UNIVERSITY OF TORONTO SCARBOROUGH

Wnter 2016 EXAMINATIONS

CSC A20H
Duration — 2 hours 45 mins

No Aids Allowed

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

Last Name: _____

First Name: _____

---

*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. Answer each question directly on the examination paper, in the space provided. (If you need more space for one of your solutions, use the blank page and indicate **clearly** which part of your work should be marked.)

Be aware that concise, well thought-out answers will be rewarded over long rambling ones. Also, unreadable answers will be given zero (0) so write legibly.

In the programming questions, assume all input is valid. Make sure to indent your code properly. Your code will be marked based on correctness and style.

Some questions in the exam follow on other questions. When answering a question you may assume that all functions in the preceding questions have been implemented correctly, and you may make use of these functions.

1: _____/20
2: _____/ 5
3: _____/15
4: _____/12
5: _____/18
6: _____/ 5

TOTAL: _____/75

Good Luck!

Question 1. [20 MARKS]

Part (a) [4 MARKS]

Rewrite the code below using a single `return` statement.

```
if a or (b and not c):
    return False
else:
    return True
```

Part (b) [4 MARKS]

Compete the following function.

```
def round_list(float_list):
    ‘‘‘(list) -> list of list
    Given a list of floats, return a list of lists that is the same
    length as float_list, where each element is a 2-element list containing
    the float from the corresponding element of float_list and its value
    rounded to the nearest integer.
    Requirement: the list has at least one element. Hint: you may use the built
    in function round().
    >>> round_list([3.4, 5.0, 6.9])
    [[3.4, 3], [5.0, 5], [6.9, 7]]
     ’’’
```

Part (c) [4 MARKS]

Complete the following function **without** using `sort()`.

```
def second_smallest(data):
    ```(list of int) -> int
    Given a list of positive integers, return the second smallest
    integer in the list. If there are less than 2 elements in the list,
    return -1.
    >>> second_smallest([4, 2, 1, 3])
    2
    >>> second_smallest([2])
    -1
    '''
```

### Part (d) [4 MARKS]

Complete the docstring examples for the function `count_char`. Make sure your three examples are meaningful. Then, implement the function `count_char` *without* using `str` method (count).

```
def count_char(text, letter):
    ```(str, str) -> int
    Return the number of times the single character string letter,
    occurs in the string text.
    >>>


    >>>


    >>>



    '''
```

Part (d) (CONTINUED)

```
def count_char(text, letter):
    ‘‘‘(str, str) -> int
    Return the number of times the single character string letter,
    occurs in the string text.
    ’’’
```

Part (e) [4 MARKS]

Consider the following function `mystery()`

```
def mystery ():
   data = ‘12345Blast-off’
   i = 0
   while data[i:i+1].isdigit():
       data = data[0:i] + ’   ’ + data[i:]
       i = i + 2
   return data
```

For each iteration of the while loop, list the values of the variables `i` and `data`.

i	data
0	’12345Blast-off’

Question 2. [5 MARKS]

Write a function that will print decreasing numbers in a triangle shape, like below. Include a docstring.

```
>>> print_triangle(3)
012
01
0

>>> print_triangle(4)
0123
012
01
0
>>> print_triangle(5)
01234
0123
012
01
0
```

```
def print_triangle(size)
    '''









    '''
```

Question 3. [15 MARKS]

For this question you will be writing a program that reads in price data from a file and enters it into a dictionary to be accessed by other functions. You will write functions described below using the file and dictionary. Each line of the input file has the following format:

```
product,date,price
```

The `product` is a string, the `date` is a string of the format 'YYYYMMDD' and the `price` is a float. You may assume that data in the file is separated by commas and that there are not extra spaces surrounding the commas.

Complete the following functions according to their docstring descriptions. As required on your assignments, you must avoid duplicate code by having your functions call others when appropriate. A portion of your mark will be based on code reuse. Note that if you **cannot** complete one function you can still complete the others.

Part (a) [5 MARKS]

```
def file_to_dict(f):
    ```(file) -> dict
    Given an open input file return a dictionary whose key:value pairs
    are product:dict where the dict is a dictionary with key:value pairs
    date:price associated with that product. For example, a single entry
    dictionary may look like this: {'apples':{'20160101':1.99, '20160601':2.99}}.
    '''
```
```

Part (b) [5 MARKS]

```
def date_dict(d):
   ```(dict) -> dict
   Parameter d is a dict where each key is a product and each value is
   a dict with key:value pairs date:price. Return a new dict where each key is a date
    and each value is a dict with key:value pairs product:price.

   >>> d1 = {'apples':{'20160101':1.99, '20160601':2.99}, 'pears':{20160101:3.99}}.
   >>> d2 = date_dict(d1)
   >>> d2 == {'20160101':{'apples':1.99, 'pears':3.99}, '20160601':{'apples':2.99}}
   True
   '''
```

**Part (c)** [5 MARKS]

```
def most_expensive_month(f, product):
    ```(file, str) -> str
    Given an open input file and a product, return the date when the product
    is the most expensive. HINT: you may want to call one of your previous functions.
    '''
```

