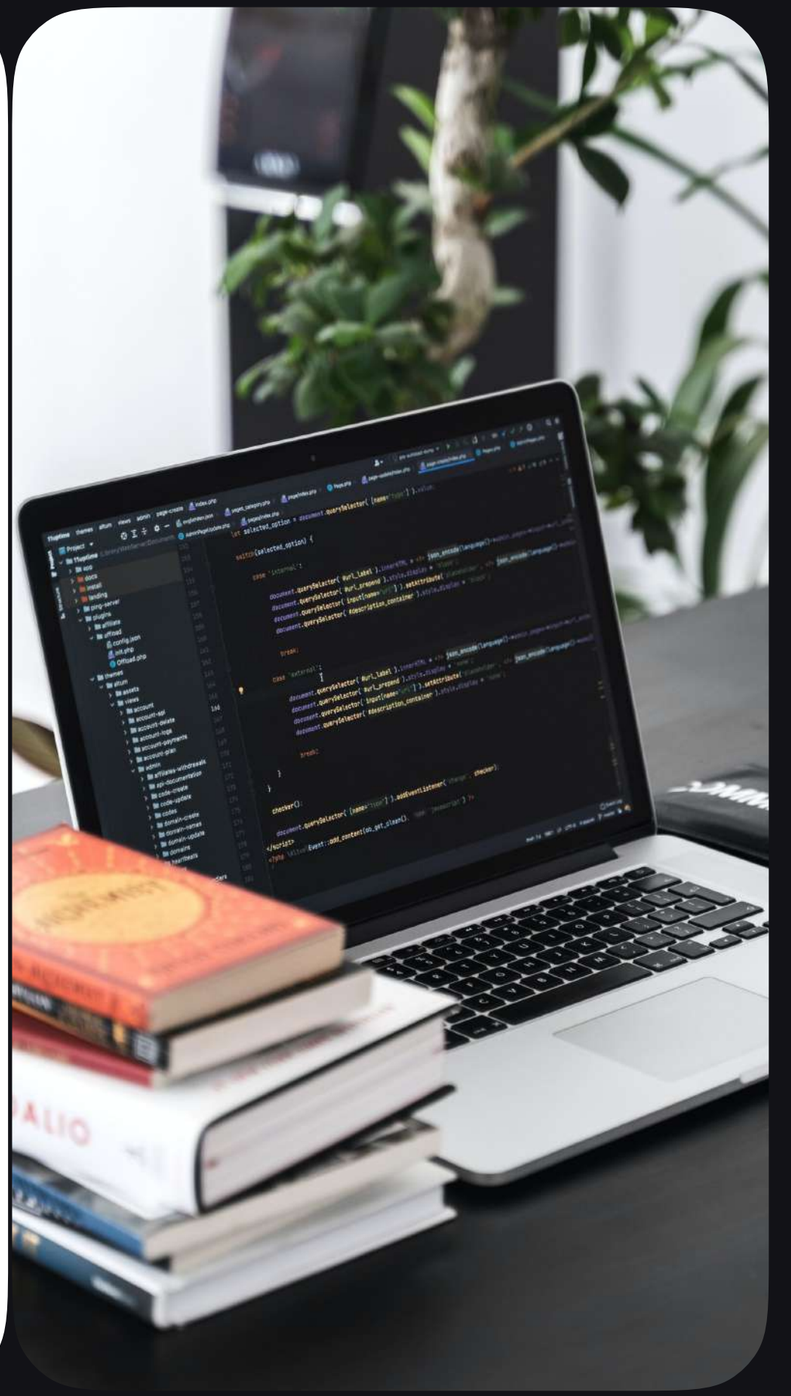
Модуль 2 Занятие 7

Полиморфизм



Полиморфизм

Полиморфные функции

Реализация в Python

Виды полиморфизма

Практика

Полиморфизм

Полиморфизм — это принцип объектно ориентированного программирования, согласно которому, существует возможность выполнять действия над разными объектами единым образом.

В контексте программирования это означает, что одна и та же функция, может вести себя по-разному в зависимости от типа объекта.

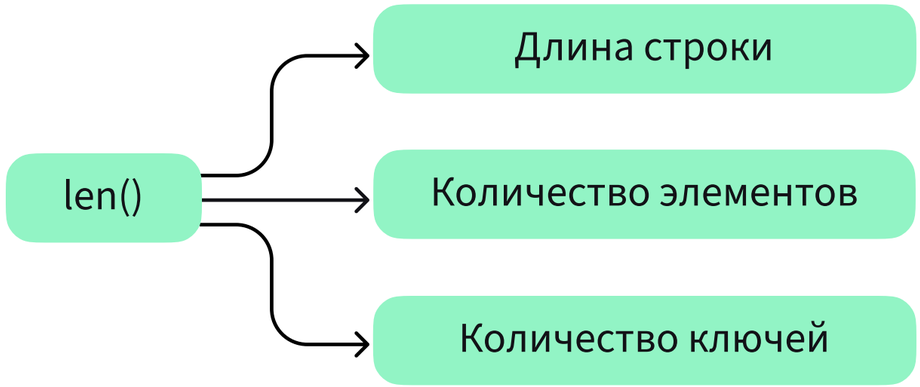
Полиморфизм стандартных функций

В языке Python существуют функции, способные работать с разными типами, например, одну и ту же функцию len() можно вызывать с аргументами разного типа:

```
print(len('abcde'))
print(len([1, 2, 3]))
print(len({'Hello': 'Python'}))
```

