



UNIVERSIDADE DE VASSOURAS

Curso de Graduação em Engenharia Software

Aula 5

## Laboratório de Programação Orientada à Objeto

**Prof. Diego Ramos Inácio**

Geógrafo

Mestrando em Engenharia de Biossistemas

Especialista em Topografia e Sensoriamento Remoto

Specialist in GIS and Data Modeling em Digimap



# Decoradores em Python

Uma visão geral dos principais decoradores usados para métodos e atributos em classes Python.



# O que são Decoradores?

## Funções Especiais

Alteram o comportamento de métodos ou funções em Python.

## Aplicação

Aplicados com @ antes do nome da função ou método.

## Utilização

Usados para controle de acesso, modificação de comportamento e encapsulamento de lógica.

# @classmethod

1

## Propósito

Transforma um método em um método da classe, não da instância.

2

## Argumento

Recebe a classe como primeiro argumento (cls).

3

## Uso

Permite criar métodos que manipulam atributos e comportamentos da classe inteira, não de instâncias específicas.

Exemplo:

```
class Student: count = 0 @classmethod def increment(cls): cls.count += 1
```



# @staticmethod

1

## Propósito

Declara um método que não depende de atributos da instância ou da classe.

2

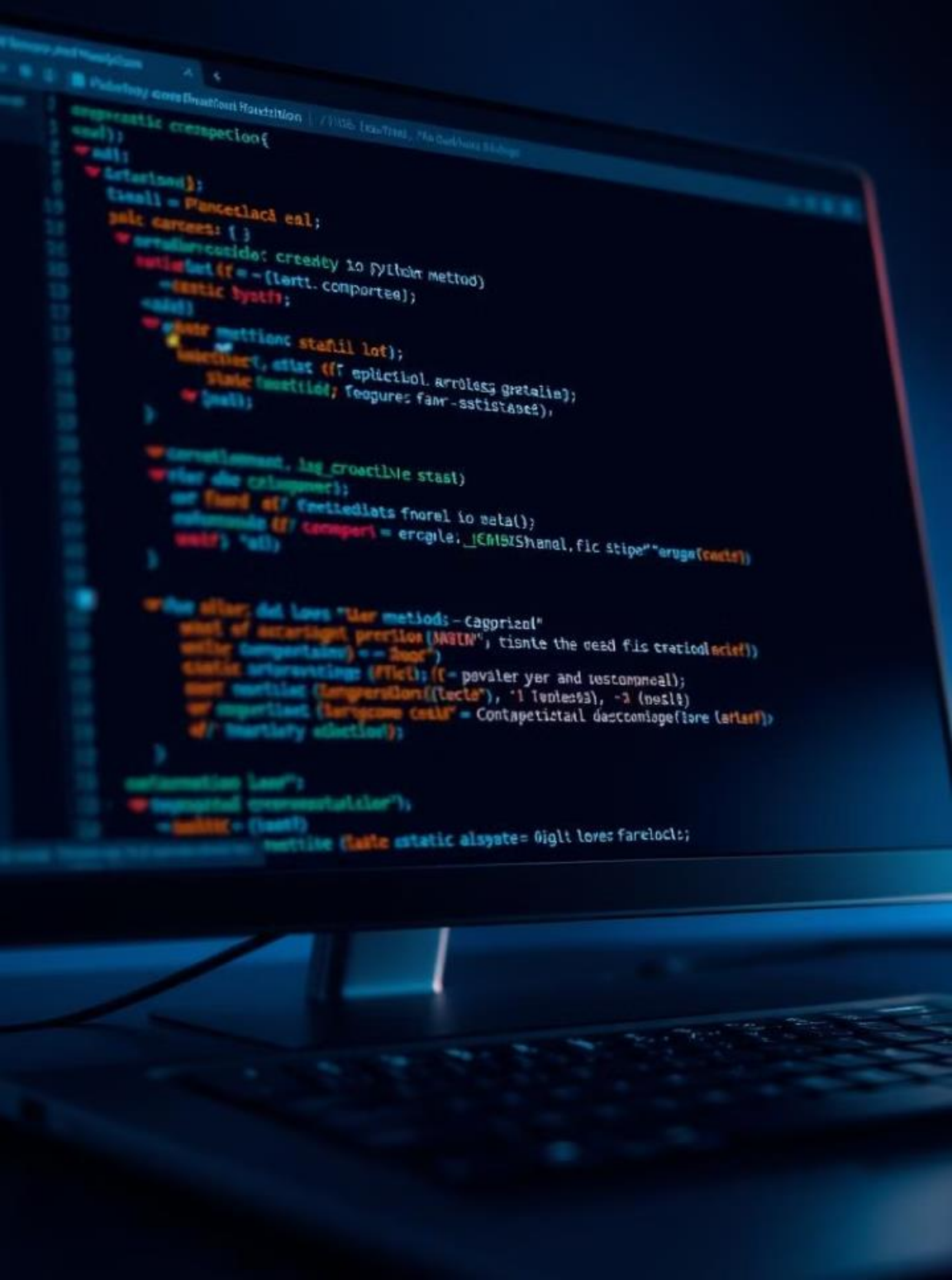
## Argumento

Não recebe self ou cls.

3

## Uso

Funções que fazem parte da lógica da classe, mas não precisam acessar dados da classe/instância.



