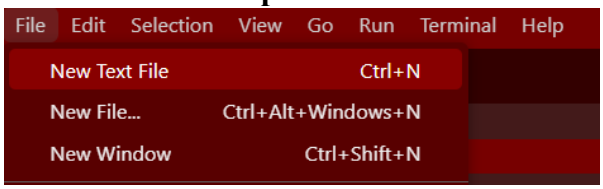
Упражнение 7 - ЕФП

Задача 1. Създайте вашата първа програма „hello world“.

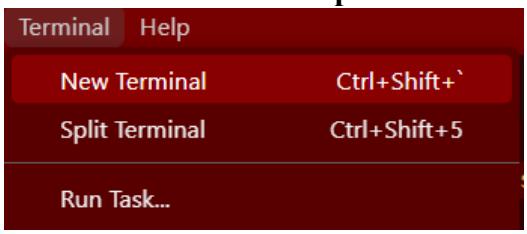
1. **От меню File изберете New Text File**



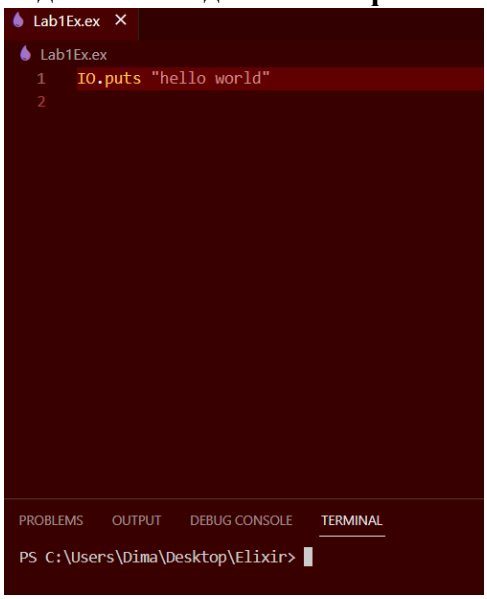
2. **Запишете следния код във файла:**

```
IO.puts "hello world"
```

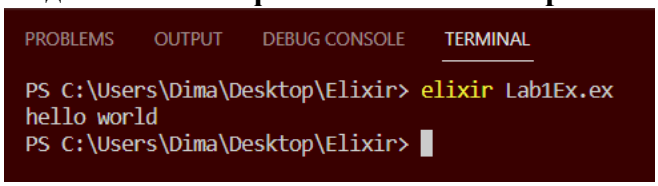
3. **Запазете файла с име: Lab1Ex. Разширението се добавя автоматично *.ex**
4. **От меню Terminal изберете:**



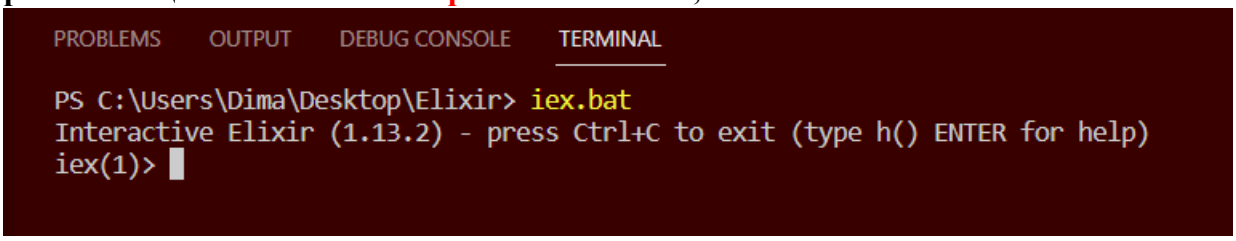
5. **Задайте пътя до вашият файл:**



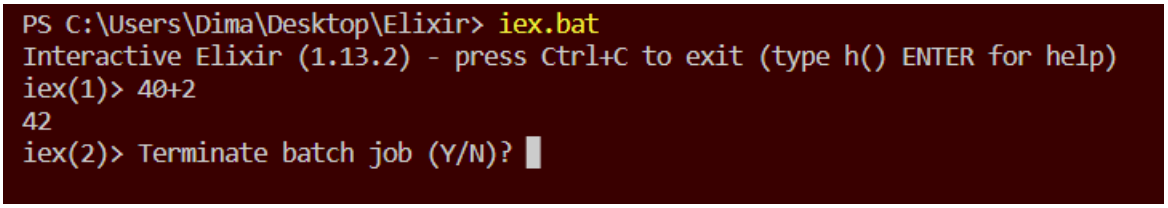
6. **За да изпълните файла запишете в терминала: elixir името_на_файла.ex**



Задача 2. Създайте вашата първа програма, като използвате интерактивния режим за работа. За целта запишете в терминала - iex.bat, ако сте на Windows.



