

CIS 211

Winter 2020 Final Exam

This exam contains four programming problems. For each programming problem, there is an accompanying file of starter code in this same archive. There is also a “quiz” question in Canvas corresponding to each problem. You should edit the the starter code and test it, then upload the edited file to the corresponding quiz question.

Beware the misleading Canvas timer. Our exam period is 8am to 10am unless you have previously made arrangements with AEC for extended test time. I will give a few minutes leeway for file upload. You must keep track of time for yourself.

This exam is not designed to test how fast you can type or how fast you can reproduce memorized examples. Do keep track of your time, but don't rush. If you get stuck on a problem, go on to the next one and come back.

1. [20 points]

The starter code for this problem is `q1_chooser.py`.

A `Chooser` is an object that selects one item from a non-empty list of integers. `Chooser` is an abstract base class.

Two concrete subclasses of `chooser` are provided for you in the starter code:

- `First`, which always choose the first item in the list, and
- `Last`, which always chooses the last item in the list.

You must finish two additional `Chooser` classes. `Maximizer` should choose the largest item in a list. `Minimizer` should choose the smallest item in a list.

The only method you should override is `_choose`. Do not write unnecessary methods. Your code should be quite short.

2. [20 points]

The starter code for this problem is `q2_rodents.py`.

There are three dangers in the fire swamps: flame spurts, lightning sand, and rodents of unusual size (R.O.U.S.). Given a list of rodents (objects of class `Rodent`), function `r_o_u_s` selects those who are unusually large. We say that a rodent is *unusually* large if its weight is at least twice the average weight of rodents in the list.

For example, suppose we have a rat Robby, who weight 12 grams, a rat Randy, who weights 4 grams, and a rat Morgan, who weighs 8 grams. The average weight of Robby, Randy, and Morgan is $12 + 8 + 4 = 24$, and their average weight is 8 grams. None of them weighs at least 16 grams (twice the average of 8 grams), so `r_o_u_s([Robby, Randy, Morgan])` would return `[]`, the empty list.



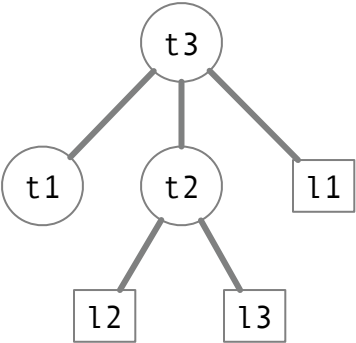
Suppose Robby still weighs 12 grams but Randy still weights 3 grams, and Morgan also weights 3 grams. Now their average weight is 6 grams, and 12 grams is $2 * 6$ grams, so `r_o_u_s([Robby, Randy, Morgan])` would return `[Robby]`.

You must write `r_o_u_s`. It must run in linear time, so I recommend a two-pass algorithm.

3. [20 points]

The starter code for this problem is `q3_tree_size.py`.

We define the *size* of a tree to be the number of nodes in the tree, including both inner nodes and leaf nodes.

Tree	Size	Note
	1	Inner node, no children
	1	Leaf node
	6	Typical tree

Define method `size` in classes `Inner` and `Leaf` to return the correct size.

4. [20 points]

The starter code for this problem is `q4_all_column_increasing.py`.

Function `all_col_increasing` returns True if and only if each column of a rectangular grid is *strictly increasing*. Strictly increasing means that each element is greater than the element before (if any).

We assume the grid passed to `all_col_increasing` is rectangular (each row of the grid has the same number of elements), but it may not be square (the number of rows might not be the same as the number of columns). The number of rows or the number of columns could be zero.

All columns are strictly increasing in these grids:

1	2
---	---

3	2	1
4	3	2
5	4	3

-20	-30	-10
-18	-28	-8
-5	-2	-5

Not all columns are strictly increasing in these grids:

5	10	20
6	7	21
7	8	10

5	6	7	8
6	7	8	8

5	6	7
6	7	8
7	8	9
8	9	8

You must finish function `all_col_increasing`. Do not create additional lists.