Лабораторная работа 2

1. Загрузить среду программирования.

2. Выполнить задачи по варианту. Номер варианта равен номеру рабочего места.

3. Представить результат преподавателю.

Варианты:

1	1. Реализуйте функцию sum_neg_squares(List), которая возвращает сумму квадратов всех отрицательных чисел в списке List. sum_pos_squares([-3,a,false,-3,1]) => 18 2. Не смотря на определение в модуле lists стандартной библиотеки, реализуйте функцию dropwhile(Pred, List). Она возвращает то, что остаётся от списка List после отбрасывания начальных элементов, на которых Pred возвращает true. dropwhile(fun(X) -> X < 10 end, [1,3,9,11,6]) => [11, 6] 3. Реализуйте функцию antimap(ListF, X), которая принимает список функций одного аргумента ListF и значение X, и возвращает список результатов применения всех функций из ListF к X. antimap([fun(X) -> X + 2 end, fun(X) -> X*3 end], 4) => [6, 12] 4. Реализуйте функцию solve(Fun, A, B, Eps), которая находит приближённо (с ошибкой не больше Eps) корень уравнения Fun(X) = 0 на отрезке [A, B] или точку разрыва, в которой Fun меняет знак. Можно считать, что F(A) <= 0 <= F(B) (как известно, в таком случае корень заведомо существует). Проще всего это сделать, деля отрезок пополам и смотря, на концах какой половины различаются знаки Fun. solve(fun(X) -> X*X - 2 end, 0, 2, 0.001) => 1.414 (приближенно)
2	1. Реализуйте функцию list_heads(List), которая возвращает список первых элементов непустых списков, входящих в List и игнорирует любые другие элементы List. list_heads([[1,2,3], {true,3}, [4,5], []]) => [1,4] 2. Не смотря на определение в модуле lists стандартной библиотеки, реализуйте функцию takewhile(Pred, List). Она возвращает такой начальный отрезок списка List, для всех элементов которого Pred возвращает true. В отличие от filter, она заканчивает работу, как только найдёт элемент, на котором Pred вернёт false. takewhile(fun(X) -> X < 10 end, [1,3,9,11,6]) => [1,3,9] 3. Реализуйте функцию iterate(F, N), которая возвращает функцию, применяющую F к своему аргументу N раз (т.е., например, (iterate(F, 2))(X) == F(F(X)))

	F1 = iterate(fun(X) -> {X} end, 2), F1(1) => {{1}} 4. Реализуйте функцию integrate(F, N), принимающую функцию F из действительных чисел в действительные числа) и целое число N, и возвращающую функцию 2 аргументов: (integrate(F, N))(A, B) приближенно равно определённому интегралу F от A до B (для подсчёта которого отрезок разбивается на N частей). F1 = integrate(fun(X) -> X end, 100), F1(0, 1) => 0.5 (приближенно)
3	1. Реализуйте функцию list_lengths(List), которая возвращает список длин списков, входящих в List и пропускает все остальные элементы. list_lengths([[1,2,3], {true,3}, [4,5], []]) => [3,2,0] 2. Не смотря на определение в модуле lists стандартной библиотеки, реализуйте функцию all(Pred, List). Она возвращает true, если Pred возвращает true для всех элементов List, и false, если это не так. all(fun(X) -> X < 10 end, [1,3,9,11,6]) => false all(fun(X) -> X < 10 end, [1,3,9,6]) => true 3. Реализуйте функцию min_value(F, N), которая возвращает минимальное значение функции F на целых числах от 1 до N. max_value(fun(X) -> X rem 5 end, 10) => 0 4. Реализуйте функцию group_by(Fun, List), которая разбивает список List на отрезки, на идущих подряд элементах которых Fun (предикат от двух переменных) возвращает true. group_by(fun(X, Y) <- X =< Y end, [1,2,4,3,2,5]) => [[1,2,4], [3], [2,5]]
4	1. Реализуйте функцию min_positive_number(List), которая возвращает минимальное положительное число, входящее в List. Если положительных чисел нет, функция должна вернуть атом error. min_positive_number([3,a,false,-3,1]) => 1 2. Не смотря на определение в модуле lists стандартной библиотеки, реализуйте функцию zipwith(Fun, List1, List2). Возвращается список значений Fun (функции от двух аргументов) на аргументах, взятых из списков List1 и List2. В случае разных длин списков функция должна выкинуть исключение. zipwith(fun(X, Y) -> {X, Y} end, [1,2,3], [4,5,6]) => [{1,4}, {2,5}, {3,6}] 3. Реализуйте функцию iteratemap(F, X0, N), которая возвращает список длины N, состоящий из результатов последовательного применения F к X0. iteratemap(fun(X) -> X*2 end, 1, 4) => [1, 2, 4, 8]

