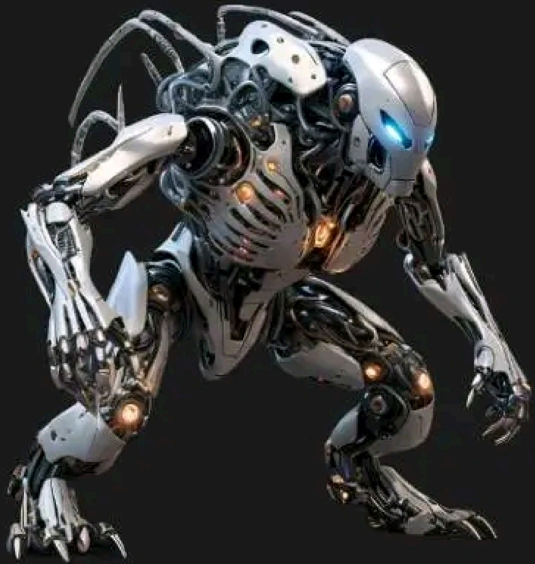
POLYMORPHISM IN

PYTHON



python™

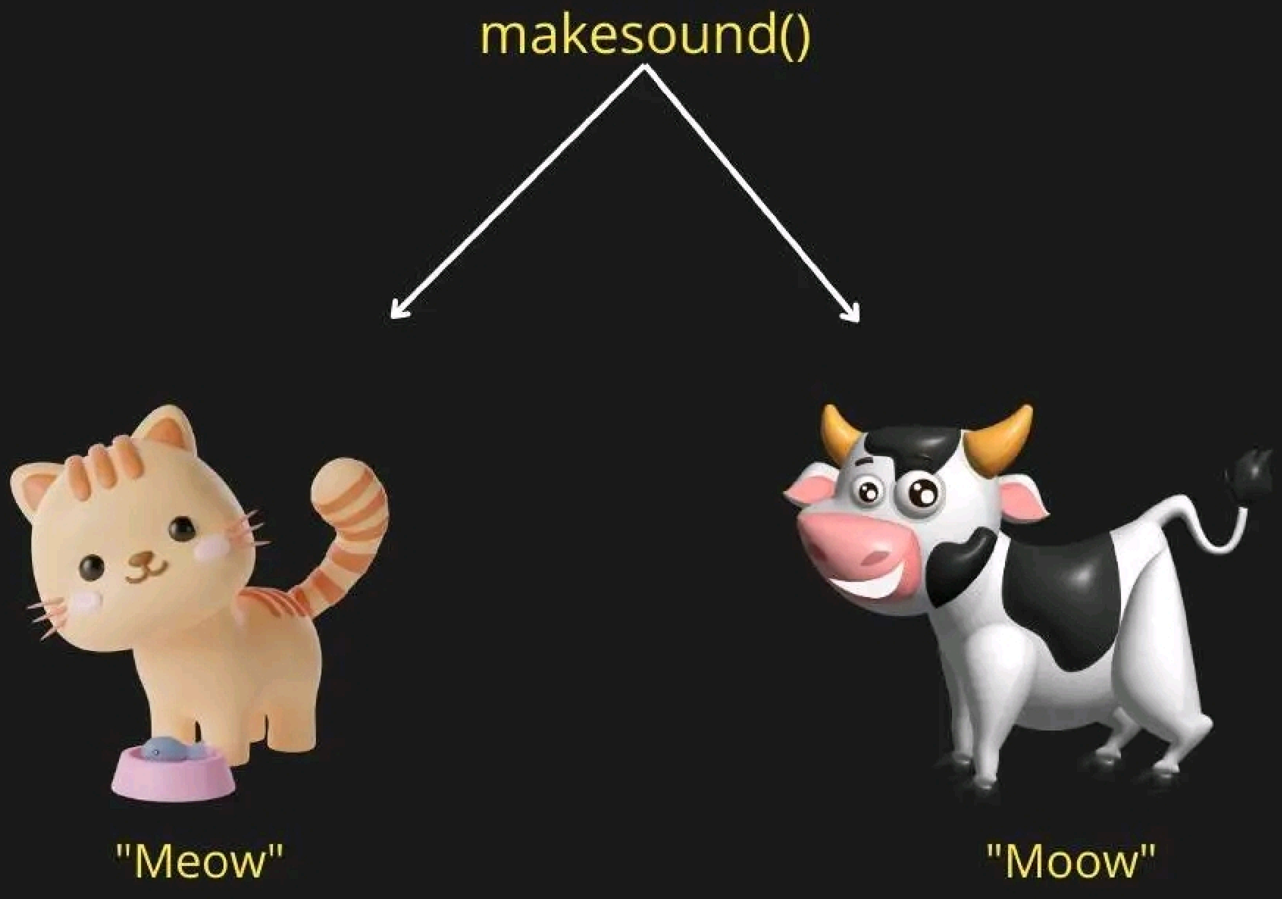
NEXT

SWIPE

Polymorphism 🔥

Polymorphism means **"many shapes"** in programming.

