PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

DECEMBER EXAMINATIONS

CSC 108H1F
Duration — 3 hours

Examination Aids: None

Student Number: |_|_|_|_|_|_|_|_|_|_|

Last (Family) Name(s): _____

First (Given) Name(s): _____

Instructor: (circle one) Campbell Gries

---

*Do **not** turn this page until you have received the signal to start.*
(In the meantime, please fill out the identification section above, and read the instructions below *carefully*.)

This final examination consists of 9 questions on 18 pages (including this one). *When you receive the signal to start, please make sure that your copy of the examination is complete.*

Comments and docstrings are not required except where indicated, although they may help us mark your answers. They may also get you part marks if you can't figure out how to write the code.

You do not need to put `import` statements in your answers.

You may not use `break` or `continue` on this exam.

If you use any space for rough work, indicate clearly what you want marked.

Assume all input is valid unless otherwise indicated; there is no need to error-check.

Good Luck!

MARKING GUIDE

1: _____/ 6
2: _____/ 6
3: _____/ 8
4: _____/ 8
5: _____/10
6: _____/16
7: _____/16
8: _____/10
9: _____/10

TOTAL: _____/90

CONT'D...

Question 1. [6 MARKS]

Write a function `warmest_day` that takes a list of `str` days and `float` temperatures in the format of the example below and returns the day with the highest temperature.

```
L = [‘Monday’, 15.4, ‘Tuesday’, 17.1, ‘Wednesday’, 16.9]
```

You may assume the list is non-empty. If the list contains two days with the warmest temperature, return either one.

```
def warmest_day(t_list):
    warmest_index = 1
    for i in range(3, len(t_list), 2):
        if t_list[warmest_index] < t_list[i]:
            warmest_index = i

    return t_list[warmest_index - 1]
```

Question 2. [6 MARKS]

In the table below, trace the variable values during execution of `mystery('ever', 'e')`. For each blank in the table, fill in the specified variable's value after the corresponding line has executed. Write "not reached" if that line was not executed. You do **not** need to complete the section of the table labeled '...'. You only need to complete the section for the last iteration if the loop iterates 3 or more times. In the last line, write the value returned by this function call.

```
def mystery(a, b):
  i = 0

  while a.find(b, i) >= 0:
    i = a.find(b, i)
    c = a[:i]
    d = a[i:]
    a = c + b + d
    i += 2
  return a
```

Show variable values after each line has executed:			
a: 'ever' b: 'e'			
i: 0			
During 1st iteration	During 2nd iteration	...	During last iteration
a: 'ever' b: 'e' i: 0	a: 'eever' b: 'e' i: 2		a: 'eeveer' b: 'e' i: 5
i: 0	i: 2		i:
c: ''	c: 'eev'		c:
d: 'ever'	d: 'er'		d:
a: 'eever'	a: 'eeveer'		i:
i: 2	i: 5		i:
value returned: 'eeveer'			

Question 3. [8 MARKS]

Write docstrings for the following functions. Include assumptions about the parameter values.

Part (a) [4 MARKS]

```
def enigma(L):
    '''

    '''Given a list of lowercase words, L, return the word that comes
    second in alphabetical order.  The list will contain at least two words.'''

    '''
    a = min(L)
    loc = L.index(a)
    L.remove(a)
    b = min(L)
    L.insert(loc, a)
    return b
```

Here is an example of a call: `enigma(['michael', 'jen', 'paul', 'andrew'])`.

Part (b) [4 MARKS]

```
def mystery(L, d):
    '''

    '''Return a dictionary where the keys are values in d and the values are items from L
    that occur as keys in d.'''

    '''
    res = {}
    for v in L:
        if v in d:
            res[d[v]] = v
    return res
```

Here is an example of a call: `mystery([3, 4, 5, 6, 7], {7:'a', 3:'b', 5:'c'})`.

Question 4. [8 MARKS]

Part (a) [4 MARKS] Complete the following function:

```
def find_keys(d, v):
    '''Return a list of keys in dictionary d that are associated with value v.'''

    values = []

    for key in d:
        if d[key] == v:
            values.append(v)

    return values
```

Part (b) [4 MARKS]

Fill in the rest of the table of test inputs and expected results below. We have filled in the first row, indicating that a call to `find_keys({}, 1)` should return the empty list.

