CSC A08 2013 Final Exam
Duration — 3 hours
Aids allowed: none

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

Instructor: Brian Harrington

Last Name: _____ **First Name:** _____

---

*Do **not** turn this page until you have received the signal to start.*
Please fill out the identification section
and read all instructions before starting.
Good Luck!

This midterm consists of 9 questions on 24 pages (including this one). *When you receive the signal to start, please make sure that your copy is complete.* Proper documentation is required for all functions and code blocks. If you use any space for rough work, indicate clearly what you want marked. There are some extra blank pages which you can use for rough work, and an API at the end of the test. You may detach the API, but anything detached from the test will **not** be marked. Please read all questions thoroughly before starting on any work.

1: _____/ 5

2: _____/ 5

3: _____/15

4: _____/10

5: _____/15

6: _____/10

7: _____/ 5

8: _____/20

9: _____/ 5

TOTAL: _____/90

[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.]

Question 1. [5 MARKS]

Write the output of the following code in the space provided. Remember that `print(x, end = '')` prints the value of `x` and prevents the next print statement from advancing to the next line.

```
my_list = ['*','x','-','#']
count = 0
for i in my_list:
    for j in range(0,len(my_list)):
        if(j + count > 2):
            print(i, end='')
        else:
            print('+', end = '')
    print('')
    count += 1
```

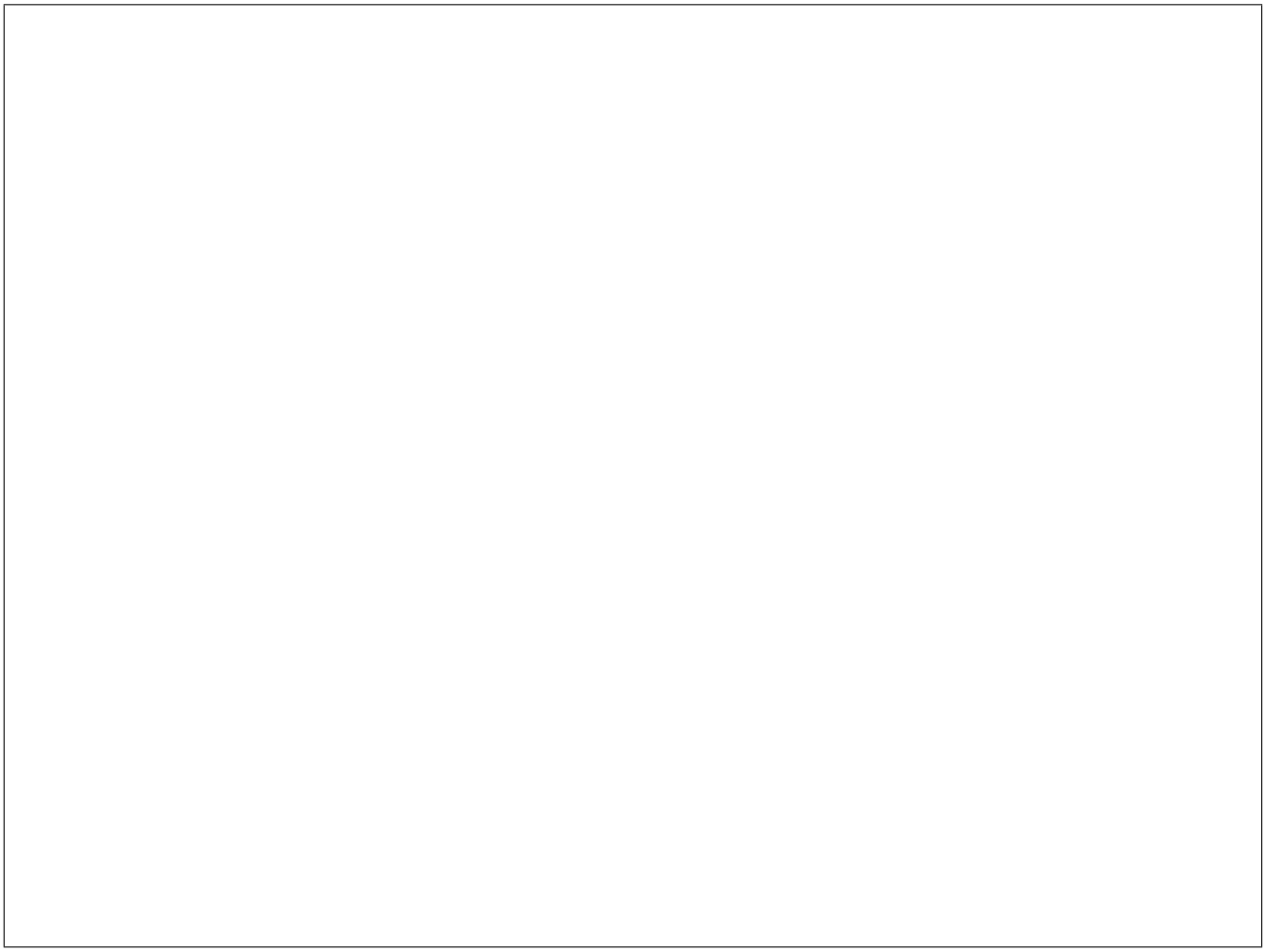
[Use the space below for rough work. This page will not be marked unless you clearly indicate the part of your work that you want us to mark.]

