**Conceito entre Polimorfismo e Herança**

Herança em Python

Primeiramente, nós vamos entender, sobre o que é Herança, em Python, a herança ela é uma definição de propriedades e métodos, que promove e facilita a manutenção e consegue criar uma hierarquia de código apresentado de uma classe objeto!

**Vamos entender como funciona esse processo da Herança:**

* Primeiramente, a gente temos que definir uma **classe** que ele quer ocupar, e assim chamaremos qualquer nome da classe desejada! Assim, por diante inicia o começo da Classe do Objeto.
* Usamos o método Construtor, chamado: **\_\_init\_\_.** Que podemos criar atributos dentro da instância, não esqueceremos de chamar o **Método (def)** antes dele, porque é essencialpara definir o construtor da classe e os atributos dentro dela, que são os argumentos do construtor.
* Criamos o **self** que se refere a instância da nossa classe criada, os nossos atributos. Para que a gente conseguisse atribuir algum valor referindo a propriedades da instância.
* Temos também o método construtor **super().** Serve para chamar uma classe original, ou seja, uma subclasse que é capaz de chamar o construtor da primeira classe e atribuir novas instâncias e atributos, dentro da nossa classe original.

**Com base nisso, nesses tópicos nós conseguimos definir os detalhes da classe.**

Por exemplo, eu, o Alyson, vou criar uma classe chamado Animais dentro do Python, para mostrar um exemplo básico, de como é feito desse sistema de Herança, logo abaixo:

**Meu código:**

class Animais: # Definindo a Classe Animais

    def \_\_init\_\_(self, nome, idade):

        self.nome = nome

        self.idade = idade

    def \_\_str\_\_(self):

        return f'Nome: {self.nome}, Idade: {self.idade}'

class Cachorro(Animais): # Subclasse de Cachorro

    def \_\_init\_\_(self, nome, raça, idade):

        super().\_\_init\_\_(nome, idade)

        self.raça = raça

    def \_\_str\_\_(self):

        return f'Nome: {self.nome}, Raça: {self.raça}, Idade: {self.idade}'

class Gato(Animais): # Subclasse de Gato

    def \_\_init\_\_(self, nome, raça, idade):

        super().\_\_init\_\_(nome, idade)

        self.raça = raça

    def \_\_str\_\_(self):

        return f'Nome: {self.nome}, Raça: {self.raça}, Idade: {self.idade}'

# Um lugar para criar instâncias corretamente de cada um deles.

Cachorro = Cachorro('Suzy', 'Canine', 4)

Gato = Gato('Branquelo', 'Felino', 2)

print(Cachorro)

print(Gato)

Polimorfismo em Python

Agora, nós vamos entender, o que significa o Polimorfismo em Python, bem diferente do que a herança, que acabamos de conhecer agora, ele é uma função polimórfica, que aceita qualquer objeto dentro da classe. Ou seja, ela é a capacidade de criar uma subclasse e dar à vida aos nossos atributos dentro de uma superclasse, para saber qual método definido deve ser invocado.

* Definição de **Subclasse** que herda os mesmos objetivos da **Superclasse (classe original).**
* Consegue herdar novos métodos e atributos se for para criar e definir a **Subclasse.**

Bom, chegamos na conclusão, que o significado de Polimorfismo é Permitir que métodos tenham o mesmo nome, mas comportamentos diferentes, dependendo da classe desejada!

Vamos mostrar, o exemplo, disso:

**Meu código:**

class Console:  # Criação da classe Console de VideoGame

    def \_\_str\_\_(self):

        return (f'1- {Playstation1()}\n'

                f'2- {Playstation2()} \n'

                f'3- {Playstation3()} \n'

                f'4- {Playstation4()} \n'

                f'5- {Playstation5()} \n')

class Playstation1 (Console): # Criação de Subclasse que herda ao nosso SuperClasse

    def \_\_str\_\_(self):

        return 'Playstation 1'

class Playstation2 (Console):

    def \_\_str\_\_(self):

        return 'Playstation 2'

class Playstation3 (Console):

    def \_\_str\_\_(self):

        return 'Playstation 3'

class Playstation4 (Console):

    def \_\_str\_\_(self):

        return 'Playstation 4'

class Playstation5 (Console):

    def \_\_str\_\_(self):

        return 'Playstation 5'

# Instância

console = Console()

print(console)