Nome: João Geiger Piza
Repositório no Github: https://github.com/joaogpizza/Lipai
Códigos das Videoaulas:
aula01.py:

```
""" Aula 01 - Introdução a Orientação a objetos """

# paradigma de programação

# Calcular a área e o perimetro de um retangulo
# area = base * altura
# perimetro = 2 * (base + altura)
# Estrutura para armazenar os valores necessários para os calculos
retangulo1 = {
    'base': 10.0,
    'altura': 5.0
}

retangulo2 = {
    'base': 6.0,
    'altura': 3.0
}

# Realizar os cálculos


def calcular_area(retangulo):
    return retangulo['base'] * retangulo['altura']

def calcular_perimetro(retangulo):
    return 2 * (retangulo['base'] + retangulo['altura'])


print(calcular_area(retangulo1))
print(calcular_area(retangulo2))
print(calcular_perimetro(retangulo1))
print(calcular_perimetro(retangulo2))


# Classe representa um conceito
# Classe representa um retangulo
# Classe possui atributos base e altura
# Classe possui métodos (função dentro da classe)
class Retangulo:
```

```
    def __init__(self, base, altura):
        self.base = base
        self.altura = altura

    def calcular_area(self):
        return self.base * self.altura

    def calcular_perimetro(self):
        return 2 * (self.base + self.altura)


# Instanciando objetos do tipo Retangulo
# Chamando o método construtor
retangulo1 = Retangulo(10.0, 5.0)
retangulo2 = Retangulo(6.0, 3.0)

print(type(retangulo1), retangulo1)
print(type(retangulo2), retangulo2)

print(
    retangulo1.base,
    retangulo1.altura,
    retangulo1.calcular_area(),
    retangulo1.calcular_perimetro()
)
print(
    retangulo2.base,
    retangulo2.altura,
    retangulo2.calcular_area(),
    retangulo2.calcular_perimetro()
)
```

aula02.py:

```
""" Aula 02 - Atributos de classe e instância """


# Classe pessoa possui
# atributos de instancia: nome e email
# atributos de classe: especie
class Pessoa:
    especie = 'Humano'

    def __init__(self, nome, email):
```

```
        self.nome = nome
        self.email = email


pessoa1 = Pessoa('Maria da Silva', 'maria@email.com')
pessoa2 = Pessoa('João Santos', 'joao@email.com')
pessoa3 = Pessoa('Pedro Santos', 'pedro@email.com')

# alterar um atributo de classe na instância (objeto)
#altera somente para aquela instância
pessoa1.especie = 'Alienigena'

# alterando um atributo de classe na classe
# altera todos os objetos e na classe também
Pessoa.especie = 'Alienigenas do Passado'

print(pessoa1.nome, pessoa1.email, pessoa1.especie)
print(pessoa2.nome, pessoa2.email, pessoa2.especie)
print(pessoa3.nome, pessoa3.email, pessoa3.especie)

print(Pessoa.especie)
```

aula03.py:

```
""" Aula 03 - Métodos de classe """

class Retangulo:
    def __init__(self, base, altura):
        self.base = base
        self.altura = altura

    @classmethod
    def from_list(cls, lista):
        return cls(lista[0], lista[1])

    @classmethod
    def from_string(cls, rep_retangulo):
        base, altura = rep_retangulo.split(sep=',')
        return cls(float(base), float(altura))

    def calcular_area(self):
        return self.base * self.altura

    def calcular_perimetro(self):
```

```
        return 2 * (self.base + self.altura)

retangulo1 = Retangulo(10.0, 5.0)
retangulo2 = Retangulo(6.0, 3.0)
retangulo3 = Retangulo.from_list([20.0, 3.5])
retangulo4 = Retangulo.from_string("55.4,13.5")

print(retangulo3.base, retangulo3.altura, retangulo3.calcular_area())
print(retangulo4.base, retangulo4.altura, retangulo4.calcular_area())
```