	4. Реализуйте функцию diff(F, DX), которая принимает функцию F от одного аргумента и шаг DX, и возвращает функцию одного аргумента: приближение к производной функции F. F1 = diff(fun(X) -> X*X end, 0.001), F1(1.0) => 2.000 (приближенно) Попробуйте вспомнить (или найти) формулу, которая даёт лучшее приближение для производной, чем (F(X + DX) - F(X))/DX, но она тоже принимается.
5	1. Реализуйте функцию sum_neg_squares(List), которая возвращает сумму квадратов всех отрицательных чисел в списке List. sum_pos_squares([-3,a,false,-3,1]) => 18 2. Не смотря на определение в модуле lists стандартной библиотеки, реализуйте функцию dropwhile(Pred, List). Она возвращает то, что остаётся от списка List после отбрасывания начальных элементов, на которых Pred возвращает true. dropwhile(fun(X) -> X < 10 end, [1,3,9,11,6]) => [11, 6] 3. Реализуйте функцию antimap(ListF, X), которая принимает список функций одного аргумента ListF и значение X, и возвращает список результатов применения всех функций из ListF к X. antimap([fun(X) -> X + 2 end, fun(X) -> X*3 end], 4) => [6, 12] 4. Реализуйте функцию solve(Fun, A, B, Eps), которая находит приближённо (с ошибкой не больше Eps) корень уравнения Fun(X) = 0 на отрезке [A, B] или точку разрыва, в которой Fun меняет знак. Можно считать, что F(A) <= 0 <= F(B) (как известно, в таком случае корень заведомо существует). Проще всего это сделать, деля отрезок пополам и смотря, на концах какой половины различаются знаки Fun. solve(fun(X) -> X*X - 2 end, 0, 2, 0.001) => 1.414 (приближенно)
6	1. Реализуйте функцию list_heads(List), которая возвращает список первых элементов непустых списков, входящих в List и игнорирует любые другие элементы List. list_heads([[1,2,3], {true,3}, [4,5], []]) => [1,4] 2. Не смотря на определение в модуле lists стандартной библиотеки, реализуйте функцию takewhile(Pred, List). Она возвращает такой начальный отрезок списка List, для всех элементов которого Pred возвращает true. В отличие от filter, она заканчивает работу, как только найдёт элемент, на котором Pred вернёт false. takewhile(fun(X) -> X < 10 end, [1,3,9,11,6]) => [1,3,9] 3. Реализуйте функцию iterate(F, N), которая возвращает функцию, применяющую F к своему аргументу N раз (т.е., например, (iterate(F, 2))(X) == F(F(X))) F1 = iterate(fun(X) -> {X} end, 2), F1(1) => {{1}}

