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

You must get 40% or above on this exam to pass the course (at least 28 out of 72); otherwise, your final course grade will be no higher than 47.

This final examination paper consists of 12 questions on 22 pages (including this one). When you receive the signal to start, please make sure that your copy of the final examination is complete.

- 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.

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

TOTAL: _____/72
### Question 1. [8 marks]

For each code fragment in the table below, indicate whether or not the code causes an error. If it runs without error, give the output. If it has an error, explain the cause of the error.

<table>
<thead>
<tr>
<th>Code</th>
<th>Error? (yes or no)</th>
<th>Output or Cause of Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>print(7 / 4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>print(7 // 4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>print(7 % 4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L = [1, 2, 3, 4] print(L[4])</td>
<td></td>
<td></td>
</tr>
<tr>
<td>words = ['ready', 'set', 'go'] print(words[3])</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s = 'time' s[0] = 'l' print(s[0])</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d = {2: ['free', 'and', 'easy'], 0: ['laid', 'back'], 1: ['happy', 'go', 'lucky']} print(d[1][-1][1:])</td>
<td></td>
<td></td>
</tr>
<tr>
<td>letters = ['a', 'b', 'c'] letters[0] = 321 print(letters)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Question 2. [4 marks]

In the box below, write what is printed when the following program is executed.

def triple(x):
    print('Tripling', x)
    return x * 3

def subtract(y, z):
    print('Subtracting', y, z)
    return y - z

if subtract(20, triple(10)) > 0 and subtract(-25, triple(5)) > 0:
    print('Woot!')
else:
    print('Yikes!', subtract(triple(3), 0))
Question 3.  [2 marks]

The following function does not always do what the docstring claims (although the type contract is correct).

```python
def filter_positives(L):
    """ (list of int) -> list of int

    Return a new list containing the elements of L that are positive (greater
    than 0).
    """

    acc = []
    for item in L:
        if item <= 0:
            return acc
        else:
            acc.append(item)

    return acc
```

In the table below, write the simplest possible test case that reveals the problem.

<table>
<thead>
<tr>
<th>Test Case Description</th>
<th>L</th>
<th>Expected Return Value According to Docstring</th>
<th>Actual Return Value Based on Code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Question 4. [4 marks]

Consider the following code.

```python
def do_something(a, b):
    a.insert(0, 'z')
    b = ['z'] + b

a = ['a', 'b', 'c']
a1 = a
a2 = a[:]

b = ['a', 'b', 'c']
b1 = b
b2 = b[:]

do_something(a, b)
```

Once this code has been executed, what are the values that the variables now refer to?

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
</tr>
<tr>
<td>a1</td>
<td></td>
</tr>
<tr>
<td>a2</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
</tr>
<tr>
<td>b1</td>
<td></td>
</tr>
<tr>
<td>b2</td>
<td></td>
</tr>
</tbody>
</table>
Question 5. [5 marks]

Some, but not all, of the lines containing a comment can be reached. All of these comment lines have this form, where \( N \) is a number: 

\[
\text{# Condition } N 
\]

In the table at the bottom of this page, complete each row by specifying one set of inputs to the function that will cause the comment line to be reached. If there are no possible inputs that would allow a particular comment to be reached, indicate this with 'X's in the corresponding table row cells. If the value of a variable is not relevant, write 'ANY' in the corresponding table row cell. We have filled in the first two rows for you so that you can see what we want.

```python
def mystery_code_paths(b1, b2, i, s):
    """ (bool, bool, int, str) -> NoneType ""

    if False:
        # Condition 0
    elif b1:
        # Condition 1

    if (b1 or b2) and i == 4:
        if i > 10:
            # Condition 2
        elif not b1:
            # Condition 3
        elif b1 and s == '':
            # Condition 4
        elif s == str(i):
            # Condition 5
    else:
        # Condition 6
```

<table>
<thead>
<tr>
<th>Condition</th>
<th>b1</th>
<th>b2</th>
<th>i</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>True</td>
<td>ANY</td>
<td>ANY</td>
<td>ANY</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Question 6. [5 marks]