aula04.py:

```
""" Aula 04 - Propriedades """

# forma de controlar acesso aos atributos de uma instancia
# formas personalizadas de obter e alterar o valor de um atributo

class Retangulo:
    def __init__(self, base, altura):
        self.base = base
        self.altura = altura

    # getter
    @property
    def base(self):
        return self._base

    # setter
    @base.setter
    def base(self, value):
        if value <= 0.0:
            raise ValueError('A base deve ser maior que zero')
        self._base = value

    # getter
    @property
    def altura(self):
        return self._altura

    # setter
    @altura.setter
    def altura(self, value):
        if value <= 0.0:
            raise ValueError('A altura deve ser maior que zero')
```

```
        self._altura = value

    @classmethod
    def from_list(cls, lista):
        return cls(lista[0], lista[1])

    @classmethod
    def from_string(cls, rep_retangulo):
        base, altura = rep_retangulo.split(sep=',')
        return cls(float(base), float(altura))

    def calcular_area(self):
        return self.base * self.altura

    def calcular_perimetro(self):
        return 2 * (self.base + self.altura)


retangulo1 = Retangulo(3.0, 3.0)

retangulo1.base = 3.0
retangulo1.altura = 3.0
print(retangulo1.base)
```

aula05.py:

```
""" Aula 05 - Métodos especiais """


class Retangulo:
    def __init__(self, base, altura):
        self.base = base
        self.altura = altura

    def calcular_area(self):
        return self.base * self.altura

    def calcular_perimetro(self):
        return 2 * (self.base + self.altura)

    # __str__(self)
    # representação do objeto como string para humanos
    def __str__(self):
        return f'Retangulo[base={self.base}, altura={self.altura}]'
```

```
    # __repr__(self)
    # representação do objeto como string para recriar
    # esse objeto
    # logging, debug
    # representação canonica
    def __repr__(self):
        return f'Retangulo({self.base},{self.altura})'


retangulo1 = Retangulo(10.0, 5.0)
retangulo2 = Retangulo(3.0, 14.0)

print(retangulo1.__repr__())

retangulo3 = eval('Retangulo(7.5,12.3)')
retangulo4 = eval(repr(retangulo3))

print(retangulo1)
print(retangulo2)
print(retangulo3)
print(retangulo4)
```

aula06.py:

```
""" Aula 06 - equal e hash code """

nome1 = 'João'
nome2 = 'João'

print(nome1 == nome2)


class Pessoa:
    def __init__(self, cpf, nome):
        self.cpf = cpf
        self.nome = nome


    def __eq__(self, value):
        if isinstance(value, self.__class__):
            return self.cpf == value.cpf
        return False
```

```
    def __hash__(self):
        return hash(self.cpf)

    def __repr__(self):
        return f'Pessoa({self.cpf},{self.nome})'


pessoa1 = Pessoa('100100100-10', 'João')
pessoa2 = Pessoa('100100100-10', 'João')
pessoa3 = Pessoa('100100100-11', 'Maria')

pessoas = {pessoa1, pessoa2, pessoa3}
print(pessoas)
print(pessoa1 == pessoa2)


pessoas_lista = [pessoa1, pessoa2, pessoa3]
print(pessoas_lista)

print(pessoas_lista.count(pessoa1))
```

aula07.py:

```
""" Aula07 - Relacionamentos entre classes """


class Endereco:
    def __init__(self, cep, numero):
        self.cep = cep
        self.numero = numero

    def __str__(self):
        return f'Endereco[cep={self.cep}, numero={self.numero}]'


class Telefone:
    def __init__(self, ddd, numero):
        self.ddd = ddd
        self.numero = numero

    def __str__(self):
        return f'Telefone[ddd={self.ddd}, numero={self.numero}]'
```

```
class Pessoa:
    def __init__(self, cpf, nome, telefone, endereco):
        self.cpf = cpf
        self.nome = nome
        self.telefone = telefone
        self.enderecos = [endereco]

    def add_endereco(self, endereco):
        self.enderecos.append(endereco)

    def print_enderecos(self):
        print(self.nome)
        for endereco in self.enderecos:
            print(endereco)

    def __str__(self):
        return f'Pessoa[cpf={self.cpf}, nome={self.nome},
telefone={self.telefone}]'


telefone = Telefone('11', '1111-1111')
pessoa1 = Pessoa('112332131', 'João da Silva', telefone,
Endereco('02233039', 123))
pessoa1.add_endereco(Endereco('92332323', 55))

pessoa2 = Pessoa('223232', 'Maria da Silva', telefone,
Endereco('1231221', 33))

pessoa3 = Pessoa('3333333', 'Pedro da Silva', telefone,
Endereco('1231221', 33))

print(pessoa1)
print(pessoa1.cpf, pessoa1.nome, pessoa1.telefone)
print(pessoa1.telefone.ddd, pessoa1.telefone.numero)

print(pessoa2)

pessoa1.print_enderecos()
pessoa2.print_enderecos()
pessoa3.print_enderecos()
```

