PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

APRIL 2018 EXAMINATIONS

CSC 108 H1S
Instructor(s): Gries, Morris, and Smith

Duration—3 hours

No Aids Allowed

You must earn at least 28 out of 70 marks (40%) on this final examination in order to pass the course. Otherwise, your final course grade will be no higher than 47%.

Student Number: ⎣⎵⎵⎵⎵⎵⎵⎵⎵⎵⎦ **UTORid:** ⎣⎵⎵⎵⎵⎵⎵⎵⎵⎦

Family Name(s): ______________________ **First Name(s):** ______________________

Do ***not*** *turn this page until you have received the signal to start.*
In the meantime, please read the instructions below *carefully*.

This Final Examination paper consists of 11 questions on 21 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 paper is complete and fill in your name and student number above.*

- Comments and docstrings are not required except where indicated, although they may help us mark your answers.
- You do not need to put `import` statements in your answers. You may assume everything you need from `typing` has been imported.
- 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 pages or take the exam apart.

Marking Guide

# 1:	______/ 8
# 2:	______/ 6
# 3:	______/ 6
# 4:	______/ 5
# 5:	______/ 6
# 6:	______/ 6
# 7:	______/ 8
# 8:	______/ 5
# 9:	______/ 5
# 10:	______/ 6
# 11:	______/ 9
TOTAL:	______/70

Good Luck!

Question 1. [8 MARKS]

Part (a) [5 MARKS] Using the Function Design Recipe, write a function called `str_find` that has the same behaviour as the `str.find` method (but without the optional argument). Example calls are provided below. Complete the function, including header, docstring, and body.

You can find information about `str.find` on the help pages at the end of this exam paper. Do NOT use `str.find` or `str.index` in your solution. You may use other `str` methods.

```
    """

    >>> str_find('dogscats', 'cat')
    4
    >>> str_find('dogscats', 'horse')
    -1
    """
```

Part (b) [3 MARKS] Consider a similar function `str_count` that has the same behaviour as `str.count` (but without the optional arguments). The function header and description are given below.

```
def str_count(s: str, sub: str) -> int:
    """Return the number of non-overlapping occurrences of substring sub in string s.
    """
```

Add three more distinct test cases to the table below that should be used to test this function. No marks will be given for duplicated test cases.

Test Case Description	s	sub	Return Value
1 occurrence at start	`'caterpillar'`	`'cat'`	1
1 occurrence with overlap	`'aaa'`	`'aa'`	1

Question 2. [6 MARKS]

Part (a) [2 MARKS] Complete the body of function `str_capitalize` so that it has the same behaviour as method `str.capitalize`. The docstring is provided below.

You can find information about `str.capitalize` on the help pages at the end of this exam paper. Do NOT use `str.capitalize` in your solution. You may use other `str` methods.

```
def str_capitalize(name: str) -> str:
    """Return the capitalized version of name.

    Precondition: len(name) > 0 and name.isalpha()

    >>> str_capitalize('CoLiN')
    'Colin'
    >>> str_capitalize('jacqueline')
    'Jacqueline'
    >>> str_capitalize('PAUL')
    'Paul'
    """
```

Part (b) [4 MARKS]

Complete the docstring example in the box below, then write the body of the function. You do not need to write a second example. Your solution *must* use the function in Part (a) as a helper function.

```
def capitalize_all(names: List[str]) -> None:
    """Replace each name in names with its capitalized version.

    >>> names = ['CoLiN', 'jacqueline', 'PAUL']



    """
```

Question 3. [6 marks]

Each of the following functions has 1 or more bugs. Read them carefully and then complete the table at the bottom of the page.

```
def second_biggest(nums: List[int]) -> int:
  """Return the second-largest number in nums. Precondition: len(nums) >= 2
  """
  return max(nums) - 1

def all_negative(nums: List[int]) -> bool:
  """Return True if and only if every number in nums is negative.
  """
  for num in nums:
    if num >= 0:
      return False
    else:
      return True

def names_by_first_letter(name_list: List[str]) -> Dict[str, str]:
  """Return a dictionary mapping each first letter of the names in name_list to
  a list of names beginning with that letter.

  Precondition: len(name) > 0 and all names in name_list are capitalized.

  >>> names_by_first_letter(['Jen', 'Tom'])
  {'J': ['Jen'], 'T': ['Tom']}
  """
  res = {}
  for name in name_list:
    if name not in res:
      res[name[0]] = [name]
    else:
      res[name[0]].append(name)
  return res

def fluffiness(s: str) -> int:
  """Return the "fluffiness" of s. A string's fluffiness is equal to the number
  of characters it contains which appear in "fluffy".
  """
  total = 0
  for fluffy_char in "fluffy":
    if fluffy_char in s:
      total = total + 1
  return total
```

Demonstrate that these functions are buggy by filling in the table below with an input for each function where the actual return value is not the same as the expected return value (according to the description in the docstring). The first row has been filled in as an example.

