MAXIMUM EDUCATION

DS
Программирование
Python

Конструкторы, атрибут self

Введение

С помощью какого ключевого слова создаются классы в Python?

С помощью какого ключевого слова создаются классы в Python?

С помощью ключевого слова class

С помощью какого ключевого слова создаются классы в Python?

С помощью ключевого слова class

Какие два вида информации описываются внутри класса?

Введение

С помощью какого ключевого слова создаются классы в Python?

С помощью ключевого слова class

Какие два вида информации описываются внутри класса?

Свойства и методы

Введение

С помощью какого ключевого слова создаются классы в Python?

С помощью ключевого слова class

Какие два вида информации описываются внутри класса?

Свойства и методы

Что содержится в свойствах класса?

Введение

С помощью какого ключевого слова создаются классы в Python?

С помощью ключевого слова class

Какие два вида информации описываются внутри класса?

Свойства и методы

Что содержится в свойствах класса?

В свойствах содержится информация о данных класса

Введение

С помощью какого ключевого слова создаются классы в Python?

С помощью ключевого слова class

Какие два вида информации описываются внутри класса?

Свойства и методы

Что содержится в свойствах класса?

В свойствах содержится информация о данных класса

Что из себя представляют методы класса?

Введение

С помощью какого ключевого слова создаются классы в Python?

С помощью ключевого слова class

Какие два вида информации описываются внутри класса?

Свойства и методы

Что содержится в свойствах класса?

В свойствах содержится информация о данных класса

Что из себя представляют методы класса?

Методы – это функции для работы со свойствами класса

Введение

- ***С помощью какого ключевого слова создаются классы в Python?***

  С помощью ключевого слова class

- ***Какие два вида информации описываются внутри класса?***

  Свойства и методы

- ***Что содержится в свойствах класса?***

  В свойствах содержится информация о данных класса

- ***Что из себя представляют методы класса?***

  Методы – это функции для работы со свойствами класса

- ***С помощью какого символа можно обращаться к свойствам и методам класса?***

Введение

С помощью какого ключевого слова создаются классы в Python?

С помощью ключевого слова class

Какие два вида информации описываются внутри класса?

Свойства и методы

Что содержится в свойствах класса?

В свойствах содержится информация о данных класса

Что из себя представляют методы класса?

Методы – это функции для работы со свойствами класса

С помощью какого символа можно обращаться к свойствам и методам класса?

С помощью точки

Методы класса, параметр self

Методы класса, параметр self

```
class Car:      # создаем класс Car
    self.mark = 'Toyota'     # марка
    self.model = 'Corolla'   # модель
    self.color = 'blue'      # цвет
    self.speed = 0           # скорость

# создаем два экземпляра класса Car
my_car_1 = Car()
my_car_2 = Car()
```

Методы класса, параметр self

```
class Car:     # создаем класс Car
    self.mark = 'Toyota'     # марка
    self.model = 'Corolla'   # модель
    self.color = 'blue'      # цвет
    self.speed = 0           # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine():
        print('Работает метод set_power_engine' )

# создаем два экземпляра класса Car
my_car_1 = Car()
my_car_2 = Car()
```

Методы класса, параметр self

```
class Car:     # создаем класс Car
    self.mark = 'Toyota'     # марка
    self.model = 'Corolla'   # модель
    self.color = 'blue'      # цвет
    self.speed = 0           # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine():
        print('Работает метод set_power_engine')

# создаем два экземпляра класса Car
my_car_1 = Car()
my_car_2 = Car()
Car.set_power_engine()
```

Методы класса, параметр self

```
class Car:    # создаем класс Car
    self.mark = 'Toyota'    # марка
    self.model = 'Corolla'  # модель
    self.color = 'blue'     # цвет
    self.speed = 0          # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine():
        print('Работает метод set_power_engine')

# создаем два экземпляра класса Car
my_car_1 =
my_car_2 =
```

```
>>> Работает метод set_power_engine
```

Методы класса, параметр self

```
class Car:     # создаем класс Car
    self.mark = 'Toyota'     # марка
    self.model = 'Corolla'   # модель
    self.color = 'blue'      # цвет
    self.speed = 0           # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine():
        print('Работает метод set_power_engine')

# создаем два экземпляра класса Car
my_car_1 = Car()
my_car_2 = Car()
my_car_1.set_power_engine()
```

Методы класса, параметр self

```
class Car:    # создаем класс Car
    self.mark = 'Toyota'    # марка
    self.model = 'Corolla'  # модель
    self.color = 'blue'     # цвет
    self.speed = 0          # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine():
        print('Работает метод set_power_engine')

# создаем два экземпляра класса Car
my_car_1 =
my_car_2 =
my_car_1.se
```

```
>>> TypeError: Car.set_power_engine() takes 0
positional arguments but 1 was given
```

Методы класса, параметр self

self – параметр, передающийся первым при создании метода класса. Он содержит в себе ссылку на объект, который вызывает метод.

```
                    self
                     ↓
def set_power_engine():
```

Методы класса, параметр self

```
class Car:     # создаем класс Car
    self.mark = 'Toyota'     # марка
    self.model = 'Corolla'   # модель
    self.color = 'blue'      # цвет
    self.speed = 0           # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine():
        print('Работает метод set_power_engine' )

# создаем два экземпляра класса Car
my_car_1 = Car()
my_car_2 = Car()
Car.set_power_engine(my_car_1)
```

Методы класса, параметр self

```
class Car:     # создаем класс Car
    self.mark = 'Toyota'     # марка
    self.model = 'Corolla'   # модель
    self.color = 'blue'      # цвет
    self.speed = 0           # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine(self, power_engine):
        print('Работает метод set_power_engine' )

# создаем два экземпляра класса Car
my_car_1 = Car()
my_car_2 = Car()
Car.set_power_engine()
```

Методы класса, параметр self

```
class Car:    # создаем класс Car
    self.mark = 'Toyota'    # марка
    self.model = 'Corolla'  # модель
    self.color = 'blue'     # цвет
    self.speed = 0          # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine():
        print('Работает метод set_power_engine')

# создаем два экземпляра класса Car
my_car_1 =
my_car_2 =
```

```
>>> TypeError: Car.set_power_engine() missing
1 required positional argument: 'self'
```

Методы класса, параметр self

```
class Car:     # создаем класс Car
    self.mark = 'Toyota'     # марка
    self.model = 'Corolla'   # модель
    self.color = 'blue'      # цвет
    self.speed = 0           # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine(self, power_engine):
        print('Работает метод set_power_engine' )

# создаем два экземпляра класса Car
my_car_1 = Car()
my_car_2 = Car()
Car.set_power_engine(my_car_1)
```

Методы класса, параметр self

```
class Car:     # создаем класс Car
    self.mark = 'Toyota'     # марка
    self.model = 'Corolla'   # модель
    self.color = 'blue'      # цвет
    self.speed = 0           # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine(self, power_engine):
        print('Работает метод set_power_engine' )

# создаем два экземпляра класса Car
my_car_1 = Car()
my_car_2 = Car()
```

Методы класса, параметр self

```
class Car:     # создаем класс Car
    self.mark = 'Toyota'     # марка
    self.model = 'Corolla'   # модель
    self.color = 'blue'      # цвет
    self.speed = 0           # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine(self, power_engine):
        print('Работает метод set_power_engine' )

# создаем два экземпляра класса Car
my_car_1 = Car()
my_car_2 = Car()
my_car_1.set_power_engine()
```

Методы класса, параметр self

```
class Car:     # создаем класс Car
    self.mark = 'Toyota'     # марка
    self.model = 'Corolla'   # модель
    self.color = 'blue'      # цвет
    self.speed = 0           # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine(self, power_engine):
        self.power_engine = power_engine

# создаем два экземпляра класса Car
my_car_1 = Car()
my_car_2 = Car()
my_car_1.set_power_engine(200)
```

