# Midterm Test

CSC108H1F / LEC0101/0102

October 16 2019, 10:10am — Duration: **75 minutes**

Aids Allowed: **None**

**First (Given) Name(s):**

**Last (Family) Name(s):**

**10-Digit Student Number:**

**UTORid (e.g., `pitfra12`):**

---

*Do **not** turn this page until you have received the signal to start.*
In the meantime, write your name, student number, and UTORid above
(please do this now!) and read the instructions below *carefully*.

---

- This term test consists of 6 questions on 9 pages (including this one), printed on both sides of the paper. *When you receive the signal to start, please make sure that your copy of the test is complete.*
- Answer each question directly on the test paper, in the space provided, and use a "blank" page for rough work. If you need more space for one of your solutions, use one of the "blank" pages and *indicate clearly the part of your work that should be marked.*
- 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.
- Do not remove any pages from the exam booklet.

*Good Luck!*

## Question 1. [6 marks]

This question has six parts. For each question on the left-hand side, circle the letter(s) in front of the corresponding answer(s) on the right-hand side.

Circle the best docstring description for the function below according to the Function Design Recipe:

```
def double(num: int) -> int:
    return num * 2
```

(A) Doubles the value of num.
(B) Doubles the number.
(C) Return the number multiplied by two.
(D) Return num doubled.

---

Circle the answer that best describes what is printed when the following code is run:

```
s = ‘Oct’
s = s + ‘31’
print(type(s[4]))
```

(A) `<class ‘list’>`
(B) `<class ‘int’>`
(C) `<class ‘str’>`
(D) `<class ‘float’>`
(E) `<class ‘NoneType’>`
(F) Nothing is printed because an error occurs
(G) No error occurs, and something else is printed

---

Circle all of the code snippets that would print `rat` if

```
s = ‘stars’
```

(A) `print(s[1:3])`
(B) `print(s[-2] + s[2] + s[1])`
(C) `print(s[3:0:-1])`
(D) `print(s[-2:-5:1])`
(E) `print(s[::-1])`
(F) None of the above will print `rat`

---

Circle all of the answers that correctly describe what is printed when the following code is run:

```
i = 4

while i < 20:
    print(i)
    i = i + 5
```

(A) The first number printed is 4
(B) The first number printed is 9
(C) There are four numbers printed in total
(D) There are sixteen numbers printed in total
(E) The number 15 is printed
(F) None of the above correctly describe what is printed

---

Circle all of the expressions that would evaluate to `True`:

(A) `‘o’ in [‘Welcome’, ‘to’, ‘CSC108’]`
(B) `len(‘what\’s’) in [2, 4, 6, 8]`
(C) `[1, 2, 3, 4, 5] in len(‘hello’)`
(D) `‘g’ in (‘abc’ + ‘defg’)`
(E) `‘aabbcc’ == ‘abc’ * 2`
(F) None of the expressions evaluate to `True`

---

Circle all of the code snippets that would result in variable `f` referring to the `int` value 12 and variable `g` referring to the `int` value 2.

(A)
```
f = 24
g = 9
g = f // g
f = f // g
```

(B)
```
f = 24
g = 12
g = f / g
f = f / g
```

(C)
```
f = 12
g = 3
if g >= f:
    f = g
if g < f:
    g = g - 1
```

(D)
```
f = 12 // 2 + 2
g = 16
g = g // f
```

## Question 2. [3 marks]

### Part (a) [2 marks]

Consider the problem of writing function `are_consecutive_nums` that has the following docstring description:

```
"""Return True if and only if i, j and k are consecutive numbers in ascending order.

>>> are_consecutive_nums(4, 5, 6)
True
>>> are_consecutive_nums(3, 2, 1)
False
>>> are_consecutive_nums(3, 3, 4)
False
"""
```

Several solution attempts are given below. Some are correct and some are incorrect.
**Circle the letter in front of each solution attempt that *correctly* implements the function.**

```
(A) def are_consecutive_nums(i: int, j: int, k: int) -> bool:
        if i + 1 == j:
            if j + 1 == k:
                return True
        return False

(B) def are_consecutive_nums(i: int, j: int, k: int) -> bool:
        if i == j - 1:
            return True
        elif j == k - 1:
            return True
        else:
            return False

(C) def are_consecutive_nums(i: int, j: int, k: int) -> bool:
        if i + 1 == j or j + 1 == k:
            return True
        else:
            return False

(D) def are_consecutive_nums(i: int, j: int, k: int) -> bool:
        if i + 1 != j:
            return False
        elif j + 1 != k:
            return False
        else:
            return True
```

