Polymorphism Using Duck Typing

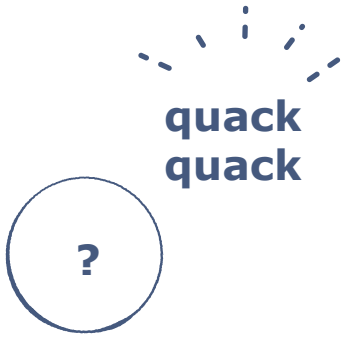
In this lesson, we will be implementing polymorphism using duck typing.

We'll cover the following

Duck typing is one of the most useful concepts in Object-Oriented Programming in Python. Using duck typing, we can implement polymorphism without using inheritance. The name may sound a bit funny right now, but we will come to understand it at the end of the lesson!

What is Duck Typing?

We say that if an object *quacks* like a duck, *swims* like a duck, *eats* like a duck or in short, *acts* like a duck, that object is a duck.



Dynamic Typing

Duck typing extends the concept of **dynamic typing** in Python.

> Dynamic typing means we can change the type of an object later in the code.

Due to the dynamic nature of Python, duck typing allows the user to use any object that provides the required behavior without the constraint that it has to be a subclass. See the code below for a better understanding of dynamic typing in Python:

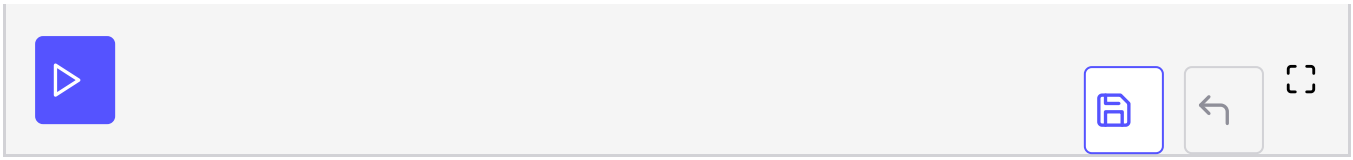
```
x = 5  # type of x is an integer
print(type(x))

x = "Educative"  # type x is now string
print(type(x))
```

Implementing Duck Typing

Let us see an example to implement this:

```
class Dog:
    def Speak(self):
        print("Woof woof")


class Cat:
    def Speak(self):
        print("Meow meow")


class AnimalSound:
    def Sound(self, animal):
        animal.Speak()


sound = AnimalSound()
dog = Dog()
cat = Cat()

sound.Sound(dog)
```

```
sound.Sound(cat)
```



Explanation

- In **line 13**, the type of `animal` is not defined in the definition of the method `Sound`.
- Type of `animal` is determined when the method is called, so it does not matter which object type you are passing as a parameter in the `Sound()` method, what matters is that the `Speak()` method **should be defined in all the classes** whose objects are passed in the `Sound()` method.
- We can use any property or method of `animal` in the `AnimalSound` class as long as it is declared in that class.

Conclusion

Now coming back to why it is called Duck typing. So, if a bird speaks like a duck, swims like a duck, and eats like a duck; that bird is a duck.

Similarly, in the above example, `animal` object does not matter in the definition of the `Sound` method as long as it has the associated behavior, `speak()`, defined in the object's class definition. In layman terms, since both the animals, *dog* and *cats*, can speak like *animals*, they both are *animals*. This is how we have achieved polymorphism without inheritance.

In the next lesson, we will discuss how to use abstract base classes in Python.