Aula 02 - POO em Python: Construtores e Instâncias

Introdução ao tema

Construtor é um método especial utilizado para criar e inicializar um objeto de uma classe.

Nesse aula vamos trabalhar com:

- Como criar um construtor.
- Diferentes tipos de construtores.
- Criando e acessando variáveis de instância.
- Modificar valores de variáveis de instância.
- Criando e acessando variáveis de classe.
- Escopo de variáveis

Construtor em Python

Um construtor é executado automaticamente na criação de um objeto.

A ideia principal de um construtor é declarar e inicializar variáveis de membro de instâncias de uma classe.

O construtor possuí um conjunto de instruções que são executadas no momento de criação de um objeto. Por exemplo, ao executar o comando `objeto_pessoa = Person()`, O Python irá chamar o construtor da classe `Person`.

Sintaxe

```
def __init__(self):
    #   Corpo do construtor
```

Exemplo Real

```
class Student:

    # Construtor e inicialização da instância de variável/atributos
    def __init__(self, name):
        print("Executando dentro do construtor...")
        self.name = name
        print("Variáveis inicializadas!")

    # Instância de Método
    def show(self):
        print("Oi, meu nome é ", self.name)

# Criando um objeto usando o construtor
objeto_1 = Student("Marcos")
objeto_1.show()
```

Saída

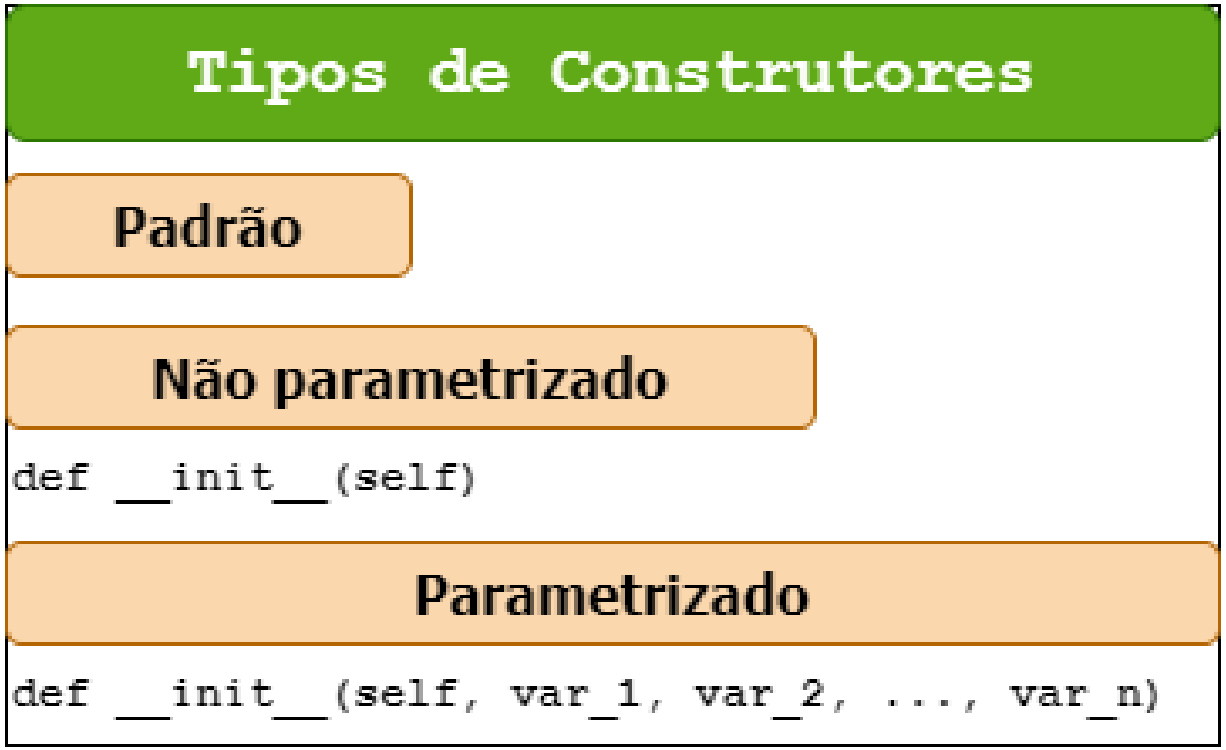
```
Executando dentro do construtor...
Variáveis inicializadas!

Oi, meu nome é Marcos
```

- O objeto "*objeto_1*" é criado usando o construtor
- Ao criar o objeto da classe `Student`, `name` é passado como argumento para o método `__init__` (construtor).
- Vários objetos da classe `Student` podem ser criados passando nomes diferentes.

