CMPUT 275 - Introduction to Tangible Computing II
Final Exam - Winter 2015
Department of Computing Science, University of Alberta
Instructors: Zachary Friggstad, Leah Hackman & Csaba Szepesvári

Last Name: _____

ID: _____

Instructions

- Print your last name and ID clearly above.
- You have 180 minutes to complete the final, although it was designed to be completed 120 minutes.
- There should be 8 questions and 13 pages in this exam booklet. You are responsible for checking that your exam booklet is complete.
- This is an **OPEN BOOK** exam. You may use your notes, laptop, Arduino, or an internet search. You may not directly query anyone (through email, IM, text message, etc.) for answers: all answers must be your own. As usual, you should document any and all sources used.
- Place most answers in the spaces provided on the question pages. The answers to all questions are straight forward and brief. Written answers should fit easily into the space provided. Code answers will be submitted via eClass, but they should not require overly complex programming. Think about each question a bit before answering it.
- This exam counts 30% toward your final grade in this course. This exam is worth 100 points. The weight of each question is indicated in square brackets by the question number.

Question	1	2	3	4	5	6	7	8	Total
Out of	10	12	12	18	10	10	16	14	102

Question 1 [10 marks]: Consider the following function.

```
def foo(bar):
    c = []
    m = [None, 0]
    for item in bar:
        f = False
        for i in c:
            if i[0] == item:
                i[1] = i[1] + 1
                if( i[1] > m[1]):
                    m = i
                f = True
                break
        if not f:
            c.append([item, 1])
            if 1 > m[1]:
                m = [item,1]
    return m
```

Part 1.1 [1 marks]:: Given the list:

```
nums = [5, 1, 6, 3, 2, 1, 3, 5, 3]
```

what is the output of `foo(nums)`?

Part 1.2 [1 marks]:: Given the list:

```
strs = ["Hello", "World", "What", "A", "Wonderful", "World"]
```

what is the output of `foo(strs)`?

Part 1.3 [1 marks]:: Describe in *one sentence* what this function does.

Part 1.4 [7 marks]:: Refactor this function and write out your new version in the space provided below. Remember that when refactoring it is important to consider the readability, ease of understanding, and efficiency of code.

Question 2 [12 marks]: Quick Coding. Each question below should be able to be solved in one or two lines of code. Write your code in the space below each subquestion. Note: an *expression* is a single line of python which can be evaluated to get a value; when asked for an expression you should not be defining a *function*.

Part 2.1 [3 marks]: You are provided with a variable `nums` which contains integers, and a variable `my_num` which contains an integer. Generate a list of booleans of the same length as nums, in which the ith value is `True` if and only if the ith value of nums is greater than my_num. Otherwise, the ith value is `False`.

Part 2.2 [3 marks]: Write an *expression* that evaluates to a list of all possible dice rolls using 2 six-sided die (with sides number 1,2,3,4,5,6), such that both dice are either even or odd. Each entry in the list should be a tuple of length 2 whose elements are the values on the dice. In this case, (3,1), (1,3), (2,4) should all be on the list as they are dice rolls where the two dice are both odd or even. Note that (2,1) should not be in the list as one dice is odd and one dice is even. You will not be given full marks if you write out all possible dice rolls explicitly.

Part 2.3 [3 marks]: Suppose you are representing grocery items as a tuple (*item_name, item_cost, item_weight*). For example, ("Cheddar Cheese", 8.99, 600) is a tuple that says cheese is on sale this week for $8.99 for 600 grams of cheese. You are given a variable `groceries` that is a list of grocery tuples. Write an *expression* that finds the grocery item which has the lowest cost per gram.

Part 2.4 [3 marks]: Write a short *function* named `mean` that can take any non-zero number of arguments, each arguement being a number, and returns their arithmetic mean (i.e. the average) of the arguements.

Question 3 [12 marks]: Indicate whether each statement is `True` of `False`. Write the whole word.

Part 3.1 [2 marks]: $n^2 = \Theta(n^3)$

Part 3.2 [2 marks]: $5(n+1) = \Omega(n)$

Part 3.3 [2 marks]: $\frac{n^3}{100} + 5n^2 = \Theta(n^3)$

Part 3.4 [2 marks]: $n \log_2 n = \Omega(n)$

Part 3.5 [2 marks]: $n \log_2 n = O(n)$

Part 3.6 [2 marks]: $10^{100} = \Theta(1)$

Question 4 [18 marks]:

Pascal's triangle is a table of numbers where an entry in the table is equal to the sum of the entry directly above and the entry directly to the left. The first row and column consist entirely of 1s. Here is a portion of Pascal's triangle.

1	1	1	1	1	1	1	1
1	2	3	4	5	6	7	
1	3	6	10	15	21		
1	4	10	20	35			
1	5	15	35				
1	6	21					
1	7						
1							

More explicitly, the entry in row i and column j is calculated as follows:

$$T(i,j) = \begin{cases} 1 & \text{if i} = 0 \text{ or j} = 0 \\ T(i-1,j) + T(i,j-1) & \text{otherwise} \end{cases}$$

Part 4.1 [12 marks]: Write a python function `pascal_triangle` that takes two non-negative arguments, i and j, and returns the value $T(i,j)$ using *memoization*.

Part 4.2 [6 marks]: What is the running time of computing a value $T(i,j)$ using the memoized implementation of `pascal_triangle`?

