PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

DECEMBER 2017 EXAMINATIONS

CSC 108 H1F
Instructor(s): Campbell, Fairgrieve, and Smith

PLEASE HAND IN

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 12 questions on 24 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 Student Number, UTORid and Name 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.
- No error checking is required: assume all function arguments have the correct type and meet any preconditions.
- 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:	_____/ 6
# 2:	_____/ 4
# 3:	_____/ 6
# 4:	_____/ 4
# 5:	_____/ 4
# 6:	_____/ 6
# 7:	_____/ 6
# 8:	_____/ 3
# 9:	_____/ 5
# 10:	_____/ 6
# 11:	_____/12
# 12:	_____/ 8
TOTAL:	_____/70

Question 1. [6 MARKS]

Each of the following sets of Python statements will result in an error when the code is run. In the table below, briefly explain why each error occurs.

Python statements	**Briefly explain why each error occurs**
`stations = ('Pape', 'King', 'Kipling')` `stations[0] = 'St. George'`	tuples are not mutable

`st_to_line = {'Chester': 2,`
`'Davisville': 1,`
`'Union': 1}`
`if st_to_line['Dundas'] == 1:`
`print('Yonge-University')`
	'Dundas' is not a key in stations

`i = 0`
`lines = [1, 2, 3, 4]`
`while lines[i] != 5 and i < len(lines):`
`print(lines[i])`
`i = i + 1`
	IndexError: list index out of range

`stops = ['Christie', 'Bay', 'Spadina']`
`sorted_stops = stops.sort()`
`reversed_stops = sorted_stops.reverse()`
	sort returns None. reverse() cannot be called on None

`station = 'Sheppard West'`
`station[-4] = 'E'`
`station[-3] = 'a'`
	strings are not mutable

`# Assume the next line runs without error.`
 	
`f = open('stations.txt')`
 	
`f.read()`
`f.readline()[0]`
	the string is empty; index out of range

Question 2. [4 MARKS]

Fill in the boxes with the while loop condition and the while loop body required for the function to work as described in its docstring. See the bottom of this page for examples.

```
def get_and_verify_password(password: str) -> bool:
    """Repeatedly prompt the user to enter their password until they get it
    correct or until they guess wrong three times.  Return True if and only if
    the password was entered correctly.
    """

    msg = 'Enter your password: '
    guess = input(msg)
    num_guesses = 1
    while guess != password and num_guesses < 3:
        num_guesses = num_guesses + 1
        guess = input(msg)

    return guess == password
```

Question 3. [6 MARKS]

In this question, you are to write code that uses a Python dictionary where each key represents the name of a meal (*e.g.,* `'stew'`, `'eggs'`) and the associated value represents a list of table numbers (*e.g.,* `1`, `2`, `3`), with one list item for each meal order. If there are, for example, three orders for `'stew'` at table `2`, then `2` will appear three times in the list of table numbers associated with `'stew'`.

Part (a) [3 MARKS] Complete the following function according to its docstring.

```
def get_num_orders(meal_to_tables: Dict[str, List[int]], meal: str) -> int:
    """Return the number of orders for meal in meal_to_tables.

    >>> m_to_t = {'stew': [4, 1], 'eggs': [6]}
    >>> get_num_orders(m_to_t, 'stew')
    2
    >>> get_num_orders(m_to_t, 'eggs')
    1
    >>> get_num_orders(m_to_t, 'brussel sprouts')
    0
    """

    if meal in meal_to_tables:
        return len(meal_to_tables[meal])
    else:
        return 0

    # alternate:
    #
    #if meal in meal_to_tables:
    #    return len(meal_to_tables[meal])
    #return 0
```

