

UNIVERSIDADE CESUMAR - UNICESUMAR

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Curso:** Engenharia de Software | | | | | | | | **Série:** 6S | | | | **Turma:** A | | | | | **Turno:** Noite | | |
| **Professor(a):** Thiago Bussola da Silva | | | | | | | | **Horário:** | | | | | | | | | | | |
| **Acadêmico (a): Felipe Gabriel Oliveira Machado** | | | | | | | | | | | | | | | | **RA: 21078749-2** | | | |
| **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 consistem em produzir o mesmo resultado para uma determinada entrada sem alterar nada, trabalhando junto com a programação funcional que consiste em dividir os problemas em funções simples e livres de problemas colaterais, o sistema irá mitigar a maior quantidade de bugs possível, tendo assim uma cobertura de testes muito maior durante a construção da aplicação, deixando-as muito bem testadas.\_

**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.

Imutabilidade: Os dados não sofrem mutação, assim elimina a capacidade de alterar o estado fora do escopo, mitigando a quantidade de bugs. Focos nas funções: Os problemas são decompostos em funções simples, ausentando efeitos colaterais, as funções puras produzem os mesmo resultados para uma determinada entrada e não alteram nada. As primeiras classes da programação funcional podem reutilizar facilmente as funções em todas as aplicações. Fácil de testar: Por serem pequenas funções, mitigam efeitos colaterais e por sua conta facilitam os testes, junto ao Elixir está uma biblioteca denomidada ExUnit que fornece tudo o que é necessário para testar de forma abrangente as aplicações.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

defmodule Verificador do

def verificar\_par\_ou\_impar(numero) when rem(numero, 2) == 0 do

IO.puts("par")

end

def verificar\_par\_ou\_impar(numero) do

IO.puts("ímpar")

end

end

Verificador.verificar\_par\_ou\_impar(1)

Verificador.verificar\_par\_ou\_impar(2)

Professor, sei que a questão é apenas para verificar se o número é par, mas eu um fiz tratamento para que quando não for par ele retorne como “ímpar”.

OBS: O “1” e o “2” foram os números utilizados para testar a função.

OBS2: No [tutorialspoint](https://www.tutorialspoint.com/execute_elixir_online.php) ele alerta que é melhor utilizar o “\_” no lugar de “numero”, mas dava erro.

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

---------------------------------------------- Resposta na próxima página----------------------------------------------

defmodule Calculadora do

def dobro\_de\_cada\_elemento(lista) do

Enum.map(lista, fn elemento -> elemento \* 2 end)

end

end

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

resultado = Calculadora.dobro\_de\_cada\_elemento(lista)

IO.inspect(resultado)

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

defmodule ListaUtils do

defp ultimo\_elemento([]), do: nil

defp ultimo\_elemento([elemento]), do: elemento

defp ultimo\_elemento([\_ | tail]), do: ultimo\_elemento(tail)

def ultimo(lista) do

ultimo\_elemento(lista)

end

end

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

ultimo = ListaUtils.ultimo(lista)

IO.inspect(ultimo)

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

defmodule Fatorial do

def calcular\_fatorial(0), do: 1

def calcular\_fatorial(n) when n > 0, do: n \* calcular\_fatorial(n - 1)

def calcular\_fatorial(\_), do: "Número inválido"

end

numero = 6

resultado = Fatorial.calcular\_fatorial(numero)

IO.puts("O fatorial de #{numero} é #{resultado}")

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

defmodule ListaUtils do

def aplicar\_funcao(lista, funcao) do

Enum.map(lista, &funcao.(&1))

end

end

quadrado = fn x -> x \* x end

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

resultado = ListaUtils.aplicar\_funcao(lista, quadrado)

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 ListaUtils do

def filtrar\_lista(lista, funcao\_filtro) do

Enum.filter(lista, funcao\_filtro)

end

end

funcao\_filtro = fn x -> x >= 3 end

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

resultado = ListaUtils.filtrar\_lista(lista, funcao\_filtro)

IO.inspect(resultado)

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

defmodule Fibonacci do

def gerar\_fibonacci(0), do: []

def gerar\_fibonacci(1), do: [0]

def gerar\_fibonacci(2), do: [0, 1]

def gerar\_fibonacci(n) when n > 2 do

gerar\_fibonacci(n, [1, 0])

end

defp gerar\_fibonacci(2, acc), do: Enum.reverse(acc)

defp gerar\_fibonacci(n, [a, b | tail]) when n > 2 do

gerar\_fibonacci(n - 1, [a + b, a | [b | tail]])

end

end

n = 10

sequencia = Fibonacci.gerar\_fibonacci(n)

IO.inspect(sequencia)

**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 TurmaUtils do

def calcular\_media(notas\_da\_turma) do

todas\_as\_notas = Enum.flat\_map(notas\_da\_turma, fn {\_, notas} -> notas end)

{soma, quantidade} =

Enum.reduce(todas\_as\_notas, {0.0, 0}, fn nota, {soma\_atual, quantidade\_atual} ->

{soma\_atual + nota, quantidade\_atual + 1}

end)

if quantidade == 0 do

"A turma não possui notas válidas."

else

soma / quantidade

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 = TurmaUtils.calcular\_media(notas\_da\_turma)

IO.puts("A média da turma é: #{media}")