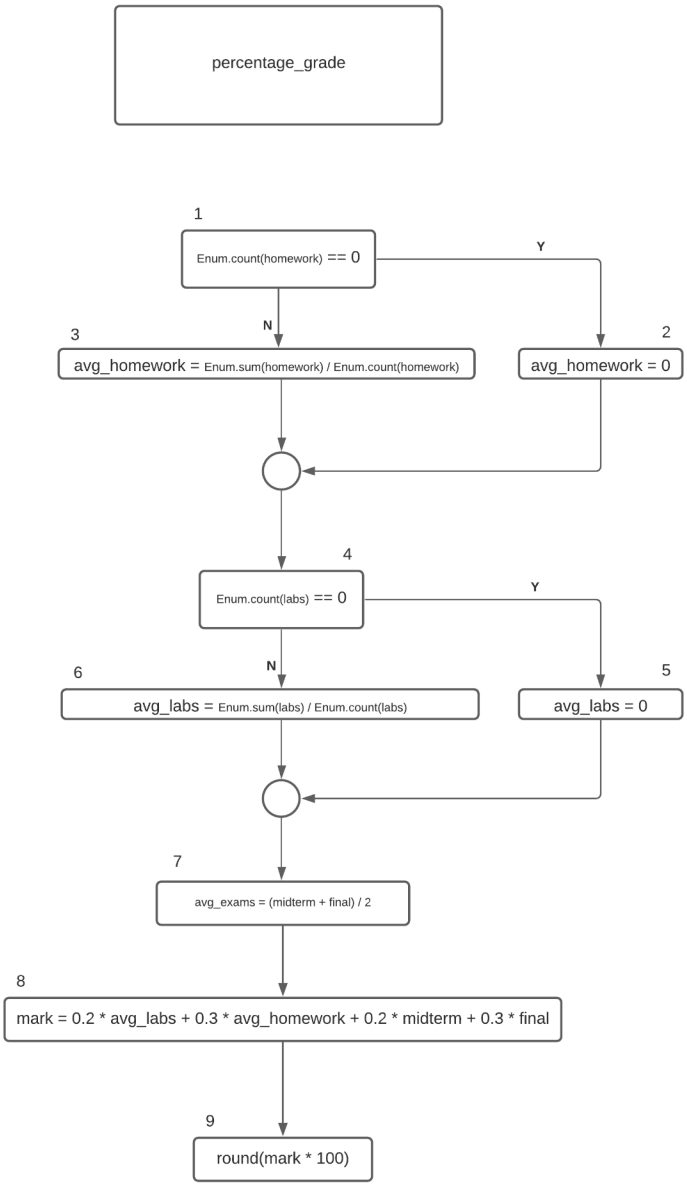
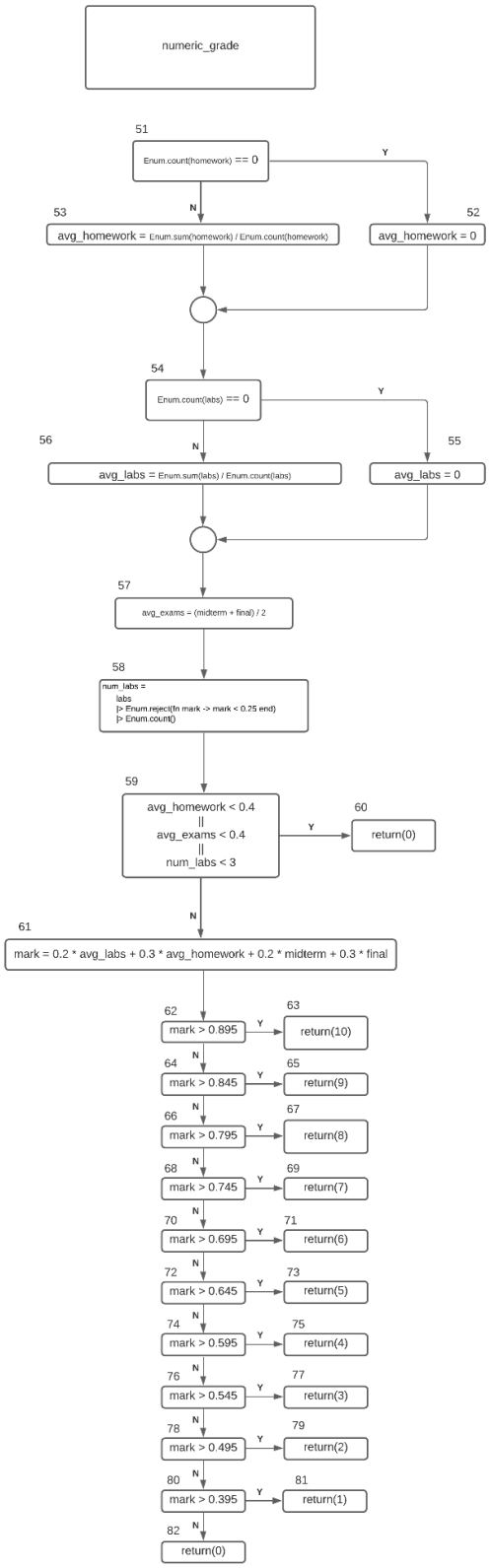
Assignment 2

