Actividad 4

Herencia

Explicación de la herencia y su importancia en la reutilización de código

En el contexto de la Programación Orientada a Objetos, la herencia es un mecanismo fundamental que permite la creación de nuevas clases basadas en clases existentes. La clase recién creada hereda los atributos y métodos de la clase base (o superclase), lo que facilita la reutilización de código y promueve la extensibilidad del software.

De nición de Herencia: **La herencia es un principio de la POO que permite la creación de una nueva clase (clase hija o subclase) basada en una clase ya existente (clase padre o superclase). La clase hija hereda los atributos y métodos de la clase padre, y puede agregar, modi car o extender su funcionalidad.**

Explicación de la herencia y su importancia en la reutilización de código

Características Clave: **Clase Padre (Superclase): La clase existente de la cual se heredan atributos y métodos. Clase Hija (Subclase): La nueva clase que se crea, basada en la clase padre, heredando sus características.**

```python
class Animal:
    def __init__(self, nombre):
        self.nombre = nombre

    def hacer_sonido(self):
        pass

class Perro(Animal):
    def hacer_sonido(self):
        return "Guau"
```

Sintaxis en Python:

En Python, la herencia se logra mediante la declaración de la clase hija y la especificación de la clase padre entre paréntesis después del nombre de la clase hija.

+ info

En este ejemplo, la clase Perro hereda de la clase Animal. La clase Perro tiene acceso al atributo nombre y al método hacer_sonido de la clase Animal.

Explicación de la herencia y su importancia en la reutilización de código

Importancia en la Reutilización de Código:

- **Evita Duplicación de Código:** la herencia permite la creación de clases especializadas que heredan comportamientos comunes de una clase base. Esto evita duplicar código, ya que la lógica compartida está en la clase base.
- **Promueve la Consistencia:** al centralizar la funcionalidad común en la clase base, cualquier cambio realizado en la lógica se re eja automáticamente en todas las clases hijas
- **Facilita la Extensibilidad:** Se pueden crear nuevas clases basadas en clases existentes para agregar funcionalidad adicional sin modi car la clase original.

Explicación de la herencia y su importancia en la reutilización de código

Ejemplo de Aplicación Práctica: Supongamos que tenemos una clase Vehículo que tiene atributos y métodos comunes a todos los vehículos. Podemos crear clases hijas específicas como Coche y Camión que hereden de Vehículo y luego agregar atributos y métodos específicos para cada tipo de vehículo.

+ info

```python
class Vehiculo:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo

    def arrancar(self):
        print(f"{self.marca} {self.modelo} ha arrancado.")

class Coche(Vehiculo):
    def acelerar(self):
        print(f"{self.marca} {self.modelo} está acelerando.")

class Camion(Vehiculo):
    def cargar(self):
        print(f"{self.marca} {self.modelo} está cargando mercancía.")
```

En este caso, las clases Coche y Camión heredan de la clase Vehículo. Esto facilita la reutilización de código, ya que las características comunes de todos los vehículos se encuentran en la clase base Vehículo.

Explicación de la herencia y su importancia en la reutilización de código

Ejercicio: Definición de Clases con Herencia

Define una clase base llamada Animal con atributo nombre.

Crea dos clases hijas, Perro y Gato, que hereden de Animal.

Agrega un método hacer_sonido en cada clase hija que devuelva un sonido característico.

Solución:

```
class Animal:
    def __init__(self, nombre):
        self.nombre = nombre

class Perro(Animal):
    def hacer_sonido(self):
        return "Guau"

class Gato(Animal):
    def hacer_sonido(self):
        return "Miau"
```

Explicación de la herencia y su importancia en la reutilización de código

Ejercicio: Uso de Métodos Heredados

Extiende el ejercicio anterior.

Crea instancias de las clases Perro y Gato, y llama al método hacer_sonido para cada una.

Accede al atributo nombre de las instancias. **Solución**:

```
# Código adicional al ejercicio 1
perro = Perro("Buddy")
gato = Gato("Whiskers")

# Llamadas a métodos y acceso a atributos
print(f"{perro.nombre} hace: {perro.hacer_sonido()}")
print(f"{gato.nombre} hace: {gato.hacer_sonido()}")
```

Explicación de la herencia y su importancia en la reutilización de código

Ejercicio: Herencia Múltiple

Define dos clases base, Vehículo y Eléctrico.
Crea una clase híbrida llamada Coche Eléctrico que herede de ambas clases.
Agrega atributos y métodos específicos para Coche Electrico.

Solución:

```python
class Vehiculo:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo

class Electrico:
    def __init__(self, autonomia):
        self.autonomia = autonomia

class CocheElectrico(Vehiculo, Electrico):
    def __init__(self, marca, modelo, autonomia):
        Vehiculo.__init__(self, marca, modelo)
        Electrico.__init__(self, autonomia)

    def cargar_bateria(self):
        print(f"{self.marca} {self.modelo} está cargando la batería.")
```

Explicación de la herencia y su importancia en la reutilización de código

Ejercicio: Herencia Múltiple

Define dos clases base, Vehículo y Eléctrico. Crea una clase híbrida llamada Coche Eléctrico que herede de ambas clases. Agrega atributos y métodos específicos para Coche Electrico.

Solución:

+ info

```
# Código adicional al ejercicio 1
class Gato(Animal):
    def hacer_sonido(self):
        return "Ronroneo"

# Creación de instancias y llamada a métodos
gato = Gato("Whiskers")
print(f"{gato.nombre} hace: {gato.hacer_sonido()}")
```

Estos ejercicios te permitirán practicar conceptos clave relacionados con la herencia en Python y la POO. Puedes adaptar la complejidad según el nivel de los participantes.

TIC

TALENTO TECH

Universidad Tecnológica de Pereira

faceIT