

Curso: Engenharia de Software		Série: 6S	Turma: A	Turno: Noite
Professor(a): Thiago Bussola da Silva		Horário:		
Acadêmico (a): João Vitor Polloni Cordeiro				RA: 21120606-2
Disciplina: Paradigmas de Programação				Data: 26/09/2023
Prova	Prova Prática	Atividades de estudo programadas (AEP)	Prova integrada	Nota final do bimestre

INSTRUÇÕES PARA REALIZAÇÃO DA PROVA:

- ⇒ Os dados do cabeçalho deverão ser preenchidos com letra maiúscula. E as questões deverão ser respondidas com letra legível.
- ⇒ É vedado, durante a prova, o porte e/ou o uso de aparelhos sonoros, fonográficos, de comunicação ou de registro eletrônico ou não, tais como: notebooks, celulares, tablets e similares.
- ⇒ A prova é individual e sem consulta, deverá ser respondida a caneta azul ou preta. Prova escrita a lápis não dá direito à revisão. Não é permitido o uso de corretivo.
- ⇒ É obrigatória a permanência do acadêmico 1 (uma) hora em sala de aula após o início da prova.
- ⇒ Não será permitida a entrada na sala de aula após 10 minutos do início da prova.
- ⇒ É obrigatória a assinatura da lista de presença impressa na qual constam RA, nome e curso.
- ⇒ O valor de cada questão está ao lado da mesma.
- ⇒ Todas as respostas devem constar no espaço destinado e autorizado pelo professor, à resposta.
- ⇒ Em caso de qualquer irregularidade comunicar ao Professor ou fiscal de sala.
- ⇒ Ao término da prova, levante o braço e aguarde o atendimento do professor ou do fiscal.

1ºbim.		2ºbim.		1ªsub.		2ªsub.		1ºsem.		2º sem.	
--------	--	--------	--	--------	--	--------	--	--------	--	---------	--

QUADRO PARA O PROFESSOR - REGISTRO DE NOTAS	
Questão 1	
Questões 2	
Questão 3	
Questão 4	
Questão 5	
Questão 6	
Questões 7	
Questões 8	
Questão 9	
Questão 10	

Instruções - Leia com atenção!

Preencha os campos do cabeçalho da prova

Regras para a prova.

Os únicos sites que você pode acessar para consultar suas dúvidas sobre sintaxe são:

<https://elixirschool.com/pt/lessons/basics/documentation>

<https://elixir-lang.org/docs.html>

O uso de qualquer outro site, chat GPT, Github está **proibido**, caso o aluno acesse outra fonte de pesquisa a prova será zerada.

Compiladores: Será permitido o uso de compiladores online para que você possa validar a implementação das soluções propostas para os exercícios. Você está autorizado a utilizar os seguintes compiladores:

https://www.tutorialspoint.com/execute_elixir_online.php

<https://onecompiler.com/elixir>

O uso do **Replit não está autorizado** e caso o aluno acesse essa ferramenta a prova será zerada.

Você pode criar arquivos .exs para a resolução da prova e fazer o zip para enviar eles. Ou você pode copiar o código de resposta e colar abaixo da pergunta correspondente no arquivo .docx

Você pode converter sua prova para pdf ao enviar, lembre-se de enviar os arquivos .exs ou de colocar as respostas na prova para a entrega.

Caso você não entregue o arquivo .docx / pdf e os .exs (caso tenha seja de sua preferência) a prova será zerada.

Questão 1 - [1 ponto] - Explique a diferença entre funções puras e funções de ordem superior em programação funcional. Dê exemplos de cada uma.

Funções puras são aquelas que mantêm seus valores inalterados não importa o comando, sempre retorna o mesmo valor, já as funções de ordem superior são funções que chamam outras funções de variáveis ou Dão como resposta outras funções.

