PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

DECEMBER 2014 EXAMINATIONS

CSC 108 H1F
Instructor(s): Campbell and Fairgrieve

Duration — 3 hours

No Aids Allowed

You must earn at least 28 out of 70 Total Marks (40%) on this final examination in order to pass the course.

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 23 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.
- You may not use `break` or `continue` on this exam.
- If you use any space for rough work, indicate clearly what you want marked.

MARKING GUIDE

1: _____/ 5
2: _____/ 4
3: _____/ 6
4: _____/ 3
5: _____/11
6: _____/ 6
7: _____/ 6
8: _____/ 8
9: _____/10
10: _____/11

TOTAL: _____/70

Question 1. [5 MARKS]

Beside each code fragment in the table below, write what is printed. If the code would cause an error, write ERROR and give a brief explanation.

Code	Output or Cause of Error

`L1 = [1, 2, 3]`
`L2 = [L1, L1]`
`print(L2)`
	`[[1, 2, 3], [1, 2, 3]]`

`L3 = [1, 2, 3]`
`L4 = [L3, L3]`
`L3.append(4)`
`print(L4)`
	`[[1, 2, 3, 4], [1, 2, 3, 4]]`

`s = 'hello'`
`L5 = [s, s]`
`s = 'hello' + ' there'`
`print(L5)`
	`['hello', 'hello']`

`D1 = {}`
`D1['a'] = 1`
`D1['b'] = 2`
`D1['a'] = 3`
`print(len(D1))`
	2

`D2 = {1: 'hi', 2: 'bye'}` `print(D2[-1])`	KeyError: -1 is not a key in D2

Question 2. [4 MARKS]

The function call `range(len(L) - 1, 0, -1)` produces a sequence of integers from one less than the length of the list `L` down to one. For example, if variable `L` refers to `[10, 7, 1, 6, 8]`, then calling `range(len(L) - 1, 0, -1)` will produce integers `4`, `3`, `2`, and `1`.

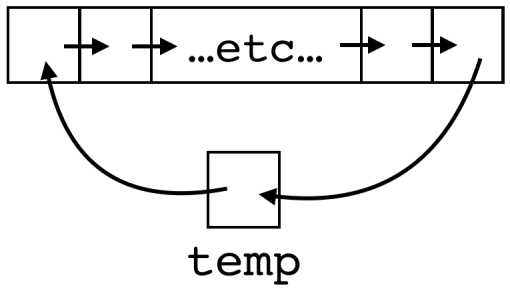
The function below moves all items in list `L` one position to the right, with the rightmost item moving to the leftmost position.

```
def shift_right(L):
    temp = L[len(L) - 1]
    for i in range(len(L) - 1, 0, -1):
       L[i] = L[i - 1]

    L[0] = temp
```

For example, if `L` initially refers to `[1, 2, 3, 4, 5]`, then after `shift_right` has been executed, `L` would refer to `[5, 1, 2, 3, 4]`.

If we were to show the effect of moving the elements of a list in this way in a diagram, it might look something like this:



Write Python code that moves all items in list `L` one position to the **left**, with the leftmost item moving to the rightmost position.

```
def shift_left(L):

    temp = L[0]                    # line 1
    for i in range(len(L) - 1):    # line 2
       L[i] = L[i + 1]             # line 3

    L[-1] = temp                   # line 4
```

Question 3. [6 MARKS]

According to Assignment 3, a *syllable* is a phoneme whose last character is 0, 1, or 2. Here is the header of a helper function that we wrote for our solution to Assignment 3:

```
def is_syllable(phoneme):
    """ (str) -> bool

    Return True iff phoneme is a syllable.
    """
```

In our Assignment 3 solution, we wrote another helper function named `count_syllables` with this header:

```
def count_syllables(words_pronunciation):
    """ (list of list of str) -> int

    Return the number of syllables in words_pronunciation.

    >>> count_syllables([['N', 'OW1'], ['G', 'UW1', 'F', 'IY0']])
    3
    """
```

Here are the lines of the function body of `count_syllables` in random order and with indentation removed:

```
    for phoneme in word_sounds:
    return num
    for word_sounds in words_pronunciation:
    num = 0
    if is_syllable(phoneme):
    num = num + 1
```

Complete the function body of `count_syllables` by using the six lines given above in the correct order and with the correct indentation:

```
    num = 0
    for word_sounds in words_pronunciation:
        for phoneme in word_sounds:
            if is_syllable(phoneme):
                num = num + 1

    return num
```

Question 4. [3 MARKS]

