PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

AUGUST 2015 EXAMINATIONS

CSC 108 H1Y
Instructor: Papadopoulou

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.*

You must get 40% or above on this exam to pass the course (at least 32 out of 80); otherwise, your final course grade will be no higher than 47.

This final examination paper consists of 11 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.*

- Comments and docstrings are not required except where indicated, although they may help us mark your answers.
- You do not need to put `import` statements in your answers.
- No error checking is required: assume all user input and all argument values are valid.
- 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: _____/ 5
2: _____/ 5
3: _____/ 4
4: _____/ 7
5: _____/14
6: _____/ 6
7: _____/ 7
8: _____/ 7
9: _____/ 6
10: _____/ 6
11: _____/13

TOTAL: _____/80

Good Luck!

CONT'D...

Question 1. [5 MARKS]

Fill in the boxes below with what you would see in your Python shell if you were to type the following expressions. If that code would result in an error, then write ERROR and provide a brief explanation in the corresponding box.

```
>>> 'Good Luck!'.count('o')
```

(a)

```
>>> 'Do Your Best'.isupper()
```

(b)

```
>>> date_to_day = {['August', 4, 2015]: 'Tuesday',
                   ['August', 17, 2015]: 'Monday'}
```

(c)

```
>>> # Assume output_file is a file open for writing
>>> output_file.write('Hi!\n')
```

(d)

```
>>> L1 = [10, 20, 30, 20]
>>> val = L1.remove(20)
>>> print(L1)
```

(e)

Question 2. [5 MARKS]

Complete the two docstring examples of this function in the boxes below according to their returned values. There may be several correct answers. Providing any of them will earn you full marks.

Then, write the body of the function according to the provided docstring description.

```
def count_even_numbers(num1, num2):
    """ (int, int) -> int

    Precondition: num1 <= num2

    Return the number of all even numbers (integers) with values between num1 and num2,
    including these two boundaries.

    >>> 
    0

    >>> 
    2
    """

    # Write the body of the function below.
```

Question 3. [4 MARKS]

Here is a function definition (the docstring examples have been left out on purpose):

```
def mystery(word_list, index):
    """ (list of str, int) -> int

    Precondition: index is a valid index for word_list.
    """

    if 2 <= index <= 4:
        word_list[index] = 'Python'
    elif index == 3:
        word_list[2] = 'programming'
    else:
        word_list.append('Yes')

    print(word_list)
    return index
```

Fill in the table below:

Function Call	Return Value	Output (i.e., what is printed)
`L1 = ['a', 'random', 'word:', 'mix']` `mystery(L1, 3)`		
`L2 = ['nested', 'calls']` `mystery(L2, mystery(L2, 0))`		
`L3 = ['CSC108', '-', '-', 'course']` `mystery(L3, mystery(L3, 1) + 1)`		

Question 4. [7 MARKS]

Part (a) [5 MARKS]

You are trying to schedule a movie night with some friends and decide to use your Python programming skills to set-up a poll! Complete the body of the function below according to its docstring description.

```
def create_availability_table(num_names, num_timeslots, initial_state):
    """ (int, int, str) -> list of list of str

    Precondition: num_names > 0 and num_timeslots > 0

    Return an availability table with num_names rows and num_timeslots columns
    where each table entry contains the initial_state message.
    Each table row contains all the availability information of a given person.
    Modifying a single entry of this table should not impact the contents of another entry.

    >>> create_availability_table(2, 3, 'unknown')
    [['unknown', 'unknown', 'unknown'],
     ['unknown', 'unknown', 'unknown']]
    >>> create_availability_table(1, 1, 'free')
    [['free']]
    """
```

Part (b) [2 MARKS]

One of your friends also decided to write the `create_availability_table` function, but when you tested his code you ran into the following incorrect behavior: even though the table had the correct dimensions and initial contents, when you'd set your preference for a given timeslot, this timeslot would also be updated for all your friends (i.e., all rows in the table).

