PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

DECEMBER 2018 EXAMINATIONS

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

Duration — 3 hours

No Aids Allowed

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

Student Number: _____ **UTORid:** _____

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

---

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

This Final Examination paper consists of 10 questions on 21 pages (including this one), printed on both sides of the paper. *When you receive the signal to start, please make sure that your copy of the paper is complete and fill in your 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.
- As a student, you help create a fair and inclusive writing environment. If you possess an unauthorized aid during an exam, you may be charged with an academic offence.

Marking Guide

# 1:	_____/ 6
# 2:	_____/ 6
# 3:	_____/ 5
# 4:	_____/ 6
# 5:	_____/ 6
# 6:	_____/ 6
# 7:	_____/12
# 8:	_____/ 6
# 9:	_____/ 7
# 10:	_____/10
TOTAL:	_____/70

Question 1. [6 MARKS]

Part 1: Select the option on the right that best describes what happens when the code on the left is run.

```
L = [1, 1, 2, 2, 3, 4]
for item in L:
    if item % 2 == 1:
        item = item + 1
print(L)
```

(A) `[1, 1, 2, 2, 3, 4]` is printed
(B) `[2, 2, 2, 2, 4, 4]` is printed
(C) A different list is printed
(D) An error occurs when the code is run

Part 2: Select the option on the right that best describes what happens when the code on the left is run.

```
L = ['cat', 5, 'dog', 4]
f = open('output.txt', 'w')
for item in L:
    f.write(str(item))
f.close()
f = open('output.txt', 'r')
print(f.readlines())
f.close()
```

(A) `['cat\n', '5\n', 'dog\n', '4\n']` is printed
(B) `['cat, '5', 'dog', '4']` is printed
(C) `['cat5dog4']` is printed
(D) A different list is printed
(E) An error occurs when the code is run

Part 3: Select the option on the right that best describes what happens when the code on the left is run.

```
s = 'big orange cats are best'
result = False
for each in s.split():
    if each == 'cats':
        result = True
    else:
        result = False
if result:
    print('cats found')
else:
    print('no cats found')
```

(A) `cats found` is printed
(B) `no cats found` is printed
(C) Something else is printed
(D) An error occurs when the code is run

Part 4: Select the option below that best describes what happens when this code is run.

```
shoes = ['Saucony', 'Asics', 'Asics', 'NB', 'Saucony', 'Nike', 'Asics', 'Adidas', 'Saucony', 'Asics']
shoes_to_places = {}
for i in range(len(shoes)):
    shoes_to_places[shoes[i]].append(i + 1)
print(shoes_to_places)
```

(A) `{'Saucony': [1, 5, 9], 'Asics': [2, 3, 7, 10], 'NB': [4], 'Nike': [6], 'Adidas': [8]}` is printed
(B) `{'Saucony': [0, 4, 8], 'Asics': [1, 2, 6, 9], 'NB': [3], 'Nike': [5], 'Adidas': [7]}` is printed
(C) `{'Saucony': [9], 'Asics': [10], 'NB': [4], 'Nike': [6], 'Adidas': [8]}` is printed
(D) `{'Saucony': [1], 'Asics': [2], 'NB': [4], 'Nike': [6], 'Adidas': [8]}` is printed
(E) Something else is printed
(F) An error occurs when the code is run

Part 5:
Consider the following code.

```
D = {'cat': 5}
x = VALUE1
D[x] = VALUE2
```

Circle ALL of the options below that could used be in place of `VALUE1` so the code would run without error.

(A) `'dog'`
(B) `'cat'`
(C) `x`
(D) `['a', 'b']`
(E) `('a', 'b')`
(F) `{'a': 'b'}`
(G) `D['cat']`
(H) None of the above

Circle ALL of the options below that could used be in place of `VALUE2` so the code would run without error.