Consider the function `remove_extra_spaces` given below. The two `while` loop conditions are missing. Write code in each box so that the function does what the docstring says it should.

```
def remove_extra_spaces(s):
    """ (str) -> str

    Return a string that is the same as s but with each group of multiple spaces
    replaced by a single space.

    >>> remove_extra_spaces('a     b')
    'a b'
    >>> remove_extra_spaces('   a   b c  ')
    ' a b c '
    >>> remove_extra_spaces('      ')
    ' '
    >>> remove_extra_spaces('')
    ''
    """

    single_space = ' '    # there is one space between these two quotes
    result = ''           # there is no space between these two quotes
    i = 0

    while [ i < len(s)                                    ] :

        current_character = s[i]
        result = result + current_character
        i = i + 1
        if current_character == single_space:

            while [ i < len(s) and s[i] == single_space   ] :

                i = i + 1

    return result
```

Question 5. [11 MARKS]

Recall from Assignment 1 that a *tweet* is a string that is between 1 and 140 characters long (inclusive). A *username* is a string of letters and/or digits that is between 1 and 14 characters long (inclusive). A username is mentioned in a tweet by including *@username* in the tweet.

Unlike Assignment 1, for this question, the tweet `'I love @dancing'` mentions only username `dancing`, but not `dan` or other substrings of `dancing`.

Note: you may assume that the tweets will not contain any punctuation.

Part (a) [7 MARKS] Complete the following function according to its docstring description.

```
def build_username_to_mention_count(tweet_list):
    """ (list of str) -> dict of {str: int}

    Return a dictionary where each key is a username mentioned in tweet_list
    and each value is the total number of times this username was mentioned in
    all of the tweets in tweet_list.

    >>> tweets = ['hi @me and @you', '@me yesterday', 'i saw @you', '@yo @you there']
    >>> d = build_username_to_mention_count(tweets)
    >>> d == {'yo': 1, 'me': 2, 'you': 3}
    True
    """

    username_to_mentions = {}                              # line 1

    for tweet in tweet_list:                               # line 2
        tweet_parts = tweet.split()                        # line 3

        for part in tweet_parts:                           # line 4

            if part.startswith('@'):                       # line 5
                username = part[1:]                        # line 6
                if username not in username_to_mentions:   # line 7
                    username_to_mentions[username] = 0     # line 8
                username_to_mentions[username] += 1        # line 9

    return username_to_mentions                            # line 10
```

Part (b) [4 MARKS] Here is the header of a helper function that has been implemented:

```
def invert_dictionary(d):
    """ dict of {str: int} -> dict of {int: str}

    Precondition: the values in d's key/value pairs are all different

    Return a new dictionary that is the inverse of d. (See example below.)

    >>> d = invert_dictionary({'me': 2, 'yo': 1, 'you': 3})
    >>> d == {2: 'me', 1: 'yo', 3: 'you'}
    True
    """
```

Complete the following function according to its docstring description. When possible, use the function from Part (a) and the `invert_dictionary` function as helper functions.

```
def most_mentioned_username(tweet_list):
    """ (list of str) -> str

    Preconditions:
    - at least one of the strings in tweet_list will include a mention
    - each username mentioned in tweet_list will have a unique total number
      of mentions in tweet_list (i.e., no ties for the number of mentions)

    Return the username with the most mentions in tweet_list.

    >>> tweets = ['hi @you there', '@me yesterday', 'saw @you']
    >>> most_mentioned_username(tweets)
    'you'
    """


    username_to_mentions = build_username_to_mentions(tweet_list)
    mentions_to_username = invert_dictionary(username_to_mentions)

    mentions = list(mentions_to_username.keys())
    max_mentions = max(mentions)

    return mentions_to_username[max_mentions]
```

Question 6. [6 MARKS]

Consider this function header:

```
def replace_items(L, D):
    """ (list of int, dict of {int: int}) -> NoneType

    Some of the items in L may be keys in D.  Replace those items with the
    associated values in D.

    >>> L = [1]
    >>> replace_items(L, {1: 2})
    >>> L
    [2]
    >>> L = [3]
    >>> replace_items(L, {1: 2})
    >>> L
    [3]
    >>> L = [1, 3, 1]
    >>> replace_items(L, {1: 2, 3: 4, 5: 6})
    >>> L
    [2, 4, 2]
    """
```

Part (a) [3 MARKS] Write the body of the function.

```
    for i in range(len(L)):
        if L[i] in D:
            L[i] = d[L[i]]
```

Part (b) [3 MARKS] Consider this possible solution:

```
    for k in D:
        for i in range(len(L)):
            if L[i] == k:
                L[i] = D[k]
```

This code passes all the above doctests but is incorrect. Complete the table below with one test case that demonstrates that this code is incorrect.

