INSTITUTO FEDERAL
Brasília

BOLSA FUTURO DIGITAL
PROGRAMA DE FORMAÇÃO ACELERADA

Gabarito da Lista de Exercícios

1.

```
class ProgressaoAritmetica:
    def __init__(self, a1, r, n):
        self.a1 = a1
        self.r = r
        self.n = n

    def gerar_termos(self):
        termos = []
        for i in range(self.n):
            termo = self.a1 + i * self.r
            termos.append(termo)
        return termos

    def calcular_soma(self):
        # Fórmula da soma da PA: S = n * (a1 + an) / 2
        an = self.a1 + (self.n - 1) * self.r
        soma = self.n * (self.a1 + an) / 2
        return soma


def main():
    print("==== Progressão Aritmética ====")
    a1 = float(input("Digite o primeiro termo (a1): "))
```

```
    r = float(input("Digite a razão (r): "))
    n = int(input("Digite o número de termos (n): "))

    pa = ProgressaoAritmetica(a1, r, n)

    termos = pa.gerar_termos()
    print("\nTermos da P.A.:")
    contador = 1
    for termo in termos:
        print(f"Termo {contador}: {termo}")
        contador += 1

    soma = pa.calcular_soma()
    print(f"\nSoma dos {n} termos: {soma}")

if __name__ == "__main__":
    main()
```

2.

```
class AnalisadorDeString:
    def __init__(self, texto):
        self.texto = texto

    def numero_de_caracteres(self):
        return len(self.texto)

    def em_maiusculas(self):
        return self.texto.upper()

    def em_minusculas(self):
        return self.texto.lower()

    def contar_vogais(self):
        vogais = 'aeiouAEIOU'
        contador = 0
        for caractere in self.texto:
            if caractere in vogais:
```

```
                contador += 1
        return contador

    def contem_ifb(self):
        return "IFB" in self.texto.upper()


def main():
    texto = input("Digite uma string: ")
    analisador = AnalisadorDeString(texto)

    print("\n=== Análise da String ===")
print(f"Número de caracteres
{analisador.numero_de_caracteres()}")
    print(f"Em maiúsculas: {analisador.em_maiusculas()}")
    print(f"Em minúsculas: {analisador.em_minusculas()}")
    print(f"Número de vogais: {analisador.contar_vogais()}")

    if analisador.contem_ifb():
        print("A substring 'IFB' aparece no texto (independente
de maiúsculas/minúsculas).")
    else:
        print("A substring 'IFB' NÃO aparece no texto.")


if __name__ == "__main__":
    main()
```

3.

```
class Criptografador:
    def __init__(self, frase):
        self.frase = frase

    def criptografar(self):
        substituicoes = {
            'a': '4', 'A': '4',
            'e': '3', 'E': '3',
```

```
            'i': '1', 'I': '1',
            'o': '0', 'O': '0',
            'u': '8', 'U': '8'
        }
        frase_criptografada = ''
        for caractere in self.frase:
            if caractere in substituicoes:
                frase_criptografada += substituicoes[caractere]
            else:
                frase_criptografada += caractere
        return frase_criptografada


def main():
    frase = input("Digite uma frase para criptografar: ")
    criptografador = Criptografador(frase)

    resultado = criptografador.criptografar()
    print("\nFrase criptografada:")
    print(resultado)


if __name__ == "__main__":
    main()
```