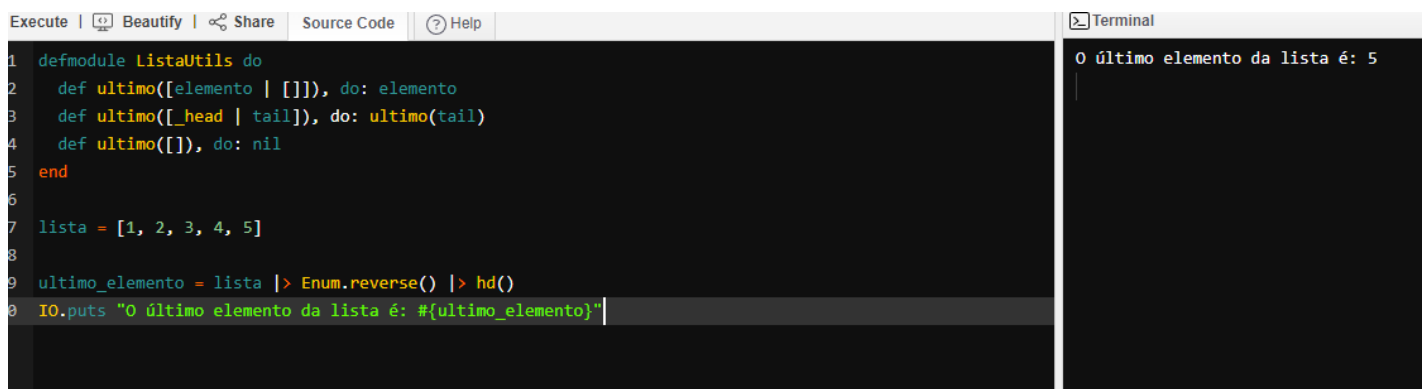
Questão 2 - [1 ponto] - Discorra sobre as vantagens da linguagem de programação elixir e em que tipo de projeto ou cenário devemos optar pelo uso dessa tecnologia.

Elixir é uma linguagem de programação muito bem otimizada com a ocorrência de poder ter funções simples e de ordem superior, trazendo maior manutenção e entendimento do Software ocasionando assim mais praticidade de manutenção e entendimento do mesmo

Questão 3 - [0,5 pontos] - Escreva uma função que verifique se um número é par.

Questão 4 - [0,5 pontos] - Implemente uma função que calcule o dobro de cada elemento em uma lista. Mesma Questão 7

Questão 5 - [0,5 ponto] - Crie uma função que retorne o último elemento de uma lista.



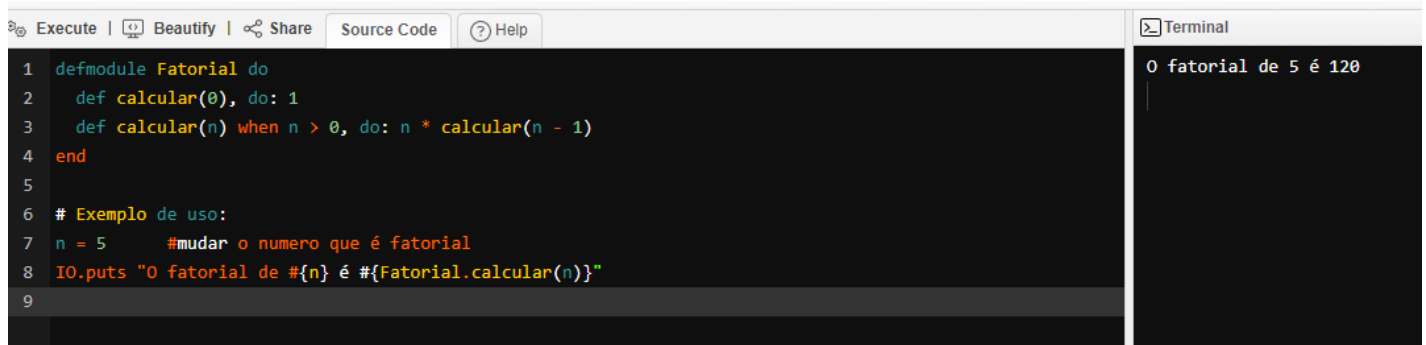
```
Execute | Beautify | Share | Source Code | Help | Terminal
1 defmodule ListaUtils do
2   def ultimo([elemento | []], do: elemento)
3   def ultimo([_head | tail]), do: ultimo(tail)
4   def ultimo([], do: nil)
5 end
6
7 lista = [1, 2, 3, 4, 5]
8
9 ultimo_elemento = lista |> Enum.reverse() |> hd()
10 IO.puts "O último elemento da lista é: #{ultimo_elemento}"
```

```
defmodule ListaUtils do
  def ultimo([elemento | []], do: elemento)
  def ultimo([_head | tail]), do: ultimo(tail)
  def ultimo([], do: nil)
end
```

```
lista = [1, 2, 3, 4, 5]
```

```
ultimo_elemento = lista |> Enum.reverse() |> hd()
IO.puts "O último elemento da lista é: #{ultimo_elemento}"
```

