CSCA08 Fall 2014 Midterm Exam
Duration — 100 minutes
Aids allowed: none

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

Instructor: Brian Harrington

Last Name: ______________________

First Name: ______________________

UtorID (Markus Login): ______________________

Please place a checkmark (✓) beside your tutorial session

Tutorial Number	Date/Time	Room	TA Name	Check
TUT0001	WE 16-17	IC328	Roleen Nunes	
TUT0002	WE 19-20	IC320	Ekin Ozcelik	
TUT0003	MO 10-11	BV361	Kiwi Ganeshamoorthy	
TUT0004	TU 09-10	BV361	Anastasios Exacoustos	
TUT0005	TH 17-18	BV361	Kenneth Ma	
TUT0006	TH 18-19	BV361	William Zhou	
TUT0007	TH 19-20	BV361	Minty Zhang	
TUT0008	TU 14-15	BV361	Denning Campbell	
TUT0009	FR 09-10	HW308	Cinny Cao	
TUT0010	FR 10-11	HW308	Haozhang Li	
TUT0011	FR 13-14	BV260	Umair Idris	
TUT0012	FR 14-15	BV260	Harmen Kahlon	
TUT0013	TU 09-10	IC120	Faisal Usmani	
TUT0014	TU 11-12	IC120	David Xing	
TUT0015	TH 12-13	IC320	Eric Wang	
TUT0016	TU 10-11	AA206	Ray Chu	
TUT0017	TH 18-19	AA205	Kenneth Ma	
TUT0018	FR 09-10	BV355	Ben Cooper	
TUT0019	FR 10-11	BV355	Pat McGee	
TUT0020	FR 10-11	IC204	Nick Dujay	

*Do **not** turn this page until you have received the signal to start.*

This exam consists of 5 questions on 14 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. Please read all questions thoroughly before starting on any work.
We have provided you with grids for your answers, this is simply to help you show the indentation of your code and you are not required to adhere to the grids in any specific way.

1: _____/ 5

# 2: _____/15

# 3: _____/10

# 4: _____/10

# 5: _____/10

TOTAL: _____/50

[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:

```
my_list = ['A', 'B', 'C', 'D', 'E']
count = 0
while(count <5):
    res = ''
    for i in range(0,5):
        if(i >= count):
            res += my_list[count]
        else:
            res += '+'
    count += 1
    print(res)
```

[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. [15 MARKS]

Write the output of the following code in the space provided.

```
def f1(L):
    print(L[0])
    return L[-1]

def f2(L):
    L.insert(0, L[0] + L[1])
    L[1] = 2 * f1(L)
    print(L.pop())
    return (L[-1] + L[-2])

my_list = [2, 5]
print("STEP 1")
print(f1(my_list))
print(my_list)

my_list = [2, 5]
print("STEP 2")
print(f2(my_list))
print(my_list)

my_list = ["one", "two", "three", "four"]
print("STEP 3")
print(f1(my_list))
print(my_list)

my_list = ["one", "two", "three", "four"]
print("STEP 4")
print(f2(my_list))
print(my_list)
```

```
# create an empty list to use as a result
# loop through every element in the input list
    # loop through each character in the string
        # 2 cases to deal with here:
            # case 1: the result list has a string at the correct index,
            #         just add this character to the end of that string
            # case 2: the result list doesn't have enough elements,
             #        need to create a new element to store this character
```

Question 3. [10 marks]

Our good friend Nick started writing a function. Your job is to complete it. Fortunately for you, Nick left his comments on the previous page. Complete Nick's code in the space provided.

```
def transpose(strlist):
    ''' (list of str) -> list of str

    Return a list of m strings, where m is the length of a longest string
    in strlist, if strlist is not empty, and the i-th string returned
    consists of the i-th symbol from each string in strlist, but only from
    strings that have an i-th symbol, in the order corresponding to the
    order of the strings in strlist.
    Return [] if strlist contains no nonempty strings.

    >>> transpose([])
    >>> []
    >>> transpose([''])
    >>> []
    >>> transpose(['transpose', '', 'list', 'of', 'strings'])
    >>> ['tlos', 'rift', 'asr', 'nti', 'sn', 'pg', 'os', 's', 'e']
    '''
```

Mangled Code

```
def is_close(s,t):
def is_rotation(s,t):
found_rotation = False
found_rotation == False
found_rotation = True
found_rotation == True
i = 0
i = 1
i = i + 1
i = i + 1
if(s[i] != t[i]):
if(s[i] == t[i]):
if(t == s[i:] + s[:i]):
if(t != s[i:] + s[:i]):
num_mismatches = 0
num_mismatches = 1
num_mismatches = num_mismatches - 1
num_mismatches = num_mismatches + 1
return (num_mismatches <= 1)
return (num_mismatches == 1)
return found_rotation
while (i < len(s) and found_rotation):
while (i < len(s) and not found_rotation):
while (i < len(s) and num_mismatches < 2):
while (i < len(s) or found_rotation):
while (i < len(s) or not found_rotation):
while (i < len(s) or num_mismatches < 2):
```

Question 4. [10 marks]

Last night, Nick wrote a function called `is_close` that takes two strings of equal length and returns True iff they differ in exactly one character. So if the first string is `ABCD`, then `AXCD`, or `XBCD` are close, but `ABCD`, `XBCX` and `abcd` aren't.

Unfortunately, while Nick slept, the **CODE MANGLER** struck. Deleting all his comments, removing all indentation and re-arranging the lines of code. And to make matters worse, there seems to be a bunch of code from another program shuffled in with this code. The mangled code is on the previous page. Help Nick by re-creating the `is_close` function (complete with internal & external commenting) in the space below.

Mangled Code

```
def is_close(s,t):
def is_rotation(s,t):
found_rotation = False
found_rotation == False
found_rotation = True
found_rotation == True
i = 0
i = 1
i = i + 1
i = i + 1
if(s[i] != t[i]):
if(s[i] == t[i]):
if(t == s[i:] + s[:i]):
if(t != s[i:] + s[:i]):
num_mismatches = 0
num_mismatches = 1
num_mismatches = num_mismatches - 1
num_mismatches = num_mismatches + 1
return (num_mismatches <= 1)
return (num_mismatches == 1)
return found_rotation
while (i < len(s) and found_rotation):
while (i < len(s) and not found_rotation):
while (i < len(s) and num_mismatches < 2):
while (i < len(s) or found_rotation):
while (i < len(s) or not found_rotation):
while (i < len(s) or num_mismatches < 2):
```

Question 5. [10 MARKS]

Nick just remembered that he had another function in that file (that must be where some of that other code came from). The function was called `is_rotation`, and tested whether one string was a rotation of another. If the first string is `ABCD`, then valid rotations are: `BCDA`, `CDAB` and `DABC`. The code mangler didn't delete any code, but he did add some extra lines. Help Nick by re-creating his original `is_rotation` function below.

[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.
  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.
```