PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

DECEMBER 2015 EXAMINATIONS

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

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 9 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: _____/ 9
2: _____/10
3: _____/ 4
4: _____/ 8
5: _____/10
6: _____/10
7: _____/ 7
8: _____/ 8
9: _____/ 9

TOTAL: _____/75

Question 1. [9 MARKS]

Part (a) [4 MARKS] For Assignment 2, we provided the starter code shown in the first column of the table below.

Make whatever changes are necessary to the function `encrypt_letter` so that it can be used to shift by `n` places, where `n` is a nonnegative integer less than 26, instead of by 3 places. Rewrite any lines that need to be modified in the second column. If a specific line requires no modification, you can leave it blank. We have already modified the first line for you.

Original Function	**Modified Function**
`def encrypt_letter(letter):`	`def encrypt_letter(letter, n):`
`    """ (str) -> str`	`    (str, int) -> str`
`    Precondition: len(letter) == 1 and letter.isupper()`	`    Precondition: len(letter) == 1 and letter.isupper() and 0 <= n < 26`
`    Return letter encrypted by shifting 3 places to the right.`	`    Return letter encrypted by shifting n places to the right.`
`    >>> encrypt_letter('V')`	`    >>> encrypt_letter('V', 1)`
`    'Y'` `    """`	`    'W'`
`    ord_diff = ord(letter) - ord('A')`	
`    new_char_ord = (ord_diff + 3) % 26`	`    new_char_ord = (ord_diff + n) % 26`
`    return chr(new_char_ord + ord('A'))`	

Part (b) [5 MARKS] Complete the following function according to its docstring description. Note that your solution to this part should **not** use your solution to Part (a).

```
def encrypt_message(plaintext):
    """ (str) -> str

    Precondition: plaintext contains only letters (i.e., alphabetic characters)
    and single spaces.

    Return the ciphertext generated by reversing each word in plaintext.

    >>> encrypt_message('HELLO COMPUTER SCIENCE WORLD')
    'OLLEH RETUPMOC ECNEICS DLROW'
    >>> encrypt_message('Good Luck')
    'dooG kcuL'
    """

    words = plaintext.split()
    ciphertext = ''
    for word in words:
        ciphertext = ciphertext + word[::-1] + ' '
    return ciphertext.strip()

    # Alternative solution
    words = plaintext.split()
    ciphertext = ''
    for word in words:
        reversed = ''
        for letter in word:
            reversed = letter + reversed
        ciphertext += reversed + ' '
    return ciphertext.strip()
```

Question 2. [10 MARKS]

Part (a) [5 MARKS] Complete the following function according to its docstring description.

You *must* use this algorithm:

- Create a copy of the original list.
- Create a new empty list to hold the biggest three numbers.
- Find the biggest number in the copy of the original list, add it to a new list, and remove it from a copy of the original list. Repeat twice more.

```
def top_three(scores):
    """ (list of int) -> list of int

    Precondition: len(scores) >= 3

    Return a new list that contains the three largest numbers from scores.

    >>> top_three([5, 4, 6, 3, 1, 2, 7])
    [7, 6, 5]
    >>> top_three([6, 2, 11, 12, 7, 12, -3])
    [12, 12, 11]
    """

    # Start with a copy of the list so we don't modify the original.
    scores_copy = scores[:]

    biggest = []

    for i in range(3):
        curr_biggest = max(scores_copy)
        biggest.append(curr_biggest)
        scores_copy.remove(curr_biggest)

    return biggest
```

Part (b) [5 MARKS] Complete the following function according to its docstring description.

You *must* use this algorithm:

- Remove the smallest number from the list repeatedly until only the three largest numbers remain in the list.

```
def top_three_mutate(scores):
    """ (list of int) -> NoneType

    Precondition: len(scores) >= 3

    Modify scores so that it contains only the three largest numbers,
    in the same order they appear in the original list.

    >>> my_scores = [5, 4, 6, 3, 1, 2, 7]
    >>> top_three_mutate(my_scores)
    >>> my_scores
    [5, 6, 7]
    >>> my_scores = [6, 2, 12, 11, 7, 12, -3]
    >>> top_three_mutate(my_scores)
    >>> my_scores
    [12, 11, 12]
    """

    while len(scores) > 3:
        scores.remove(min(scores))
```

Question 3. [4 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**

`cats = ['Fluffy', 'Mittens']`
`message = 'I love ' + cats[1:]`
`print(message)`
	cannot concatenate a str and a list

`animals_to_foods = {'cat': 'kibble', \`
`  'dog': 'bones', 'horse': 'hay'}`
`animals_to_foods['cat'] = 'tuna'`
`print(animals_to_foods[0])`
	0 is not a key in the dict, and/or dicts don't have indices

