# Prototype

## Цепочка прототипов в JavaScript – это механизм, позволяющий объектам наследовать свойства и методы других объектов.

1. **Что такое Prototype?**
   - Все объекты в JavaScript имеют свойство `prototype`, которое указывает на другой объект. Если JavaScript не может найти требуемое свойство или метод в объекте, он переходит к его прототипу, и так далее вверх по цепочке, пока не достигнет базового объекта `Object.prototype`, у которого прототипа нет, или не найдет требуемое свойство/метод. Если свойство не найдено, возвращается `undefined`.
2. **Создание цепочки прототипов:**
   - С помощью функции-конструктора и свойства `prototype`:

```
function Animal() {}
Animal.prototype.eats = true;
Animal.prototype.walk = function() {
    console.log("Walking...");
};
function Rabbit() {}
Rabbit.prototype = Object.create(Animal.prototype);
Rabbit.prototype.jump = function() {
    console.log("Jumping...");
};
let rabbit = new Rabbit();
rabbit.walk();  // Walking...
rabbit.jump();  // Jumping...
```

   - Используя классы и ключевое слово `extends`:

```
class Animal {
    eats = true;
    walk() {
        console.log("Walking...");
    }
}
class Rabbit extends Animal {
    jump() {
        console.log("Jumping...");
    }
}
```

```
let rabbit = new Rabbit();
rabbit.walk();  // Walking...
rabbit.jump();  // Jumping...
```

3. **Работа с цепочкой прототипов:**
    - Чтобы проверить, является ли объект прототипом другого объекта, можно использовать метод `Object.prototype.isPrototypeOf`:

```
console.log(Animal.prototype.isPrototypeOf(rabbit));  // true
```

    - Для определения прототипа объекта используется функция `Object.getPrototypeOf`:

```
console.log(Object.getPrototypeOf(rabbit) === Rabbit.prototype);  // true
```

4. **Изменение и расширение цепочек прототипов:**
    - Цепочку прототипов можно модифицировать, добавляя или изменяя свойства и методы прототипов:

```
Rabbit.prototype.run = function() {
    console.log("Running...");
};
rabbit.run();  // Running...
```

Цепочка прототипов - это мощный механизм в JavaScript, позволяющий объектам делиться поведением и свойствами. Современные классы предлагают более чистый и понятный синтаксис для работы с прототипами и наследованием.