Question 1. [13 marks]

Part (a) [4 marks]
Consider this program:

```python
def square(x):
    """(number) -> number
    """
    print('LINE A:', x)
    x = x * x
    print('LINE B:', x)
    return x

if __name__ == '__main__':
    x = 5
    print('LINE C:', x)
    square(x)
    print('LINE D:', x)
    square(x + 1)
    print('LINE E:', x)
```

Write what this program prints, one line per box. There are more boxes than you need; leave unused ones blank.

<table>
<thead>
<tr>
<th>Solution:</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINE C: 5</td>
</tr>
<tr>
<td>LINE A: 5</td>
</tr>
<tr>
<td>LINE B: 25</td>
</tr>
<tr>
<td>LINE D: 5</td>
</tr>
<tr>
<td>LINE A: 6</td>
</tr>
<tr>
<td>LINE B: 36</td>
</tr>
<tr>
<td>LINE E: 5</td>
</tr>
</tbody>
</table>

Part (b) [3 marks]
Consider this program:

```python
list1 = ['hat', 'glove', 'scarf']
list2 = list1[:]
list2.append('mitt')
list3 = list2
list3[1] = 'boot'

print(list1)
print(list2)
print(list3)
```

Write what this program prints, one line per box.

<table>
<thead>
<tr>
<th>Solution:</th>
</tr>
</thead>
<tbody>
<tr>
<td>['hat', 'glove', 'scarf']</td>
</tr>
<tr>
<td>['hat', 'boot', 'scarf', 'mitt']</td>
</tr>
<tr>
<td>['hat', 'boot', 'scarf', 'mitt']</td>
</tr>
</tbody>
</table>
Part (c) [4 MARKS]

Consider this function:

```python
def returns_what(x):
    """ (int) -> object """
    if x ** 2 <= 100:
        if x % 4 == 2:
            return "six"
        elif not x + 5 > 2:
            return x - 34
    else:
        if x < 0 and abs(x) > 3:
            return False
        else:
            return x / 10
```

In the table below are 4 calls to function `returns_what`. Beside each call, write the value returned by the call and that value’s type.

<table>
<thead>
<tr>
<th>Call</th>
<th>Return Value</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>returns_what(20)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>returns_what(10)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>returns_what(3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>returns_what(-100)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Part (d) [2 MARKS]

Consider this code:

```python
def f1(x, y):
    """ (int, int) -> int """
    print('In f1', x, y)
    return x * y

def f2(x, y):
    """ (int, int) -> int """
    print('In f2', x, y)
    return x + y

if __name__ == '__main__':
    print(f1(f2(1, 2), f1(3, 4)))
```

Write what this program prints, one line per box. There are more boxes than you need; leave unused ones blank.

```
In f2 1 2
In f1 3 4
In f1 3 12
36
```
Question 2. [8 marks]

This question has you write the bodies of two functions. Complete each function according to its docstring.

Part (a) [3 marks]

Note: you will most likely not need all of the space on this page.

def sum_squares(L):
    """ (list of list of number) -> list of number

    Return a list where each item is the sum of the squares of the items from
    the corresponding sublist of L.

    >>> sum_squares([[1, 2, 3], [7], [], [-8, 1]])
    [14, 49, 0, 65]
    """

    result = []
    for inner_list in L:
        sum = 0
        for item in inner_list:
            sum += item * item
        result.append(sum)

    return result
Part (b)  [3 marks]

def sum_squares_mutate(L):
    """(list of list of number) -> NoneType

    Replace each sublist of L with a single integer that is the sum of the
    squares of the sublist's items.