Question 2. [5 MARKS]

Write the output of the following code in the space provided. You may choose the order in which the keys of dictionaries are processed.

```
my_text = "I-loved-CSCA08"
my_dict = {}
for i in range(0,len(my_text)-1):
    my_dict[my_text[i]] = my_text[i + 1]

for letter in my_dict:
    print(letter, " - ", my_dict[letter])
```



[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.]

Question 3. [15 MARKS]

Write the output of the following code in the space provided

```
def my_func(input_list):
    input_list[1][2] = 99
    input_list[0] = 100

def my_func2(input_list):
    my_list = input_list[:]
    my_list[1][2] = 99
    my_list[0] = 100
    return my_list

def my_func3(input_list):
    input_list[1] = [1, [2, 3, 4], 5]
    my_func2(input_list[1])
    return input_list

x = 7
print("STEP1:",x)
y = x
x = 9
print("STEP2:",x,"-",y)
x = 1
y = [1, 2, 3]
z = ['a','b','c']
print("STEP3:",y[x],"-",z[y[x]])
x = 1
y = [1, 2, 3]
z = [y,x,y]
y = [4,5,6]
print("STEP4:",y,'-',z)
x = [1, [3, 2, 1], [1, 2]]
y = my_func(x)
print("STEP5:",x,"-",y)
x = [1, [3, 2, 1], [1, 2]]
y = my_func2(x)
print("STEP6:",x,"-",y)
x = [1, [3, 2, 1], [1, 2]]
y = my_func3(x)
z = my_func3(y)
print("STEP7:",x,"-",y)
x = [1, [3, 2, 1], [1, 2]]
y = my_func(my_func2(my_func3(x)))
print("STEP8:",x,"-",y)
```

[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.]

Question 4. [10 MARKS]

Write a function called `triple_cartesian_product` that takes three lists and prints the cartesian product of the lists (each item in each list should be paired with every item in every other list). You do not need to provide examples in your DocString. We have provided an example for you below:

```
>>> triple_cartesian_product([True, False], [1, 2], ['a', 'b'])
True 1 a
True 1 b
True 2 a
True 2 b
False 1 a
False 1 b
False 2 a
False 2 b
```

An example `grades.csv` file

```
name,grade,exercises
brian,100,8
nick,99,7
paco,8,2
anya,75.4,6
richard,0,0
```

[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.]

Question 5. [15 MARKS]

Write a program (not a function) that reads a csv file called `'grades.csv'` whose format can be seen on the previous page, and prints the average number of exercises completed by students who achieved final grades of 80 or higher. You may make the following simplifying assumptions:

- The column headers will always be exactly as given here (though not necessarily in the same order)
- `grade` will always be a float
- `exercises` will always be an integer
- There will be no spaces anywhere in the file, or commas inside the `name` fields
- `grades.csv` will be a valid `csv` file (e.g., equal number of rows for each column)

[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.]

Question 6. [10 MARKS]

Complete the following code according to its DocString, where `table` is the same table representation used in assignment 2. A sample run of the function is given below. You may assume that the input is a valid table (at least one column, all columns have the same number of rows)

```
>>> my_table = {'A':['a','b','c'],'X':['x','y','z']}
>>> print_csv(my_table)
X,A
x,a
y,b
z,c
```

```
def print_csv(input_table):
    '''(table) -> NoneType
    Print table in .csv format. The order of the columns is not guaranteed.
    '''
```

[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.]