Part (b) [3 MARKS] Complete the following function according to its docstring.

```
def order_meal(meal_to_tables: Dict[str, List[int]], meal: str, table: int) -> None:
    """Modify meal_to_tables to include a new order for meal at table.  Place
    table at the end of the list of table number(s) associated with meal.

    >>> m_to_t = {}
    >>> order_meal(m_to_t, 'stew', 4)
    >>> m_to_t == {'stew': [4]}
    True
    >>> order_meal(m_to_t, 'stew', 1)
    >>> m_to_t == {'stew': [4, 1]}
    True
    >>> order_meal(m_to_t, 'eggs', 6)
    >>> m_to_t == {'stew': [4, 1], 'eggs': [6]}
    True
    """

    if meal not in meal_to_tables:
        meal_to_tables[meal] = []
```

```
    meal_to_tables[meal].append(table)

    # alternate:
    #
    # if meal in meal_to_tables:
    #     meal_to_tables[meal].append(table)
    # else:
    #     meal_to_tables[meal] = [table]
```

Question 4. [4 MARKS]

Complete the following function according to its docstring.

```
def char_count(s: str, words: List[str]) -> List[int]:
    """Return a new list in which each item is the number of times
    that the character at the corresponding position of s appears in
    the string at the corresponding position of words.
    Lowercase and uppercase characters are considered different.

    Precondition: len(s) == len(words)

    # In the example below, 'a' is in 'apple' 1 time,
    # 'n' is in 'banana' 2 times, and
    # 'b' is in 'orange' 0 times.
    >>> char_count('anb', ['apple', 'banana', 'orange'])
    [1, 2, 0]
    >>> char_count('xdaao', ['cat', 'dog', 'cat', 'banana', 'cool'])
    [0, 1, 1, 3, 2]
    >>> char_count('fW', ['sandwiches', 'waffles'])
    [0, 0]
    """

    counts = []
    for i in range(len(words)):
        counts.append(words[i].count(s[i]))
    return counts
```

Question 5. [4 MARKS]

The docstring below is correct. However, the code in the function body contains one or more bugs. As a result the function does not work as specified in the docstring.

```
def increment_sublist(L: List[int], start: int, end: int) -> None:
    """Modify L so that each element whose index is in the range from start (inclusive)
    to end (exclusive) is incremented by 1.

    Precondition: 0 <= start < end <= len(L)

    >>> a_list = [10, 20, 30, 40, 50, 60]
    >>> increment_sublist(a_list, 0, 3)
    >>> a_list
    [11, 21, 31, 40, 50, 60]
    """
    for value in L[start:end]:
        value = value + 1
```

Part (a) [1 MARK]

Complete the example below to show what happens when the buggy function body given above is used.

```
>>> a_list = [10, 20, 30, 40, 50, 60]
>>> increment_sublist(a_list, 0, 3)
>>> a_list
```

Answer: `[10, 20, 30, 40, 50, 60]`

Part (b) [3 MARKS]

Write a new function body that correctly implements the function as described in its docstring above.

```
def increment_sublist(L: List[int], start: int, end: int) -> None:
    """ <Docstring omitted>
    """

    for i in range(start, end):
        L[i] = L[i] + 1

    # alternate solution
    for i in range(len(L[start:end]):
        L[start+i] = L[start+i] + 1
```

Question 6. [6 MARKS]

In each of the following, circle the **best** answer that follows directly below the question.

Part (a) [1 MARK] If you were searching a **sorted** list of one million unique items for a particular value, and the value being searched for was the second item in the sorted list, which algorithm would take the least time?

Answer: linear search

Part (b) [1 MARK] If you were searching a **sorted** list of one million unique items for a particular value, and the value being searched for was not in the sorted list, which algorithm would take the least time to discover that it was not in the list?

Answer: binary search

Part (c) [1 MARK] If you had an **unsorted** list of one million unique items, and knew that you would only search it once for a value, which of the following algorithms would be the fastest?

Answer: use linear search on the unsorted list

Part (d) [1 MARK] Our sorting code completes all passes of the algorithm, even if the list becomes sorted before the last pass. After how many passes of the **bubble sort** algorithm on the list `[ 3, 1, 6, 4, 9, 8 ]` could we stop because the list has become sorted?