heranca.py:

```
""" Herança """
```

```python
class Pessoa: # SUPER CLASSE
    def __init__(self, nome, sobrenome, cpf):
        print("Entrei no SUPER CONSTRUTOR")
        self.nome = nome
        self.sobrenome = sobrenome
        self.cpf = cpf

    def obtem_nome_completo(self):
        return f'{self.nome} {self.sobrenome}'

class Cliente(Pessoa): # SUB CLASSE
    def __init__(self, nome, sobrenome, cpf):
        super().__init__(nome, sobrenome, cpf)
        self.compras = []

class Funcionario(Pessoa):
    def __init__(self, nome, sobrenome, cpf, salario):
        super().__init__(nome, sobrenome, cpf)
        self.salario = salario

    def calcula_pagamento(self):
        return self.salario - ((10/100) * self.salario)

class Programador(Funcionario):
    def __init__(self, nome, sobrenome, cpf, salario, bonus):
        super().__init__(nome, sobrenome, cpf, salario)
        self.bonus = bonus

    def calcula_pagamento(self):
        pagamento_salario = super().calcula_pagamento()
        return pagamento_salario + self.bonus

programador = Programador("José", "Augusto", "123.123.123-12", 5000,
200)
print(programador.obtem_nome_completo())
print(programador.calcula_pagamento())

# cliente = Cliente("Paulo", "Mulotto", "123.123.123-12")
# print(cliente.obtem_nome_completo())
# print(type(cliente))
```

Exercícios:

ex01.py:

```
""" Exercicio 01 S3.A3 """

class Aluno:
    """ Classe Principal do exercício """
    def __init__(self, prontuario, nome, email):
        self.prontuario = prontuario
        self.nome = nome
        self.email = email

    @classmethod
    def from_string(cls, string):
        """ Pega uma string do tipo 'Prontuario,Nome,Email' para criar
um Aluno """
        prontuario, nome, email = string.split(',')
        return cls(prontuario, nome, email)

    def __str__(self):
        return f'Aluno[prontuario={self.prontuario}, nome={self.nome},
email={self.email}]'

    @property
    def prontuario(self):
        return self._prontuario

    @prontuario.setter
    def prontuario(self, valor):
        if valor:
            self._prontuario = valor
        else:
            raise ValueError('Prontuario nulo/vazio')

    @property
    def nome(self):
        return self._nome

    @nome.setter
    def nome(self, valor):
        if valor:
            self._nome = valor
        else:
            raise ValueError('Nome nulo/vazio')
```

```
    @property
    def email(self):
        return self._email

    @email.setter
    def email(self, valor):
        if valor:
            self._email = valor
        else:
            raise ValueError('Email nulo/vazio')

    def __eq__(self, value):
        """ 2 Alunos são iguais se tiverem o mesmo prontuário """
        if isinstance(value, self.__class__):
            return self.prontuario == value.prontuario
        return False

    def __hash__(self):
        return hash(self.prontuario)

# Testando a classe
aluno1 = Aluno('SP0101', 'João da Silva', 'joao@email.com')
aluno2 = Aluno.from_string('SP0202,Pâmela da Silva,pamela@email.com')
aluno3 = Aluno('SP0101', 'Maria da Silva', 'maria@email.com')

alunos = [aluno1, aluno2, aluno3]
for aluno in alunos:
    print(aluno)
print('--------------------')
print(aluno1 == aluno3)
print('--------------------')
alunosConjunto = { aluno1, aluno2, aluno3 }
for aluno in alunosConjunto:
    print(aluno)
```

ex02.py:

```
""" Exercício 02 S3.A3 """

class Projeto:
    """ Classe Projeto """
    def __init__(self, codigo, titulo, responsavel):
        self.codigo = codigo
```

```
        self.titulo = titulo
        self.responsavel = responsavel

    @classmethod
    def from_string(cls, string):
        codigo, titulo, responsavel = string.split(',')
        return cls(int(codigo), titulo, responsavel)

    def __str__(self):
        return f'Projeto[codigo={self.codigo}, titulo={self.titulo},
responsavel={self.responsavel}]'

    def __eq__(self, value):
        if isinstance(value, self.__class__):
            return self.codigo == value.codigo
        return False

    def __hash__(self):
        return hash(self.codigo)

    @property
    def codigo(self):
        return self._codigo

    @codigo.setter
    def codigo(self, valor):
        if not isinstance(valor, int):
            raise ValueError('Código deve ser um inteiro')
        if valor <= 0:
            raise ValueError('Código <= 0')
        self._codigo = valor

    @property
    def titulo(self):
        return self._titulo

    @titulo.setter
    def titulo(self, valor):
        if not valor:
            raise ValueError('Titulo nulo/vazio')
        self._titulo = valor

    @property
```

```
    def email(self):
        return self._email

    @email.setter
    def email(self, valor):
        if not valor:
            raise ValueError('Email nulo/vazio')
        self._email = valor

# Testando a classe
projeto1 = Projeto(133, 'Pesquisa', 'Juan')
projeto2 = Projeto.from_string('1,Laboratório de Desenvolvimento de
Software,Pedro Gomes')
projeto3 = Projeto(133, 'Teste de classes', 'Yuri Alberto
Protagonista')

projetos = [projeto1, projeto2, projeto3]
for projeto in projetos:
    print(projeto)

print('--------------')
print(projeto1 == projeto3)
print('--------------')

projetosConjunto = { projeto1, projeto2, projeto3 }
for projeto in projetosConjunto:
    print(projeto)
```

