Polymorphism All Versions

Examples

Duck Typing

Polymorphism without inheritance. As long as the classes contain the same functions it doesn't care that one isn't the other.

```
class Duck:
    def quack(self):
        print("Quaaaaaack!")
    def feathers(self):
        print("The duck has white and gray feathers.")

class Person:
    def quack(self):
        print("The person imitates a duck.")
    def feathers(self):
        print("The person takes a feather from the ground and shows it.")
    def name(self):
        print("John Smith")

def in_the_forest(obj):
    obj.quack()
    obj.feathers()

donald = Duck()
john = Person()
in_the_forest(donald)
in_the_forest(john)
```

Basic Polymorphism

Polymorphism is the ability to perform an action on an object regardless of its type. This is generally implemented by creating a base class and having two or more subclasses that all implement methods with the same signature. Any other function or method that manipulates these objects can call the same methods regardless of which type of object it is operating on, without needing to do a type check first. In object-oriented terminology when class X extend class Y , then Y is called super class or base class and X is called subclass or derived class.

```
class Shape:
    """
    This is a parent class that is intended to be inherited by other classes
    """

    def calculate_area(self):
        """
        This method is intended to be overridden in subclasses.
        If a subclass doesn't implement it but it is called, NotImplemented will be raised.

        """
        raise NotImplemented

class Square(Shape):
    """
    This is a subclass of the Shape class, and represents a square
    """
    side_length = 2     # in this example, the sides are 2 units long

    def calculate_area(self):
        """
        This method overrides Shape.calculate_area(). When an object of type
        Square has its calculate_area() method called, this is the method that
        will be called, rather than the parent class' version.

        It performs the calculation necessary for this shape, a square, and
        returns the result.
        """
        return self.side_length * 2

class Triangle(Shape):
    """
    This is also a subclass of the Shape class, and it represents a triangle
    """
    base_length = 4
    height = 3
```

We should see this output:

```
None
4
6.0
```

What happens without polymorphism?
Without polymorphism, a type check may be required before performing an action on an object to determine the correct method to call. The following **counter example** performs the same task as the previous code, but without the use of polymorphism, the `get_area()` function has to do more work.

```
class Square:

    side_length = 2

    def calculate_square_area(self):
        return self.side_length ** 2

class Triangle:

    base_length = 4
    height = 3

    def calculate_triangle_area(self):
        return (0.5 * self.base_length) * self.height

def get_area(input_obj):

    # Notice the type checks that are now necessary here. These type checks
    # could get very complicated for a more complex example, resulting in
    # duplicate and difficult to maintain code.

    if type(input_obj).__name__ == "Square":
        area = input_obj.calculate_square_area()

    elif type(input_obj).__name__ == "Triangle":
        area = input_obj.calculate_triangle_area()

    print(area)

# Create one object of each class
square_obj = Square()
triangle_obj = Triangle()

# Now pass each object, one at a time, to the get_area() function and see the
# result.
get_area(square_obj)
get_area(triangle_obj)
```

We should see this output:

```
4
6.0
```

Important Note
Note that the classes used in the counter example are "new style" classes and implicitly inherit from the object class if Python 3 is being used. Polymorphism will work in both Python 2.x and 3.x, but the polymorphism counterexample code will raise an exception if run in a Python 2.x interpreter because type(input_obj). **name** will return "instance" instead of the class name if they do not explicitly inherit from object, resulting in area never being assigned to.

Syntax

Parameters

Remarks