алгоритмика

Модуль 5. Урок 5.

Объектно-ориентированное программирование. Наследование

Подтверждение квалификации

Класс

- единое название для многих объектов;
- в программировании: общее описание того, как должны быть устроены эти объекты.

это — машина

Объект

Класс объекта

Знания про все такие объекты

Подтверждение квалификации

Экземпляр класса

— это объект, созданный по описанию, запрограммированному в классе.

экземпляр = Класс()

Объект получает всё, что знает и умеет класс.

Свойства

Методы

Создаём объект

Описываем класс:

Свойства

Методы

Подтверждение квалификации

«Черепашка» — это класс Turtle

Рассмотрим создание *экземпляра* класса Turtle:

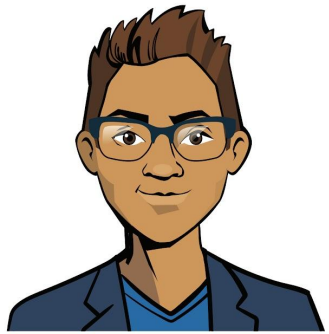
```
t1 = Turtle()
```

Экземпляр класса

Конструктор класса

Имя класса со скобками является **командой**, создающей новый объект этого класса.

Результат работы — ссылка, указывающая на объект (хранится в переменной).

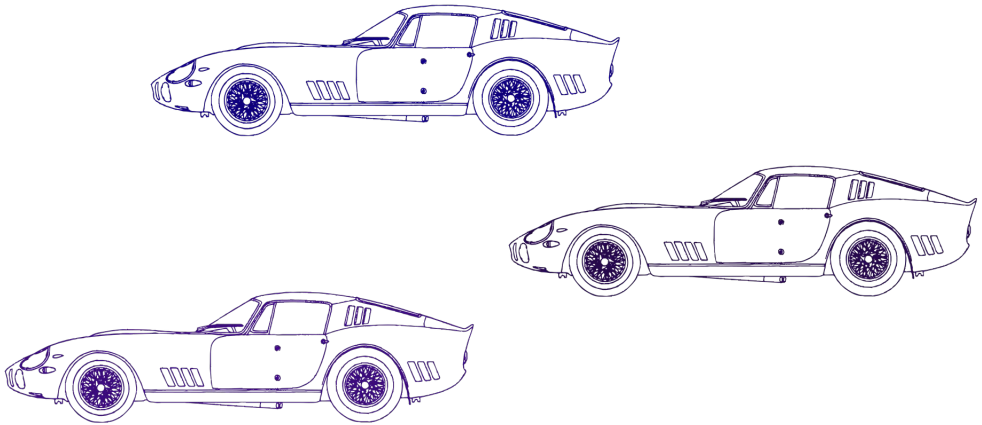


Подтверждение квалификации

Конструктор

— это метод, который автоматически вызывается при создании объекта. Он создаёт экземпляр класса.

```
def __init__(self, параметры) — название конструктора.
```



Подтверждение квалификации

Создание класса

class (*в англ. — «класс»*) — команда, создающая класс.

self (*в англ. — «сам, себя»*) — текущий объект класса.

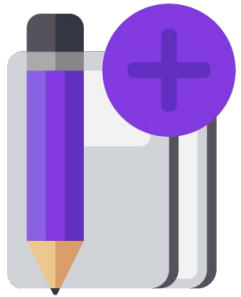
```
class Имя класса ():
    def __init__(self, Данные ):
        self. Свойство = Данные

    def print_info(self):
        print('Информация об объекте:', self. Свойство )

Экземпляр = Имя класса ( Свойство )
```

Конструктор с процессом создания экземпляра класса.