Make sure that the test inputs you are proposing are different from each other. Also, don't test error cases; for example, *don't* provide invalid input such as an `int` instead of the expected dictionary.

d	v	**Expected result**
`{}`	`1`	`[]`

Question 5. [10 MARKS]

Part (a) [4 MARKS] Complete the following function:

```
def swap_blue(pic1, pic2):
  '''Swap the blue color values of all the pixels in pic1 with the blue color values of the
  corresponding pixels in pic2. Assume that pic1 and pic2 have the same width and height.'''
```

```
    pic1_pixels = get_pixels(pic1)
    pic2_pixels = get_pixels(pic2)

    for i in range(len(pic1_pixels)):
        temp = pic1_pixels[i].get_blue()
        pic1_pixels[i].set_blue(pic2_pixels[i].get_blue())
        pic2_pixels[i].set_blue(temp)
```

Part (b) [6 MARKS] Complete the following nose test for `swap_blue`:

```
def test_swap_blue():
  '''Test swap_blue using two small (2 x 2) pictures, one black and one blue. After
  swap_blue has been called on the two pictures, the black picture should have turned
  blue, and the blue picture should have turned black.'''
```

```
    pic1 = make_empty_picture(2, 2)
    set_pixels(pic1, blue)
    pic2 = make_empty_picture(2, 2)

    swap_blue(pic1, pic2)

    for x in range(2):
        for y in range(2):
            pic1_pixel = get_pixel(pic1, x, y)
            pic2_pixel = get_pixel(pic2, x, y)
            assert get_color(pic1_pixel) == black and get_color(pic2_pixel) == blue, \
                   "Failed to swap pixel colors correctly at %d, %d" % (x, y)
```

Question 6. [16 MARKS]

This question has you write a program to manage an inventory. The question has three parts.

Part (a) [4 MARKS] Complete the following function:

```
def read_inventory(f):
    '''Given an open file that contains lines of the form
        ITEM,COUNT,COST
    (where ITEM is a str, COUNT is an int, and COST is a float), return a dictionary in which
    the ITEMs are the keys and the values are (COUNT, COST) tuples. The ITEMs are unique.'''


    inv_dict = {}
    for line in f:
        line = line.strip().split(",")
        inv_dict[line[0]] = (int(line[1]), float(line[2]))
    return inv_dict
```

Part (b) [2 MARKS] Complete the following function, which operates on a dictionary like the one returned by `read_inventory`.

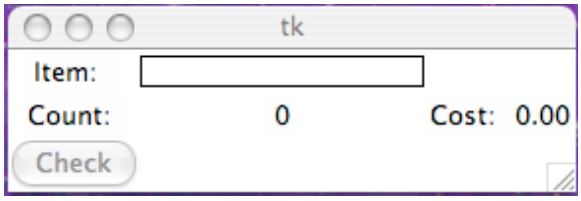
```
def set_inv_data(inv_dict, item, Tkvar_count, Tkvar_cost):
    '''inv_dict is a dictionary with strings as keys and tuples of int, float pairs as
    values.  Set the Tkvariables to the count and cost associated with item in inv_dict.

    Tkvar_count is an IntVar, and Tkvar_cost is a DoubleVar.'''


    Tkvar_count.set(inv_dict[item][0])
    Tkvar_cost.set("%.2f" % inv_dict[item][1])
```

Part (c) [10 MARKS]

Write a program that builds a GUI to present inventory management information. The inventory data that your program will need is in the file `inv_data.txt`; this file contains lines of the form `ITEM,COUNT,COST`. Your GUI should look like the example below.



When the user pushes the button `Check`, the count and cost associated with the item entered into the `Entry` widget should be displayed.

