Herança

Em Python, a herança é implementada por meio da declaração da classe filha, que herda de uma classe pai. A classe filha, também conhecida como subclasse, herda todos os atributos e métodos da classe pai, também chamada de superclasse. Isso significa que a subclasse pode acessar e utilizar todos os atributos e métodos da superclasse, além de adicionar novos atributos e métodos específicos, se necessário.

Para utilizar a herança em Python, é necessário seguir alguns passos. Primeiro, declaramos a classe pai com seus atributos e métodos. Em seguida, declaramos a classe filha, utilizando a palavra-chave “class” seguida do nome da classe filha e o nome da classe pai entre parênteses, indicando a herança. Dentro da classe filha, podemos adicionar novos atributos e métodos ou sobrescrever os existentes da classe pai.

Um exemplo prático de utilização da herança em Python é criar uma classe Animal como superclasse, com atributos e métodos genéricos para todos os animais, como nome, idade e método de fazer som. Em seguida, podemos criar subclasses como Cachorro, Gato e Pássaro, que herdam os atributos e métodos da classe Animal, mas também podem ter atributos e métodos específicos, como latir, miar e voar, respectivamente.

Exemplo:

class Personagem: #Definimos a classe Personagem

    def \_\_init\_\_(self, nome, vida): #metodo construtor \_\_init\_\_ e atribuímos os    parametros nome e vida

        self.nome = nome

        self.vida = vida

class Heroi(Personagem):  # Herói herda de Personagem

    def \_\_init\_\_(self, nome, vida, habilidade):

        super().\_\_init\_\_(nome, vida)  # Chamando o construtor da classe mãe

        self.habilidade = habilidade

heroi1 = Heroi("Superman", 100, "Voo")  # Criando um herói

print(heroi1.nome)  # Saída: Superman

print(heroi1.vida)  # Saída: 100

print(heroi1.habilidade)  # Saída: Voo

A Classe Herói herda(extende) a classe Personagem, ou seja, herói vira classe filho e Personagem vira classe pai. Então, a classe Herói herda os atributos da classe pai, o nome e a vida e tem um atributo exclusivo dela, a habilidade.

Polimorfismo

O polimorfismo é outro conceito importante na programação orientada a objetos e também está presente na linguagem de programação Python. O polimorfismo permite que um objeto possa ser tratado de diferentes maneiras, dependendo do contexto em que é utilizado.

Em Python, o polimorfismo é alcançado por meio do uso de métodos com o mesmo nome em diferentes classes, mas com implementações diferentes. Isso significa que um método pode ter comportamentos diferentes em classes diferentes, permitindo que o mesmo método seja chamado para objetos de diferentes classes, produzindo resultados diferentes.

O polimorfismo é especialmente útil quando se trabalha com herança. Por exemplo, considerando as classes Cachorro, Gato e Pássaro mencionadas anteriormente, todas elas podem ter um método chamado “emitir som”, mas cada uma delas pode ter uma implementação diferente desse método. Quando chamamos o método “emitir som” para um objeto de uma dessas classes, o comportamento será diferente dependendo da classe do objeto.

Exemplo:

class Personagem:

  def \_\_init\_\_(self, nome, vida):

      self.nome = nome

      self.vida = vida

  class Heroi(Personagem):

      def \_\_init\_\_(self, nome, vida, habilidade):

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

          self.habilidade = habilidade

  class Vilao(Personagem):  # Adicionando a classe Vilão

      def \_\_init\_\_(self, nome, vida, poder):

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

          self.poder = poder

  def atacar(personagem):  # Função para atacar, pode ser chamada por heróis ou vilões

      print(f"{personagem.nome} está atacando!")

  heroi1 = Heroi("Superman", 100, "Voo")

  vilao1 = Vilao("Lex Luthor", 80, "Inteligência")

  atacar(heroi1)  # Chamando a função atacar() com um herói

  atacar(vilao1)  # Chamando a função atacar() com um vilão

A classe filho e a classe herói herdam o atributo da classe Personagem, ou seja ambos podem atacar.

class Personagem:

