Тема 1.7.2 Наследование

Наследование – возможность на основе одного класса создавать другие классы.

Класс, наследующий от другого, автоматически получает все атрибуты и методы первого класса.

Исходный класс называется *родителем*, новый класс – *потомком.*

Класс-потомок наследует атрибуты и методы класса-родителя, но при этом может определять собственные атрибуты и методы.

В класс-потомок можно добавить сколько угодно атрибутов и методов.

Метод __init__() класса-потомка

При определении класса-потомка можно вызывать метод __init__() класса-родителя.

При этом происходит инициализация атрибутов, определенных в методе __init__() класса-родителя, и они становятся доступны для класса-потомка.

```
class Car():
    ''' Простая модель авто'''
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0
```

```
    def get_decriptive_name(self):
        long_name = f"{self.year} {self.make} {self.model}"
        return long_name.title()

    def read_odometer(self):
        print(f"Эта машина имеет {self.odometer_reading} миль
пробега")

    def update_odometer(self, mileage):
        if mileage > = self.odometer_reading:
            self.odometer_reading = mileage
        else:
            print("изменить нельзя")

    def increment_odometer(self, miles):
        self.odometer_reading+ = miles

class ElectricCar(Car):

    def __init__(self, make, model, year):
        super().__init__(make, model, year)

my_tesla = ElectricCar('tesla', 'model s', 2019)
print(my_tesla.get_decriptive_name())
```

Добавление атрибутов и методов класса-потомка

Добавим в class ElectricCar(Car) атрибут self.battery_size и зададим ему значение 75.

Добавим в class ElectricCar(Car) метод describe_battery(self), который будет выводить:

print(f"Мощность аккумулятора этой машины {self.battery_size}").

В программу включим вызов метода my_tesla.describe_battery().

```
class Car():
    ''' Простая модель авто'''
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0

    def get_decriptive_name(self):
        long_name = f"{self.year} {self.make} {self.model}"
        return long_name.title()

    def read_odometer(self):
        print(f"Эта машина имеет {self.odometer_reading} миль пробега")

    def update_odometer(self, mileage):
```

```
        if mileage > = self.odometer_reading:
            self.odometer_reading = mileage
        else:
            print("изменить нельзя")

    def increment_odometer(self, miles):
        self.odometer_reading+ = miles

class ElectricCar(Car):

    def __init__(self, make, model, year):
        super().__init__(make, model, year)
        self.battery_size = 75

    def describe_battery(self):
        print(f"Мощность аккумулятора этой машины
{self.battery_size}")



my_tesla = ElectricCar('tesla', 'model s', 2019)
print(my_tesla.get_decriptive_name())
my_tesla.describe_battery()
```

Переопределение методов класса-родителя

Любой метод класса-родителя в классе-потомке можно переопределить.

Для этого в классе-потомке нужно создать метод с тем же именем, что у класса-родителя.

Действительным станет именно этот метод, а метод в классе-родителе будет игнорироваться.

Например, в классе-родителе Car есть метод tech(self) - пройти техосмотр.

В классе-потомке этот метод можно переопределить.

```
class ElectricCar(Car):


    ……..


    def tech(self):
        print("Этому авто техосмотр не требуется")


…….
my_tesla = ElectricCar('tesla', 'model s', 2019)
print(my_tesla.get_decriptive_name())
my_tesla.describe_battery()
my_tesla.tech()
```