L	D	Expected Return Value	Actual Return Value
`[ 0, 1 ]`	`{ 0:1, 1:2 }`	`[ 1, 2 ]`	it could return `[2, 2]`

The problem will occur if a list item is replaced by a value that is a key in the dictionary.

Question 7. [6 MARKS]

Consider this function header:

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

    Precondition: no uppercase letter appears more than once in
    message; no lowercase letter appears more than once in message.

    Return a list that contains two lists. Each lowercase letter from
    message appears as an item in the first list and each uppercase letter from
    message appears as an item in the second list.  The letters should appear
    in the lists in the same order as they appear in message.  Characters
    in message that are not letters do not appear in either list.
    """
```

In the table below, we have outlined two test cases for `organize_letters`. Add six more test cases chosen to test the function thoroughly. Do not include duplicate test cases.

Test Case Description	**message**	**Return Value**
empty string	`''`	`[[],[]]`
one lowercase letter	`'p'`	`[['p'], []]`
one uppercase letter	`'R'`	`[[],['R']]`
one non-letter	`'?'`	`[[],[]]`
multiple lowercase letters	`'abc'`	`[['a', 'b', 'c'], []]`
multiple uppercase letters	`'XYZ'`	`[[], ['X', 'Y', 'Z']]`
multiple non-letters	`'123'`	`[[], []]`
mix of uppercase, lowercase, and non-letters	`'HAIRY5Oth'`	`[['t', 'h'], ['H', 'A', 'I', 'R', 'Y']]`

Question 8. [8 MARKS]

Suppose you have two files that are each in alphabetic order. For example, the files `f1.txt` and `f2.txt` given below:

`f1.txt`:

```
plante
potvin
sawchuk
wregget
```

`f2.txt`:

```
dryden
plante
roy
```

Complete the function on the following page according to its docstring description. For the example files shown above, the function should return:

```
['dryden\n', 'plante\n', 'plante\n', 'potvin\n', 'roy\n', 'sawchuk\n', 'wregget\n']
```

You **must not** use the `list` method `sort`.

Note that the files may have lines in common. If that is the case, the common lines will appear more than once in the list that is returned. (In the example above, the line `'plante\n'` appears in both files.)

You will be marked not only on correctness, but also on design. Hint: take advantage of the fact that each file is already sorted.

Use the following space for rough work. The code written on this page will not be marked unless you indicate that it should be.

(Question continued from previous page.)

```
def merge_files(file_one, file_two):
    """ (file open for reading, file open for reading) -> list of str

    Precondition: both file_one and file_two are in alphabetic order;
    both file_one and file_two contain at least one line; each line
    contains one lowercase word.

    Return the combination of the lines from file_one and file_two as a list of
    strings that is in alphabetic order.

    >>> f1 = open('f1.txt')
    >>> f2 = open('f2.txt')
    >>> merge_files(f1, f2)
    ['dryden\n', 'plante\n', 'plante\n', 'potvin\n', 'roy\n', 'sawchuk\n', 'wregget\n']
    """

    # list accumulator for lines
    merged_lines = []

    # read first line from each file
    line_one = file_one.readline()
    line_two = file_two.readline()

    # while neither file exhausted, accumulate a line and read a new line
    while line_one != "" and line_two != "":
        # select appropriate line
        if line_one <= line_two:
            merged_lines.append(line_one)
            line_one = file_one.readline()
        else:
            merged_lines.append(line_two)
            line_two = file_two.readline()

    # accumulate remaining lines in unexhausted file
    if line_one == "":
        # file_one exhausted
        # write file_two leftover and get remaining from file_two
        merged_lines.append(line_two)
        lastLines = file_two.readlines()
    else:
        # file_two exhausted
        # write file_one leftover and get remaining from file_one
        merged_lines.append(line_one)
        lastLines = file_one.readlines()

    merged_lines.extend(lastLines)

    return merged_lines
```

Question 9. [10 MARKS]

In this question, you will develop two classes to keep track of restaurants and reviews of restaurants.

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

```
class Review:
    """ A review of a restaurant. """
```

Part (a) [2 MARKS]

Complete method `__init__` for class `Review`.

Note: you will most likely not need all of the space on this page.

