PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

DECEMBER 2013 EXAMINATIONS

CSC 108 H1F
Instructors: Craig and Gries

Duration — 3 hours

Examination Aids: None

Student Number: |_|_|_|_|_|_|_|_|_|

Family Name(s): ________________________________

Given Name(s): ________________________________

Do **not** *turn this page until you have received the signal to start.*
In the meantime, please read the instructions below *carefully.*

This final examination paper consists of 8 questions on 20 pages (including this one). *When you receive the signal to start, please make sure that your copy of the final examination is complete.*
Comments and docstrings are not required except where indicated, although they may help us mark your answers. They may also get you part marks if you can't figure out how to write the code.
You may not use `break` or `continue` on this exam.
If you use any space for rough work, indicate clearly what you want marked.

# 1:	______/13
# 2:	______/ 8
# 3:	______/ 3
# 4:	______/ 9
# 5:	______/10
# 6:	______/ 5
# 7:	______/10
# 8:	______/ 14
TOTAL:	______/ 72

Good Luck!

Question 1. [13 MARKS]

Part (a) [4 MARKS]

Consider this program:

```
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.

Part (b) [3 MARKS]

Consider this program:

```
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.

Part (c) [4 MARKS]

Consider this function:

```
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.

Call	Return Value	Return Type
`returns_what(20)`		
`returns_what(10)`		
`returns_what(3)`		
`returns_what(-100)`		

Part (d) [2 MARKS]

Consider this code:

```
def f1(x, y):
    """ (int) -> int """

    print('In f1', x, y)
    return x * y


def f2(x, y):
    """ (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.

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]
    """
```

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]
    """
```

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 str method may be helpful for answering this question:

```
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.

```
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.
    """

    prompt = 'Enter a password at least ' + min + ' characters long ' + \
             'with at least one non-alphabetic letter: '

    password = input(prompt)
    while                                                              :

        # The password wasn't good enough.  Ask again.
        password = input(prompt)

    return password
```

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.

Test Case Description	**list1**	**list2**	**Return Value**
empty lists	`[]`	`[]`	`True`
identical single item lists	`['A']`	`['A']`	`True`
different single item lists	`[1]`	`[2]`	`False`

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

```
def is_permutation(list1, list2):
    """ (list of object, list of object) -> bool

    Return True iff list1 is a permutation of list2.
    """
```

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.

```
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.

    >>> t = {'b.0': ['g', 'i', 'k'], 'b.1': ['h', 'j', 'l']}
    >>> num_rows(t)
    3
    """
```

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.

country	age	brand
GB	34	b1
FR	21	b2
CA	19	b3

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

(
 [['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 tuple where the first item is a representation of table where
    each row is a list of strings.  The second item in the tuple is the order
    in which the columns appear in the new representation.

    >>> t = {'B': ['b1', 'b2'], 'A': ['a1', 'a2']}
    >>> convert_to_row_lists(t)
    ([['a1', 'b1'], ['a2', 'b2']], ['A', 'B'])
    """
```

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] # intial list

[1, 4, 5, 2, 6, 3, 7] # after one pass
[1, 4, 2, 5, 3, 6, 7] # after two
[1, 2, 4, 3, 5, 6, 7] # after three
[1, 2, 3, 4, 5, 6, 7] # after four
```

Which sorting algorithm is being executed? (circle one)

(a) bubble sort

(b) selection sort

(c) insertion sort

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

```
[7, 1, 4, 5, 2, 6, 3] # initial list

[1, 7, 4, 5, 2, 6, 3] # after one pass
[1, 2, 4, 5, 7, 6, 3] # after two
[1, 2, 3, 5, 7, 6, 4] # after three
[1, 2, 3, 4, 7, 6, 5] # after four
[1, 2, 3, 4, 5, 6, 7] # after five
```

Which sorting algorithm is being executed? (circle one)

(a) bubble sort

(b) selection sort

(c) insertion sort

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:	______________________
After four passes:	______________________

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?

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?

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.

Code	How many times is 'cheers!' printed?	Complexity (circle one)

`for i in range(len(L)):`
`    print('cheers!')`
`for item in L:`
`    print('cheers!')`
		constant
		linear
		quadratic
		something else

`for item in L[10:]:`
`    print('cheers!')`
		constant
		linear
		quadratic
		something else

`i = 0`
`while i < len(L)`
`    print('cheers!')`
`    i = i + len(L) // 10`
		constant
		linear
		quadratic
		something else

`for item in L[1000:2000]:`
`    print('cheers!')`
		constant
		linear
		quadratic
		something else

`for i in range(len(L))`
`    for item in L[:i]:`
`        print('cheers!')`
		constant
		linear
		quadratic
		something else

Question 8. [14 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
        """
```

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.

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

        Precondition: there is at least 1 bottle left.

        Record that we drank one bottle of this Wine.




        """
```

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.

Note: For the rest of this question, you should assume that there is a `__str__` method in class `Wine` that 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`:

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

        Create a WineCellar with an empty wine list.

        >>> cellar = WineCellar()
        >>> cellar.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.

```
    def add(self, wine):
        """ (WineCellar, Wine) -> 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)'
        """
```

Part (f) [3 MARKS] Complete method get_wines_to_replace in class WineCellar:

```
def get_wines_to_replace(self):
    """ (WineCellar, Wine) -> list of Wine

    Return a list of the wines that need replacing: the ones where there are 0
    bottles left and that are liked.

    >>> cellar = WineCellar()
    >>> cellar.add(Wine('Painter Bridge', 0))
    >>> cellar.add(Wine('Santa Alicia', 1))
    >>> cellar.add(Wine('Cesari Amarone', 0))
    >>> w = Wine('Fuzion Shiraz Malbec', 0)
    >>> cellar.add(w)
    >>> w.like_it = False
    >>> res = cellar.get_wines_to_replace()
    >>> len(res)
    2
    >>> str(res[0]) == 'Wine(Painter Bridge, 0, True)'
    True
    >>> str(res[1]) == 'Wine(Cesari Amarone, 0, True)'
    True
    """
```

Total Marks = 70

[Use the space below for rough work. This page will **not** *be marked, unless you clearly indicate the part of your work that you want us to mark.]*

*[Use the space below for rough work. This page will **not** be marked, unless you clearly indicate the part of your work that you want us to mark.]*

[Use the space below for rough work. This page will **not** *be marked, unless you clearly indicate the part of your work that you want us to mark.]*

CONT'D...

Short Python function/method descriptions:

```
__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
  max(a, b, c, ...) -> object
    With a single iterable argument, return its largest item.
    With two or more arguments, return the largest argument.
  min(iterable) -> object
  min(a, b, c, ...) -> object
      With a single iterable argument, return its smallest item.
      With two or more arguments, return the smallest argument.
  print(value, ..., sep=' ', end='\n') -> 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.
```