

Выполнил: Белкин Никита

Задание 1.

```
Erlang
File Edit Options View Help
Erlang/OTP 25 [erts-13.1.5] [source] [64-bit] [smp:4:4] [ds:4:4:10] [async-threads:1] [jit:ns]

Eshell V13.1.5 (abort with ^G)
1> rd(person, {id, name, age, gender}).
person
2> Persons = [#person{id = 1, name = "Bob", age = 23, gender = male}, #person{id = 2, name = "Kate", age = 20, gender = female}, #person{id = 3, name = "Jack", age = 34, gender = male}, #person{id = 4, name = "Nata", age = 54, gender = female}].
[#person{id = 1,name = "Bob",age = 23,gender = male},
 #person{id = 2,name = "Kate",age = 20,gender = female},
 #person{id = 3,name = "Jack",age = 34,gender = male},
 #person{id = 4,name = "Nata",age = 54,gender = female}]
3> [FirstPerson | _] = Persons.
[#person{id = 1,name = "Bob",age = 23,gender = male},
 #person{id = 2,name = "Kate",age = 20,gender = female},
 #person{id = 3,name = "Jack",age = 34,gender = male},
 #person{id = 4,name = "Nata",age = 54,gender = female}]
4> FirstPerson.
#person{id = 1,name = "Bob",age = 23,gender = male}
5> [_ , SecondPerson, _ , _] = Persons.
[#person{id = 1,name = "Bob",age = 23,gender = male},
 #person{id = 2,name = "Kate",age = 20,gender = female},
 #person{id = 3,name = "Jack",age = 34,gender = male},
 #person{id = 4,name = "Nata",age = 54,gender = female}]
6> SecondPerson.
#person{id = 2,name = "Kate",age = 20,gender = female}
7> SecondName = SecondPerson#person.name.
"Kate"
8> SecondAge = SecondPerson#person.age.
20
9> [_ , #person{name = SecondName, age = SecondAge} | _Rest] = Persons.
[#person{id = 1,name = "Bob",age = 23,gender = male},
 #person{id = 2,name = "Kate",age = 20,gender = female},
 #person{id = 3,name = "Jack",age = 34,gender = male},
 #person{id = 4,name = "Nata",age = 54,gender = female}]
10> SecondName.
"Kate"
11> SecondAge.
20
12> Persons.
[#person{id = 1,name = "Bob",age = 23,gender = male},
 #person{id = 2,name = "Kate",age = 20,gender = female},
 #person{id = 3,name = "Jack",age = 34,gender = male},
 #person{id = 4,name = "Nata",age = 54,gender = female}]
13> SecondPerson#person{age = 21}.
#person{id = 2,name = "Kate",age = 21,gender = female}
14> Persons.
[#person{id = 1,name = "Bob",age = 23,gender = male},
 #person{id = 2,name = "Kate",age = 20,gender = female},
 #person{id = 3,name = "Jack",age = 34,gender = male},
 #person{id = 4,name = "Nata",age = 54,gender = female}]
15> SecondPerson.
#person{id = 2,name = "Kate",age = 20,gender = female}
16> █
```

Сначала создадим список records, который содержит данные о 4х человек.

Затем выполняем команды:

1. [FirstPerson | _] = Persons.

Значение FirstPerson будет равно первому элементу списка Persons, т.е. #person{id = 1, name = "Karl", age = 87, gender = male}.

2. `[_ , SecondPerson, _, _] = Persons.`
Значение `SecondPerson` будет равно второму элементу списка `Persons`, т.е. `#person{id = 2, name = "Kate", age = 86, gender = female}`.
3. `SecondName = SecondPerson#person.name.`
`SecondAge = SecondPerson#person.age.`
`SecondName` будет равно значению поля `name` второго элемента списка `Persons`, т.е. `"Kate"`.
`SecondAge` будет равно значению поля `age` второго элемента списка `Persons`, т.е. `86`.
4. `[_ , #person{ name = SecondName, age = SecondAge } | _Rest] = Persons.`
`SecondName.`
`SecondAge.`
Значение `SecondName` и `SecondAge` уже были определены в предыдущем шаге.
Эта команда проверяет, что второй элемент списка `Persons` содержит поля `name` и `age`, равные `SecondName` и `SecondAge` соответственно, и присваивает этот элемент к `#person{ name = SecondName, age = SecondAge }`.
Значение `SecondName` будет равно `"Kate"`.
Значение `SecondAge` будет равно `86`.
5. `Persons.`
Команда возвращает список `Persons` без изменений.
6. `SecondPerson#person{ age = 21 }.`
Эта команда создает новый элемент типа `#person` с полем `age`, равным `21`, на основе второго элемента списка `Persons`. Однако, эта команда не изменяет список `Persons` и `SecondPerson`.