Consider a dictionary where each key is a 3-element tuple containing the code for an airport, its city, and its status (domestic or international), and each value is the number of flights per day for that airport. For example:

\[
d = \{(\text{‘YYZ’}, \text{‘Toronto’}, \text{‘international’}): 1200, \\
         (\text{‘YTZ’}, \text{‘Toronto’}, \text{‘international’}): 250, \\
         (\text{‘YOW’}, \text{‘Ottawa’}, \text{‘international’}): 300, \\
         (\text{‘YYG’}, \text{‘Charlottetown’}, \text{‘domestic’}): 20\}
\]

The header for function `get_flights_by_category` appears below. It has two parameters:

- the first parameter, `airport_to_flights`, is a dictionary of the form described above, and
- the second parameter, `category`, is a string: one of ‘code’, ‘city’, or ‘status’.

`get_flights_by_category` returns a new dictionary based on `airport_to_flights` in which the keys are either the codes, cities, or statuses (as specified by `category`) and the values are the total number of flights associated with each key. Here is a sample shell interaction that uses variable \(d\), initialized above.

\[
>>> \text{get\_flights\_by\_category}(d, \text{‘city’})
\]

\{
\text{‘Toronto’}: 1450, \
\text{‘Charlottetown’}: 20, \
\text{‘Ottawa’}: 300
\}

Complete function `get_flights_by_category`. Use good style: avoid duplicate code.

```python
def get_flights_by_category(airport_to_flights, category):
    """ (dict of {tuple of (str, str, str): int}, str) -> dict of {str: int} """
```
Question 7. [10 marks]

The *median* value of a list of numbers is a number where half the values from the list are larger and half are smaller. If the list were sorted, the median value is:

1. The middle value (if there are an odd number of values), or
2. The average of the two middle values (if there are an even number of values).

Part (a) [5 marks]

Complete the following function. You *must* use this algorithm: call `sort` on the list and then return the middle value (or the average of the two middle values if the list has even length).

```python
def median1(L):
    """ (list of number) -> number

    Precondition: len(L) >= 1

    Return the median of the numbers in L.
    """

    >>> median1([4, 3, 1, 5, 7])
    4
    >>> median1([4, 3, 1, 5])
    3.5
    """

    # Start with a copy of the list so we don’t modify the original.
    L = L[:]
```
Part (b) [5 marks]

Complete the following function. You must use this algorithm: continually remove the largest and smallest values from the list and return either the last value (if there is only one left) or the average of the two last values (if there are two left).

Do not sort the list, and you must not use any for loops.

Functions max and min may be helpful, as well as one or more list methods.

def median2(L):
    """ (list of number) -> number

    Precondition: len(L) >= 1

    Return the median of the numbers in L.
    
    >>> median2([4, 3, 1, 5, 7])
    4
    >>> median2([4, 3, 1, 5])
    3.5
    """

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

    # Continually remove the largest and smallest values.
    while len(L) > 2:
        L.remove(max(L))
        L.remove(min(L))
    
    # If there's one left, return it.
    if len(L) == 1:
        return L[0]
    
    # Otherwise, return the average of the last two values.
    return (L[-1] + L[-2]) / 2

    """
Question 8.  [10 marks]

A coffee shop lets customers purchase coffee with pre-loaded electronic cards. They can reload their cards by specifying an amount of money to be added. Each card is also programmed with a default dollar amount that gets reloaded when the purchaser does not specify an amount.

Complete the Card class on this page and the next to match the comments. You may not change any of the provided code and your code must work correctly with it. Complete the main block, using the methods from the Card class. (Do not access the instance variables directly.)