Function	Input	Actual Return Value	Expected Return Value
`second_biggest`	`[1, 3]`	2	1
`all_negative`			
`names_by_first_letter`			
`fluffiness`			

Question 4. [5 MARKS]

In the function below, the parameters are parallel lists. Fill in the boxes to complete the function according to its docstring.

```
def diff_region(L1: List[int], L2: List[int]) -> Tuple[int]:
    """Return a 2-item tuple containing:
      - the index where L1 and L2 first differ, and
      - the index where L1 and L2 last differ.

    The second index should be negative.

    Precondition: len(L1) == len(L2) and len(L1) > 0 and L1 != L2

    >>> L1 = [0, 1, 6, 7, 2, 3, 4]
    >>> L2 = [0, 1, 8, 9, 2, 3, 4]
    >>> diff_region(L1, L2)
    (2, -4)
    """

    # find the index of the leftmost differing elements
    left = 0
    while $\boxed{\phantom{xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx}}$:
        $\boxed{\phantom{xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx}}$

    # find the index of the rightmost differing elements
    right = -1
    while $\boxed{\phantom{xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx}}$:
        $\boxed{\phantom{xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx}}$

    return $\boxed{\phantom{xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx}}$
```

Question 5. [6 MARKS]

Read the docstring of `set_negatives_to_zero` below carefully so you understand what the function does. Then read the comments outlining the algorithm used to solve the problem.

Complete the function by filling in each box with the letter that corresponds to the line of code that correctly implements that step of the algorithm, so that the function is correct according to its docstring.

```
def set_negatives_to_zero(L: List[List[int]]) -> None:
    """Modify the sublists of L so that all negative values are set to 0.
    If a resulting sublist contains only 0's, replace that sublist with
    an empty list.

    >>> L = [[1, -1], [0, -2, -3], [0]]
    >>> set_negatives_to_zero(L)
    >>> L
    [[1, 0], [], []]
    >>> L = [[1, 2, 3], [-1, -2, -3], [5, 1, -2, -6, 3]]
    >>> set_negatives_to_zero(L)
    >>> L
    [[1, 2, 3], [], [5, 1, 0, 0, 3]]
    """

    # iterate over L [      ]

        # iterate over the current sublist [      ]

            # if the current element is negative [      ]

                # then set it to 0 [      ]

        # if there are only 0's in the sublist [      ]

            # then replace the sublist with an empty list [      ]
```

(A) `for i in range(len(L)):`
(B) `for i in range(len(L[i])):`
(C) `for i in sublist:`
(D) `for item in sublist:`
(E) `for j in range(len(L[i])):`
(F) `for sublist in L:`
(G) `for sublist in range(L):`

(H) `if i == 0:`
(I) `if i < 0:`
(J) `if item < 0:`
(K) `if j < 0:`
(L) `if L[i] == 0:`
(M) `if L[i][j] < 0:`
(N) `if sublist == 0:`
(O) `if sum(i) == 0:`
(P) `if sum(L[i]) == 0:`
(Q) `if sum(sublist) == 0:`
(R) `if 0 in L[i]:`
(S) `if 0 in sublist:`

(T) `i = 0`
(U) `i = []`
(V) `item = 0`
(W) `j = 0`
(X) `L[i] = []`
(Y) `L[i][j] = 0`
(Z) `sublist = []`

Question 6. [6 MARKS]

Answer the questions below about the following function:

```
def is_level(L: List[int]) -> bool:
    """Return True if and only if there is no gap larger than 1 between any
    two adjacent numbers in L.

    >>> is_level([1, 3])
    False
    >>> is_level([1, 2, 3, 4, 5, 4, 5])
    True
    >>> is_level([1, 2, 3, 2, 3, 1])
    False
    """
    for i in range(len(L) - 1):
        if abs(L[i] - L[i + 1]) > 1:
            return False
    return True
```

Let k be the number of elements in `L`.

Part (a) [1 MARK] Give a formula in terms of k to describe exactly how many comparisons happen in `is_level` in the *best case*.