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

        self.nome = nome

        self.vida = vida

    def atacar(self):  # Método genérico de ataque

        pass  # Esse método pode ser sobrescrito pelas subclasses

class Heroi(Personagem):

    def \_\_init\_\_(self, nome, vida, habilidade):

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

        self.habilidade = habilidade

    def atacar(self):  # Sobrescrevendo o método atacar na classe Heroi

        print(f"{self.nome} está atacando com sua habilidade de {self.habilidade}!")

class Vilao(Personagem):  # Adicionando a classe Vilão

    def \_\_init\_\_(self, nome, vida, poder):

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

        self.poder = poder

    def atacar(self):  # Sobrescrevendo o método atacar na classe Vilao

        print(f"{self.nome} está atacando com seu poder de {self.poder}!")

# Função de ataque agora chama o método específico de cada personagem

def atacar(personagem):

    personagem.atacar()

# Criando instâncias dos personagens

heroi1 = Heroi("Superman", 100, "Voo")

vilao1 = Vilao("Lex Luthor", 80, "Inteligência")

# Chamando a função atacar() com diferentes tipos de personagens

atacar(heroi1)  # Chamando o método atacar do herói

atacar(vilao1)  # Chamando o método atacar do vilão

Agora o método atacar se comporta diferente nas classes filhas, a Classe Herói quando chama o método atacar ele exibe a habilidade, já quando a Classe vilão chama a o método atacar ele exibe o poder.

Diferenças

A Herança permite criar novas classes com base em classes existentes, promovendo a reutilização de código e estabelecendo relações hierárquicas entre objetos. Com a herança, você pode economizar tempo e esforço, evitando a duplicação de código e mantendo seu código mais organizado.

O Polimorfismo, por sua vez, possibilita tratar objetos de diferentes classes de forma uniforme. Isso significa que, independentemente do tipo específico do objeto, você pode trabalhar com ele de maneira flexível e consistente, simplificando o código e tornando-o mais robusto.

Ou seja, a herança ocorre quando uma classe herda (baita redundância, mas tudo bem) atributos ou métodos de sua superclasse, ou seja, o comportamento de sua superclasse.

Já o polimorfismo ocorre quando uma classe redefine o comportamento de sua superclasse, sobrescrevendo métodos ou alterando o valor de atributos, por exemplo.

Ou seja, o polimorfismo pode ocorrer à partir da herança.

Vantagens

Para facilitar a generalização de algoritmos e estruturas de dados. Imagine que sem ela teria que ter um método que recebe JanelaPrincipal, outro método que faz exatamente a mesma coisa e muda só que ele recebe JanelaCliente, outro para JanelaProduto, e assim por diante. Seria um absurdo. O polimorfismo permite usar um só método que se vira para escolher o código correto a usar.

Além disso, uma das principais vantagens é a reutilização de código, uma vez que é possível criar classes genéricas que podem ser utilizadas de maneira flexível em diferentes contextos. Além disso, o polimorfismo torna o código mais legível e organizado, facilitando a manutenção e a evolução do sistema.

* Reutilização de código: As subclasses herdam os membros da superclasse, o que reduz a duplicação de código.
* Extensibilidade: Novas funcionalidades podem ser adicionadas facilmente às subclasses, sem modificar a superclasse.
* Organização: A hierarquia de classes facilita a compreensão e a manutenção do código.

Referências

<https://www.dio.me/articles/desvendando-a-magia-da-heranca-e-polimorfismo-em-java>

<https://www.dio.me/articles/heranca-e-polimorfismo-em-c>

<https://www.alura.com.br/apostila-python-orientacao-a-objetos/heranca-e-classes-abstratas?srsltid=AfmBOorXn_QEWdzuc0ysZBY-sND2RXrMa-N9-nAalvm8v6KJKB0KfogI>

<https://labvital.com.br/glossario/o-que-e-polimorfismo/#:~:text=Vantagens%20do%20Polimorfismo&text=Uma%20das%20principais%20vantagens%20%C3%A9,e%20a%20evolu%C3%A7%C3%A3o%20do%20sistema>

<https://www.dio.me/articles/programacao-orientada-a-objetospoo-heranca>