PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

APRIL 2016 EXAMINATIONS

CSC 108 H1S
Instructor(s): Smith and Fairgrieve

Duration—3 hours

No Aids Allowed

You must earn at least 30 out of 75 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: |_|_|_|_|_|_|_|_|_|_|_|_|_|

Last (Family) Name(s): _____

First (Given) 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 10 questions on 22 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.
- No error checking is required: assume all user input and all argument values are valid.
- Do not use `break` or `continue` on this exam.
- 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:	_____/ 7
# 2:	_____/ 6
# 3:	_____/14
# 4:	_____/ 7
# 5:	_____/ 4
# 6:	_____/ 6
# 7:	_____/ 5
# 8:	_____/12
# 9:	_____/ 6
# 10:	_____/ 8
TOTAL:	_____/75

Good Luck!

OVER. . .

Question 1. [7 MARKS]

Part (a) [4 MARKS]

Complete this function according to its docstring description.

```
def sum_values_above_threshold(value_string, threshold):
    """ (str, int) -> int

    Precondition: value_string.isdigit() returns True

    Return the sum of the individual digits in value_string that are
    greater than threshold.

    >>> sum_values_above_threshold('153382', 4)
    13
    >>> sum_values_above_threshold('12345', 5)
    0
    """
```

Part (b) [3 MARKS]

In the table below, we have outlined one test case for `sum_values_above_threshold`. Add three more test cases that could be part of a complete set of cases for thoroughly testing the function. Do not include duplicate test cases. You must not use the arguments from the docstring examples.

Test Case Description	Arguments	Expected Return Value
only last digit above threshold	`'1234', 3`	`4`

Question 2. [6 MARKS]

Each of the following sets of Python statements will result in an error message being displayed when the code is run. Explain briefly the cause of each error in the table below.

Python statements	**Explain briefly why an error message is displayed**

`food_to_count = {'burger': 3, \`
`  'fries': 5, 'salad': 7}`
`food_to_count['salad'] = 1`
`print(food_to_count[2])`	

`foods = ['soup', 'waffle', 'pizza']`
`for food in foods:`
`  food[0] = food[0].upper()`
`print(foods)`	

`grades = [75, 68, 82]`
`grades = grades.extend([90])`
`print(grades[-1])`	

`full_name = ['Jovi', 'Jon', 'Bon']`
`name = full_name[1:] + full_name[0]`
`print(name)`	

`names_to_ages = \`
`  {['Ann', 'Lee']: [21, 19]}`
`names_to_ages['Fan'] = 22`
`print(names_to_ages)`	

`filename = 'data.txt'`
`file_lines = filename.readlines()`
`print(file_lines[-1])`	

Question 3. [14 MARKS]

Part (a) [7 MARKS] Tweets have a maximum length, and so you may want to shorten a message to make it fit in a single tweet. One way to do that would be to remove unnecessary space characters. In this question, you can assume that there are no whitespace characters other than space (`' '`) in a message.

One algorithm that removes unnecessary spaces is as follows:

1. Remove unnecessary spaces from the start and end of the message using the string method `strip()`.

2. Initialize the new message to contain the first character of the stripped message (which will be a non-space character).

3. Iterate over each character in the rest of the stripped message. If a character is not a space, add it to the new message. If a character is a space, add it to the new message only if the previous character added to the new message was not a space.

4. Return the new message.

Complete the body of the `remove_unnecessary_spaces` function by filling in the boxes below to implement the given algorithm. Use the constant `SPACE` rather than the string literal `' '` in your answer.

```
def remove_unnecessary_spaces(message):
    """ (str) -> str

    Precondition: message contains at least one non-space character

    Return message with any occurrences of more than one consecutive space
    removed, and any spaces at the start or end removed.

    >>> remove_unnecessary_spaces(' I like Python.  Welcome to  108!    ')
    'I like Python. Welcome to 108!'
    >>> remove_unnecessary_spaces('Hello,             can you hear me?')
    'Hello, can you hear me?'
    """

    SPACE = ' '
    message = message.strip()

    new_message = [                                                    ]
    for ch in [                                                    ]:
        if [                                                    ]:
            [                                                    ]
    return [                                                    ]
```

Part (b) [5 MARKS] The function below uses `remove_unnecessary_spaces` as a helper function to remove unnecessary spaces from a list of messages that are potential tweets. Unfortunately, `shorten_messages` has some bugs.