Part (b) [1 MARK] Briefly describe the property of `L` that causes the best case for `is_level`.

Part (c) [1 MARK] Circle the term that best describes the *best case* running time of `is_level`.

constant   linear   quadratic   something else

Part (d) [1 MARK] Give a formula in terms of k to describe exactly how many comparisons happen in `is_level` in the *worst case*.

Part (e) [1 MARK] Briefly describe the property of `L` that causes the worst case for `is_level`.

Part (f) [1 MARK] Circle the term that best describes the *worst case* running time of `is_level`.

constant   linear   quadratic   something else

Question 7. [8 MARKS]

As you may know, there is an Ontario election this year. The three major parties in the election are the Liberal party, the Progressive Conservative (PC) party, and the National Democratic (NDP) party.

Often, a party wins several elections in a row. For example, a Liberal party leader has won the last four elections: Kathleen Wynne won in 2014, and before that Dalton McGuinty won in 2011, 2007, and 2003. Before that, Mike Harris led the PC party, which won in 1999 and 1995.

An *elections file* containing this information has a series of records that look like this:

```
Party: # consecutive wins
Election Year [1 per win]
```

Here is an example of an elections file containing the Ontario election results for the past three decades:

```
Liberal: 4
2014
2011
2007
2003
PC: 2
1999
1995
NDP: 1
1990
Liberal: 2
1987
1985
```

A question one might ask is "In what years did a particular party win an election?" We might represent this as follows; we'll call this dictionary an *elections dictionary*:

```
{'Liberal': ['2014', '2011', '2007', '2003', '1987', '1985'],
 'PC': ['1999', '1995'],
 'NDP': ['1990']}
```

The function on the next page is designed to return such a dictionary. Complete the body of the function.

You can assume that the parameter refers to an open file that matches the format of an elections file as described above.

Question 7 continued...

```
def get_elections(efile: TextIO) -> Dict[str, List[str]]:
    """Return an elections dictionary based on the information in efile.
    """

    # An elections dictionary.
    wins = {}
















    return wins
```

Question 8. [5 marks]

Each of the following code snippets involves writing to a plaintext file. In the box beside each snippet of code, fill in the contents of the file after running the code. The contents of the box should match how the file would appear when opened in a text editor like Wing101.

Use ␣ to clearly indicate spaces. For example, write `Hello world!` as `Hello␣world!`

Part (a) [1 mark]

```
f = open('a.txt', 'w')
f.write('what')
f.write('time')
f.write('is it?')
f.close()
```

Part (b) [1 mark]

```
f = open('b.txt', 'w')
f.write('what\ntime\nis it?')
f.close()
```

Part (c) [1 mark]

```
f = open('c.txt', 'w')
name = 'Felix'
f.write('My dog is name.upper()')
f.close()
```

Part (d) [1 mark]

```
f = open('d.txt', 'w')
s = '\"woof\", said Felix'
f.write(s)
f.close()
```

Part (e) [1 mark]

```
f = open('e.txt', 'w')
f.write('Hello\n')
f.write('\n')
f.close()
f = open('e.txt', 'w')
f.write('Felix!')
f.close()
```

Question 9. [5 MARKS]

The version of selection sort we analyzed in the course had a sorted part in the front containing the smallest elements, and had an unsorted part after that. The algorithm in the PCRS videos and the worksheets repeatedly found the smallest element in the unsorted part and added it to the end of the sorted part.

A variation of the selection sort algorithm is to have the sorted part at the end containing the largest elements, and have the unsorted part in the front of the list. In this variation, the algorithm repeatedly finds the largest element in the unsorted part and adds it to the beginning of the sorted part.

Complete the code below to implement this variation on the selection sort algorithm.

```
def get_index_of_largest(L: list, i: int) -> int:
    """Return the index of the largest item in L[:i + 1].

    >>> get_index_of_largest([2, 7, 3, 5, 9], 3)
    1
    """

    index_of_largest = [        ]

    for j in range([                              ]):
        if L[j] [        ] L[index_of_largest]:
            index_of_largest = [        ]

    return index_of_largest


def selection_sort(L: list) -> None:
    """Sort the items of L into non-descending order.

    >>> L = [4, 2, 5, 6, 7, 3, 1]
    >>> selection_sort(L)
    >>> L
    [1, 2, 3, 4, 5, 6, 7]
    """

    # iterate backwards on the indices of L
    for i in range([                              ]):

        index_of_largest = get_index_of_largest(L, i)
        L[index_of_largest], L[i] = L[i], L[index_of_largest]
```