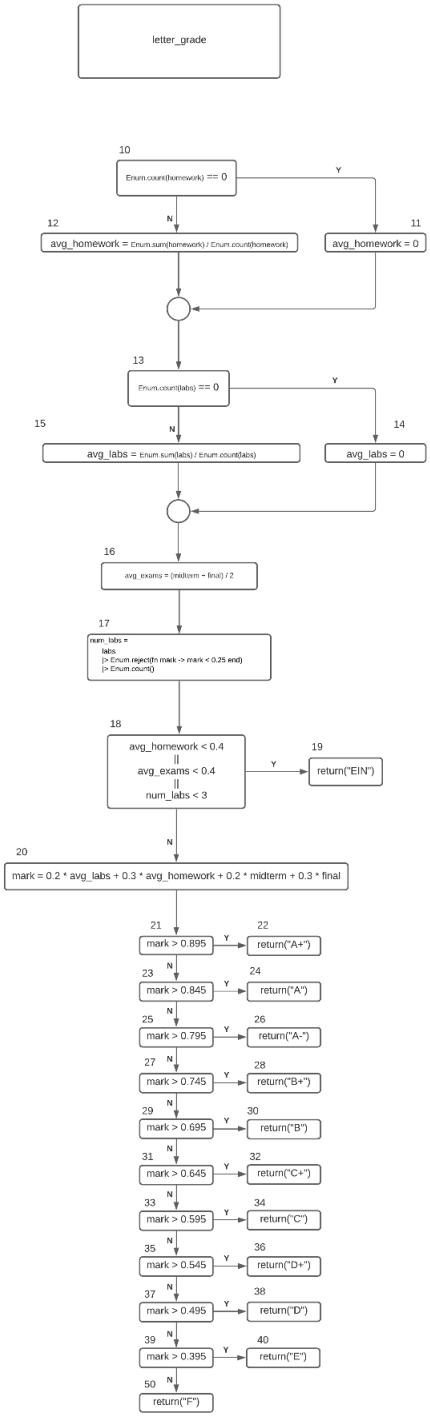
By Gaberiel Cordovado (300110852)
And Michael Kagnew (300113347)

Question 1.1

Draw the simplified control flow graph corresponding to each of the methods percentage_grade, letter_grade, and numeric_grade.




numeric_grade
51
Enum.count(homework) == 0
Y
N
53
avg_homework = Enum.sum(homework) / Enum.count(homework)
52
avg_homework = 0
54
Enum.count(labs) == 0
Y
N
56
avg_labs = Enum.sum(labs) / Enum.count(labs)
55
avg_labs = 0
57
avg_exams = (midterm + final) / 2
58
num_labs =
labs
|> Enum.reject(fn mark -> mark < 0.25 end)
|> Enum.count()
59
avg_homework < 0.4
||
avg_exams < 0.4
||
num_labs < 3
Y
60
return(0)
N
61
mark = 0.2 * avg_labs + 0.3 * avg_homework + 0.2 * midterm + 0.3 * final
62
mark > 0.895
Y
63
return(10)
N
64
mark > 0.845
Y
65
return(9)
N
66
mark > 0.795
Y
67
return(8)
N
68
mark > 0.745
Y
69
return(7)
N
70
mark > 0.695
Y
71
return(6)
N
72
mark > 0.645
Y
73
return(5)
N
74
mark > 0.595
Y
75
return(4)
N
76
mark > 0.545
Y
77
return(3)
N
78
mark > 0.495
Y
79
return(2)
N
80
mark > 0.395
Y
81
return(1)
N
82
return(0)


letter_grade
10
Enum.count(homework) == 0
Y
N
12
avg_homework = Enum.sum(homework) / Enum.count(homework)
11
avg_homework = 0
13
Enum.count(labs) == 0
Y
N
15
avg_labs = Enum.sum(labs) / Enum.count(labs)
14
avg_labs = 0
16
avg_exams = (midterm + final) / 2
17
num_labs =
labs
|> Enum.reject(fn mark -> mark < 0.25 end)
|> Enum.count()
18
avg_homework < 0.4
||
avg_exams < 0.4
||
num_labs < 3
Y
19
return("EIN")
N
20
mark = 0.2 * avg_labs + 0.3 * avg_homework + 0.2 * midterm + 0.3 * final
21
mark > 0.895
Y
22
return("A+")
N
23
mark > 0.845
Y
24
return("A")
N
25
mark > 0.795
Y
26
return("A-")
N
27
mark > 0.745
Y
28
return("B+")
N
29
mark > 0.695
Y
30
return("B")
N
31
mark > 0.645
Y
32
return("C+")
N
33
mark > 0.595
Y
34
return("C")
N
35
mark > 0.545
Y
36
return("D+")
N
37
mark > 0.495
Y
38
return("D")
N
39
mark > 0.395
Y
40
return("E")
N
50
return("F")