4.

```
class Produto:
    def __init__(self, nome, preco, quantidade):
        self.__nome = nome
        self.__preco = preco
        self.__quantidade = quantidade

    # Getters
    def get_nome(self):
        return self.__nome

    def get_preco(self):
```

```
        return self.__preco

    def get_quantidade(self):
        return self.__quantidade

    # Setters
    def set_preco(self, preco):
        if preco >= 0:
            self.__preco = preco

    def set_quantidade(self, quantidade):
        if quantidade >= 0:
            self.__quantidade = quantidade

    def calcular_total(self):
        return self.__preco * self.__quantidade


class CarrinhoDeCompras:
    def __init__(self):
        self.produtos = []

    def adicionar_produto(self, produto):
        self.produtos.append(produto)

    def remover_produto(self, nome):
        for produto in self.produtos:
            if produto.get_nome().lower() == nome.lower():
                self.produtos.remove(produto)
                return True
        return False

    def listar_produtos(self):
        if not self.produtos:
            print("Carrinho vazio.")
        else:
            print("\nProdutos no carrinho:")
```

```
            for produto in self.produtos:
                print(produto)

    def calcular_total(self):
        total = 0
        for produto in self.produtos:
            total += produto.calcular_total()
        return total


def main():
    carrinho = CarrinhoDeCompras()

    while True:
        print("\n==== Menu do Carrinho ====")
        print("1. Adicionar produto")
        print("2. Remover produto")
        print("3. Listar produtos")
        print("4. Calcular total")
        print("5. Sair")
        opcao = input("Escolha uma opção: ")

        if opcao == "1":
            nome = input("Nome do produto: ")
            preco = float(input("Preço: R$ "))
            quantidade = int(input("Quantidade: "))
            produto = Produto(nome, preco, quantidade)
            carrinho.adicionar_produto(produto)
            print("Produto adicionado!")

        elif opcao == "2":
            nome = input("Nome do produto a remover: ")
            if carrinho.remover_produto(nome):
                print("Produto removido.")
            else:
                print("Produto não encontrado.")
```

```
        elif opcao == "3":
            carrinho.listar_produtos()

        elif opcao == "4":
            total = carrinho.calcular_total()
            print(f"Total da compra: R$ {total:.2f}")

        elif opcao == "5":
            print("Encerrando o programa.")
            break

        else:
            print("Opção inválida. Tente novamente.")


if __name__ == "__main__":
    main()
```

5.

```
class Documento:
    def __init__(self, titulo, conteudo):
        self.__titulo = titulo
        self.__conteudo = conteudo

    def get_titulo(self):
        return self.__titulo

    def get_conteudo(self):
        return self.__conteudo


class Impressora:
    def imprimir(self, documento):
        print("\n=== Impressão do Documento ===")
        print(f"Título: {documento.get_titulo()}")
        print("Conteúdo:")
        print(documento.get_conteudo())
```

```
        print("================================")


def main():
    documentos = []
    impressora = Impressora()

    while True:
        print("\n==== MENU ====")
        print("1. Criar novo documento")
        print("2. Listar documentos")
        print("3. Imprimir documento")
        print("4. Sair")
        opcao = input("Escolha uma opção: ")

        if opcao == "1":
            titulo = input("Digite o título do documento: ")
            conteudo = input("Digite o conteúdo do documento: ")
            doc = Documento(titulo, conteudo)
            documentos.append(doc)
            print("Documento criado com sucesso!")

        elif opcao == "2":
            if not documentos:
                print("Nenhum documento criado ainda.")
            else:
                print("\n=== Lista de Documentos ===")
                for i, doc in enumerate(documentos):
                    print(f"{i + 1}. {doc.get_titulo()}")

        elif opcao == "3":
            if not documentos:
                print("Nenhum documento disponível para
impressão.")
            else:
                print("\nEscolha o número do documento para
imprimir:")
```

```
                for i, doc in enumerate(documentos):
                    print(f"{i + 1}. {doc.get_titulo()}")

                escolha = input("Número: ")
                if escolha.isdigit():
                    escolha = int(escolha)
                    if 1 <= escolha <= len(documentos):
                        impressora.imprimir(documentos[escolha - 1])
                    else:
                        print("Número inválido.")
                else:
                    print("Entrada inválida. Digite um número.")

            elif opcao == "4":
                print("Encerrando o programa.")
                break

            else:
                print("Opção inválida. Tente novamente.")


    if __name__ == "__main__":
        main()
```