Make whatever changes are necessary to the original body of the function `shorten_messages` so that it correctly matches its docstring. Write any lines that need to be modified in the **Modified Function Body** table. If any line(s) should be removed completely, write `DELETE` in the corresponding row. If a specific line requires no modification, you can leave its corresponding row blank in the **Modified Function Body** table. Do not add extra lines to the function body.

```
def shorten_messages(message_list):
    """ (list of str) -> NoneType

    Modify message_list so that any occurrences of consecutive spaces in each
    message, as well as any unnecessary spaces at the start and end of each message,
    are removed.

    >>> message_list = ['Hi       there   !!   ', ' CSC  108']
    >>> shorten_messages(message_list)
    >>> message_list
    ['Hi there !!', 'CSC 108']
    """
```

Line	Original Function Body
1.	`    for message in message_list:`
2.	`        message = remove_unnecessary_spaces(message)`
3.	`    return message_list`

Line	Modified Function Body
1.	
2.	
3.	

Part (c) [1 MARK] Assume you have a list that contains k messages, where k is a very large number. Give an expression in terms of k for how many times `remove_unnecessary_spaces` is called in ***the original function body***.

Part (d) [1 MARK] Which of the following terms best describes your Part (c) expression? Circle one.

constant linear quadratic something else

Question 4. [7 MARKS]

Part (a) [4 MARKS]

Complete the docstring for the function below by writing the description, giving two examples that return different values, and stating any necessary preconditions.

```
def mystery(s1, s2, s3):
    """ (str, str, str) -> int









    """

    count = 0
    for i in range(len(s1)):
        if s1[i] == s2[i] and s2[i] == s3[i]:
            count = count + 1
    return count
```

In the following Parts, assume the string `s1` has k characters and that the arguments satisfy the function preconditions.

Part (b) [1 MARK]

Give an expression in terms of k that represents the **maximum** number of times the line `count = count + 1` could be executed. [ ]

Part (c) [1 MARK]

Consider the particular case that k equals 3. Give an example of the arguments to `mystery` that would cause the line `count = count + 1` to be executed the **maximum** number of times.

`mystery(` [ ] `)`

Part (d) [1 MARK]

Give an expression in terms of k that represents the **minimum** number of times the line `count = count + 1` could be executed. [ ]

Question 5. [4 MARKS]

Consider the following two function definitions. Beside each code fragment in the table below, write what is printed when the code fragment is executed. Assume all of the code is contained and run in the same Python module.

```
def first_function(values):
    """ (list of int) -> NoneType
    """

    for i in range(len(values)):
        if values[i] % 2 == 1:
            values[i] = values[i] + 1

def second_function(value):
    """ (int) -> int
    """

    if value % 2 == 1:
        value = value + 1
    return value
```

Code	Output

`a = [1, 2, 3]`
`b = 1`
`first_function(a)`
`second_function(b)`
`print(a)`
`print(b)`	

`a = [1, 2, 3]`
`b = 1`
`print(first_function(a))`
`print(second_function(b))`	

Question 6. [6 MARKS]

Use the following function definitions to answer the questions on the following page.

```
def read_student_info(filename):
    """ (str) -> list of object

    Return a list representation of the student information stored in the
    file named filename.

    >>> read_student_info('student.txt')
    ['Gries,Paul', ['CSC 108', 'CSC 148', 'CSC 165', 'MAT 137']]
    """

    student_file = open(filename, 'r')
    student_name = read_student_name(student_file)
    course_list = read_student_courses(student_file)
    student_file.close()
    return [student_name, course_list]



def read_student_name(file):
    """ (file open for reading) -> str

    Return a string of the form LAST,FIRST containing the name of the student
    whose information is stored in file.
    """
    for line in file:
        first_name = file.readline().strip()
        last_name = file.readline().strip()
    return last_name + ',' + first_name



def read_student_courses(file):
    """ (file open for reading) -> list of str

    Return a list containing the courses taken by the student whose information
    is stored in file.
    """

    course_list = []
    for line in file:
        course_list.append(line.strip())
    return course_list
```

Question continued on following page...