Question 1.2

Provide a white box test design for 100% branch coverage of the methods percentage_grade, letter_grade, and numeric_grade. Your test suite will be evaluated on the number of its test cases (try to have the smallest possible number of test cases that will allow 100% branch coverage). Use the following template for your test case design:

Tests **1-12** input data into the **letter_grade()** function.
Tests **13-24** input data into the **numeric_grade()** function.
Tests **25-26** input data into the **percentage_grade()** function.

* The branch-cover associated with each test case corresponds to the edge between two statements that has NOT been covered by any test-case before. Hence, after each test, all branches should be mentioned ONCE in the table under the “branches covered” column.

=> ex. Test case 2 will not contain branch edges 10 to 20 because they have been covered by test case 1.

* some conditions are grouped together for visual purposes, definitions are under the table.

Test Case Number	Test Data	Expected Result For Test Case	Conditions Covered	Branches covered
1, 13, 25	Homework = [0.99] Labs = [0.99,0.99,0.99] Midterm = 0.99 Final = 0.99	**(1)** "A+" **(13)** 10 **(25)** 99	Common Conditions[1] mark > 0.895	**(1)** 10-12-13-15-16-17-18-20-21-22 **(13)** 51-53-54-56-57-58-59-61-62-63 **(25)** 1-3-4-6-7-8-9
2, 14	Homework = [0.85] Labs = [0.85, 0.85, 0.85] Midterm = 0.85 Final = 0.85	**(2)** “A” **(14)** 9	Common Conditions[1] mark < 0.895 mark > 0.845	**(2)** 23-24 **(14)** 64-65
3, 15	Homework = [0.80] Labs = [0.80, 0.80, 0.80] Midterm = 0.80 Final = 0.80	**(3)** “A-” **(15)** 8	Common Conditions[1] mark < 0.845 mark > 0.795	**(3)** 25-26 **(15)** 66-67
4, 16	Homework =]0.75] Labs = [0.75,0.75,0.75] Midterm = 0.75	**(4)** “B+” **(16)** 7	Common Conditions[1] mark < 0.795 mark > 0.745	**(4)** 27-28 **(16)** 68-69

	Final = 0.75			
5, 17	Homework = [0.70] Labs = [0.70,0.70,0.70] Midterm = 0.70 Final = 0.70	**(5)** “B” **(17)** 6	Common Conditions[1] mark < 0.745 mark > 0.695	**(5)** 29-30 **(17)** 70-71
6, 18	Homework = 0.65 Labs = [0.65,0.65,0.65] Midterm = 0.65 Final = 0.65	**(6)** “C+” **(18)** 5	Common Conditions[1] mark < 0.695 mark > 0.645	**(6)** 31-32 **(18)** 72-73
7, 19	Homework = [0.60] Labs = [0.60,0.60,0.60] Midterm = 0.60 Final = 0.60	**(7)** “C” **(19)** 4	Common Conditions[1] mark < 0.645 mark > 0.595	**(7)** 33-34 **(19)** 74-75
8, 20	Homework = [0.55] Labs = [0.55,0.55,0.55] Midterm = 0.55 Final = 0.55	**(8)** “D+” **(20)** 3	Common Conditions[1] mark < 0.595 mark > 0.545	**(8)** 35-36 **(20)** 76-77
9, 21	Homework = [0.50] Labs = [0.50,0.50,0.50] Midterm = 0.50 Final = 0.50	**(9)** “D” **(21)** 2	Common Conditions[1] mark < 0.545 mark > 0.495	**(9)** 37-38 **(21)** 78-79
10, 22	Homework = [0.40] Labs = [0.40,0.40,0.40] Midterm = 0.40 Final = 0.40	**(10)** “E” **(22)** 1	Common Conditions[1] mark < 0.495 mark > 0.395	**(10)** 39-40 **(22)** 80-81
11, 23	Homework = [0.40] Labs = [0,0,0,0.25,0.25,0.25,0.25,0,0,0,0,0,0,0,0] Midterm = 0.4 Final = 0.4	**(11)** “F” **(23)** 0	Common Conditions[1] mark < 0.395	**(11)** 39-50 **(23)** 80-82
12, 24, 26	Homework = [] Labs = [] Midterm = 0.3 Final = 0.3	**(12)** “EIN” **(24)** 0 **(26)** 15	Uncommon Conditions[2] Enum.count(homework) == 0 Enum.count(labs) == 0	**(12)**10-11-13-14, 18-19 **(24)** 51-52-54-55-57, 59-60 **(26)** 1-2-4-5-7