Вывод:

```
5
3
1
```



Полиморфизм операторов

Один и тот же оператор сложения работает с разными объектами:

```
print('Hello, ' + 'world!')
print(123 + 234)
print('123' + '234')
```

Вывод:

```
Hello, world!
357
123234
```

Такое поведение стандартных функций и операторов — это реализация полиморфизма в Python.

Как реализовать полиморфизм в Python

```
class Rectangle:
  def __init__(self, a, b):
    self.a = a
    self.b = b

    def get_rectangle_area(self):
      return self.a * self.b

class Circle:
  def __init__(self, r):
    self.r = r

  def get_circle_area(self):
    return 3.14 * self.r ** 2



r = Rectangle(5, 6)
c = Circle(4)
print(r.get_rectangle_area())
print(c.get_circle_area())
```

Создадим класс Rectangle и добавим метод get_rectangle_area() и класс Circle и метод get_circle_area().

Все работает и можно получить площади фигур. Есть ли проблема?

Пример

```
class Rectangle:
  def __init__(self, a, b):
      self.a = a
      self.b = b

  def get_rectangle_area(self):
      return self.a * self.b

class Circle:
  def __init__(self, r):
      self.r = r

  def get_circle_area(self):
      return 3.14 * self.r ** 2


shapes = [Rectangle(5, 6), Circle(4)]
for shape in shapes:
    if isinstance(shape, Rectangle):
        print(shape.get_rectangle_area())
    elif isinstance(shape, Circle):
        print(shape.get_circle_area())
```

Если объекты добавить в список, то, для того, чтобы понять какой метод нужно будет вызывать для конкретного объекта, придется добавить условия.

Все работает, но такая реализация требует дополнительных проверок.

Полиморфизм в Python

```
class Rectangle:
  def __init__(self, a, b):
    self.a = a
    self.b = b

  def get_area(self):
    return self.a * self.b

class Circle:
  def __init__(self, r):
    self.r = r

  def get_area(self):
    return 3.14 * self.r ** 2


shapes = [Rectangle(5, 6), Circle(4)]
for shape in shapes:
  print(shape.get_area())
```

Можно использовать одинаковое название метода во всех классах и тогда никаких дополнительных проверок не потребуется.

Метод get_area() имеет единое имя, но работает с разными объектами.

Утиная типизация

Утиная типизация — это концепция, согласно которой тип объекта не важен, а важны лишь свойства и методы, которыми этот объект обладает. Это значит, что при работе с объектом не проверяется его тип, вместо этого проверяются свойства и методы этого объекта.

Такой подход позволяет реализовать полиморфизм и работать с объектами единым образом. Достаточно, чтобы все эти объекты поддерживали необходимый набор методов.

Если это выглядит как утка, плавает как утка и крякает как утка, то это, вероятно, и есть утка.

Пример

```
class Rectangle:
  def __init__(self, a, b):
    self.a = a
    self.b = b

  def get_area(self):
    return self.a * self.b

class Circle:
  def __init__(self, r):
    self.r = r

  def get_area(self):
    return 3.14 * self.r ** 2

class Square(Rectangle):
  def __init__(self, a):
    super().__init__(a, a)


shapes = [Rectangle(5, 6), Circle(4), Square(3)]
for shape in shapes:
  print(shape.get_area())
```

Можно заметить, что такая реализация полиморфизма позволяет вообще не определять метод get_area() в классе Square. Метод родительского класса Rectangle может работать с экземпляром класса Square.

Виды полиморфизма

Условно можно выделить несколько видов полиморфизма:

- ad-hoc (специальный) полиморфизм
- параметрический полиморфизм (универсальный)
- полиморфизм подтипов (включений)

Пример ad-hoc полиморфизма в Pascal

```
program Example;

function Add( x, y : Integer ) : Integer;
begin
    Add := x + y
end;

function Add( s, t : String ) : String;
begin
    Add := Concat( s, t )
end;

begin
    Writeln(Add(1, 2));
    Writeln(Add('Hello, ', 'World!'));
end.
```

Такой полиморфизм может быть реализован за счёт перегрузки (переопределения) функций — описываются разные функции, в которых задаются разные типы аргументов. Язык программирования сам выбирает какую функцию в какой момент использовать в зависимости от типа.

Итоги

- Полиморфизм — это принцип ООП, согласно которому одна и та же функция может работать с объектами разных типов
- Поведение стандартных функций и операторов в Python — примеры полиморфизма в Python
- Чтобы реализовать полиморфизм в Python, можно использовать одинаковое название метода в разных классах
- Выделяют несколько видов полиморфизма: ad-hoc (специальный), параметрический (универсальный), полиморфизм подтипов (включений)