6.

```
    class Funcionario:
        def __init__(self, nome, salario):
            self.nome = nome
            self.salario = salario



    class Departamento:
        def __init__(self, nome):
            self.nome = nome
            self.funcionarios = []

        def adicionar_funcionario(self, funcionario):
```

```
        self.funcionarios.append(funcionario)

    def listar_funcionarios(self):
        if not self.funcionarios:
            print("Nenhum funcionário neste departamento.")
        else:
            for f in self.funcionarios:
                print(f"{f.nome} - R$ {f.salario:.2f}")

    def media_salarial(self):
        if not self.funcionarios:
            return 0
        soma = 0
        for f in self.funcionarios:
            soma += f.salario
        return soma / len(self.funcionarios)


def main():
    funcionarios = []
    departamentos = []

    while True:
        print("\n=== MENU ===")
        print("1. Criar funcionário")
        print("2. Criar departamento")
        print("3. Adicionar funcionário ao departamento")
        print("4. Listar funcionários de um departamento")
        print("5. Mostrar média salarial do departamento")
        print("6. Sair")
        opcao = input("Opção: ")

        if opcao == "1":
            nome = input("Nome do funcionário: ")
            salario = float(input("Salário: "))
            funcionarios.append(Funcionario(nome, salario))
            print("Funcionário criado.")
```

```
        elif opcao == "2":
            nome = input("Nome do departamento: ")
            departamentos.append(Departamento(nome))
            print("Departamento criado.")

        elif opcao == "3":
            if funcionarios and departamentos:
                print("\nFuncionários:")
                for i in range(len(funcionarios)):
                    print(f"{i + 1}. {funcionarios[i].nome}")
                i_func = int(input("Escolha o número do funcionário:
")) - 1

                print("\nDepartamentos:")
                for j in range(len(departamentos)):
                    print(f"{j + 1}. {departamentos[j].nome}")
                j_dep = int(input("Escolha o número do
departamento: ")) - 1


departamentos[j_dep].adicionar_funcionario(funcionarios[i_fun
c])
                print("Funcionário adicionado ao departamento.")
            else:
                print("Crie funcionários e departamentos primeiro.")

        elif opcao == "4":
            for i in range(len(departamentos)):
                print(f"{i + 1}. {departamentos[i].nome}")
            escolha = int(input("Escolha o número do
departamento: ")) - 1
            departamentos[escolha].listar_funcionarios()

        elif opcao == "5":
            for i in range(len(departamentos)):
                print(f"{i + 1}. {departamentos[i].nome}")
```

```
            escolha = int(input("Escolha o número do
departamento: ")) - 1
            media = departamentos[escolha].media_salarial()
            print(f"Média salarial: R$ {media:.2f}")

        elif opcao == "6":
            print("Encerrando.")
            break

        else:
            print("Opção inválida.")



if __name__ == "__main__":
    main()
```

7.

```
class Casa:
    class __Comodo:  # Classe interna (composição)
        def __init__(self, nome, area):
            self.__nome = nome
            self.__area = area

        def get_nome(self):
            return self.__nome

        def get_area(self):
            return self.__area

    def __init__(self):
        self.__comodos = []

    def adicionar_comodo(self, nome, area):
        comodo = self.__Comodo(nome, area)
        self.__comodos.append(comodo)

    def listar_comodos(self):
```

```
        if not self.__comodos:
            print("Nenhum cômodo foi adicionado ainda.")
        else:
            print("Cômodos da casa:")
            for comodo in self.__comodos:
                print(f"- {comodo.get_nome()} ({comodo.get_area()}
m²)")

    def calcular_area_total(self):
        total = 0
        for comodo in self.__comodos:
            total += comodo.get_area()
        return total


def main():
    casa = None

    while True:
        print("\n=== MENU ===")
        print("1. Criar nova casa")
        print("2. Adicionar cômodo")
        print("3. Listar cômodos")
        print("4. Calcular área total")
        print("5. Sair")
        opcao = input("Escolha uma opção: ")

        if opcao == "1":
            casa = Casa()
            print("Casa criada com sucesso.")

        elif opcao == "2":
            if casa is None:
                print("Crie a casa primeiro.")
            else:
                nome = input("Nome do cômodo: ")
                area = float(input("Área do cômodo (m²): "))
```

```
                casa.adicionar_comodo(nome, area)
                print("Cômodo adicionado.")

            elif opcao == "3":
                if casa is None:
                    print("Crie a casa primeiro.")
                else:
                    casa.listar_comodos()

            elif opcao == "4":
                if casa is None:
                    print("Crie a casa primeiro.")
                else:
                    total = casa.calcular_area_total()
                    print(f"Área total da casa: {total:.2f} m²")

            elif opcao == "5":
                print("Encerrando o programa.")
                break

            else:
                print("Opção inválida.")


    if __name__ == "__main__":
        main()
```

8.

```
    # Classe base
    class Personagem:
        def __init__(self, nome, nivel):
            self.nome = nome
            self.nivel = nivel

        def atacar(self):
            print(f"{self.nome} realiza um ataque genérico.")
```

```
# Subclasse Guerreiro
class Guerreiro(Personagem):
    def __init__(self, nome, nivel, forca):
        super().__init__(nome, nivel)
        self.forca = forca

    def atacar(self):
        print(f"{self.nome} ataca com sua espada! (Força:
{self.forca})")


# Subclasse Mago
class Mago(Personagem):
    def __init__(self, nome, nivel, mana):
        super().__init__(nome, nivel)
        self.mana = mana

    def atacar(self):
        print(f"{self.nome} lança uma bola de fogo! (Mana:
{self.mana})")


# Demonstração
def main():
    personagem = Personagem("Aventureiro", 1)
    guerreiro = Guerreiro("Thorin", 5, 80)
    mago = Mago("Gandalf", 10, 120)

    lista_personagens = [personagem, guerreiro, mago]

    print("\n--- Ação dos Personagens ---")
    for p in lista_personagens:
        p.atacar()


if __name__ == "__main__":
```

```
    main()
```