Question 6 continued...

For the `student.txt` file provided on the right, the function call `read_student_info('student.txt')` is expected to return `['Gries,Paul', ['CSC 108', 'CSC 148', 'CSC 165', 'MAT 137']]`. However, it doesn't. You know for sure that the `read_student_info` function is correct, but there is a bug in one of the other two functions.

File contents

```
Paul
Gries
CSC 108
CSC 148
CSC 165
MAT 137
```

Part (a) [2 MARKS]

Given the function definitions on the previous page, what is actually returned by the function call `read_student_info('student.txt')` if the contents of `'student.txt'` is as shown above on the right? (The first line of the file is `Paul`.)

Part (b) [1 MARK]

The `read_student_info` function on the previous page is correct, and *exactly one* of the other two has a bug. Circle the name of the function that has the bug.

`read_student_name` `read_student_courses`

Part (c) [3 MARKS]

Rewrite the complete body of the buggy function below, so that the code would now return the expected result.

Question 7. [5 MARKS]

The code below partially implements a bidirectional version of selection sort (sometimes called 'cocktail sort') where the sorted part of the list grows from each end towards the middle. On each pass of the while loop, the largest item in the unsorted part is sorted to the top, and the smallest item is sorted to the bottom. The variables `bottom` and `top` represent the indices of the start and end of the unsorted part of the list.

Use the following function definitions to answer the questions on the following page.

```
def cocktailsort(lst):
    """ (list of number) -> NoneType

    Modify lst to sort the items from smallest to largest.

    >>> my_list = [4, 2, 5, 8, 6, 7, 3, 1]
    >>> cocktailsort(my_list)
    >>> my_list
    [1, 2, 3, 4, 5, 6, 7, 8]
    """

    top = len(lst) - 1
    bottom = 0

    while top > bottom:
        sort_to_top(lst, bottom, top)
        top = top - 1
        sort_to_bottom(lst, bottom, top)
        bottom = bottom + 1

def sort_to_top(lst, bottom, top):
    """ (list of number, int, int) -> NoneType

    Modify lst to swap the largest item in lst[bottom: top + 1] to index top.

    >>> my_list = [1, 2, 4, 6, 3, 5, 7, 8]
    >>> sort_to_top(my_list, 2, 5)
    >>> my_list
    [1, 2, 4, 5, 3, 6, 7, 8]
    """

    index_of_largest = bottom

    for j in range(bottom + 1, top + 1):
        if lst[j] > lst[index_of_largest]:
            index_of_largest = j

    lst[index_of_largest], lst[top] = lst[top], lst[index_of_largest]
```

Question continued on following page...

Question 7 continued...

Part (a) [4 MARKS] Write the function body for `sort_to_bottom` to complete the implementation. You can assume that each of the items in `lst` have a different value.

```
def sort_to_bottom(lst, bottom, top):
    """ (list of number, int, int) -> NoneType

    Modify lst to swap the smallest item in lst[bottom: top + 1] to index bottom.

    >>> my_list = [1, 2, 4, 5, 3, 6, 7, 8]
    >>> sort_to_bottom(my_list, 2, 4)
    >>> my_list
    [1, 2, 3, 5, 4, 6, 7, 8]
    """

    # complete the function body here
```

Part (b) [1 MARK]

Which of the following statements best describes the runtime behaviour of this sorting algorithm? Circle one.

(A) This algorithm does a different number of comparisons on a best case input vs a worst case input.

(B) This algorithm does not have a different best and worst case number of comparisons.

Question 8. [12 MARKS]

Part (a) [8 MARKS]

In Assignment 3, you used a pronunciation dictionary, a `dict of {str: list of str}` that mapped words to their phonemes. You also wrote a function named `last_phonemes` that extracted the part of the phoneme list that could be used to determine if words rhymed. We have provided the docstring for `last_phoneme` on the following page for your reference, and you can assume the body of `last_phoneme` has been implemented.