Методы класса, параметр self

```
class Car:     # создаем класс Car
    self.mark = 'Toyota'     # марка
    self.model = 'Corolla'   # модель
    self.color = 'blue'      # цвет
    self.speed = 0           # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine(self, power_engine):
        self.power_engine = power_engine

# создаем два экземпляра класса Car
my_car_1 = Car()
my_car_2 = Car()
my_car_1.set_power_engine(200)
```

Методы класса, параметр self

```
class Car:     # создаем класс Car
    self.mark = 'Toyota'     # марка
    self.model = 'Corolla'   # модель
    self.color = 'blue'      # цвет
    self.speed = 0           # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine(self, power_engine):
        self.power_engine = power_engine

# создаем два экземпляра класса Car
my_car_1 = Car()
my_car_2 = Car()
my_car_1.set_power_engine(200)
# выводим на экран значение свойства power_engine
print(f'Мощность двигателя: {my_car_1.power_engine}')
```

Методы класса, параметр self

```
class Car:    # создаем класс Car
    self.mark = 'Toyota'    # марка
    self.model = 'Corolla'  # модель
    self.color = 'blue'     # цвет
    self.speed = 0          # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine():
        print('Работает метод set_power_engine')

# создаем два экземпляра класса Car
my_car_1 =
my_car_2 =
```

```
>>> Мощность двигателя: 200
```

Методы класса, параметр self

```
class Car:    # создаем класс Car
    self.mark = 'Toyota'    # марка
    self.model = 'Corolla'  # модель
    self.color = 'blue'     # цвет
    self.speed = 0          # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine(self, power_engine):
        self.power_engine = power_engine

# создаем два экземпляра класса Car
my_car_1 = Car()
my_car_2 = Car()
my_car_1.set_power_engine(200)
# выводим на экран локальные атрибуты для объектов my_car_2
print(my_car_2.__dict__)
```

Методы класса, параметр self

```
class Car:    # создаем класс Car
    self.mark = 'Toyota'    # марка
    self.model = 'Corolla'  # модель
    self.color = 'blue'     # цвет
    self.speed = 0          # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine(self, power_engine):
        self.power_engine = power_engine

# создаем два экземпляра класса Car
my_car_1 =
my_car_2 =
my_car_1.se
# выводим на экран локальные атрибуты для объектов my_car_2
print(my_car_2.__dict__)
```

```
>>> {}
```

Методы класса, параметр self

```
class Car:    # создаем класс Car
    self.mark = 'Toyota'    # марка
    self.model = 'Corolla'  # модель
    self.color = 'blue'     # цвет
    self.speed = 0          # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine(self, power_engine):
        self.power_engine = power_engine

# создаем два экземпляра класса Car
my_car_1 = Car()
my_car_2 = Car()
my_car_1.set_power_engine(200)
# выводим на экран локальные атрибуты для объектов my_car_1
print(my_car_1.__dict__)
```

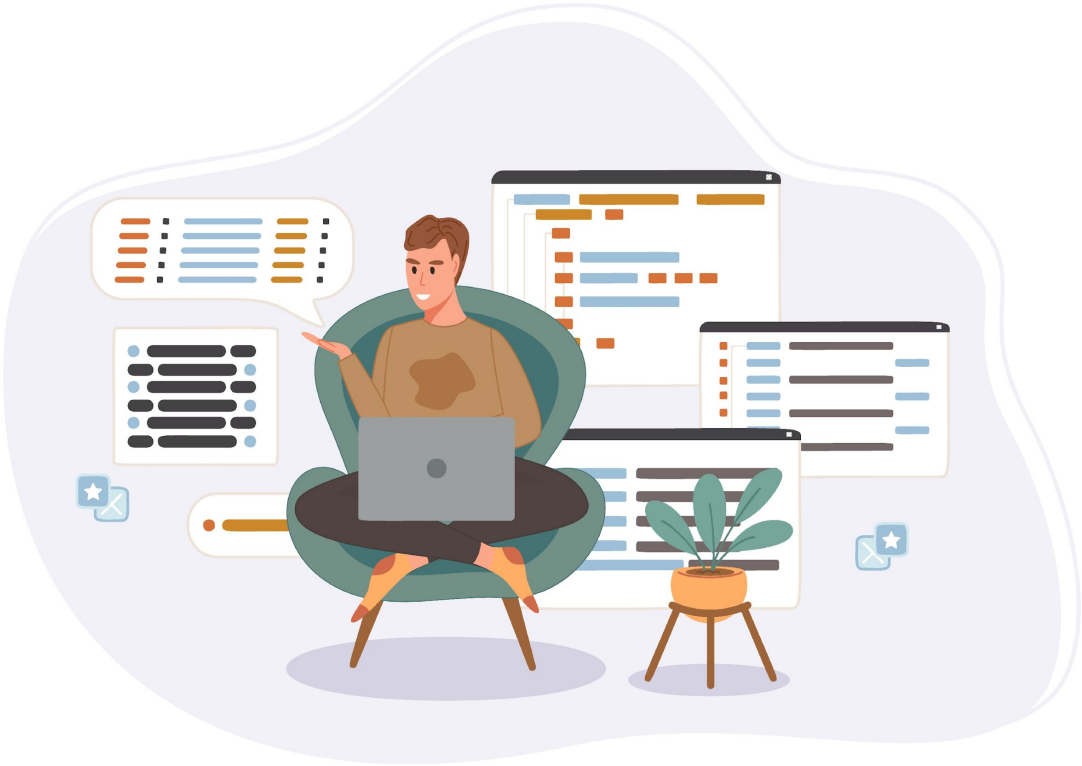
Методы класса, параметр self

```
class Car:    # создаем класс Car
    self.mark = 'Toyota'    # марка
    self.model = 'Corolla'  # модель
    self.color = 'blue'     # цвет
    self.speed = 0          # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine():
        print('Работает метод set_power_engine')

# создаем два экземпляра класса Car
my_car_1 =
my_car_2 =
```

```
>>> {'power_engine': 200}
```

Методы класса, параметр self

__dict__ - «магический» атрибут, возвращающий все локальные атрибуты объекта



Методы класса, параметр self

```
class Car:     # создаем класс Car
    self.mark = 'Toyota'     # марка
    self.model = 'Corolla'   # модель
    self.color = 'blue'      # цвет
    self.speed = 0           # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine(self, power_engine):
        self.power_engine = power_engine

# создаем два экземпляра класса Car
my_car_1 = Car()
my_car_2 = Car()
my_car_1.set_power_engine(200)
print(Car.__dict__)
```

Методы класса, параметр self

```
class Car:    # создаем класс Car
    self.mark = 'Toyota'    # марка
    self.model = 'Corolla'  # модель
    self.color = 'blue'     # цвет
    self.speed = 0          # скорость
    # метод для создания нового свойства - power_engine
```

```
>>> {'__module__': '__main__', 'mark':'Toyota', 'model':
'Corolla', 'color': 'blue', 'speed': 0, 'set_power_engine':
<function Car.set_power_engine at 0x0000014678288A40>,
'__dict__': <attribute '__dict__' of 'Car' objects>,
'__weakref__': <attribute '__weakref__' of 'Car' objects>,
'__doc__': None}
```

Методы класса, параметр self

```
class Car:     # создаем класс Car
    self.mark = 'Toyota'     # марка
    self.model = 'Corolla'   # модель
    self.color = 'blue'      # цвет
    self.speed = 0           # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine(self, power_engine):
        self.power_engine = power_engine
    # метод для вывода информации о свойствах класса
    def show_info(self):

# создаем два экземпляра класса Car
my_car_1 = Car()
my_car_2 = Car()
```