9.

```
class Participante:
    def __init__(self, nome, email):
        self.nome = nome
        self.email = email

    def emitirCertificado(self):
        return f"{self.nome} - Certificado genérico de
participação."


class Aluno(Participante):
    def __init__(self, nome, email, curso):
        super().__init__(nome, email)
        self.curso = curso

    def emitirCertificado(self):
        return f"{self.nome} concluiu o curso de {self.curso} com
sucesso."


class Instrutor(Participante):
    def __init__(self, nome, email, especialidade):
        super().__init__(nome, email)
        self.especialidade = especialidade

    def emitirCertificado(self):
        return f"{self.nome} participou como palestrante na área
de {self.especialidade}."


def main():
    participantes = []

    while True:
```

```
        print("\n=== MENU ===")
        print("1. Cadastrar participante")
        print("2. Listar participantes")
        print("3. Emitir certificados")
        print("0. Sair")
        opcao = input("Escolha uma opção: ")

        if opcao == "1":
            print("\nCadastrar:")
            print("1. Aluno")
            print("2. Instrutor")
            tipo = input("Tipo de participante: ")

            nome = input("Nome: ")
            email = input("Email: ")

            if tipo == "1":
                curso = input("Curso: ")
                participantes.append(Aluno(nome, email, curso))
                print("Aluno cadastrado com sucesso.")
            elif tipo == "2":
                especialidade = input("Especialidade: ")
                participantes.append(Instrutor(nome, email,
especialidade))
                print("Instrutor cadastrado com sucesso.")
            else:
                print("Tipo inválido.")

        elif opcao == "2":
            if not participantes:
                print("Nenhum participante cadastrado.")
            else:
                print("\n=== Participantes Cadastrados ===")
                for p in participantes:
                    tipo = "Aluno" if isinstance(p, Aluno) else
"Instrutor"
                    print(f"{p.nome} ({tipo}) - {p.email}")
```

```
            elif opcao == "3":
                if not participantes:
                    print("Nenhum participante cadastrado.")
                else:
                    print("\n=== Certificados ===")
                    for p in participantes:
                        print(p.emitirCertificado())

            elif opcao == "0":
                print("Encerrando o programa.")
                break

            else:
                print("Opção inválida. Tente novamente.")


    if __name__ == "__main__":
        main()
```

