Polymorphism in Python

Что такое Полиморфизм?

Полиморфизм происходит от греческих слов Poly (много) и morphism (формы). Это означает, что одно и то же имя функции может использоваться для разных типов. Это делает программирование более интуитивным и простым.
В Python у нас есть разные способы определения полиморфизма. Итак, давайте продолжим и посмотрим, как полиморфизм работает в Python.

Полиморфизм в Python:
Дочерний класс наследует все методы родительского класса. Однако в некоторых ситуациях метод, унаследованный от родительского класса, не совсем подходит для дочернего класса. В таких случаях вам придется повторно реализовать метод в дочернем классе.

Существуют разные методы использования полиморфизма в Python. Вы можете использовать различные функции, методы класса или объекты для определения полиморфизма.

Полиморфизм с функциями и объектами

Вы можете создать функцию, которая может принимать любой объект , допуская полиморфизм.
Давайте возьмем пример и создадим функцию с именем « func() », которая будет принимать объект, который мы назовем «obj». Теперь давайте дадим функции что-то делать, используя объект ' obj ', который мы ей передали. В этом случае давайте вызовем методы type() и color() , каждый из которых определен в двух классах «Помидор» и «Яблоко».
Теперь вам нужно создать экземпляры классов «Помидор» и «Яблоко», если у нас их еще нет:

```
class Tomato():
     def type(self):
       print("Vegetable")
     def color(self):
       print("Red")
class Apple():
     def type(self):
       print("Fruit")
     def color(self):
       print("Red")

def func(obj):
       obj.type()
       obj.color()

obj_tomato = Tomato()
```

```
obj_apple = Apple()
func(obj_tomato)
func(obj_apple)

Output:
Vegetable
Red
Fruit
Red
```

Полиморфизм с методами класса

Python одинаково использует два разных типа классов.

Здесь вам нужно создать цикл for, который выполняет итерацию по кортежу объектов. Затем вы должны вызывать методы, не заботясь о том, к какому типу класса относится каждый объект. Мы предполагаем, что эти методы действительно существуют в каждом классе.

Вот пример , показывающий полиморфизм с классом:

```
class India():
     def capital(self):
       print("New Delhi")
     def language(self):
       print("Hindi and English")
class USA():
     def capital(self):
       print("Washington, D.C.")
     def language(self):
       print("English")
obj_ind = India()
obj_usa = USA()
for country in (obj_ind, obj_usa):
country.capital()
country.language()

Output:
New Delhi
Hindi and English
Washington, D.C.
English
```

Полиморфизм с наследованием

Полиморфизм в python определяет методы в дочернем классе, которые имеют то же имя, что и методы в родительском классе. При наследовании дочерний класс наследует методы

родительского класса. Кроме того, в дочернем классе можно изменить метод, унаследованный от родительского класса.

В основном это используется в тех случаях, когда метод, унаследованный от родительского класса, не подходит для дочернего класса. Этот процесс повторной реализации метода в дочернем классе известен как переопределение метода . Вот пример, демонстрирующий полиморфизм с наследованием:

```
class Bird:
     def intro(self):
       print("There are different types of birds")
     def flight(self):
       print("Most of the birds can fly but some cannot")
class parrot(Bird):
     def flight(self):
       print("Parrots can fly")
class penguin(Bird):
     def flight(self):
       print("Penguins do not fly")
obj_bird = Bird()
obj_parr = parrot()
obj_peng = penguin()
obj_bird.intro()
obj_bird.flight()
obj_parr.intro()
obj_parr.flight()
obj_peng.intro()
obj_peng.flight()

Output:
There are different types of birds
Most of the birds can fly but some cannot
There are different types of bird
Parrots can fly
There are many types of birds
Penguins do not fly
```