А изход от този режим Ctrl+C



Структури в Elixir

Структурите са разширения, изградени върху Map, които осигуряват проверка по време на компилация и настройка по подразбиране.

1. Дефиниране на структура

За дефиниране на структурата се използва конструкцията defstruct:

```
defmodule User do
  defstruct name: "John", age: 27
end
```

Списъкът с ключови думи, използвани с defstruct, определя какви полета ще има структурата заедно със стойностите по подразбиране. Структурите вземат името на модула, в който са дефинирани, в случая това име е User.

Създаване на потребителски структури става по подобен на map-а начин:

```
new_john = %User{})
ayush = %User{age: 20, name: "Ayush"}
megan = %User{name: "Megan"})
%User{age:27,name:"Megan"}
```

Структурите предоставят гаранции по време на компилация, че само полета, дефинирани с defstruct, могат да съществуват в структурата. По този начин не могат да бъдат дефинирани собствени полета, след като вече е създадена структурата в модула.

```
% User{oops::field}
** (CompileError) iex:3: unknown key:oops  for struct User
```

2. Достъп и актуализиране на структури

Осъществяването на достъп и актуализиране на полетата на структурата става по подобен начин както при map-а. Същите техники (и синтаксис) се прилагат и за структурите.

Пример: Да се актуализира потребителя в структурата User.

```
defmodule User do
  defstruct name: "John", age: 27
end
john = %User{}

#john right now is: %User{age: 27, name: "John"}
#To access name and age of John,
IO.puts(john.name)

IO.puts(john.age)
```

Изход:
John
27

Актуализиране на стойностти в структурата става чрез прилагане на същата процедура, която се използва при map-а:

meg = %{john | name: "Meg"}

Структурите могат да се използват при съвпадение на шаблони, за съвпадение по стойност със специфични ключове и за да се гарантира, че съответстващата стойност е структура от същия тип като съответстващата стойност.

3. Структура - особености

Дефиницията на структура User е:

```
iex> defmodule User do
...>   defstruct name: "John", age: 27
...> end
```

Създаването на потребителските структури:

```
iex> %User{}
%User{age: 27, name: "John"}

iex> %User{name: "Meg"}
%User{age: 27, name: "Meg"}
```

Структурите предоставят гаранции по време на компилация, че само полетата, дефинирани чрез **defstruct**, ще могат да съществуват в структура:

```
iex> %User{oops: :field}
** (CompileError) iex:3: unknown key :oops for struct User
```

Достъпът и актуализирането на полетата на структурата използва сходна на map-а техника.

```
iex> john = %User{}
%User{age: 27, name: "John"}

iex> john.name
"John,,

iex> meg = %{john | name: "Meg"}

%User{age: 27, name: "meg"}
iex> %{meg | oops: :field}
** (ArgumentError) argument error
```

Когато използва синтаксиса за актуализация (|), виртуалната машина е наясно, че няма да се добавят нови ключове към структурата, което позволява на map-а отдолу да споделя своята структура в паметта. В горния пример и John, и Meg споделят една и съща ключова структура в паметта.