ex03.py:

```
""" Exercício 03 S3.A3 """

# Classes Aluno e Projeto (dos exs anteriores)

class Aluno:
    """ Classe Principal do exercício """
    def __init__(self, prontuario, nome, email):
        self.prontuario = prontuario
        self.nome = nome
        self.email = email

    @classmethod
    def from_string(cls, string):
```

```
        """ Pega uma string do tipo 'Prontuario,Nome,Email' para criar
um Aluno """
        prontuario, nome, email = string.split(',')
        return cls(prontuario, nome, email)

    def __str__(self):
        return f'Aluno[prontuario={self.prontuario}, nome={self.nome},
email={self.email}]'

    @property
    def prontuario(self):
        return self._prontuario

    @prontuario.setter
    def prontuario(self, valor):
        if valor:
            self._prontuario = valor
        else:
            raise ValueError('Prontuario nulo/vazio')

    @property
    def nome(self):
        return self._nome

    @nome.setter
    def nome(self, valor):
        if valor:
            self._nome = valor
        else:
            raise ValueError('Nome nulo/vazio')

    @property
    def email(self):
        return self._email

    @email.setter
    def email(self, valor):
        if valor:
            self._email = valor
        else:
            raise ValueError('Email nulo/vazio')

    def __eq__(self, value):
```

```
        """ 2 Alunos são iguais se tiverem o mesmo prontuário """
        if isinstance(value, self.__class__):
            return self.prontuario == value.prontuario
        return False

    def __hash__(self):
        return hash(self.prontuario)

class Projeto:
    """ Classe Projeto """
    def __init__(self, codigo, titulo, responsavel):
        self.codigo = codigo
        self.titulo = titulo
        self.responsavel = responsavel

    @classmethod
    def from_string(cls, string):
        codigo, titulo, responsavel = string.split(',')
        return cls(int(codigo), titulo, responsavel)

    def __str__(self):
        return f'Projeto[codigo={self.codigo}, titulo={self.titulo},
responsavel={self.responsavel}]'

    def __eq__(self, value):
        if isinstance(value, self.__class__):
            return self.codigo == value.codigo
        return False

    def __hash__(self):
        return hash(self.codigo)

    @property
    def codigo(self):
        return self._codigo

    @codigo.setter
    def codigo(self, valor):
        if not isinstance(valor, int):
            raise ValueError('Código deve ser um inteiro')
        if valor <= 0:
            raise ValueError('Código <= 0')
        self._codigo = valor
```

```
    @property
    def titulo(self):
        return self._titulo

    @titulo.setter
    def titulo(self, valor):
        if not valor:
            raise ValueError('Titulo nulo/vazio')
        self._titulo = valor

    @property
    def email(self):
        return self._email

    @email.setter
    def email(self, valor):
        if not valor:
            raise ValueError('Email nulo/vazio')
        self._email = valor

class Participacao:
    def __init__(self, codigo, data_inicio, data_fim, aluno, projeto):
        self.codigo = codigo
        self.data_inicio = data_inicio
        self.data_fim = data_fim
        self.aluno = aluno
        self.projeto = projeto

    @classmethod
    def from_string(cls, string):
        partes = string.split(',')
        return cls(codigo = partes[0],
                   data_inicio = partes[1],
                   data_fim = partes[2],
                   aluno = Aluno.from_string(partes[3] + ',' +
partes[4] + ',' + partes[5]),
                   projeto = Projeto.from_string(partes[6] + ',' +
partes[7] + ',' + partes[8])
                  )

    def __str__(self):
```

```
        return (f'Participação[codigo={self.codigo}, data de
inicio={self.data_inicio}, ' +
                f'data de fim = {self.data_fim}, aluno = {self.aluno},
projeto = {self.projeto}')

    def __eq__(self, valor):
        if isinstance(valor, self.__class__):
            return self.codigo == valor.codigo
        return False

    def __hash__(self):
        return hash(self.codigo)

    @property
    def codigo(self):
        return self._codigo

    @codigo.setter
    def codigo(self, valor):
        if not valor:
            raise ValueError('Código nulo/vazio')
        self._codigo = valor

    @property
    def data_inicio(self):
        return self._data_inicio

    @data_inicio.setter
    def data_inicio(self, valor):
        if not valor:
            raise ValueError('Data de início nula/vazia')
        self._data_inicio = valor

    @property
    def data_fim(self):
        return self._data_fim

    @data_fim.setter
    def data_fim(self, valor):
        if not valor:
            raise ValueError('Data de fim nula/vazia')
        self._data_fim = valor
```

```
    @property
    def aluno(self):
        return self._aluno

    @aluno.setter
    def aluno(self, valor):
        if not valor or not isinstance(valor, Aluno):
            raise ValueError('Aluno informado incorretamente')
        self._aluno = valor

    @property
    def projeto(self):
        return self._projeto

    @projeto.setter
    def projeto(self, valor):
        if not valor or not isinstance(valor, Projeto):
            raise ValueError('Projeto informado incorretamente')
        self._projeto = valor

# Testando a classe
aluno1 = Aluno.from_string('SP0101,João da Silva,joao@email.com')
aluno2 = Aluno.from_string('MG0202,Breno Bidon,bidon@corinthians.com')
projeto1 = Projeto.from_string('123321,Pesquisa sobre Corinthians,Yuri
Alberto')
projeto2 = Projeto.from_string('122221,Vasco é minusculo,Memphis
Depão')

participacao1 = Participacao('SP471231', '23/04/2025', '21/12/2025',
aluno1, projeto1)
participacao2 = Participacao('RJ000000', '23/04/25', '21/12/2025',
aluno2, projeto1)
participacao3 = Participacao('SP471231', '01/09/1910', '22/12/2025',
aluno2, projeto2)

participacoes = [participacao1, participacao2, participacao3]
for participacao in participacoes:
    print(participacao)

print('--------------------')
print(participacao1 == participacao3)
print('--------------------')
```

```
participacoesConjunto = {participacao1, participacao2, participacao3}
for participacao in participacoesConjunto:
    print(participacao)
```