Answer: 1

Part (e) [1 MARK] Our sorting code completes all passes of the algorithm, even if the list becomes sorted before the last pass. After how many passes of the **insertion sort** algorithm on the list `[ 9, 8, 3, 1, 6, 4 ]` could we stop because the list has become sorted?

Answer: 6

Part (f) [1 MARK] Our sorting code completes all passes of the algorithm, even if the list becomes sorted before the last pass. After how many passes of the **selection sort** algorithm on the list `[ 9, 8, 6, 4, 3, 1 ]` could we stop because the list has become sorted?

Answer: 3

Question 7. [6 MARKS]

Complete the following function according to its docstring.

Your code must not mutate the parameters!

```
def collect_sublists(L: List[List[int]], threshold: int) -> List[List[int]]:
    """Return a new list containing the sublists of L in which all the values
    in the sublist are above threshold.

    Precondition: all sublists of L have length >= 1

    >>> collect_sublists([[1, 2, 3], [4, 5, 6], [7, 8, 9]], 5)
    [[7, 8, 9]]
    >>> collect_sublists([[15, 20], [10, 11], [30, 40], [7, 17]], 10)
    [[15, 20], [30, 40]]
    """

    result = []
    for sublist in L:
        above_threshold = True
        for value in sublist:
            if value <= threshold:
                above_threshold = False
        if above_threshold:
            result.append(sublist)
    return result

    # alternate:
    # result = []
    # for sublist in L:
    #    if min(sublist) > threshold:
    #        result.append(sublist)
    # return result
```

Question 8. [3 MARKS]

Fill in the boxes to complete the docstring examples for the function below.

```
def mystery(L: List[str], D: Dict[str, str]) -> None:
    """ <Description omitted>

    >>> list1 = ['I', 'love', 'midterms']
    >>> dict1 = {'midterms': 'finals'}
    >>> mystery(list1, dict1)
    >>> list1
    [                                                                    ]

    >>> dict2 = [                                                        ]
    >>> mystery(list1, dict2)
    >>> list1
    ['We', 'love', 'programming']
    >>> list3 = ['m', 'y', 'q', 'p', 'w', 'm']
    >>> dict3 = {'m': 'r', 'q': 'r'}
    >>> mystery(list3, dict3)
    >>> list3
    [                                                                    ]

    """

    for key in D:
        index = L.index(key)
        L[index] = D[key]
```

```
Box 1:    ['I', 'love', 'finals']
Box 2:    {'I': 'We', 'finals': 'programming'}
Box 3:    ['r', 'y', 'r', 'p', 'w', 'm']
```

Question 9. [5 MARKS]

Part (a) [4 MARKS] The docstring below is correct. However, the code in the function body contains one or more bugs. As a result the function does not work as specified in the docstring.

```
def is_valid_word(potential_word: str, word_list: List[str]) -> bool:
    """Return True if and only if potential_word is one of the items in word_list.

    >>> is_valid_word('cat', ['cat', 'dog', 'fox'])
    True
    >>> is_valid_word('wombat', ['cat', 'dog', 'fox'])
    False
    """
    for word in word_list:
        if potential_word in word:
            return True
        else:
            return False
```

Complete the unittest code below so that: (1) the assertions both **fail** when the buggy function body given above is used, and (2) the assertions both **pass** when a function body that correctly implements the function as described in its docstring is used. Both arguments must have the correct type. Assume that the `is_valid_word` function has been correctly imported and may be called as written below.

```
class TestIsValidWord(unittest.TestCase):

    def test_case1(self):
        potential_word = 
        word_list = 
        actual = is_valid_word(potential_word, word_list)
        expected = False
        self.assertEqual(actual, expected)

    def test_case2(self):
        potential_word = 
        word_list = 
        actual = is_valid_word(potential_word, word_list)
        expected = True
        self.assertEqual(actual, expected)
```

SOLUTIONS

For `test_case1`: (1) `word_list` can be any non-empty list of strings with a non-empty string as its first element. `potential_word` can be anything that is a proper substring of the first element of `word_list`. For example, `'ca'` and `['cat']`. (2) `word_list == []` and any `potential_word` would work too.