Read all of starter code and comments on this page and the next before writing any code.

class Card:
    
    """A pre-paid coffee card."""
    # Write the constructor.

    def __str__(self):
        """ (Card) -> str
        Return a string representation of this object.
        """
        return "Owner: " + self.owner + " Balance: " + str(self.balance)

    def reload_default(self):
        """ (Card) -> NoneType
        Add the default dollar amount to this card’s balance.
        """
        self.reload(self.default_reload)

    # Write the reload method.
# Write the buy_coffee method, which has a parameter indicating the amount paid.
# If the card has enough money to cover the purchase, the amount is deducted.
# If the card does not have enough money to cover the purchase, the balance
# is unchanged. Return True iff the card had enough money to cover the purchase.

if __name__ == '__main__':
    ## Create a card for Karen with an initial balance of $100 and a
    ## default reload value of $50.
    card_for_karen = Card("Karen", 100, 50)

    ## Have Karen buy 25 coffees at a cost of $2.50 each.

    ## Create a card for Sven with an initial balance of $50 and a
    ## default reload value of $20.

    ## Have Sven buy a fancy coffee at a cost of $5.

    ## Reload Karen’s card with her default reload value.

    ## Have Sven put an additional $5.75 on his card.

    ## Print the owner and balance of both cards.
Question 9. [3 marks]

A matrix is a rectangular array of numbers. A square matrix is a matrix where the number of rows and columns is the same. The perimeter of a square matrix is all of the items on the outside edges. Here is an example of a 4 by 4 square matrix where the items on the perimeter are in bold:

\[
\begin{array}{cccc}
  a & b & c & d \\
  e & f & g & h \\
  i & j & k & l \\
  m & n & o & p \\
\end{array}
\]

Relative to the number of rows of parameter matrix, this function has quadratic running time:

```python
def perimeter(matrix):
    """ (list of list of str) -> (list of str)
    Precondition: The matrix is square: the length of each inner list in
    matrix is len(matrix).
    Return a sorted list of the strings that are in the outside
    e of matrix.
    >>> L = [['a', 'b', 'c', 'd'], ['e', 'f', 'g', 'h'],
           ['i', 'j', 'k', 'l'], ['m', 'n', 'o', 'p']]
    >>> perimeter(L)
    ['a', 'b', 'c', 'd', 'e', 'h', 'i', 'l', 'm', 'n', 'o', 'p']
    ""
    size = len(matrix)
    # Gather the external rows (the first and last rows).
    result = matrix[0] + matrix[size - 1]
    # Gather the internal rows.
    for i in range(1, size - 1):
        for j in range(size):
            if i == 0 or i == size - 1 or j == 0 or j == size - 1:
                result.append(matrix[i][j])
    result.sort()
    return result
```

One the next page, rewrite the body of the function so that it has the same functionality, but linear running time.
def perimeter(matrix):
    """ (list of list of str) -> (list of str)
    [Docstring is the same as before.]
    """

Question 10.  [9 marks]

On the next page is the header for function remove_greater_than, which takes a dictionary \( d \) of \{int: int\} as input, removes from that dictionary all the entries whose keys are greater than at least one of the values associated with the other keys in \( d \), and returns the number of entries that were removed.

Part (a)  [4 marks]

In the table below, we have outlined one test case for remove_greater_than. Add four more test cases chosen to test the function as thoroughly as possible.

<table>
<thead>
<tr>
<th>Test Case Description</th>
<th>Dictionary Before Call</th>
<th>Dictionary After Call</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>empty dictionary</td>
<td>{}</td>
<td>{}</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Part (b)  [5 MARKS] Write the function body.

def remove_greater_than(d):
    """ (dict of {int: int}) -> int

    Remove all entries of d whose keys are greater than at least one of the
    values associated with the other keys in d. Return the number of entries
    that were removed.
    """

**Question 11.** [10 marks]

*Word ladder* is a single-player game in which a player tries to link an initial word (e.g., *cold*) to a goal word (e.g., *warm*) by finding a chain of words to link the two. Words next to each other in the chain must differ by exactly one letter. For example: *cold* → *cord* → *card* → *ward* → *warm*.

This question has you write a word ladder game. We first have you write two helper functions, and in the last part you will write a main program.

**Part (a) [3 marks]** Write the body for function `differ_by_one`.

```python
def differ_by_one(word1, word2):
    """ (str, str) -> bool
    Return True iff word2 can be formed from word1 by changing exactly one letter.
    """
    >>> differ_by_one('cat', 'cot')
    True
    >>> differ_by_one('abc', 'aBc')
    True
    >>> differ_by_one('abc', 'abc')
    False
    >>> differ_by_one('abc', 'abcd')
    False
    """
```
Part (b) [2 MARKS] Write the body for function `get_word_list`.

```python
def get_word_list(word_file):
    """ (file open for reading) -> list of str

    word_file contains one word per line. Return a list that contains
    each word from word_file with leading and trailing whitespace removed.
    """
```

Part (c) [5 MARKS]

Complete the main program on the next page. Avoid duplicate code by calling function `differ_by_one` as a helper. You should do this even if your `differ_by_one` is incomplete/incorrect: when we mark this part, we will assume that function works correctly.