Структурите могат също да се използват при съвпадение на шаблони, както за съвпадение на стойността на конкретни ключове, така и за гарантиране, че съответстващата стойност е структура от същия тип като съответстващата стойност.

```
iex> %User{name: name} = john
%User{age: 27, name: "John"}

iex> name
"John"

iex> %User{} = %{}
** (MatchError) no match of right hand side value: %{}
```

4. Сходство на struct и map

В горния пример съвпадението на шаблони работи, защото под структурата има обикновен map с фиксиран набор от полета. Като map, структурата съхранява „специално“ поле с име __struct__, което съдържа името на структурата:

```
iex> is_map(john)
true

iex> john.__struct__
User
```

Забележете, че беше посочена, като структурата обикновен map, защото нито един от протоколите, внедрени за map, не е наличен за структура. Например, не може да се изброява или да се осъществява достъп до структура:

```
iex> john = %User{}
```

```
%User{age: 27, name: "John"}

iex> john[:name]
** (Protocol.UndefinedError) protocol Access not implemented for %User{age: 27, name:
"John"}

iex> Enum.each john, fn({field, value}) -> IO.puts(value) end
** (Protocol.UndefinedError) protocol Enumerable not implemented for %User{age: 27, name:
"John"}
```

Структурата също не е речник и следователно не може да се използва с функциите от модула Dict:

```
iex> Dict.get(%User{}, :name)
** (UndefinedFunctionError) undefined function: User.fetch/2
```

Въпреки това, тъй като структурите са само map, те работят с функциите от модула map:

```
iex> kurt = Map.put(%User{}, :name, "Kurt")
%User{age: 27, name: "Kurt"}

iex> Map.merge(kurt, %User{name: "Takashi"})
%User{age: 27, name: "Takashi"}

iex> Map.keys(john)
[:__struct__, :age, :name]
```

Задачи:

Задача 1: Да се създаде структура Book с полета: isbn, author, title, description, price, genre. Да се създаде функция за редактиране на данни. Да се създаде нов обект на структурата. Да се направи Update на стойностите на структурата. Да се изведе резултата.

```
defmodule Book do
  # Деклариране на структура
  defstruct isbn: " 9786192406202",
            author: "Rudyard Kipling",
            title: "The Jungle Book",
            description: "It is about the Jungle",
            price: 22.41,
            genre: "novel"

# Редактиране на данни
  def callstruct() do
    IO.puts("========Default стойности на структурата==========")
    # Вмъкване на данни
    books = %Book{}
```

```
    IO.puts(books.isbn)
    IO.puts(books.author)
    IO.puts(books.title)
    IO.puts(books.description)
    IO.puts(books.price)
    IO.puts(books.genre)
  end
```

2. Създаване на нов обект на структура:

```
def makeObj(a, b, c, d, e, f) do
    # Въвеждане на данни в структурата
    books  = %Book{isbn: a, author: b, title: c, description: d, price: e, genre: f}
    IO.puts("==========Въведете стойности==========")
    IO.puts(books.isbn)
    IO.puts(books.author)
    IO.puts(books.title)
    IO.puts(books.description)
    IO.puts(books.price)
    IO.puts(books.genre)
```

3. Update на стойностите на структурата:

```
# Редактиране на данни в структурата
    # Update Fields of object
    IO.puts("==========След Update==========")

upd = %{
      books
      | isbn: "Update_ISBN",
        author: "Update_Author",
        title: "Update_Title",
        description: "Update_Description",
        price: 99.99,
        genre: "Update_Genre"
    }
    IO.inspect(upd)
  end
end
```

4. Извеждане на резултата:

```
isbn = IO.gets(„Enter ISBN: ") |> String.trim()
author = IO.gets("Enter Author: ") |> String.trim()
title = IO.gets("Enter Title: ") |> String.trim()
description = IO.gets("Enter description: ") |> String.trim()
price = IO.gets("Enter price : ") |> String.trim() |> String.to_float()
genre = IO.gets("Enter genre: ") |> String.trim()
IO.puts(Book.makeObj({isbn, author, title, description, price, genre}))
```