Tipos de Construtores

Em Python temos três tipos de construtores:

1. Construtor padrão.
2. Construor não parametrizado.
3. Construtor parametrizado.



1. Construtor padrão

Python fornecerá um construtor padrão se nenhum construtor for definido. Ele (construtor) será adicionado por padrão quando não incluímos na classe. Não executa nenhuma tarefa, mas inicializa os objetos. É um construtor vazio sem um corpo.

Importante

- O construtor padrão não está presente no arquivo .py de origem. Ele é inserido no código durante a compilação se não existir.
- Se um construtor for implementado, o construtor padrão não será adicionado.

Exemplo Real

```
class Employee:

    def show():
        print("Dentro do método show()")

# Criando o objeto Employee (Empregado) e chamando o método Show()
empregado = Employee()
empregado.show()
```

Saída

```
Dentro do método show()
```

É possível observar que mesmo não tendo o construtor na classe é possível instânciar um objeto para a classe.

2. Construtor não parametrizado

Um construtor sem argumentos é chamado de construtor não parametrizado.

Este construtor não aceita os argumentos durante a criação do objeto. Em vez disso, ele inicializa cada objeto com o mesmo conjunto de valores.

Exemplo Real

```
class Company:

    # Construtor não parametrizado (sem argumentos)
    def __init__(self):
        self.name = "Rarolabs Serviços de Informática"
        self.address = "Rua Paul Bouthilier, 266 - Mangabeiras"

    # Método para imprimir membros de dados (atributos)
    def show(self):
        print('Nome:', self.name, '\nEndereço:', self.address)

# Instânciando objetos da classe
rarolabs = Company()
academy = Company()

# Chamando o método show de cada um dos objetos da classe
rarolabs.show()
academy.show()
```

Saída

```
Nome: Rarolabs Serviços de Informática
Endereço: Rua Paul Bouthilier, 266 - Mangabeiras

Nome: Rarolabs Serviços de Informática
Endereço: Rua Paul Bouthilier, 266 - Mangabeiras
```

É possível observar no exemplo, que não é enviado nenhum argumento para o construtor ao instãnciar o objeto.

3. Construtor parametrizado

Um construtor com parâmetros ou argumentos definidos é chamado de construtor parametrizado. Ele permite passar valores diferentes para cada objeto no momento da criação.

O primeiro parâmetro para constructor `self` é uma referência ao que está sendo construído, o restante dos argumentos são fornecidos pelo programador.

Um constructor parametrizado pode ter qualquer número de argumentos.

Exemplo Real

```
class Employee:
    # Construtor parametrizado
    def __init__(self, name, age, salary):
        self.name = name
        self.age = age
        self.salary = salary

    # Método para imprimir membros de dados (atributos)
    def show(self):
        print(f'Nome do Funcionário: {self.name}\nIdade: {self.age}\nSalário: R$ {self.salary}')

# Instânciando objetos da classe
julio_cesar = Employee('Júlio Cesar', 23, 7500)
julio_cesar.show()

kelly_joana = Employee('Kelly Joana', 41, 8500)
kelly_joana.show()
```

Saída

```
Nome do Funcionário: Júlio Cesar
Idade: 23
Salário: R$ 7500

Nome do Funcionário: Kelly Joana
Idade: 41
Salário: R$ 8500
```

É possível observar no exemplo, a definição de um construtor que recebe três argumentos (parametros).

- **Construtor com valores padrão**: Python permite definir um construtor com valores padrões. O valor padrão será usado se for passado argumentos para o construtor no momento da criação do objeto.