```
15 <delete the cgreber for sett/>
10 <@Student class (, an first get lert;
26 fire -goe " perpret, leam the "sedect or seet//r",
    );
25
20 Student: clatuls. ", ny' {
27     };
28 }
29
```

# @property

1

## Propósito

Transforma métodos em atributos acessíveis diretamente.

2

## Uso

Permite encapsular lógica de acesso e definir "getters" e "setters" para controle de atributos.

3

## Exemplo

Classe "Student" com atributo "age" encapsulado.

# Comparação Entre Decoradores

```
class Student:
    count = 0
    @classmethod
    def increment(cls):
        cls.count += 1
```

@classmethod

Manipula a classe em vez de instâncias.

```
class MathHelper:
    @staticmethod
    def add(x, y):
        return x + y
```

@staticmethod

Não acessa dados da classe ou da instância.

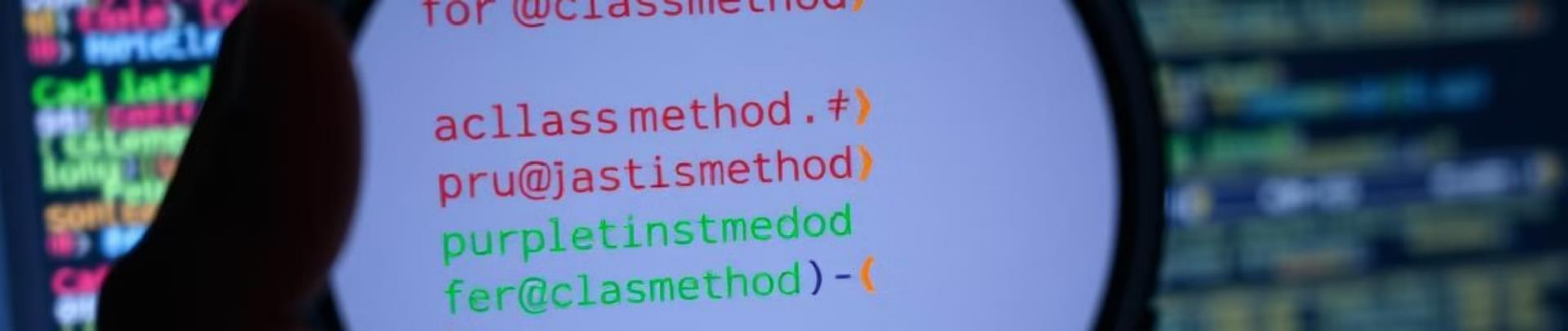
```
class Student:
    def __init__(self, name, age):
        self._age = age

    @property
    def age(self):
        return self._age

    @age.setter
    def age(self, value):
        if value >= 0:
            self._age = value
```

@property

Acessa métodos como atributos.



```
for @classmethod)
aclassmethod.#)
pru@staticmethod)
purpletinstmedod
fer@classmethod)-(
```

# Quando Utilizar Cada Um?

1

@classmethod: Dados ou comportamentos compartilhados por todas as instâncias.