Подтверждение квалификации

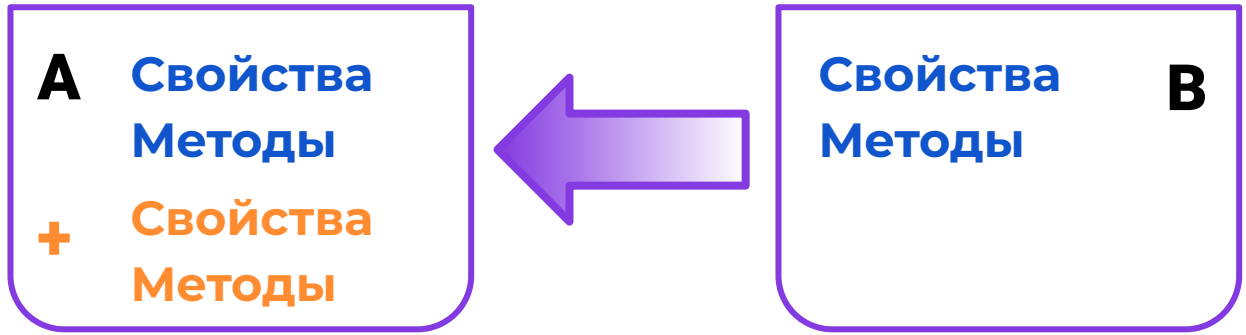
«Мозговой штурм»:

Наследование

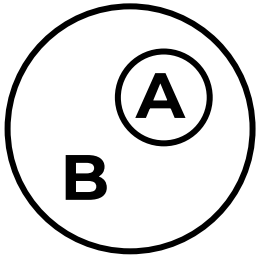


Наследование

Наследование классов помогает **перенести все умения**, написанные ранее для **более общего класса**, в другой, более частный класс, **класс-наследник**.



Класс-наследник **Суперкласс**



Класс A вложен в класс B

«Мозговой штурм»

Виды наследования

Вид	*Комментарий*
Класс-наследник дополняется **новыми методами**, а новые свойства не вводятся.	Новый конструктор не нужен, можно использовать конструктор суперкласса!
Класс-наследник дополняется и **новыми свойствами**, и **новыми методами**.	Требуется дополнение конструктора суперкласса новыми свойствами.

«Мозговой штурм»

Создание класса-наследника

Пусть суперкласс уже написан, тогда,
чтобы создать класс-наследник нужно:

- при создании наследника указать *имя суперкласса*;
- дополнить класс-наследник необходимыми методами.

```
class Cat(Animal):
    def hello(self, voice):
        print(self.voice)

    def Название метода(self, Значение):
        Действие с объектом и св-вами
```

Вариант с введением ***только новых методов****.*

При создании экземпляра наследника будет вызван конструктор суперкласса!

«Мозговой штурм»

Создание класса-наследника

Чтобы создать класс-наследник нужно:

- при создании наследника указать *имя суперкласса*;
- создать конструктор, ввести свойства суперкласса и добавить новые;
- дополнить класс-наследник необходимыми методами.

```
class Cat(Animal):
    def __init__(self, name, age):
        super().__init__(age)
        self.imya = name
    def Название метода(self, Значение):
        Действие с объектом и св-вами
```

*Вариант с **введением нового свойства**.*

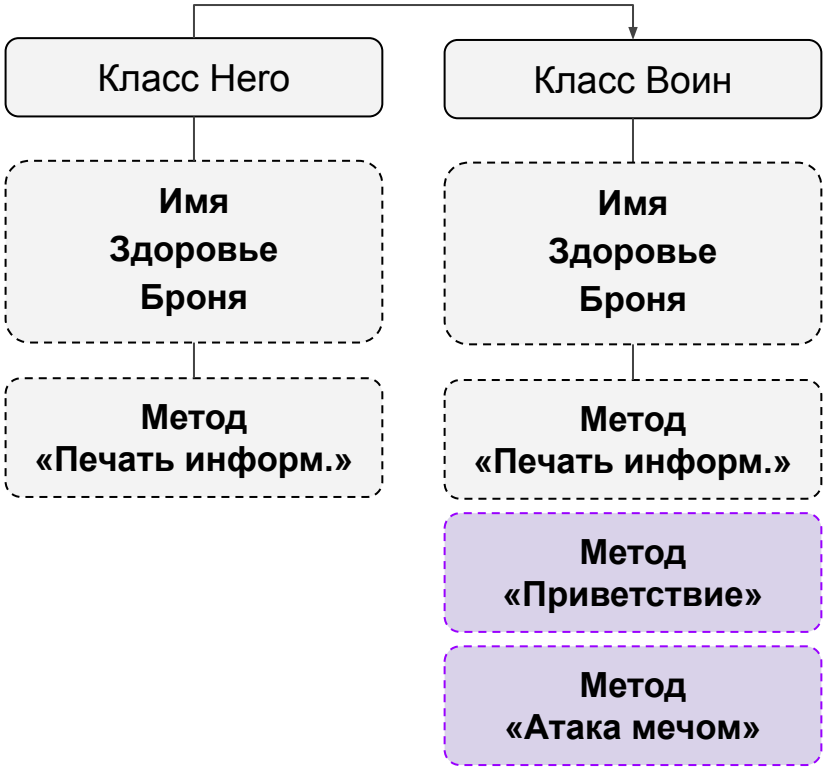
Конструктор перенимает свойства суперкласса и добавляет новое.