Exemplo Real

```
class Student:

    def __init__(self, name, age=0, classroom=1):
        self.name = name
        self.age = age
        self.classroom = classroom


    def show(self):
        print(f'Nome do Estudante: {self.name}\nIdade: {self.age}\nTurma: {self.classroom}')


roberto = Student('Roberto')
roberto.show()

andre_araujo = Student('André Araujo', 13)
andre_araujo.show()

andre_araujo = Student('Alberto Silva', 17, 4)
andre_araujo.show
```

Saída

```
Nome do Estudante: Roberto
Idade: 0
Turma: 1

Nome do Estudante: André araujo
Idade: 13
Turma: 1

Nome do Estudante: Alberto Silva
Idade: 17
Turma: 4
```

Palavra-Chave Self em Python

Ao definir métodos de instância para uma classe, usamos a palavra reservada `self` como o primeiro parâmetro. Usando self, podemos acessar a variável de instância e o método de instância do objeto.

O primeiro argumento `self` se refere ao objeto atual.

Ao chamar um método de instância por meio de um objeto, o compilador Python passa implicitamente a referência do objeto como o primeiro argumento, normalmente chamado de `self`.

Observação:

Obs: Não é obrigatório nomear o primeiro parâmetro como self. Pode-se dar qualquer nome, mas ele deve ser o primeiro parâmetro de um método de instância.

Variáveis de Instância e Variáveis de Classe

Existe diferentes formas de declaração de variáveis em Python, como:

- **Variáveis de instância**: São chamadas de campos ou atributos de um objeto.
- **Variáveis locais**: Variáveis em um método ou bloco de código.
- **Parâmetros**: Variáveis em declarações de métodos.
- **Variáveis de classe** : Esta variável é compartilhada entre todos os objetos de uma classe.

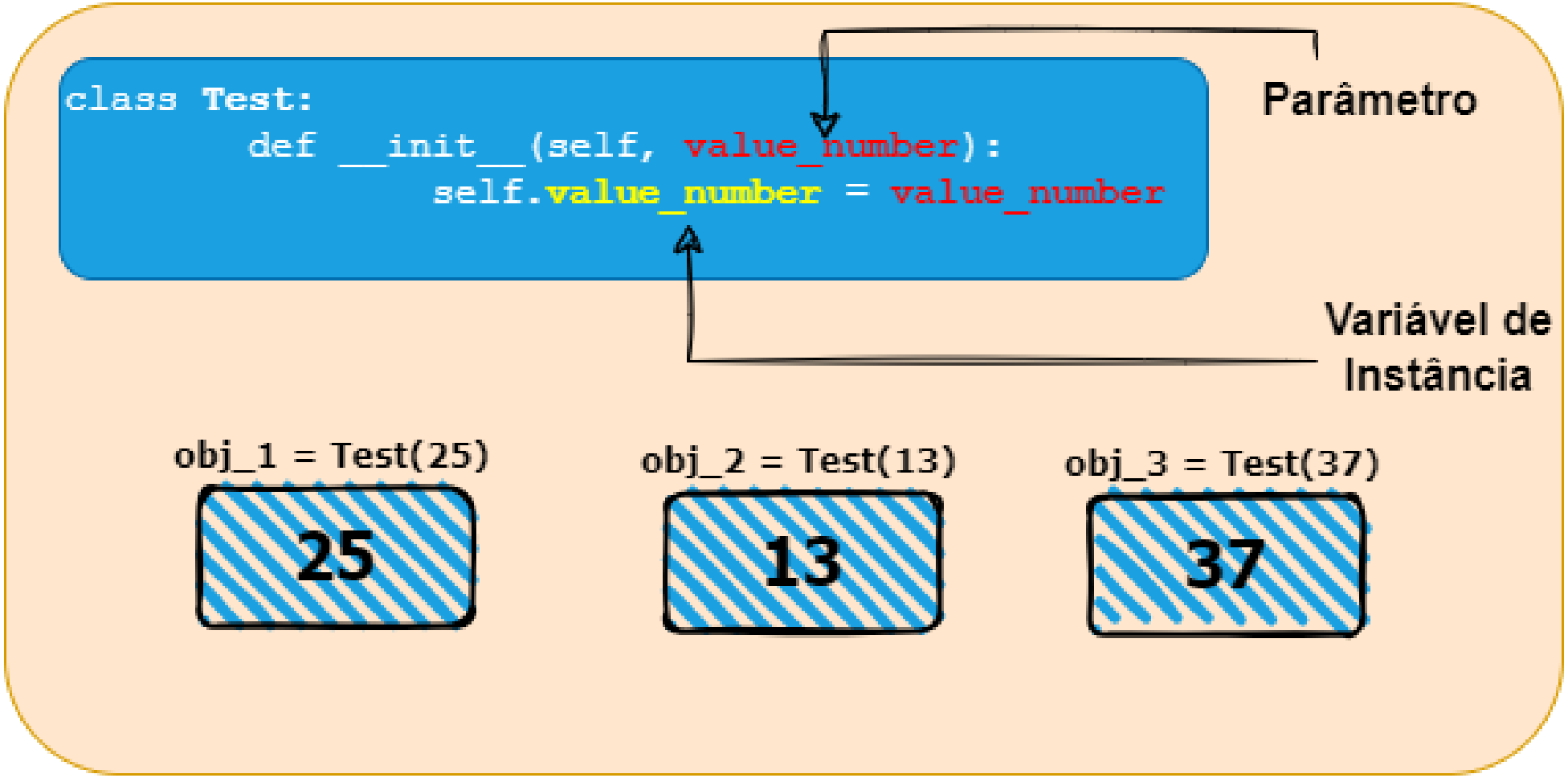
Em POO em Python, quando projetamos uma classe, usamos ***variáveis de instância*** e ***variáveis de classe***.