To play the game, the player repeatedly guesses the next word in the chain. If the word entered is not valid (it does not differ from the current word by exactly one letter or it does not appear in the file of words), the player is prompted to re-enter a word until a valid word is entered.

Once a valid word is entered, if it is the goal word, the game ends. If it is not the goal word, the player is prompted to make another move using the word they just formed as the current word. The player continues to make moves until the goal word is formed. The object of the game is to form the goal word in as few moves as possible, so the number of moves is tracked (only valid moves are counted).
print("Welcome to word ladder!")

initial = 'cold'
current = initial
goal = 'warm'

# one possible solution is cold -> cord -> card -> ward -> warm

num_moves = 0
word_file = open('words.txt')
word_list = get_word_list(word_file)
word_file.close()

while current != goal:
    print("Current: ", current, ", Goal: ", goal)
    next = input("Please enter the next word in the chain: ")

print("You solved the word ladder in", num_moves, "moves!")
Question 12. [2 marks]

Consider this code:

def g(L):
    """ (list of int) -> int """

    x = 0
    for i in range(len(L)):
        if i != L[i]:
            for j in range(len(L)):
                x = x + j
        else:
            x = x + 1

    return x

For each question below, a correct answer earns 1 mark, a blank answer earns 0 marks, and an incorrect answer earns -0.5 marks. Do not guess.

Part (a) [1 mark] Let \( n \) be the length of the list \( L \). In the best case, which of the following most accurately describes how the runtime of function \( g \) grows as \( n \) grows? Circle one.

(a) It grows linearly, like \( n \) does. (b) It grows quadratically, like \( n^2 \) does.
(c) It grows more than quadratically. (d) It does not grow.

Part (b) [1 mark] Let \( n \) be the length of the list \( L \). In the worst case, which of the following most accurately describes how the runtime of function \( g \) grows as \( n \) grows? Circle one.

(a) It grows linearly, like \( n \) does. (b) It grows quadratically, like \( n^2 \) does.
(c) It grows more than quadratically. (d) It does not grow.
[Use the space below for rough work. This page will not be marked, unless you clearly indicate the part of your work that you want us to mark.]
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`  
  With a single iterable argument, return its largest item.

* `max(a, b, c, ...) -> object`  
  With two or more arguments, return the largest argument.

* `min(iterable) -> object`  
  With a single iterable argument, return its smallest item.

* `min(a, b, c, ...) -> object`  
  With two or more arguments, return the smallest argument.

* `print(value, ..., sep=' ', end='
') -> 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.

* `F.seek() -> NoneType`  
  Set file position to offset in file, if possible.

* `F.tell() -> int`  
  Return current file position.

* `F.write(str) -> NoneType`  
  Write str to file.

* `F.writelines(list) -> NoneType`  
  Write list to file.
F.readlines() \rightarrow \text{list of str}

Return a list of the lines from the file. Each string ends in a newline.

\textbf{list:}

\begin{itemize}
  \item \texttt{x in L} \rightarrow \text{bool}
    \begin{itemize}
      \item Produce True if \( x \) is in \( L \) and False otherwise.
    \end{itemize}
  \item \texttt{L.append(\textit{x})} \rightarrow \text{NoneType}
    \begin{itemize}
      \item Append \( x \) to the end of the list \( L \).
    \end{itemize}
  \item \texttt{L.index(\textit{value})} \rightarrow \text{int}
    \begin{itemize}
      \item Return the lowest index of \( \textit{value} \) in \( L \).
    \end{itemize}
  \item \texttt{L.insert(index, \textit{x})} \rightarrow \text{NoneType}
    \begin{itemize}
      \item Insert \( x \) at position \( \text{index} \).
    \end{itemize}
  \item \texttt{L.pop()} \rightarrow \text{object}
    \begin{itemize}
      \item Remove and return the last item from \( L \).
    \end{itemize}
  \item \texttt{L.remove(\textit{value})} \rightarrow \text{NoneType}
    \begin{itemize}
      \item Remove the first occurrence of \( \textit{value} \) from \( L \).
    \end{itemize}
  \item \texttt{L.reverse()} \rightarrow \text{NoneType}
    \begin{itemize}
      \item Reverse \textsc{in place}.
    \end{itemize}
  \item \texttt{L.sort()} \rightarrow \text{NoneType}
    \begin{itemize}
      \item Sort the list in ascending order.
    \end{itemize}
\end{itemize}