10.

```
    # Classe base
class Personagem:
    def __init__(self, nome, constelacao):
        self.nome = nome
        self.constelacao = constelacao


    def apresentar(self):
        print(f"{self.nome}, cavaleiro da constelação de
    {self.constelacao}.")
```

```
# Classe Cavaleiro de Bronze
class CavaleiroDeBronze(Personagem):
    def __init__(self, nome, constelacao, poder_de_luta):
        super().__init__(nome, constelacao)
        self.poder_de_luta = poder_de_luta

    def golpe_especial(self):
        print(f"{self.nome} executa seu golpe especial com poder de
      luta {self.poder_de_luta}!")


# Classe Cavaleiro de Ouro
class CavaleiroDeOuro(Personagem):
    def __init__(self, nome, constelacao, casa_do_zodiaco):
        super().__init__(nome, constelacao)
        self.casa_do_zodiaco = casa_do_zodiaco

    def defender_casa(self):
        print(f"{self.nome} defende a casa de {self.casa_do_zodiaco}
      com honra!")


# Herança múltipla: Cavaleiro que combina os dois
class CavaleiroHibrido(CavaleiroDeBronze, CavaleiroDeOuro):
    def __init__(self, nome, constelacao, poder_de_luta,
      casa_do_zodiaco):
```

```
        CavaleiroDeBronze.__init__(self, nome, constelacao,
      poder_de_luta)
        self.casa_do_zodiaco = casa_do_zodiaco

    def golpe_especial(self):
        print(f"{self.nome} realiza um golpe híbrido com poder de luta
      {self.poder_de_luta}!")

    def defender_casa(self):
        print(f"{self.nome} protege a casa de {self.casa_do_zodiaco}
      com poder total!")


def main():
    personagens = []

    while True:
        print("\n=== MENU ===")
        print("1. Cadastrar cavaleiro")
        print("2. Listar personagens")
        print("3. Executar habilidades")
        print("0. Sair")
        opcao = input("Escolha uma opção: ")

        if opcao == "1":
            print("\nTipo de cavaleiro:")
```

```
    print("1. Cavaleiro de Bronze")
    print("2. Cavaleiro de Ouro")
    print("3. Cavaleiro Híbrido")
    tipo = input("Tipo: ")

    nome = input("Nome: ")
    constelacao = input("Constelação: ")

    if tipo == "1":
        poder = input("Poder de luta: ")
        personagem = CavaleiroDeBronze(nome, constelacao,
poder)
    elif tipo == "2":
        casa = input("Casa do zodíaco: ")
        personagem = CavaleiroDeOuro(nome, constelacao,
casa)
    elif tipo == "3":
        poder = input("Poder de luta: ")
        casa = input("Casa do zodíaco: ")
        personagem = CavaleiroHibrido(nome, constelacao,
poder, casa)
    else:
        print("Tipo inválido.")
        continue

    personagens.append(personagem)
```

```
        print("Cavaleiro cadastrado com sucesso.")

    elif opcao == "2":
        if not personagens:
            print("Nenhum personagem cadastrado.")
        else:
            print("\n--- Personagens ---")
            for p in personagens:
                p.apresentar()

    elif opcao == "3":
        if not personagens:
            print("Nenhum personagem cadastrado.")
        else:
            print("\n--- Habilidades ---")
            for p in personagens:
                print(f"\n{p.nome}:")
                if isinstance(p, CavaleiroDeBronze):
                    p.golpe_especial()
                if isinstance(p, CavaleiroDeOuro):
                    p.defender_casa()

    elif opcao == "0":
        print("Encerrando o sistema.")
        break
```

```
        else:
            print("Opção inválida. Tente novamente.")


if __name__ == "__main__":
    main()
```

11.

```
from abc import ABC, abstractmethod


# Classe abstrata
class VeiculoTransporte(ABC):
    def __init__(self, placa, capacidade_passageiros):
        self.placa = placa
        self.capacidade_passageiros =
      capacidade_passageiros


    @abstractmethod
    def calcularCustoOperacional(self):
        pass



# Subclasse Ônibus
class Onibus(VeiculoTransporte):
```

```
    def __init__(self, placa, capacidade_passageiros,
      consumo_por_km):
        super().__init__(placa, capacidade_passageiros)
        self.consumo_por_km = consumo_por_km

    def calcularCustoOperacional(self):
        return self.consumo_por_km * 6.00


# Subclasse Metrô
class Metro(VeiculoTransporte):
    def __init__(self, placa, capacidade_passageiros,
      consumo_energia_por_km):
        super().__init__(placa, capacidade_passageiros)
        self.consumo_energia_por_km =
      consumo_energia_por_km

    def calcularCustoOperacional(self):
        return self.consumo_energia_por_km * 0.80


def main():
    veiculos = []

    while True:
```

```
  print("\n=== MENU ===")
  print("1. Cadastrar Ônibus")
  print("2. Cadastrar Metrô")
  print("3. Mostrar custos operacionais")
  print("0. Sair")
  opcao = input("Escolha uma opção: ")


  if opcao == "1":
      print("\nCadastro de Ônibus")
      try:
          placa = input("Placa: ").strip()
          if placa == "":
              raise ValueError("A placa não pode estar
vazia.")
          capacidade = int(input("Capacidade de
passageiros: "))
          if capacidade <= 0:
              raise ValueError("A capacidade deve ser
positiva.")
          consumo = float(input("Consumo por km
(litros/km): "))
          if consumo <= 0:
              raise ValueError("O consumo deve ser
positivo.")
          veiculos.append(Onibus(placa, capacidade,
consumo))
          print("Ônibus cadastrado com sucesso!")
```

```
        except ValueError as e:
            print(f"Erro: {e}")


    elif opcao == "2":
        print("\nCadastro de Metrô")
        try:
            placa = input("Identificação: ").strip()
            if placa == "":
                raise ValueError("A identificação não pode estar
vazia.")
            capacidade = int(input("Capacidade de
passageiros: "))
            if capacidade <= 0:
                raise ValueError("A capacidade deve ser
positiva.")
            consumo = float(input("Consumo por km
(kWh/km): "))
            if consumo <= 0:
                raise ValueError("O consumo deve ser
positivo.")
            veiculos.append(Metro(placa, capacidade,
consumo))
            print("Metrô cadastrado com sucesso!")
        except ValueError as e:
            print(f"Erro: {e}")


    elif opcao == "3":
```

```
            if not veiculos:
                print("Nenhum veículo cadastrado.")
            else:
                print("\n--- Custos Operacionais por Km ---")
                for v in veiculos:
                    tipo = "Ônibus" if isinstance(v, Onibus) else
        "Metrô"
                    custo = v.calcularCustoOperacional()
                    print(f"{tipo} {v.placa}: R$ {custo:.2f} por km")

        elif opcao == "0":
            print("Encerrando o sistema.")
            break

        else:
            print("Opção inválida. Tente novamente.")


    if __name__ == "__main__":
        main()
```