For `test_case2`: `word_list` can be any list of 2 or more strings. `potential_word` can be any element in `word_list` other than the first element. For example, `'dog'` and `['cat', 'dog', 'fox']`.

Part (b) [1 MARK] Circle the term below that best describes the number of times the loop iterates when the buggy version of the `is_valid_word` function given in Part (a) is called.

(a) [constant] linear quadratic something else

Question 10. [6 MARKS]

Consider this code:

```
def mystery(n: int) -> None:
    """ <Docstring omitted.>
    """
    for i in range(n):
        for j in range(n):
            if i == j:
                print(i + j)
```

Part (a) [1 MARK]

What is printed when `mystery(3)` is executed?

```
0
2
4
```

Part (b) [1 MARK]

Write an English description of what function `mystery` prints in terms of `n`.

```
ANSWER: Print each even number from 0 to (n-1)*2 on separate lines.
    OR: Print the first n even natural numbers on separate lines.
```

Part (c) [1 MARK]

For function `mystery` the best and worst case running times are the same. Circle the term below that best describes the running time of the `mystery` function as written above.

(a) constant linear [quadratic] something else

Part (d) [2 MARKS]

The code above can be rewritten to complete the same task, but with a reduced running time. Write the body of a new version of `mystery` in which the running time expressed in terms of `n` is improved.

```
def mystery_improved(n: int) -> None:
    """ <Docstring omitted.>
    """
    for i in range(n):
        print(i + i)
```

Part (e) [1 MARK]

Circle the term below that best describes the running time of the your `mystery_improved` function.

(a) constant [linear] quadratic something else

Question 11. [12 MARKS]

Part (a) [6 MARKS]

Station data is stored in a comma separated values (CSV) file with one station's ID, name, latitude, and longitude per line in that order. Here is an example station data CSV file:

```
1,Allen,43.667158,-79.4028
12,Bayview,43.656518,-79.389
8,Chester,43.648093,-79.384749
17,Davisville,43.66009,-79.385653
```

Given the example station data CSV file opened for reading, function `build_dictionaries` returns:

```
({1: [43.667158, -79.4028], 12: [43.656518, -79.389],
  8: [43.648093, -79.384749], 17: [43.66009, -79.385653]},
 {1: 'Allen', 12: 'Bayview', 8: 'Chester', 17: 'Davisville'})
```

Complete the function `build_dictionaries` according to the example above and its docstring below. Assume the given file has the correct format.

```
def build_dictionaries(f: TextIO) -> Tuple[Dict[int, List[float]], Dict[int, str]]:
    """Return a tuple of two dictionaries with station data from f.  The first dictionary
    has station IDs as keys and station locations (two item lists with latitude and longitude)
    as values.  The second dictionary has station IDs as keys and station names as values.

    Precondition: station IDs in f are unique
    """

      id_to_name = {}
      id_to_location = {}

      for line in f:
            tokens = line.strip().split(',')   # note: strip() not required
            id_to_name[tokens[0]] = tokens[1]
            id_to_location[tokens[0]] = [float(tokens[2]), float(tokens[3])]

      return (id_to_location, id_to_name)
```

Part (b) [6 MARKS]

You may assume the function `get_distance` has been implemented:

```
def get_distance(lat1: float, long1: float, lat2: float, long2: float) -> float:
    """Return the distance between the location at lat1 and long1 and
    the location at lat2 and long2.
    """
```