	4. Реализуйте функцию integrate(F, N), принимающую функцию F из действительных чисел в действительные числа) и целое число N, и возвращающую функцию 2 аргументов: (integrate(F, N))(A, B) приближенно равно определённому интегралу F от A до B (для подсчёта которого отрезок разбивается на N частей). F1 = integrate(fun(X) -> X end, 100), F1(0, 1) => 0.5 (приближенно)
7	1. Реализуйте функцию list_lengths(List), которая возвращает список длин списков, входящих в List и пропускает все остальные элементы. list_lengths([[1,2,3], {true,3}, [4,5], []]) => [3,2,0] 2. Не смотря на определение в модуле lists стандартной библиотеки, реализуйте функцию all(Pred, List). Она возвращает true, если Pred возвращает true для всех элементов List, и false, если это не так. all(fun(X) -> X < 10 end, [1,3,9,11,6]) => false all(fun(X) -> X < 10 end, [1,3,9,6]) => true 3. Реализуйте функцию min_value(F, N), которая возвращает минимальное значение функции F на целых числах от 1 до N. max_value(fun(X) -> X rem 5 end, 10) => 0 4. Реализуйте функцию group_by(Fun, List), которая разбивает список List на отрезки, на идущих подряд элементах которых Fun (предикат от двух переменных) возвращает true. group_by(fun(X, Y) <- X =< Y end, [1,2,4,3,2,5]) => [[1,2,4], [3], [2,5]]
8	1. Реализуйте функцию min_positive_number(List), которая возвращает минимальное положительное число, входящее в List. Если положительных чисел нет, функция должна вернуть атом error. min_positive_number([3,a,false,-3,1]) => 1 2. Не смотря на определение в модуле lists стандартной библиотеки, реализуйте функцию zipwith(Fun, List1, List2). Возвращается список значений Fun (функции от двух аргументов) на аргументах, взятых из списков List1 и List2. В случае разных длин списков функция должна выкинуть исключение. zipwith(fun(X, Y) -> {X, Y} end, [1,2,3], [4,5,6]) => [{1,4}, {2,5}, {3,6}] 3. Реализуйте функцию iteratemap(F, X0, N), которая возвращает список длины N, состоящий из результатов последовательного применения F к X0. iteratemap(fun(X) -> X*2 end, 1, 4) => [1, 2, 4, 8] 4. Реализуйте функцию diff(F, DX), которая принимает функцию F от одного аргумента и шаг DX, и возвращает функцию одного аргумента: приближение к производной функции F.

	F1 = diff(fun(X) -> X*X end, 0.001), F1(1.0) => 2.000 (приближенно) Попробуйте вспомнить (или найти) формулу, которая даёт лучшее приближение для производной, чем (F(X + DX) - F(X))/DX, но она тоже принимается.
9	1. Реализуйте функцию sum_neg_squares(List), которая возвращает сумму квадратов всех отрицательных чисел в списке List. sum_pos_squares([-3,a,false,-3,1]) => 18 2. Не смотря на определение в модуле lists стандартной библиотеки, реализуйте функцию dropwhile(Pred, List). Она возвращает то, что остаётся от списка List после отбрасывания начальных элементов, на которых Pred возвращает true. dropwhile(fun(X) -> X < 10 end, [1,3,9,11,6]) => [11, 6] 3. Реализуйте функцию antimap(ListF, X), которая принимает список функций одного аргумента ListF и значение X, и возвращает список результатов применения всех функций из ListF к X. antimap([fun(X) -> X + 2 end, fun(X) -> X*3 end], 4) => [6, 12] 4. Реализуйте функцию solve(Fun, A, B, Eps), которая находит приближённо (с ошибкой не больше Eps) корень уравнения Fun(X) = 0 на отрезке [A, B] или точку разрыва, в которой Fun меняет знак. Можно считать, что F(A) <= 0 <= F(B) (как известно, в таком случае корень заведомо существует). Проще всего это сделать, деля отрезок пополам и смотря, на концах какой половины различаются знаки Fun. solve(fun(X) -> X*X - 2 end, 0, 2, 0.001) => 1.414 (приближенно)
10	1. Реализуйте функцию list_heads(List), которая возвращает список первых элементов непустых списков, входящих в List и игнорирует любые другие элементы List. list_heads([[1,2,3], {true,3}, [4,5], []]) => [1,4] 2. Не смотря на определение в модуле lists стандартной библиотеки, реализуйте функцию takewhile(Pred, List). Она возвращает такой начальный отрезок списка List, для всех элементов которого Pred возвращает true. В отличие от filter, она заканчивает работу, как только найдёт элемент, на котором Pred вернёт false. takewhile(fun(X) -> X < 10 end, [1,3,9,11,6]) => [1,3,9] 3. Реализуйте функцию iterate(F, N), которая возвращает функцию, применяющую F к своему аргументу N раз (т.е., например, (iterate(F, 2))(X) == F(F(X))) F1 = iterate(fun(X) -> {X} end, 2), F1(1) => {{1}} 4. Реализуйте функцию integrate(F, N), принимающую функцию F из действительных чисел в действительные числа) и целое число N, и возвращающую функцию 2 аргументов: (integrate(F, N))(A, B) приближенно равно определённому интегралу F от A до B (для подсчёта которого отрезок разбивается на N частей).