Question 10. [6 marks]

Write the body of the following function according to its docstring.

```
def organize_walks(walks: List[Tuple[str, str, int]]) -> Dict[str, Dict[str, int]]:
    """Return a dictionary based on the data in walks where the keys are dog names;
    the values are dictionaries where the keys are dog walker names and the values
    are the total distance that walker walked a particular dog.

    walks is a list of tuples, where each tuple has the form
    (<dog name>, <walker name>, <distance walked>)
    representing a single walk.

    >>> organize_walks([('Uli', 'Jeff', 3), ('Uli', 'Jeff', 5)])
    {'Uli': {'Jeff': 8}}
    >>> organize_walks([('Felix', 'Bob', 5), ('Fido', 'Bob', 2), ('Felix', 'Bob', 3), \
    ('Fluffy', 'Ann', 10), ('Felix', 'Ann', 1), ('Fluffy', 'Ann', 10)])
    {'Felix': {'Bob': 8, 'Ann': 1}, 'Fido': {'Bob': 2}, 'Fluffy': {'Ann': 20}}
    """
```

Question 11. [9 marks]

As you'll recall, Assignment 3 had you process airports, direct-flight routes between the airports, and flight sequences (a series of routes). This question explores an alternative representation, and adds a new feature: the number of daily trips on a route.

Here is the header and docstring for class `Route`, as well as its `__init__` and `set_daily_trips` methods.

```
class Route:
    """ Information about a route. """

    def __init__(self, start_code: str, end_code: str) -> None:
        """Initialize a new route with start start_code, end end_code, and zero daily trips.

        >>> route1 = Route('YYZ', 'LHR')
        >>> route1.start
        'YYZ'
        >>> route1.end
        'LHR'
        >>> route1.daily_trips
        0
        """
        self.start = start_code
        self.end = end_code
        self.daily_trips = 0

    def set_daily_trips(self, daily_trips: int) -> None:
        """Set the number of daily trips on this route to daily_trips.

        Precondition: daily_trips >= 0

        >>> route1 = Route('YYZ', 'LHR')
        >>> route1.daily_trips
        0
        >>> route1.set_daily_trips(3)
        >>> route1.daily_trips
        3
        """
        self.daily_trips = daily_trips
```

Use this to help you complete the questions on the following pages.

Part (a) [2 marks] Complete the body of the `__str__` method in the `Route` class.

```
    def __str__(self) -> str:
        """Return a string representation of this route.

        >>> route1 = Route('YYZ', 'LHR')
        >>> route1.set_daily_trips(3)
        >>> str(route1)
        'Route from YYZ to LHR has 3 daily trip(s).'
        """
```

Here is the header and docstring for class `FlightSequence`. In the following questions, you will complete the methods in the class `FlightSequence`, using the class `Route` in your answers.

```
class FlightSequence:
    """ Information about a FlightSequence, which contains a sequence of Routes. """
```

Part (b) [1 mark] Complete the body of this class `FlightSequence` method according to its docstring.

```
    def __init__(self, name: str) -> None:
        """Initialize a new flight sequence that is named name with an empty list
        of routes.

        >>> fs1 = FlightSequence('Trip to Paris')
        >>> fs1.name
        'Trip to Paris'
        >>> fs1.routes
        []
        """
```

Part (c) [4 marks] Complete the body of this class `FlightSequence` method according to its docstring. Use class `Route` when possible.

```
def add_route(self, start_airport: str, end_airport: str) -> bool:
    """Return True and add the new route with start airport start_airport
    and end airport end_airport to the routes list if and only if they are
    the start and end of a valid route in the current flight sequence.

    That is, if the new route is a continuation of the flight sequence (
    the last element in this flight sequence ends where the new route
    starts), or if there are currently no routes in the flight sequence,
    then add the new route and return True. Otherwise, return False.

    >>> fs1 = FlightSequence('Trip to Paris')
    >>> fs1.routes
    []
    >>> fs1.add_route('YYZ', 'LHR')
    True
    >>> str(fs1.routes[0])
    'Route from YYZ to LHR has 0 daily trip(s).'
    >>> fs1.add_route('LHR', 'CDG')
    True
    >>> fs1.add_route('YYZ', 'CDG')
    False
    >>> len(fs1.routes)
    2
    """
```

Part (d) [2 marks]