    >>> L = [[1, 2, 3], [7], [], [-8, 1]]
    >>> sum_squares_mutate(L)
    >>> L
    [14, 49, 0, 65]
    """

    for i in range(len(L)):
        sum = 0
        for item in L[i]:
            sum += item * item

        L[i] = sum

Part (c)  [2 marks]

You almost certainly have duplicate code between your two functions. Write a helper function that you
could have used to eliminate that duplicate code. Do not rewrite your functions for parts (a) and (b).
Question 3. [3 marks]

This method may be helpful for answering this question:

```python
isalpha(...)
S.isalpha() -> bool

Return True if all characters in S are alphabetic
and there is at least one character in S, False otherwise.
```

Consider this code. The `while` expression is missing. Write it so that the function does what the docstring says it should.

```python
def get_valid_password(min):
    """ (int) -> str

    Precondition: min >= 1

    Repeatedly ask a user to input a password until they enter one that is at
    least min characters long and that contains at least one non-alphabetic
    character. Return the resulting password.
    """

    while len(password) < min or not password.isalpha():
```
Question 4. [9 marks]

A permutation of a list is a list that has all the same items but possibly in a different order. Function is_permutation takes two lists and returns True iff its arguments are permutations of each other.

Part (a) [4 marks]

In the table below, we have outlined three test cases for is_permutation. Add four more test cases chosen to test the function as thoroughly as possible.

Note: there are more than four good answers to choose from.

<table>
<thead>
<tr>
<th>Test Case Description</th>
<th>list1</th>
<th>list2</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>empty lists</td>
<td>[]</td>
<td>[]</td>
<td>True</td>
</tr>
<tr>
<td>identical single item lists</td>
<td>['A']</td>
<td>['A']</td>
<td>True</td>
</tr>
<tr>
<td>different single item lists</td>
<td>[1]</td>
<td>[2]</td>
<td>False</td>
</tr>
<tr>
<td>longer permutation</td>
<td>[1, 2, 3]</td>
<td>[3, 1, 2]</td>
<td>True</td>
</tr>
<tr>
<td>no common items</td>
<td>[1, 2, 3]</td>
<td>[4, 5, 6]</td>
<td>False</td>
</tr>
<tr>
<td>doubled but not perm</td>
<td>[1, 2, 2]</td>
<td>[2, 1, 1]</td>
<td>False</td>
</tr>
<tr>
<td>some common</td>
<td>[1, 2, 3]</td>
<td>[2, 1, 4]</td>
<td>False</td>
</tr>
</tbody>
</table>

Part (b) [5 marks] Write the function body. (You do not need to add examples to the docstring.)

```python
def is_permutation(list1, list2):
    """ (list, list) -> bool
    Return True iff list1 is a permutation of list2.
    ""

    # algorithm 1 (make 2 copies sort them and compare
    list1_copy = list1[:]
    list1_copy.sort()
    list2_copy = list2[:]
    list2_copy.sort()
    return list1_copy == list2_copy

    # algorithm 2 (make copy of list2, remove each item from list1 in it)
    # Dan: this is buggy
    list2_copy = list2[:]
    for item in list1:
        if item in list2_copy:
            list2_copy.remove(item)
        else:
            # add more code here
```

Page 6 of 17
return False
return True
Question 5. [10 marks]

In Assignment 3, you worked with SQuEaL tables represented by dictionaries where each key is a string representing the name of a column and each value is a list of strings representing the items in that column from the top row to the bottom.

Part (a) [2 marks] Complete function `num_rows` below according to its docstring.

```python
def num_rows(table):
    """ (dict of {str: list of str}) -> int

    Precondition: all table columns have the same number of rows.

    Return the number of rows in table.
    """
    random_key = list(table.keys())[0]
    return len(table[random_key])
```

Part (b) [8 marks]

Another way to represent the contents of a table would be to use a tuple of two lists. The first one is a list of lists where each inner list is the contents of a row, and the second one is a list of column names indicating the order in which the data is stored in each row in the first list.

For example, the table shown below on the left is represented by both the dictionary and the tuple to the right.

Notice that the column order in the tuple representation is unpredictable, because dictionaries are not ordered.