	F1 = integrate(fun(X) -> X end, 100), F1(0, 1) => 0.5 (приближенно)
11	1. Реализуйте функцию list_lengths(List), которая возвращает список длин списков, входящих в List и пропускает все остальные элементы. list_lengths([[1,2,3], {true,3}, [4,5], []]) => [3,2,0] 2. Не смотря на определение в модуле lists стандартной библиотеки, реализуйте функцию all(Pred, List). Она возвращает true, если Pred возвращает true для всех элементов List, и false, если это не так. all(fun(X) -> X < 10 end, [1,3,9,11,6]) => false all(fun(X) -> X < 10 end, [1,3,9,6]) => true 3. Реализуйте функцию min_value(F, N), которая возвращает минимальное значение функции F на целых числах от 1 до N. max_value(fun(X) -> X rem 5 end, 10) => 0 4. Реализуйте функцию group_by(Fun, List), которая разбивает список List на отрезки, на идущих подряд элементах которых Fun (предикат от двух переменных) возвращает true. group_by(fun(X, Y) <- X =< Y end, [1,2,4,3,2,5]) => [[1,2,4], [3], [2,5]]
12	1. Реализуйте функцию min_positive_number(List), которая возвращает минимальное положительное число, входящее в List. Если положительных чисел нет, функция должна вернуть атом error. min_positive_number([3,a,false,-3,1]) => 1 2. Не смотря на определение в модуле lists стандартной библиотеки, реализуйте функцию zipwith(Fun, List1, List2). Возвращается список значений Fun (функции от двух аргументов) на аргументах, взятых из списков List1 и List2. В случае разных длин списков функция должна выкинуть исключение. zipwith(fun(X, Y) -> {X, Y} end, [1,2,3], [4,5,6]) => [{1,4}, {2,5}, {3,6}] 3. Реализуйте функцию iteratemap(F, X0, N), которая возвращает список длины N, состоящий из результатов последовательного применения F к X0. iteratemap(fun(X) -> X*2 end, 1, 4) => [1, 2, 4, 8] 4. Реализуйте функцию diff(F, DX), которая принимает функцию F от одного аргумента и шаг DX, и возвращает функцию одного аргумента: приближение к производной функции F. F1 = diff(fun(X) -> X*X end, 0.001), F1(1.0) => 2.000 (приближенно) Попробуйте вспомнить (или найти) формулу, которая даёт лучшее приближение для производной, чем (F(X + DX) - F(X))/DX, но она тоже принимается.