Методы класса, параметр self

```
class Car:     # создаем класс Car
    self.mark = 'Toyota'     # марка
    self.model = 'Corolla'   # модель
    self.color = 'blue'      # цвет
    self.speed = 0           # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine(self, power_engine):
        self.power_engine = power_engine
    # метод для вывода информации о свойствах класса
    def show_info(self):
        print(f'Марка: {self.mark}')
        print(f'Модель: {self.model}')
        print(f'Цвет: {self.color}')
        print(f'Текущая скорость: {self.speed}')
# создаем два экземпляра класса Car
my_car_1 = Car()
my_car_2 = Car()
```

Методы класса, параметр self

```
class Car:     # создаем класс Car
    self.mark = 'Toyota'     # марка
    self.model = 'Corolla'   # модель
    self.color = 'blue'      # цвет
    self.speed = 0           # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine(self, power_engine):
        self.power_engine = power_engine
    # метод для вывода информации о свойствах класса
    def show_info(self):
        print(f'Марка: {self.mark}')
        print(f'Модель: {self.model}')
        print(f'Цвет: {self.color}')
        print(f'Текущая скорость: {self.speed}')
# создаем два экземпляра класса Car
my_car_1 = Car()
my_car_2 = Car()

# вызываем метод show_info() для объекта my_car_2
my_car_2.show_info()
```

Методы класса, параметр self

```
class Car:    # создаем класс Car
    self.mark = 'Toyota'    # марка
    self.model = 'Corolla'  # модель
    self.color = 'blue'     # цвет
    self.speed = 0          # скорость
    # метод для создания нового свойства - power_engine
```

```
>>> Марка: Toyota
    Модель: Corolla
    Цвет: blue
    Текущая скорость: 0
```

```
# вызываем метод show_info() для объекта my_car_2
my_car_2.show_info()
```

Методы класса, параметр self

```
class Car:     # создаем класс Car
    self.mark = 'Toyota'     # марка
    self.model = 'Corolla'   # модель
    self.color = 'blue'      # цвет
    self.speed = 0           # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine(self, power_engine):
        self.power_engine = power_engine
    # метод для вывода информации о свойствах класса
    def show_info(self):
        print(f'Марка: {self.mark}')
        print(f'Модель: {self.model}')
        print(f'Цвет: {self.color}')
        print(f'Текущая скорость: {self.speed}')
    # метод для возвращения значений свойств класса
    def get_params(self):

# создаем два экземпляра класса Car
my_car_1 = Car()
my_car_2 = Car()
```

Методы класса, параметр self

```
class Car:     # создаем класс Car
    self.mark = 'Toyota'     # марка
    self.model = 'Corolla'   # модель
    self.color = 'blue'      # цвет
    self.speed = 0           # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine(self, power_engine):
        self.power_engine = power_engine
    # метод для вывода информации о свойствах класса
    def show_info(self):
        print(f'Марка: {self.mark}')
        print(f'Модель: {self.model}')
        print(f'Цвет: {self.color}')
        print(f'Текущая скорость: {self.speed}')
    # метод для возвращения значений свойств класса
    def get_params(self):
        return (self.mark, self.model, self.color, self.speed)

# создаем два экземпляра класса Car
my_car_1 = Car()
my_car_2 = Car()
```

Методы класса, параметр self

```
class Car:    # создаем класс Car
    self.mark = 'Toyota'    # марка
    self.model = 'Corolla'  # модель
    self.color = 'blue'     # цвет
    self.speed = 0          # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine(self, power_engine):
        self.power_engine = power_engine
    # метод для вывода информации о свойствах класса
    def show_info(self):
        print(f'Марка: {self.mark}')
        print(f'Модель: {self.model}')
        print(f'Цвет: {self.color}')
        print(f'Текущая скорость: {self.speed}')
    # метод для возвращения значений свойств класса
    def get_params(self):
        return (self.mark, self.model, self.color, self.speed)

# создаем два экземпляра класса Car
my_car_1 = Car()
my_car_2 = Car()

# получаем значения свойств объекта my_car_1 и сохраняем в кортеж params_car_1
params_car_1 = my_car_1.get_params()
print(params_car_1)
```

Методы класса, параметр self

```
class Car:    # создаем класс Car
    self.mark = 'Toyota'    # марка
    self.model = 'Corolla'  # модель
    self.color = 'blue'     # цвет
    self.speed = 0          # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine():
        print('Работает метод set_power_engine')

# создаем два экземпляра класса Car
my_car_1 =
my_car_2 =
```

```
>>> ('Toyota', 'Corolla', 'blue', 0)
```

Инициализатор и финализатор

```
class Car:     # создаем класс Car
    self.mark = 'Toyota'     # марка
    self.model = 'Corolla'   # модель
    self.color = 'blue'      # цвет
    self.speed = 0           # скорость
    # инициализатор класса
    def __init__(self, mark, model, color, speed):

    # метод для создания нового свойства - power_engine
    def set_power_engine(self, power_engine):
        self.power_engine = power_engine
    # метод для вывода информации о свойствах класса
    def show_info(self):
        print(f'Марка: {self.mark}')
        print(f'Модель: {self.model}')
        print(f'Цвет: {self.color}')
        print(f'Текущая скорость: {self.speed}')
    # метод для возвращения значений свойств класса
    def get_params(self):
        return (self.mark, self.model, self.color, self.speed)
```

Инициализатор и финализатор

```
class Car:     # создаем класс Car
    self.mark = 'Toyota'     # марка
    self.model = 'Corolla'   # модель
    self.color = 'blue'      # цвет
    self.speed = 0           # скорость
    # инициализатор класса
    def __init__(self, mark, model, color, speed):
        print('Работает метод __init__')
    # метод для создания нового свойства - power_engine
    def set_power_engine(self, power_engine):
        self.power_engine = power_engine
    # метод для вывода информации о свойствах класса
    def show_info(self):
        print(f'Марка: {self.mark}')
        print(f'Модель: {self.model}')
        print(f'Цвет: {self.color}')
        print(f'Текущая скорость: {self.speed}')
    # метод для возвращения значений свойств класса
    def get_params(self):
        return (self.mark, self.model, self.color, self.speed)
```

Инициализатор и финализатор

```
class Car:    # создаем класс Car
    self.mark = 'Toyota'    # марка
    self.model = 'Corolla'  # модель
    self.color = 'blue'     # цвет
    self.speed = 0          # скорость
    # инициализатор класса
    def __init__(self, mark, model, color, speed):
        print('Работает метод __init__')
    # метод для создания нового свойства - power_engine
    def set_power_engine(self, power_engine):
        self.power_engine = power_engine
    # метод для вывода информации о свойствах класса
    def show_info(self):
        print(f'Марка: {self.mark}')
        print(f'Модель: {self.model}')
        print(f'Цвет: {self.color}')
        print(f'Текущая скорость: {self.speed}')
    # метод для возвращения значений свойств класса
    def get_params(self):
        return (self.mark, self.model, self.color, self.speed)
# создаем экземпляр класса Car с заданными параметрами
my_car_1 = Car()
```

Инициализатор и финализатор

```
class Car:    # создаем класс Car
    self.mark = 'Toyota'    # марка
    self.model = 'Corolla'  # модель
    self.color = 'blue'     # цвет
    self.speed = 0          # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine():
        print('Работает метод set_power_engine')

# создаем два экземпляра класса Car
my_car_1 =
my_car_2 =
```

```
>>> Работает метод __init__
```

