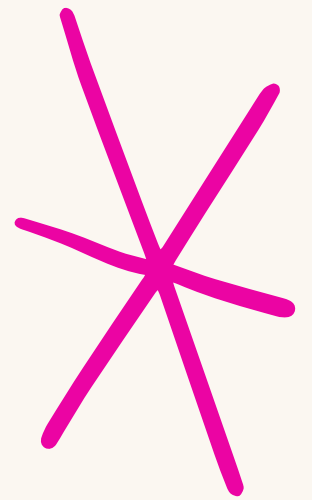
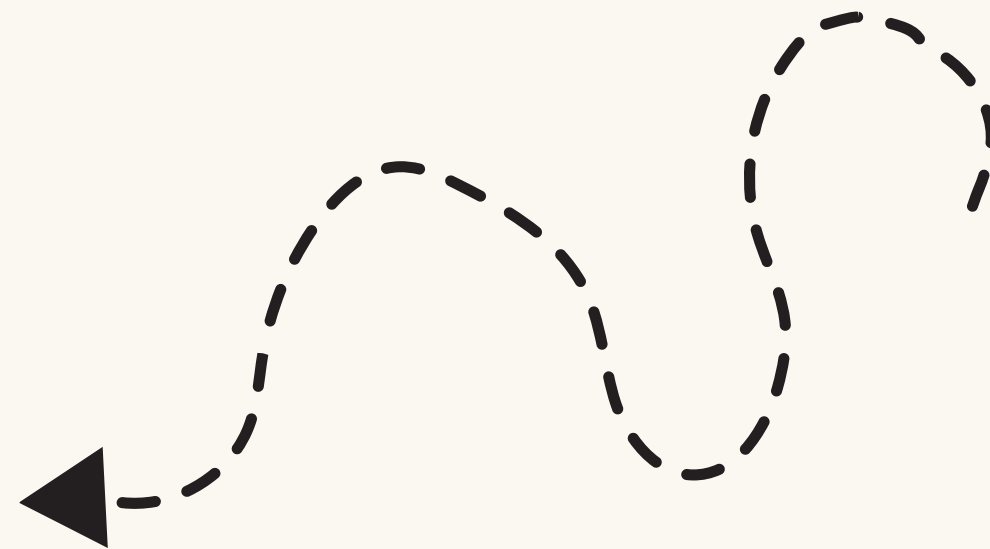