Question 4. [12 MARKS]

A music database is being used for keeping track of a music collection for an online store. Two tables are being used, `Album` and `Tracks`. The ID column in each table refers to an *album ID* (not a track ID). The price for any one track is always $1.99. Album prices can vary.

```
CREATE TABLE Album(ID INTEGER, Artist TEXT, Album TEXT, Price REAL)

CREATE TABLE Tracks(Title TEXT, ID INTEGER, Price REAL)
```

An example of rows in the `Album` table:

1	'The Beatles'	'One'	24.99
2	'The Beatles'	'Yellow Submarine'	12.99
3	'Blue Rodeo'	'Greatest Hits'	14.99

An example of rows in the `Tracks` table:

'Strawberry Fields'	1	1.99
'5 Days in May'	3	1.99
'Help!'	2	1.99

Write an SQL query to retrieve the data described below from the database. Note: you do not need to write any code to connect to the database; you only need to write the SQL query (e.g., 'SELECT ...').

Part (a) [3 MARKS]

The album titles by the band 'The Beatles'.

Part (b) [3 MARKS]

The track titles from all the albums by artist 'Taylor Swift'.

Part (c) [3 MARKS]

The artist names, album title, price for the album and the cost of buying all the tracks on the album individually.

Part (d) [3 MARKS]

Pairs of artists and album titles where the album title is the same.

Question 5. [18 MARKS]

This question has you write a program to manage a music collection and uses the database from Question 4. The question has two parts and assumes the database has already been created and populated with the following tables:

```
CREATE TABLE Album(ID INTEGER, Artist TEXT, Album TEXT, Price REAL)

CREATE TABLE Tracks(Title TEXT, ID INTEGER, Price REAL)
```

You may find the following function helpful:

```
def run_query(db, q, args=None):
    '''Run the query q on database db and return the result.  If provided,
    args is a tuple of arguments for the query.'''

    con = sqlite.connect(db)
    cur = con.cursor()
    if args == None:
        cur.execute(q)
    else:
        cur.execute(q, args)
    result = cur.fetchall()
    cur.close()
    con.close()
    return result
```

Part (a) [9 MARKS]

Complete functions `get_max_id` (below) and `insert_data` (on the next page) according to their docstring descriptions.

```
def get_max_id(db):
    '''Return the largest album ID in the database. '''
```

```
def insert_data(db, table_name, data_tuple):
    '''Insert a new row into the table table_name (either Album or Tracks) of
    database db with the data in the tuple data_tuple.'''

    # Connect to the database
    con = sqlite.connect(db)
    cur = con.cursor()

    if table_name == 'Album':
        # Insert the new row into the Album table
        # COMPLETE THIS LINE:
        cur.execute(                                                    )

    else:
        # Insert the new row into the Tracks table
        # COMPLETE THIS LINE
        cur.execute(                                                    )



    cur.close()
    con.commit()
    con.close()
```

Part (b) [9 MARKS]

Write a program that allows a user to add an album to their music collection. Your program will first prompt the user to enter the album name, artist name and price. Using this information, your program should update the database with the new album and then repeatedly ask the user for track titles and add them to the database.

The database, `'music.db'` has already been created and populated with data.

A sample interaction with a user may look like this:

```
Enter album name: All in Time
Enter artist name: Jim Cuddy
Enter album price: 19.99
```

Your program should insert a new row in the Album table with this album title, artist and price.

Your program will then repeatedly prompt for track information until the user enters 'Q'. For example:

```
Enter track title or Q: Trouble
Enter track title or Q: Disappointment
Enter track title or Q: Q
```

After your program completes, the Album table should have a new entry that could look like this:

100	'Jim Cuddy'	'All in Time'	19.99

The Tracks table should include a new line for each of the new tracks. For example it might look like this:

'Trouble'	100	1.99
'Disappointment'	100	1.99

We have provided some starter code to get you started and lay out the steps you need to complete. Make sure to fill in the code as indicated by the comments and to use the variable names defined. You should call the functions `get_max_id` and `insert_data` as needed.

```
if __name__ == "__main__":

    db = 'music.db'

    # Prompt user for album, artist and price
    #
    # ADD CODE BELOW









    # Add the album to the database by
    # getting a unique ID for the album and then inserting the album
    # ADD CODE BELOW
```

```
    # Ask the user for the track titles and add to the database
    # ADD CODE BELOW
```

Question 6. [5 MARKS]

Write a program that continually asks the user for a word and only displays (`prints`) those words that have not been displayed before. For example if a sequence of words were

```
hello, good, bye, hello, holidays, happy, sad, holidays, happy, merry, happy, ...
```

then the following would be displayed:

```
hello
good
bye
holidays
happy
sad
merry
...
```

You will be graded on the *correctness* and *simplicity* of your code. Write your program here:

```
if __name__ == '__main__':
```

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 if you wish.

```
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.

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 if you wish.

Total Marks = 75

This sheet will not be marked. You may remove it if you wish.