We have provided some starter code to import the modules you need and to define the window and frame. Make sure to fill in the code requested in the comments and to use the variable names defined.

```
from Tkinter import *
import inv_helpers          # Contains the helper functions from parts (a) and (b)

# Open the inventory file and make the inventory dictionary.
# ADD CODE BELOW

inv_dict = inv_helpers.read_inventory(open("inv_data.txt"))
\else
\vspace{1in}
\fi

\begin{verbatim}
# Create a Tkinter window and add the first label.
window = Tk()
frame = Frame(window)
frame.pack()

itemLabel = Label(frame, text="Item:  ")
itemLabel.grid(row=0, column=0)

# Add the text entry box to the frame.
# ADD CODE BELOW

entry = Entry(frame)
entry.grid(row=0, column=1)

countDescrip = Label(frame, text="Count: ")
countDescrip.grid(row=1, column=0)
countVar = IntVar()
countLabel = Label(frame, textvar=countVar)
```

```
countLabel.grid(row=1, column=1)
costDescrip = Label(frame, text="Cost: ")
costDescrip.grid(row=1, column=2)
costVar = StringVar()
costVar.set("0.00")
costLabel = Label(frame, textvar=costVar)
costLabel.grid(row=1, column=3)

enter_command = lambda : inv_helpers.set_inv_data(inv_dict, entry.get(), countVar, costVar)
check_button = Button(frame, text="Check", command=enter_command)
check_button.grid(row=2, column=0)

window.mainloop()
```

Question 7. [16 MARKS]

This question has you write a class called `vending_machine`. The question has three parts.

Part (a) [8 MARKS] Start defining class `vending_machine`; include a constructor and a `__str__` method.

The constructor takes a list of `(item, price, num_in_stock)` tuples where item is a `str`, price is a `float`, and `num_in_stock` is an `int`.

Method `__str__` returns a string representing the current inventory of the machine. For example, given the inventory `[("soda", 1.0, 10), ("chips", 0.7, 6), ("crackers", 0.25, 3)]`, the return value is:

```
"soda $1.00 (10)\nchips $0.70 (6)\ncrackers $0.25 (3)\n"
```

```
class vending_machine(object):
    def __init__(self, inv_list):
        self.inventory = inv_list

    def __str__(self):
        inv_str = ""
        for item in self.inventory:
            inv_str += ...
```

Part (b) [3 MARKS] Complete the following helper method for `vending_machine`.

```
    def _find_item(self, item_name):
        '''item_name is a str that represents an item sold by the vending_machine; return the
        index of the tuple in the inventory list whose first element is item_name.'''
```

NO SOLUTION YET

Part (c) [5 MARKS]

Define three methods for the `vending_machine` class. Each method takes `self` and a `str` as parameters. You should call method `_find_item` from part (b).

- `price`: Return the price of the given item as a `float`.
- `has_item`: Return `True` if the requested item is in stock (`num_in_stock` is greater than zero) and `False` otherwise.
- `vend`: If the given item is in stock, decrement the number in stock and return `True`. If the item is not in stock, return `False`.

For all three items, you may assume that the user will only ask for items that exist in the inventory list.

```
NO SOLUTION YET
```

Question 8. [10 MARKS]

Complete the following functions.

Part (a) [6 MARKS]

```
def encrypt_character(c, mapping=[]):
  '''Return the encrypted version of lowercase character c (a string).

  If the mapping is not specified, the character's encrypted value is the next character in the
  alphabet: 'a' is encrypted as 'b', 'b' as 'c', and so on. 'z' is encrypted as 'a'.

  If the mapping is specified, it is used for the encryption: it is a list of lists where each
  element is a 2-item list containing a character and its encrypted version (both of type str).
  This list will contain c as a character.'''


    c_encrypt = ''

    if mapping == []:
        if c == 'z':
            c_encrypt = 'a'
        else:
            c_encrypt = chr(ord('a') + 1)
    else:
        for item in mapping:
            if item[0] == c:
                c_encrypt = item[1]

    return c_encrypt
```