(A) `'dog'`
(B) `'cat'`
(C) `x`
(D) `['a', 'b']`
(E) `('a', 'b')`
(F) `{'a': 'b'}`
(G) `D['cat']`
(H) None of the above

1. A 2. C 3. B 4. F 5(a) A, B, E, G 5(b) all of A-G

Question 2. [6 MARKS]

A *board* is a list of lists of length 1 strings, where each inner list has the same length. Precondition for all three functions below: the first parameter represents a board with at least one row and column.

Part (a) [1 MARK] Complete the following function according to its docstring.

```
def get_row(board: List[List[str]], row_num: int) -> List[str]:
    """Return row row_num of board.

    >>> b = [['a', 'b', 'c'], ['d', 'e', 'f'], ['g', 'h', 'i']]
    >>> get_row(b, 0)
    ['a', 'b', 'c']
    >>> get_row(b, 2)
    ['g', 'h', 'i']
    """

    return board[row_num]
```

Part (b) [2 MARKS] Complete the following function according to its docstring.

```
def get_column(board: List[List[str]], column_num: int) -> List[str]:
    """Return column column_num of board.

    >>> b = [['a', 'b', 'c'], ['d', 'e', 'f'], ['g', 'h', 'i']]
    >>> get_column(b, 0)
    ['a', 'd', 'g']
    >>> get_column(b, 2)
    ['c', 'f', 'i']
    """
    col = []
    for row in board:
        col.append(row[column_num])
    return col
```

Part (c) [3 MARKS] Complete the following function according to its docstring. Call at least one of the functions from Part (a) or Part (b) as a helper function.

```
def get_rotated_board(original: List[List[str]]) -> List[List[str]]:
    """Return a new board that is the same as board original but with its rows and columns swapped.
    (Row 0 is swapped with column 0, row 1 is swapped with column 1, and so on.)

    >>> b = [['a', 'a', 'a', 'a'], ['b', 'b', 'b', 'b'], ['c', 'c', 'c', 'c']]
    >>> get_rotated_board(b)
    [['a', 'b', 'c'], ['a', 'b', 'c'], ['a', 'b', 'c'], ['a', 'b', 'c']]
    """

    new_board = []
    for i in range(len(original[0])):
        new_board.append(get_column(original, i))
    return new_board
```

Question 3. [5 MARKS]

A palindrome is a string that is read the same from front-to-back and back-to-front. For example, `noon` and `racecar` are both palindromes.

There are several ways to check whether a string is a palindrome. One algorithm compares the first letter to the last letter, the second letter to the second last letter, and so on, stopping when the middle of the string is reached or when a mismatch is found.

Part (a) [3 MARKS] Complete the function below using the algorithm described above. Do not modify code that is given outside of a box and do not add code outside of a box.

```
def is_palindrome(s: str) -> bool:
    """Return True if and only if s is a palindrome.

    >>> is_palindrome('noon')
    True
    >>> is_palindrome('racecar')
    True
    >>> is_palindrome('dented')
    False
    """

    offset = 0
    while i != len(s) // 2 and s[i] == s[i - 1 - offset]:
        i = i + 1
        offset = offset + 1

    Alternate solution:
    while i < len(s) // 2 and s[i] == s[len(s) - i - 1]:
       i = i + 1
```

Part (b) [1 MARK] For a string `s` of length `n`, where `n` is divisible by 2, how many times will the `while` loop in `is_palindrome` iterate in the *worst case*?

Solution: `n / 2`

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

(a) constant | linear | quadratic | something else

Question 4. [6 MARKS]

Complete the function below to according to its docstring.

```
def update_messages(msgs: List[str], lens: List[int]) -> None:
    """Update the list of the strings in msgs so their lengths are adjusted to
    match the value at the corresponding position in lens. Strings that are too
    long should be shortened by leaving off characters at the end and strings
    that are too short should be lengthened by adding underscores to the end.

    Precondition: len(msgs) == len(lens) and all values in lens are >= 0

    >>> messages = ['aardvark', 'bear', 'cat', 'dog']
    >>> update_messages(messages, [6, 5, 5, 3])
    >>> messages
    ['aardva', 'bear_', 'cat__', 'dog']
    >>> messages = ['', 'cat']
    >>> update_messages(messages, [3, 0])
    >>> messages
    ['___', '']
    """

    # Sample solution - other correct solutions are acceptable
    for i in range(len(msgs)):
        if len(msgs[i]) >= lens[i]:  # could be two cases
            msgs[i] = msgs[i][:lens[i]]
        else:
            padding = '_' * (lens[i] - len(msgs[i]))
            msgs[i] = msgs[i] + padding
```

