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

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

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

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

Lecture Section: **(circle one)**: L0101 (MWF1) L0301 (MWF2)
Instructor: Jacqueline Smith 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.

```
days = ['Mon', 'Tues']
days.append('Wed') + ['Thurs', 'Fri']
print(days)
```

(A) `['Mon', 'Tues']`
(B) `['Mon', 'Tues', 'Wed']`
(C) `['Mon', 'Tues', 'Thurs', 'Fri']`
(D) `['Mon', 'Tues', 'Wed', 'Thurs', 'Fri']`
(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

---

```
s = 'Valentine\'s' + 'Day'
print(s[8:13])
```

(A) `esDay`
(B) `es Da`
(C) `es Day`
(D) `e'sDa`
(E) `e's D`
(F) `e's Da`
(G) some other characters are printed
(H) an error occurs during the assignment statement

```
L = ['roses', 'red', 'violets',
    'blue', 'sugar', 'sweet']
s = ''
for i in range(len(L) // 3):
    s = s + L[i * 3][1]
print(s)
```

(A) `rb`
(B) `ol`
(C) `rrv`
(D) `oei`
(E) `rrvbss`
(F) `oeiluw`
(G) some other characters are printed
(H) an error occurs inside the loop body

```
print(int(2.999) > int('2'))
```

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

```
print('CAT!'.find('!') == 3 or 7 / 0 > 0)
```

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

Question 2. [4 marks]

Part (a) [2 marks]

You need to write a function that returns whether there is an odd digit in a string at a particular index. Both the string and the index 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.

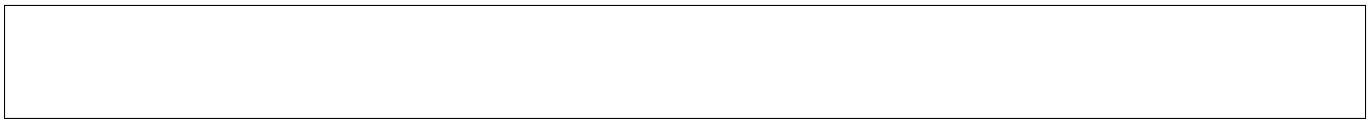
```
"""
```

```
"""
```

Part (b) [2 marks]

You've decided you want to write a function that returns whether a string contains only characters that appear in another string. 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.



```
"""
>>> contains_chars('cat', 'ewoks are cuter')
True
>>> contains_chars('abracadabra', 'abcdr')
True
>>> contains_chars('a', 'bcd')
False
"""

# DO NOT WRITE THE FUNCTION BODY
```

Question 3. [4 marks]

Complete the following function according to its docstring.

```
def latte_order(shots: int, milk: str, flavour: str) -> str:
    """Return the latte order with the given espresso shots, milk, and flavour,
    formatted as follows:
    <shots> shot <flavour> latte with <milk> milk

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

    If milk is 'A', replace it with 'almond', and if milk is 'S', replace it
    with 'soy'.

    Precondition: milk is one of 'A', 'S', '2%', '1%'

    >>> latte_order(2, 'A', 'vanilla')
    '2 shot vanilla latte with almond milk'
    >>> latte_order(1, '2%', 'pumpkin spice')
    '1 shot pumpkin spice latte with 2% milk'
    >>> latte_order(3, 'S', '')
    '3 shot latte with soy milk'
    """
```

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

    >>> find_lowercase('cats')
    0
    >>> find_lowercase('123Hello')
    4
    >>> find_lowercase('CAT!')
    4
    """
    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_3_consecutive_digits(s: str) -> bool:
    """Return True if and only if s has 3 digits that appear consecutively
    (i.e., next to each other) in s, and False otherwise.

    >>> has_3_consecutive_digits('abc123')
    True
    >>> has_3_consecutive_digits('a1b2c3')
    False
    """
```

Question 6. [3 marks]

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

```
def get_percent(s: str, ch: str) -> float:
    """Return the percentage of characters in s that are equal to ch.
    The percentage should be between 0.0 and 1.0.

    Precondition: len(s) >= 1 and len(ch) == 1

    >>> get_percent('hello', 'l')
    0.4
    >>> get_percent('Testing 123', 'z')
    0.0
    >>> get_percent('Valentines', 'a')
    0.1
    """
```

[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.
```