\textbf{str:}

\begin{itemize}
  \item \texttt{x in s} \rightarrow \text{bool}
    \begin{itemize}
      \item Produce True if and only if \( x \) is in \( s \).
    \end{itemize}
  \item \texttt{str(\textit{x})} \rightarrow \text{str}
    \begin{itemize}
      \item Convert an object into its string representation, if possible.
    \end{itemize}
  \item \texttt{S.count(\textit{sub}[\[, \textit{start}[\[, \textit{end}\]\]]]} \rightarrow \text{int}
    \begin{itemize}
      \item Return the number of non-overlapping occurrences of substring \( \textit{sub} \) in string \( S[\textit{start}:\textit{end}] \). Optional arguments \( \textit{start} \) and \( \textit{end} \) are interpreted as in slice notation.
    \end{itemize}
  \item \texttt{S.find(\textit{sub}[\[, \textit{i}\]\]} \rightarrow \text{int}
    \begin{itemize}
      \item Return the lowest index in \( S \) (starting at \( S[\textit{i}] \), if \( \textit{i} \) is given) where the string \( \textit{sub} \) is found or \(-1\) if \( \textit{sub} \) does not occur in \( S \).
    \end{itemize}
  \item \texttt{S.index(\textit{sub})} \rightarrow \text{int}
    \begin{itemize}
      \item Like \texttt{find} but raises an exception if \( \textit{sub} \) does not occur in \( S \).
    \end{itemize}
  \item \texttt{S.isdigit()} \rightarrow \text{bool}
    \begin{itemize}
      \item Return True if all characters in \( S \) are digits and False otherwise.
    \end{itemize}
  \item \texttt{S.lower()} \rightarrow \text{str}
    \begin{itemize}
      \item Return a copy of the string \( S \) converted to lowercase.
    \end{itemize}
  \item \texttt{S.lstrip([\textit{chars}\])} \rightarrow \text{str}
    \begin{itemize}
      \item Return a copy of the string \( S \) with leading whitespace removed.
      \item If \( \textit{chars} \) is given and not \texttt{None}, remove characters in \( \textit{chars} \) instead.
    \end{itemize}
  \item \texttt{S.replace(\textit{old}, \textit{new})} \rightarrow \text{str}
    \begin{itemize}
      \item Return a copy of string \( S \) with all occurrences of the string \( \textit{old} \) replaced with the string \( \textit{new} \).
    \end{itemize}
  \item \texttt{S.rstrip([\textit{chars}\])} \rightarrow \text{str}
    \begin{itemize}
      \item Return a copy of the string \( S \) with trailing whitespace removed.
      \item If \( \textit{chars} \) is given and not \texttt{None}, remove characters in \( \textit{chars} \) instead.
    \end{itemize}
  \item \texttt{S.split([\textit{sep}\])} \rightarrow \text{list of str}
    \begin{itemize}
      \item Return a list of the words in \( S \), using string \( \textit{sep} \) as the separator and any whitespace string if \( \textit{sep} \) is not specified.
    \end{itemize}
  \item \texttt{S.strip()} \rightarrow \text{str}
    \begin{itemize}
      \item Return a copy of \( S \) with leading and trailing whitespace removed.
    \end{itemize}
  \item \texttt{S.upper()} \rightarrow \text{str}
    \begin{itemize}
      \item Return a copy of the string \( S \) converted to uppercase.
    \end{itemize}
\end{itemize}