Now, you will extend the poetry checker to create a poetry assistant that would help a poet come up with rhyming words. A poetry assistant is a `dict of {tuple of str: list of str}` where the keys are the last phonemes (converted from lists to tuples), and the values are a list of words that end with those last phonemes. That is, a poetry assistant will map rhyming sounds to a list of words that end with that rhyming sound.

Complete the body of the `build_poetry_assistant` function to match this description and its docstring.

```
def build_poetry_assistant(words_to_phonemes):
    """ (dict of {str: list of str}) -> dict of {tuple of str: list of str}

    Return a poetry assistant dictionary from the words to phonemes in words_to_phonemes.

    >>> word_to_phonemes = {'BEFORE': ['B', 'IH0', 'F', 'AO1', 'R'],
    ...                     'THE': ['DH', 'AH0'], 'A': ['AH0'],
    ...                     'POEM': ['P', 'OW1', 'AH0', 'M'], 'OR': ['AO1', 'R']}
    >>> actual = build_poetry_assistant(words_to_phonemes)
    >>> expected = {('AH0',): ['THE', 'A'], ('AH0', M'): ['POEM'],
    ...             ('AO1', 'R'): ['BEFORE', 'OR']}
    >>> actual == expected
    True
    """
```

Question 8 continued...

```
def last_phonemes(phoneme_list):
    """ (list of str) -> list of str

    Return the last vowel phoneme and any subsequent consonant phoneme(s) from
    phoneme_list, in the same order as they appear in phoneme_list.

    >>> last_phonemes(['AE1', 'B', 'S', 'IH0', 'N', 'TH'])
    ['IH0', 'N', 'TH']
    >>> last_phonemes(['IH0', 'N'])
    ['IH0', 'N']
    """
```

Part (b) [4 MARKS] Once you have built the poetry assistant dictionary, a poet could use it to look up a word and find words that rhyme with that word. The `find_rhymes` function should produce a list of all words that rhyme with a word, not including the original word.

Complete the body of the `find_rhymes` function to match this description and its docstring. Do not mutate the contents of the poetry assistant dictionary (`phonemes_to_words`) in your function!

```
def find_rhymes(phonemes_to_words, word):
    """ (dict of {tuple of str: list of str}, str) -> list of str

    Precondition: word.isalpha() and word.isupper() are True, and word appears in
    exactly one value list in the phonemes_to_words dictionary

    Return a list of all words in phonemes_to_words that rhyme with word.
    Do not include word in the list.

    >>> phonemes_to_words = {('AO1', 'R'): ['BEFORE', 'OR'],
    ...                      ('AH0', 'M'): ['POEM'], ('AH0',): ['THE', 'A']}
    >>> find_rhymes(phonemes_to_words, 'OR')
    ['BEFORE']
    >>> find_rhymes(phonemes_to_words, 'POEM')
    []
    """
```

Question 9. [6 MARKS]

Part (a) [4 MARKS] Consider the following function header and docstring:

```
def loyalty_status(points):
    """ (int) -> str

    Precondition: points >= 0

    Return 'Elite' if points is 2500 or more, 'Regular' if points is 1000 or more
    but less than 2500, and 'New' otherwise.
    """
```

In the table below, we have outlined one test case for `loyalty_status`. Add four more test cases chosen to test the function thoroughly. Do not include duplicate test cases.

Test Case Description	`points`	Expected Return Value
over 2500	2501	`'Elite'`

Part (b) [2 MARKS] Fill in each box below with the number of the condition given on the right that is needed to correctly implement the body of the `loyalty_status` function. There are more conditions than you will need. Only fill in the boxes; do not add extra statements to the function body or change the other statements.

```
    status = 'New'

    if [          ]:
        status = 'Elite'

    if [          ]:
        status = 'Regular'

    return status
```

(1) `points > 2500`

(2) `points >= 2500`

(3) `points > 1000`

(4) `points >= 1000`

(5) `1000 <= points < 2500`

(6) `1000 <= points <= 2500`

(7) `points < 2500`

(8) `points < 1000`

Question 10. [8 MARKS]

In this question, you will develop two classes to represent users of Facebook's Messenger, and Messenger Chats. No knowledge of these applications is assumed or required.