<table>
<thead>
<tr>
<th>country</th>
<th>age</th>
<th>brand</th>
</tr>
</thead>
<tbody>
<tr>
<td>GB</td>
<td>34</td>
<td>b1</td>
</tr>
<tr>
<td>FR</td>
<td>21</td>
<td>b2</td>
</tr>
<tr>
<td>CA</td>
<td>19</td>
<td>b3</td>
</tr>
</tbody>
</table>

```

dict_data = {
    'country': ['GB', 'FR', 'CA'],
    'age': ['34', '21', '19'],
    'brand': ['b1', 'b2', 'b3']
}

tuple_data = ([['GB', 'b1', '34'], ['FR', 'b2', '21'], ['CA', 'b3', '19']],
              ['country', 'brand', 'age'])
```

On the next page, complete function `convert_to_row_lists`. It takes a SQuEaL table represented by a dictionary (as we did in A3) and returns the tuple of two lists representation. You may call function `num_rows` even if you were not able to correctly complete it.
def convert_to_row_lists(table):
    """ (dict of {str: list of str}) -> tuple of (list of list of str, list of str)
    Return a representation of table where each row is a list of strings.
The columns are sorted alphabetically by column name.
    >>> t = {'B': ['b1', 'b2'], 'A': ['a1', 'a2']}
    >>> convert_to_row_lists(t)
    [['a1', 'b1'], ['a2', 'b2']]"
    
    tablekeys = list(table.keys())
    row_list = []
    for i in range(num_rows(table)):
        row = []
        for key in tablekeys:
            row.append(table[key][i])
        row_list.append(row)
    return (row_list, tablekeys)
Question 6. [5 marks]

Part (a) [1 mark] The list below is shown after each pass of a sorting algorithm.

\[
[7, 1, 4, 5, 2, 6, 3] \text{ # initial list} \quad \text{Which sorting algorithm is being executed? (circle one)}
\]

[1, 4, 5, 2, 6, 3, 7] \text{ # after one pass} \quad (a) \text{ bubble sort}

[1, 4, 2, 5, 3, 6, 7] \text{ # after two} \quad (b) \text{ selection sort}

[1, 2, 4, 3, 5, 6, 7] \text{ # after three} \quad (c) \text{ insertion sort}

[1, 2, 3, 4, 5, 6, 7] \text{ # after four}

Part (b) [1 mark] The list below is shown after each pass of a sorting algorithm.

\[
[7, 1, 4, 5, 2, 6, 3] \text{ # initial list} \quad \text{Which sorting algorithm is being executed? (circle one)}
\]

[1, 7, 4, 5, 2, 6, 3] \text{ # after one pass} \quad (a) \text{ bubble sort}

[1, 2, 4, 5, 7, 6, 3] \text{ # after two} \quad (b) \text{ selection sort}

[1, 2, 3, 5, 7, 6, 4] \text{ # after three} \quad (c) \text{ insertion sort}

[1, 2, 3, 4, 7, 6, 5] \text{ # after four}

[1, 2, 3, 4, 5, 6, 7] \text{ # after five}

Part (c) [1 mark]

List \[6, 5, 2, 3, 7, 1, 4\] is being sorted using insertion sort. Fill in the blanks to show the list after the next two passes.

After one pass: \[6, 5, 2, 3, 7, 1, 4\]

After two passes: \[5, 6, 2, 3, 7, 1, 4\]

After three passes: \[\underline{\text{\quad \quad \quad \quad \quad}}\]

After four passes: \[\underline{\text{\quad \quad \quad \quad \quad}}\]

Part (d) [1 mark]

Some number of iterations of selection sort have been performed on a list, resulting in this list:
\[1, 2, 4, 5, 3, 8, 7, 6, 9\]

What is the maximum number of passes that could have been performed so far?

2

Part (e) [1 mark]

What additional piece of information do you know about the sorted section during selection sort that you don’t know during insertion sort? Solution: that the items are in their action spot.
Question 7. [10 marks]

Each code fragment in the table below operates on list $L$, which has length $k$ where $k$ is very large — at least in the tens of thousands. For each fragment, give an expression in terms of $k$ for how many times 'cheers!' is printed, and circle whether the behaviour is constant, linear, quadratic or something else.