```
    def __init__(self, reviewer_name, review_comments, is_recommended):
        """ (Review, str, str, bool) -> NoneType

        Record the reviewer's name reviewer_name, the reviewer's comments
        review_comments, and the reviewer's recommendation is_recommended.

        >>> review = Review('Lynn Crawford', 'Delicious food!', True)
        >>> review.reviewer
        'Lynn Crawford'
        >>> review.comments
        'Delicious food!'
        >>> review.recommend
        True
        """



        self.reviewer = reviewer_name
        self.comments = review_comments
        self.recommend = is_recommended
```

Part (b) [2 MARKS] Here is the header, type contract, and description for method `change_comments` in class `Review`. Add an example that creates a `Review` object, changes the comments, and shows that those comments have been changed. Also write the body of the method.

```
def change_comments(self, new_comments):
    """ (Review, str) -> NoneType

    Update the comments for this review to new_comments.

    >>> review = Review('Michael Smith', 'Too salty.', False)
    >>> review.change_comments('Please use less salt. :-) ')
    >>> review.comments
    'Please use less salt. :-) '
    """

    self.comments = new_comments
```

Part (c) [2 MARKS]

Write a `__str__` method in class `Review` to return a string representation of the `Review`.

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

    Return a string representation of this review.

    >>> review = Review('Susur Lee', 'Excellent!', True)
    >>> str(review)
    'recommended by Susur Lee: Excellent!'
    >>> review = Review('Mark McEwan', 'Room for improvement.', False)
    >>> str(review)
    'not recommended by Mark McEwan: Room for improvement.'
    """

    result = 'recommended by {0}: {1}'.format(self.reviewer, self.comments)
    if not self.recommend:
        result = 'not ' + result
    return result
```

Here is the header and two methods for class `Restaurant`. In part (d) and (e), you will write two additional methods for class `Restaurant`.

```
class Restaurant:
    """ Information about a particular restaurant including its name,
    price range, the types of cuisines it serves, and reviews."""

    def __init__(self, name, price_range, cuisine_list):
        """ (Restaurant, str, str, list of str) -> NoneType

        Record the restaurant's name name, price range price_range, and types of cuisines
        cuisines_list. There are initially no reviews of this restaurant.

        >>> rest = Restaurant('Dumplings R Us', '$$', ['Chinese', 'Japanese'])
        >>> rest.name
        'Dumplings R Us'
        >>> rest.price_range
        '$$'
        >>> rest.cuisine_list
        ['Chinese', 'Japanese']
        >>> rest.reviews
        []
        """

        # Assume this method body has been correctly implemented.


    def add_review(self, review):
        """ (Restaurant, Review) -> NoneType

        >>> rest = Restaurant('Mexican Grill', '$$$', ['Mexican'])
        >>> rest.reviews
        []
        >>> review = Review('Susur Lee', 'Excellent!', True)
        >>> rest.add_review(review)
        >>> str(rest.reviews[0])
        'recommended by Susur Lee: Excellent!'
        """

        # Assume this method body has been correctly implemented.
```

Part (d) [1 MARK]

Complete method `__eq__` in class `Restaurant`:

```
def __eq__(self, other):
    """ (Restaurant, Restaurant) -> bool

    Return whether this restaurant has the same name and price range
    as other.

    >>> r1 = Restaurant('Dumplings R Us', '$$', ['Chinese', 'Japanese'])
    >>> r2 = Restaurant('Dumplings R Us', '$$', ['Chinese', 'Japanese'])
    >>> r3 = Restaurant('Deep Fried Everything', '$', ['Canadian'])
    >>> r1 == r2
    True
    >>> r2 == r3
    False
    """

    return self.name == other.name and self.price_range == other.price_range
```

Part (e) [3 MARKS] Complete method `recommended_percentage` in class `Restaurant`:

```
def recommended_percentage(self):
    """ (Restaurant) -> number

    Precondition: this restaurant has at least one review

    Return the percentage of reviews that recommend this restaurant.

    >>> rest = Restaurant('Mexican Grill', '$$$', ['Mexican'])
    >>> rest.add_review(Review('Susur Lee', 'Excellent!', True))
    >>> rest.add_review(Review('Mark McEwan', 'Room for improvement.', False))
    >>> rest.recommended_percentage()
    50.0
    """

    positive = 0
    total = 0

    for review in self.reviews:
        if review.recommend:
            positive += 1
        total += 1

    return positive / total * 100
```

Question 10. [11 MARKS]