Question 5. [6 MARKS]

Each of the functions below has a correct docstring, but one or more bugs in the code. In the table below each function, write two test cases. The first test case should not indicate a bug in the function (the test case would pass), while the second test case should indicate a bug (the test case would fail or an error would occur). If the code would stop due to an error on the second test case, write 'Error' in the **Actual Return Value** column for that test. Both test cases should pass on a correct implementation of the function.

```
def find_a_digit(s: str) -> int:
    """Return the index of the first digit in s, or the length of s if there are no digits.
    """
    i = 0
    while not s[i].isdigit():
        i = i + 1
    return i
```

Solution:

	Input	Actual Return Value	Expected Return Value
Passes	*any str with digit, eg.* 'a1b2'	*index of first digit, eg.* 1	*matches actual, eg.* 1
Fails	*any str without digit, eg.* 'abc'	An error occurs	*len of given str, eg.* 3

```
def has_a_fluffy(s: str) -> bool:
    """Return True if and only if s contains at least one character that is in 'fluffy'.
    """
    for ch in s:
        if ch in 'fluffy':
            return True
        else:
            return False
```

Solution:

Function	Input	Actual Return Value	Expected Return Value
Passes Example #1	*any str with first char fluffy, eg.* 'fox'	True	True
Passes Example #2	*any str no fluffy, eg.* 'cat'	False	False
Fails	*any str with fluffy char later, eg.* 'off'	False	True

```
def get_year_of_study(num_credits: float) -> int:
    """Return the year of study at UofT given the number of credits num_credits, using these rules:

    14.0 credits or more    4th year
    9.0 to 13.5 credits     3rd year
    4.0 to 8.5 credits      2nd year
    < 4.0 credits           1st year
    """
    if num_credits >= 4.0:
        return 2
    elif num_credits >= 9.0:
        return 3
    elif num_credits < 4.0:
        return 1
    else:
        return 4
```

Solution:

Function	Input	Actual Return Value	Expected Return Value
Passes	*anything < 9.0, eg.* 5.0	2	2
Fails	*anything >= to 9.0, eg.* 10.0	2	3

Question 6. [6 MARKS]

Consider this function:

```
def mystery(s: str) -> str:
    """ <docstring omitted> """
    i = 0
    result = ''
    while i < len(s) and not s[i] == '.':
        result = result + s[i]
        i = i + 1
    return result
```

Part (a) [1 MARK]

Give an example of an argument of length 4 that would produce the *best case* behaviour of this function.

`Solution: Any string of length 4 that begins with a .`

Part (b) [1 MARK]

How many assignment statements would be executed on one call to this function with an argument of length 4 that produces the best case?

`Solution: 2`

Part (c) [1 MARK]

Circle the term below that best describes the *best case* running time of this function on a length `n` string.

(a) [constant] linear quadratic something else

Part (d) [1 MARK]

Give an example of an argument of length 4 that would produce the *worst case* behaviour of this function.

`Solution: Any string of length 4 that does not contain a .`

Part (e) [1 MARK]

How many assignment statements would be executed on one call to this function with an argument of length 4 that produces the worst case?

`Solution: 10`

Part (f) [1 MARK]

Circle the term below that best describes the *worst case* running time of this function on a length `n` string.

(a) constant [linear] quadratic something else

Question 7. [12 MARKS]

Emotion analysis categorizes text as positive, negative, or neutral. For example, an emotion analysis program might categorize the text "happy birthday" as positive, and the text "it was disastrous" as negative.