2

@staticmethod: Lógica que não depende de instâncias ou da classe.

3

@property: Controle sobre como atributos são acessados ou modificados.



# @classmethod (Python) vs. static (Java)

## Python

O método da classe recebe a classe como argumento (cls)

Pode acessar/modificar atributos da classe.

```
class Student:
    count = 0
    @classmethod
    def increment(cls):
        cls.count += 1
```

## Java

O método static pertence à classe

Pode acessar apenas atributos estáticos.

```
class Student {
    static int count = 0;
    static void increment() {
        count++;
    }
}
```

# @staticmethod (Python) vs. static (Java)

```
class MathHelper:  
    @staticmethod  
    def add(x, y):  
        return x + y
```

Python

Método não depende de instâncias ou classe

Chamado diretamente na classe

Java

Métodos estáticos não dependem de instâncias

Chamado diretamente na classe

```
class MathHelper {  
    static int add(int x, int y) {  
        return x + y;  
    }  
}
```

```
class Student {  
    private int age;  
  
    public int getAge() {  
        return age;  
    }  
  
    public void setAge(int age) {  
        if (age >= 0) {  
            this.age = age;  
        }  
    }  
}
```

```
class Student:  
    def __init__(self, name, age):  
        self._age = age  
  
    @property  
    def age(self):  
        return self._age  
  
    @age.setter  
    def age(self, value):  
        if value >= 0:  
            self._age = value
```

## @property (Python) vs. Getters/Setters (Java)

Python: @property transforma métodos em atributos acessíveis diretamente.

Java: Utiliza getters e setters para controlar o acesso a atributos privados.

# Comparação de Funcionalidades

@classmethod (Python) = static methods (Java): Ambos acessam a classe, não a instância.

@staticmethod (Python) = static methods (Java): Não dependem de instâncias.

@property (Python) = Getters/Setters (Java): Encapsulam o acesso a atributos privados.

# Quando Usar Cada Um?



@classmethod / static

Afeta todos os objetos da classe.



@staticmethod / static

Função que não depende de instâncias.



@property / getters/setters

Controle de acesso a atributos.



