PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

AUGUST 2014 EXAMINATIONS

CSC 108 H1Y
Instructor: Craig Hagerman

Duration — 3 hours

Examination Aids: None

Warning: You must get 40% or above on this exam to pass the course

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.*

You must get 40% or above on this exam to pass the course; otherwise, your final course grade will be no higher than 47.
This final examination paper consists of 10 questions on 22 pages (including this one). *When you receive the signal to start, please make sure that your copy of the final examination is complete.* Fill in the identification section above, and write your student number where indicated at the bottom of every page (except page 1).
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:	_____/ 5
# 3:	_____/10
# 4:	_____/ 4
# 5:	_____/ 3
# 6:	_____/ 6
# 7:	_____/ 7
# 8:	_____/ 5
# 9:	_____/10
# 10:	_____/12
TOTAL:	_____/75

Good Luck!

CONT'D...

Question 1. [13 MARKS]

Part (a) [3 MARKS]

Consider this program:

```
L = [3, 6, 9]
X = L.reverse()
Y = L[:]
L.append(100)

print(X)
print(id(Y) == id(L))
print(L)
```

Write what this program prints, one line per box.

Part (b) [4 MARKS]

Consider this program in file `words.py`:

```
def add_yo(word):
    print(__name__)
    word = word + 'YO'
    print ("The new word is:", word)
    return word

if __name__ == '__main__':
    word = 'Hey'
    print('Original word:', word)
    add_yo(word)
    print('Current word:', word)
    word = add_yo(word) + '?'
    print('Final word:', word)
```

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

Part (c) [2 MARKS]

Consider this code:

```
def f1(x, y):
    print('#1:', x, y)
    return x * y


def f2(x, y):
    print('#2:', x, y)
    return x - y

if __name__ == '__main__':
    print(f1(f2(7, 4), f1(2, 5)))
```

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

Part (d) [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)`		

Question 2. [5 MARKS]

The following expressions all have problems, so unfortunately they will all result in an error and will not be evaluated. Explain the errors in the table below.

Expression	Briefly explain the error
`metal = "adamantium"` `metal[10]`	
`["2014", " August "].extend(13)`	
`[" Computer ", " Science "].strip()`	

`founder_to_company = {`
`['Gates'] : "Microsoft",`
`['Zuckerberg'] : Facebook }`	

`"Amy is " + 19 + " years old"`	

Question 3. [10 MARKS]

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

```
The chr function will likely be helpful. It takes an int and returns the letter that
corresponds to that letter. e.g. The letter 'A' has value 65, 'B' is 66 and so on
up to 'z' which has value 122

>>> help(chr)
chr(...)
        chr(i) -> Unicode character

        Return a Unicode string of one character with ordinal i; 0 <= i <= 0x10ffff.

>>> chr(65)
'A'
>>> chr(66)
'B'
```

Part (a) [5 MARKS]

```
def decode_letters(L):
    """ (list of list of ints) -> list of str

    Precondition: the list and sublists are non-empty
                  int values of the sublist are between 65 and 122 inclusive

    Return a list where each item is a string made up of the equivalent characters
    represented by the numbers in the corresponding sublist of L.

    >>> decode_letters([[65], [102, 97, 116], [99, 97, 116] ])
    ['a', 'fat', 'cat']
    >>> decode_letters([[67, 111, 109, 112], [83, 99, 105], [114, 111, 99, 107, 115]])
    ['Comp', 'Sci', 'rocks']
    """
```

Part (b) [3 MARKS]

```
def decode_letters_mutate(L):
    """ (list of list of ints) -> NoneType

    Precondition: the list and sublists are non-empty
                  int values of the sublist are between 65 and 122 inclusive

    Replace each sublist of L with a string made up of the equivalent characters
    represented by the numbers in the corresponding sunlist of L.

    >>> L = [[65], [102, 97, 116], [99, 97, 116] ]
    >>> decode_letters(L)
    >>>> L
    ['a', 'fat', 'cat']
    """
```

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 4. [4 MARKS]

You saw your csc108 instructor's grade book one day and noticed that he keeps all the final grades in a plain text file with just two items on each line: a CDF id and a final grade separated by commas. It looks like this:

```
c301234,76.5
c498765,80.2
```