Part (a) [4 MARKS]

Words are scored from -5 (most negative) to 5 (most positive). Word scoring data is stored in a comma separated values (CSV) file with a word, followed by its score, on each line.

Here is an example word scoring CSV file:

```
amazing,4
happy,3
anxious,-2
disastrous,-3
awesome,4
outstanding,5
woohoo,3
```

Each line in the file contains one word that contains only lowercase letters (no punctuation), followed by one comma, and then a valid score. There are no blank lines in the file. Each word has at least one character.

Neutral words, which would have a score of 0, are not included in the CSV file.

Given the example CSV file opened for reading, function `read_word_scores` should return:

```
{'amazing': 4, 'happy': 3, 'anxious': -2, 'disastrous': -3, 'awesome': 4,
'outstanding': 5, 'woohoo': 3}
```

Complete the function `read_word_scores` according to the example above and the docstring below. You may assume the argument file has the correct format.

```
def read_word_scores(f: TextIO) -> Dict[str, int]:
    """Return a dictionary that has words from f as keys and their
    corresponding scores as values.

    Precondition: words in f are unique
    """

    word_to_score = {}
    for line in f:
        tokens = line.strip().split(',')
        word_to_score[tokens[0]] = int(tokens[1])
    return word_to_score
```

Part (b) [4 MARKS] Complete the function below according to its docstring:

```
def score_document(text: str, word_to_score: Dict[str, int]) -> float:
    """Return the emotion analysis score for text.  The score is the average of the score for
    each word in text using word_to_score.  If a word from text does not appear in word_to_score,
    its score is 0.

    Precondition: text contains only words made up of only lowercase alphabetic characters
    (no punctuation), and each word is separated by a space. text contains at least one word.

    >>> word_to_score = {'amazing': 4, 'happy': 3, 'anxious': -2, \
    'disastrous': -3, 'awesome': 4, 'outstanding': 5, 'woohoo': 3}
    >>> score_document('everything is awesome', word_to_score)
    1.3333333333333333
    >>> score_document('outstanding and amazing', word_to_score)
    3.0
    >>> score_document('it is', word_to_score)
    0.0
    >>> score_document('i am happy but anxious', word_to_score)
    0.2
    """

    total_score = 0
    word_count = 0
    for word in text.split():
        if word in word_to_score:
            total_score = total_score + word_to_score[word]
        word_count = word_count + 1

    return total_score / word_count
```

Part (c) [4 MARKS] Complete the function below according to its docstring:

```
def find_similar_words(word: str, word_to_score: Dict[str, int]) -> List[str]:
    """Return a list of words from word_to_score that are similar to word. Two words
    are considered similar if they start with the same letter, have the same length,
    and have the same score in word_to_scores.  A word cannot be similar to itself.

    Precondition: word is lowercase and word is in word_to_score

    >>> word_to_score = {'amazing': 4, 'happy': 3, 'anxious': -2, \
    'disastrous': -3, 'awesome': 4, 'outstanding': 5, 'woohoo': 3}
    >>> find_similar_words('amazing', word_to_score)
    ['awesome']
    >>> find_similar_words('happy', word_to_score)
    []
    """

    score = word_to_score[word]
    similar_words = []
    for another_word in word_to_score:
        if word != another_word and score == word_to_score[another_word] \
           and another_word.startswith(word[0]) and len(word) == len(another_word):
            similar_words.append(another_word)
    return similar_words
```

Question 8. [6 MARKS]

Complete the function below to according to its docstring.

