CSC 108H1 S 2018 Midterm Test
Duration — 50 minutes
Aids allowed: none

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

UTORid: |_|_|_|_|_|_|_|_|

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

Lecture Section: L5101 (W6-9)
Instructor: Paul Gries

---

*Do **not** turn this page until you have received the signal to start.*
(Please fill out the identification section above, **write your name on the back of the test**, and read the instructions below.)
Good Luck!

This midterm is double-sided, and consists of 6 questions and a list of function/method descriptions. *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.
- **Do not remove any pages from the exam booklet.**

1: _____/ 5
2: _____/ 4
3: _____/ 4
4: _____/ 3
5: _____/ 5
6: _____/ 3

TOTAL: _____/24

Question 1. [5 MARKS]

For each code fragment in the table below, select the answer that best describes the printed output, or the error that occurs when the code is run.

```
diff = float(2) - int(str(2))
print(diff != 0)
```

(A) `True`
(B) `False`
(C) an error occurs converting one of the arguments

```
phrase = ['All', 'the', 'chocolate']
phrase.insert(3, 'now') + ['please']
print(phrase)
```

(A) `['All', 'the', 'chocolate']`
(B) `['All', 'the', 'chocolate', now']`
(C) `['All', 'the', 'chocolate', 'please']`
(D) `['All', 'the', 'chocolate', 'now', 'please']`
(E) some other list is printed
(F) an error occurs because + is not defined for lists
(G) an error occurs because + is not defined for non-list and list

---

```
print('CAT!'.isupper() or 1 / 0 == 5)
```

(A) `True`
(B) `False`
(C) a `ZeroDivisionError` occurs
(D) another error occurs

```
L = ['for', 'this', 'was', 'on',
    'seynt', 'Volantynys', 'day']
chaucer = ''
for i in range(len(L) // 3):
    chaucer = chaucer + L[2 * i][i]
print(chaucer)
```

(A) `fa`
(B) `fi`
(C) `or`
(D) `fay`
(E) `ftw`
(F) `for`
(G) `ftwosVd`
(H) some other characters are printed
(I) an error occurs inside the loop body

```
s = 'Je suis desja d\'amour tanne'
print(s[s.find('j') : s.find('j') + 6])
```

(A) `Je sui`
(B) `Je suis desja d'a`
(C) `jad'am`
(D) `ja d'`
(E) `ja d'a`
(F) `ja d'am`
(G) some other characters are printed
(H) an error occurs during the assignment statement

Question 2. [4 marks]

Part (a) [2 marks]

You need to write a function that returns the number of characters that two strings have in common. Both strings are parameters to the function. Do Step 1 of the Function Design Recipe: write two example function calls and their expected results. As always, choose a good name for your function.

You do not need to write any other steps of the Function Design Recipe.

```
"""
```

```
"""
```

Part (b) [2 marks]

You've decided you want to write a function that capitalizes the first number of characters in a string, where the string is made up only of alphabetic characters. Below are some example calls such as you might produce during Step 1 of the Function Design Recipe. Fill in the function header, including the type contract. As always, select good parameter names.

You do not need to write a description. You do not need to write the function body.

```
"""
>>> capitalize_first('abc', 2)
'ABc'
>>> capitalize_first('aa', 0)
'aa'
"""

# DO NOT WRITE THE FUNCTION BODY
```

Question 3. [4 marks]

Complete the following function according to its docstring.

```
def bagel_order(bagel_type: str, cream_cheese: str, toasted: bool) -> str:
    """Return the bagel order with the given bagel_type, cream_cheese, and
    whether or not it is toasted.

    If toasted is False, the format is as follows:
    <bagel_type> bagel with <cream_cheese> cream cheese

    If toasted is True, the format is as follows:
    <bagel_type> bagel toasted with <cream_cheese> cream cheese

    If cream_cheese is '', then use 'regular'.
    If cream_cheese is 'no', then omit the last part of the string.

    There should exactly one space between each word in the order, and no
    extra leading or trailing spaces.

    >>> bagel_order('poppy seed', '', True)
    'poppy seed bagel toasted with regular cream cheese'
    >>> bagel_order('everything', 'light', False)
    'everything bagel with light cream cheese'
    >>> bagel_order('plain', 'no', True)
    'plain bagel toasted'
    """
```

Question 4. [3 MARKS]

Fill in the box with the while loop condition required for the function to work as described in its docstring.

```
def find_digit(word: str) -> int:
    """Return the index of the first digit character in word, or the length of word
    if it does not contain any digit characters.

    >>> find_digit('!Ba4262')
    3
    >>> find_digit('123Hello')
    0
    >>> find_digit('cats!')
    5
    """
    i = 0
    while                                                              :
        i = i + 1
    return i
```

Question 5. [5 MARKS]

Complete the function body below according to its docstring. Hint: consider using `range` on your answer.

```
def has_pair(s: str) -> bool:
    """Return True if and only if s has 2 consecutive characters
    (i.e., next to each other) that are the same, and False otherwise.

    >>> has_pair('programming!')
    True
    >>> has_pair('Llama')
    False
    """
```

Question 6. [3 marks]

Complete the following function according to its docstring, **without using the method `str.count`**.

```
def count_alphanumeric(s: str) -> float:
    """Return the percentage of characters in s that are alphanumeric
    (letters or digits).
    The percentage should be between 0.0 and 1.0.

    Precondition: len(s) >= 1

    >>> count_alphanumeric('csc108')
    1.0
    >>> count_alphanumeric('I love 108')
    0.8
    >>> count_alphanumeric('!!!!')
    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.]

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

Short Python function/method descriptions:

```
__builtins__:
  float(x) -> float
    Convert a string or number to a floating point number, if possible.
  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
    Print the value.
  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
    Produce True if and only if x is in string 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.isalnum() -> bool
    Return True if and only if all characters in S are alphanumeric
    and there is at least one character is 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 the S converted to lowercase.
  S.upper() -> str
    Return a copy of S converted to uppercase.
list:
  x in L -> bool
    Produce True if and only if x is in list L
  L.append(object) -> NoneType
    Append object to end of 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.insert(index, object) -> NoneType
    Insert object into list L at index.
```