Here is a function that uses the *Bubble Sort* algorithm to sort the items in a list from smallest to largest:

```
def bubble_sort(L):
    """ (list) -> NoneType

    Sort the items of L from smallest to largest.
    """

    end = len(L) - 1

    while end != 0:
        # start of a pass of the Bubble Sort algorithm
        for i in range(end):                              # Line 1
            if L[i] > L[i + 1]:                           # Line 2
                L[i], L[i + 1] = L[i + 1], L[i]           # Line 3
        end = end - 1                                     # Line 4
        # end of a pass of the Bubble Sort algorithm
```

Part (a) [2 MARKS] One *pass* of the algorithm is executed in each iteration of the `while` loop. In the implementation of Bubble Sort shown above, a total of `len(L) - 1` passes of the algorithm will be executed. If the list passed to `bubble_sort` was almost sorted, it may become sorted after just a few passes. If the list is in sorted order at the start of a pass, which line of code in the body of `bubble_sort` will **not** be executed during the pass? Indicate your response by circling one of the line numbers below.

Line 1 Line 2 [Line 3] Line 4

Part (b) [4 MARKS] We can determine that the list has become sorted by keeping track of whether or not the line of code identified in Part (a) is executed. If that line of code is not executed, the list is sorted and no additional passes need to be executed.

Using the approach described above, revise the body of the `bubble_sort` function. In your revised version, once it has been determined that the list is in sorted order, no further passes should be executed. Do **not** add any new loops and do **not** use `return`, `break` or `continue`. HINT: Add exactly one new variable and add code to what is given below.

```
    end = len(L) - 1
    is_sorted = False # NEW
    while end != 0 and not is_sorted: # NEW (revised condition)

        # execute one pass of the Bubble Sort algorithm
        is_sorted = True # NEW
        for i in range(end):
            if L[i] > L[i + 1]:
                L[i], L[i + 1] = L[i + 1], L[i]
                if_sorted = False #NEW

        end = end - 1

    # Alternative solution
```

```
    end = len(L) - 1

    while end != 0:  # (Don't deduct mark for missing colon.)

        # execute one pass of the Bubble Sort algorithm
        shifted = 0 # NEW
        for i in range(end):
            if L[i] > L[i + 1]:
                L[i], L[i + 1] = L[i + 1], L[i]
                shifted += 1 # NEW

        end = end - 1 # or INDENT and add as else-clause below
        if shifted == 0:  #NEW
            end = 0       #NEW
```

Part (c) [1 MARK] Let `n` represent the length of the list `L`. For the original version of *Bubble Sort* from Part (a), in the **worst case**, the runtime of this function grows quadratically as `n` grows.

Which of the following most accurately describes how the runtime of Part (a) `bubble_sort` grows as `n` grows, in the **best case**? Circle one.

(a) It grows linearly, like n does. **(b) It grows quadratically, like n^2 does.**
(c) It grows less than linearly. (d) It grows more than quadratically.

Part (d) [1 MARK] Let `n` represent the length of the list `L`. For the revised version of bubble sort from Part (b), which of the following most accurately describes how the runtime of Part (b) `bubble_sort` grows as `n` grows, in the **worst case**? Circle one.

(a) It grows linearly, like n does. **(b) It grows quadratically, like n^2 does.**
(c) It grows less than linearly. (d) It grows more than quadratically.

Part (e) [1 MARK] Let `n` represent the length of the list `L`. For the revised version of bubble sort from Part (b), which of the following most accurately describes how the runtime of Part (b) `bubble_sort` grows as `n` grows, in the **best case**? Circle one.

(a) It grows linearly, like n does. (b) It grows quadratically, like n^2 does.
(c) It grows less than linearly. (d) It grows more than quadratically.

Part (f) [1 MARK]

Consider the following lists:

```
L1 = [9, 8, 7, 6, 5, 4, 3, 2, 1]        L2 = [8, 1, 3, 6, 9, 7, 4, 2, 5]
```

Which list would cause *Selection Sort* to do more **comparisons**?

Circle one.

`L1` `L2` **they would require an equal number**

Part (g) [1 MARK]

Consider the following lists:

```
L1 = [9, 8, 7, 6, 5, 4, 3, 2, 1]        L2 = [8, 1, 3, 6, 9, 7, 4, 2, 5]
```

Which list would cause *Insertion Sort* to do more **comparisons**?

Circle one.

`L1` `L2` they would require an equal 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.

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.
  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.
  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.
  open(name[, mode]) -> file open for reading, writing, or appending
    Open a file.  Legal modes are "r" (read), "w" (write), and "a" (append).
  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 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 ends in a 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() -> object
    Remove and return the last item from L.
  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.
```