Using `get_distance` as a helper function, complete function `get_closest_station` according to its docstring:

```
def get_closest_station(lat: float, long: float, id_to_location: Dict[int, List[float]],
    id_to_name: Dict[int, str]) -> str:
    """Return the name of the station in id_to_name and id_to_location that is
    closest to latitude lat and longitude long. You may assume that exactly one
    station is closest.

    Precondition: id_to_location and id_to_name have the same keys
                  and len(id_to_location) >= 1

    >>> id_to_location = {3: [40.8, -73.97], 4: [43.6, -79.4], 11: [51.5, -0.1]}
    >>> id_to_name = {3: 'Grand Central', 4: 'Union', 11: 'Blackfriars'}
    >>> get_closest_station(43.5, -79.6, id_to_location, id_to_name)
    'Union'
    """


    curr_station = ''
    curr_closest = -1

    for station_id in id_to_location:
        distance = get_distance(lat, long, id_to_location[station_id][0],
            id_to_location[station_id][1])
        if curr_closest == -1 or distance < curr_closest:
            curr_closest = distance
            curr_station = id_to_name[station_id]

    return curr_station

    # alternate solution 1 - accumulate distances, use min to find minimum
    #   distance and then deduce name
    distances = []
    for station_id in id_to_location:
        distances.append(get_distance(lat, long, id_to_location[station_id][0],
            id_to_location[station_id][1]))

    index_of_closest = distances.index(min(distances))
    closest_id = list(id_to_location.keys())[index_of_closest]
    return id_to_name[closest_id]
```

```
    # alternate solution 2 - keep parallel distances and ids lists instead
    # of extracting id for dict keys
    distances = []
    ids = []
    for station_id in id_to_location:
        distances.append(get_distance(lat, long, id_to_location[station_id][0],
            id_to_location[station_id][1]))
        ids.append(station_id)

    index_of_closest = distances.index(min(distances))
    closest_id = ids[index_of_closest]
    return id_to_name[closest_id]
```

Question 12. [8 MARKS]

In this question, you will develop a class `Restaurant` to represent a restaurant with tables. You may assume that class `Table` has been implemented and imported, and must use class `Table` when you define class `Restaurant`. The help for `Table` is below. You do not need to implement any part of class `Table`.

```
class Table()
 |  Information about a table.
 |
 |  Methods defined here:
 |
 |  __init__(self, table_id: int, num_seats: int, num_occupied: int) -> None
 |      Initialize a new table with table ID table_id, num_seats seats, and
 |      num_occupied seats occupied.
 |
 |      >>> table1 = Table(1, 4, 0)
 |      >>> table1.id
 |      1
 |      >>> table1.num_seats
 |      4
 |      >>> table1.num_occupied
 |      0
 |
 |  __str__(self) -> str
 |      Return a string representation of this table.
 |
 |      >>> table4 = Table(4, 6, 3)
 |      >>> str(table4)
 |      'Table 4: 3 of 6 seats occupied'
 |
 |  set_occupancy(self, num_occupied: int) -> None
 |      Set the number of seats occupied at this table to num_occupied.
 |
 |      Precondition: self.num_seats >= num_occupied
 |
 |      >>> table1 = Table(1, 4, 0)
 |      >>> table1.num_occupied
 |      0
 |      >>> table1.set_occupancy(3)
 |      >>> table1.num_occupied
 |      3
```

Here is the header and docstring for class `Restaurant`.

```
class Restaurant:
    """Information about a Restaurant."""
```

Part (a) [1 MARK] Complete the body of this class `Restaurant` method according to its docstring.

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

        >>> rest1 = Restaurant('UofT Diner')
        >>> rest1.name
        'UofT Diner'
        >>> rest1.tables
        []
        """

        self.name = name
        self.tables = []
```

Part (b) [3 MARKS] Complete the body of this class `Restaurant` method according to its docstring. Use class `Table` when possible.

```
    def add_table(self, num_seats: int) -> None:
        """Add a new table with num_seats seats to this restaurant. Table IDs should
        be in the order that tables are added to this restaurant starting from 1.
        No seats at the table are occupied.

        Precondition: num_seats >= 1

        >>> rest1 = Restaurant('UofT Diner')
        >>> rest1.tables
        []
        >>> rest1.add_table(4)
        >>> str(rest1.tables[0])
        'Table 1: 0 of 4 seats occupied'
        >>> rest1.add_table(6)
        >>> str(rest1.tables[1])
        'Table 2: 0 of 6 seats occupied'
        """

        table_number = len(self.tables) + 1
        table = Table(table_number, num_seats, 0)
        self.tables.append(table)
```

Part (c) [4 MARKS]

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

```
def calculate_occupancy(self) -> float:
    """Return the percentage of seats that are occupied for all tables in
    this restaurant.