13	1. Реализуйте функцию sum_neg_squares(List), которая возвращает сумму квадратов всех отрицательных чисел в списке List. sum_pos_squares([-3,a,false,-3,1]) => 18 2. Не смотря на определение в модуле lists стандартной библиотеки, реализуйте функцию dropwhile(Pred, List). Она возвращает то, что остаётся от списка List после отбрасывания начальных элементов, на которых Pred возвращает true. dropwhile(fun(X) -> X < 10 end, [1,3,9,11,6]) => [11, 6] 3. Реализуйте функцию antimap(ListF, X), которая принимает список функций одного аргумента ListF и значение X, и возвращает список результатов применения всех функций из ListF к X. antimap([fun(X) -> X + 2 end, fun(X) -> X*3 end], 4) => [6, 12] 4. Реализуйте функцию solve(Fun, A, B, Eps), которая находит приближённо (с ошибкой не больше Eps) корень уравнения Fun(X) = 0 на отрезке [A, B] или точку разрыва, в которой Fun меняет знак. Можно считать, что F(A) <= 0 <= F(B) (как известно, в таком случае корень заведомо существует). Проще всего это сделать, деля отрезок пополам и смотря, на концах какой половины различаются знаки Fun. solve(fun(X) -> X*X - 2 end, 0, 2, 0.001) => 1.414 (приближенно)
14	1. Реализуйте функцию list_heads(List), которая возвращает список первых элементов непустых списков, входящих в List и игнорирует любые другие элементы List. list_heads([[1,2,3], {true,3}, [4,5], []]) => [1,4] 2. Не смотря на определение в модуле lists стандартной библиотеки, реализуйте функцию takewhile(Pred, List). Она возвращает такой начальный отрезок списка List, для всех элементов которого Pred возвращает true. В отличие от filter, она заканчивает работу, как только найдёт элемент, на котором Pred вернёт false. takewhile(fun(X) -> X < 10 end, [1,3,9,11,6]) => [1,3,9] 3. Реализуйте функцию iterate(F, N), которая возвращает функцию, применяющую F к своему аргументу N раз (т.е., например, (iterate(F, 2))(X) == F(F(X))) F1 = iterate(fun(X) -> {X} end, 2), F1(1) => {{1}} 4. Реализуйте функцию integrate(F, N), принимающую функцию F из действительных чисел в действительные числа) и целое число N, и возвращающую функцию 2 аргументов: (integrate(F, N))(A, B) приближенно равно определённому интегралу F от A до B (для подсчёта которого отрезок разбивается на N частей). F1 = integrate(fun(X) -> X end, 100), F1(0, 1) => 0.5 (приближенно)
15	1. Реализуйте функцию list_lengths(List), которая возвращает список длин списков, входящих в List и пропускает все остальные элементы.

	list_lengths([[1,2,3], {true,3}, [4,5], []]) => [3,2,0] 2. Не смотря на определение в модуле lists стандартной библиотеки, реализуйте функцию all(Pred, List). Она возвращает true, если Pred возвращает true для всех элементов List, и false, если это не так. all(fun(X) -> X < 10 end, [1,3,9,11,6]) => false all(fun(X) -> X < 10 end, [1,3,9,6]) => true 3. Реализуйте функцию min_value(F, N), которая возвращает минимальное значение функции F на целых числах от 1 до N. max_value(fun(X) -> X rem 5 end, 10) => 0 4. Реализуйте функцию group_by(Fun, List), которая разбивает список List на отрезки, на идущих подряд элементах которых Fun (предикат от двух переменных) возвращает true. group_by(fun(X, Y) <- X =< Y end, [1,2,4,3,2,5]) => [[1,2,4], [3], [2,5]]
16	1. Реализуйте функцию min_positive_number(List), которая возвращает минимальное положительное число, входящее в List. Если положительных чисел нет, функция должна вернуть атом error. min_positive_number([3,a,false,-3,1]) => 1 2. Не смотря на определение в модуле lists стандартной библиотеки, реализуйте функцию zipwith(Fun, List1, List2). Возвращается список значений Fun (функции от двух аргументов) на аргументах, взятых из списков List1 и List2. В случае разных длин списков функция должна выкинуть исключение. zipwith(fun(X, Y) -> {X, Y} end, [1,2,3], [4,5,6]) => [{1,4}, {2,5}, {3,6}] 3. Реализуйте функцию iteratemap(F, X0, N), которая возвращает список длины N, состоящий из результатов последовательного применения F к X0. iteratemap(fun(X) -> X*2 end, 1, 4) => [1, 2, 4, 8] 4. Реализуйте функцию diff(F, DX), которая принимает функцию F от одного аргумента и шаг DX, и возвращает функцию одного аргумента: приближение к производной функции F. F1 = diff(fun(X) -> X*X end, 0.001), F1(1.0) => 2.000 (приближенно) Попробуйте вспомнить (или найти) формулу, которая даёт лучшее приближение для производной, чем (F(X + DX) - F(X))/DX, но она тоже принимается.
17	1. Реализуйте функцию sum_neg_squares(List), которая возвращает сумму квадратов всех отрицательных чисел в списке List. sum_pos_squares([-3,a,false,-3,1]) => 18