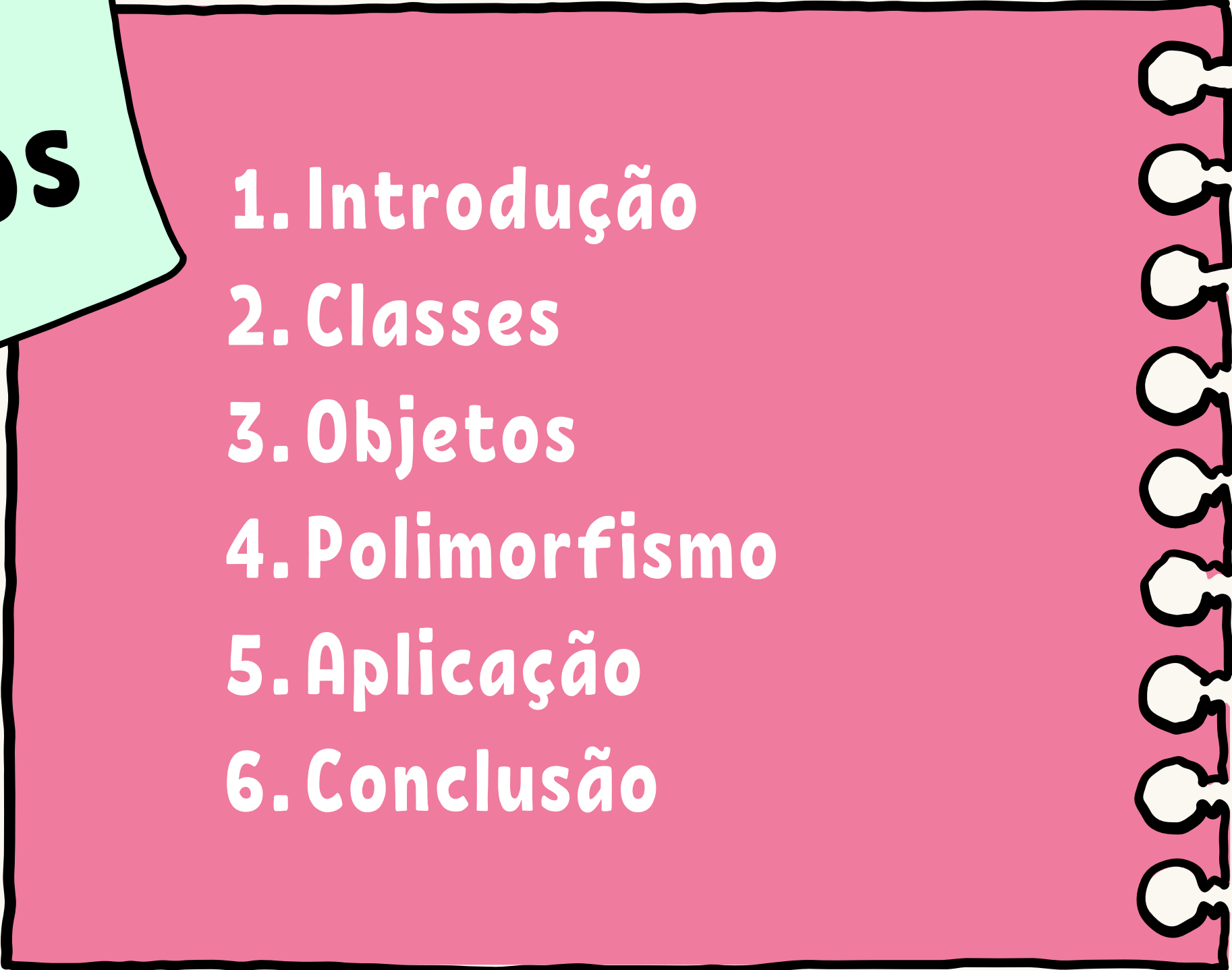
Python

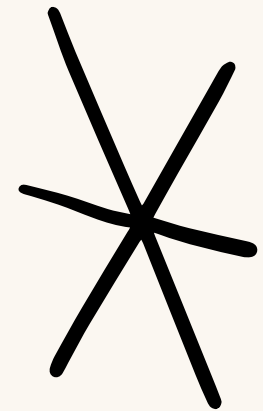


CLASSES, OBJETOS  
& POLIMORFISMO



# Tópicos

- 
1. Introdução
  2. Classes
  3. Objetos
  4. Polimorfismo
  5. Aplicação
  6. Conclusão





# O BÁSICO SOBRE:

## CLASSES

Em Python, classes são estruturas que permitem a criação de objetos que possuem propriedades e métodos associados. Elas são utilizadas para organizar e estruturar o código de forma mais modular e orientada a objetos.

## OBJETOS

Um objeto em Python é uma instância de uma classe, representando uma entidade única com atributos e comportamentos específicos.

## POLIMORFISMO

A palavra "polimorfismo" significa "muitas formas" e na programação refere-se a métodos/funções/operadores com o mesmo nome que podem ser executados em muitos objetos ou classes.

# CLASSES

Saber mais sobre:



## • O QUE SÃO?

Em Python, classes são estruturas que permitem a criação de objetos que possuem propriedades e métodos associados. Elas são utilizadas para organizar e estruturar o código de forma mais modular e orientada a objetos.

## • FUNÇÃO `__INIT__()`

Os exemplos acima são classes e objetos em sua forma mais simples e não são realmente úteis em aplicações da vida real.

Para entender o significado das classes, temos que entender a função integrada `__init__()`.

## • PARA QUE SERVEM?

As classes em Python são uma parte fundamental da programação orientada a objetos e fornecem uma maneira eficaz de modelar e organizar o código, promovendo a reutilização, a modularidade e a abstração.

## • FUNÇÃO `__STR__()`

A função `__str__()` controla o que deve ser retornado quando o objeto da classe é representado como uma string.

Se a função `__str__()` não estiver definida, a representação em string do objeto será retornada.



# OBJETOS

Saber mais sobre:



- **O QUE SÃO?**

Um objeto em Python é uma instância de uma classe, representando uma entidade única com atributos e comportamentos específicos.

- **METEDOS**

Objetos também podem conter métodos. Métodos em objetos são funções que pertencem ao objeto. Vamos criar um método na classe Usuario assegurar:

- **PARA QUE SERVEM?**

Em Python, um objeto serve para representar uma entidade única que possui características e ações específicas, baseadas em uma classe. Eles permitem a modelagem de dados e comportamentos de forma modular e reutilizável.

- **PARAMETRO SELF**

O parâmetro self é uma referência à instância atual da classe e é usado para acessar variáveis que pertencem à classe. Não precisa ser nomeado self, você pode chamá-lo como quiser, mas deve ser o primeiro parâmetro de qualquer função da classe.