In Python, it allows objects of different classes to be treated as objects of a common superclass. The key idea is that the same operation can behave differently across different classes.



python™

NEXT

SWIPE

Built-in Polymorphic Functions

Python has several built-in functions that exhibit polymorphism, such as len(), max(), and min(). These functions can work with different types of data without modification.

Using Polymorphism in the len() Function

The len() function returns the length of an object and works with strings, lists, tuples, and more.

```
print(len("hello"))
# Output: 5 (number of characters in the string)

print(len([1, 2, 3, 4]))
# Output: 4 (number of items in the list)
```

This flexibility makes Python's built-in functions powerful and reusable across various data types.

python™

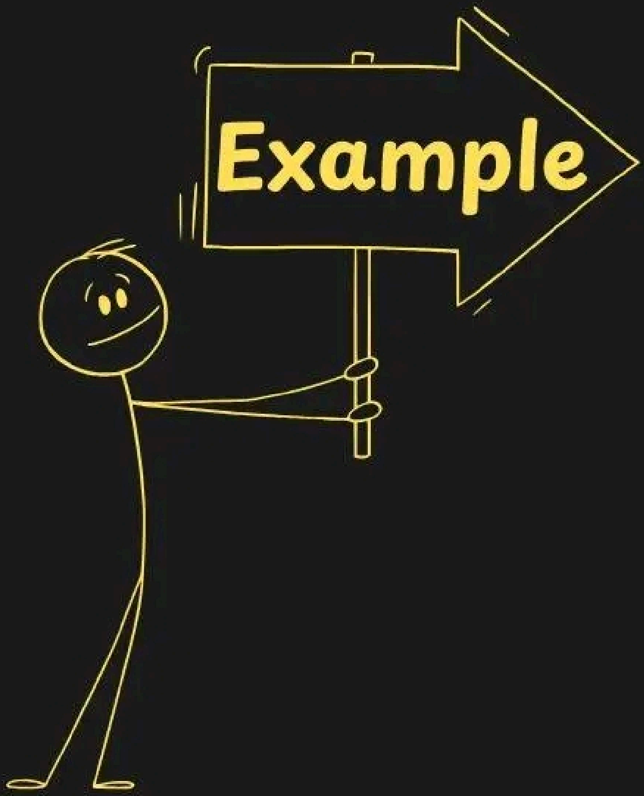
NEXT

Method Overriding

Method overriding is a way to achieve polymorphism in object-oriented programming.

It allows a subclass to provide a specific implementation of a method that is already defined in its superclass.