Part (b) [4 MARKS]

```
def encrypt(f, mapping=[]):
  '''Given an open file f that contains a text message, return the encrypted version of that
  message (as a string).

  If the mapping is not specified, the character's encrypted value is the next character in the
  alphabet: 'a' is encrypted as 'b', 'b' as 'c', and so on. 'z' is encrypted as 'a'.

  If the mapping is specified, it is used for the encryption: it is a list of lists where each
  element is a 2-item list containing a character and its encrypted version (both of type str).
  The mapping will contain all of the characters in the message.

  The message will contain only lowercase characters, newlines, and spaces, and the mapping will
  contain only characters.  In the encrypted version, spaces and newlines remain unchanged.'''

    encrypted_message = ''

    for line in f:
        for character in line:
            if character not in [' ', '\n']:
                character = encrypt_character(character, mapping)
            encrypted_message += character

    return encrypted_message
```

Question 9. [10 MARKS]

The Dutch national flag is divided into three horizontal areas:

Someone has given you a picture of the flag with the columns scrambled: in each column, the reds, whites, and blues have all been mixed up. (No pixels have been moved left or right; only up and down.) The program on the opposite page fixes the picture—but the function `fix_column` is not yet written.

For example, if you are given the picture on the left below (where "R" means a red pixel, "W" a white, and "B" a blue), then the program will fix the picture so that it looks like the picture on the right.

```
R W B R R B          R R R R R R
W R R B R B          R R R R R R
B R W R W W    ->    W W W W W W
B B R B B R          B B B B B B
R B B W B R          B B B B B B
```

Your task is to write the code for `fix_column` on the next page.

You can't remember if the flag's bands of colour have the same height, so you can't just make the top third red.

You have five options for answering. Indicate clearly which one you are attempting by circling the option below.

- You can leave it blank. We will give you 0 marks.
- You can write "I don't know how to solve this question." We will give you 2 marks!
- You can solve it by counting colours in a column and setting the pixel values according to those counts for a maximum of 5 marks.
- You can solve it using a general sorting algorithm such as insertion sort for a maximum of 7 marks.
- You can solve it in linear time, using `get_color` on each pixel exactly once (not counting its use in `swap_color`), for the full 10 marks.

For the last three options, put the appropriate code into `fix_row` on the next page.

```
from picture import *

def swap_color(pic, col, i, j):
    '''Swap the colours of the two pixels in pic at locations (col, i) and (col, j).'''
    p1 = get_pixel(pic, col, i)
    p2 = get_pixel(pic, col, j)
    c = get_color(p1)
    set_color(p1, get_color(p2))
    set_color(p2, c)

def fix_column(pic, col):
    '''Fix column col of pixels in picture pic so that the red pixels are at the top,
    the white ones in the middle, and the blue ones at the bottom.'''

    # list[0 .. i - 1] is red
    # list[i+1 .. j - 1] is white
    # list[k + 1 .. len(list) - 1] is blue
    i, j, k = 0, 0, get_height(pic) - 1
    while j != k + 1:
        px = get_pixel(pic, col, j)
        c = get_color(px)

        if c == red:
            swap_color(pic, col, j, i)
            i += 1
            j += 1
        elif c == white:
            j += 1
        else: # blue
            swap_color(pic, col, j, k)
            k -= 1

def fix_flag(pic):
    '''Sort each column of pixels in pic so that the red pixels are at the top, the white
    ones in the middle, and the blue ones at the bottom.'''
    for col in range(get_width(pic)):
        fix_column(pic, col)
```