<table>
<thead>
<tr>
<th>Code</th>
<th>How many times is 'cheers!' printed?</th>
<th>Complexity (circle one)</th>
</tr>
</thead>
</table>
| for i in range(len(L)):  
  print('cheers!')  
for item in L:  
  print('cheers!') | $2k$ | constant |
| | | linear |
| | | quadratic |
| | | something else |
| for item in L[10:]:  
  print('cheers!') | $10-k$ | constant |
| | | linear |
| | | quadratic |
| | | something else |
| i = 0  
while i < len(L)  
  print('cheers!')  
i = i + len(L) // 10 | $10$ | constant |
| | | linear |
| | | quadratic |
| | | something else |
| for item in L[1000:2000]:  
  print('cheers!') | $1000$ | constant |
| | | linear |
| | | quadratic |
| | | something else |
| for i in range(len(L))  
  for item in L[:i]:  
    print('cheers!') | $k \cdot (k-1) / 2$ | constant |
| | | linear |
| | | quadratic |
| | | something else |
Question 8. [12 marks]

Some people have wine cellars where they store bottles of wine. In this question, you will develop two
classes to keep track of the kinds of wine in a cellar, how many bottles of each kind there are, and whether
the person likes or dislikes each kind of wine.

Here is the header and docstring for class Wine.

class Wine:
    ""
    Information about a type of wine in our wine cellar, including the name,
    how many bottles there are, and whether the wine is liked.
    ""

Part (a) [2 marks]

Complete method __init__ for class Wine.

Note: you will most likely not need all of the space on this page.

    def __init__(self, name, num):
        """ (Wine, str, int) -> NoneType

        Record that there are num bottles of a wine named name. Assume that we like it.

        >>> w = Wine('Painter Bridge', 6)
        >>> w.name
        'Painter Bridge'
        >>> w.num_left
        6
        >>> w.like_it
        True
        """

        self.name = name
        self.num_left = num
        self.like_it = True
Part (b) [2 marks]

Here is the header, type contract, and description for method `drink` in class `Wine`. Add an example that creates a `Wine` object, drinks one of the bottles, and verifies that there is now one fewer bottle of that `Wine`. Also write the body of the method.

```python
def drink(self):
    """ (Wine) -> NoneType

    Precondition: there is at least 1 bottle left.

    Record that we drank one bottle of this Wine.
    """
    self.num_left = self.num_left - 1
```

Part (c) [3 marks]

Write an `__eq__` method in class `Wine` that allows us to compare two `Wine` objects to see if a wine is already in a cellar. Consider two wines equal if they have the same name. Follow the function design recipe.

```python
def __eq__(self, other):
    """ (Wine, Wine) -> bool

    Return whether this Wine and other are the same.
    """
    return self.name == other.name
```

Note: For the rest of this question, you should assume that there is a `__str__` method in class `Wine` that returns strings of this form: `Wine(Painter Bridge, 3, True)`
class WineCellar:
    """ Information about the bottles of wine in a wine cellar. """

Part (d)  [1 MARK] Complete method `__init__` in class WineCellar:

```python
def __init__(self):
    """ (WineCellar) -> NoneType

    Create a WineCellar with an empty wine list.

    >>> cellar = WineCellar()
    >>> cellar.wine_list
    []
    ""

    self.wine_list = []
```