## O que mais podemos fazer eles?

- Apagar Prioridades
- Apagar Objeto
- Modificar Prioridades
- Declaração de aprovação



# MAIS SOBRE OBJETOS

### Apagar Prioridades

Você pode excluir propriedades em objetos usando a palavra-chave del:

```
del p1.age
```

### Apagar Objeto

Você pode excluir objetos usando a palavra-chave del:

```
del p1
```

### Modificar Prioridades

Você pode modificar propriedades em objetos como este:

```
p1.age = 40
```

### Declaração de aprovação

As definições de classe não podem estar vazias, mas se por algum motivo você tiver uma definição de classe sem conteúdo, coloque a instrução pass para evitar erros.

```
class Person:  
    pass
```



**Usuario 1**

Nome: -Lara  
Idade: -17  
Genero: -Feminino  
Turma: -1 Pi



**Usuario 2**

Nome: -Alexandre  
Idade: -16  
Genero: -Maculino  
Turma: -2 Pi



**Usuario 3**

Nome: -Rodrigo  
Idade: -16  
Genero: -Indefinido  
Turma: -2 Pi



**Usuario 4**

Nome: -Pedro  
Idade: -16  
Genero: -Maculino  
Turma: -3 Pi





# Polimorfismo

## Saber Mais

A palavra "polimorfismo" significa "muitas formas" e na programação refere-se a métodos/funções/operadores com o mesmo nome que podem ser executados em muitos objetos ou classes.

Um exemplo de função Python que pode ser usada em diferentes objetos é a função `len()`.

1

### Funções do Polimorfismo:

#### String

Para strings `len()` retorna o número de caracteres:

```
x = "Hello World!"  
print(len(x))
```

Terminal: 12

#### Tuple

Para tuples `len()` retorna o número de itens na tuple:

```
mytuple = ("apple", "banana", "cherry")  
print(len(mytuple))
```

Terminal: 3

#### String

Para dicionários, `len()` retorna o número de pares chave/valor no dicionário:

```
thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
print(len(thisdict))
```

Terminal: 3

2

### Exemplo na pratica:

```
# Usar polimorfismo para mostrar informações  
usuarios = [usuario1, usuario2, usuario3, usuario4]  
  
# Iteração da lista de usuários  
for usuario in usuarios:  
    # Chamar o método mostrar_informacoes, o qual funciona de maneira polimórfica  
    usuario.mostrar_informacoes()
```

# Aplicação Prática

## COMO FAZER?

- 1 Primeiro devemos criar a classe, no caso eu criei a classe usuário para representar alguns dos alunos da Escola Profissional do Fundão.
- 2 De seguida devemos “dizer” á classe o que ela devera conter eu usei os termos: nome, idade, genero e turma a fim de representar os programadores.
- 3 Como já temos a calsse e o que a mesma deve conter vamos criar agora os objetos que seguiram os requisitos que prosupus a classe anteriormente.
- 4 Neste caso usei o Polimorfismo para mostrar as informações do usuário dizendo que os usuarios sao:  
O usuario 1, de seguida o usuario 2, o usuario3 e 4
- 5 Por fim vamos criar uma lista que irá repetir os requisitos da classe par cada objeto preenchidos com os seus dados, logo iremos chama-la para ser mostrada no terminal e o nosso código está feito de maneira, fácil rápida e eficiente

Classe:  
Usuario  
(aluno)

