UNIVERSIDADE CESUMAR - UNICESUMAR

Curso: Engenharia de Software	**Série:** 6S	**Turma:** A	**Turno:** Noite
Professor(a): Thiago Bussola da Silva	**Horário:**		
Acadêmico (a):Gustavo Henrique Correia de Souza			**RA:20149350-2**
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 sempre produzem o mesmo resultado com os mesmos argumentos

---

Funções de ordem superior são funções que podem

---

receber outras funções como argumentos

---

e retornar funções como resultados.

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 funcional com ênfase em

---

concorrência e escalabilidade,Suas vantagens incluem

---

alta tolerância a falhas, programação funcional,

---

testabilidade e um ecossistema robusto.

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

```
defmodule VerificaPar do
  defpar?(n) when rem(n, 2) == 0, do: true
  defpar?(_), do: false
end
```

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

```
defmodule MathFunctions do
  def dobrar(lista) do
    Enum.map(lista, fn elemento -> elemento * 2 end)
  end
end
minha_lista = [1, 2, 3, 4, 5]
resultado = MathFunctions.dobrar_elementos(minha_lista)
IO.inspect(resultado)
```

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

```
defmodule ListFunctions do
  def ultimo_elemento([]), do: nil
  def ultimo_elemento([elemento]), do: elemento
  def ultimo_elemento([_ | tail]), do: ultimo_elemento(tail)
end
minha_lista = [1, 2, 3, 4, 5]
ultimo = ListFunctions.ultimo_elemento(minha_lista)
IO.inspect(ultimo)
```

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

```
defmodule MathFunctions do
  def calcular_fatorial(0), do: 1
  def calcular_fatorial(n) when n > 0, do: n * calcular_fatorial(n - 1)
end
IO.inspect(MathFunctions.calcular_fatorial(5))
IO.inspect(MathFunctions.calcular_fatorial(0))
```

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

```
defmodule ListFunctions do
  def aplicar_funcao(lista, funcao) do
    Enum.map(lista, funcao)
  end
end

minha_lista = [1, 2, 3, 4, 5]
funcao_dobro = fn x -> x * 2 end

resultado = ListFunctions.aplicar_funcao(minha_lista, funcao_dobro)
IO.inspect(resultado)
```

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

```
defmodule ListFunctions do
  def filtrar_lista(lista, funcao_filtro) do
    Enum.filter(lista, funcao_filtro)
  end
end
```

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

```
defmodule TurmaFunctions do
  def calcular_media(notas_da_turma) when is_list(notas_da_turma) do
    {notas_validas, _} =
      Enum.reduce(notas_da_turma, {[], 0}, fn
        {_aluno, []}, {notas, total} ->
          {notas, total}

        {_aluno, notas}, {notas, total} ->
          {notas ++ notas, total + length(notas)}
      end)

    case length(notas_validas) do
      0 -> {:error, "Nenhuma nota válida encontrada"}
      _ ->
        media = Enum.reduce(notas_validas, 0, &(&1 + &2)) / length(notas_validas)
        {:ok, media}
    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", []}
]

resultado = TurmaFunctions.calcular_media(notas_da_turma)
IO.inspect(resultado)
```