Инициализатор и финализатор

```
class Car:     # создаем класс Car
    # инициализатор класса
    def __init__(self, mark, model, color, speed):
        print('Работает метод __init__')
        self.mark = 'Toyota'    # марка
        self.model = 'Corolla'  # модель
        self.color = 'blue'     # цвет
        self.speed = 0          # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine(self, power_engine):
        self.power_engine = power_engine
    # метод для вывода информации о свойствах класса
    def show_info(self):
        print(f'Марка: {self.mark}')
        print(f'Модель: {self.model}')
        print(f'Цвет: {self.color}')
        print(f'Текущая скорость: {self.speed}')
    # метод для возвращения значений свойств класса
    def get_params(self):
        return (self.mark, self.model, self.color, self.speed)
# создаем экземпляр класса Car с заданными параметрами
my_car_1 = Car()
```

Инициализатор и финализатор

```
class Car:     # создаем класс Car
    # инициализатор класса
    def __init__(self, mark, model, color, speed):
        print('Работает метод __init__')
        self.mark = 'Toyota'    # марка
        self.model = 'Corolla'  # модель
        self.color = 'blue'     # цвет
        self.speed = 0          # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine(self, power_engine):
        self.power_engine = power_engine
    # метод для вывода информации о свойствах класса
    def show_info(self):
        print(f'Марка: {self.mark}')
        print(f'Модель: {self.model}')
        print(f'Цвет: {self.color}')
        print(f'Текущая скорость: {self.speed}')
    # метод для возвращения значений свойств класса
    def get_params(self):
        return (self.mark, self.model, self.color, self.speed)
# создаем экземпляр класса Car с заданными параметрами
my_car_1 = Car()
print(my_car_1.__dict__)
```

Инициализатор и финализатор

```
class Car:    # создаем класс Car
    self.mark = 'Toyota'    # марка
    self.model = 'Corolla'  # модель
    self.color = 'blue'     # цвет
    self.speed = 0          # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine():
        print('Работает метод set_power_engine')

# создаем два экземпляра класса Car
my_car_1 =
my_car_2 =
```

```
>>> {'mark': 'Toyota', 'model': 'Corolla',
'color': 'blue', 'speed': 0}
```

Инициализатор и финализатор

```
class Car:     # создаем класс Car
    # инициализатор класса
    def __init__(self, mark, model, color, speed):
        print('Работает метод __init__')
        self.mark = 'Toyota'    # марка
        self.model = 'Corolla'  # модель
        self.color = 'blue'     # цвет
        self.speed = 0          # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine(self, power_engine):
        self.power_engine = power_engine
    # метод для вывода информации о свойствах класса
    def show_info(self):
        print(f'Марка: {self.mark}')
        print(f'Модель: {self.model}')
        print(f'Цвет: {self.color}')
        print(f'Текущая скорость: {self.speed}')
    # метод для возвращения значений свойств класса
    def get_params(self):
        return (self.mark, self.model, self.color, self.speed)
# создаем экземпляр класса Car с заданными параметрами
my_car_1 = Car()
```

Инициализатор и финализатор

```
class Car:     # создаем класс Car
    # инициализатор класса
    def __init__(self, mark, model, color, speed):
        self.mark = mark     # марка
        self.model = model   # модель
        self.color = color   # цвет
        self.speed = speed   # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine(self, power_engine):
        self.power_engine = power_engine
    # метод для вывода информации о свойствах класса
    def show_info(self):
        print(f'Марка: {self.mark}')
        print(f'Модель: {self.model}')
        print(f'Цвет: {self.color}')
        print(f'Текущая скорость: {self.speed}')
    # метод для возвращения значений свойств класса
    def get_params(self):
        return (self.mark, self.model, self.color, self.speed)
# создаем экземпляр класса Car с заданными параметрами
my_car_1 = Car()
```

Инициализатор и финализатор

```
class Car:     # создаем класс Car
    # инициализатор класса
    def __init__(self, mark, model, color, speed):
        self.mark = mark     # марка
        self.model = model   # модель
        self.color = color   # цвет
        self.speed = speed   # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine(self, power_engine):
        self.power_engine = power_engine
    # метод для вывода информации о свойствах класса
    def show_info(self):
        print(f'Марка: {self.mark}')
        print(f'Модель: {self.model}')
        print(f'Цвет: {self.color}')
        print(f'Текущая скорость: {self.speed}')
    # метод для возвращения значений свойств класса
    def get_params(self):
        return (self.mark, self.model, self.color, self.speed)
# создаем экземпляр класса Car с заданными параметрами
my_car_1 = Car('Nissan', 'Juke', 'red', 0)
```

Инициализатор и финализатор

```
class Car:     # создаем класс Car
    # инициализатор класса
    def __init__(self, mark, model, color, speed):
        self.mark = mark     # марка
        self.model = model   # модель
        self.color = color   # цвет
        self.speed = speed   # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine(self, power_engine):
        self.power_engine = power_engine
    # метод для вывода информации о свойствах класса
    def show_info(self):
        print(f'Марка: {self.mark}')
        print(f'Модель: {self.model}')
        print(f'Цвет: {self.color}')
        print(f'Текущая скорость: {self.speed}')
    # метод для возвращения значений свойств класса
    def get_params(self):
        return (self.mark, self.model, self.color, self.speed)
# создаем экземпляр класса Car с заданными параметрами
my_car_1 = Car('Nissan', 'Juke', 'red', 0)
# выводим на экран значения всех свойств объекта my_car_1
print(my_car_1.get_params())
```

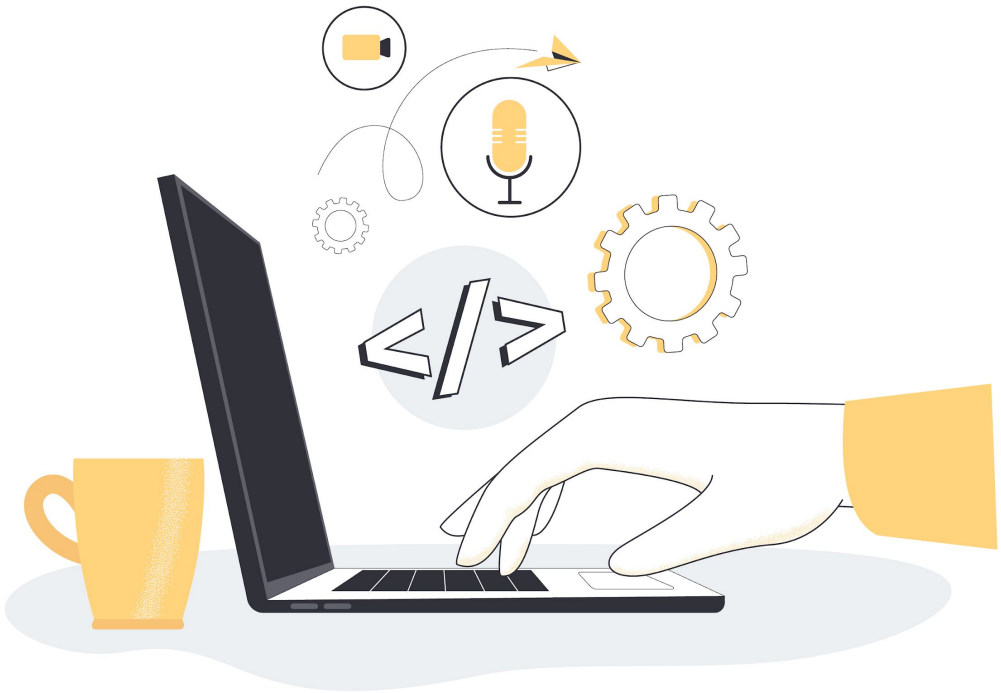
Инициализатор и финализатор

```
class Car:    # создаем класс Car
    # инициализатор класса
    def __init__(self, mark, model, color, speed):
        self.mark = mark      # марка
        self.model = model    # модель
        self.color = color    # цвет
        self.speed = speed    # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine(self, power_engine):
        self.power_engine = power_engine
    # метод д
    def sho
        pr
        print(f'Модель: {self.model}')
        print(f'Цвет: {self.color}')
        print(f'Текущая скорость: {self.speed}')
    # метод для возвращения значений свойств класса
    def get_params(self):
        return (self.mark, self.model, self.color, self.speed)
# создаем экземпляр класса Car с заданными параметрами
my_car_1 = Car('Nissan', 'Juke', 'red', 0)
# выводим на экран значения всех свойств объекта my_car_1
print(my_car_1.get_params())
```

```
>>> ('Nissan', 'Juke', 'red', 0)
```

Инициализатор и финализатор

__init__() - «магический» метод, который срабатывает сразу после успешного создания объекта класса. Позволяет задавать начальные параметры для объекта класса.