You decided to use the Python Shell to demonstrate this behavior to your friend. Write one or two expressions next to the Shell prompts below to explain to your friend the cause of this bug. Add a one sentence comment before that.

```
>>> movie_schedule = create_availability_table(4, 5, 'unknown')
>>> # Assume you know your row number as well as the timeslot each
>>> # table column corresponds to.

>>> movie_schedule[0][1] = 'free'
>>> print(movie_schedule)
[['unknown', 'free', 'unknown', 'unknown', 'unknown']
 ['unknown', 'free', 'unknown', 'unknown', 'unknown']
 ['unknown', 'free', 'unknown', 'unknown', 'unknown']
 ['unknown', 'free', 'unknown', 'unknown', 'unknown']]

>>> #



>>>



>>>
```

Question 5. [14 MARKS]

In this question you decided it's better to use a dictionary to keep track of your friends free time slots.

Part (a) [6 MARKS]

Complete the following function according to its docstring description.

```
def build_person_to_free_timeslots(avail_list):
    """ (list of (str, int) tuple) -> dict of {str: list of int}

    Preconditions:
    - In the provided list of tuples, avail_list, the first element of each tuple
    is a person's name, while the second tuple element is a person's free timeslot.
    - The same person's name might appear in multiple tuples in avail_list.
    - Duplication is possible; the same timeslot might appear as a second element
    in multiple tuples in avail_list with the same person as a first element.

    Return a dictionary where each key is a person's name from avail_list and
    associated with that key is an unordered values list containing all
    the free timeslots for that person.
    There should be no duplicates in the available timeslots values list
    in the dictionary; each timeslot should appear only once for a given person.

    >>> friends_schedule = [('Maria', 8), ('James', 3), ('Lisa', 7), ('Maria', 3),\
                            ('Lisa', 7)]
    >>> build_person_to_free_timeslots(friends_schedule)
    {'Maria': [8, 3], 'James': [3],  'Lisa': [7]}
    """
```

Part (b) [5 MARKS]

In order to find the timeslot that works for all your friends you need to invert the dictionary created in Part (a). Complete the following helper function according to its docstring description:

```
def invert_dictionary(person_to_timeslots):
    """ dict of {str: list of int} -> dict of {int: list of str}

    Return a new dictionary that is the inverse of person_to_timeslots.
    (See example below)

    >>> dict1 = {'Maria': [8, 3], 'James': [3],  'Lisa': [7]}
    >>> invert_dict1 = invert_dictionary(dict1)
    >>> invert_dict1 == {8: ['Maria'], 3: ['Maria', 'James'], 7: ['Lisa']}
    True
    """
```

Part (c) [3 MARKS]

Now all is left is to find the timeslot when most of your friends are free. We started writing this code for you. Fill in the missing code in the boxes below. You are allowed to call the helper function from Part (b).

```
# Assume a variable person_to_timeslots exists which refers
# to a "friends to free timeslots" dictionary object.

current_best_timeslot = -1 # -1 indicates no timeslot was found
max_avail_friends = 1 # You need at least two people.


tmp_dict = [                                                    ]
for key in [                                                    ] :
    if max_avail_friends < [                                    ] :
        max_avail_friends = [                                   ]
        current_best_timeslot = key

if current_best_timeslot == -1:
    print('No common timeslot found!')
else:
    print('Best timeslot is', current_best_timeslot, 'and it works for',
          max_avail_friends, 'friends!')
```

Question 6. [6 MARKS]

Write the body of the following function according to its docstring description.

```
def find_cheap_flights(flights_file, origin_city, max_cost):
    """ (file open for reading, str, int) -> list of str

    Preconditions:
    - flights_file contains 0 or more lines with the following format:
      city of origin-destination city-cost
      For example:
      Toronto-Vancouver-320
    - No city in the file flights_file will contain a '-' in its name.
    - max_cost > 0
    - No city of origin - destination city pair will be duplicated in flights_file file.

    Return a sorted list (in ascending order of name) of all the possible destination
    cities in file flights_file with city of origin origin_city and flight cost
    less than or equal to max_cost.
    """
```

Question 7. [7 MARKS]

