



Programming 2

Dries Decuyper
Serhat Erdogan
Janne Gilis
Bart Thumas

Overriding
Super

AGENDA

- Overriding
- Super



Overriding

- Definition
- Same Method Signature
- Inheritance Requirement
- Access to Superclass Method

DEFINITION

- Overriding refers to the ability of a subclass to provide a specific implementation for a method that is already defined in its superclass
- It allows a subclass to replace or extend the behavior of a method inherited from the superclass

SAME METHOD SIGNATURE

```
class Animal:
    def speak(self):
        print("Animal speaks")

class Dog(Animal):
    def speak(self):
        print("Dog barks")

my_dog = Dog()
my_dog.speak()  # Output: Dog barks
```

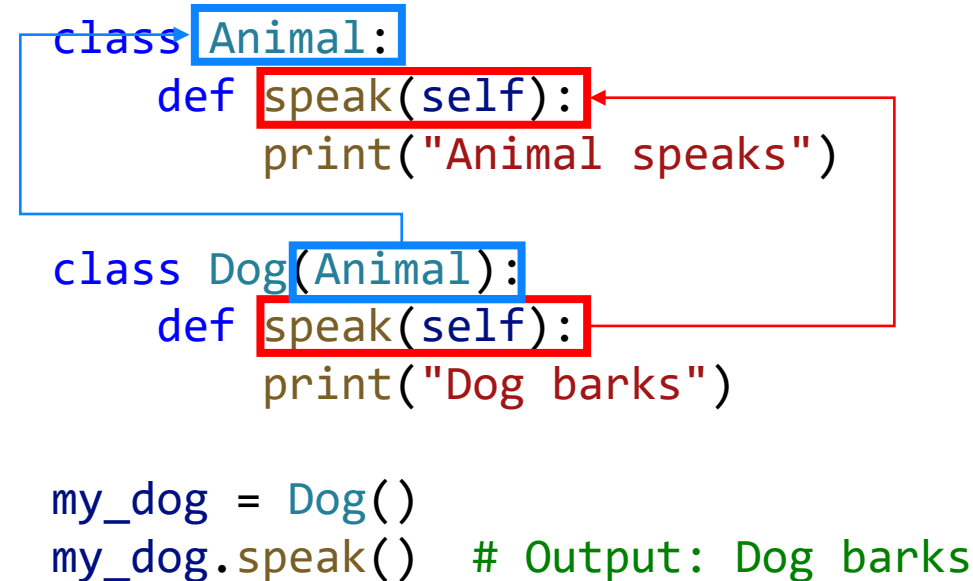
- Overriding occurs when a subclass provides a method with the same name, parameters, and return type as a method in its superclass

INHERITANCE REQUIREMENTS

```
class Animal:
    def speak(self):
        print("Animal speaks")

class Dog(Animal):
    def speak(self):
        print("Dog barks")

my_dog = Dog()
my_dog.speak()  # Output: Dog barks
```



- Overriding is meaningful in the context of inheritance. The overridden method must be inherited from a superclass

ACCESS TO SUPERCLASS METHOD

```
class Animal:
    def speak(self):
        print("Animal speaks")
```

```
class Dog(Animal):
    def speak(self):
        super().speak()
        print("Dog barks")
```

```
my_dog = Dog()
my_dog.speak()
# Output:
# Animal speaks
# Dog barks
```

- Despite overriding, a subclass can still access the overridden method from the superclass using `super()`

EXERCISE

- Try the following exercises
- 02-00
> 09-overriding



Super

- Definition
- Usage in Method Overriding
- Method Resolution Order (MRO)
- Accessing Superclass Attributes

DEFINITION

- `super()` is a built-in Python function used to call methods and access attributes of a superclass from a subclass
- It is commonly used in the context of method overriding to invoke the overridden method in the superclass

USAGE IN METHOD OVERRIDING

```
class Animal:
    def speak(self):
        print("Animal speaks")
```

```
class Dog(Animal):
    def speak(self):
        super().speak()
        print("Dog barks")
```

```
my_dog = Dog()
my_dog.speak()
# Output:
# Animal speaks
# Dog barks
```

- In a subclass, `super()` is often used to call the overridden method in the superclass before or after adding new functionality

METHOD RESOLUTION ORDER (MRO)

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

class Dog(Animal):
    def __init__(self, age, breed):
        super().__init__(age) # Call the constructor of the superclass
        self.breed = breed

    def display_info(self):
        print(f"Dog of age {self.age} and breed {self.breed}")

my_dog = Dog(age=3, breed="Labrador")
my_dog.display_info()
# Output:
# Dog of age 3 and breed Labrador
```

- `super()` is essential in maintaining the Method Resolution Order (MRO), which defines the order in which base classes are searched when looking for a method

ACCESSING SUPERCLASS ATTRIBUTES

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

    @property
    def age(self):
        return self._age

    @age.setter
    def age(self, value):
        self._age = value
```

```
class Dog(Animal):
    def __init__(self, age, breed):
        super().__init__(age)
        self.breed = breed

    def get_age_in_human_years(self):
        # Accessing the 'age' attribute from the
        # superclass using super()
        human_years = super().age * 7
        return human_years

my_dog = Dog(age=3, breed="Labrador")
human_years = my_dog.get_age_in_human_years()
print(f"Dog's age in human years: {human_years}")
# Output:
# Dog's age in human years: 21
```

- Besides method invocation, `super()` can be used to access attributes of the superclass

EXERCISE

- Try the following exercises
- 02-00
> 10-super