	2. Не смотря на определение в модуле lists стандартной библиотеки, реализуйте функцию dropwhile(Pred, List). Она возвращает то, что остаётся от списка List после отбрасывания начальных элементов, на которых Pred возвращает true. dropwhile(fun(X) -> X < 10 end, [1,3,9,11,6]) => [11, 6] 3. Реализуйте функцию antimap(ListF, X), которая принимает список функций одного аргумента ListF и значение X, и возвращает список результатов применения всех функций из ListF к X. antimap([fun(X) -> X + 2 end, fun(X) -> X*3 end], 4) => [6, 12] 4. Реализуйте функцию solve(Fun, A, B, Eps), которая находит приближённо (с ошибкой не больше Eps) корень уравнения Fun(X) = 0 на отрезке [A, B] или точку разрыва, в которой Fun меняет знак. Можно считать, что F(A) <= 0 <= F(B) (как известно, в таком случае корень заведомо существует). Проще всего это сделать, деля отрезок пополам и смотря, на концах какой половины различаются знаки Fun. solve(fun(X) -> X*X - 2 end, 0, 2, 0.001) => 1.414 (приближенно)
18	1. Реализуйте функцию list_heads(List), которая возвращает список первых элементов непустых списков, входящих в List и игнорирует любые другие элементы List. list_heads([[1,2,3], {true,3}, [4,5], []]) => [1,4] 2. Не смотря на определение в модуле lists стандартной библиотеки, реализуйте функцию takewhile(Pred, List). Она возвращает такой начальный отрезок списка List, для всех элементов которого Pred возвращает true. В отличие от filter, она заканчивает работу, как только найдёт элемент, на котором Pred вернёт false. takewhile(fun(X) -> X < 10 end, [1,3,9,11,6]) => [1,3,9] 3. Реализуйте функцию iterate(F, N), которая возвращает функцию, применяющую F к своему аргументу N раз (т.е., например, (iterate(F, 2))(X) == F(F(X))) F1 = iterate(fun(X) -> {X} end, 2), F1(1) => {{1}} 4. Реализуйте функцию integrate(F, N), принимающую функцию F из действительных чисел в действительные числа) и целое число N, и возвращающую функцию 2 аргументов: (integrate(F, N))(A, B) приближенно равно определённому интегралу F от A до B (для подсчёта которого отрезок разбивается на N частей). F1 = integrate(fun(X) -> X end, 100), F1(0, 1) => 0.5 (приближенно)
19	1. Реализуйте функцию list_lengths(List), которая возвращает список длин списков, входящих в List и пропускает все остальные элементы. list_lengths([[1,2,3], {true,3}, [4,5], []]) => [3,2,0] 2. Не смотря на определение в модуле lists стандартной библиотеки, реализуйте функцию all(Pred, List).

	Она возвращает true, если Pred возвращает true для всех элементов List, и false, если это не так. all(fun(X) -> X < 10 end, [1,3,9,11,6]) => false all(fun(X) -> X < 10 end, [1,3,9,6]) => true 3. Реализуйте функцию min_value(F, N), которая возвращает минимальное значение функции F на целых числах от 1 до N. max_value(fun(X) -> X rem 5 end, 10) => 0 4. Реализуйте функцию group_by(Fun, List), которая разбивает список List на отрезки, на идущих подряд элементах которых Fun (предикат от двух переменных) возвращает true. group_by(fun(X, Y) <- X =< Y end, [1,2,4,3,2,5]) => [[1,2,4], [3], [2,5]]
20	1. Реализуйте функцию min_positive_number(List), которая возвращает минимальное положительное число, входящее в List. Если положительных чисел нет, функция должна вернуть атом error. min_positive_number([3,a,false,-3,1]) => 1 2. Не смотря на определение в модуле lists стандартной библиотеки, реализуйте функцию zipwith(Fun, List1, List2). Возвращается список значений Fun (функции от двух аргументов) на аргументах, взятых из списков List1 и List2. В случае разных длин списков функция должна выкинуть исключение. zipwith(fun(X, Y) -> {X, Y} end, [1,2,3], [4,5,6]) => [{1,4}, {2,5}, {3,6}] 3. Реализуйте функцию iteratemap(F, X0, N), которая возвращает список длины N, состоящий из результатов последовательного применения F к X0. iteratemap(fun(X) -> X*2 end, 1, 4) => [1, 2, 4, 8] 4. Реализуйте функцию diff(F, DX), которая принимает функцию F от одного аргумента и шаг DX, и возвращает функцию одного аргумента: приближение к производной функции F. F1 = diff(fun(X) -> X*X end, 0.001), F1(1.0) => 2.000 (приближенно) Попробуйте вспомнить (или найти) формулу, которая даёт лучшее приближение для производной, чем (F(X + DX) - F(X))/DX, но она тоже принимается.
21	1. Реализуйте функцию sum_neg_squares(List), которая возвращает сумму квадратов всех отрицательных чисел в списке List. sum_pos_squares([-3,a,false,-3,1]) => 18 2. Не смотря на определение в модуле lists стандартной библиотеки, реализуйте функцию dropwhile(Pred, List). Она возвращает то, что остаётся от списка List после отбрасывания начальных элементов, на которых Pred возвращает true.