Question 7. [5 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.

In the table below, we have outlined three test cases for `is_permutation`. Add five more test cases chosen to test the function as thoroughly as possible.

Note: there are more than five good answers to choose from.

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

[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.]

Question 8. [20 MARKS]

For this question, be extra careful to read the whole question before you start writing anything.

Part (a) [15 MARKS]

Write classes for the `Child` and `Santa` data types. Each child has a name, and an address, and can either believe in Santa or not. Santa objects don't take any parameters when they're created. Children have a method called `naughty_or_nice`, which tells Santa how they've behaved this year. All children start out as nice, and three other methods: `misbehave` (which makes them naughty), `learn_lesson` (which makes them nice), and `give_present`, which takes a string representing a present.

`Santa` has the method `give_presents`, which takes a `set` of children and a `set` of presents and gives each child a random present from the set iff they've been nice.

Part (b) [5 MARKS] Write a small piece of code that creates three children (you can choose their names, etc) and a Santa object, and tells Santa to give the children presents (you may also choose the presents they get).

For both parts of this question, you do not have to provide any docstrings or comments (though if you can't make something work, internal comments may get you part marks).[1]

[1] Now aren't you glad you read the whole question?

[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:
You may tear this page off, but if you do so, you must not include any work on it (front or back) that you wish to have marked.

```
__builtins__:
  abs(number) -> number
    Return the absolute value of the given number.
  max(a, b, c, ...) -> value
    With two or more arguments, return the largest argument.
  min(a, b, c, ...) -> value
    With two or more arguments, return the smallest argument.
  isinstance(object, class-or-type-or-tuple) -> bool
    Return whether an object is an instance of a class or of a subclass thereof.
    With a type as second argument, return whether that is the object's type.
  int(x) -> int
    Convert a string or number to an integer, if possible.  A floating point argument
    will be truncated towards zero.
  float(x) -> float
    Convert a string or number to a floating point number, if possible.
  str(x) -> str
    Convert an object into a string representation.
str:
  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.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 and only if all characters in S are digits
    and there is at least one character in S.
  S.islower() --> 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.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 S converted to lowercase.
  S.replace(old, new) -> str
    Return a copy of string S with all occurrences of the string old replaced
    with the string new.
  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.startswith(prefix) -> bool
    Return True if S starts with the specified prefix and False otherwise.
  S.strip() --> str
    Return a copy of S with leading and trailing whitespace removed.
  S.upper() --> str
    Return a copy of S converted to uppercase.
```

```
list:
  append(...)
    L.append(object) -- append object to end
  count(...)
    L.count(value) -> integer -- return number of occurrences of value
  index(...)
    L.index(value, [start, [stop]]) -> integer -- return first index of value.
    Raises ValueError if the value is not present.
  insert(...)
    L.insert(index, object) -- insert object before index
  pop(...)
    L.pop([index]) -> item -- remove and return item at index (default last).
    Raises IndexError if list is empty or index is out of range.
  remove(...)
    L.remove(value) -- remove first occurrence of value.
    Raises ValueError if the value is not present.
set:
  clear(...)
    S.clear() -> None. Remove all elements from S.
  pop(...)
    S.pop() -> item. Remove and return an arbitrary set element.
    Raises KeyError if the set is empty.
  remove(...)
    S.remove(k) -> None. Remove k from S, raise KeyError if k is not a member of S
dict:
  keys(...)
    D.keys() -> a set-like object containing all of D's keys
  values(...)
    D.values() -> a set-like object containing all of D's values
  items(...)
    D.items() -> a set-like object containing all key-value pairs in D as tuples
    in the format (key, value)
  clear(...)
    D.clear() -> None. Remove all items from D
  get(...)
     D.get(k[,d]) -> returns D[k] if k is in D, otherwise returns d.  d defaults to None.
  popitem(...)
    D.popitem() -> (key, value). Remove and return the key-value pair as a tuple.
    Raise KeyError if D is empty.
object:
  __init__(...)
    x.__init__(...) initializes x; called automatically when a new object is created
  __str__(...)
    x.__str__() <==> str(x)
```