To receive full marks on this function, your solution must not remove any keys from the parameter `card_to_total`, even temporarily, such as by using the method `dict.clear()`.

```
def update_amounts(card_to_total: Dict[str, int], amts: List[Tuple[str, int]]) -> None:
    """Update card_to_total with the amounts for each card from amts.

    >>> D = {}
    >>> update_amounts(D, [('visa', 1000), ('amex', 50), ('visa', 500)])
    >>> D
    {'visa': 1500, 'amex': 50}
    >>> update_amounts(D, [('amex', 50), ('visa', 1000), ('mc', 100)])
    >>> D
    {'visa': 2500, 'amex': 100, 'mc': 100}
    >>> update_amounts(D, [('paypal', 25), ('amex', -100)])
    >>> D
    {'visa': 2500, 'amex': 0, 'mc': 100, 'paypal': 25}
    """

    for item in amts:
        if item[0] not in card_to_total:
            card_to_total[item[0]] = 0
        card_to_total[item[0]] = card_to_total[item[0]] + item[1]
```

Question 9. [7 MARKS]

Recall the algorithms insertion sort, selection sort, and bubble sort. Throughout this question, lists are to be sorted into ascending (increasing) order.

Part (a) [1 MARK]

Consider the following lists:

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

Which list would cause *Selection Sort* to do more **comparisons**? Circle one of the following.

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

Part (b) [1 MARK]

Consider the following lists:

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

Which list would cause *Insertion Sort* to do more **comparisons**? Circle one of the following.

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

Part (c) [5 MARKS] Consider the following lists. For each list, and each algorithm, consider whether *at least* two passes of the algorithm could have been completed on the list. If at least two passes could have been completed, check the box corresponding to that list and algorithm.

	Insertion?	**Selection?**	**Bubble?**
`[1, 2, 4, 6, 5, 7, 3]`	✓	✓	
`[1, 3, 4, 6, 5, 7, 2]`	✓		
`[1, 3, 4, 5, 2, 6, 7]`	✓		✓
`[5, 4, 3, 2, 1, 6, 7]`			✓
`[1, 2, 4, 5, 6, 7, 8]`	✓	✓	✓

Question 10. [10 MARKS]

In this question, you will complete some docstrings and write method bodies in classes to represent bridge and highway data.

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

```
class Bridge:
    """A bridge and its data"""
```

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

```
    def __init__(self, name: str, span_list: List[float]) -> None:
        """Initialize a bridge with name and the spans from span_list.

        >>> b = Bridge('CNR Subway', [13.1, 10.5, 11.2])
        >>> b.name
        'CNR Subway'
        >>> b.spans
        [13.1, 10.5, 11.2]
        """
         self.name = name
         self.spans = span_list
```

Part (b) [2 MARKS] Complete the body of this class `Bridge` method according to its docstring.

```
    def extend_bridge(self, new_span: float, at_start: bool) -> None:
        """Update this bridge to include a new span new_span at either the start or
        end of the list of spans, according to at_start.

        >>> b = Bridge('CNR Subway', [13.1, 10.5, 11.2])
        >>> b.extend_bridge(5.5, True)
        >>> b.spans
        [5.5, 13.1, 10.5, 11.2]
        >>> b.extend_bridge(8.8, False)
        >>> b.spans
        [5.5, 13.1, 10.5, 11.2, 8.8]
        """
         if at_start:
             self.spans.insert(0, new_span)
         else:
             self.spans.append(new_span)
```

Part (c) [1 MARK] Complete the body of the class `Bridge __eq__` method according to its docstring. Use the provided class `Bridge` helper method `get_length` in your solution. You can assume `get_length` has been implemented according to its docstring.

```
    def get_length(self) -> float:
        """Return the total length of the spans in this bridge.

        >>> b = Bridge('CNR Subway', [1.5, 1.5, 1.0])
        >>> b.get_length()
        4.0
        """
        # assume this method has been correctly implemented
```

```
    def __eq__(self, other: 'Bridge') -> bool:
        """Return True if and only if this bridge has the same name and length as other.

        >>> b1 = Bridge('CNR Subway', [1.5, 1.5, 1.0])
        >>> b2 = Bridge('CNR Subway', [1.0, 1.0, 2.0])
        >>> b1 == b2
        True
        >>> b3 = Bridge('CNR Subway', [1.0, 1.0, 2.5])
        >>> b1 == b3
        False
        """

         return self.name == other.name and \
                self.get_length() == other.get_length()
```

