

Классы



Класс

В объектно-ориентированном программировании класс – это расширяемый шаблон кода для создания объектов, который устанавливает в них начальные значения (свойства) и реализацию поведения (методы).

Класс

Классы в ECMAScript 6 представляют собой синтаксические ухищрения, маскирующие текущий подход, основанный на конструкторах и прототипах.

Синтаксис классов не вводит новую объектно-ориентированную модель, а предоставляет более простой и понятный способ создания объектов и организации наследования.

Класс

```
function Person(name, age){
    this.name = name;
    this.age = age;
}

Person.prototype.sayName = function(){
    alert(this.name);
};

Person.prototype.getOlder = function(years){
    this.age += years;
};
```

```
class Person {
    constructor(name, age) {
        this.name = name;
        this.age = age;
    }
    sayName() {
        alert(this.name);
    }
    getOlder(years) {
        this.age += years;
    }
}
```

Задание 1

Переписать функцию-конструктор в виде класса:

```
function Car(name, year, color) {  
  this.name = name;  
  this.year = year;  
  this.color = color;  
  
  this.changeColor = function (newColor) {  
    this.color = newColor;  
  }  
}
```

```
let car = new Car('BMW', 2012, 'red');  
car.changeColor('black');
```

Класс

Определение класса начинается с ключевого слова `class`, за которым следует имя типа. Свойства и методы класса определяются в фигурных скобках. Чтобы определить метод, достаточно указать его имя и скобки, а ключевое слово `function` не требуется.

Если метод называется `constructor`, он работает как конструктор класса (аналогично функции `Person`). Все остальные методы и свойства, определенные в фигурных скобках класса, применяются к прототипу.

Класс

Метод `constructor` — специальный метод, необходимый для создания и инициализации объектов, созданных, с помощью класса. В классе может быть только один метод с именем `constructor`

Класс

```
class Person {  
  
    constructor(name) {  
        this.name = name;  
    }  
  
    sayHi() {  
        alert(this.name);  
    }  
  
}  
  
let user = new Person("Alex");  
user.sayHi();
```

Когда вызывается `new Person`:

Создаётся новый объект.

`constructor` запускается с заданным аргументом и сохраняет его в `this.name`.

Класс. Геттеры и сеттеры

```
class Person {  
  constructor(name) {  
    // + ВЫЗОВ СЕТТЕРА  
    this.name = name;  
  }  
  get name() { return this._name; }  
  set name(value) {  
    if (value.length < 1) { alert(" Too small");  
      return; }  
    this._name = value;  
  }  
}
```

```
let user = new Person("Alex");  
alert(user.name); // Alex
```

Класс. Свойства

```
class Person {  
  name = "Someone";  
  
  sayHi() {  
    alert(`Hello, ${this.name}!`);  
  }  
}
```

Задание 2

Ваша задача реализовать класс Person. Вы должны заполнить метод Constructor, чтобы принять имя как строку и возраст как число, реализовать метод getInfo получения информации, который должен вернуть, например, "johns age 34"

Класс. Наследование.

Главное преимущество классов над более традиционным javascript-синтаксисом - простота реализации наследования. Можно использовать простой синтаксис, общий для многих языков: ключевое слово `extends`.

```
class Employee extends Person {  
  
}
```

Класс. Наследование.

Если мы определим свой метод с таким же именем в дочернем классе, то он будет использоваться взамен родительского

Обычно мы не заменяем родительский метод, а скорее делаем новый на его основе, изменяя или расширяя его функциональность. Мы делаем что-то в нашем методе и вызываем родительский метод до/после или в процессе с помощью ключевого слова “super”

Класс. Наследование.

- `super.method(...)` вызывает родительский метод.
- `super(...)` вызывает родительский конструктор (работает только внутри нашего конструктора).

```
hide() {  
    super.hide(); // вызываем родительский метод hide  
    this.delete(); // и затем delete  
}
```

Класс. Наследование.

Наследование конструкторов:

```
constructor(name, age) {  
    super(name);  
    this.age = age;  
}
```

Задание 3

Ваша задача - реализовать класс Cat, который расширяет Animal, и заменить метод speak, чтобы вернуть имя кошки + мяу. например «Mr Whiskers meows.». Атрибут name передается с this.name

```
class Animal {  
  constructor(name) {  
    this.name = name;  
  }  
  speak() {  
    alert(this.name + ' says hi');  
  }  
}
```


Проверка класса: "instanceof"

Чтобы проверить, к какому классу принадлежит объект, с учётом наследования:

```
obj instanceof Class
```

Задание 4

Создать класс Person со стандартными свойствами (имя, возраст) и действиями(идти, разговаривать, есть) человека, от него наследовать классы Ребенок и взрослый со своими особенными и методами свойствами (н-р, у ребенка - плакать или ходить в школу)

Задание 5

Реализовать с помощью классов функциональность из задачи 3 темы “прототипы” :

Реализовать на основе прототипного наследования создание модальных окон (например, базовая функция модальное окно, с методами показа и скрытия, от которого наследуются функции создания предупреждающего окна, запрещающего окна, окна с успешным выполнением)

Задание 7

Создать класс Ball, тип которого regular, если аргумент не передан, super, если передан аргумент "superBall"

```
ball1 = new Ball();  
ball2 = new Ball("superBall");
```

```
ball1.ballType //=> "regular"  
ball2.ballType //=> "superBall"
```