12.

```
from abc import ABC, abstractmethod

# Classe totalmente abstrata
class Lutador(ABC):
```

```
    @abstractmethod
    def obter_nome(self):
        pass

    @abstractmethod
    def obter_poder(self):
        pass

    @abstractmethod
    def atacar(self):
        pass


# Subclasses
class Saiyajin(Lutador):
    def __init__(self, nome, poder):
        self.nome = nome
        self.poder = poder

    def obter_nome(self):
        return self.nome

    def obter_poder(self):
```

```
        return self.poder

    def atacar(self):
        print(f"{self.nome} se transforma em Super Saiyajin e lança um ataque devastador!")

class Androide(Lutador):
    def __init__(self, nome, poder):
        self.nome = nome
        self.poder = poder

    def obter_nome(self):
        return self.nome

    def obter_poder(self):
        return self.poder

    def atacar(self):
        print(f"{self.nome} usa energia infinita para disparar rajadas de Ki!")

class Namekuseijin(Lutador):
    def __init__(self, nome, poder):
```

```
        self.nome = nome
        self.poder = poder

    def obter_nome(self):
        return self.nome

    def obter_poder(self):
        return self.poder

    def atacar(self):
        print(f"{self.nome} estica os braços e ataca com golpes
     precisos!")



def main():
    lutadores = []

    while True:
        print("\n=== TORNEIO DE ARTES MARCIAIS ===")
        print("1. Cadastrar lutador")
        print("2. Listar lutadores")
        print("3. Simular ataque")
        print("0. Sair")
        opcao = input("Escolha uma opção: ")
```

```
  if opcao == "1":
      print("\nTipos de lutador:")
      print("1. Saiyajin")
      print("2. Androide")
      print("3. Namekuseijin")
      tipo = input("Escolha o tipo: ")

      try:
          nome = input("Nome do lutador: ").strip()
          if not nome:
              raise Exception("Erro: o nome não pode estar
vazio.")

          poder_input = input("Nível de poder: ")
          if not poder_input.strip():
              raise Exception("Erro: o nível de poder não
pode estar vazio.")
          poder = int(poder_input)
          if poder <= 0:
              raise Exception("Erro: o nível de poder deve
ser um número positivo.")

          if tipo == "1":
              lutador = Saiyajin(nome, poder)
          elif tipo == "2":
```

```
                lutador = Androide(nome, poder)
            elif tipo == "3":
                lutador = Namekuseijin(nome, poder)
            else:
                print("Tipo inválido.")
                continue

            lutadores.append(lutador)
            print(f"{lutador.obter_nome()} foi cadastrado com
sucesso!")

        except Exception as e:
            print(e)

    elif opcao == "2":
        if not lutadores:
            print("Nenhum lutador cadastrado.")
        else:
            print("\n--- Lutadores Inscritos ---")
            for i, l in enumerate(lutadores, start=1):
                print(f"{i}. {l.obter_nome()} - Poder:
{l.obter_poder()}")

    elif opcao == "3":
        if not lutadores:
```

```
            print("Nenhum lutador para atacar.")
        else:
            for i, l in enumerate(lutadores, start=1):
                print(f"{i}. {l.obter_nome()}")
            try:
                escolha = int(input("Escolha o número do
lutador: "))
                if 1 <= escolha <= len(lutadores):
                    print(f"\n{lutadores[escolha -
1].obter_nome()} vai atacar!")
                    lutadores[escolha - 1].atacar()
                else:
                    raise Exception("Número inválido para
escolha de lutador.")
            except Exception as e:
                print(f"Erro: {e}")

    elif opcao == "0":
        print("Encerrando o torneio. Até a próxima!")
        break

    else:
        print("Opção inválida. Tente novamente.")
```

```
if __name__ == "__main__":
    main()
```