`ages = [22, 19, 23, 18, 34, 31, 20]`
`ages = ages.sort()`
`print(ages[0])`
	sort returns None, so ages refers to None

`words = ['waffle', 'pancake', 'omelette']`
`for word in words:`
`  word[0] = word[0].upper()`
`print(words)`
	strings aren't mutable

Question 4. [8 MARKS]

Complete the body of the function below according to its docstring description.

```
def change_letters(word_list, char, position):
    """ (list of str, str, int) -> NoneType

    Precondition: len(char) == 1 and position >= 0

    Modify word_list so that character char appears at index position of each
    item in the list. If position is not a valid index for a particular
    item, extend that string by filling all missing characters with char,
    up until the character at index position.

    >>> words = ['chocolate', 'milk']
    >>> change_letters(words, 's', 0)
    >>> words
    ['shocolate', 'silk']
    >>> words = ['abcd', 'efg', '']
    >>> change_letters(words, 'A', 4)
    >>> words
    ['abcdA', 'efgAA', 'AAAAA']
    """

    for i in range(len(word_list)):
        if position < len(word_list[i]):
            word_list[i] = word_list[i][:position] + char + word_list[i][position + 1:]
        else:
            word_list[i] = word_list[i] + char * (position - len(word_list[i]) + 1)
            # Alternative solution
            # for j in range(len(word_list[i], position + 1):
            #     word_list[i] = word_list[i] + char * j
```

Question 5. [10 MARKS]

Part (a) [3 MARKS] Write the body of the function according to its docstring description. You must use the given constant.

```
CURRENT_YEAR = 2015

def get_age(birthday_string):
    """ (str) -> int

    Precondition: birthday_string has format YYYY-MM-DD and represents a valid
    year, month, and day, no later than the end of CURRENT_YEAR.

    Return the age of the person with the birthdate birthday_string at the end
    of the year CURRENT_YEAR.

    >>> get_age('2006-08-08')
    9
    >>> get_age('1997-12-31')
    18
    >>> get_age('2015-01-01')
    0
    """

    return CURRENT_YEAR - int(birthday_string[:4])
```

Part (b) [7 MARKS]

You have a (potentially large) file named `skype_data.txt` that contains information about Skype users. Each user has three pieces of information in the file, each labelled with one of the keywords `USERNAME`, `LOCATION`, or `BIRTHDATE`. The order of the information can vary, but all three pieces for one user will appear before the next user begins. Each keyword is on its own line, and the line below contains the particular information for the user.

An example of a possible `skype_data.txt` file is shown on the right.

We want a new file that contains the ages of the users from the `skype_data.txt` file, with one line per user, in the same order as they appear in the data file. The age will be computed as in Part (a). For example, the file based on the `skype_data.txt` data file on the right would be:

```
18
9
44
```

```
USERNAME
skyper_108
LOCATION
Toronto
BIRTHDATE
1997-12-31
USERNAME
orange_cat
BIRTHDATE
2006-08-08
LOCATION
Toronto
LOCATION
Ottawa
USERNAME
jtrudeau
BIRTHDATE
1971-12-25
```

... Part (b) continued on the following page ...

Part (b) continued...

Write the body of the `write_age_file` function according to its docstring description so that you will be able to generate the new file. You must use the function you wrote in Part (a) as a helper function.

```
def write_age_file(data_file, age_file):
    """ (file open for reading, file open for writing) -> NoneType

    Precondition: data_file is a valid Skype data file.

    Write the age of each user in the data_file to its own line in age_file.
    The age of a user is as defined in Part (a).
    """


    for line in data_file:

        if line.strip() == 'BIRTHDATE':
            birthdate = data_file.readline().strip()
            age_file.write(str(get_age(birthdate)) + '\n')
```

Question 6. [10 MARKS]

Part (a) [2 MARKS] Complete the docstring example below to show that the `can_fill_order` function does not modify its parameters.

```
def can_fill_order(order_dict, inventory_dict):
    """ (dict of {str: str}, dict of {str: int}) -> bool

    Return True iff the quantity (dict value) of every item (dict key) in inventory_dict
    is greater than or equal to the quantity of the item ordered in order_dict.
    If an item in order_dict is not in inventory_dict, return False.

    >>> inventory = {'shirt': 2, 'mug': 2}
    >>> can_fill_order({'Ann': 'mug', 'Bob': 'mug', 'Lee': 'mug'}, inventory)
    False
    >>> can_fill_order({'Ann': 'shirt', 'Bob': 'mug', 'Lee': 'mug'}, inventory)
    True
    >>> can_fill_order({'Ann': 'mug', 'Bob': 'mug', 'Lee': 'hat'}, inventory)
    False
    >>> inventory = {'shirt': 2, 'mug': 2}
    >>> order = {'Ann': 'mug', 'Bob': 'mug'}
    >>> result = can_fill_order(order, inventory)
    >>> order == {'Ann': 'mug', 'Bob': 'mug'}
    True
    >>> inventory == {'shirt': 2, 'mug': 2}
    True
    """
```