[1] Common Conditions (Enum.count(homework) != 0, Enum.count(labs) != 0, Avg_homwork > 0.4, Avg_exams > 0.4, Num_labs > 3)
[2] Uncommon Conditions (Avg_homework < 0.4, Avg_exams < 0.4, Num_labs < 3)

Question 1.3

Provide an implementation of your test suite using ExUnit.
* code embedded within the zipped-repository

Question 1.4

What is the degree of statement coverage obtained? If you weren't able to obtain 100% coverage, explain why. Please be sure to attach screenshots of coverage results. Elixir's coverage tool is primitive, as it only provides statement level accuracy. How might you address the limitations of a testing tool that only provides statement level coverage?

```
Finished in 0.2 seconds (0.1s async, 0.1s sync)
29 tests, 0 failures

Randomized with seed 163000

Generating cover results ...

Percentage | Module
-----------|-------------------------
     0.00% | GradesWeb
     0.00% | GradesWeb.ChannelCase
     0.00% | GradesWeb.ErrorHelpers
     9.09% | GradesWeb.PageLive
    50.00% | GradesWeb.LayoutView
    50.00% | GradesWeb.UserSocket
    66.67% | GradesWeb.ErrorView
    75.00% | Grades.Application
    75.00% | GradesWeb.Router
    75.00% | GradesWeb.Telemetry
   100.00% | Grades
   100.00% | Grades.Calculator
   100.00% | GradesWeb.ConnCase
   100.00% | GradesWeb.Endpoint
   100.00% | GradesWeb.Router.Helpers
-----------|-------------------------
    74.16% | Total

Generated HTML coverage results in "cover" directory
```

Coverage results from `mix test --cover` command

The degree of statement coverage for the calculator module comes out to 100%. The limitations of the testing coverage only to statements will miss potential logic issues that are only identifiable in other coverage types, such as branch and/or condition coverage. Statement coverage only indicates that each statement has been executed once, but does not test if the true/false logic in conditional statements are achievable, which is not done by statement coverage. To address this issue, one could either use languages that provide other types of coverage, or manually test for other coverages in flow charts, such as what we did in question 1.1.

Question 2.1

Extract a helper method **avg** to clean up the duplicate code like...

```
avg_homework =
  if Enum.count(homework) == 0 do
    0
  else
    Enum.sum(homework) / Enum.count(homework)
  end
```

Commit: 75fd539f00b01fef9fe2d187a1c0b632f2c4f9bc
Commit-Title: `refractor calculation for the average of a list of floats`
Commit-Message: `replaced repetitive code regarding the avg grade of a list of float values, this was replaced by an avg function that takes one parameter, the list of floats, and returns a float representing the avg grade of those values. Lines 19, 21, 28, 30, 61, 63 reflect these changes. All test cases pass`

* image of commit on next page of the document *

58 grades/lib/grades/calculator.ex

```diff
@@ -1,37 +1,33 @@
 defmodule Grades.Calculator do
+
+  # Question 2.1 (Gabriel Cordovado)
+  # This function will take in the homework and lab lists seperatly to reduce the amount of code
+  # that is being duplicated within the percentage_grade, letter_grade, and numeric_grade classes
+  #
+  # @param list_of_n_items: list of floats
+  # @returns an float representing the average of N items within the list of floats
+  #
+  def avg(list_of_n_items) do
+    if Enum.count(list_of_n_items) == 0 do
+      0
+    else
+      Enum.sum(list_of_n_items) / Enum.count(list_of_n_items)
+    end
+  end
+
   def percentage_grade(%{homework: homework, labs: labs, midterm: midterm, final: final}) do
-    avg_homework =
-      if Enum.count(homework) == 0 do
-        0
-      else
-        Enum.sum(homework) / Enum.count(homework)
-      end
+    avg_homework = avg(homework)  # refractor 2.1
 
-    avg_labs =
-      if Enum.count(labs) == 0 do
-        0
-      else
-        Enum.sum(labs) / Enum.count(labs)
-      end
+    avg_labs = avg(labs)  # refractor 2.1
 
     mark = 0.2 * avg_labs + 0.3 * avg_homework + 0.2 * midterm + 0.3 * final
     round(mark * 100)
   end
 
   def letter_grade(%{homework: homework, labs: labs, midterm: midterm, final: final}) do
-    avg_homework =
-      if Enum.count(homework) == 0 do
-        0
-      else
-        Enum.sum(homework) / Enum.count(homework)
-      end
+    avg_homework = avg(homework)  # refractor 2.1
 
-    avg_labs =
-      if Enum.count(labs) == 0 do
-        0
-      else
-        Enum.sum(labs) / Enum.count(labs)
-      end
+    avg_labs = avg(labs)  # refractor 2.1
 
     avg_exams = (midterm + final) / 2
 
@@ -62,19 +58,9 @@ defmodule Grades.Calculator do
   end
 
   def numeric_grade(%{homework: homework, labs: labs, midterm: midterm, final: final}) do
-    avg_homework =
-      if Enum.count(homework) == 0 do
-        0
-      else
-        Enum.sum(homework) / Enum.count(homework)
-      end
+    avg_homework = avg(homework)  # refractor 2.1
 
-    avg_labs =
-      if Enum.count(labs) == 0 do
-        0
-      else
-        Enum.sum(labs) / Enum.count(labs)
-      end
+    avg_labs = avg(labs)  # refractor 2.1
 
     avg_exams = (midterm + final) / 2
```

