CSC 108H5 S 2019 Term Test 1
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 5 questions on 6 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: _____/ 2
3: _____/ 4
4: _____/ 5
5: _____/ 5

TOTAL: _____/20

Question 1. [4 MARKS]

For fhe following code fragment, draw a line through any code that is *not* executed. (P.S. Remember what we learned about short circuiting in lecture.)

```
weather = "rainy"

has_umbrella = True

if has_umbrella:

    if weather != "rainy" and has_umbrella:

        print("Leave the umbrella at home.")

    else:

        print("Don't forget your umbrella!")

else:

    print("Buy an umbrella.")

if weather != "sunny" and has_umbrella:

    print("Good preparation.")

elif weather == "rainy" and has_umbrella:

    print("Lucky you!")

print("Have a good day!")
```

Question 2. [2 MARKS]

For this question, you are being marked not only on correctness but also on simplicity. For example, you will not get full marks if you use `== True` or `== False`.

Suppose that `x`, `y`, and `z` are variable names that already refer to Boolean values. Write a logical expression (like you did for mini-exercise 1) that evaluates to `True` iff **at least one** of the three variables (`x`, `y`, `z`) is False.

Question 3. [4 MARKS]

Each of the functions below take in a string and a number and does something with them. For each function, write a good docstring description explaining what it does. You do NOT need to write any examples. Just the description (not a line-by-line description of the code, but a docstring-style general description) is needed. Also, complete the type contract by filling in the return type.

Part (a) [2 MARKS]

```
def mystery(s, n):
    """
    (str, int) -> ________________
```

```
    """

    i = 0
    while i < len(s):
        if not s[i] == '*':
            return False
        i += n

    return True
```

Part (b) [2 MARKS]

```
def mystery(s, n):
    """
    (str, int) -> ________________
```

```
    """

    i = 0
    c = 0
    while i < len(s):
        if s[i].isdigit():
            c += 1
        i = i + 1
    return c == n
```

Question 4. [5 MARKS]

Consider the following function.

```
def contains_aa(s: str) -> bool:
    """Return True if and only if s contains the substring 'aa'
    (i.e., 2 letter ''s next to each other), and False otherwise.
    """

    for i in range(len(s)-1):
        if (s[i] == "a") and (s[i+1] == "a"):
           return True
        else:
           return False
```

Part (a) [2 MARKS]

There are some values of s on which this function will return the wrong output. Would any of the following strings lead to an **incorrect** result (not matching the docstring description of the function)? Clearly circle Yes or No beside each option.

(a) `'aaa'` This returns **wrong** output This returns **correct** output

(b) `'aaab'` This returns **wrong** output This returns **correct** output

(c) `'baaa'` This returns **wrong** output This returns **correct** output

(d) `'bcaaacb'` This returns **wrong** output This returns **correct** output

(e) `'a'` This returns **wrong** output This returns **correct** output

(f) `'baba'` This returns **wrong** output This returns **correct** output

(g) `'bcc'` This returns **wrong** output This returns **correct** output

(h) `''` This returns **wrong** output This returns **correct** output

Part (b) [1 MARK]

Briefly explain what mistake in the code causes the wrong results to occur.

Part (c) [2 MARKS]

Annotate (make edits by crossing out and replacing things) in the body of the function above to fix this mistake from happening. Your changes should be as **minimal** as you can make them. That is, do NOT rewrite the whole code; that will not get you the marks for this part.

Question 5. [5 MARKS]

Complete the following function according to its docstring. You may use either a while loop or for loop for this question; choose whichever you prefer.

```
def upper_every_n(s: str, n: int) -> bool:
    """Return True if and only if s contains an uppercase letter at
    every n_th index in the string, starting with the first letter.

    >>> upper_every_n('QbFkAmLp', 2)
    True
    >>> upper_every_n('bBkAmLp', 2)
    False
    >>> upper_every_n('GxpxEjklR', 4)
    True
    """
```

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.
  input([prompt]) -> str
    Read a string from the user. Provide the prompt to the user, if given.
  abs(x) -> number
    Return the absolute value of x.
  chr(i) -> str
    Return the string character associated with the given ASCII integer.
  ord(c) -> int
    Return the integer ASCII code associated with the one-character string.
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.
```