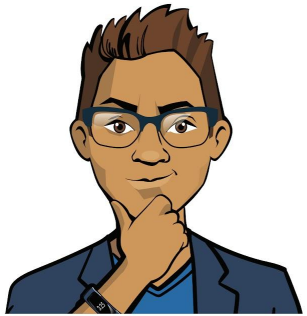
«Мозговой штурм»

Рассмотрим тренировочную задачу

Имеется фрагмент кода с классом Hero.

Задача: реализовать класс-наследник Warrior по данной схеме.





«Мозговой штурм»

Рассмотрим тренировочную задачу

Имеется фрагмент кода с классом Hero.

Задача: реализовать класс-наследник Warrior по данной схеме.

```
class Hero():
    #конструктор класса
    def __init__(self, a, b, c):
        self.name = a #строка
        self.health = b #число
        self.armor = c #число
    #печать параметров персонажа:
    def print_info(self):
        print('Уровень здоровья:', self.health)
        print('Уровень брони:', self.armor, '\n')
```

Суперкласс

```
class Warrior(Hero):
    def hello(self):
```

Указываем имя суперкласса, у которого заимствуем конструктор.

Приветствие воина («Верхом на коне появился воин...»).

Печатаем параметры методом print_info()

```
    def attack(self, enemy):
```

Текст про атаку мечом («Храбрый воин атакует мечом...»).

Большая сила удара (например, 15).

Класс-наследник

Рассмотрим тренировочную задачу

Имеется фрагмент кода с классом Hero.

Задача: реализовать класс-наследник Warrior по данной схеме.

```
warrior1 = Warrior('Henry', 100, 50)
warrior1.hello()
warrior.print_info()
warrior1.attack(<Имя врага>)
```

```
class Warrior(Hero):
    def hello(self):
```

Приветствие воина
(«Верхом на коне появился воин...»).

Печатаем параметры методом print_info().

```
    def attack(self, enemy):
```

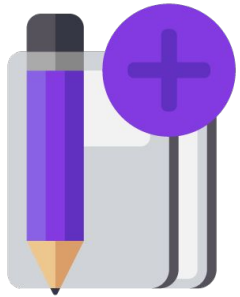
Текст про атаку мечом
(«Храбрый воин атакует мечом...»).

Большая сила удара (например, 15).

Перерыв

«Мозговой штурм»:

Игра «Hit It!»

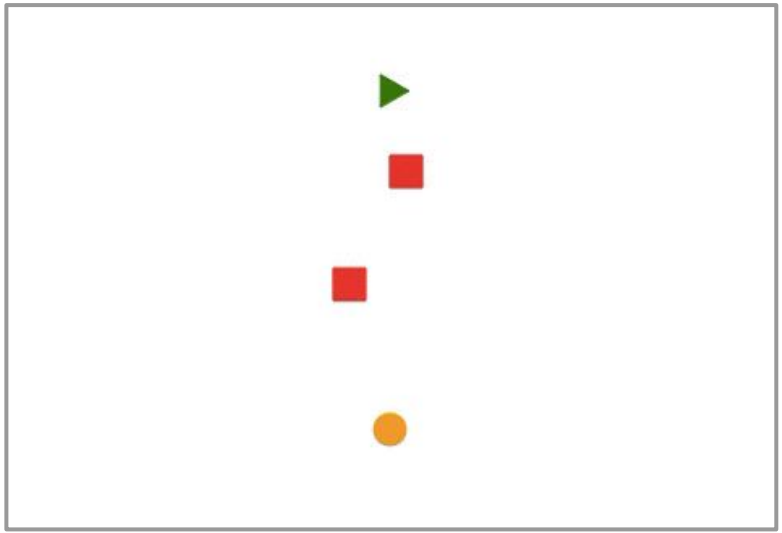


Запрограммируем игру «Hit It!»