Variável de Instância em Python

Se o valor de uma variável varia de um objeto para outro, essas variáveis são chamadas de ***variáveis de instância***.

Variáveis de instância não são compartilhadas por objetos. Cada objeto tem sua própria cópia do atributo de instância. Isso significa que para cada objeto de uma classe, o valor da variável de instância é diferente.

Variáveis de instância são usadas dentro do método de instância.



Criando e modificando valores de Variáveis de Instâncias

Variáveis de instância são declaradas dentro de um método usando `self`. Usa-se o construtor para definir e inicializar as variáveis de instância.

A seguir é criado duas variáveis de instância chamadas `name` e `age` na classe `Student`.

Exemplo Real

```
class Student:
    # Construtor
    def __init__(self, name, age):
        # Variáveis de instância
        self.name = name
        self.age = age

obj_1 = Student("Mario", 20)

# acessando a variável/atributo
print('Nome:', obj_1.name)
print('Idade:', obj_1.age)


obj_2 = Student("Ramon", 22)

# acessando a variável/atributo
print('Nome:', obj_2.name)
print('Idade:', obj_2.age)
```

Saída

```
Nome: Mario
Idade: 20

Nome: Ramon
Idade: 22
```

É possível modificar o valor da variável de instância e atribuir um novo valor a ela usando a referência do objeto.

Quando se altera os valores da variável de instância de um objeto, apenas o objeto em questão tem seus valores alterados, isso não é refletido em outros objetos.

Exemplo Real

```
# alterando o valor dos atributos dos objetos
obj_1.age = 55
obj_2.nome = "Marcelo"

print('Nome:', obj_1.name)
print('Idade:', obj_1.age)

print('Nome:', obj_2.name)
print('Idade:', obj_2.age)
```

Saída

```
Nome: Mario
Idade: 55

Nome: Marcelo
Idade:  22
```

Variável de Classe em Python

Se o valor de uma variável não varia de objeto para objeto, chamamos esse tipo de ***variáveis de classe*** ou ***estáticas***.

