Week 19

1. Класс — это шаблон, описывающий свойства и методы, которыми будет обладать любой объект, созданный на основе этого класса. Класс - это только структура, а объекты - это уже конкретные переменные с заполненными полями (свойствами) на основе этой структуры.   
   Примеры:  
   класс - дети; экземпляры - мальчики и девочки;  
   класс - одежда в шкафу; экземпляры - платья, рубашки, пальто….;  
   класс - еда в холодильнике; экземпляры - колбаса, сыр, хлеб ….ж

Например, класс JavaScript для создания автомобиля теперь имеет вот такой простой синтаксис:

class Car {

constructor(brand, model) {

this.brand = brand; //бренд

this.model = model; //модель

}

drive() {

console.log(`Поехали на ${this.brand} ${this.model}!`);

}

}

let car1 = new Car('Tesla', '3');

car1.drive(); //Поехали на Tesla 3!

let car2 = new Car('Land Cruiser', '200');

car2.drive(); //Поехали на Land Cruiser 200!

2.   
В жизни:  
Ресторан   
классы: меню (состав блюд, масса, стоимость); сотрудники (официанты, повара, уборщики); система оценки качества (удовлетворенность клиентов, отзывы сотрудников, отзывы критиков).  
Пассажирский поезд  
классы: сотрудники (проводники, машинист, помощники, охрана); удобства (расположение мест, количество человек в купе, наличие ресторана, выбор еды); качество перевозки (скорость, задержки, удобное время прибытия и отправления).

Квартира  
классы: жители (пол, возраст, характер), удобства (площадь, этаж, вид из окна, количество комнат), соседи (возраст, активность днем, активность ночью, наличие шумных животных).

В программировании:  
Игры в телефоне (возрастные ограничения, качество графики, скорость загрузки, популярность).

Приложение для смартфона (стоимость скачивания, количество загрузок, отзывы).  
Сайт в интернете (качество верстки (адаптивность, например), скорость загрузки, функциональность, внешняя привлекательность, удобство подачи информации, простота регистрации и обратной связи.

3. Функции-конструкторы технически являются обычными функциями. Но есть два соглашения:

1. Имя функции-конструктора должно начинаться с большой буквы.
2. Функция-конструктор должна выполняться только с помощью оператора «new".  
     
   Функция-конструктор описывает шаблон, по которому будут создаваться объекты (экземпляры). В основе такого шаблона лежит перечисление свойств и методов, которыми будет обладать объект, созданный на основе единой структуры (класса). Для всех встроенных типов данных в JavaScript существуют встроенные функции-конструкторы.  
   Например, когда мы объявляем строковую переменную:  
   let str = 'Доброе утро!’;  
   на самом деле мы неявным образом вызываем встроенную функцию-конструктор:

В JavaScript есть два типа конструкторов  
**1. Встроенные конструкторы**

Это легкодоступные конструкторы, которые поставляются в комплекте со средой исполнения. Пользователь просто должен вызвать их и альт, работа сделана. Примерами встроенных конструкторов являются Array, Date и Object.

**2. Пользовательские Конструкторы**

Это конструкторы, объявленные и определенные программистом для использования в приложении. Программист также может определять свойства и методы своих собственных пользовательских типов. Они также известны как пользовательские конструкторы. По соглашению все конструкторы JavaScript заключаются в предложения. Это говорит человеку, использующему их, что эта функция должна вызываться с использованием нового ключевого слова.

4. function bike() {

console.log(this.name)

}

let name="ninja"

let obj1 = {name: "pomidor", bike: bike}

let obj2 = {name: "site", bike: bike}

bike();

obj1.bike();

obj2.bike();  
  
Выводит:  
[Log] pomidor (homework.html, line 12)

[Log] site (homework.html, line 12)  
  
bike(); ничего не выводит, так как в свойствах не указан параметр name.  
У объектов 1 и 2 в свойствах указаны имена, следовательно функция их выводит.

5. Методы и свойства класса, которые не привязаны к конкретному объекту и могут существовать без создания экземпляра и слова `new`, называются \*статическими\*.

В классе такие методы обозначаются ключевым словом `static`. Обычно статические методы используются для реализации функций, принадлежащих классу, но не к каким-то конкретным его объектам.  
  
Статические свойства также возможны, они выглядят как свойства класса, но с `static` в начале:

class Article {

static publisher = "Эксмо";

}

alert( Article.publisher ); // Эксмо

Это то же самое, что и прямое присваивание `Article`:

Article.publisher = "Эксмо";

Зачем еще это может быть нужно? Очень удобно использовать статические методы для создания классов-калькуляторов, классов-утилит, классов-помощников для работы с какими-то другими классами.  
  
6. **Свойства - геттеры и сеттеры**

Есть два типа свойств объекта.

Первый тип это *свойства-данные (data properties)*. Мы уже знаем, как работать с ними. Все свойства, которые мы использовали до текущего момента, были свойствами-данными.

Второй тип свойств мы ещё не рассматривали. Это *свойства-аксессоры (accessor properties)*. По своей сути это функции, которые используются для присвоения и получения значения, но во внешнем коде они выглядят как обычные свойства объекта.

Геттеры и сеттеры

Свойства-аксессоры представлены методами: «геттер» – для чтения и «сеттер» – для записи. При литеральном объявлении объекта они обозначаются get и set:  
  
let obj = {

get propName() {

// геттер, срабатывает при чтении obj.propName

},

set propName(value) {

// сеттер, срабатывает при записи obj.propName = value

}

};  
Геттер срабатывает, когда obj.propName читается, сеттер – когда значение присваивается.

Например, у нас есть объект user со свойствами name и surname:  
let user = {

name: "John",

surname: "Smith"

};  
А теперь добавим свойство объекта fullName для полного имени, которое в нашем случае "John Smith". Само собой, мы не хотим дублировать уже имеющуюся информацию, так что реализуем его при помощи аксессора:  
let user = {

name: "John",

surname: "Smith",

get fullName() {

return `${this.name} ${this.surname}`;

}

};

alert(user.fullName); // John Smith  
Снаружи свойство-аксессор выглядит как обычное свойство. В этом и заключается смысл свойств-аксессоров. Мы не *вызываем* user.fullName как функцию, а *читаем* как обычное свойство: геттер выполнит всю работу за кулисами.

На данный момент у fullName есть только геттер. Если мы попытаемся назначить user.fullName=, произойдёт ошибка:  
let user = {

get fullName() {

return `...`;

}

};

user.fullName = "Тест"; // Ошибка (у свойства есть только геттер)  
  
Давайте исправим это, добавив сеттер для user.fullName:  
let user = {

name: "John",

surname: "Smith",

get fullName() {

return `${this.name} ${this.surname}`;

},

set fullName(value) {

[this.name, this.surname] = value.split(" ");

}

};

// set fullName запустится с данным значением

user.fullName = "Alice Cooper";

alert(user.name); // Alice

alert(user.surname); // Cooper  
  
В итоге мы получили «виртуальное» свойство fullName. Его можно прочитать и изменить.

7. class Person {

construcnor(name) {

this.name = name

}

}

const member = new Person("John");

console.log(typeof member);  
  
выводит object. Потому что new Person - это объект, созданный на основе класса Персона.  
  
8. const person = {

name: "Valera",

age: 23

}

let city = person.city;

city = "Amsterdam";

console.log(person);

Выводит: {name: "Valera", age: 23}  
это выводятся свойства объекта Персона.