Дементьев Никита Евгеньевич

Рассмотрим тему **lists comprehention.**

```
1> List = [1,2,b,c,g,6,5].
[1,2,b,c,g,6,5]
2> [X || X <-List, X>0].
[1,2,b,c,g,6,5]
3> [X || X <-List, X>2].
[b,c,g,6,5]
4> [X || X <-List, is_integer(X), X>2].
[6,5]
```

Рисунок 1 – пример 1

В качестве примера возьмем и зададим любой список и попробуем его отфильтровать. Во второй строчке берется список X, а дальше как в математике, X принадлежит списку List, а также X больше 0 или 2, как например, в третьей строчке, помимо того можно добавить еще фильтров и как в строке 4 можно убрать все буквы и оставить только числа.

```
7> [X * 3 || X <-List, is_integer(X), X>2].
[18,15]
```

Рисунок 2 – пример 2

Также можно к списку добавить действий, например: умножение, деление и т.д. На выходе мы получаем список, как на рис.2 умноженный на 3, но данное действие не сработает если убрать фильтр is_integer(X), так как нельзя атом умножить на число.

```
9> List1 = [1,2].
[1,2]
10> List2 = [4,b].
[4,b]
11> List3 = [7,nn].
[7,nn]
12> [{X,Y,Z} || X <- List1, Y <-List2, Z<-List3].
[{1,4,7},
 {1,4,nn},
 {1,b,7},
 {1,b,nn},
 {2,4,7},
 {2,4,nn},
 {2,b,7},
 {2,b,nn}]
```

Рисунок 3 – пример 3

lists comprehention позволяет работать с несколькими списками, в данном случае выведется несколько кортежей. Которые перебираются по порядку.

```
13> Max = 20.
20
14> Lengthes = lists:se
search/2  seq/2     seq/3
14> Lengthes = lists:seq(1, Max).
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
15> [{X,Y,Z} || X<-Lengthes,Y<-Lengthes,Z<-Lengthes,X<Y,X*X+Y*Y=:= Z*Z].
[{3,4,5},{5,12,13},{6,8,10},{8,15,17},{9,12,15},{12,16,20}]
```

Рисунок 4 – пример 4

В четвертом примере показано, как легко и компактно в одну строку можно найти все пифагоровы тройки в списке, состоящем из 20 чисел и без повторений с помощью фильтров.

В качестве своего примера выведем из списка пользователей мужчин, имена и пользователей, ID которых больше двух и меньше 5.

```
17> lists1:users().
[{user,1,"Sasha",male,23},
 {user,2,"Misha",male,27},
 {user,3,"Lida",female,35},
 {user,4,"Bob",male,18},
 {user,5,"Lera",female,21}]
```

Рисунок 5 – список пользователей

```
-spec users1() -> list(gender).
users1() ->
        [User || {user, _,_, Gender, _} = User <- users(), Gender =:=male].

-spec users2() -> list(name).
users2() ->
        [Name || {user, _,Name,_, _} <- users()].

-spec users3() -> list(id).
users3() ->
        [User || {user, Id,_,_, _} = User <- users(),Id > 2 andalso Id < 5 ].
```

Рисунок 6 – условия для вывода по условию из списка

В качестве генератора используем метод pattern matching, для того чтобы определить какого пола пользователь, имя или id.

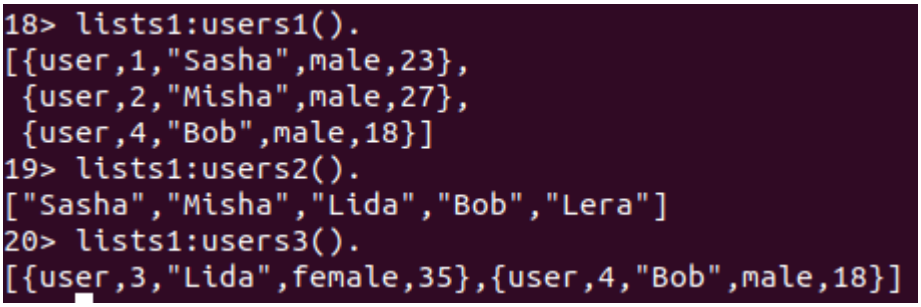

Рисунок 7 – результаты вывода

В 18 строке вывелся список состоящий из мужчин, в 19 строке имена пользователей по порядку и в 20 строке пользователи, у которых id > 2 и не превышает 5.

Программа запускается командой **c(lists1).** Также можно запустить тесты командой **make test3.**

Командой **make test** можно запустить тесты факториала в виде обычной рекурсии и хвостовой. Разница вида рекурсии в том, что в хвостовой вводится аккумулятор, который сохраняет в себе каждый круг рекурсии, в следствии чего экономя память, а в обычной рекурсии пока все не вычислится программа будет занимать память.