Задача 2: Да се създаде структура Employees с полета: firstname, middlename, surname, job, workingdays, vacationday. Да се въведат 3-ма служителя. Да се изчисли месечната заплата monthlysalary на всеки от служителите. Да се сортират служителите по име и по заплата. Данните да се записват в текстов файл, с опция за допълване на нови служители.

```
defmodule Tech do
  @enforce_keys [:firstname, :middlename, :surname, :job, :workingdays, :vacationday]
  defstruct @enforce_keys ++ [:monthlysalary]
  defmodule Employees do
    def listofemployees() do
      employee1 = %Tech{
        firstname: "Ivan",
        middlename: "Petrov",
        surname: "Iliev",
        job: "Front-end programmer",
        workingdays: 30,
        vacationdays: 3
      }
      employee2 = %Tech{
        firstname: "Ilian",
        middlename: "Dimitrov",
        surname: "Todorov",
        job: "Database analytics",
        workingdays: 40,
        vacationday: 4
      }
employee3 = %Tech{
        firstname: "Radoslava",
        middlename: "Dragostinova",
        surname: "Kostova",
        job: "Back-end programmer",
        workingdays: 60,
        vacationday: 6
      }
      employee1 = %{
        employee1
        | monthlysalary: employee1.workingdays * 21
      }
      employee2 = %{
        employee2
        | monthlysalary: employee2.workingdays * 21
      }
      employee3 = %{
        employee3
        | monthlysalary: employee3.workingdays * 21
      }

employees = [employee1, employee2, employee3]
```

```
    employees =
      employees
      |> Enum.sort_by(&Map.fetch(&1, :name))
      |> Enum.sort_by(&Map.fetch(&1, :job))
    {:ok, file} = File.open("employees.txt", [:append])
    for x <- 0..2 do
      IO.binwrite(
        file,
        "#{Enum.at(employees, x).firstname}
        #{Enum.at(employees, x).middlename}
        #{Enum.at(employees, x).surname}
        #{Enum.at(employees, x).monthlysalary}
        #{Enum.at(employees, x).job}
        #{Enum.at(employees, x).workingdays}
        #{Enum.at(employees, x).vacationday}\n"
      )
    end
  end
 end
end
# Entry employess
#  def enterEmployees() do
#  end
# end
```

Elixir - File IO

1) Отваряне на файл

За да се отвори файл се използват 2 функции:

- **{:ok, file} = File.open("newfile")**
- **file = File.open!("newfile")**

Разликата между функцията File.open и функцията File.open!().

Функцията **File.open** винаги връща кортеж. Ако файлът е отворен успешно, той връща първата стойност в кортежа като **:ok** , а втората стойност е литерал от тип **io_device.** Ако е причинена грешка, той ще върне кортеж с първа стойност **:error** и втора стойност **причина за възникването.**

Функцията **File.open!()** от друга страна ще върне **io_device**, ако файлът е отворен успешно, в противен случай ще предизвика грешка.

За да се отвори файл **само за четене и в режим на кодиране utf-8**, се използва:

- **file = File.open!("newfile", [:read, :utf8])**

2) Запис във файл

Има 2 начина за запис във файл. Първият използва функцията за запис от модула File:

- **File.write("newfile", "Hello")**

Този начин на запис **не трябва да се използва, ако се правят няколко записа в един и същ файл**. Всеки път, когато се извиква тази функция, се отваря файлов дескриптор и се създава нов процес за запис във файла.

Ако се правят няколко записа в цикъл, се отваря файла чрез File.open и се пише в него, като се използват методите в IO модула.
Пример:

```
#Open the file in read, write and utf8 modes.
file = File.open!("newfile_2", [:read, :utf8, :write])

#Write to this "io_device" using standard IO functions
IO.puts(file, "Random text")
```

Могат да се да използват и други методи на IO модул, като **IO.write** и **IO.binwrite** за запис във файл отворени като **io_device.**

3) Четене от файл

Има 2 начина **за четене от файл.** Първият, използва функцията **read** от модула File.

- **IO.puts(File.read("newfile"))**

В случая се получава кортеж с първия елемент **:ok** и втори, като съдържание на **newfile.**

Също така може да се използва **File.read! функция, която да ни върне** съдържанието на файловете.

4) Затваряне на файл

След приключване на използването на файл отворен чрез функцията File.open тя се затваря с функцията File.close:

- **File.close(file)**

Задължителни задачи за самостоятелна работа

1. Задача със структура:

Да се дефинира структура Student, в която се съхраняват данни за име, факултетен номер и среден успех. Да се създадат функции за добавяне на нов запис, изтриване на запис по зададен критерий. Да се създаде списък от студенти и да се разпечата неговото съдържание. Данните от структурата да се записват успоредно и в текстов файл, като се извършва предварително проверка за въведен вече факултетет номер.

За реализиране на структурата да се използва меню:

1. Enter new student
2. Delete student
3. Show existing students
4. Update Student
5. Exit