В рамках тренинга по созданию собственных объектов запрограммируем интерактивную игру «Hit It!».

Цель игры — обойти препятствия и поймать целевой объект.

Ожидаемый вид игры:



«Мозговой штурм»

Техническое задание

Цель игры — обойти препятствия и поймать целевой объект.

Требования к игре:

1. **Основной объект** управляется пользователем с клавиатуры.
2. Не менее **двух объектов** перемещаются по экрану автоматически и мешают достижению цели.
3. Условие победы: игрок касается **целевого объекта**.
   Тогда препятствия исчезают.

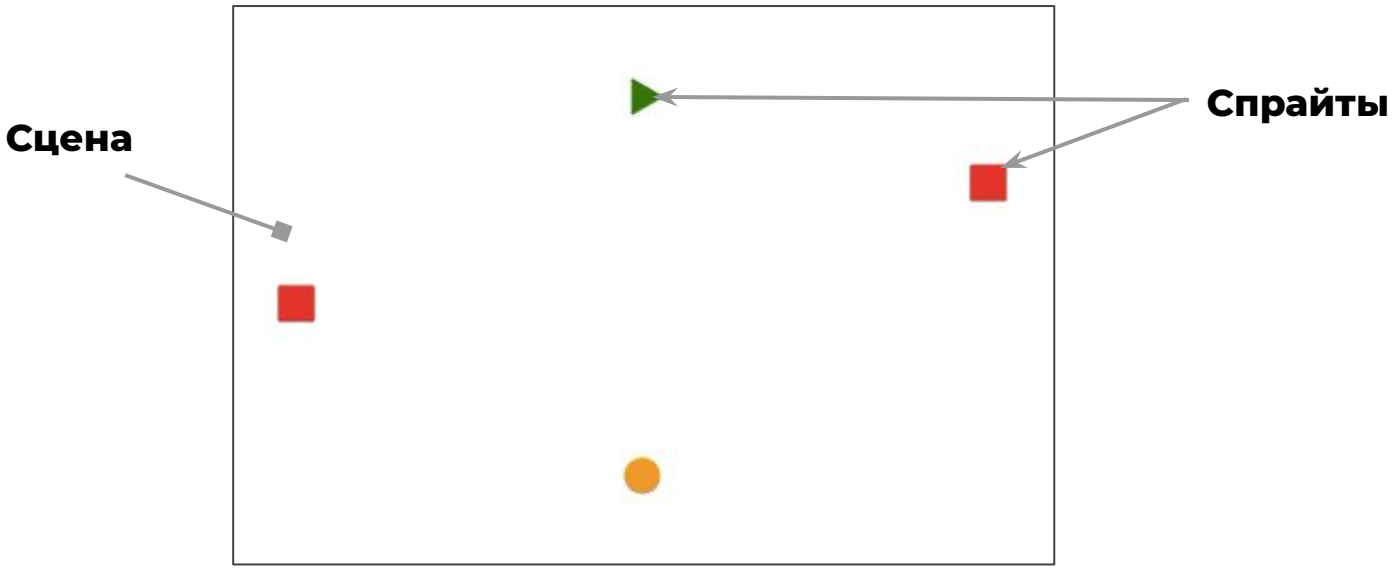
   Условие поражения: игрок касается **любого препятствия**.
   Тогда исчезает целевой объект.

«Мозговой штурм»

Термины для создания игр

Сцена — это «фон» игры. То, по чему перемещаются все объекты.

