PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO SCARBOROUGH

December 2017 EXAMINATIONS

CSCA20H3
Duration — 3 hours

Examination Aids: None

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

Last (Family) Name(s): _____

First (Given) Name(s): _____

Instructor: Bretscher

---

*Do **not** turn this page until you have received the signal to start.*
(In the meantime, please fill out the identification section above, and read the instructions below *carefully*.)

This final examination consists of 6 questions on 16 pages (including this one). *When you receive the signal to start, please make sure that your copy of the 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 do not need to put `import` statements in your answers.

If you use any space for rough work, indicate clearly what you want marked.

Assume all input is valid unless otherwise indicated; there is no need to error-check.

Good Luck!

MARKING GUIDE

# 1: _____/	32
# 2: _____/	12
# 3: _____/	20
# 4: _____/	20
# 5: _____/	12
# 6: _____/	4
TOTAL: _____/	100

CONT'D...

Question 1. [32 MARKS]

Complete the following functions according to their docstring descriptions.

Part (a) [4 MARKS]

```
def dog_years(y):
    ```(int) -> int
    Given the age of a dog in human years, return the age of the dog in
    dog years calculated by multiplying the human years by 7.
    >>> dog_years(10)
    70
    '''
```

### Part (b) [4 MARKS]

| Age | Category |
|---|---|
| $< 18$ | minor |
| $\geq 18$ and $< 65$ | adult |
| $\geq 65$ | senior |

Table 1: Age Chart

```
def age_category(age):
    ```(int) -> str
    Given an age, use the Age Chart above to return the correct category for the age.
    >>> age_category(10)
    `minor'
    '''
```

Question 1. (CONTINUED)

Part (c) [4 MARKS]

```
def process(f1, f2, filename):
    '''(file, file, str) -> NoneType
    Given two open files, f1 and f2,  and a str filename, open filename and write to the
    open file all of the lines that f1 and f2 do not have in common.'''
```

Part (d) [4 MARKS]

```
def contents(L):
    '''(list of (list of int)) -> list of int
    Given a nested list L where each inner list contains ints, return
    a list of ints that contains all of the ints from the inner lists of L.
    >>> contents([[1, 2, 3], [4, 5], [6]])
    [1, 2, 3, 4, 5, 6]
    '''
```

Part (e) [4 MARKS]

```
def add_grades(names, grades):
    '''(list of str, list of float) -> list of str
    Given a list of names and a corresponding list of grades, return a list of strings
    where each string is the name, colon, grade.  You may assume that the 2 given
    lists have the same length.
    >>> add_grades(['Anna', 'Joe', 'Bob'], [90.0, 76.6, 55.1])
    ['Anna:90.0', 'Joe:76.6', 'Bob:55.1']
    '''
```

Part (f) [4 MARKS] A call to the function `guess_my_num(10)` behaves like this:

```
Enter a guess: 5
Too low! Enter a guess: 20
Too high! Enter a guess: 11
Too high! Enter a guess: 10
Correct! you took 4 guesses.
```

Complete the function below.

```
def guess_my_num(x):
    '''(int) -> NoneType
    Repeatedly ask the user for a guess until they enter number x.
    If they guess a number higher than x, print 'Too high'. If they guess
    a number lower than x, print 'Too low'. When the user guesses the
    correct number, print the number of guesses it took.
    '''
```

Part (g) [4 MARKS]

```
def crack_the_code(msg, code):
    '''(str, str) -> list of list
    Given a string msg, and an encoded version of this string, code,
    return a list of two lists where the first list contains the unique letters from
    the message and the second list contains the corresponding encoded version
    of the letter.  I.e., each letter from msg appears once in the first sublist and the
    corresponding letter from code appears once at the same index in the second sublist.

    For example, below the letter 'e' maps to 'b'.

    >>> crack_the_code('hello there', 'abccd eabfb')
    [['h','e', 'l', 'o', 't', 'r'], ['a', 'b', 'c', 'd', 'e', 'f']]
```