Question 2.2

Extract a helper method **failed_to_participate** to clean up duplicate code like...

```
avg_homework < 0.4 || avg_exams < 0.4 || num_labs(labs) < 3
```

Commit: [2d1bd4f8b481d440db698266b94a86df5d8a9df3]
Commit-Title: `Refactor 2.2`
Commit-Message: `wrapped the calculation if the student was able to participate in marking in a single function to return true or false. Affects lines 60, and 93`

16 grades/lib/grades/calculator.ex

```diff
@@ -26,6 +26,15 @@ defmodule Grades.Calculator do
 26  26     def calculate_grade(avg_labs, avg_homework, midterm, final) do
 27  27       0.2 * avg_labs + 0.3 * avg_homework + 0.2 * midterm + 0.3 * final
 28  28     end
     29 +
     30 +     # Question 2.2 (Michael Kagnew)
     31 +   # The formula used to determine if the student is able to participate in the marking scheme.
     32 +   # @param avg_homework, avg_exams, num_labs : boolean
     33 +   # @returns a boolean determining if the student is able to participate in the grade calculation.
     34 +   #
     35 +   def failed_to_participate(avg_homework, avg_exams, num_labs) do
     36 +     avg_homework < 0.4 || avg_exams < 0.4 || num_labs < 3
     37 +   end
 29  38
 30  39     def percentage_grade(%{homework: homework, labs: labs, midterm: midterm, final: final}) do
 31  40       avg_homework = avg(homework)  # refractor 2.1
@@ -48,7 +57,7 @@ defmodule Grades.Calculator do
 48  57         |> Enum.reject(fn mark -> mark < 0.25 end)
 49  58         |> Enum.count()
 50  59
 51     -     if avg_homework < 0.4 || avg_exams < 0.4 || num_labs < 3 do
     60 +     if failed_to_participate(avg_homework, avg_exams, num_labs) do #refractor 2.2
 52  61         "EIN"
 53  62       else
 54  63         mark = calculate_grade(avg_labs, avg_homework, midterm, final)  # refractor 2.3
@@ -81,7 +90,7 @@ defmodule Grades.Calculator do
 81  90         |> Enum.reject(fn mark -> mark < 0.25 end)
 82  91         |> Enum.count()
 83  92
 84     -     if avg_homework < 0.4 || avg_exams < 0.4 || num_labs < 3 do
     93 +     if failed_to_participate(avg_homework, avg_exams, num_labs) do #refractor 2.2
 85  94         0
 86  95       else
 87  96         mark = calculate_grade(avg_labs, avg_homework, midterm, final)  # refractor 2.3
@@ -101,4 +110,7 @@ defmodule Grades.Calculator do
101 110         end
102 111       end
103 112     end
    113 +
    114 +
    115 +
104 116   end
```

Question 2.3

Extract a helper method **calculate_grade** to clean up duplicate code like...

```
0.2 * avg_labs + 0.3 * avg_homework + 0.2 * midterm + 0.3 * final
```

Commit: [3cc0560ac15b411f583cfd6b488284cc85e6251d]

Commit-Title: `refractor 2.3 abstracted to formula to calculate the student final grade into calculate_grade`

Commit-Message: `the formula was placed in multiple functions within the Grade.calculator module, this should ease code readability and maintainability. Affects lines 35, 54, and 87. All coverage tests pass`