### Part (b) [1 mark]

Fill in the box below to correctly implement function `are_consecutive_nums` using a single *return* statement.

```
def are_consecutive_nums(i: int, j: int, k: int) -> bool:

    return
```

**Question 3.** [4 marks]

Complete the following function according to its docstring. You must use the constant `VOWELS` in your solution.

```
VOWELS = ‘aeiou’

def get_lowercase_vowels(s: str) -> str:
    """Return a new string with all characters in s that are also in VOWELS, or
    an empty string if there are no lowercase vowels in s.

    >>> get_lowercase_vowels(‘aardvark’)
    ‘aaa’
    >>> get_lowercase_vowels(‘TWO camels!’)
    ‘ae’
    >>> get_lowercase_vowels(‘123xyzABC’)
    ‘’
    """
```

## Question 4. [4 MARKS]

Fill in the boxes below to correctly complete the body of function `get_sum_matches` according to its docstring. Do not change the code outside of the boxes.

```
def get_sum_matches(lst: List[int]) -> List[int]:
    """Return a new list that contains each item in lst that is equal to the sum
    of the items on either side of it in lst.  The first and last items of lst
    must not be included because they do not have items on both sides.

    Precondition: len(lst) >= 3

    >>> get_sum_matches([5, 6, 1, -5])
    [6, 1]
    >>> get_sum_matches([5, 11, 6, -5, 2, 6])
    [11, 6]
    >>> get_sum_matches([5, 6, 6, 5])
    []
    """

    matches = [                    ]

    for i in range([                                                  ]):

        if [                                                         ]:
            # Append the current item to the matches list.
            [                                              ]

    return matches
```

Space for rough work (this will not be marked):

## Question 5. [4 marks]

### Part (a) [2 marks]

Fill in the box with the while loop condition required for the function to work as described in the docstring. Do not change the code outside of the box.

```
def find_index_of_last_digit(s: str) -> int:
    """Return the index of the last occurrence of a digit in s or -1 if s contains no digits.

    >>> find_index_of_last_digit('csc108')
    5
    >>> find_index_of_last_digit('801csc')
    2
    >>> find_index_of_last_digit('Comp Sci')
    -1
    """
    i = len(s) - 1

    while [                                        ] :
        i = i - 1
    return i
```

### Part (b) [2 marks]

Fill in the two boxes below to correctly implement the following function. Do not change the code outside of the boxes. **Your code must call `find_index_of_last_digit` from Part (a).**

```
def find_index_of_first_digit(s: str) -> int:
    """Return the index of the first occurrence of a digit in s or len(s) if s contains
    no digits.

    >>> find_index_of_first_digit('csc108')
    3
    >>> find_index_of_first_digit('801csc')
    0
    >>> find_index_of_first_digit('Comp Sci')
    8
    """

    reversed_s = [                  ]

    return [                                             ]
```

## Question 6. [4 marks]

Complete the following function according to its docstring. You must use the constant `PUNCTUATION` in your solution.

```
PUNCTUATION = ‘?!.,’

def is_punctuated_tweet_word(s: str) -> bool:
    """Return True if and only if s is a valid tweet word followed by
    exactly one character in PUNCTUATION. A valid tweet word contains only
    alphanumeric characters and underscores. Also, a valid tweet word contains
    at least one alphanumeric character.

    Precondition: len(s) >= 1

    >>> is_punctuated_tweet_word("hello_world!")
    True
    >>> is_punctuated_tweet_word("WeTheNorth!")
    True
    >>> is_punctuated_tweet_word("how?are?you?")
    False
    >>> is_punctuated_tweet_word("Yes!!!")
    False
    """
```

*Use the space on this "blank" page for scratch work, or for any solution that did not fit elsewhere.*
***Clearly label each such solution with the appropriate question and part number.***