UNIVERSIDADE CESUMAR - UNICESUMAR

Curso: Engenharia de Software	**Série:** 6S	**Turma:** A	**Turno:** Noite
Professor(a): Thiago Bussola da Silva	**Horário:**		
Acadêmico (a): Felipe Defendi Prado			**RA: 210094882**
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 podem ser mais facilmente combinadas e compostas para criar programas complexos. Isso promove o reuso de código e a modularidade. Uma função pura é uma função que atende a duas condições, produz o mesmo resultado para os mesmos argumentos em todas as chamadas. Ela não causa efeitos colaterais, ou seja, não modifica nada fora de seu escopo. Já as funções de ordem superior são as que aceitam outras funções como argumentos e retornam funções como resultados. Exemplo pura e ordem superior:

Ordem superior defmodule Funcao do
Pura def par(num) do
r1 = rem(num, 2)
if r1 == 0 do
IO.puts("É par")
end
if r1 != 0 do
IO.puts("É impar")
end
end
end

Funcao.par(10) //colocar o numero desejado para testar funcao

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.

As vantagens da linguagem de programação elixir são que é uma linguagem funcional baseada em Erlang, e por isso nela se pode realizar sempre, funções puras, de ordem superior, imutáveis, o que confere uma maior segurança e confiabilidade para a linguagem. Além disso você pode ter um sistema totalmente escalável baseado nessa linguagem, então ela é ideal para projetos que vão crescer com o tempo. Por fim a linguagem elixir é utilizada geralmente em projetos complexos e que devem ser escalonáveis.

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

```
defmodule Funcao do
  def par(num) do
    r1 = rem(num, 2)
    if r1 == 0 do
       IO.puts("É par")
    end
```

```
    if r1 != 0 do
        IO.puts("É impar")
    end
  end
end

Funcao.par(10) //colocar o numero desejado para testar funcao
```

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

```
defmodule Funcao do
    def dobrarLista(lista) do
        Enum.map(lista, fn(x) -> x * 2 end)
    end
end

lista = [1, 2, 3, 4, 5]
IO.inspect(Funcao.dobrarLista(lista))
```

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

```
defmodule Funcao do
    def ultimoElemento(lista) do
        tam = length(lista)
         IO.inspect(Enum.at(lista, tam - 1))
    end
end

lista = [1, 2, 3, 4, 5, 6, 9]
Funcao.ultimoElemento(lista)
```

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

```
defmodule Funcao do
  def fatorial(0), do: 1
  def fatorial(n), do: n * fatorial(n - 1)
end

IO.puts(Funcao.fatorial(4))
```

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

elemento de uma lista.

```
defmodule Funcao do
   def somar_dobrar_lista(somar(), lista) do
      def somar(num) do
         num + 1
      end
      Enum.map(lista, fn(x) -> somar(x * 2) end
   end
end

lista = [1, 2, 3, 4, 5]
IO.inspect(Funcao.somar_dobrar_lista(lista))
```

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 Funcao do
   def filtrar(lista) do
      Enum.filter(lista, fn x -> rem(x, 2) == 0 end)
   end
end
lista = [1, 2, 3, 4, 5, 6, 7, 8, 9]

Funcao.filtrar(lista)
```

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

```
defmodule Fibonacci do
  def sequencia(n) when n <= 0, do: []
  def sequencia(1), do: [0]
  def sequencia(n) do
    sequencia(n, [0, 1])
  end

  defp sequencia(2, [a, b]), do: [a, b]
  defp sequencia(n, [a, b]) when n > 2 do
    proximo = a + b
    [a | sequencia(n - 1, [b, proximo])]
  end
end
n = 10
```

```
resultado = Fibonacci.sequencia(n)
IO.inspect(resultado)
```

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 Funcao do
    def calcular_media(notas) do
        Enum.filter(notas, fn(x) -> x.isInteger()  end)
        Enum.reduce(notas, fn x, acc -> x + acc end)
        tam = length(notas)
        IO.inspects(notas / tam)
    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", []},
]

Funcao.calcular_media(notas_da_turma)
```