

Принципы ООП

План урока

- Инкапсуляция
- Наследование
- Полиморфизм

ИНКАПСУЛЯЦИЯ

это процесс сокрытия внутренних
деталей объекта и предоставления
доступа только через четко
определенные методы или свойства

Инкапсуляция

ooo | код

```
class Car:  
    def __init__(self, make, model):  
        self._make = make  
        self._model = model  
    def get_make(self):  
        return self._make  
    def get_model(self):  
        return self._model  
  
my_car = Car("Toyota", "Camry")  
print(my_car.get_make())  
print(my_car.get_model())
```

ooo | вывод

```
>>> Toyota  
>>> Camry
```

НАСЛЕДОВАНИЕ

это процесс создания нового класса
путем приобретения свойств
и поведения существующего класса

Наследование

```
ooo class Rectangle

def __init__(self, color, width, height):
    self.color = color
    self.width = width
    self.height = height
def print_color(self):
    print(f"Я фигура цвета {self.color}!")
def get_area(self):
    return self.width * self.height
def get_perimeter(self):
    return 2 * (self.width + self.height)
```

```
ooo class Shape

def __init__(self, color):
    self.color = color
def print_color(self):
    print(f"Я фигура цвета {self.color}!")
def get_area(self):
    pass
def get_perimeter(self):
    pass
```

Наследование

```
ooo class Triangle

def __init__(self, color, base, height):
    self.color = color
    self.base = base
    self.height = height
def print_color(self):
    print(f"Я фигура цвета {self.color}!")
def get_area(self):
    return 0.5 * self.base * self.height
def get_perimeter(self):
    return 3 * self.base
```

Наследование

```
ooo | class Shape  
  
def __init__(self, color):  
    self.color = color  
def print_color(self):  
    print(f"Я фигура цвета {self.color}!")  
def get_area(self):  
    pass  
def get_perimeter(self):  
    pass
```

ooo | код

```
# Создаем объекты прямоугольника и треугольника  
rect = Rectangle("red", 5, 4)  
tri = Triangle("blue", 3, 6)  
# Используем наследованные методы  
print(f"Rectangle area: {rect.get_area()}")  
print(f"Rectangle perimeter: {rect.get_perimeter()}")  
print(f"Triangle area: {tri.get_area()}")  
print(f"Triangle perimeter: {tri.get_perimeter()}")  
rect.print_color() # Я фигура цвета red!  
tri.print_color() # Я фигура цвета blue!
```

ПОЛИМОРФИЗМ

позволяет обрабатывать объекты разных классов одним и тем же способом

Это означает, что одно имя метода может иметь разные реализации в разных классах

Полиморфизм способствует гибкости и переиспользованию кода

Полиморфизм

```
ooo class Rectangle

def __init__(self, color, width, height):
    self.color = color
    self.width = width
    self.height = height
def print_color(self):
    print(f"Я фигура цвета {self.color}!")
def get_area(self):
    return self.width * self.height
def get_perimeter(self):
    return 2 * (self.width + self.height)
```

```
ooo class Triangle

def __init__(self, color, base, height):
    self.color = color
    self.base = base
    self.height = height
def print_color(self):
    print(f"Я фигура цвета {self.color}!")
def get_area(self):
    return 0.5 * self.base * self.height
def get_perimeter(self):
    return 3 * self.base
```