ex04.py:

```
""" Exercício 04 S3.A3 """

# Classe Produto alterada
class Projeto:
    """ Classe Projeto """

    def __init__(self, codigo, titulo, responsavel):
        self.codigo = codigo
        self.titulo = titulo
        self.responsavel = responsavel
        self.participacoes = []

    def add_participacao(self, participacao):
        """Adiciona uma participação ao projeto."""
        if not isinstance(participacao, Participacao):
            raise ValueError('Participação informada incorretamente')
        self.participacoes.append(participacao)

    @classmethod
    def from_string(cls, string):
        codigo, titulo, responsavel = string.split(',')
        return cls(int(codigo), titulo, responsavel)

    def __str__(self):
        resultado = f'Projeto[codigo={self.codigo},
titulo={self.titulo}, responsavel={self.responsavel}, códigos das
participações=['
        for i in range(len(self.participacoes)):
            resultado += f'{(self.participacoes[i]).codigo}'
            if i == len(self.participacoes) - 1:
                resultado += ']'
                continue
            resultado += ', '
        return resultado + ']'

    def __eq__(self, value):
        if isinstance(value, self.__class__):
            return self.codigo == value.codigo
```

```
        return False

    def __hash__(self):
        return hash(self.codigo)

    @property
    def codigo(self):
        return self._codigo

    @codigo.setter
    def codigo(self, valor):
        if not isinstance(valor, int):
            raise ValueError('Código deve ser um inteiro')
        if valor <= 0:
            raise ValueError('Código <= 0')
        self._codigo = valor

    @property
    def titulo(self):
        return self._titulo

    @titulo.setter
    def titulo(self, valor):
        if not valor:
            raise ValueError('Titulo nulo/vazio')
        self._titulo = valor

    @property
    def email(self):
        return self._email

    @email.setter
    def email(self, valor):
        if not valor:
            raise ValueError('Email nulo/vazio')
        self._email = valor

# Classe Aluno (identica)
class Aluno:
    """ Classe Aluno """
    def __init__(self, prontuario, nome, email):
        self.prontuario = prontuario
        self.nome = nome
```

```
        self.email = email

    @classmethod
    def from_string(cls, string):
        """ Pega uma string do tipo 'Prontuario,Nome,Email' para criar
um Aluno """
        prontuario, nome, email = string.split(',')
        return cls(prontuario, nome, email)

    def __str__(self):
        return f'Aluno[prontuario={self.prontuario}, nome={self.nome},
email={self.email}]'

    @property
    def prontuario(self):
        return self._prontuario

    @prontuario.setter
    def prontuario(self, valor):
        if valor:
            self._prontuario = valor
        else:
            raise ValueError('Prontuario nulo/vazio')

    @property
    def nome(self):
        return self._nome

    @nome.setter
    def nome(self, valor):
        if valor:
            self._nome = valor
        else:
            raise ValueError('Nome nulo/vazio')

    @property
    def email(self):
        return self._email

    @email.setter
    def email(self, valor):
        if valor:
            self._email = valor
```

```
        else:
            raise ValueError('Email nulo/vazio')