Part (h) [4 MARKS]

```
def decode(code, decoder):
    '''(str, list of list) -> str
    Given an encoded message, code, and a list of 2 lists where each character
    in the second list is the encoded version of the corresponding character in the
    first list.  Return a str that is the decoded version of the str code.  Notice
    that this means that each character in code can be found in the second inner list.
    The returned string will consist of the corresponding characters from the first
    inner list.

    >>> decode('bob', [['c', 't', 'i', 'z'], ['o', 'i', 'b', 'a']])
    'ici'
    >>> decode('abcd', [['h','e', 'l', 'p'], ['a', 'b','c','d']])
    'help'
    '''
```

Question 2. [12 MARKS]

Part (a) [9 MARKS]

The function `mystery` below is labelled with line numbers on the left-hand side. Consider the execution of this function, during the call:

```
mystery('neat')
```

With the argument above, line 10 executes 2 times.

```
   def  mystery(s):
     '''(    ) ->




     >>> mystery('esso')


     '''
1    t = ''
2    i = 0
3    c = ''
4    while i < len(s)
5        c = s[i]
6        if c.isupper():
7              t = c.lower() + t + c.lower()
8        else:
9              t = c.upper() + t + c.upper()
10       i = i + 2
11   return t
```

In the table below, list the values of the variables `c`, `i` and `t` before line 10 has executed and after it has executed each of the two times.

Variable	before the while loop	After line 10 has executed the 1st time	After line 10 has executed the 2nd time
c			
t			
i			

Part (b) [3 MARKS]

Fill in the missing docstring for function `mystery`. Make sure you complete the doctest example and the types.

Question 3. [20 MARKS]

Consider a CSV file `exams.csv` in *exam time table format*. A few lines from this file are shown on the right. Complete the following functions. You may assume that any files read are already open and are in *exam time table format* with all headers removed.

```
Code,Date,Time
CSCA20,2017-12-13,14:00
CSCA67,2017-12-16,9:00
CSCA08,2017-12-18,9:00
```

Part (a) [5 MARKS]

```
def create_course_dict(openfile):
    ‘‘‘(file) -> dict
    Given an open csv file in exam time table format,  return a course dictionary
    where each key is a course and each corresponding value is a list [date, time].
    For example, the entries for the lines in exams.csv would be:
    {’CSCA20’ :  [‘2017-12-13’, ‘14:00’], ’CSCA67’ :  [‘2017-12-16’, ‘9:00’],
    ’CSCA08 : [’2017-12-18’, ’9:00’]}
    ’’’
```

Part (b) [5 MARKS]

```
def create_date_dict(d):
   ‘‘‘(dict) -> dict
   Given a course dictionary where each key is a course code and the value is a
   list a date str and a time str, return a date dictionary where each key is a date
   and each corresponding value is a list of courses with an exam on that date.
   >>> create_date_dict({’CSCA20’:[‘2017-12-13’,‘14:00’], ’CSCA67’:[‘2017-12-16’,‘9:00’]})
   {’2017-12-13’ : [’CSCA20’], ‘2017-12-16’ : [’CSCA67’]}
```

Question 3. (CONTINUED)

Part (c) [5 MARKS]

```
def output_chrono(d):
    '''(dict) -> NoneType
    Given a course dictionary where each key is a course and each corresponding
    value is a list with two items, a date string and a time string. Print out the
    course codes for the exams in chronological order one course code per line,
    i.e., from earliest date to latest date.  You only need to consider the date, not the
    time.  You may use previous functions even if you were unable to complete them.
    >>> output_chrono({'CSCA20':['2017-12-13', '14:00'], 'CSCA67':['2017-12-16', '9:00']})
    CSCA20
    CSCA67
    '''
```

Part (d) [5 MARKS]

```
def busiest_day(d):
   '''(dict) -> str
    Given a date dictionary where each key is a date and each corresponding
    value is a list of courses with an exam on that date. Return the date with
    the most exams.  If more than one date has the most exams return one of the dates.
    >>> busiest_day({'2017-12-13' : ['CSCA20', 'CSCC73'], '2017-12-16' : ['CSCA67']})
    '2017-12-13'
    '''
```

Question 4. [20 MARKS]

Recall the database that we used in A3. Similar tables are defined below. Beside each table definition is an example. Write each query below. You only need to write the `SELECT` statement. You do not need to call `run_query` etc.