Part (b) [8 MARKS] Write the body of the function according to its docstring description.

```
    # Sample solution
    count = {}
    for order in order_dict:
        if order_dict[order] not in count:
            count[order_dict[order]] = 0
        count[order_dict[order]] += 1

    for item in count:
        if item not in inventory_dict or count[item] > inventory_dict[item]:
            return False
    return True
```

Question 7. [7 MARKS]

Part (a) [3 MARKS] The letters in the English alphabet can be separated into two categories: vowels (`a, e, i, o, u`) and consonants (non-vowels, the letters that are not vowels). Note that with this definition, the letter `y` is considered to be a consonant. Consider the following function header and docstring:

```
def separate_message(message):
    """ (str) -> list of str

    Precondition: message.isalpha() and message.islower()

    Return a two-item list in which the first item is a str that contains the different vowels
    that occur in message, in the same order as they first appear in message, and the second
    item is a str that contains the different consonants (non-vowels) that occur in message,
    in the same order as they first appear in message.
    """
```

In the table below, we have outlined one test case for `separate_message`. 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.

Test Case Description	`message`	Expected Return Value
multi-character string, repeated vowels and consonants	`'papadopoulou'`	`['aou', 'pdl']`
empty string	`''`	`['','']`
one character string, no vowel	`'p'`	`['', 'p']`
one character string, vowel	`'a'`	`['a','']`
multi-character string, consonants only, no repeats	`'bcdf'`	`['', 'bcdf']`
multi-character string, vowels only, no repeats	`'aeio'`	`['aeio', '']`
multi-character string, vowels and consonants, no repeats	`'computer'`	`['oue', 'cmptr']`
multi-character string, repeated consonants only	`'bbddbc'`	`['', 'bdc']`
multi-character string, repeated vowels only	`'aeaeiiae'`	`['aei', '']`

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

```
def frequent_flyer_status(miles):
    """ (int) -> str

    Precondition: miles >= 0

    Return 'Gold' if miles is at least 1000, 'Silver' if miles is at least 500 and less
    than 1000, and 'Bronze' otherwise.
    """
```

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

Test Case Description	`miles`	Expected Return Value
under 500	`250`	`'Bronze'`
500	`500`	`'Silver'`
between 500 and 1000	`750`	`'Silver'`
1000	`1000`	`'Gold'`
over 1000	`1111`	`'Gold'`

Question 8. [8 MARKS]

The `bogosort` algorithm is a very simple algorithm for sorting a list. In the first step, we check to see if the list is sorted. If it is, the algorithm is finished and no more steps are taken. If it isn't, we randomly rearrange the items in the list and go back to the first step. The Python `random` module function `shuffle` can be used to randomly rearrange the items in a list. (The `shuffle` function is described in the API at the end of the exam.) Here is a Python implementation of the `bogosort` algorithm, with one function missing:

```
import random

def bogosort(lst):
    """ (list of int) -> NoneType

    Modify lst to sort the items from smallest to largest.

    >>> my_list = [42, 17, 56]
    >>> bogosort(my_list)
    >>> my_list
    [17, 42, 56]
    """

    while not is_sorted_list(lst):
        random.shuffle(lst)
```

Part (a) [4 MARKS]

Complete the body of the `is_sorted_list` function by filling in the four boxes below, and thereby finish the code needed to run `bogosort`. Recall that a list is sorted if and only if every pair of adjacent items in the list are in sorted order.

```
def is_sorted_list(lst):
    """ (list of int) -> bool

    Return True if and only if the items in lst are sorted from smallest to largest.

    >>> is_sorted_list([12, 12, 2015])
    True
    >>> is_sorted_list([11, 1, 2016])
    False
    """

    for i in range(1,len(lst)):
        if lst[i] < lst[i-1]:
            return False
    return True
```

Part (b) [1 MARK]

Write a list of length 4 for which the `is_sorted_list` function would have its **best case** running time. In the best case, `is_sorted_list` would perform the fewest possible number of comparisons. Place your answer in the box.

$4, 1, 2, 3$

Part (c) [1 MARK]

Circle the term below that best describes the **best case** running time of the `is_sorted_list` function.

(a) constant linear quadratic something else

Part (d) [1 MARK]

Write a list of length 4 for which the `is_sorted_list` function would have its **worst case** running time. In the worst case, `is_sorted_list` would perform the most possible number of comparisons. Place your answer in the box.

$1, 2, 3, 4$

Part (e) [1 MARK]

Circle the term below that best describes the **worst case** running time of the `is_sorted_list` function.

constant (b) linear quadratic something else

Question 9. [9 MARKS]

In this question, you will develop two classes to represent Skype users and Skype calls.