Инициализатор и финализатор

```
class Car:     # создаем класс Car
    # инициализатор класса
    def __init__(self, mark='', model='', color='', speed=0):
        self.mark = mark     # марка
        self.model = model   # модель
        self.color = color   # цвет
        self.speed = speed   # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine(self, power_engine):
        self.power_engine = power_engine
    # метод для вывода информации о свойствах класса
    def show_info(self):
        print(f'Марка: {self.mark}')
        print(f'Модель: {self.model}')
        print(f'Цвет: {self.color}')
        print(f'Текущая скорость: {self.speed}')
    # метод для возвращения значений свойств класса
    def get_params(self):
        return (self.mark, self.model, self.color, self.speed)
# создаем экземпляр класса Car с заданными параметрами
my_car_1 = Car('Nissan', 'Juke', 'red', 0)
# выводим на экран значения всех свойств объекта my_car_1
print(my_car_1.get_params())
```

Инициализатор и финализатор

```
class Car:     # создаем класс Car
    # инициализатор класса
    def __init__(self, mark='', model='', color='', speed=0):
        self.mark = mark     # марка
        self.model = model   # модель
        self.color = color   # цвет
        self.speed = speed   # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine(self, power_engine):
        self.power_engine = power_engine
    # метод для вывода информации о свойствах класса
    def show_info(self):
        print(f'Марка: {self.mark}')
        print(f'Модель: {self.model}')
        print(f'Цвет: {self.color}')
        print(f'Текущая скорость: {self.speed}')
    # метод для возвращения значений свойств класса
    def get_params(self):
        return (self.mark, self.model, self.color, self.speed)
# создаем экземпляр класса Car с заданными параметрами
my_car_1 = Car('Nissan', 'Juke', 'red', 0)
my_car_2 = Car()  # создаем экземпляр класса Car
# выводим на экран значения всех свойств объекта my_car_2
print(my_car_2.get_params())
```

Инициализатор и финализатор

```
class Car:    # создаем класс Car
    # инициализатор класса
    def __init__(self, mark='', model='', color='', speed=0):
        self.mark = mark     # марка
        self.model = model   # модель
        self.color = color   # цвет
        self.speed = speed   # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine(self, power_engine):
        self.power_engine = power_engine
    # метод д
    def sho
        pr
        print(f'Модель: {self.model}')
        print(f'Цвет: {self.color}')
        print(f'Текущая скорость: {self.speed}')
    # метод для возвращения значений свойств класса
    def get_params(self):
        return (self.mark, self.model, self.color, self.speed)
# создаем экземпляр класса Car с заданными параметрами
my_car_1 = Car('Nissan', 'Juke', 'red', 0)
my_car_2 = Car()  # создаем экземпляр класса Car
# выводим на экран значения всех свойств объекта my_car_2
print(my_car_2.get_params())
```

```
>>> ('', '', '', 0)
```

Инициализатор и финализатор

```
class Car:     # создаем класс Car
    # инициализатор класса
    def __init__(self, mark='', model='', color='', speed=0):
        self.mark = mark     # марка
        self.model = model   # модель
        self.color = color   # цвет
        self.speed = speed   # скорость
    # метод для создания нового свойства - power_engine
    def set_power_engine(self, power_engine):
        self.power_engine = power_engine
    # метод для вывода информации о свойствах класса
    def show_info(self):
        print(f'Марка: {self.mark}')
        print(f'Модель: {self.model}')
        print(f'Цвет: {self.color}')
        print(f'Текущая скорость: {self.speed}')
    # метод для возвращения значений свойств класса
    def get_params(self):
        return (self.mark, self.model, self.color, self.speed)
# создаем экземпляр класса Car с заданными параметрами
my_car_1 = Car('Nissan', 'Juke', 'red', 0)
my_car_2 = Car()  # создаем экземпляр класса Car
```

Инициализатор и финализатор

```
class Car:     # создаем класс Car
    # инициализатор класса
    def __init__(self, mark='', model='', color='', speed=0):
        self.mark = mark     # марка
        self.model = model   # модель
        self.color = color   # цвет
        self.speed = speed   # скорость
    # финализатор класса
    def __del__(self):

    # метод для создания нового свойства - power_engine
    def set_power_engine(self, power_engine):
        self.power_engine = power_engine
    # метод для вывода информации о свойствах класса
    def show_info(self):
        print(f'Марка: {self.mark}')
        print(f'Модель: {self.model}')
        print(f'Цвет: {self.color}')
        print(f'Текущая скорость: {self.speed}')
    # метод для возвращения значений свойств класса
    def get_params(self):
        return (self.mark, self.model, self.color, self.speed)
# создаем экземпляр класса Car с заданными параметрами
my_car_1 = Car('Nissan', 'Juke', 'red', 0)
my_car_2 = Car()  # создаем экземпляр класса Car
```

Инициализатор и финализатор

```
class Car:     # создаем класс Car
    # инициализатор класса
    def __init__(self, mark='', model='', color='', speed=0):
        self.mark = mark     # марка
        self.model = model   # модель
        self.color = color   # цвет
        self.speed = speed   # скорость
    # финализатор класса
    def __del__(self):
        print('Сработал метод __del__')
    # метод для создания нового свойства - power_engine
    def set_power_engine(self, power_engine):
        self.power_engine = power_engine
    # метод для вывода информации о свойствах класса
    def show_info(self):
        print(f'Марка: {self.mark}')
        print(f'Модель: {self.model}')
        print(f'Цвет: {self.color}')
        print(f'Текущая скорость: {self.speed}')
    # метод для возвращения значений свойств класса
    def get_params(self):
        return (self.mark, self.model, self.color, self.speed)
# создаем экземпляр класса Car с заданными параметрами
my_car_1 = Car('Nissan', 'Juke', 'red', 0)
my_car_2 = Car()  # создаем экземпляр класса Car
```

Инициализатор и финализатор

```
class Car:     # создаем класс Car
    # инициализатор класса
    def __init__(self, mark='', model='', color='', speed=0):
        self.mark = mark      # марка
        self.model = model    # модель
        self.color = color    # цвет
        self.speed = speed    # скорость
    def __del__(self):        # финализатор класса
        print('Сработал метод __del__')
    # метод
    def se
        se
    # метод
    def show_info(self):
        print(f'Марка: {self.mark}')
        print(f'Модель: {self.model}')
        print(f'Цвет: {self.color}')
        print(f'Текущая скорость: {self.speed}')
    # метод для возвращения значений свойств класса
    def get_params(self):
        return (self.mark, self.model, self.color, self.speed)
# создаем экземпляр класса Car с заданными параметрами
my_car_1 = Car('Nissan', 'Juke', 'red', 0)
my_car_2 = Car()  # создаем экземпляр класса Car
```

```
>>> Сработал метод __del__
    Сработал метод __del__
```

Инициализатор и финализатор

__del__() - «магический» метод, который срабатывает перед удалением объекта класса.

Создайте класс **Hero** со следующими свойствами:

- имя (строка)
- здоровье (целое число)
- наносимый урон (целое число)
- защита (целое число)