	dropwhile(fun(X) -> X < 10 end, [1,3,9,11,6]) => [11, 6] 3. Реализуйте функцию antimap(ListF, X), которая принимает список функций одного аргумента ListF и значение X, и возвращает список результатов применения всех функций из ListF к X. antimap([fun(X) -> X + 2 end, fun(X) -> X*3 end], 4) => [6, 12] 4. Реализуйте функцию solve(Fun, A, B, Eps), которая находит приближённо (с ошибкой не больше Eps) корень уравнения Fun(X) = 0 на отрезке [A, B] или точку разрыва, в которой Fun меняет знак. Можно считать, что F(A) <= 0 <= F(B) (как известно, в таком случае корень заведомо существует). Проще всего это сделать, деля отрезок пополам и смотря, на концах какой половины различаются знаки Fun. solve(fun(X) -> X*X - 2 end, 0, 2, 0.001) => 1.414 (приближенно)
22	1. Реализуйте функцию list_heads(List), которая возвращает список первых элементов непустых списков, входящих в List и игнорирует любые другие элементы List. list_heads([[1,2,3], {true,3}, [4,5], []]) => [1,4] 2. Не смотря на определение в модуле lists стандартной библиотеки, реализуйте функцию takewhile(Pred, List). Она возвращает такой начальный отрезок списка List, для всех элементов которого Pred возвращает true. В отличие от filter, она заканчивает работу, как только найдёт элемент, на котором Pred вернёт false. takewhile(fun(X) -> X < 10 end, [1,3,9,11,6]) => [1,3,9] 3. Реализуйте функцию iterate(F, N), которая возвращает функцию, применяющую F к своему аргументу N раз (т.е., например, (iterate(F, 2))(X) == F(F(X))) F1 = iterate(fun(X) -> {X} end, 2), F1(1) => {{1}} 4. Реализуйте функцию integrate(F, N), принимающую функцию F из действительных чисел в действительные числа) и целое число N, и возвращающую функцию 2 аргументов: (integrate(F, N))(A, B) приближенно равно определённому интегралу F от A до B (для подсчёта которого отрезок разбивается на N частей). F1 = integrate(fun(X) -> X end, 100), F1(0, 1) => 0.5 (приближенно)
23	1. Реализуйте функцию list_lengths(List), которая возвращает список длин списков, входящих в List и пропускает все остальные элементы. list_lengths([[1,2,3], {true,3}, [4,5], []]) => [3,2,0] 2. Не смотря на определение в модуле lists стандартной библиотеки, реализуйте функцию all(Pred, List). Она возвращает true, если Pred возвращает true для всех элементов List, и false, если это не так. all(fun(X) -> X < 10 end, [1,3,9,11,6]) => false all(fun(X) -> X < 10 end, [1,3,9,6]) => true