Задание 2.

```
Erlang
File Edit Options View Help
Eshell V13.1.5 (abort with ^G)
1> Persons = [{#id => 1, name => "Bob", age => 23, gender => male}, #{id => 2, name => "Kate", age => 20, gender => female}, #{id => 3, name => "Jack", age => 34, gender => male}, #{id => 4, name => "Nata", age => 54, gender => female}].
2> [FirstPerson | _] = Persons.
3> FirstPerson.
4> [_ , _ , #{name := Name, age := Age}, _] = Persons.
5> Name.
6> Age.
7> [_First, _Second, #{name := Name, age := Age} | _Rest] = Persons.
8> Name.
9> Age.
10> Persons.
11> FirstPerson#{age := 24}.
12> Persons.
13> FirstPerson.
14> FirstPerson#{address := "Mira 31"}.
** exception error: bad key: address
    in function maps:update/3
       called as maps:update(address,"Mira 31",
                             #{age => 23,gender => male,id => 1,name => "Bob"})
    *** argument 3: not a map
    in call from erl_eval:'-expr/6-fun-0-/2' (erl_eval.erl, line 309)
    in call from lists:foldl/3 (lists.erl, line 1350)
```

Сначала создадим список maps, который содержит данные о 4х человек.

Затем выполняем команды:

1. `[FirstPerson | _] = Persons.`
FirstPerson получит значение первого элемента списка Persons, т.е. `#{id => 1, name => "Bob", age => 23, gender => male}`.
2. `[_ , _ , #{name := Name, age := Age}, _] = Persons.`
Значение Name будет "Jack", так как это имя третьего человека в списке Persons.
Значение Age будет 34, так как это возраст третьего человека в списке Persons.
3. `[_First, _Second, #{name := Name, age := Age} | _Rest] = Persons.`
В Name и Age также присваиваются значения "Jack" и 34 соответственно. Данная команда завершилась успешно, потому что первые два элемента списка не участвуют в привязке переменных, а третий элемент является map с полями name и age, которые удовлетворяют шаблону `#{name := Name, age := Age}`.

4. Persons.

Список Persons не изменился.

5. FirstPerson#{age := 24}.

Выполнение FirstPerson#{age := 24} создаст новую map, отличную от исходной, но переменная FirstPerson не изменилась. Список Persons также не изменился.

6. FirstPerson#{address := "Mira 31"}.

При попытке выполнить команду возникает исключение "bad key: address", так как поле address отсутствует в исходной map.

Чтобы добавить поле address в FirstPerson, можно сделать следующее:

```
15> FirstPersonWithAddress = maps:put(address, "Mira 31", FirstPerson).
#{address => "Mira 31",age => 23,gender => male,id => 1,
  name => "Bob"}
16> Persons.
[#{age => 23,gender => male,id => 1,name => "Bob"},
 #{age => 20,gender => female,id => 2,name => "Kate"},
 #{age => 34,gender => male,id => 3,name => "Jack"},
 #{age => 54,gender => female,id => 4,name => "Nata"}]
17> FirstPerson.
#{age => 23,gender => male,id => 1,name => "Bob"}
18> FirstPersonWithAddress.
#{address => "Mira 31",age => 23,gender => male,id => 1,
  name => "Bob"}
```

Здесь мы добавили поле address со значением "Mira 31" в FirstPerson, и результат будет сохранен в FirstPersonWithAddress. Исходный список Persons и FirstPerson не изменится, так как map является неизменяемой структурой данных.