Также в классе **Hero** должны быть описаны следующие методы:

- инициализация с начальными параметрами героя
- получить статус о параметрах героя
- увеличить защиту
- нанести урон
- получить урон

Примечания:

- метод увеличения защиты должен увеличивать текущую защиту героя в 1.5 раза
- уровень защиты игрока не может подниматься выше значения 100
- урон должен получаться героем, учитывая уровень защиты (формула заблокированного урона: *нанесенный урон * защита / 100*)

```
class Hero:  # создаем класса Hero
```

```
class Hero:  # создаем класса Hero
    # инициализатор класса
    def __init__(self, name, health, damage, defense):
```

```
class Hero:  # создаем класса Hero
    # инициализатор класса
    def __init__(self, name, health, damage, defense):
        self.name = name          # имя
```

```
class Hero: # создаем класса Hero
    # инициализатор класса
    def __init__(self, name, health, damage, defense):
        self.name = name          # имя
        self.health = health      # здоровье
```

```
class Hero: # создаем класса Hero
    # инициализатор класса
    def __init__(self, name, health, damage, defense):
        self.name = name          # имя
        self.health = health      # здоровье
        self.damage = damage      # наносимый урон
```

```
class Hero: # создаем класса Hero
    # инициализатор класса
    def __init__(self, name, health, damage, defense):
        self.name = name          # имя
        self.health = health      # здоровье
        self.damage = damage      # наносимый урон
        self.defense = defense    # защита
```

```
class Hero: # создаем класса Hero
    # инициализатор класса
    def __init__(self, name, health, damage, defense):
        self.name = name          # имя
        self.health = health      # здоровье
        self.damage = damage      # наносимый урон
        self.defense = defense    # защита
    # получаем статус параметров героя
    def get_status(self):
```

```
class Hero: # создаем класса Hero
    # инициализатор класса
    def __init__(self, name, health, damage, defense):
        self.name = name          # имя
        self.health = health      # здоровье
        self.damage = damage      # наносимый урон
        self.defense = defense    # защита
    # получаем статус параметров героя
    def get_status(self):
        print(f'Имя: {self.name}')
```

```
class Hero: # создаем класса Hero
    # инициализатор класса
    def __init__(self, name, health, damage, defense):
        self.name = name          # имя
        self.health = health      # здоровье
        self.damage = damage      # наносимый урон
        self.defense = defense    # защита
    # получаем статус параметров героя
    def get_status(self):
        print(f'Имя: {self.name}')
        print(f'Здоровье:{self.health}')
```

```
class Hero: # создаем класса Hero
    # инициализатор класса
    def __init__(self, name, health, damage, defense):
        self.name = name          # имя
        self.health = health      # здоровье
        self.damage = damage      # наносимый урон
        self.defense = defense    # защита
    # получаем статус параметров героя
    def get_status(self):
        print(f'Имя: {self.name}')
        print(f'Здоровье:{self.health}')
        print(f'Урон: {self.damage}')
```

```
class Hero: # создаем класса Hero
    # инициализатор класса
    def __init__(self, name, health, damage, defense):
        self.name = name          # имя
        self.health = health      # здоровье
        self.damage = damage      # наносимый урон
        self.defense = defense    # защита
    # получаем статус параметров героя
    def get_status(self):
        print(f'Имя: {self.name}')
        print(f'Здоровье:{self.health}')
        print(f'Урон: {self.damage}')
        print(f'Защита:{self.defense}')
        print('----------------------------------------------------' )
```

```
class Hero: # создаем класса Hero
    # инициализатор класса
    def __init__(self, name, health, damage, defense):
        self.name = name          # имя
        self.health = health      # здоровье
        self.damage = damage      # наносимый урон
        self.defense = defense    # защита
    # получаем статус параметров героя
    def get_status(self):
        print(f'Имя: {self.name}')
        print(f'Здоровье:{self.health}')
        print(f'Урон: {self.damage}')
        print(f'Защита:{self.defense}')
        print('----------------------------------------------------' )
    # метод увеличения защиты героя
    def increase_defense(self):
```

```
class Hero: # создаем класса Hero
    # инициализатор класса
    def __init__(self, name, health, damage, defense):
        self.name = name          # имя
        self.health = health      # здоровье
        self.damage = damage      # наносимый урон
        self.defense = defense    # защита
    # получаем статус параметров героя
    def get_status(self):
        print(f'Имя: {self.name}')
        print(f'Здоровье:{self.health}')
        print(f'Урон: {self.damage}')
        print(f'Защита:{self.defense}')
        print('----------------------------------------------------' )
    # метод увеличения защиты героя
    def increase_defense(self):
            self.defense *= 1.5
```

```
class Hero:  # создаем класса Hero
    # инициализатор класса
    def __init__(self, name, health, damage, defense):
        self.name = name          # имя
        self.health = health      # здоровье
        self.damage = damage      # наносимый урон
        self.defense = defense    # защита
    # получаем статус параметров героя
    def get_status(self):
        print(f'Имя: {self.name}')
        print(f'Здоровье:{self.health}')
        print(f'Урон: {self.damage}')
        print(f'Защита:{self.defense}')
        print('----------------------------------------------------' )
    # метод увеличения защиты героя
    def increase_defense(self):
        if self.defense < 100:    # если значение защиты меньше 100
            self.defense *= 1.5           # увеличение защиты в 1.5 раза
```

```
class Hero: # создаем класса Hero
    # инициализатор класса
    def __init__(self, name, health, damage, defense):
        self.name = name          # имя
        self.health = health      # здоровье
        self.damage = damage      # наносимый урон
        self.defense = defense    # защита
    # получаем статус параметров героя
    def get_status(self):
        print(f'Имя: {self.name}')
        print(f'Здоровье:{self.health}')
        print(f'Урон: {self.damage}')
        print(f'Защита:{self.defense}')
        print('----------------------------------------------------' )
    # метод увеличения защиты героя
    def increase_defense(self):
        if self.defense * 1.5 < 100:     # если значение защиты меньше 100
            self.defense *= 1.5          # увеличение защиты в 1.5 раза
```

```
class Hero: # создаем класса Hero
    # инициализатор класса
    def __init__(self, name, health, damage, defense):
        self.name = name          # имя
        self.health = health      # здоровье
        self.damage = damage      # наносимый урон
        self.defense = defense    # защита
    # получаем статус параметров героя
    def get_status(self):
        print(f'Имя: {self.name}')
        print(f'Здоровье:{self.health}')
        print(f'Урон: {self.damage}')
        print(f'Защита:{self.defense}')
        print('----------------------------------------------------' )
    # метод увеличения защиты героя
    def increase_defense(self):
        if self.defense * 1.5 < 100:     # если значение защиты меньше 100
            self.defense *= 1.5          # увеличение защиты в 1.5 раза
        else:
```

```
class Hero: # создаем класса Hero
    # инициализатор класса
    def __init__(self, name, health, damage, defense):
        self.name = name          # имя
        self.health = health      # здоровье
        self.damage = damage      # наносимый урон
        self.defense = defense    # защита
    # получаем статус параметров героя
    def get_status(self):
        print(f'Имя: {self.name}')
        print(f'Здоровье:{self.health}')
        print(f'Урон: {self.damage}')
        print(f'Защита:{self.defense}')
        print('----------------------------------------------------' )
    # метод увеличения защиты героя
    def increase_defense(self):
        if self.defense * 1.5 < 100:     # если значение защиты меньше 100
            self.defense *= 1.5          # увеличение защиты в 1.5 раза
        else:
            print('Достигнут максимальный уровень защиты!' )
```

```
class Hero:  # создаем класса Hero
    # инициализатор класса
    def __init__(self, name, health, damage, defense):
        self.name = name          # имя
        self.health = health      # здоровье
        self.damage = damage      # наносимый урон
        self.defense = defense    # защита
    # получаем статус параметров героя
    def get_status(self):
        print(f'Имя: {self.name}')
        print(f'Здоровье:{self.health}')
        print(f'Урон: {self.damage}')
        print(f'Защита:{self.defense}')
        print('----------------------------------------------------' )
    # метод увеличения защиты героя
    def increase_defense(self):
        if self.defense * 1.5 < 100:     # если значение защиты меньше 100
            self.defense *= 1.5          # увеличение защиты в 1.5 раза
        else:
            print('Достигнут максимальный уровень защиты!' )
        print(f'Текущий уровень защиты: {self.defense}')
        print('----------------------------------------------------' )
```

```
class Hero: # создаем класса Hero
    # инициализатор класса
    def __init__(self, name, health, damage, defense):
        self.name = name          # имя
        self.health = health      # здоровье
        self.damage = damage      # наносимый урон
        self.defense = defense    # защита
    # получаем статус параметров героя
    def get_status(self):
        print(f'Имя: {self.name}')