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

```
class SkypeUser:
    """ Information about a particular Skype user. """
```

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

```
    def __init__(self, skype_username, skype_location, skype_contacts):
        """ (SkypeUser, str, str, list of str) -> NoneType

        Initialize a new Skype user that has username skype_username,
        location skype_location and contacts skype_contacts.

        >>> user1 = SkypeUser('uoft123', 'Toronto', ['debbie', 'paul'])
        >>> user1.username
        'uoft123'
        >>> user1.location
        'Toronto'
        >>> user1.contacts
        ['debbie', 'paul']
        """


        self.username = skype_username
        self.location = skype_location
        self.contacts = skype_contacts
```

Part (b) [1 MARK] Here is the header and docstring for method `__str__` in class `SkypeUser`. Complete the body of this method.

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

        Return a string representation of this Skype user.

        >>> user1 = SkypeUser('uoft123', 'Toronto', ['debbie'])
        >>> print(user1)
        uoft123 lives in Toronto and has 1 contact(s).
        >>> user2 = SkypeUser('mel', 'Vancouver', ['paul', 'debbie'])
        >>> print(user2)
        mel lives in Vancouver and has 2 contact(s).
        """


        return "{0} lives in {1} and has {2} contact(s).".format(self.username, \
             self.location, len(self.contacts))
```

Part (c) [2 MARKS] Here is the header and docstring for method `same_contacts` in class `SkypeUser`. Complete the body of this method.

```
def same_contacts(self, user_contacts):
    """ (SkypeUser, list of str) -> bool

    Return True iff this Skype user has the same contacts, in any order,
    as the ones in user_contacts. The elements in user_contacts and
    the contacts of this Skype user may be reordered.

    >>> user1 = SkypeUser('uoft123', 'Toronto', ['debbie', 'paul'])
    >>> contact_list1 = ['paul', 'debbie']
    >>> contact_list2 = []
    >>> user1.same_contacts(contact_list1)
    True
    >>> user1.same_contacts(contact_list2)
    False
    """


    contacts = self.contacts[:]
    other_user_contacts = user_contacts[:]
    contacts.sort()
    other_user_contacts.sort()
    return contacts == other_user_contacts

    # Alternatively they could do:
    return sorted(self.contacts) == sorted(user_contacts)
```

Part (d) [2 MARKS] Here is the header and docstring for method `__eq__` in class `SkypeUser`. Complete the body of this method.

```
def __eq__(self, other_user):
    """ (SkypeUser, SkypeUser) -> bool

    Return True iff this Skype user has the same contacts as
    Skype user other_user.

    >>> user1 = SkypeUser('uoft123', 'Toronto', ['debbie', 'paul'])
    >>> user2 = SkypeUser('mel', 'Vancouver', ['paul', 'debbie'])
    >>> user1 == user2
    True
    >>> user3 = SkypeUser('uoft_cs1', 'Toronto', \
    ['debbie', 'ioana', 'alex', 'paul'])
    >>> user1 == user3
    False
    """

    return self.same_contacts(other_user.contacts)
```

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

```
class SkypeCall:
    """ Information about a Skype call. """
```

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

```
    def __init__(self, call_id, call_initiator):
        """ (SkypeCall, int, SkypeUser) -> NoneType

        Initialize a SkypeCall with a call id call_id and a list of members
        that initially only contains Skype user call_initiator.

        >>> user1 = SkypeUser('uoft123', 'Toronto', ['debbie'])
        >>> call1 = SkypeCall(201, user1)
        >>> call1.call_id
        201
        >>> call1.call_members == [user1]
        True
        """
        self.call_id = call_id
        self.call_members = [call_initiator]
```

Part (f) [2 MARKS] Here is the header and docstring for method `add_members` in class `SkypeCall`. Complete the body of this method.

```
    def add_members(self, potential_members):
        """ (SkypeCall, list of SkypeUser) -> int

        Precondition: All existing members of this Skype call share the same
        location and there is at least one member in the call.

        Add call members from potential_members to this Skype call and return
        the number of newly added call members.
        A person from potential_members can only be added to this call if
        their location is the same as the location of all the other members of
        that call.

        >>> user1 = SkypeUser('uoft123', 'Toronto', ['debbie'])
        >>> user2 = SkypeUser('mel', 'Vancouver', [])
        >>> user3 = SkypeUser('max', 'Toronto', [])
        >>> call1 = SkypeCall(201, user1)
        >>> call1.add_members([user2, user3])
        1
        >>> # only Skype user with username max was added.
        """
        added_members = 0
        for member in potential_members:
            if member.location == self.call_members[0].location:
```

```
            self.call_members.append(member)
            added_members += 1
    return added_members
```

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.

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.

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.

random:
  shuffle(L) -> NoneType
    Randomly shuffle (re-order) list L in place, and return None.
```