Questão 6 - [1 ponto] - Implemente uma função que calcule o fatorial de um número usando recursão.



```
1 defmodule Fatorial do
2   def calcular(0), do: 1
3   def calcular(n) when n > 0, do: n * calcular(n - 1)
4 end
5
6 # Exemplo de uso:
7 n = 5      #mudar o numero que é fatorial
8 IO.puts "O fatorial de #{n} é #{Fatorial.calcular(n)}"
9
```

```
defmodule Fatorial do
  def calcular(0), do: 1
  def calcular(n) when n > 0, do: n * calcular(n - 1)
end

# Exemplo de uso:
n = 5      #mudar o numero que é fatorial
IO.puts "O fatorial de #{n} é #{Fatorial.calcular(n)}"
```

Questão 7 - [1 ponto] - Escreva uma função que aplique uma função passada como argumento a cada elemento de uma lista.



```
1 defmodule FuncoesLista do
2   def aplicar_funcao(_, [], acc), do: Enum.reverse(acc)
3   def aplicar_funcao(fun, [head | tail], acc) do
4     resultado = fun.(head)
5     aplicar_funcao(fun, tail, [resultado | acc])
6   end
7 end
8
9 lista = [1, 2, 3, 4, 5]
10 fun = fn x -> x * 2 end
11 resultado = FuncoesLista.aplicar_funcao(fun, lista, [])
12 IO.inspect resultado
13
```

```
defmodule FuncoesLista do
  def aplicar_funcao(_, [], acc), do: Enum.reverse(acc)
  def aplicar_funcao(fun, [head | tail], acc) do
    resultado = fun.(head)
    aplicar_funcao(fun, tail, [resultado | acc])
  end
end
```

```

lista = [1, 2, 3, 4, 5]
fun = fn x -> x * 2 end
resultado = FuncoesLista.aplicar_funcao(fun, lista, [])
IO.inspect resultado

```

Questão 8 - [1 ponto] - Escreva uma função que filtra os elementos de uma lista com base em uma função de filtro passada como argumento.

Questão 9 - [1 ponto] - Crie uma função que gere os primeiros "n" números da sequência de Fibonacci.

Questão 10 - [2,5 ponto] - Crie uma função que calcule a média de uma turma.

O exemplo abaixo demonstra como são passadas as notas dos alunos pertencentes a uma turma.

Você deve utilizar o método `reduce` para calcular a média.

```

notas_da_turma = [
  {"Alice", [9.5, 8.0, 7.5]},
  {"João", [8.0, 7.0, 6.5]},
  {"Pedro", [9, 9.5, 9.0]},
  {"Lucas", []},
]

```

Dicas: Utilize `map` ou `flatMap` para extrair todas as notas dos alunos em uma única lista

```

1  defmodule Turma do
2    def calcular_media(notas_da_turma) do
3      {total_notas, soma_notas} =
4        Enum.reduce(notas_da_turma, {0, 0.0}, fn {_aluno, notas}, {count,
5          sum} ->
6            {count + length(notas), sum + Enum.sum(notas)}
7          end)
8
9      if total_notas > 0 do
10       soma_notas / total_notas
11     end
12   end
13 end
14
15 notas_da_turma = [
16   {"Alice", [9.5, 8.0, 7.5]},
17   {"João", [8.0, 7.0, 6.5]},
18   {"Pedro", [9, 9.5, 9.0]},
19   {"Lucas", []},
20 ]
21
22 media = Turma.calcular_media(notas_da_turma)
23 IO.puts("A média da turma é #{media}")

```

A média da turma é 8.222222222222221

```

defmodule Turma do
  def calcular_media(notas_da_turma) do
    {total_notas, soma_notas} =

```

```
Enum.reduce(notas_da_turma, {0, 0.0}, fn {_aluno, notas}, {count, sum} ->
  {count + length(notas), sum + Enum.sum(notas)}
end)
```

```
if total_notas > 0 do
  soma_notas / total_notas
end
end
end
notas_da_turma = [
  {"Alice", [9.5, 8.0, 7.5]},
  {"João", [8.0, 7.0, 6.5]},
  {"Pedro", [9, 9.5, 9.0]},
  {"Lucas", []},
]
media = Turma.calcular_media(notas_da_turma)
IO.puts("A média da turma é #{media}")
```