        print(f'Здоровье:{self.health}')
        print(f'Урон: {self.damage}')
        print(f'Защита:{self.defense}')
        print('----------------------------------------------------' )
    # метод увеличения защиты героя
    def increase_defense(self):
        if self.defense * 1.5 < 100:     # если значение защиты меньше 100
            self.defense *= 1.5          # увеличение защиты в 1.5 раза
        else:
            print('Достигнут максимальный уровень защиты!' )
        print(f'Текущий уровень защиты: {self.defense}')
        print('----------------------------------------------------' )
    # метод нанесения урона по врагу
    def make_damage():
```

```
class Hero: # создаем класса Hero
    # инициализатор класса
    def __init__(self, name, health, damage, defense):
        self.name = name          # имя
        self.health = health      # здоровье
        self.damage = damage      # наносимый урон
        self.defense = defense    # защита
    # получаем статус параметров героя
    def get_status(self):
        print(f'Имя: {self.name}')
        print(f'Здоровье:{self.health}')
        print(f'Урон: {self.damage}')
        print(f'Защита:{self.defense}')
        print('----------------------------------------------------' )
    # метод увеличения защиты героя
    def increase_defense(self):
        if self.defense * 1.5 < 100:     # если значение защиты меньше 100
            self.defense *= 1.5          # увеличение защиты в 1.5 раза
        else:
            print('Достигнут максимальный уровень защиты!' )
        print(f'Текущий уровень защиты: {self.defense}')
        print('----------------------------------------------------' )
    # метод нанесения урона по врагу
    def make_damage(self, enemy):
```

```
class Hero:  # создаем класса Hero
    # инициализатор класса
    def __init__(self, name, health, damage, defense):
        self.name = name          # имя
        self.health = health      # здоровье
        self.damage = damage      # наносимый урон
        self.defense = defense    # защита
    # получаем статус параметров героя
    def get_status(self):
        print(f'Имя: {self.name}')
        print(f'Здоровье:{self.health}')
        print(f'Урон: {self.damage}')
        print(f'Защита:{self.defense}')
        print('----------------------------------------------------' )
    # метод увеличения защиты героя
    def increase_defense(self):
        if self.defense * 1.5 < 100:     # если значение защиты меньше 100
            self.defense *= 1.5          # увеличение защиты в 1.5 раза
        else:
            print('Достигнут максимальный уровень защиты!' )
        print(f'Текущий уровень защиты: {self.defense}')
        print('----------------------------------------------------' )
    # метод нанесения урона по врагу
    def make_damage(self, enemy):
        print(f'Атака по персонажу {enemy.name}!')
```

```
class Hero: # создаем класса Hero
    # инициализатор класса
    def __init__(self, name, health, damage, defense):
        self.name = name          # имя
        self.health = health      # здоровье
        self.damage = damage      # наносимый урон
        self.defense = defense    # защита
    # получаем статус параметров героя
    def get_status(self):
        print(f'Имя: {self.name}')
        print(f'Здоровье:{self.health}')
        print(f'Урон: {self.damage}')
        print(f'Защита:{self.defense}')
        print('----------------------------------------------------' )
    # метод увеличения защиты героя
    def increase_defense(self):
        if self.defense * 1.5 < 100:     # если значение защиты меньше 100
            self.defense *= 1.5          # увеличение защиты в 1.5 раза
        else:
            print('Достигнут максимальный уровень защиты!' )
        print(f'Текущий уровень защиты: {self.defense}')
        print('----------------------------------------------------' )
    # метод нанесения урона по врагу
    def make_damage(self, enemy):
        print(f'Атака по персонажу {enemy.name}!')
        print('----------------------------------------------------' )
```

```
print('-----------------------------------------------------')


# метод получения урона, учитывая защиту
def get_damage():
```

```
print('----------------------------------------------------')
enemy.get_damage()  # вызываем метод get_damage() у вражеского героя

# метод получения урона, учитывая защиту
def get_damage():
```

```
print('-----------------------------------------------------')
enemy.get_damage()  # вызываем метод get_damage() у вражеского героя

# метод получения урона, учитывая защиту
def get_damage(self, damage):
```

```
print('-----------------------------------------------------')
enemy.get_damage(self.damage)  # вызываем метод get_damage() у вражеского героя

# метод получения урона, учитывая защиту
def get_damage(self, damage):
```

```
print('-----------------------------------------------------')
enemy.get_damage(self.damage)  # вызываем метод get_damage() у вражеского героя

# метод получения урона, учитывая защиту
def get_damage(self, damage):
    absorbed_damage =
```

```
print('-----------------------------------------------------')
enemy.get_damage(self.damage)  # вызываем метод get_damage() у вражеского героя

# метод получения урона, учитывая защиту
def get_damage(self, damage):
    absorbed_damage = damage  * self.defense / 100  # формула поглощенного урона
```

```
print('-----------------------------------------------------')
enemy.get_damage(self.damage)  # вызываем метод get_damage() у вражеского героя

# метод получения урона, учитывая защиту
def get_damage(self, damage):
    absorbed_damage = damage  * self.defense / 100  # формула поглощенного урона
    final_damage =
```

```
print('-----------------------------------------------------')
enemy.get_damage(self.damage)  # вызываем метод get_damage() у вражеского героя

# метод получения урона, учитывая защиту
def get_damage(self, damage):
    absorbed_damage = damage  * self.defense / 100  # формула поглощенного урона
    final_damage  = damage  - absorbed_damage   # вычисление финального урона
```

```
print('-----------------------------------------------------')
enemy.get_damage(self.damage)  # вызываем метод get_damage() у вражеского героя

# метод получения урона, учитывая защиту
def get_damage(self, damage):
    absorbed_damage = damage  * self.defense / 100  # формула поглощенного урона
    final_damage  = damage - absorbed_damage   # вычисление финального урона
    print(f'По герою {self.name} нанесли урон {final_damage}!')
```

```
print('-----------------------------------------------------')
enemy.get_damage(self.damage)  # вызываем метод get_damage() у вражеского героя

# метод получения урона, учитывая защиту
def get_damage(self, damage):
    absorbed_damage = damage  * self.defense / 100  # формула поглощенного урона
    final_damage  = damage - absorbed_damage   # вычисление финального урона
    print(f'По герою {self.name} нанесли урон {final_damage}!')
    self.health -= final_damage  # уменьшение здоровья героя
```

```
print('-----------------------------------------------------' )
enemy.get_damage(self.damage)  # вызываем метод get_damage() у вражеского героя

# метод получения урона, учитывая защиту
def get_damage(self, damage):
    absorbed_damage = damage  * self.defense / 100  # формула поглощенного урона
    final_damage  = damage - absorbed_damage   # вычисление финального урона
    print(f'По герою {self.name} нанесли урон {final_damage}!')
    self.health -= final_damage  # уменьшение здоровья героя
    print('-----------------------------------------------------' )
```

```
        print('-----------------------------------------------------' )
        enemy.get_damage(self.damage)  # вызываем метод get_damage() у вражеского героя

    # метод получения урона, учитывая защиту
    def get_damage(self, damage):
        absorbed_damage = damage  * self.defense / 100  # формула поглощенного урона
        final_damage  = damage - absorbed_damage   # вычисление финального урона
        print(f'По герою {self.name} нанесли урон {final_damage}!')
        self.health -= final_damage  # уменьшение здоровья героя
        print('-----------------------------------------------------' )
# создаем два экземпляра класса Hero
hero_1 = Hero('Артур', 100, 20, 5)
hero_2 = Hero('Робин', 80, 30, 4)
```

```
        print('-----------------------------------------------------' )
        enemy.get_damage(self.damage)  # вызываем метод get_damage() у вражеского героя

    # метод получения урона, учитывая защиту
    def get_damage(self, damage):
        absorbed_damage = damage  * self.defense / 100  # формула поглощенного урона
        final_damage  = damage - absorbed_damage   # вычисление финального урона
        print(f'По герою {self.name} нанесли урон {final_damage}!')
        self.health -= final_damage  # уменьшение здоровья героя
        print('-----------------------------------------------------' )
# создаем два экземпляра класса Hero
hero_1 = Hero(name='Артур', health=100, damage=20, defense=5)
hero_2 = Hero('Робин', 80, 30, 4)
```

```
        print('-----------------------------------------------------' )
        enemy.get_damage(self.damage)  # вызываем метод get_damage() у вражеского героя

    # метод получения урона, учитывая защиту
    def get_damage(self, damage):
        absorbed_damage = damage  * self.defense / 100  # формула поглощенного урона
        final_damage  = damage - absorbed_damage   # вычисление финального урона
        print(f'По герою {self.name} нанесли урон {final_damage}!')
        self.health -= final_damage  # уменьшение здоровья героя
        print('-----------------------------------------------------' )
# создаем два экземпляра класса Hero
hero_1 = Hero(name='Артур', health=100, damage=20, defense=5)
hero_2 = Hero('Робин', 80, 30, 4)
# получаем статус о параметрах Артура
hero_1.get_status()
# увеличиваем защиту Артура
hero_1.increase_defense()
```

