Вопросы 💎

1. Чем отличается класс от объекта? Приведите примеры и того, и другого, отличные от указанных в уроке.

Объект (object) — это набор свойств, каждое из которых состоит из названия и значения.

А класс – это только схема (шаблон) создания и работы какого-то объекта. То есть класс определяет все свойства, которые характеризуют группу объектов, описывает множество всех объектов (экземпляров данного класса).

Например, класс «животные из красной книги», может описывать множество всех животных, внесенных в красную книгу (тип, класс, отряд, семейство, род, вид, популяция, место обитания и т.д.). А «снежный барс» может быть объектом (экземпляром класса «животные из красной книги»), представляющим собой конкретное животное из книги. И иметь ровно столько же свойств, сколько и родительский класс (не больше и не меньше).

2. Приведите примеры объектно-ориентированного подхода в реальной жизни.

Возможный вариант ответа: например, библиотека могла бы содержать классы Книга (название, автор, год выпуска, количество страниц), Сотрудники (ФИО, должность, зарплата, дата начала работы) и Читатели (ФИО, контакты, номер читательского билета).

Примеры объектно-ориентированного подхода в реальной жизни:
Таксопарк: классы Машины (марки, модели, пробег, класс, цвета и пр.), Водители (ФИО, стаж, данные водительского удостоверения), Пассажиры (ФИО, контакты, история поездок).

Арбитражный суд Москвы мог бы содержать классы Су`дьи (ФИО, стаж, подразделение, образование, кабинет), Помощники судей с аналогичными свойствами, Судебные приставы (ФИО, иные данные, стаж, разрешение на оружие), Дела (№ дела, стороны, судья, инстанция, суть спора, стадия рассмотрения).

Приведите примеры объектно-ориентированного подхода в программировании.

Возможный вариант ответа: например, товары в интернет магазине (название товара, тип, цена), комментарии в соцсети (автор комментария, дата, текст комментария), видео в Youtube (название видео, его адрес, автор и дата загрузки).

Примеры объектно-ориентированного подхода в программировании:
Список работников на портале организации (ФИО, наименование отдела, должность, стаж, контактные данные, адрес офиса).
Форма обратной связи на сайте (заявитель, его контактные данные, тема обращения, суть запроса, дата запроса).
Театральная афиша на сайте (название спектакля, длительность, дата и время, название театра, краткое описание).

3. Что такое конструктор? Самостоятельно изучите и напишите, какие бывают виды конструкторов.

Конструктор — это специальный метод, который вызывается при создании объекта. То есть это функция, которая используется, чтобы создавать однотипные объекты. Когда создаётся

новый объект с помощью оператора new (экземпляр класса), в этот момент вызывается constructor.
По соглашениям конструкторы вызывают с помощью ключевого слова new, а также называют с большой буквы, причём обычно не глаголом, а существительным.
Существуют конструкторы - явные и неявные (по умолчанию). То есть, при создании экземпляра класса без явного указания метода-конструктора, его поля будут проинициализированы значениями по умолчанию. Также можно создать экземпляр с конструктором без аргументов. Конструктор, который будет принимать аргументы, называется параметризованный конструктор (parameterized constructor).

Также существуют конструкторы копирования (copy constructor) - это специальный конструктор, который принимает в качестве аргумента экземпляр того же класса для создания нового объекта на основе переданного. Такие конструкторы применяются тогда, когда необходимо создать копию сложного объекта, но при этом мы не хотим использовать метод clone().

4. Что выведет код? Почему именно так?

```
function bike() {
    console.log(this.name)
}

var name = 'ninja'
var obj1 = {name: "pomidor", bike: bike}
var obj2 = {name: "site", bike: bike}

bike()              //ninja
obj1.bike()         //pomidor
obj2.bike()         //site
// this не является фиксированным, и может использоваться в любой функции, не обязательно в конструкторе. В данном случае значение this внутри функции bike() вычисляется во время выполнения кода и зависит от контекста. То есть: bike() назначена двум разным объектам (obj1 и obj2) и одной переменной (name), поэтому функция имеет различное значение «this» при вызовах.
```

5. Чем статические свойства и методы отличаются от нестатических? В каких ситуациях они применяются?

Статические методы применяются к самому классу, а не к отдельным объектам. В отличие от обычных нестатических методов, которые определяют поведение объекта, статические методы определяют поведение для всего класса. Поэтому для их вызова применяется имя класса, а не имя объекта.

Статические методы нельзя переопределить, но при этом они будут общими для всех экземпляров класса, поэтому часто статические методы для этого и используются, когда нужно не привязывать метод к каждому объекту индивидуально, а применить сразу ко всем, часто это бывает нужно в расчетах, например.

Статические методы также используются в классах, относящихся к базам данных, для поиска/сохранения/удаления вхождений в базу данных.

6. Самостоятельно изучите, что такое геттеры и сеттеры, приведите пример класса с их использованием.

Геттеры и сеттеры относятся к *свойствам-аксессорам (accessor properties)*. По своей сути это функции, которые используются для присвоения и получения значения, но во внешнем коде они выглядят как обычные свойства объекта.

Геттер - функция чтения свойства (используется для получения значения свойства, возвращает значение или undefined).
Сеттер - функция записи свойства (используется для установки значения свойства. Принимает единственным аргументом новое значение, присваиваемое свойству, и записывает его.
Обозначаются как get и set:

```
let obj = {
    get propName() {
        // геттер, срабатывает при чтении obj.propName
    },
    set propName(value) {
        // сеттер, срабатывает при записи obj.propName = value
    }
};
```

Это дает возможность снабдить такие методы дополнительными обработками. Например, сеттер при записи значения в поле объекта, может проверить тип, или входит ли значение в диапазон допустимых (валидация), модифицировать объект (например, дату привести в нужный формат). В геттер же можно добавить кэширование, если актуальное значение на самом деле лежит в базе данных.

Например, есть объект user со свойствами name и surname:

```
let user = {
    name: "Cersei",
    surname: "Lannister"
};
```

Добавление свойства объекта fullName для полного имени при помощи аксессора get:

```
let user = {
    name: "Cersei",
    surname: "Lannister",
    get fullName() {
        return `${this.name} ${this.surname}`;
    }
};
alert(user.fullName); // Cersei Lannister
```

Снаружи свойство-аксессор выглядит как обычное свойство. Нет стандартного вызова user.fullName как функции.
Чтобы получить значение FullName, нужно добавить Setter:

```
let user = {
    name: "Cersei",
    surname: "Lannister",
    get fullName() {
        return `${this.name} ${this.surname}`;
    },
    set fullName(value) {
        [this.name, this.surname] = value.split(" ");
    }
```

```
};

// set fullName запустится с данным значением
user.fullName = "Lena Headey";
alert(user.name);       // Lena
alert(user.surname);    // Headey
```

7. Что выведет код? Почему именно так?

```
class Person {
    constructor(name) {
    this.name = name;
    }
}
const member = new Person("John")
console.log(typeof member)                //object
// Person - класс, member - объект, который был создан через конструктор на основе существующего класса Person с параметром (name="John"). Поэтому выводится тип объекта member - object
```

8. Что выведет код? Почему именно так?

```
const person = {
    name: "Valera",
    age: 23
}
let city = person.city
city = "Amsterdam"
console.log(person) // {name: 'Valera', age: 23}
                    // age: 23
                    // name: "Valera"

// свойство city со значением "Amsterdam" не попало в объект, так как запись неверная. Верно так:
const person = {
    name: "Valera",
    age: 23
}
person.city = "Amsterdam"
console.log(person)
```