PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

AUGUST EXAMINATIONS

CSC 108H1Y
Instructor: Daniel Zingaro

Duration — three hours

Examination Aids: None.

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

Last (Family) Name(s): ______________________________

First (Given) Name(s): ______________________________

---

*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*.)

Marking Guide

# 1:	______/10
# 2:	______/ 6
# 3:	______/ 7
# 4:	______/16
# 5:	______/10
# 6:	______/ 8
# 7:	______/ 9
# 8:	______/ 7
# 9:	______/ 7
# 10:	______/ 8
TOTAL:	______/88

This examination consists of 10 questions on 20 pages (including this one).
Instructions:

- Check to make sure that you have all 20 pages.
- Read the entire exam before you start.
- Not all questions are of equal value, so budget your time accordingly.
- You do not need to add `import` lines or do error checking
- If you use any space for rough work, indicate clearly what you want marked.

Good Luck!

Question 1. [10 MARKS]

Part (a) [6 MARKS]

The left-hand column in the table below shows a series of statements **to be interpreted by the Python shell**. For each statement **that produces output**, show the expected output in the right-hand column. If a statement causes an error, simply write "error" for that statement.

Python statements	**Output**
`lst = [1,2,3]`	
`len (lst + lst)`	
`len([lst] + [lst])`	
`len([lst] + lst)`	
`len(lst.append(4))`	
`len(lst * 2) * 2`	
`len(lst)`	

Part (b) [4 MARKS]

The left-hand column in the table below shows a series of statements **to be interpreted by the Python shell**. For each statement **that produces output**, show the expected output in the right-hand column. If a statement causes an error, simply write "error" for that statement.

Python statements	**Output**
`s = 'hello'`	
`s[::]`	
`s[:-2]`	
`s[-1::-1]`	

Question 2. [6 MARKS]

Consider text files where each line contains one or more integers. For each line, the first integer is at position 0, the second is at position 1, the third is at position 2, and so on. A **column of data** is defined as all of the integers in the file with the same position. Column 0 therefore represents each integer at position 0, column 1 represents each integer at position 1, and so on. For example, in this file:

```
2 4
6
10 15 20 30
```

column 0 is:

```
2 6 10
```

column 1 is:

```
4 15
```

column 2 is:

```
22
```

and column 3 is:

```
30
```

Write a function that takes an open file object (not a filename!) referring to a file of this format, and returns a list of the **sums** of each column. For the sample file above, your function would return:

```
[18, 19, 22, 30]
```

(Write your function on the next page.)

```
def column_sums (f):
  '''Return the list of column sums from open file object f.'''
```

Question 3. [7 MARKS]

Part (a) [5 MARKS]

In this question, you are **not** allowed to use a loop. Complete the following function according to its docstring description. Read the docstring carefully!

```
def make_bricks (num_small, num_big, goal):
  '''Given num_small small bricks, each of width 1; and
  num_big big bricks, each of width 5, return True if and only if
  we can make a row of bricks of goal width by choosing
  from these bricks. For example, make_bricks (2, 2, 7)
  returns True.'''
```

Part (b) [2 MARKS]

Assume in this part of the question that small bricks have a width of 2 and big bricks have a width of 5. Given 3 small bricks and 3 big bricks, is it possible to reach a goal width of 16? Answer **only** yes or no.

Question 4. [16 MARKS]

A **segment** of a Python list `L` is a **nonempty** slice of the list `L[i:j]`. Define a **balanced segment** as a segment containing the same number of 0's and 1's. For example, in the list `[1, 0, 1, 1, 0]`, we have the following list of balanced segments:

```
[[1, 0], [0, 1], [0, 1, 1, 0], [1, 0]]
```

Notice that the same balanced segment can exist multiple times in the list: here, we have two different segments of `[1, 0]`. The **longest balanced segment** has length 4.

Part (a) [6 MARKS]

Write the following function according to its docstring.

```
def all_balanced (L):
  '''L is a list of 0's and 1's.
  Return the list of balanced segments from L.'''
```

Part (b) [3 MARKS]

Does your `all_balanced` function take time **linear** in the length of `L`? Answer Yes or No (1 mark) and give some justification for your answer (2 marks).

Part (c) [3 MARKS]

Write the following function according to its docstring. (Read the docstring carefully so you know what is being asked!) You must make a call to `all_balanced` in your solution. You can earn marks on this question even if your `all_balanced` is incorrect.

```
def len_longest_balanced_segment(L):
  '''L is a list of 0's and 1's.
  Return the length of the longest balanced segment from L;
  return 0 if L has no balanced segments.'''
```

Part (d) [4 MARKS]

Assume that we want to use Nose to test `len_longest_balanced_segment`. Describe three test cases that each test different "categories" of inputs. To describe each test case, give the list you would pass to `len_longest_balanced_segment`, the output you expect, and justification as to why your test case was chosen. Do **not** write any code.