Question 5 [10 marks]:

We commonly use infix notation when we write mathematical equations – meaning we write equations with the operator in the middle of the arguments like so:

```
(1 + 2) * (3 + 4)
```

But, we could also choose to write math equations in prefix notation, where the operator comes before both arguments. So our previous example becomes:

```
* (+ 1 2) (+ 3 4)
```

We can store this prefix notation using nested lists in Python like so:

```
[ '*', [ '+', [1], [2] ], [ '+', [3], [4] ] ]
```

Write a function `exp_eval` which takes in an expression in this nested list format and returns the value you get by evaluating this expression. So in the case of

```
exp_eval([ '*', [ '+', [1], [2] ], [ '+', [3], [4] ] ])
```

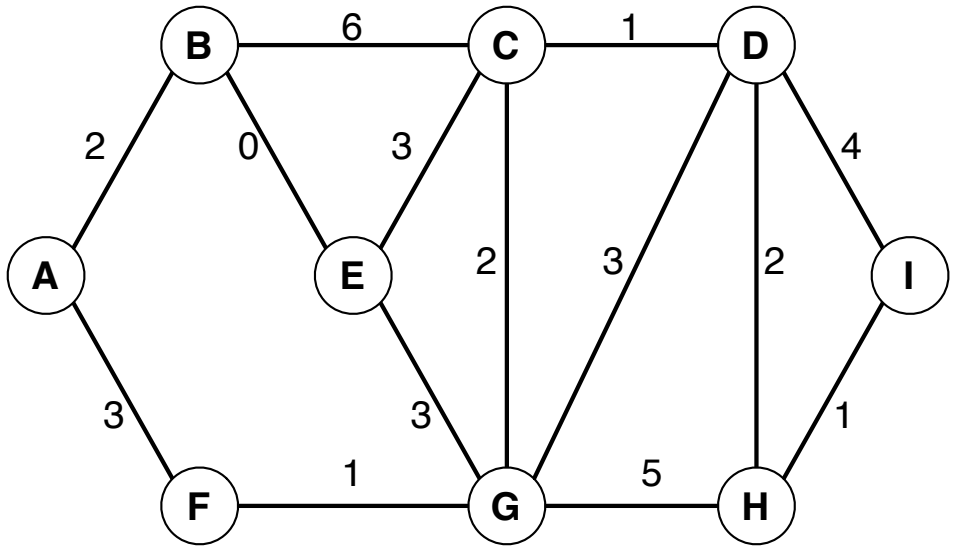
the returned value should be `21`. Write `doctests` for this function to demonstrate its behaviours and test the correctness of your implementation.

The allowed operators are `'+'` (addition), `'-'` (subtraction), `'*'` (multiplication), `'/'` (division). The numbers can be either floating point numbers, or integers.

The method should raise `ValueError` with a meaningful error message when the input is not of the expected format. Leave the handling of divide by zero errors to Python.

Submit your code using eclass.

Question 6 [10 marks]: Consider the following undirected graph. The number beside each edge indicates the length of the edge.



We assume that the neighbours of a vertex are listed in alphabetic order. For example, the neighbours of **E** are [**B**, **C**, **G**].

Ignore the edge weights in the first two questions below.

Part 6.1 [3]: Consider a depth-first search traversal of the graph starting from vertex **A**. What order would the vertices be first removed from the stack? When adding the neighbours of a vertex to the stack, add them in alphabetic order.

Part 6.2 [3 marks]: Now consider a breadth-first search traversal of the graph starting from vertex **A**. What order would the vertices be first removed from the queue? Again, when adding the neighbours of a vertex to the queue, consider them in alphabetic order.

Part 6.3 [4 marks]: Now run Dijkstra's algorithm starting from vertex **A**. List the nodes in order of their first removal from the heap. Also write the length of the shortest path from **A** to every other node.

Question 7 [16 marks total]: You are given a list of tuples. The length of this list is n. Each tuple has exactly two elements. The list is ordered such that the first elements of the tuples are increasing along the list, and the second elements of the tuples are decreasing along the list. For example,

```
example_list=[(2,9), (2, 7), (6, 6), (8, 2)]
```

is an example of a valid list, as all the first tuple elements are increasing in the list, and all the second tuple elements are decreasing in the list. Write a function `find_intersection` to determine if there is a tuple of two equal elements (ie: where `tuple[0] == tuple[1]`). If such a tuple exists, return the index `i` for that tuple in the list. If there are multiple tuples with equal elements, return the index of any of the valid tuples. If no such tuple exists, return `None`. For the example list provided above, you would return `2`. Your code should run in $O(\log(n))$ time. Submit your code, including doctests, in a file named `"equal tuple.py"` via eclass. Nothing on this page will be marked.

Question 8 [14 marks]:

Part 8.1 [3 marks]: You are given elements `2, 3, 7, 14, 23, 29, 36`. Draw a binary search tree containing those elements with a *height* of 2.

Part 8.2 [2 marks]: What order would you insert the elements above in order to construct the binary search tree you drew above? (Note: assume you're using a plain binary search tree implementation, not a self-balancing tree like AVL trees).

Order: _____ _____ _____ _____ _____ _____ _____

Part 8.3 [1 marks]: What order would you insert the elements in order to construct a binary search tree with a *height* of 6?

Order: _____ _____ _____ _____ _____ _____ _____

Part 8.4 [4 marks]: Describe an algorithm for finding the predecessor of a node in a binary search tree (the predecessor of a node is the largest node in the tree that is smaller than your given node).

Part 8.5 [2 marks]: What is the run time of your algorithm for finding the predecessor?