1, 1, 1, 1 ], [ True, False, True ] ] )` evaluates to `[ 1, 1, 1, 1, True, False, True ].`

      A run of `test-join.py` tester should produce output

```
[1]
[True, False, True, 3.14, 1, 5, 9]
[1, 1, 1, 1, 'join']
```

11. (11 points) Develop `m05.py`. The module defines a function *count*(). A simple tester `test-count.py` is available.

    - Function *count*() takes a single parameter *sheet* representing a dataset (i.e., *sheet* is a list of rows with each row being a list of data values. The function returns the number of data values in the sheet (you cannot assume each row has the same number of elements). The function makes no changes to *sheet*. For example, the invocation `m05.series([ [ 1, 3, 5 ], [ 7, 9 ], [ 11 ], [ 12 ] ] )` evaluates to `7.`

      A run of tester `test-count.py` should produce output

```
0 6 12
```

12. (11 points) Develop module `m06.py`. The module defines a function *examine*(). A simple tester `test-examine.py` is available.

    - Function *examine*() takes three parameters *s*, *t*, and *n*, where *s* is a list of strings, *t* is a string, and *n* is an integer. The function returns a new list whose elements are the strings in list *s* that contains *n* occurrences of *t*. The function does not print anything and does not read any input. The function makes no changes to list *s*. For example, the invocation `m06.examine(['zipper','wrench', 'sneers','accent','citrus'],'en',1 )` evaluates to `['wrench','accent'].`

      A run of tester `test-examine.py` should produce output

```
['cheesy', 'measles', 'seeing']
['parent', 'carols']
['wigwam']
[]
```