Question 5. [10 MARKS]

Part (a) [6 MARKS]

Write the following function according to its docstring.

```
def all_keys_values (d):
  '''Return True exactly when each key in d is also a value in d,
  and each value in d is also a key in d.
  For example, all_keys_values ({2:5, 4:4, 5:2}) returns True.'''
```

Part (b) [4 MARKS]

Write a Nose test for the function `all_keys_values`. Your test should call `all_keys_values` with the dictionary `{4:5, 5:4}`. Your test should result in an informative error message should `all_keys_values` fail the test case. Do **not** `import` any modules, and do **not** write an `if __name__` block.

```
def test_all_keys_values_two_keys ():
```

Question 6. [8 MARKS]

In class, we discussed inverting a dictionary. Each key in the inverted dictionary is a value from the original dictionary, and each value in the inverted dictionary is a list of keys from the original dictionary associated with that value.

Part (a) [3 MARKS]

Give an example of a dictionary that **cannot** be inverted, according to this definition, and briefly explain why it cannot be inverted.

Part (b) [2 MARKS]

Briefly explain why we use a **list of values** for each value in the inverted dictionary, rather than a single value.

Part (c) [3 MARKS]

Given a dictionary `d`, is it guaranteed that there is only one inverted dictionary for `d`? Answer yes or no. (1 mark). If yes, justify your answer; if no, give an example of a dictionary that has more than one possible inversion. (2 marks).

Question 7. [9 MARKS]

Part (a) [3 MARKS]

Python strings (objects of type `str`) have a `join` method. Python help gives the following for this method:

```
Help on built-in function join:
join(...)
S.join(sequence) -> string
Return a string which is the concatenation of the strings in the
sequence.  The separator between elements in the sequence is S.
```

The following statement is executed at the shell. What is the output?

```
'a'.join (['b', 'c'])
```

Part (b) [6 marks]

Write the following function according to its docstring. Note that each block of characters in **s must be** followed by a space, including the final block.

```
def block_string(s, c):
  '''s is a string, c is a one-character string. Return True exactly when
  s consists entirely of increasing-length blocks of character c,
  each followed by a space. For example, block_string ('a aa aaa ', 'a')
  returns True, but block_string ('aa a aaa ', 'a') returns False.
  Each block in s must be one character longer than the previous block.'''
```

Question 8. [7 MARKS]

A **star rectangle** is a rectangle whose borders are asterisks (stars), but whose interior is empty. For example, the star rectangle of height 4 and width 8 is the following.

```
********
*      *
*      *
********
```

Write the following function according to its docstring.

```
def star_rect (f, h, w):
  '''Write a star rectangle of height h and width w to open file f.'''
```

Question 9. [7 MARKS]

Here is the text of two classes, `Animal` and `Dog`. Assume these classes are defined for each subquestion below.

```
class Animal(object):

  def __init__ (self, name):
    '''Create a new animal named name.'''

    self.name = name

  def __str__ (self):
    '''Return a string representation of this animal.'''

    return 'Animal: (' + self.name + ')'

class Dog(Animal):

  def speak(self):
    '''Make a doggie noise.'''

    print 'Woof woof!'
```

Part (a) [3 MARKS]

What is the result of running the following code? If it causes an error, simply state "error"; otherwise, give the code's output.

```
d = Dog ('Sparky')
print d
```

Part (b) [2 MARKS]

What is the output of the following code?

```
a = Animal ('Sparky')
b = Animal ('Sparky')
print a == b
```

Part (c) [2 MARKS]

Briefly explain a quick test we can use to determine whether inheriting one class from another makes sense.

Question 10. [8 marks]

Part (a) [2 marks]

Python has a `sorted` function that takes a sequence `s` and returns a new list containing the elements of `s` in sorted order. Assume `lst` is a variable referencing a list. Write code that uses `sorted` to make `lst` refer to a sorted version of that list.

Part (b) [3 marks]

The following are the contents of a list after performing **three** iterations of a non-descending insertion sort.

```
[2, 4, 6, 9, 8, 13, 12, 19]
```

What are the contents of the list after one further iteration of insertion sort?

Part (c) [3 marks]

The following are the contents of a list after performing **three** iterations of a non-descending selection sort.

```
[2, 4, 6, 9, 8, 13, 12, 19]
```

What are the contents of the list after one further iteration of selection sort?

[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.]*