Vererbung in Python

Definition :

Vererbung in Python ist ein grundlegendes Konzept der objektorientierten Programmierung (OOP), das es einer Klasse (genannt Kind- oder Unterklasse) ermöglicht, Attribute und Methoden von einer anderen Klasse (genannt Eltern- oder Oberklasse) zu erben.

Dies ermöglicht die Wiederverwendung von Code und die Erstellung einer hierarchischen Beziehung zwischen Klassen.

Hier ist eine Übersicht über Vererbung in Python:

- **Oberklasse (Elternklasse)**: Die Klasse, von der geerbt wird.
- **Unterklasse (Kindklasse)**: Die Klasse, die von der Oberklasse erbt.

Vererbung in Python

Syntax :

Um eine Unterklasse zu definieren, die von einer Oberklasse erbt, verwenden Sie die folgende Syntax:

```
class Oberklasse:
    # Attribute und Methoden der Oberklasse
    pass

class Unterklasse(Oberklasse):
    # Zusätzliche Attribute und Methoden der Unterklasse
    pass
```

Vererbung in Python

Beispiel :

```
# Definition der Oberklasse
class Animal:
    def __init__(self, name):
        self.name = name

    def speak(self):
        return f"{self.name} sagt Hallo!"

# Definition einer Unterklasse
class Dog(Animal):
    def speak(self):
        return f"{self.name} sagt Woof!"

# Definition einer weiteren Unterklasse
class Cat(Animal):
    def speak(self):
        return f"{self.name} sagt Meow!"
```

Vererbung in Python

Beispiel :

```
# Erstellen von Instanzen der Unterklassen
dog = Dog("Buddy")
cat = Cat("Whiskers")

print(dog.speak())  # Output: Buddy sagt Woof!
print(cat.speak())  # Output: Whiskers sagt Meow!

t = Animal("Tiger")
print(t.speak())  # Output: Tiger sagt Hallo!
```

Vererbung in Python

Wichtige Punkte :

Vererbung von Attributen und Methoden: Die Unterklasse erbt alle Attribute und Methoden der Oberklasse, kann jedoch auch zusätzliche Attribute und Methoden haben oder vorhandene überschreiben.

Methodenüberschreibung: Eine Unterklasse kann eine in der Oberklasse definierte Methode überschreiben. Dies geschieht, indem in der Unterklasse eine Methode mit demselben Namen wie in der Oberklasse definiert wird.

Die super()-Funktion: Diese Funktion wird verwendet, um eine Methode (z.B. Konstruktor) aus der Oberklasse aufzurufen. Dies ist besonders nützlich, um die Funktionalität der geerbten Methoden zu erweitern.

Vererbung in Python

Wichtige Punkte :

Mehrfachvererbung: Python unterstützt Mehrfachvererbung, wobei eine Unterklasse von mehr als einer Oberklasse erben kann.

Dies kann zu komplexen Szenarien führen und sollte mit Bedacht verwendet werden.

Method resolution order :

In Python definiert die Methodenauflösungsreihenfolge die Reihenfolge, in der die Basisklassen bei der Ausführung einer Methode durchsucht werden. Zuerst wird die Methode oder das Attribut innerhalb einer Klasse gesucht und dann folgt es der Reihenfolge, die wir beim Erben angegeben haben. Diese Reihenfolge wird auch als Linearisierung einer Klasse bezeichnet und die Regelsätze werden als MRO (Method Resolution Order) bezeichnet.

https://www.geeksforgeeks.org/method-resolution-order-in-python-inheritance/

https://www.python.org/download/releases/2.3/mro/

Vererbung in Python

Wichtige Punkte :

Die isinstance()-Funktion: Diese Funktion überprüft, ob ein Objekt eine Instanz einer Klasse oder einer Tupel von Klassen ist.

Die issubclass()-Funktion: Diese Funktion überprüft, ob eine Klasse eine Unterklasse einer anderen Klasse oder eines Tupels von Klassen ist.

Vererbung in Python

Beispiel - MRO:

```
class Grandpa(): pass


class BaseA(Grandpa): pass


class BaseB(): pass


class Sub(BaseA, BaseB): pass


print(Sub.mro())

# Ausgabe
# [<class '__main__.Sub'>, <class '__main__.BaseA'>, <class '__main__.Grandpa'>, <class '__main__.BaseB'>, <class 'object'>]
```

Vererbung in Python

Beispiel – super() :

```
class Tier:
    def __init__(self, name):
        self.name = name

    def eat(self):
        print("Ich habe hunger")


class Hund(Tier):
    def __init__(self, name, rasse):
        # Parent (Oberklasse)-Konstruktor-Aufruf
        super().__init__(name)

        self.rasse = rasse

    def sprechen(self):
        # Parent (Oberklasse)-Methode-Aufruf
        super().eat()

        return f"{self.name} sagt Wuff! Ich bin ein {self.rasse}."


hund = Hund("Buddy", "Golden Retriever")
print(hund.sprechen())
```

```
# Ausgabe:
# Ich habe hunger
# Buddy sagt Wuff! Ich bin ein Golden Retriever.
```

Vererbung in Python

Beispiel – isinstance() , issubclass():

```
# Definition der Oberklasse
class Fahrzeug:
    def __init__(self, marke):
        self.marke = marke


# Definition einer Unterklasse
class Auto(Fahrzeug):
    def __init__(self, marke, modell):
        super().__init__(marke)
        self.modell = modell


# Definition einer weiteren Unterklasse
class Fahrrad(Fahrzeug):
    def __init__(self, marke, typ):
        super().__init__(marke)
        self.typ = typ
```

```
# Erstellen von Instanzen der Unterklassen
auto = Auto("BMW", "X5")
fahrrad = Fahrrad("Cannondale", "Mountainbike")

# Beispiel für isinstance()
print(isinstance(auto, Auto))  # Ausgabe: True
print(isinstance(auto, Fahrzeug))  # Ausgabe: True
print(isinstance(auto, Fahrrad))  # Ausgabe: False

print(isinstance(fahrrad, Fahrrad))  # Ausgabe: True
print(isinstance(fahrrad, Fahrzeug))  # Ausgabe: True
print(isinstance(fahrrad, Auto))  # Ausgabe: False

# Beispiel für issubclass()
print(issubclass(Auto, Fahrzeug))  # Ausgabe: True
print(issubclass(Fahrrad, Fahrzeug))  # Ausgabe: True
print(issubclass(Fahrzeug, Auto))  # Ausgabe: False
print(issubclass(Fahrzeug, Fahrzeug))  # Ausgabe: True
```

Vererbung in Python

Zusammenfassung

Vererbung ermöglicht eine hierarchische Klassenorganisation und die Wiederverwendung von Code, was Ihren Code modularer und leichter verwaltbar macht. Das Verständnis, wie man Vererbung richtig einsetzt, kann Ihre Fähigkeit zur Gestaltung robuster objektorientierter Anwendungen in Python erheblich verbessern.

Mehr erkunden :

https://www.python-lernen.de/vererbung-python.htm

https://www.w3schools.com/python/python_inheritance.asp

https://www.programiz.com/python-programming/inheritance