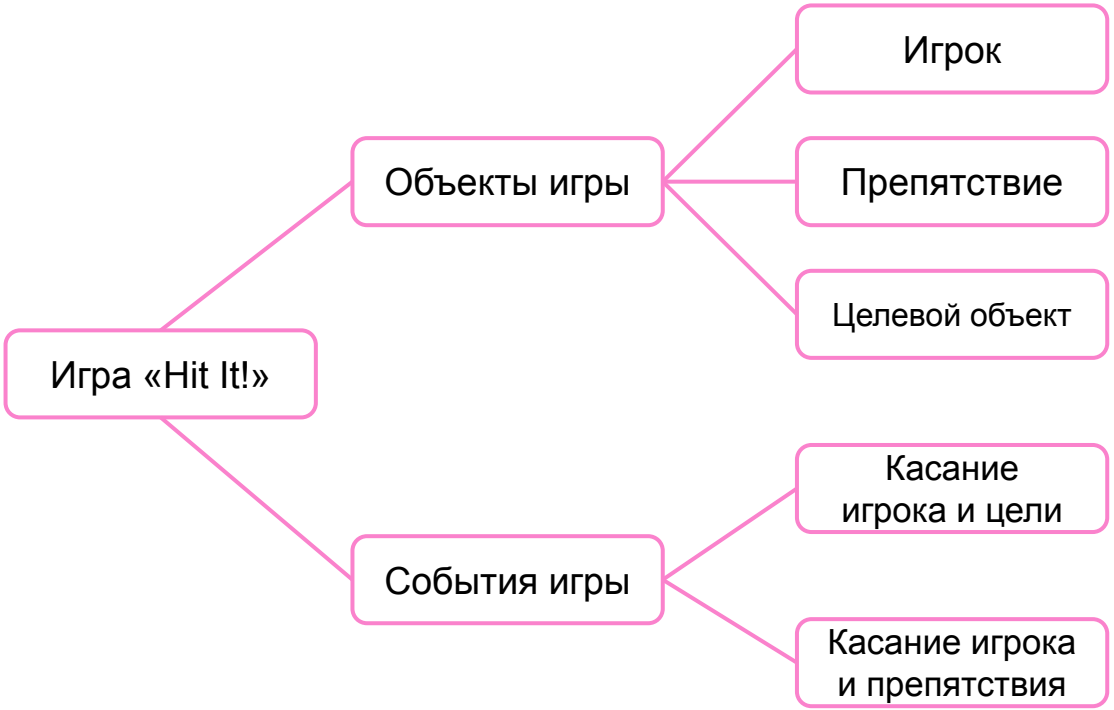
Спрайт — это любой игровой объект, отличный от сцены.



«Мозговой штурм»

Планируем работу над проектом

Изобразим функционал проекта на mind map:



Методы для отображения и перемещения есть в классе Turtle!

В классе Turtle нет — это специфика нашей игры.

Как «научить» объекты распознавать касание?

«Мозговой штурм»

Класс Sprite

Реализуем часть класса для создания спрайта с нужными свойствами (цветом, формой, положением).

Класс Sprite — наследник Turtle

```
class Sprite(Turtle):
    def __init__(self, x, y, stp=10, sp='circle', clr='black'):
        super().__init__()
        self.penup()
        self.speed(0)
        self.goto(x, y)
        self.color(clr)
        self.shape(sp)
        self.step = stp


player = Sprite(0, -100, 20, ‘turtle’, ‘violet’)
```

«Мозговой штурм»

Управление спрайтом-игроком

Способ управления объектом-черепашкой с клавиатуры нам известен.

Управлять объектом Sprite можно так же, ведь он унаследовал все свойства и методы Turtle!

Основной код программы:

```
scr = Screen()
scr.listen()


scr.onkey(player.move_up, 'Up')
scr.onkey(player.move_left, 'Left')
scr.onkey(player.move_right, 'Right')
scr.onkey(player.move_down, 'Down')
```

Обратите внимание!

Чтобы объект scr стал восприимчив к нажатию клавиш, необходимо кликнуть на экран мышкой.

После этого «заработают» и нажатия на клавиши.

«Мозговой штурм»

Управление спрайтом-игроком

Класс Sprite:

```
def move_up(self):
        self.goto(self.xcor(), self.ycor() + self.step)
```

...спрайт должен сместиться на один шаг (указывался при создании) в нужном направлении.

Основной код программы:

```
scr = Screen()
scr.listen()

scr.onkey(player.move_up, 'Up')
scr.onkey(player.move_left, 'Left')
scr.onkey(player.move_right, 'Right')
scr.onkey(player.move_down, 'Down')
```

При нажатии на клавишу «стрелка вверх»...

«Мозговой штурм»

Платформа:

Игра «Hit It!»