Complete the body of this class `FlightSequence` method according to its docstring.

```
def is_possible_sequence(self) -> bool:
    """Return True if and only if every route in this flight sequence
    has at least one daily trip.

    >>> fs1 = FlightSequence('Trip to Paris')
    >>> fs1.add_route('YYZ', 'LHR')
    True
    >>> fs1.is_possible_sequence()
    False
    >>> fs1.routes[0].set_daily_trips(1)
    >>> fs1.add_route('LHR', 'CDG')
    True
    >>> fs1.routes[1].set_daily_trips(3)
    >>> fs1.is_possible_sequence()
    True
    """
```

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

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

Short Python function/method descriptions:

```
__builtins__:
  input([prompt]) -> str
    Read a string from standard input. The trailing newline is stripped. The prompt string,
    if given, is printed without a trailing newline before reading.
  abs(x) -> number
    Return the absolute value of x.
  chr(i) -> Unicode character
    Return a Unicode string of one character with ordinal i; 0 <= i <= 0x10ffff.
  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 the list, tuple, dict, or string x.
  max(iterable) -> object
  max(a, b, c, ...) -> object
    With a single iterable argument, return its largest item.
    With two or more arguments, return the largest argument.
  min(iterable) -> object
  min(a, b, c, ...) -> object
      With a single iterable argument, return its smallest item.
      With two or more arguments, return the smallest argument.
  open(name[, mode]) -> file open for reading, writing, or appending
    Open a file.  Legal modes are "r" (read), "w" (write), and "a" (append).
  ord(c) -> integer
    Return the integer ordinal of a one-character string.
  print(value, ..., sep=' ', end='\n') -> NoneType
    Prints the values. Optional keyword arguments:
    sep:  string inserted between values, default a space.
    end:  string appended after the last value, default a newline.
  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 list starts at 0.  If step is not specified,
    the values are incremented by 1.

dict:
  D[k] --> object
    Produce the value associated with the key k in D.
  del D[k]
    Remove D[k] from D.
  k in d --> bool
    Produce True if k is a key in D and False otherwise.
  D.get(k) -> object
    Return D[k] if k in D, otherwise return None.
  D.keys() -> list-like-object of object
    Return the keys of D.
  D.values() -> list-like-object of object
    Return the values associated with the keys of D.
  D.items() -> list-like-object of tuple of (object, object)
    Return the (key, value) pairs of D, as 2-tuples.
```

```
file open for reading:
  F.close() -> NoneType
    Close the file.
  F.read() -> str
    Read until EOF (End Of File) is reached, and return as a string.
  F.readline() -> str
    Read and return the next line from the file, as a string. Retain any newline.
    Return an empty string at EOF (End Of File).
  F.readlines() -> list of str
    Return a list of the lines from the file.  Each string retains any newline.

file open for writing:
  F.close() -> NoneType
    Close the file.
  F.write(x) -> int
    Write the string x to file F and return the number of characters written.

list:
  x in L --> bool
    Produce True if x is in L and False otherwise.
  L.append(x) -> NoneType
    Append x to the end of the 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.index(value) -> int
    Return the lowest index of value in L.
  L.insert(index, x) -> NoneType
    Insert x at position index.
  L.pop([index]) -> object
    Remove and return item at index (default last).
  L.remove(value) -> NoneType
    Remove the first occurrence of value from L.
  L.reverse() -> NoneType
    Reverse *IN PLACE*.
  L.sort() -> NoneType
    Sort the list in ascending order *IN PLACE*.


str:
  x in s --> bool
    Produce True if and only if x appears as a substring in s.
  str(x) -> str
    Convert an object into its string representation, if possible.
  S.capitalize() -> str
    Return a capitalized version of S, i.e. make the first character
    have upper case and the rest lower case.
  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.endswith(S2) -> bool
    Return True if and only if S ends with S2.
  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.index(sub) -> int
  Like find but raises an exception 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 all characters in S are digits
  and there is at least one character in S, and False otherwise.
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.lstrip([chars]) -> str
  Return a copy of S with leading whitespace removed.
  If chars is given and not None, remove characters in chars instead.
S.replace(old, new) -> str
  Return a copy of string S with all occurrences of old replaced
  with new.
S.rstrip([chars]) -> str
  Return a copy of S with trailing whitespace removed.
  If chars is given and not None, remove characters in chars instead.
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(S2) -> bool
  Return True if and only if S starts with S2.
S.strip([chars]) -> str
  Return a copy of S with leading and trailing whitespace removed.
  If chars is given and not None, remove characters in chars instead.
S.upper() -> str
  Return a copy of S converted to uppercase.
```