One day when the instructor steps out of his office you decide to use your new Python skills to write a function to change your grade. Fortunately, you know which file the grades are in. Your code runs, but unfortunately it won't do what you wanted. It contains **four** bugs which will result in the function execution not matching its docstring description. In the space provided re-write the buggy lines with the bugs fixed. Do no add or remove lines

```
def perfect_A(filename, my_cdfid):
    """ (list of str, str) -> str

    Preconditions:
    (1) files_list is not empty.
    (2) Each file in files_list contains comma-separated data
    (3) All files in files_list contain passwords of different lengths.
    (4) my_cdfid is guaranteed to be in one of these files.

    Replace the line in file filename that starts with my_cdfid to record a grade
    of 100. That is, replace the relevant line with my_cdfid,100. Save the file with
    the updated information and return True if my_cdfid was found.
    """
```

#Code	If you think a given line contains a bug, rewrite it. Otherwise leave it blank.
`def perfect_A(filename, my_cdfid):`	
`    found_me = True`	
`    f = open(filename, 'r')`	
`    data = ''`	
`    for line in f:`	
`        line = line.strip()`	
`        csv_values = line.split(',')`	
`        if csv_val == 2:`	
`            if csv_val[0] = my_cdfid:`	
`                found_me = True`	
`                line = my_cdfid + ',' + '100'`	
`            data += line + '\n'`	
`    f.close()`	
`    f = open(filename, 'r')`	
`    f.write(data)`	
`    f.close()`	
`    return found_me`	

Question 5. [3 MARKS]

This str method may be helpful for answering this question:

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

  Return true if all characters in the string are digits
  and there is at least one character, 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_guess(min, max):
    """ (int, int) -> str

    Repeatedly ask a user to input a number until they enter one that is valid.
    A valid guess is a number (containing only digits) which is between min
    and max (both inclusive). Return the resulting guess
    """

    prompt = 'Enter a number between ' + str(min) + '  and  ' + str(max) +  ': '

    password = input(prompt)

    while [                                                    ]:

        # It was not a valid guess. Ask again.
        password = input(prompt)

    return password
```

Question 6. [6 MARKS]

Consider this code:

```
def rank_grades(grade_list, avg):
    """ (list of float, float) -> dict of {str: list of float}

    Return a new dictionary where each item in grade_list appears as an
    item in one of the dictionary's values lists:
    - each item less than avg is in a list associated with key 'below'.
    - each item equal to avg is in a list associated with the key 'average'
    - each item greater than avg is in a list associated with key 'above'
    If there are no items associated with one of 'below', 'average' or 'above',
    then that string should not appear as a key in the new dictionary.
    """
```

In the table below, we have outlined two test cases for `sort_gradees`. Add six more test cases chosen to test the function thoroughly.

Test Case Description	grade_list	avg	Return Value
no grades	[]	65.2	{}
one low grade	[53.1]	65.2	{'below': [53.1]}

Question 7. [7 MARKS]

Corporations typically divide the year into 3-month periods called 'Quarters'. The first quarter (Q1) is January, February, March. The second quarter (Q2) is April, May, June. The third quarter (Q3) is July, August, September and the forth quarter (Q4) October, November, December. In the past Company XYZ inc. prepared quarterly reports with letter to indicate the month it was prepared. For example the report 2009f was created in February 2009. Since some months begin with the same letter they used a unique letter for each month such that: Q1 = jfm; Q2 = ame; Q3 = ygs; Q4 = ond.

Now Company XYZ needs to change all of their old reports into a numeric integer representation. Thus 2009f would become 20090, 2012y would become 20123, and so on.

Part (a) [6 MARKS] Write the body of the following function according to its docstring description.

```
def numeric_report(report_name, month_to_quarter):
    """ (str, list of str) -> int

    Preconditions:
    - len(month_to_quarter) == 4
    - report_name contains only digits and lowercase letters
    - each letter in report_name is guaranteed to appear in month_to_quarter once
    - month_to_quarter may contain letters not in report_name

    Return a numeric name that corresponds to the alphanumeric report_name.
    Each letter from report_name is replaced with 1 + the index of the item from
    month_to_quarter that contains the letter. Digits are not replaced.