Write the body of the following function according to its docstring description. Remember you are **not** allowed to use `break` or `continue`.

```
def find_flight_destinations(flights_info_file, origin_city):
    """ (file open for reading, str) -> list of str

    Preconditions:
    - flights_info_file contains 0 or more city profiles.
    - Profiles are separated from each other via a blank line.
    - The first line of a profile is a city of origin, followed by 0 or more
      possible flight destinations from that city (one per line).
    - A profile for origin_city may or may not appear in flights_info_file.
    - A given city will not have more than 1 profile in flights_info_file.
    - If a city has a profile in flights_info_file, then it will not appear
      as a destination city in another profile in flights_info_file.

    Return a list of all possible flight destinations from origin_city
    according to file flights_info_file.
    """
```

Question 8. [7 MARKS]

Part (a) [5 MARKS]

The goal of the following code snippet is to identify and print the index of the first element in list L that is equal to or greater than value `threshold`. If no such element exists in L, then this code should print the number of elements in list L. You can assume that list L is a list of int and contains at least one element.

Unfortunately, the code below contains a bug (in the specified line) and thus does not display the expected behavior.

```
i = 0
while L[i] < threshold: # Buggy line
    i = i + 1
print(i)
```

(i) Complete the table below with an appropriate test-case that demonstrates why this code is incorrect and a brief explanation of the bug.

L	value	Expected final value for i	Brief explanation of the bug

(ii) Fix the buggy line in the box below so the code snippet matches the specification.

(iii) For a list L of a given length N, how many times does `i` get incremented with respect to the length N?

- In the best case: _____
- In the worst case: _____

(iv) In the worst case, what is the runtime complexity of this algorithm with respect to the length N of the list L? Circle the correct answer:

(a) constant (b) linear (c) quadratic (d) something else

Part (b) [2 MARKS]

Assume list L has an even number of elements.

```
val = len(L) // 2
for i in range(len(L) // val):
    print(i)
```

(i) How many times is the print statement in the code above executed? _____

(ii)How does the number of calls to `print` scale with respect to the length of the list L? Circle the correct answer:

(a) constant (b) linear (c) quadratic (d) something else

Question 9. [6 MARKS]

Part (a) [1 MARK]

In which of the following sorting algorithms you will know the smallest element of the list after the first pass? Circle all that apply. Do not guess as there is a penalty for each incorrect answer.

(a) bubble sort (b) selection sort (c) insertion sort

Part (b) [2 MARKS]

In one of the algorithms learned in this course, we swap the element at index `index_of_smallest` with the element at index `i`. Assuming both variables refer to valid list indices, write the code that performs this swap in the space below for a list L.

For full marks you should write this in **one** line of code. In all cases, you will not need any more than 3 lines of code.

_____

_____

_____

Part (c) [1 MARK]

List `[3, 8, 4, 3, 5]` is being sorted using **insertion sort**. Fill in the blanks to show the list after the next two passes.

Initial list: `[3, 8, 4, 3, 5]`

After one pass: `[3, 8, 4, 3, 5]`

After two passes: _____

After three passes: _____

Part (d) [2 MARKS]

List `['this', 'was', 'sorting']` is being sorted using **selection sort**. Fill in the blanks to show the list after the next two passes.

Initial list: `['this', 'was', 'sorting']`

After one pass: _____

After two passes: _____

Question 10. [6 MARKS]

Consider this code:

```
def count_even_numbers(num_list):
    """ (list of int) -> int

    Return the number of even elements in num_list.
    """
```

In the table below, we have outlined one test case for `count_even_numbers`. Add six more test cases chosen to test the function thoroughly. Do not include duplicate cases.

Test Case Description	num_list	Return Value
empty list	[]	0

Question 11. [13 MARKS]

In this question, you will develop two classes to keep track of paintings and art collections.

Here is the header and docstring for class Painting.

```
class Painting:
    """
    Information about a particular painting including the name, the painter,
    the year it was created, and one keyword.
    """
```

Part (a) [2 MARKS]

Complete method __init__ for class Painting.

```
    def __init__(self, painting_name, painter, creation_year, painting_keyword):
        """ (Painting, str, str, int, str) -> NoneType

        Record the painting's name painting_name, the painting's painter,
        the year creation_year this painting was created, and a keyword
        painting_keyword.

        >>> painting = Painting('Red Maple', 'A.Y. Jackson', 1914, 'landscape')
        >>> painting.name
        'Red Maple'
        >>> painting.painter
        'A.Y. Jackson'
        >>> painting.year
        1914
        >>> painting.keyword
        'landscape'
        """
```

Part (b) [2 MARKS] Here is the header, type contract, and description for method `later_work` in class `Painting`. Add one docstring example that creates two `Painting` objects, calls this method and shows the return value.