Part (e)  [3 MARKS]
Complete method `add` in class WineCellar. Hints: `value in lst` evaluates to `True` when `value` is equal to an item in `lst`. Also, `lst.index(value)` returns the index of `value` in `lst`. If you need more space for this question, please use page 16 and note here that you have done so.

```python
def add(self, wine):
    """ (WineCellar) -> NoneType

    If wine is not already in this WineCellar, add it. Otherwise, increase the number of bottles in the equivalent Wine object that is already in the cellar.

    >>> cellar = WineCellar()
    >>> cellar.add(Wine('Painter Bridge', 6))
    >>> cellar.add(Wine('Painter Bridge', 5))
    >>> len(cellar.wine_list)
    1
    >>> str(cellar.wine_list[0])
    'Wine(Painter Bridge, 11, True)'
    ""

    if wine in self.wine_list:
        w = self.wine_list[self.wine_list.index(wine)]
        w.num_left += wine.num_left
    else:
        self.wine_list.append(wine)
```
Part (f)  [3 MARKS] Complete method `get_wines_to_replace` in class `WineCellar`:

```python
def get_wines_to_replace(self):
    """ (WineCellar) -> list of Wine
    Return a list of the wines that need replacing: the ones where there are 0 bottles left and that are liked.
    """
    res = []
    for wine in self.wine_list:
        if wine.number == 0 and wine.like_it:
            res.append(wine)
    return res
```
Short Python function/method descriptions:

```python
__builtins__:
    input([prompt]) -> str
    Read a string from standard input. The trailing newline is stripped. The prompt string,
    if given, is printed without a trailing newline before reading.
abs(x) -> number
    Return the absolute value of x.
int(x) -> int
    Convert x to an integer, if possible. A floating point argument will be truncated
    towards zero.
len(x) -> int
    Return the length of the list, tuple, dict, or string x.
max(iterable) -> object
    With a single iterable argument, return its largest item.
    With two or more arguments, return the largest argument.
min(iterable) -> object
    With a single iterable argument, return its smallest item.
    With two or more arguments, return the smallest argument.
print(value, ..., sep=' ', end='
') -> NoneType
    Prints the values. Optional keyword arguments:
    sep: string inserted between values, default a space.
    end: string appended after the last value, default a newline.
open(name[, mode]) -> file open for reading, writing, or appending
    Open a file. Legal modes are "r" (read), "w" (write), and "a" (append).
range([start], stop, [step]) -> list-like-object of int
    Return the integers starting with start and ending with stop - 1 with step specifying
    the amount to increment (or decrement).
    If start is not specified, the list starts at 0. If step is not specified,
    the values are incremented by 1.
dict:
    D[k] --> object
        Produce the value associated with the key k in D.
    del D[k]
        Remove D[k] from D.
    k in d --> bool
        Produce True if k is a key in D and False otherwise.
D.get(k) -> object
    Return D[k] if k in D, otherwise return None.
D.keys() -> list-like-object of object
    Return the keys of D.
D.values() -> list-like-object of object
    Return the values associated with the keys of D.
D.items() -> list-like-object of tuple of (object, object)
    Return the (key, value) pairs of D, as 2-tuples.
file open for reading:
    F.close() -> NoneType
        Close the file.
    F.read() -> str
        Read until EOF (End Of File) is reached, and return as a string.
    F.readline() -> str
        Read and return the next line from the file, as a string. Retain newline.
        Return an empty string at EOF (End Of File).
```
F.readlines() -> list of str
Return a list of the lines from the file. Each string ends in a newline.

list:
  x in L --> bool
  Produce True if x is in L and False otherwise.
  L.append(x) --> NoneType
  Append x to the end of the list L.
  L.index(value) --> int
  Return the lowest index of value in L.
  L.insert(index, x) --> NoneType
  Insert x at position index.
  L.pop() --> object
  Remove and return the last item from L.
  L.remove(value) --> NoneType
  Remove the first occurrence of value from L.
  L.reverse() --> NoneType
  Reverse *IN PLACE*.
  L.sort() --> NoneType
  Sort the list in ascending order.

str:
  x in s --> bool
  Produce True if and only if x is in s.
  str(x) --> str
  Convert an object into its string representation, if possible.
  S.count(sub[, start[, end]]) --> int
  Return the number of non-overlapping occurrences of substring sub in
  string S[start:end]. Optional arguments start and end are interpreted
  as in slice notation.
  S.find(sub[, i]) --> int
  Return the lowest index in S (starting at S[i], if i is given) where the
  string sub is found or -1 if sub does not occur in S.
  S.index(sub) --> int
  Like find but raises an exception if sub does not occur in S.
  S.isdigit() --> bool
  Return True if all characters in S are digits and False otherwise.
  S.lower() --> str
  Return a copy of the string S converted to lowercase.
  S.lstrip([chars]) --> str
  Return a copy of the string S with leading whitespace removed.
  If chars is given and not None, remove characters in chars instead.
  S.replace(old, new) --> str
  Return a copy of string S with all occurrences of the string old replaced
  with the string new.
  S.rstrip([chars]) --> str
  Return a copy of the string S with trailing whitespace removed.
  If chars is given and not None, remove characters in chars instead.
  S.split([sep]) --> list of str
  Return a list of the words in S, using string sep as the separator and
  any whitespace string if sep is not specified.
  S.strip() --> str
  Return a copy of S with leading and trailing whitespace removed.
  S.upper() --> str
  Return a copy of the string S converted to uppercase.