18 grades/lib/grades/calculator.ex

```diff
@@ -15,12 +15,24 @@ defmodule Grades.Calculator do
15 15      end
16 16    end
17 17
18 +   # Question 2.3 (Gabriel Cordovado)
19 +   # the formula to calculate the student's offical grade is referenced multipl times throughout
20 +   # the program, this will help reduce the amount of duplicated lines and make the code more
21 +   # readable to the developer
22 +   #
23 +   # @param avg_labs, avg_homework, midterm, final : float
24 +   # @returns a float representing the students calculated grade
25 +   #
26 +   def calculate_grade(avg_labs, avg_homework, midterm, final) do
27 +     0.2 * avg_labs + 0.3 * avg_homework + 0.2 * midterm + 0.3 * final
28 +   end
29 +
18 30    def percentage_grade(%{homework: homework, labs: labs, midterm: midterm, final: final}) do
19 31      avg_homework = avg(homework)  # refractor 2.1
20 32
21 33      avg_labs = avg(labs)  # refractor 2.1
22 34
23 -     mark = 0.2 * avg_labs + 0.3 * avg_homework + 0.2 * midterm + 0.3 * final
35 +     mark = calculate_grade(avg_labs, avg_homework, midterm, final)  # refractor 2.3
24 36      round(mark * 100)
25 37    end
26 38
@@ -39,7 +51,7 @@ defmodule Grades.Calculator do
39 51      if avg_homework < 0.4 || avg_exams < 0.4 || num_labs < 3 do
40 52        "EIN"
41 53      else
42 -       mark = 0.2 * avg_labs + 0.3 * avg_homework + 0.2 * midterm + 0.3 * final
54 +       mark = calculate_grade(avg_labs, avg_homework, midterm, final)  # refractor 2.3
43 55
44 56        cond do
45 57          mark > 0.895 -> "A+"
@@ -72,7 +84,7 @@ defmodule Grades.Calculator do
72 84      if avg_homework < 0.4 || avg_exams < 0.4 || num_labs < 3 do
73 85        0
74 86      else
75 -       mark = 0.2 * avg_labs + 0.3 * avg_homework + 0.2 * midterm + 0.3 * final
87 +       mark = calculate_grade(avg_labs, avg_homework, midterm, final)  # refractor 2.3
76 88
77 89        cond do
78 90          mark > 0.895 -> 10
```

Question 2.4

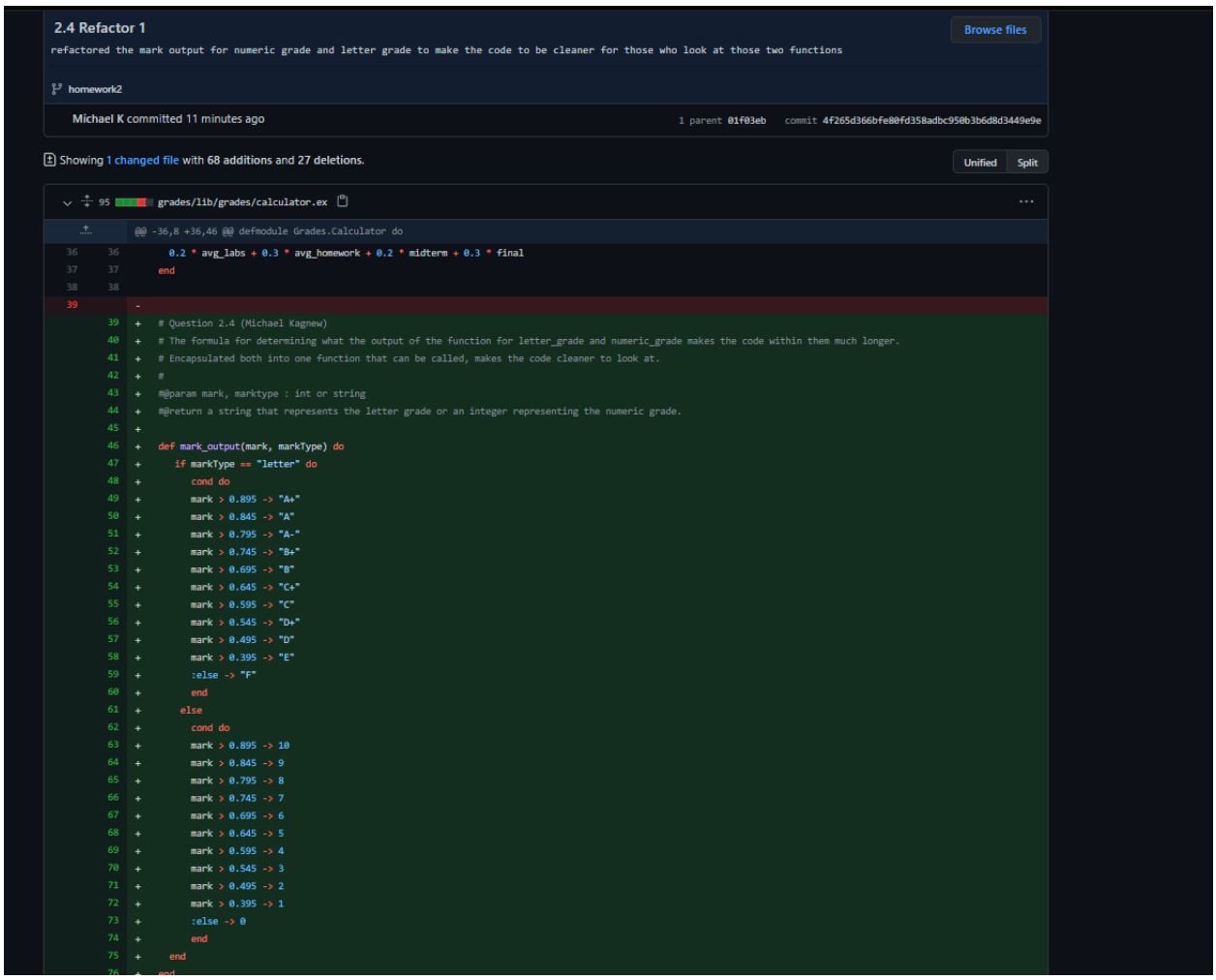
Provide at least 2 additional refactoring to the code. Your refactoring should not require additional testing, however if you encounter any bugs in the original code then please fix them separately (ensuring your tests continue to pass) before continuing to refactor.