	3. Реализуйте функцию min_value(F, N), которая возвращает минимальное значение функции F на целых числах от 1 до N. max_value(fun(X) -> X rem 5 end, 10) => 0 4. Реализуйте функцию group_by(Fun, List), которая разбивает список List на отрезки, на идущих подряд элементах которых Fun (предикат от двух переменных) возвращает true. group_by(fun(X, Y) <- X =< Y end, [1,2,4,3,2,5]) => [[1,2,4], [3], [2,5]]
24	1. Реализуйте функцию min_positive_number(List), которая возвращает минимальное положительное число, входящее в List. Если положительных чисел нет, функция должна вернуть атом error. min_positive_number([3,a,false,-3,1]) => 1 2. Не смотря на определение в модуле lists стандартной библиотеки, реализуйте функцию zipwith(Fun, List1, List2). Возвращается список значений Fun (функции от двух аргументов) на аргументах, взятых из списков List1 и List2. В случае разных длин списков функция должна выкинуть исключение. zipwith(fun(X, Y) -> {X, Y} end, [1,2,3], [4,5,6]) => [{1,4}, {2,5}, {3,6}] 3. Реализуйте функцию iteratemap(F, X0, N), которая возвращает список длины N, состоящий из результатов последовательного применения F к X0. iteratemap(fun(X) -> X*2 end, 1, 4) => [1, 2, 4, 8] 4. Реализуйте функцию diff(F, DX), которая принимает функцию F от одного аргумента и шаг DX, и возвращает функцию одного аргумента: приближение к производной функции F. F1 = diff(fun(X) -> X*X end, 0.001), F1(1.0) => 2.000 (приближенно) Попробуйте вспомнить (или найти) формулу, которая даёт лучшее приближение для производной, чем (F(X + DX) - F(X))/DX, но она тоже принимается.
25	1. Реализуйте функцию sum_neg_squares(List), которая возвращает сумму квадратов всех отрицательных чисел в списке List. sum_pos_squares([-3,a,false,-3,1]) => 18 2. Не смотря на определение в модуле lists стандартной библиотеки, реализуйте функцию dropwhile(Pred, List). Она возвращает то, что остаётся от списка List после отбрасывания начальных элементов, на которых Pred возвращает true. dropwhile(fun(X) -> X < 10 end, [1,3,9,11,6]) => [11, 6]

	3. Реализуйте функцию antimap(ListF, X), которая принимает список функций одного аргумента ListF и значение X, и возвращает список результатов применения всех функций из ListF к X. antimap([fun(X) -> X + 2 end, fun(X) -> X*3 end], 4) => [6, 12] 4. Реализуйте функцию solve(Fun, A, B, Eps), которая находит приближённо (с ошибкой не больше Eps) корень уравнения Fun(X) = 0 на отрезке [A, B] или точку разрыва, в которой Fun меняет знак. Можно считать, что F(A) <= 0 <= F(B) (как известно, в таком случае корень заведомо существует). Проще всего это сделать, деля отрезок пополам и смотря, на концах какой половины различаются знаки Fun. solve(fun(X) -> X*X - 2 end, 0, 2, 0.001) => 1.414 (приближенно)

Дополнительные задания:

5. Реализуйте функцию for(Init, Cond, Step, Body), которая работает как цикл for (I = Init; Cond(I); I = Step(I)) { Body(I) } в C-подобных языках:

поддерживается "текущее значение" I. В начале это Init.

на каждом шаге проверяется, выполняется ли условие Cond(I).

если да, то вызывается функция Body(I). Потом вычисляется новое значение как Step(I) и возвращаемся к проверке Cond.

если нет, то работа функции заканчивается.

6. Реализуйте функцию sortBy(Comparator, List), которая сортирует список List, используя Comparator для сравнения элементов. Comparator(X, Y) возвращает один из атомов less (если X < Y), equal (X == Y), greater (X > Y) для любых элементов List. Можете использовать любой алгоритм сортировки, но укажите, какой именно. Сортировка слиянием очень хорошо подходит для связных списков.

Критерии оценивания

	Задание сдано в срок	**Задание сдано позже**
Задача 1 выполнена верно		
Задача 2 выполнена верно		
Задача 3 выполнена верно		
Задача 4 выполнена верно		
Верно выполнено дополнительное задание 5		
Верно выполнено дополнительное задание 6	1	0,5
Итого	7	3,5