# ПМ3 Разработка модулей ПО.

# РО 3.1 Понимать и применять принципы объектно-ориентированного и асинхронного программирования.

# Тема 1. Введение в ООП.

# Лекция 4. Прототипы и прототипное наследование.

# Цель занятия:

Познакомиться с прототипами в JavaScript и понять механизм наследования через цепочку прототипов.

# Учебные вопросы:

1. Основы прототипов.

2. Цепочка прототипов. Механизм поиска свойств.

# 1. Основы прототипов.

**Прототип** в JavaScript — это механизм, который позволяет объектам наследовать свойства и методы друг от друга.

Каждый объект в JavaScript имеет внутреннюю ссылку на другой объект, который называется его прототипом

Когда вы пытаетесь получить доступ к свойству объекта, JavaScript сначала ищет это свойство в самом объекте, а затем — в его прототипе, и так далее, пока не найдет его или не дойдет до конца цепочки ссылок (до **null**).

**Свойство prototype у функций-конструкторов.**

Функции-конструкторы в JavaScript — это функции, используемые для создания объектов.

Когда вы создаете объект с помощью функции-конструктора (с использованием ключевого слова new), JavaScript устанавливает свойство prototype этой функции в качестве прототипа создаваемого объекта.

Пример:

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

console.log(typeof Person.prototype) // object

// Добавляем метод greet в прототип Person
Person.prototype.greet = function() {
    console.log(`Hello, my name is ${this.name}`);
};

const alice = new Person('Alice');
alice.greet(); // "Hello, my name is Alice"
```

**Как это работает:**

- Создание объекта: Когда вы создаете объект alice с помощью new Person('Alice'), JavaScript создает новый объект и устанавливает его внутреннее свойство [[Prototype]] на Person.prototype.
- Наследование: Когда вы вызываете alice.greet(), JavaScript ищет метод greet сначала в объекте alice, а затем в Person.prototype, где он и находит его.

**Зачем это нужно?**

Использование прототипов позволяет экономить память, так как все экземпляры одного и того же типа могут использовать одни и те же методы, определенные в прототипе, вместо того чтобы иметь свои собственные копии этих методов.

Таким образом, прототипы в JavaScript являются ключевым элементом наследования и организации кода.

## Различия между prototype, proto и [[Prototype]]

В JavaScript три термина — prototype, __proto__ и [[Prototype]] — относятся к механизму прототипного наследования, но они имеют разные значения и назначения.

**[[Prototype]]:**

- Внутреннее (скрытое) свойство любого объекта, которое хранит ссылку на прототип объекта, не доступное напрямую.
- Указывает на другой объект, из которого берутся свойства и методы, если их нет у самого объекта.
- По сути – ссылка на родителя.
- Доступ к нему получить напрямую нельзя.

**Пример:**

У вас есть объект **user,** если у него нет метода **getInfo()** JS пойдет искать его в **[[Prototype]].**

## __proto__

- Старый, неформальный способ получить или задать [[Prototype]].
- Работает почти везде, но считается устаревшим.
- По сути это геттер/сеттер для [[Prototype]].

**prototype**

- Свойство функции-конструктора.
- Когда вы создаете объект с помощью **new**, именно **prototype** этой функции становится **[[Prototype]]**

```
function Person(name){
    this.name = name;
}
Person.prototype.sayHello = function(){
    console.log(`Hello, I'm ${name}`);
}


const alex = new Person("Alex");
alex.sayHello(); // Hello, I'm
```

- Здесь Alex.[[prototype]] === Person.prototype

Простыми словами:

- **prototype** – это чертеж для всех объектов, созданных через new.
- **[[Prototype]]** – это невидимая цепочка наследования для конкретного объекта.
- **__proto__** - это дверца, через которую можно заглянуть или поменять этот родитель (устаревший способ)

# 2. Цепочка прототипов. Механизм поиска свойств.

Как работает цепочка прототипов?

Объект: Когда вы создаете объект, он имеет ссылку на свой прототип, который может быть другим объектом или null.

Поиск свойства: Когда вы обращаетесь к свойству объекта:

- JavaScript сначала ищет это свойство в самом объекте.
- Если оно не найдено, JavaScript переходит к прототипу объекта (через свойство [[Prototype]] или __proto__).
- Этот процесс продолжается, пока не будет найдено свойство или не будет достигнут null.

Пример:

```
Object.prototype.sayHello = function(){console.log("Hello!")}

const animal = {
  makeSound: function(){
    console.log("Sound!")
  }
}

const dog = Object.create(animal);
dog.sayHello();
```

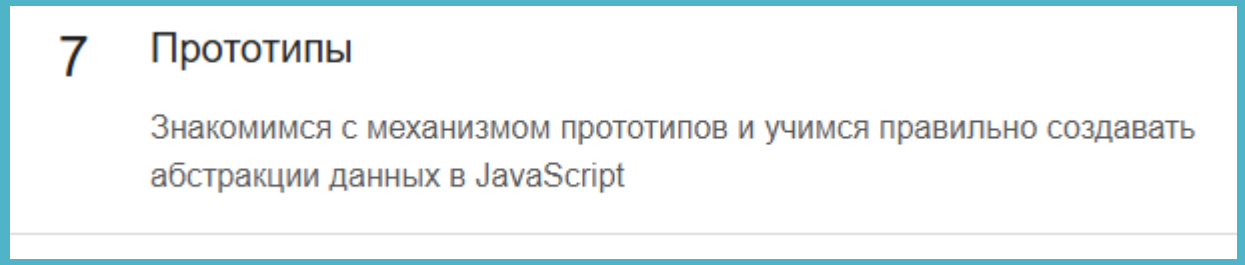
```
> dog
< ▾ {} i
    ▾ [[Prototype]]: Object
      ▸ makeSound: ƒ ()
      ▾ [[Prototype]]: Object
        ▸ sayHello: ƒ ()
```

**Механизм поиска свойства.**

- Запрос свойства: Вы вызываете dog.sayHello().
- Поиск в объекте: JavaScript ищет метод sayHello в объекте dog. Он его не находит.
- Поиск в прототипе: JavaScript переходит к dog.prototype (animal )и ненаходит его.
- Поиск в прототипе прототипа: JavaScript переходит к animal.prototype (Object) и находит его там.
- Выполнение метода: Метод вызывается.

# Домашнее задание:

1. https://ru.hexlet.io/courses/js-introduction-to-oop




2. Повторить материал лекции.

## Материалы лекций:

https://github.com/ShViktor72/Education2025

## Обратная связь:

colledge20education23@gmail.com