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

```
class MessengerUser:
    """ Information about a particular Messenger user. """
```

Part (a) [1 MARK] Here is the header and docstring for method `__init__` in class `MessengerUser`. Complete the body of this method.

```
    def __init__(self, messenger_username):
        """ (MessengerUser, str) -> NoneType

        Initialize a new Messenger user that has username messenger_username,
        and an empty friends list.

        >>> user1 = MessengerUser('tom')
        >>> user1.username
        'tom'
        >>> user1.friends
        []
        """
```

Part (b) [2 MARKS] Here is the header and docstring for method `__str__` in class `MessengerUser`. Complete the body of this method.

```
    def __str__(self):
        """ (MessengerUser) -> str

        Return a string representation of this Messenger user.

        >>> user1 = MessengerUser('tom')
        >>> user1.friends.append('jen')
        >>> print(user1)
        User tom has 1 friend(s).
        >>> user2 = MessengerUser('jen')
        >>> user2.friends.extend(['paul', 'tom'])
        >>> print(user2)
        User jen has 2 friend(s).
        """
```

Part (c) [2 MARKS] Here is the header and docstring for method `are_friends` in class `MessengerUser`.

For our purposes, we say that two MessengerUsers are friends if they appear in each other's friends list. Complete the body of this method.

```
    def are_friends(self, other_user):
        """ (MessengerUser, MessengerUser) -> bool

        Return True iff this MessengerUser and other_user are friends.

        >>> user1 = MessengerUser('tom')
        >>> user1.friends.extend(['jen', 'paul'])
        >>> user2 = MessengerUser('jen')
        >>> user2.friends.extend(['tom', 'paul'])
        >>> user3 = MessengerUser('paul')
        >>> user3.friends.extend(['jen'])
        >>> user1.are_friends(user2)
        True
        >>> user1.are_friends(user3)
        False
        """
```

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

```
class MessengerChat:
    """ Information about a Messenger Chat. """
```

Part (d) [1 MARK] Here is the header and docstring for method `__init__` in class `MessengerChat`.

Complete the body of this method.

```
    def __init__(self, chat_id, chat_initiator):
        """ (MessengerChat, int, MessengerUser) -> NoneType

        Initialize a new Messenger chat that has chat id chat_id and a list of members
        that initially only contains chat_initiator.

        >>> user1 = MessengerUser('tom')
        >>> chat1 = MessengerChat(201, user1)
        >>> chat1.chat_id
        201
        >>> chat1.chat_members == [user1]
        True
        """
```

Part (e) [2 MARKS] Here is the header and docstring for method `add_member` in class `MessengerChat`. Complete the body of this method, using the same definition of friends as in Part (c). For this method, you must call previous methods and not duplicate any existing code.

```
def add_member(self, potential_member):
    """ (MessengerChat, MessengerUser) -> bool

    Precondition: All existing members of this MessengerChat are friends
    with at least one other member, or the chat contains only the chat initiator.
    There is at least one member in the MessengerChat.

    Return True iff potential_member can be added to this chat, and if so,
    modify this MessengerChat's chat_members to add potential_member
    to the chat. A MessengerUser can be added to this chat if and only if
    they are friends with at least one other member of the chat.

    >>> user1 = MessengerUser('tom')
    >>> user1.friends.extend(['paul', 'jen'])
    >>> user2 = MessengerUser('jen')
    >>> user2.friends.extend(['tom', 'paul'])
    >>> user3 = MessengerUser('max')
    >>> user3.friends.extend(['paul'])
    >>> chat1 = MessengerChat(201, user1)
    >>> chat1.add_member(user2)
    True
    >>> chat1.chat_members == [user1, user2]
    True
    >>> chat1.add_member(user3)
    False
    >>> chat1.chat_members == [user1, user2]
    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.
  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 is in s.
  str(x) -> str
    Convert an object into its string representation, if possible.
  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 the string S converted to lowercase.
S.lstrip([chars]) -> 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, new) -> str
  Return a copy of string S with all occurrences of the string old replaced
  with the string new.
S.rstrip([chars]) -> 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]) -> 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 the string S converted to uppercase.
```