Objeto:  
nome, idade,  
genero, turma

Polimorfismo:  
usuarios  
1,2,3,4

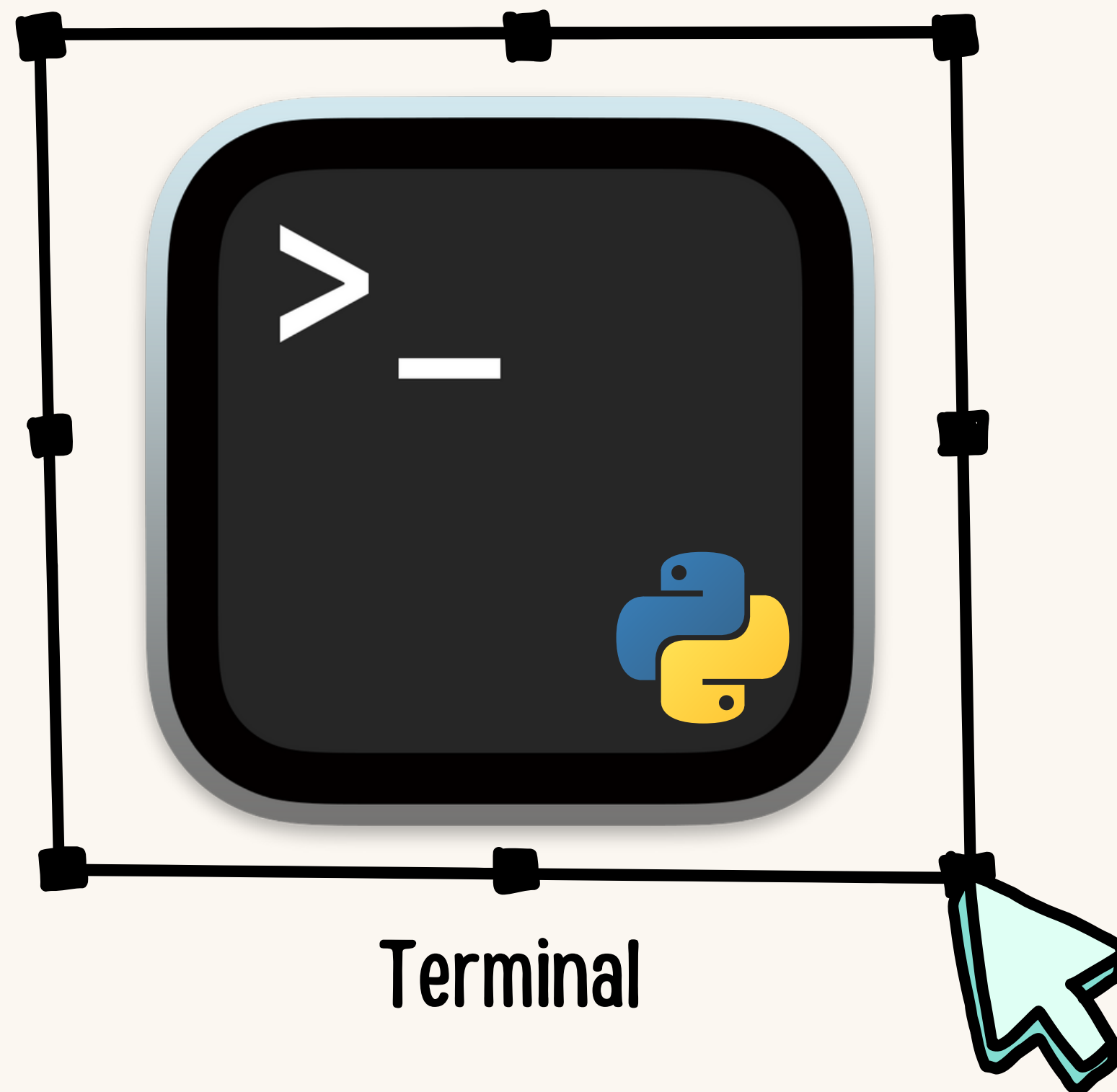
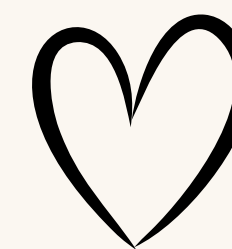
```
class Usuario:
    # Método de inicialização (construtor)
    def __init__(self, nome, idade, genero, turma):
        self.nome = nome
        self.idade = idade
        self.genero = genero
        self.turma = turma

    # Método para mostrar informações do usuário
    def mostrar_informacoes(self):
        print(f"Nome: {self.nome}")
        print(f"Idade: {self.idade}")
        print(f"Genero: {self.genero}")
        print(f"Turma: {self.turma}")
        print()
```

```
# Criar objetos
usuario1 = Usuario("Lara", 17, "Feminino", "1Pi")
usuario2 = Usuario("Alexandre", 16, "Masculino", "2Pi")
usuario3 = Usuario("Rodrigo", 16, "Indefinido", "2Pi")
usuario4 = Usuario("Pedro", 18, "Masculino", "3Pi")
```

```
# Usar polimorfismo para mostrar informações
usuarios = [usuario1, usuario2, usuario3, usuario4]

# Iteração da lista de usuários
for usuario in usuarios:
    # Chamar o método mostrar_informacoes, o qual funciona de maneira polimórfica
    usuario.mostrar_informacoes()
```



# Resultado

```
Nome: Lara  
Idade: 17  
Gênero: Feminino  
Turma: 1Pi
```

```
Nome: Alexandre  
Idade: 16  
Gênero: Masculino  
Turma: 2Pi
```

```
Nome: Rodrigo  
Idade: 16  
Genero: Indefinido  
Turma: 2Pi
```

```
Nome: Pedro  
Idade: 18  
Genero: Masculino  
Turma: 3Pi
```





PERGUNTAS



Fim! Obrigado pela atenção.