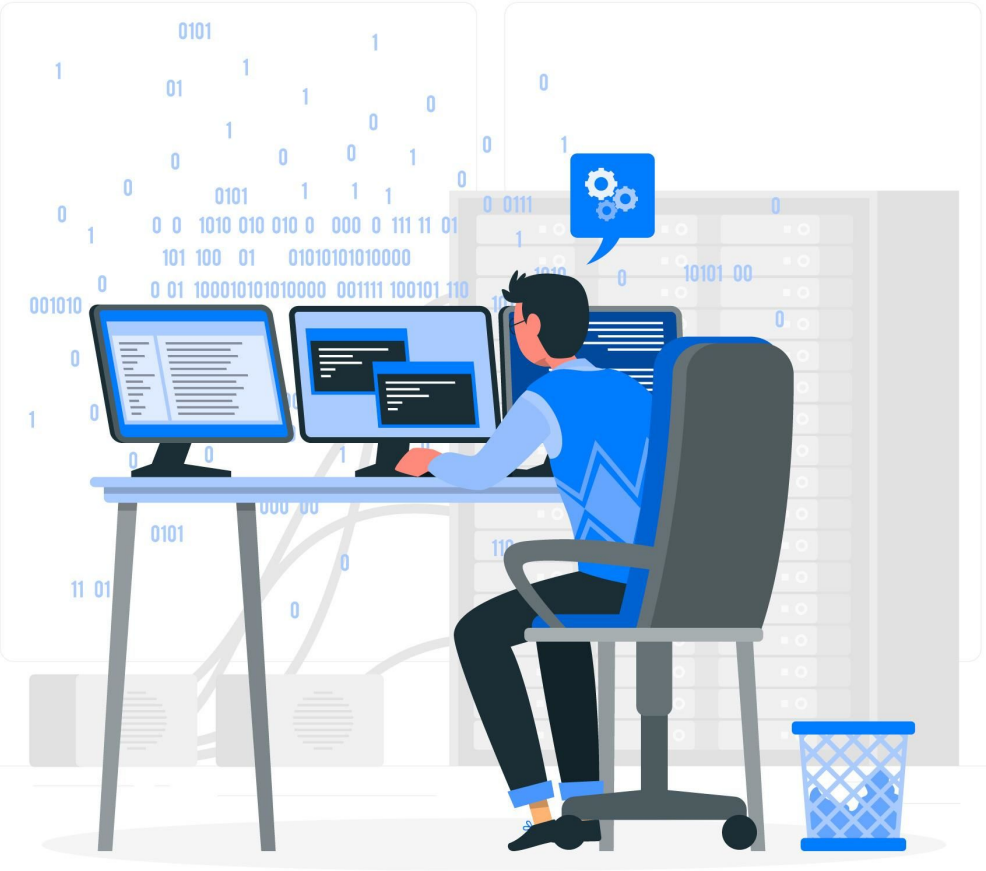
```
Имя: Артур
Здоровье:100
Урон: 20
Защита:5
-----------------------------------------
Текущий уровень защиты: 7.5
-----------------------------------------
```

```
        print('-----------------------------------------------------' )
        enemy.get_damage(self.damage)  # вызываем метод get_damage() у вражеского героя

    # метод получения урона, учитывая защиту
    def get_damage(self, damage):
        absorbed_damage = damage  * self.defense / 100  # формула поглощенного урона
        final_damage  = damage - absorbed_damage   # вычисление финального урона
        print(f'По герою {self.name} нанесли урон {final_damage}!')
        self.health -= final_damage  # уменьшение здоровья героя
        print('-----------------------------------------------------' )
# создаем два экземпляра класса Hero
hero_1 = Hero('Артур', 100, 20, 5)
hero_2 = Hero('Робин', 80, 30, 4)
# получаем статус о параметрах Артура
hero_1.get_status()
# увеличиваем защиту Артура
hero_1.increase_defense()
# получаем статус о параметрах Робина
hero_2.get_status()
# Робин наносит урон Артуру
hero_2.make_damage(hero_1)
```

```
        print('-----------------------------------------------------' )
        enemy.get_damage(self.damage)  # вызываем метод get_damage() у вражеского героя

    # метод получения урона, учитывая защиту
    def get_damage(self, damage):
        absorbed_damage = damage  * self.defense / 100  # формула поглощенного урона
        final_damage  = damage - absorbed_damage   # вычисление финального урона
        print(f'По герою {self.name} нанесли урон {final_damage}!')
        self.health -= final_damage  # уменьшение здоровья героя
        print('-----------------------------------------------------' )
# создаем два экземпляра класса Hero
hero_1 = Hero('Артур', 100, 20, 5)
hero_2 = Hero('Робин', 80, 30, 4)
# получаем статус о параметрах Артура
hero_1.get_status()
# увеличиваем защиту Артура
hero_1.increase_defense()
# получаем статус о параметрах Робина
hero_2.get_status()
# Робин наносит урон Артуру
hero_2.make_damage(hero_1)
# получаем статус о параметрах Артура
hero_1.get_status()
```

```
Имя: Робин
Здоровье:80
Урон: 30
Защита:4
---------------------------------------------------
Атака по персонажу Артур!
---------------------------------------------------
По герою Артур нанесли урон 27.75!
---------------------------------------------------
Имя: Артур
Здоровье:72.25
Урон: 20
Защита:7.5
---------------------------------------------------
```

ИТОГ



MAXIMUM EDUCATION

DS
Программирование
Python

Спасибо за внимание!