Table Name	Columns	**Examples**
Course	(ID, Code, Section, Instructor)	`Course(101, 'CSCA20', 'L01', 'Anna')` `Course(102, 'CSCA20', 'L02', 'Joe')`
Timetable	(ID, Date, Time)	`Timetable(101, '2017-12-13', '14:00')` `Timetable(102, '2017-12-13', '14:00')`
Location	(ID, Room)	`Location(102, 'IC120')` `Location(102, 'IC130')`

Table 2: Database Tables.

Part (a) [5 MARKS]

Find the course `instructors` for the course `Code` "MATA02". Your query should not return duplicates.

Part (b) [5 MARKS]

Return each course code and the location of the exams for that course code.

Part (c) [5 MARKS]

Return each course code with more than one instructor. It doesn't matter if the IDs are the same or different.

Part (d) [5 MARKS]

Return the course code and number of exam rooms for each course `ID`.

Question 5. [12 MARKS]

In this question we continue working with the database described in the previous question. Your task is to complete a function to check for exam conflicts in Part (a) and then write a main program to get user specified courses and check for exam conflicts in Part (b). We give you the `run_query` function here for your reference.

```
def run_query(db, query, args=None):
    """(str, str[, tuple]) -> list of tuple
    Execute the SQL statement stmt with arguments args in the database
    db. Return the result of executing a fetchall().
    """
    conn = sqlite.connect(db)
    cur = conn.cursor()

    if args is None:
        cur.execute(query)
    else:
        cur.execute(query, args)

    result = cur.fetchall()
    conn.commit()
    cur.close()
    conn.close()
    return result
```

Part (a) [4 MARKS]

Design and implement a function `check_conflict` that returns True if two courses have conflicting exam schedules. Return False if the exams are conflict free. A conflict is defined to be on the same day at the same time.

```
def check_conflict(db, course1, course2):
    ‘‘‘(str, str, str) -> bool

    Return True if course1 and course2 have an exam conflict and False otherwise.
    ’’’
```

Part (b) [8 MARKS]

Design and implement a program that asks a user for their courses and then checks that their exams are conflict free. Your program should repeatedly ask the user for a course until the user hits return (enters nothing). Your program should then either output "Your exams are conflict free!" or "Alert: Course A and B have a conflict." where A and B are course codes. Your program should output this for each pair of conflicts.

Here is an example when two pairs of courses have conflicts:

```
Please enter a course: CSCA20
Please enter a course: BIOD40
Please enter a course: MUSB02
Please enter a course: MATA01
Please enter a course:

Alert: Course CSCA20 and MUSB02 have a conflict.
Alert: Course BIOD40 and MATA01 have a conflict.
```

Add your code below:

```
# the database file to use
db = 'exams.db'

# Get user courses
# FILL IN YOUR CODE HERE
```

```
# check for each pair of courses whether there is a conflict.
# you use may use the function check_conflict even if you were
# unable to complete it
# FILL IN YOUR CODE HERE
```

Question 5. (CONTINUED)

More space for your answer.

Question 6. [4 MARKS]

Below are two statements to be executed in the shell. For each one, write Python statements to go before it that will guarantee that the statement will run without error. It doesn't matter what the code does, just that it runs. There are many correct answers for each. Below is an example.