# Conclusão

## Python

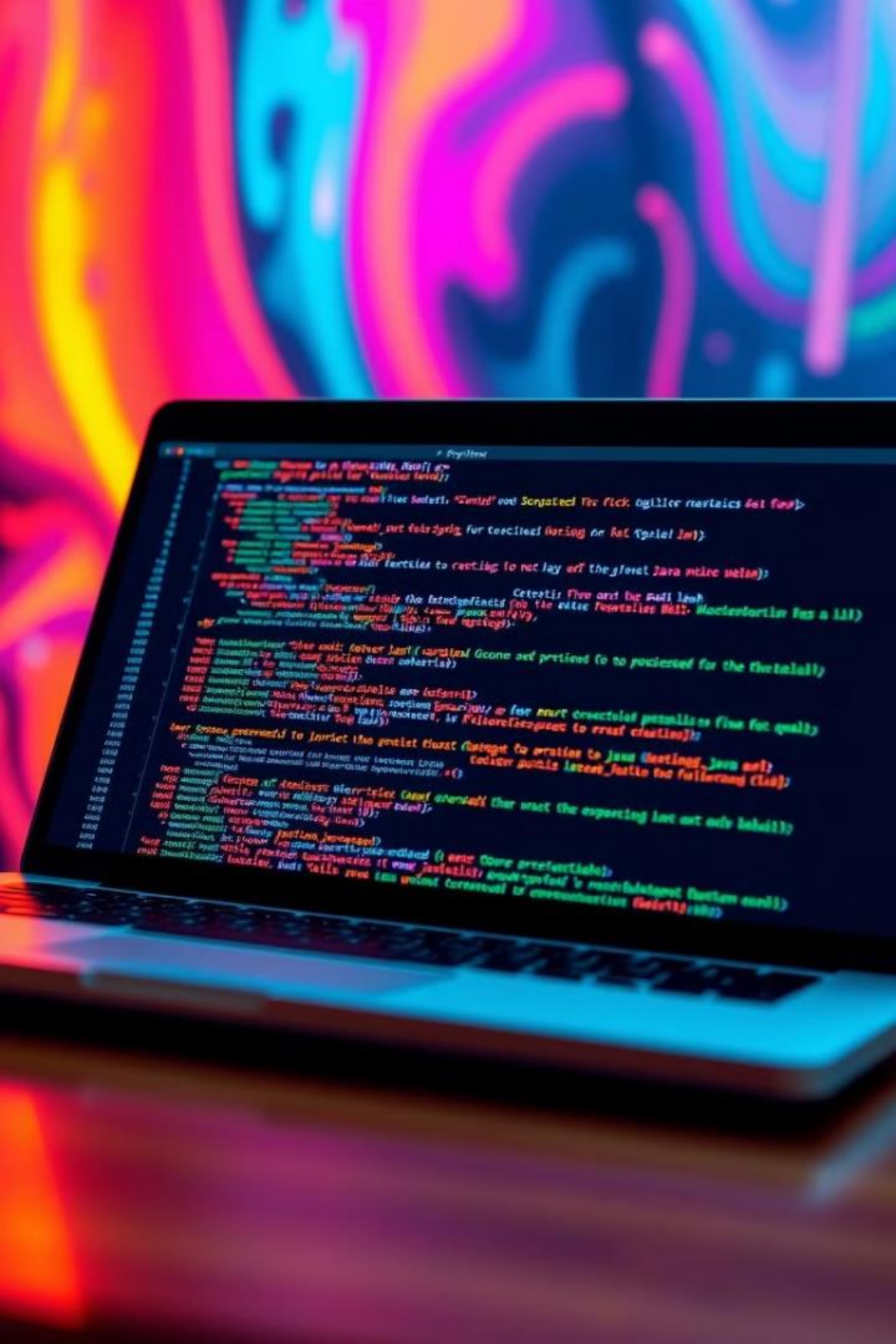
Usa decoradores para modificar o comportamento de métodos e atributos.

## Java

Usa modificadores como static e métodos de acesso (getters/setters).

## Ambos

Fornecem formas de modular e encapsular o comportamento das classes.