Short Python function/method descriptions:

```
__builtins__:
  chr(i) -> character
    Return a string of one character with ordinal i; 0 <= i < 256.
  len(x) -> integer
    Return the length of the list or string x.
  max(L) -> value
    Return the largest value in L.
  min(L) -> value
    Return the smallest value in L.
  open(name[, mode]) -> file object
    Open a file.
  ord(c) -> integer
    Return the integer ordinal of a one-character string.
  range([start], stop, [step]) -> list of integers
    Return a list containing the integers starting with stop 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.
  round(number[, ndigits]) -> floating point number
    Round a number to a given precision in decimal digits (default 0 digits).  This always
    returns a floating point number.
Color:
  black
    RGB: 0, 0, 0
  blue
    RGB: 0, 0, 255
  green
    RGB: 0, 128, 0
  red
    RGB: 255, 0, 0
  white
    RGB: 255, 255, 255
dict:
  D.get(k) --> value
    Return the value associated with the key k in D.
  D.has_key(k) --> boolean
    Return True if k is a key in D and False otherwise.
  D.keys() --> list of keys
    Return the keys of D.
  D.values() --> list of values
    Return the values associated with the keys of D.
file:
  F.close()
    Close the file.
  F.read([size]) -> read at most size bytes, returned as a string.
    If the size argument is negative or omitted, read until EOF is reached.
  F.readline([size]) -> next line from the file, as a string. Retain newline.
    A non-negative size argument limits the maximum number of bytes to return (an incomplete
    line may be returned then).  Return an empty string at EOF.
float:
  float(x) -> floating point number
    Convert a string or number to a floating point number, if possible.
```

```
int:
  int(x) -> integer
    Convert a string or number to an integer, if possible.  A floating point
    argument will be truncated towards zero.
list:
  L.append(x)
    Append x to the end of the list L.
  L.index(value) -> integer
    Returns the lowest index of value in L.
  L.insert(index, x)
    Insert x at position index.
  L.remove(value)
    Removes the first occurrence of value from L.
  L.sort()
    Sorts the list in ascending order.
Picture:
  get_blue(Pixel) --> int, get_red(Pixel) --> int, get_green(Pixel) --> int
    Return the value of the specified color (between 0 and 255) in the given pixel.
  get_color(Pixel) -> Color
    Return the color of the pixel.
  get_height(Picture) --> int
    Takes a picture as input and returns how many pixels high it is.
  get_width(picture) --> int
    Takes a Picture as input and returns how many pixels wide it is.
  get_pixel(Picture, x, y) --> Pixel
    Return the pixel at the location (x, y) in the given picture.
  get_pixels(Picture) --> list
    Takes a picture as input and returns the sequence of pixel objects in the picture.
  make_empty_picture(width, height) --> Picture
    Return a blank (black) picture of the given dimensions.
  make_picture(filename) --> Picture
    Create and return a picture from the contents of filename.
  pick_a_file() --> string
    Launch a file chooser and return a string containing the name of the file that was selected.
  set_blue(Pixel, value), set_green(Pixel, value), set_red(Pixel, value)
    Set the specified color component in the given pixel to the given value.
  set_color(Pixel, color)
    Set the pixel to the color.
  show(Picture)
    Displays the picture.
str:
  str(x) -> string
    Convert an object into its string representation, if possible.
  S.find(sub[,i]) -> integer
    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) -> integer
    Like find but raises an exception if sub does not occur in S.
  S.isdigit() --> boolean
    Return True if all characters in S are digits and False otherwise.
  S.replace(old, new) --> string
    Return a copy of string S with all occurrences of the string old replaced
    with the string new.
```

```
  S.rstrip([chars]) -> string
    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 strings
    Return a list of the words in S, using string sep as the separator and
    any whitespace string if sep is not specified.
  S.strip() --> string
    Return a copy of S with leading and trailing whitespace removed.
Tkinter:
  Button:
    Button(parent, [text=],[textvariable=], [command=]) --> button object
      Construct a button with the given parent.
  DoubleVar:
    DoubleVar() -> Tk float variable object
      Construct a Tkinter float variable.
    D.get() -> float
      Return the value of D.
    D.set(value)
      Set D to value.
  Entry:
    Entry(parent) --> entry object
      Construct an entry field with the given parent.
    E.delete(first, last)
      Delete text from first to last (non-inclusive).
      (Common first and last positions are 0 and END, respectively.)
    E.get() qw--> string
      Return the text in the entry field E.
  Frame:
    Frame(parent) --> frame object
      Construct a frame widget with the given parent.
  IntVar:
    IntVar() -> Tk int variable object
      Construct a Tk int variable.
    I.get() -> int
      Return the value of I.
    I.set(value)
      Set I to value.
  StringVar:
    StringVar() --> Tk string variable object
      Construct a Tk string variable.
    S.get() --> string
      Return value of S.
    S.set(value)
      Set S to value.
  Window:
    Tk() --> window
      Return a new window widget.
  All Widgets:
    A.grid(row=r, column=c)
      Place widget A in row r and column c.
    A.pack()
      Place widget A below the last widget that was packed.
```