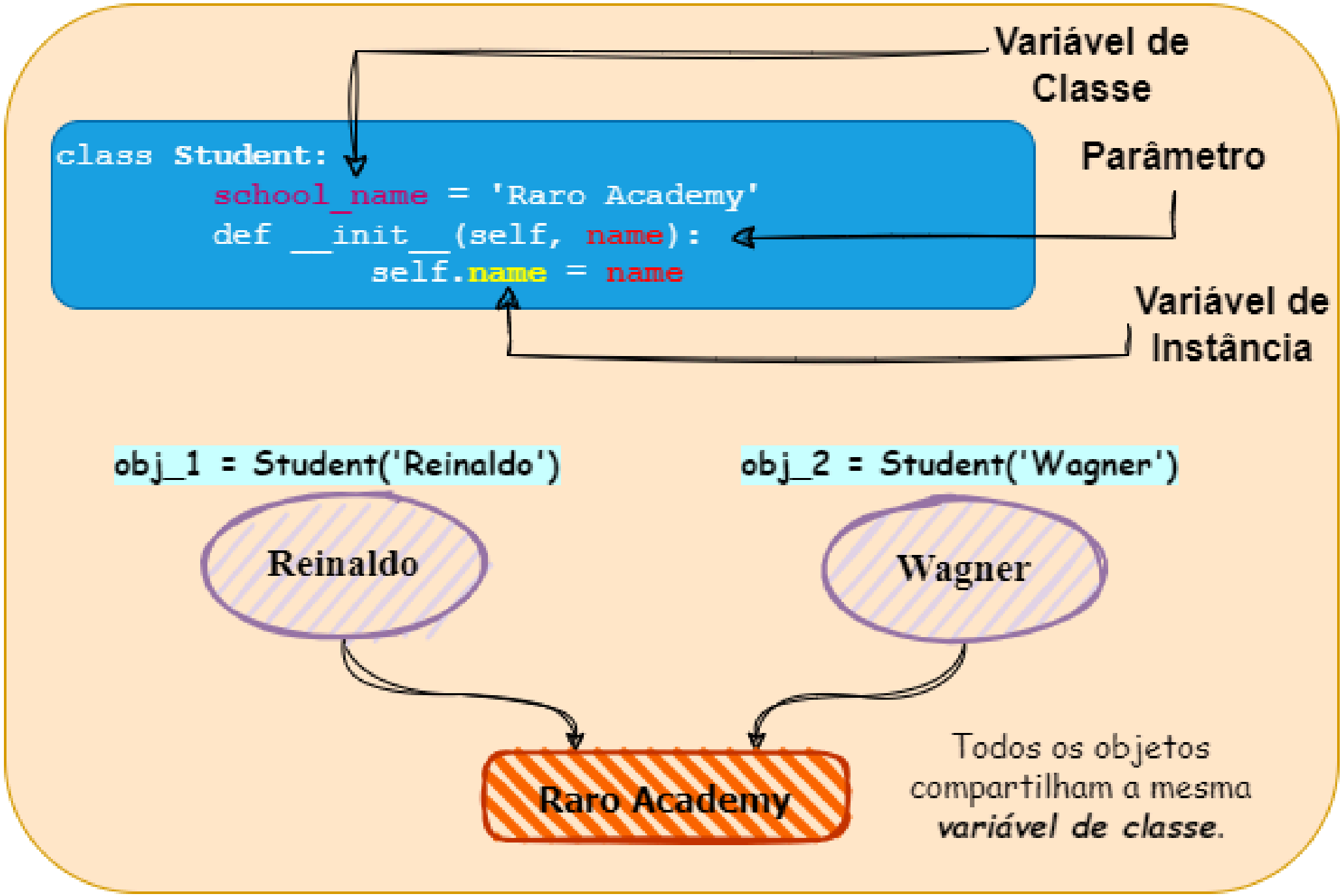
Todas as instâncias de uma classe compartilham variáveis de classe, não variando de objeto para objeto.

Em Python, variáveis de classe são declaradas quando uma classe está sendo construída. Elas ***não são definidas*** dentro de ***nenhum método de uma classe***. Por causa disso, apenas uma cópia da variável estática será criada e compartilhada entre todos os objetos de classe.

Por exemplo, na classe `Estudante()`, podemos ter diferentes ***variáveis de instância***, como `nome` e `numero_matricula`, porque o nome e o número de matrícula de cada estudante são diferentes.

No entanto, se quisermos incluir o ***nome da escola*** na classe de `Estudante()`, É preciso usar uma ***variável de classe*** em vez de uma ***variável de instância***, pois, o nome da escola é o mesmo para todos os estudantes.

Então, em vez de manter uma cópia separada em cada objeto, podemos criar uma variável de classe que manterá o nome da escola para que todos os alunos (objetos) possam compartilhá-lo.


Variável de Classe
class Student:
school_name = 'Raro Academy'
def __init__(self, name):
self.name = name
Parâmetro
Variável de Instância
obj_1 = Student('Reinaldo')
obj_2 = Student('Wagner')
Reinaldo
Wagner
Raro Academy
Todos os objetos compartilham a mesma variável de classe.