    def __eq__(self, value):
        """ 2 Alunos são iguais se tiverem o mesmo prontuário """
        if isinstance(value, self.__class__):
            return self.prontuario == value.prontuario
        return False

    def __hash__(self):
        return hash(self.prontuario)

# Classe Participacao (alterada para usar add_participacao
automaticamente)
class Participacao:
    def __init__(self, codigo, data_inicio, data_fim, aluno, projeto):
        self.codigo = codigo
        self.data_inicio = data_inicio
        self.data_fim = data_fim
        self.aluno = aluno
        self.projeto = projeto
        self.projeto.add_participacao(self)

    @classmethod
    def from_string(cls, string):
        partes = string.split(',')
        return cls(codigo = partes[0],
                   data_inicio = partes[1],
                   data_fim = partes[2],
                   aluno = Aluno.from_string(partes[3] + ',' +
partes[4] + ',' + partes[5]),
                   projeto = Projeto.from_string(partes[6] + ',' +
partes[7] + ',' + partes[8])
                  )

    def __str__(self):
        return (f'Participação[codigo={self.codigo}, data de
inicio={self.data_inicio}, ' +
                f'data de fim = {self.data_fim}, aluno = {self.aluno},
projeto = {self.projeto}')

    def __eq__(self, valor):
        if isinstance(valor, self.__class__):
```

```
            return self.codigo == valor.codigo
        return False

    def __hash__(self):
        return hash(self.codigo)

    @property
    def codigo(self):
        return self._codigo

    @codigo.setter
    def codigo(self, valor):
        if not valor:
            raise ValueError('Código nulo/vazio')
        self._codigo = valor

    @property
    def data_inicio(self):
        return self._data_inicio

    @data_inicio.setter
    def data_inicio(self, valor):
        if not valor:
            raise ValueError('Data de início nula/vazia')
        self._data_inicio = valor

    @property
    def data_fim(self):
        return self._data_fim

    @data_fim.setter
    def data_fim(self, valor):
        if not valor:
            raise ValueError('Data de fim nula/vazia')
        self._data_fim = valor

    @property
    def aluno(self):
        return self._aluno

    @aluno.setter
    def aluno(self, valor):
        if not valor or not isinstance(valor, Aluno):
```

```
            raise ValueError('Aluno informado incorretamente')
        self._aluno = valor

    @property
    def projeto(self):
        return self._projeto

    @projeto.setter
    def projeto(self, valor):
        if not valor or not isinstance(valor, Projeto):
            raise ValueError('Projeto informado incorretamente')
        self._projeto = valor

# Testando alterações
aluno1 = Aluno.from_string('SP0101,João da Silva,joao@email.com')
aluno2 = Aluno.from_string('MG0202,Breno Bidon,bidon@corinthians.com')
projeto1 = Projeto.from_string('123321,Pesquisa sobre Corinthians,Yuri
Alberto')
projeto2 = Projeto.from_string('122221,Vasco é minusculo,Memphis
Depão')

participacao1 = Participacao('SP471231', '23/04/2025', '21/12/2025',
aluno1, projeto1)
participacao2 = Participacao('RJ000000', '23/04/25', '21/12/2025',
aluno2, projeto1)
participacao3 = Participacao('SP471231', '01/09/1910', '22/12/2025',
aluno2, projeto2)

print(projeto1)
print(projeto2)
```