(1)

Commit: [4f265d366bfe80fd358adbc950b3b6d8d3449e9e]

Commit-Title: `2.4 Refactor 1`

Commit-Message: `refactored the mark output for numeric grade and letter grade to make the code to be cleaner for those who look at those two functions`



`Refactor 2.4 (1/2): half of the commit screenshot`

```
      else
        mark = calculate_grade(avg_labs, avg_homework, midterm, final)  # refractor 2.3

-       cond do
-         mark > 0.895 -> "A+"
-         mark > 0.845 -> "A"
-         mark > 0.795 -> "A-"
-         mark > 0.745 -> "B+"
-         mark > 0.695 -> "B"
-         mark > 0.645 -> "C+"
-         mark > 0.595 -> "C"
-         mark > 0.545 -> "D+"
-         mark > 0.495 -> "D"
-         mark > 0.395 -> "E"
-         :else -> "F"
-       end
+       # cond do
+       #   mark > 0.895 -> "A+"
+       #   mark > 0.845 -> "A"
+       #   mark > 0.795 -> "A-"
+       #   mark > 0.745 -> "B+"
+       #   mark > 0.695 -> "B"
+       #   mark > 0.645 -> "C+"
+       #   mark > 0.595 -> "C"
+       #   mark > 0.545 -> "D+"
+       #   mark > 0.495 -> "D"
+       #   mark > 0.395 -> "E"
+       #   :else -> "F"
+       # end
+       mark_output(mark, "letter")
      end
    end

@@ -97,19 +136,21 @@ defmodule Grades.Calculator do
      else
        mark = calculate_grade(avg_labs, avg_homework, midterm, final)  # refractor 2.3

-       cond do
-         mark > 0.895 -> 10
-         mark > 0.845 -> 9
-         mark > 0.795 -> 8
-         mark > 0.745 -> 7
-         mark > 0.695 -> 6
-         mark > 0.645 -> 5
-         mark > 0.595 -> 4
-         mark > 0.545 -> 3
-         mark > 0.495 -> 2
-         mark > 0.395 -> 1
-         :else -> 0
-       end
+       # cond do
+       #   mark > 0.895 -> 10
+       #   mark > 0.845 -> 9
+       #   mark > 0.795 -> 8
+       #   mark > 0.745 -> 7
+       #   mark > 0.695 -> 6
+       #   mark > 0.645 -> 5
+       #   mark > 0.595 -> 4
+       #   mark > 0.545 -> 3
+       #   mark > 0.495 -> 2
+       #   mark > 0.395 -> 1
+       #   :else -> 0
+       # end
+       mark_output(mark, "number")
+
    end
```

Refactor 2.4 (2/2): Other half of the commit screenshot

(2)

Commit: [4777ce16a34bcededd1250a19612191273527e89]