    >>> numeric_report("2009f", ["jfm", "ame", "ygs", "ond"])
    20091
    >>> numeric_report("2009d", ["jfm", "ame", "ygs", "ond"])
    20094
    """
```

Part (b) [1 MARK] Instead of representing the month to quarter mapping as a `list of str`, representing it as which of the following types would make implementing this function easier? (circle one)

(a) `list of list of str` (b) `dict of {int: str}` (c) `dict of {str: int}`

Question 8. [5 MARKS]

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

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

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

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.

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

[2, 5, 4, 6, 1, 3, 7] # after one pass
[2, 4, 5, 1, 3, 6, 7] # after two
[2, 4, 1, 3, 5, 6, 7] # after three
[2, 1, 3, 4, 5, 6, 7] # 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 `[7, 1, 5, 6, 3, 2, 4]` is being sorted using selection sort. Fill in the blanks to show the list after the next two passes.

After one pass: `[1, 7, 5, 6, 3, 2, 4]`
After two passes: `[1, 2, 5, 6, 3, 7, 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:
`['A', 'B', 'D', 'C', 'E', 'F']`

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

Part (e) [1 MARK]

In what case will insertion sort have the worst running time? (i.e.. for what kind of list)

Question 9. [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 Hodor! is printed, and circle whether the behaviour is constant, linear, quadratic or something else.

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

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

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

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

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

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

Question 10. [12 MARKS]

In the summer there are many *music festivals* with *bands* performing in various events. In this question you will develop two classes to keep track of *Bands*s and *Festivals*.

Here is the header and docstring for class Band.

```
class Band:
    """
    Information about a particular performer including the band's name, the genre
    of their music, the number of members in the band and how much they
    charge for a performance.
    """
```

Part (a) [2 MARKS]

Complete method `__init__` for class Band.

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

```
    def __init__(self, name, genre, num_sets, cost):
        """ (Band, str, str, int, int) -> NoneType

        Record the band's name, genre of music, number of sets they will perform num_sets
        and the cost of their performance (in dollars)

        >>> b = Band("Hannaford Silver Street Band", "jazz", 4, 1300)
        >>> b.name
        'Hannaford Silver Street Band'
        >>> b. genre
        'jazz'
        >>> b.num_sets
        4
        >>> b.rate
        1300
        """
```

Part (b) [2 MARKS]

There are sometimes additional costs (surcharges) involved in paying for a musical performance, such as paying for accommodations or security. Here is the header, type contract and description for the method `surcharge` in class `Band`. Add an example that creates a Band object, adds an additional charge of 250 to the cost of the band's performance, and verifies that the cost of the band's performance has been updated. Also write the body of the method.

```
def surcharge(self, extra_cost):
    """ (Band) -> NoneType

    Record that the cost of this Band is increased by extra_cost









    """
```

Part (c) [3 MARKS]

Write an `__eq__` method in class `Band` that allows us to compare two `Band` objects to see if they are equal. Consider two bands 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 `Band` that that returns strings of this form: `''Band("Hannaford Silver Street Band", "jazz", 4, 1300)'`

```
class Festival:
    """ Information about the musical bands in a event. """
```

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

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

        Create a Festival with an empty musical band list

        >>> f = Festival()
        >>> f.band_list
        []
        """
```

Part (e) [3 MARKS]

Complete method `add` in class `Festival`. Possibly helpful 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`.

```
    def add(self, band):
        """ (Festiva, Bandl) -> NoneType

        If band is not already in this Festival, add it.  Otherwise, increase the
        number of sets in the equivalent Band object that is already in the event.

        >>> f = Festival()
        >>> f.add(Band('Ziggy Marley, "reggae", 1,  5300))
        >>> f.add(Band('Ziggy Marley, "reggae", 2, 5300))
        >>> len(f.band_list)
        1
        >>> str(f.band_list[0])
        Band('Ziggy Marley, "reggae", 3, 5300)
        """
```

Part (f) [3 MARKS] Complete method `get_similar` in class `Festival`:

```
def get_similar(self, other_band):
      """ (Festival, Band) -> list of Band

      Return a list of Bands that play the same genre of music as band.

      >>> f = Festival()
      >>> f.add(Band("Ziggy Marley", "reggae" 1, 12, 7300.00))
      >>> f.add(Band("Buju Banton", "reggae", 1, 12, 7300.00))
      >>> f.add(Band("Hannaford Silver Street Band", "jazz", 45, 1300.00)
      >>> sim = f.get_similar(Band("Messenjah", "reggae", 1, 9, 900.00))
      >>> len(sim)
      2
      >>> sim[0] == Band("Ziggy Marley", "reggae" 1, 12, 7300.00)
      True
      >>> sim[1] == Band("Buju Banton", "reggae", 1, 12, 7300.00)
      True
       """
```

Total Marks = 75

[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.]*

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.
```