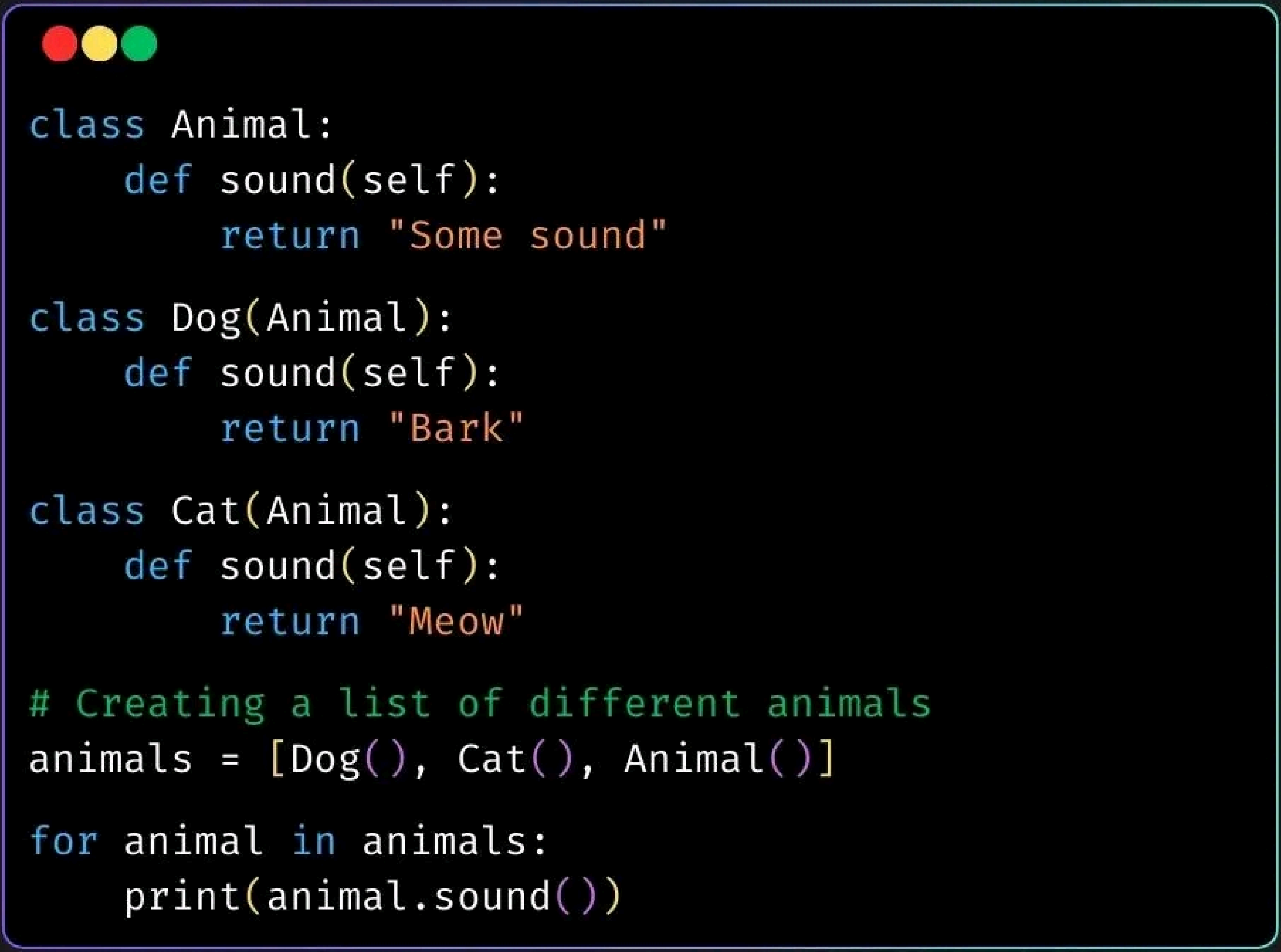
The overridden method in the subclass must have the same name and parameters as the method in the superclass.



SWIPE

Example

Let's create a simple class structure to demonstrate method overriding:



```python
class Animal:
    def sound(self):
        return "Some sound"

class Dog(Animal):
    def sound(self):
        return "Bark"

class Cat(Animal):
    def sound(self):
        return "Meow"

# Creating a list of different animals
animals = [Dog(), Cat(), Animal()]

for animal in animals:
    print(animal.sound())
```

Here, the sound() method is overridden in the Dog and Cat classes, providing their own implementations while maintaining the same method signature as the Animal class.

python™

NEXT

SWIPE

Output

When we run the previous example, we get the following output:

```
Bark
Meow
Some sound
```

Explanation:

- Dog().sound() returns "Bark" because the Dog class overrides the sound() method from the Animal class.
- Cat().sound() returns "Meow" for the same reason—the Cat class provides its own implementation of sound().
- Animal().sound() returns "Some sound" because it uses the base class implementation since no overriding occurs.

python™

NEXT

SWIPE

Polymorphism in Class Methods

Polymorphism can also be applied to class methods. This allows different classes to define a method in a way that is specific to their own class while sharing a common interface.

Example with Class Method

- Let's design a different class in the same way by adding the same methods in two or more classes.
- Next, create an object of each class.
- Next, add all object of each class.
- In the end, iterate the tuple using a for loop and call methods of a object without checking its class.

python™

NEXT

SWIPE

Example

```python
# Define classes with the same method name
class Chef:
    def activity(self):
        return "I cook delicious meals."

class Artist:
    def activity(self):
        return "I paint beautiful pictures."

class Engineer:
    def activity(self):
        return "I design innovative solutions."

# Create objects of each class
chef = Chef()
artist = Artist()
engineer = Engineer()

# Add objects to a tuple
professionals = (chef, artist, engineer)

# Iterate over the tuple and call the activity method
for professional in professionals:
    print(professional.activity())
```

NEXT

Output

```
I cook delicious meals.
I paint beautiful pictures.
I design innovative solutions.
```

Explanation:

- Each class (Chef, Artist, Engineer) defines a method activity().
- Objects are created for each class and stored in a tuple named professionals.
- The for loop iterates over this tuple and calls the activity() method on each object.
- Since Python is dynamically typed, the method call doesn't require checking the object's class.
- Polymorphism allows each object to respond appropriately with its own implementation of the method.