CSC 108H5 F 2016 Midterm Test
Duration — 50 minutes
Aids allowed: none

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

Last Name: ____________________ **First Name:** ____________________

Do ***not*** *turn this page until you have received the signal to start.*
(Please fill out the identification section above and read the instructions below.)
Good Luck!

This midterm consists of 4 questions on 8 pages (including this one). *When you receive the signal to start, please make sure that your copy is complete.*

- Comments are not required except where indicated, although they may help us mark your answers.
- No error checking is required: assume all user input and all argument values are valid.
- If you use any space for rough work, indicate clearly what you want marked.
- You may use a pencil; however, work written in pencil will not be considered for remarking.

# 1:	______/ 4
# 2:	______/ 8
# 3:	______/ 5
# 4:	______/ 3
TOTAL:	______/20

Question 1. [4 marks]

For each of the following code fragments, draw a line through any code that is *not* executed.

Part (a) [2 marks]

```
x = 3

y = 4

cond1 = (x > 3 and y == 4) or x <= 4

cond2 = x > 3 and (y == 4 or x <= 4)

cond3 = (x > 3 or y == 3) and x <= 4
```

Part (b) [2 marks]

```
age = 16

licensed = True

if licensed:

    if age < 16:

        print("Illegal driver?")

    else:

        print("Buckle up!")

else:

    print("You're not driving")

if age < 16 and licensed:

    print("Illegal driver?")

elif not licensed:

    print("You're not driving")

else:

    print("Buckle up!")
```

Question 2. [8 marks]

Assume each of the pieces of code below is being entered into the Python console. Each subquestion is independent of the others.

In each space, write what would be displayed on the console, if anything. If a line of code would not cause anything to be displayed, leave the space blank. If the code would cause an error, write ERROR, provide a brief explanation, and leave any later spaces in that subquestion blank.

Part (a) [1 mark]

```
>>> s1 = "abc"
>>> s2 = s1
>>> s2 = "xyz"
>>> s1



>>> s2
```

Part (b) [2 marks]

```
>>> s1 = "abc"
>>> s2 = s1
>>> s1.replace('c', 'z')



>>> s1



>>> s2[0] = "x"



>>> s1



>>> s2
```

Part (c) [2 marks]

```
>>> L1 = ['a', 'b', 'c']
>>> L2 = L1
>>> L1.append(4)


>>> L1


>>> L2[0] = 1


>>> L1


>>> L2
```

Part (d) [2 marks]

```
>>> s = ''
>>> for ch in 'CSC108 at UTM':
...     if ch.isupper():
...         s = s + ch
...         print(s)
```

Part (e) [1 mark]

```
>>> s = 'CSC108'
>>> index = len(s)
>>> while index > 0:
...     index = index - 1
...     print(s[index])
```

Question 3. [5 marks]

This question uses the terminology from assignment 1. Recall that a *course record* is a string composed of a three character course code, a two digit course mark, and a two digit exam mark separated by commas.

Recall also that a student's final mark is the average of their course and exam marks unless they have an 'NE' exam mark. If a student's exam mark is 'NE', then their final mark is their course mark if they are a special case. If they are not, then the 'NE' is treated as a 0.

For the purposes of this question, a student has *failed* a course if their final mark is < 50.

```
def has_failed(course_record, is_special_case):
    '''(str, bool) -> bool
    Return True if and only if the course_record has a failing final
    mark, given whether the corresponding student is_special_case.
```

Part (a) [2 marks]

Complete the docstring above by providing four examples that represent distinct cases of this function in use. To be clear: the first argument is a course record and the second represents whether or not the student is a special case.

Part (b) [3 marks]

Implement the *has_failed* function without using any *str* methods.

Question 4. [3 marks]

A “binary string” is a string composed only of the binary digits 0 and 1. Each character in the string represents a single “bit.”

This question asks you to write a function that implements a binary operation – bitwise exclusive or (XOR) – on two binary strings. For two binary strings, the bitwise XOR is computed by performing XOR on each pair of bits that have the same index. The XOR of two bits is defined as follows: XOR(0, 0) = 0, XOR(0, 1) = 1, XOR(1, 0) = 1, XOR(1, 1) = 0.

You may assume that the two string arguments will have the same length.

```
def string_xor(s1, s2):
    '''(str, str) -> str
    Return the bitwise XOR of the two binary strings s1 and s2.
    Assumption: len(s1) == len(s2)

    >>> string_xor("", "")
    ''
    >>> string_xor("0", "0")
    '0'
    >>> string_xor("1", "0")
    '1'
    >>> string_xor("1011", "0010")
    '1001'
    '''
```

Short Python function/method descriptions:

```
__builtins__:
  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 list, tuple, or string x.
  print(value) -> NoneType
    Prints the values.
  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 sequence starts at 0.  If step is not specified, the values are incremented by 1.
  str(x) -> str
    Return an object converted to its string representation, if possible.
str:
  x in s -> bool
    Return True if and only if x is in s.
  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 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 S converted to lowercase.
  S.replace(old, new) -> str
    Return a copy of 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; use string sep as the separator and
    any whitespace string if sep is not specified.
  S.upper() -> str
    Return a copy of S converted to uppercase.
  S.startswith(sub) -> bool
    Return True if and only if S starts with the substring sub.
list:
  x in L --> bool
    Return True if and only if x is in L
  L.append(object) -> NoneType
    Append object to end of L.
```