Code:

```
a = p(b % c)
```

Statements to go before this one to guarantee that it will run without error:

```
def p(n):
    return n + 1
b = 45
c = 13
```

Part (a) [2 MARKS]

Code:

```
for x in y[1:3]:
    y[0].append(x)
```

Statements to go before this one to guarantee that it will run without error:

Part (b) [2 MARKS]

Code:

```
print(blah()[y][z])
```

Statements to go before this one to guarantee that it will run without error:

*[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.]*

A parameter is optional if its name is enclosed in square brackets, `[...]`. If a parameter is omitted, any preceding comma is also omitted.

```
__builtins__:
  print(item, ..., [sep=...], [end=...])
    Print the items, separated by blanks unless sep is specified, followed by a newline
    unless end is specified.
  len(x) -> int
    Return the length of the list, tuple, dict, or string x.
  abs(number) -> number
    Return the absolute value of the given number.
  round(number[, ndigits]) -> number
    Round a number to a given precision in decimal digits (default 0 digits).
    This returns an int when called with one argument, otherwise the
    same type as the number. ndigits may be negative
  max(L) -> value
    Return the largest value in the list L (or the largest of the parameters if there are several).
  min(L) -> value
    Return the smallest value in L.
  input([prompt]) -> string
    Read a string from standard input. The trailing newline is stripped. The prompt string,
    if given, is printed without a trailing newline before reading.
  range([start], stop, [step]) -> range object that can produce a sequence of ints
    The sequence of ints starts with start and ends with stop - 1; step specifies
    the amount to increment (or decrement). If start is not specified, the sequence
    starts at 0. If step is not specified, the values are incremented by 1.
  open(name, [mode]) -> file object
    Return a file object connected to the file with the given name.  Legal modes are
    "r" (read), "w" (write), and "a" (append).

float:
  float(x) -> floating point number
    Convert a string or number to a floating point number, if possible.

int:
  int(x) -> int
    Convert a string or number to an int, if possible. A floating point parameter is
    truncated towards zero.

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

This sheet will not be marked. You may remove it.

```
dict:
  k in D -> bool
    Return True if the dictionary D has an item with key k.
  D.keys() -> object that can produce a sequence of keys
    Return a sequence object that lists the keys in D.
  D.values() -> object that can produce a sequence of values
    Return a sequence object that lists the values in D.
  D.items() -> object that can produce a sequence of key-value pairs
    Return a sequence object that lists the (key, value) 2-tuples in D.
  D.get(k, [v]) -> value
    Return the value associated with the key k in D. If k is not a key in D, return v.
  D.popitem() ->  (k, v), remove and return some (key, value) pair as a
    2-tuple; but raise KeyError if D is empty.

str:
  x in s -> bool
    Return True if x is in s and False otherwise.
  str(x) -> string
    Return the string representation of the object x, if possible.
  s.isdigit() -> bool
    Return True if all characters in s are digits and len(s) >= 1, False otherwise.
  s.islower() / s.isupper() -> bool
    Return True if all cased characters in s are lowercase/uppercase and there is
    at least one cased character in s, False otherwise.
  s.lower() / s.upper() -> string
    Return a copy of s converted to lowercase/uppercase.
  s.startswith(prefix[, start[, end]]) -> bool
    Return True if s starts with the specified prefix, False otherwise. With optional start,
    test s beginning at that position. With optional end, stop comparing s at that position.
  s.find(sub, [start, [end]]) -> int
    Return the lowest index in s (starting at s[start], if start is given, and stopping before
    s[end], if end is given) where the string sub is found, or -1 if sub does not occur in s.
  s.index(sub, [start, [end]]) -> int
    Like find but raises an exception if sub does not occur in s[start:end].
  s.count(sub, [start, [end]]) -> int
    Return the number of non-overlapping occurrences of substring sub in string s[start:end].
  s.replace(old, new, [count]) -> string
    Return a copy of string s with all occurrences of the string old replaced with the string
    new. If count (an int) is provided, only the first count occurrences are replaced.
  s.split([sep]) -> list of strings
    Return a list of the words in s, using sep to separate words. If sep is not specified,
    separate words with any whitespace string.
  s.strip([chars]) -> string
    Return a copy of s with leading and trailing whitespace removed. If chars is provided,
    remove characters in chars instead of whitespace.

math:
  sqrt(x) -> float
    Return the square root of x.
```

This sheet will not be marked. You may remove it. End of Examination