Also write the body of this method.

```
def later_work(self, other_painting, diff_years):
    """ (Painting, Painting, int) -> bool

    Return True iff this painting was created more than diff_years
    after the year painting other_painting was created.

    >>>




    """

    # Complete the method body in addition to the docstring example.
```

Part (c) [2 MARKS]

Write a `__str__` method in class `Painting` to return a string representation of a Painting object.

```
def __str__(self):
    """ (Painting) -> str

    Return a string representation of this painting.

    >>> painting = Painting('Mona Lisa', 'Leonardo da Vinci', 1506, 'portrait')
    >>> str(painting)
    'Mona Lisa created by Leonardo da Vinci (1506) - portrait'
    """
```

Here is the header and docstring for class ArtCollection.

```
class ArtCollection:
    """ Information about an art collection. """
```

Part (d) [2 MARKS] Complete method __init__ in class ArtCollection:

```
    def __init__(self, collection_name):
        """ (ArtCollection, str) -> NoneType

        Create an ArtCollection with name collection_name and an
        initially empty collection of paintings.

        >>> collection1 = ArtCollection('Group of Seven')
        >>> collection1.name
        'Group of Seven'
        >>> collection1.paintings
        []
        """
```

Part (e) [2 MARKS]

Complete method add_painting in class ArtCollection.

```
    def add_painting(self, painting):
        """ (ArtCollection, Painting) -> int

        Add painting to this art collection and return the number of paintings
        in this collection including the one just added.

        >>> collection1 = ArtCollection('Group of Seven')
        >>> p1 = Painting('Red Maple', 'A.Y. Jackson', 1914, 'landscape')
        >>> collection1.add_painting(p1)
        1
        >>> p2 = Painting('Afternoon Sun', 'Lawren Harris', 1924, 'landscape')
        >>> collection1.add_painting(p2)
        2
        """
```

Part (f) [3 MARKS] Complete method `__eq__` in class `ArtCollection`:

```
def __eq__(self, other_collection):
    """ (ArtCollection, ArtCollection) -> bool

    Precondition: Assume that the two collections will never both be empty
    at the same time (i.e., without any paintings).

    Return True iff this art collection has the same number of paintings as
    collection other_collection and the last painting added to this
    collection has the same keyword as the last painting added to
    collection other_collection.

    >>> p1 = Painting('Red Maple', 'A.Y. Jackson', 1914, 'landscape')
    >>> p2 = Painting('Afternoon Sun', 'Lawren Harris', 1924, 'landscape')
    >>> a1 = ArtCollection('Landscapes')
    >>> a1.add_painting(p1)
    1
    >>> a1.add_painting(p2)
    2
    >>> a2 = ArtCollection('Portraits')
    >>> a1 == a2
    False
    """
```

[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.
```


CONT'D...

```
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.

file open for writing:
  F.close() -> NoneType
    Close the file.
  F.write(x) -> int
    Write the string x to file F and return the number of characters written.

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.extend(iterable) -> NoneType
    Extend list L by appending elements from the iterable. Strings and lists are
    iterables whose elements are characters and list items respectively.
  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 *IN PLACE*.

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.isalpha() -> bool
  Return True if and only if all characters in S are alphabetic
  and there is at least one character in S.
S.isdigit() -> bool
  Return True if all characters in S are digits
  and there is at least one character in S, and False otherwise.
S.islower() -> bool
  Return True if and only if all cased characters in S are lowercase
  and there is at least one cased character in S.
S.isupper() -> bool
  Return True if and only if all cased characters in S are uppercase
  and there is at least one cased character in S.
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([chars]) -> str
  Return a copy of S with leading and trailing whitespace removed.
  If chars is given and not None, remove characters in chars instead.
S.upper() -> str
  Return a copy of the string S converted to uppercase.
```

Total Marks = 80

END OF FINAL EXAMINATION