1. Установите интерпретатор **ghci**.
2. Вычислите в интерпретаторе следующие выражения:
   1. 1 + 1
   2. True == False
   3. 5 + 7 \* (-7 + 1)
   4. not True
   5. False && True || True
   6. 2 \* 2 /= 4
   7. succ 9
   8. max 5 15 + 4 + min 9 (-1)

Также определите в интерпретаторе определите их тип.

1. Укажите эквивалент в языке Haskell следующего C/Java/Python выражения:

**combine(prettify(lawn),construct(house,concrete))**

* 1. combine prettify (lawn) construct (house concerete)
  2. combine (prettify lawn (counstruct house concrete))
  3. combine (prettify lawn) (construct house concrete)

1. Какой эквивалент C/Java/Python для Haskell выражения:

**send metric (double population + increase)**

* 1. send(metric(double(population+increase)))
  2. send(metric(double(population)+increase))
  3. send(metric,double(population)+increase)
  4. send(metric,double(population+increase))

1. Используя только функцию **max** из модуля **Prelude**:

max :: Int -> Int -> Int

max x y = `если x > y, то x, иначе y`

напишите выражение, которые находит максимальное число среди списка чисел:

0, -2, 99, 11, -5, 6, 1521

1. Создайте модуль **FACTORIAL**:

module FACTORIAL where

fac :: Int -> Int

fac 0 = 1

fac n = n \* fac (n - 1)

Загрузите модуль в интерпретатор, использую команду:

:load `имя\_файла\_модуля`

Команда **:load** будет искать указанный файл в текущем каталоге.

1. Вычислите **fac 5**
2. Используя **условное выражение** напишите функцию, которая возвращает **0**, если её аргумент больше **5**, иначе возвращает **1**.
3. Напишите функцию **isZero**, которая получает значение типа **Integer** и возвращает **True**, если полученное значение равно **0** и **False** иначе. Приведите сигнатуру функции **isZero**.

*Примечания:*

* 1. Используйте только сравнение с образцом, не используйте функцию сравнения.
  2. Тип логических значений в Haskell – это **Bool**.
  3. Сигнатура в Haskell имеет вид:

имя\_определения :: тип определения

1. Реализуйте функцию **max’**, которая находит максимальное число из 3-х заданных чисел.
2. Напишите выражение, которое конкатенирует строки: **“Hello”**, **“ ”**, **“world”**, **“!”**.
3. Используя команду **:info (++)** определите ассоциативность оператора (++).
   1. левоассоциативная;
   2. правоассоциативная;
   3. ассоциативная.
4. Используя команды **:info (++)** и **:info (^)** определите какой оператор имеет больший приоритет **(++)** или **(^)**.
   1. (++)
   2. (^)
5. Расставьте скобки в выражении **2^4 + 6 \* 3^2 – 1**, в соответствии с приоритетом операций в Haskell.
6. Используя команду **:t `выражение`** определите тип
   1. функции **max**;
   2. функции **head;**
   3. выражения **2 + 2**;
   4. оператора **(^)**.
7. Запишите все операторы в выражениях в **префиксной** нотации:
   1. 1 + 2
   2. 4^2 – 1
   3. 5 == 6 - 1
8. Запишите все операторы выражениях в **инфиксной** нотации:
   1. max 5 (-1)
   2. min (min 5 2) (max (-1) 4)
9. Запишите список чисел
   1. **[2,3,4,5]** без перечисления всех элементов списка.
   2. **[1,3,5,7,9]** без перечисления всех элементов списка.
10. Напишите выражение, которое вычисляет из списка **[1, 5, -3, 3, 5]**:
    1. голову списка;
    2. хвост списка;
    3. новый список, который состоит из первых трёх элементов списка;
    4. новый список, который состоит из квадратов элементов списка (используйте, генераторы списка)
    5. новый список, который содержит умноженные на 2 элементы списка, которые при умножении на три больше 8.
11. Напишите функцию, которая для данного числа **n** создаёт список всех попарных сумм чисел от **1** до **n** (Т.е. **[1+1, 1+2, 1+3,…,1+n, 2+1, 2+2,…,n+n]** всего **n\*n** элементов)

*Примечание*: Используйте генераторы списков.

1. Перепишите функцию используя охранные выражения:

f :: Int -> Int -> Int

f x y =

if x > y + 1 then 0 else

if x > y - 2 then 1 else

if x \* 2 == y + 1 then 2 else

if x + 10 == y - 2 then 3 else 4

Используйте конструкцию **where**, чтобы ввести локальные определения для подвыражений **y + 1** и **y – 2**

1. Используя сравнение с образцом реализуйте функции:

fst’ :: (a, b, c)-> a

snd’ :: (a, b, c)-> b

thd’ :: (a, b, c)-> c

1. Напишите функцию **distance**, которая получает четыре значения типа **Dobule**: **x1**, **y1**, **x2**, **y2** и возвращает евклидовое расстояние между двумя точками **(x1, y1)** и **(x2, y2)**.
2. Напишите функцию **add\_and\_double**, которая складывает два числа, а затем умножает результат сложения на два.
3. Используя функцию **add\_and\_double** реализуйте инфиксный оператор **+\***, который делает тоже самое, что и функция **add\_and\_double**.
4. Напишите функцию **solve\_quadratic\_equation** от трёх аргументов (**a**, **b** и **c**), которые являются коэффициентами квадратного уравнения **x2 + b x + c = 0**. **a**, **b**, **c**, и **x** должны быть типа **Double**. Результатом должна быть пара корней уравнения (кортеж двух значений). В случае комплексных корней, применение функции **sqrt** (находит квадратный корень) к отрицательному числу вернёт **NaN** (Not A Number). Используйте **let**-выражение или конструкцию **where** для определения квадратного корня дискриминанта уравнения.
5. Определите функцию **double\_factorial**, которая получает целое число n и вычисляет произведение всех факториалов начиная с нуля и включая n. Используйте **let**-выражения или конструкцию **where** для определения функции факториала.
6. Напишите функцию:

**ilog3 :: Integer -> Integer**

Функция должна вычислять сколько раз можно полученное значение аргумента разделить на **3** (с округлением в меньшую сторону) прежде, чем будет получен **0**.

Пример:

ilog3 20 ==> 3 т.к.

-- 20/3 = 6.66 (округляется до 6)

-- 6/3 = 2

-- 2/3 = 0.666 (округляется до 0)

ilog3 2 ==> 1

ilog3 7 ==> 2

Используйте функцию целочисленного деления **div** для деления целых чисел.