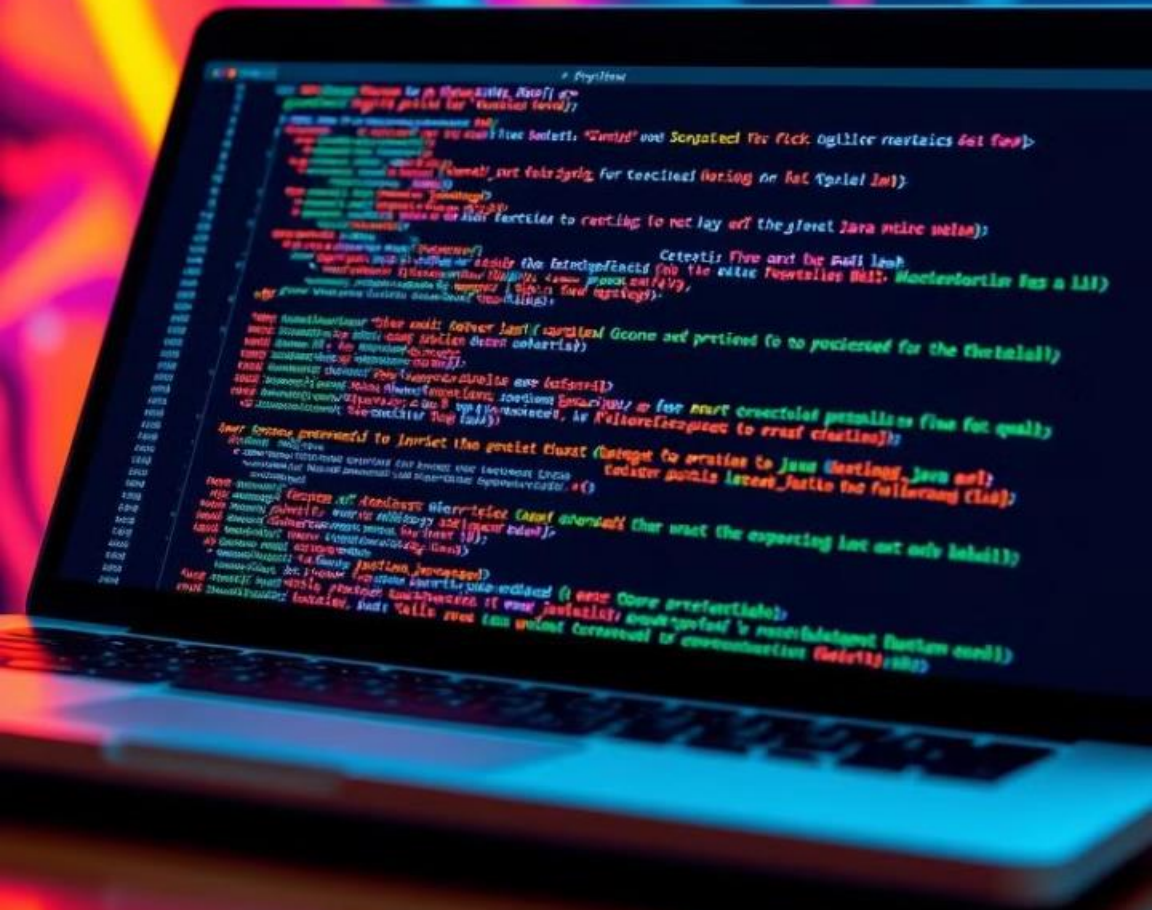
# Em Python

```
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age
        self.grades = []
```

```
def add_grade(self, grade):
    self.grades.append(grade)
```

```
def get_average_grade(self):
    if len(self.grades) == 0:
        return 0
    return sum(self.grades) / len(self.grades)
```

```
@property
def is_passing(self):
    return self.get_average_grade() >= 60
```





# Em Python

```
@classmethod
def main(cls):
    students = []

    while True:
        print("1 - Adicionar aluno")
        print("2 - Adicionar nota")
        print("3 - Verificar aprovação")
        print("4 - Sair")
        choice = int(input("Escolha uma opção: "))

        if choice == 1:
            name = input("Nome do aluno: ")
            age = int(input("Idade do aluno: "))
            student = cls(name, age) # Criando instância de Student
            students.append(student)
            print("Aluno adicionado.")
```

# Em Python

```
elif choice == 2:
    if not students:
        print("Nenhum aluno cadastrado.")
        continue
    for idx, student in enumerate(students):
        print(f"{idx + 1} - {student.name}")
    student_idx = int(input("Escolha o número do aluno: ")) - 1
    if 0 <= student_idx < len(students): # Verifica se o índice é válido
        grade = float(input("Nota do aluno: "))
        students[student_idx].add_grade(grade)
        print("Nota adicionada.")
    else:
        print("Índice de aluno inválido.")
```



# Em Python

```
elif choice == 3:
    if not students:
        print("Nenhum aluno cadastrado.")
        continue
    for student in students:
        average_grade = student.get_average_grade()
        if average_grade >= 6.0:
            status = "Aprovado"
        else:
            status = "Reprovado"
        print(f"{student.name} - Média: {average_grade} - Status: {status}")
elif choice == 4:
    print("Saindo...")
    print("Obrigado por usar o nosso sistema")
    print("Até a próxima")
    break
else:
    print("Opção inválida. Escolha novamente.")
```





## Contato



**Professor:**

Diego Ramos Inácio

**E-mail:**

[diego.inacio@univassouras.edu.br](mailto:diego.inacio@univassouras.edu.br)