Commit-Title: `refractor 2.4 - part 3`

Commit-Message: `placed the formula for calculating the number of labs into an isolated function to avoid the need for 3 repeated lines in each case of percentage_grade and numeric_grade. The changes take place on lines 106 and 121. All tests pass.`

```
68 grades/lib/grades/calculator.ex

@@ -14,13 +14,13 @@ defmodule Grades.Calculator do
14 14          Enum.sum(list_of_n_items) / Enum.count(list_of_n_items)
15 15        end
16 16      end
17    -
   17 +
18 18      # Question 2.2 (Michael Kagnew)
19 19      # The formula used to determine if the student is able to participate in the marking scheme.
20 20      # @param avg_homework, avg_exams, num_labs : boolean
21 21      # @returns a boolean determining if the student is able to participate in the grade calculation.
22 22      #
23    -   def failed_to_participate(avg_homework, avg_exams, num_labs) do
   23 +   def failed_to_participate(avg_homework, avg_exams, num_labs) do
24 24         avg_homework < 0.4 || avg_exams < 0.4 || num_labs < 3
25 25      end
26 26

@@ -35,14 +35,14 @@ defmodule Grades.Calculator do
35 35      def calculate_grade(avg_labs, avg_homework, midterm, final) do
36 36        0.2 * avg_labs + 0.3 * avg_homework + 0.2 * midterm + 0.3 * final
37 37      end
38    -
   38 +
39 39      # Question 2.4 (Michael Kagnew)
40 40      # The formula for determining what the output of the function for letter_grade and numeric_grade makes the code within them much longer.
41 41      # Encapsulated both into one function that can be called, makes the code cleaner to look at.
42 42      #
43 43      #@param mark, marktype : int or string
44 44      #@return a string that represents the letter grade or an integer representing the numeric grade.
45    -
   45 +
46 46      def mark_output(mark, markType) do
47 47         if markType == "letter" do
48 48            cond do

@@ -59,7 +59,7 @@ defmodule Grades.Calculator do
59 59            :else -> "F"
60 60            end
61 61         else
62    -         cond do
   62 +         cond do
63 63            mark > 0.895 -> 10
64 64            mark > 0.845 -> 9
65 65            mark > 0.795 -> 8

@@ -75,7 +75,20 @@ defmodule Grades.Calculator do
75 75        end
76 76      end
77 77
78    -
   78 +   # Question 2.4 (Gabriel Cordovado)
   79 +   # The formula for calculating the number of labs is the same for each instance of percentage_grade
   80 +   # and letter grade so it would be better if we just encapsulated this data and swapped it in for
```

continuous on next page - screenshot too large

```diff
+  # Question 2.4 (Gabriel Cordovado)
+  # The formula for calculating the number of labs is the same for each instance of percentage_grade
+  # and letter_grade so it would be better if we just encapsulated this data and swapped it in for
+  # where the 3 repetative lines need to me for each area
+  #
+  #@param list_lab : list of floats
+  #@return an integer representing the number of lab grades (floats) in the passed list
+  #
+  def count_labs(list_lab) do
+    list_lab
+      |> Enum.reject(fn mark -> mark < 0.25 end)
+      |> Enum.count()
+  end
+
   def percentage_grade(%{homework: homework, labs: labs, midterm: midterm, final: final}) do
     avg_homework = avg(homework)  # refractor 2.1

@@ -87,70 +100,31 @@ defmodule Grades.Calculator do

   def letter_grade(%{homework: homework, labs: labs, midterm: midterm, final: final}) do
     avg_homework = avg(homework)  # refractor 2.1
-
     avg_labs = avg(labs)  # refractor 2.1
-
     avg_exams = (midterm + final) / 2

-    num_labs =
-      labs
-      |> Enum.reject(fn mark -> mark < 0.25 end)
-      |> Enum.count()
+    num_labs = count_labs(labs)  # refractor 2.4 - 3

     if failed_to_participate(avg_homework, avg_exams, num_labs) do #refractor 2.2
       "EIN"
     else
       mark = calculate_grade(avg_labs, avg_homework, midterm, final)  # refractor 2.3
-
-      # cond do
-      #   mark > 0.895 -> "A+"
-      #   mark > 0.845 -> "A"
-      #   mark > 0.795 -> "A-"
-      #   mark > 0.745 -> "B+"
-      #   mark > 0.695 -> "B"
-      #   mark > 0.645 -> "C+"
-      #   mark > 0.595 -> "C"
-      #   mark > 0.545 -> "D+"
-      #   mark > 0.495 -> "D"
-      #   mark > 0.395 -> "E"
-      #   :else -> "F"
-      # end
       mark_output(mark, "letter")
     end
   end

   def numeric_grade(%{homework: homework, labs: labs, midterm: midterm, final: final}) do
     avg_homework = avg(homework)  # refractor 2.1
-
     avg_labs = avg(labs)  # refractor 2.1
-
     avg_exams = (midterm + final) / 2

-    num_labs =
-      labs
-      |> Enum.reject(fn mark -> mark < 0.25 end)
-      |> Enum.count()
+    num_labs = count_labs(labs)  # refractor 2.4 - 3

     if failed_to_participate(avg_homework, avg_exams, num_labs) do #refractor 2.2
       0
     else
       mark = calculate_grade(avg_labs, avg_homework, midterm, final)  # refractor 2.3
-
-      # cond do
-      #   mark > 0.895 -> 10
-      #   mark > 0.845 -> 9
-      #   mark > 0.795 -> 8
-      #   mark > 0.745 -> 7
-      #   mark > 0.695 -> 6
-      #   mark > 0.645 -> 5
-      #   mark > 0.595 -> 4
-      #   mark > 0.545 -> 3
-      #   mark > 0.495 -> 2
-      #   mark > 0.395 -> 1
-      #   :else -> 0
-      # end
       mark_output(mark, "number")
-
     end
   end
```