Criando e modificando valores de Variáveis de Instâncias

Os atributos (variáveis) de classe são declarados dentro da classe, no entanto, fora de qualquer metodo de instância.

É possível acessar uma atributo de classe usando o nome da classe ou o objeto instânciado da classe.

Exemplo Real

```
class Student:
    # Metódo(variável) de classe.
    school_name = 'Raro Academy'

    def __init__(self, name, roll_number):
        self.name = name
        self.roll_number = roll_number

object_student_1 = Student('Reinaldo', 10)
object_student_2 = Student('Wagner', 20)

# Acessando atributo de classe via objeto da classe
print(object_student_1.name, object_student_1.roll_number, object_student_1.school_name)

# Acessando atributo de classe via nome da classe
print(object_student_2.name, object_student_2.roll_number, Student.school_name)
```

Saída

```
Reinaldo 10 Raro Academy
Wagner 20 Raro Academy
```

Exemplo Real - *Acessando variável de classe dentro do construtor*

```
class Student:
    # Metódo(variável) de classe.
    school_name = 'Raro Academy'

    def __init__(self, name, roll_number):
        self.name = name
        self.roll_number = roll_number

        # Acessando atributo via comando self
        print(self.school_name)

        # Acessando atributo via nome da classe
        print(Student.school_name)

# Instânciando um Objeto
object_student = Student('Wagner', 20)
```

- Observer que é possível chamar a variável de classe tanto pelo nome da classe quanto pela comando `self`.

Saída

```
Raro Academy
Raro Academy
```

- **Modificando variável de classe**

Normalmente, atribuímos valor a uma variável de classe dentro da declaração de classe. No entanto, podemos alterar o valor da variável de classe na classe ou fora dela.

Observação: Deve-se alterar o valor da variável de classe usando o nome da classe.

Ao modificar uma variável de classe usando uma instância (Objeto), ***não será alterado o valor da variável de classe***. Em vez disso, Será criado uma ***variável de instância*** com o ***mesmo nome*** da variável de classe, o que oculta a ***variável de classe*** para ***ESSA INSTÂNCIA***.

Exemplo Real

```
class Student:
    # Metódo(variável) de classe.
    school_name = 'Raro Academy'

    def __init__(self, name, roll_number):
        self.name = name
        self.roll_number = roll_number


object_student_1 = Student('Reinaldo', 10)
object_student_2 = Student('Wagner', 20)

# Antes de mudar o valor do atributo de classe
print(object_student_1.name, object_student_1.roll_number, object_student_1.school_name)
print(object_student_2.name, object_student_2.roll_number, object_student_2.school_name)
```

```
#  Mudando o valor do atributo da classe pelo nome da classe
Student.school_name = 'Raro Academy - Curso Python 2024-2'

# Após mudança
print(object_student_1.name, object_student_1.roll_number, object_student_1.school_name)
print(object_student_2.name, object_student_2.roll_number, object_student_2.school_name)


# Mudando o valor do atributo da classe pelo objeto
object_student_1.school_name = 'Raro Labs'

# Após mudança 2
print(object_student_1.name, object_student_1.roll_number, object_student_1.school_name)
print(object_student_2.name, object_student_2.roll_number, object_student_2.school_name)
```

Saída

```
Reinaldo 10 Raro Academy
Wagner 20 Raro Academy

Reinaldo 10 Raro Academy - Curso Python 2024-2
Wagner 20 Raro Academy - Curso Python 2024-2

Reinaldo 10 Raro Labs
Wagner 20 Raro Academy - Curso Python 2024-2
```

Variável de classe vs. Variáveis de instância

A tabela a seguir mostra a diferença entre as variáveis de instância e de classe.

Variável de instância	Variável de classe
Variáveis de instância não são compartilhadas por objetos. Cada objeto tem sua própria cópia do atributo de instância.	Variáveis de classe são compartilhadas por todas as instâncias.
Variáveis de instância são declaradas dentro do construtor, ou seja, do __init__() método.	Variáveis de classe são declaradas dentro da definição de classe, mas fora de qualquer método de instância e construtor.
Ele é criado quando uma instância da classe é criada.	Ele é criado quando o programa começa a ser executado.
Alterações feitas nessas variáveis por meio de um objeto não serão refletidas em outro objeto.	Alterações feitas na variável de classe serão refletidas em todos os objetos.