    Precondition: len(self.tables) >= 1

    >>> rest1 = Restaurant('Snacks-R-Us')
    >>> rest1.add_table(2)
    >>> rest1.add_table(6)
    >>> rest1.tables[0].set_occupancy(2)
    >>> rest1.tables[1].set_occupancy(3)
    >>> rest1.calculate_occupancy()
    62.5
    """



    total_seats = 0
    total_occupied = 0

    for table in self.tables:
        total_seats += table.num_seats
        total_occupied += table.num_occupied

    return total_occupied / total_seats * 100
```

DO NOT DETACH THIS 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.

DO NOT DETACH THIS 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.

DO NOT DETACH THIS PAGE
Short Python function/method descriptions:

```
__builtins__:
  input([prompt: str]) -> 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: float) -> float
    Return the absolute value of x.
  chr(i: str) -> Unicode character
    Return a Unicode string of one character with ordinal i; 0 <= i <= 0x10ffff.
  float(x: object) -> float
    Convert x to a floating point number, if possible.
  int(x: object) -> int
    Convert x to an integer, if possible.  A floating point argument will be truncated
    towards zero.
  len(x: object) -> int
    Return the length of the list, tuple, dict, or string x.
  max(iterable: object) -> 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) -> 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: str[, mode: str]) -> TextIO
    Open a file.  Legal modes are "r" (read) (default), "w" (write), and "a" (append).
  ord(c: str) -> int
    Return the integer ordinal of a one-character string.
  print(value: object, ..., sep=' ', end='\n') -> None
    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: int], stop: int, [step: int]) -> list-like-object of int
    Return the integers starting with start and ending with stop - 1 (positive step)
    or stop + 1 (negative step), 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) -> 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[object, object]
    Return the (key, value) pairs of D, as 2-tuples.
```

DO NOT DETACH THIS PAGE

```
file open for reading (TextIO):
  F.close() -> None
    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[str]
    Return a list of the lines from the file.  Each string retains any newline.

file open for writing (TextIO):
  F.close() -> None
    Close the file.
  F.write(x: str) -> 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: object) -> None
    Append x to the end of the list L.
  L.extend(iterable: object) -> None
    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: object) -> int
    Return the lowest index of value in L, but raises an exception if value does
    not occur in S.
  L.insert(index: int, x: object) -> None
    Insert x at position index.
  L.pop([index: int]) -> object
    Remove and return item at index (default last).
  L.remove(value: object) -> None
    Remove the first occurrence of value from L.
  L.reverse() -> None
    Reverse the list *IN PLACE*.
  L.sort() -> None
    Sort the list in ascending order *IN PLACE*.

str:
  x in s --> bool
    Produce True if x is in s and False otherwise.
  str(x: object) -> str
    Convert an object into its string representation, if possible.
  S.count(sub: str[, start: int[, end: int]]) -> 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: str) -> bool
    Return True if and only if S ends with S2.
  S.find(sub: str[, i: int]) -> 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: str) -> int
    Like find but raises an exception if sub does not occur in S.
```

DO NOT DETACH THIS PAGE

```
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 the string S converted to lowercase.
S.lstrip([chars: str]) -> str
  Return a copy of the string S with leading whitespace removed.
  If chars is given and not None, remove characters in chars instead.
S.replace(old: str, new: str) -> str
  Return a copy of string S with all occurrences of the string old replaced
  with the string new.
S.rstrip([chars: str]) -> str
  Return a copy of the string S with trailing whitespace removed.
  If chars is given and not None, remove characters in chars instead.
S.split([sep: str]) -> List[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: str) -> bool
  Return True if and only if S starts with S2.
S.strip([chars: str]) -> 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 the string S converted to uppercase.
```