Now consider the class `Highway`. Here is the header and docstring for class `Highway`.

```
class Highway:
    """A highway that contains bridges and runs through regions."""
```

Part (d) [2 MARKS] Complete the body of this class `Highway` method according to its docstring.

```
    def __init__(self, hw_number: int, region: str) -> None:
        """Initialize a highway with highway number hw_number that begins in region. This new
        highway should have a length of 0 and no bridges.

        >>> h = Highway(400, 'Toronto')
        >>> h.number
        400
        >>> h.regions
        ['Toronto']
        >>> h.length
        0.0
        >>> h.bridges
        []
        """
        self.number = number
        self.regions = [region]
        self.length = 0.0
        self.bridges = []
```

Part (e) [2 MARKS] Complete the docstring of the class `Highway` method `add_bridge` according to its function body. You can assume that the method `extend_highway` has been implemented according to its docstring.

```
def extend_highway(self, extension_len: float, new_regions: List[str]) -> None:
    """Update this highway by increasing its length by extension_len, and appending any
    regions from new_regions that are not already in its regions list in the same order
    as they appear in new_regions.

    >>> h = Highway(400, 'Toronto')
    >>> h.extend_highway(50.5, ['Toronto', 'York'])
    >>> h.length
    50.5
    >>> h.regions
    ['Toronto', 'York']
    >>> h.extend_highway(20.1, ['Simcoe', 'York'])
    >>> h.length
    70.6
    >>> h.regions
    ['Toronto', 'York', 'Simcoe']
    """
    # assume this method has been correctly implemented


def add_bridge(self, new_bridge: 'Bridge', bridge_region: str) -> None:
    """Update this highway by adding new_bridge to it.

    >>> b = Bridge('CNR Subway', [1.1, 2.2, 3.3])
    >>> h = Highway(403, 'Oxford')
    >>> h.add_bridge(b, 'Brant')
    >>> h.length
```

[ ]

```
    >>> h.regions
```

[ ]

```
    """
    self.bridges.append(new_bridge)
    self.extend_highway(new_bridge.get_length(), [bridge_region])
```

Solution:

Box 1: `6.6`

Box 2: `['Oxford', 'Brant']`

Part (f) [2 MARKS] Complete the body of this class `Highway` method according to its docstring.

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

    >>> h = Highway(403, 'Oxford')
    >>> h.add_bridge(Bridge('CNR Subway', [1.1, 2.2, 3.3]), 'Brant')
    >>> h.add_bridge(Bridge('Thames River', [5.0, 5.0]), 'Oxford')
    >>> print(h)
    Highway 403 has 2 bridges with total length 16.6. The bridges are:
    CNR Subway
    Thames River
    """
    result = 'Highway {0} has {1} bridges with total length {2}. '.\
        format(self.number, len(self.bridges), self.length)
    result = result + 'The bridges are:'
    for bridge in self.bridges:
        result = result + '\n' + bridge.name
    return result
```

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

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__:
  abs(x: float) -> float
    Return the absolute value of x.
  float(x: object) -> float
    Convert x to a floating point number, if possible.
  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 input.
  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).
  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 from start (inclusive) to stop (exclusive) with step
    specifying the amount to increment (or decrement).  If start is not specified,
    the sequence starts at 0.  If step is not specified, the values are incremented by 1.
  str(x: object) -> str
    Convert an object into its string representation.
  type(x: object) -> the object's type
    Return the type of the object x.

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.
  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.
```

DO NOT DETACH THIS PAGE

```
S.find(sub: str[, start: int[, end: int]]) -> int
  Return the lowest index in S where substring sub is found, such that sub is
  contained within S[start:end], or -1 if sub is not found in S[start:end].
  Default value of start is 0 and end is len(S).
S.format([args, ...]) -> str
  Return a formatted version of S, using substitutions from args.
  The substitutions are identified by braces ('{' and '}').
S.index(sub: str[, start: int[, end: int]]) -> int
  Like S.find but raises an exception if sub does not occur in S[start:end].
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.
```