1.

Когда мы кликаем на элемент, событие идёт по дереву DOM сначала сверху вниз, потом до элемента, на котором произошло действие, а потом обратно вверх. Это три фазы:

Capturing – событие идёт сверху вниз.

Target – событие достигает того элемента, на котором кликнули.

Bubbling – событие поднимается обратно вверх.

На практике это удобно, например, если навесить обработчик на родителя, чтобы не ставить его на каждый дочерний элемент. А если нужно, чтобы событие не дошло до родителя – можно вызвать `stopPropagation()`

2.

Promise – это такой объект, который говорит: Я что-то делаю и когда закончу, дам результат. Он может быть в трёх состояниях: ожидается, успешно выполнен или ошибка.

Пример: я делаю запрос на сервер, и когда данные приходят – обрабатываю их через .then(), если ошибка – через .catch().

Можно ещё использовать async/await – это как писать обычный код, только асинхронный.

Event Loop – это как очередь задач. JS не ждёт, пока всё выполнится, а ставит асинхронные вещи в очередь, чтобы браузер не зависал.

3.

ООП – это когда мы строим программу вокруг объектов, которые хранят данные и умеют что-то делать. Представим, что объект – это как реальная вещь, например машина: у неё есть свойства (цвет, марка, скорость) и методы (ехать, тормозить).

Основные принципы:

1. Инкапсуляция

   Это когда мы скрываем внутренние детали объекта, чтобы кто-то случайно их не сломал.

   В JS это можно делать через приватные свойства (#privateProp) или просто через соглашение _property.

   Пример: у объекта Car есть приватное свойство #engineStatus, а снаружи мы можем только запускать метод startEngine().

2. Наследование

   Это когда один объект берёт свойства и методы другого и может добавлять свои.

   В JS это делается через class и extends.

   Пример: есть класс Animal, у него метод makeSound(). Мы создаём класс Dog и наследуем Animal, но makeSound() переопределяем: «Woof!».

3. Полиморфизм

   Это когда один метод может работать по-разному у разных объектов.

   Пример: у нас есть метод render() у разных компонентов Angular – у одного он выводит текст, у другого – картинку, но вызываем его одинаково.

4. Абстракция

   Это когда мы отделяем что делает объект от как он это делает.

   В JS через TypeScript можно использовать интерфейсы или абстрактные классы.

   Пример: интерфейс IShape с методом draw(), а у Circle и Rectangle своя реализация draw().

Почему это важно в Angular:

- Компоненты, сервисы и модели – это все объекты.
- Наследование и полиморфизм помогают переиспользовать код, например, создавать базовый сервис с общими методами для всех API-запросов.
- Инкапсуляция помогает, чтобы внутренние данные компонента не ломали другие части приложения.

4.

Когда вводишь URL:

1. Разбор URL

   Браузер смотрит: какой протокол (http/https), домен, путь, параметры и якорь.

   Например: https://example.com/page?id=1#section → протокол https, домен example.com, путь /page, параметр id=1, якорь #section.

2. DNS lookup

   Браузер узнаёт IP сервера через DNS. Это как телефонная книга: «кто владеет example.com».

3. Установка соединения

   Через TCP и TLS (если HTTPS) браузер соединяется с сервером.

4. Отправка HTTP-запроса

   Браузер спрашивает сервер: «Дай мне страницу».

   Сервер отвечает HTML, а вместе с ним CSS, JS и другие ресурсы.

5. Разбор и рендеринг

   HTML → DOM: строится дерево элементов.

   CSS → CSSOM: строится дерево стилей.

   Render tree: DOM + CSSOM → браузер понимает, как рисовать элементы.

   Layout → Paint → Composite: браузер рисует страницу на экран.

6. Выполнение JS

   Скрипты могут изменять DOM, добавлять интерактивность, загружать данные через fetch или Angular сервисы.

Как ускорить загрузку:

Кэш браузера и CDN – чтобы файлы приходили быстрее.

Lazy loading и code splitting – грузим только то, что нужно сразу.

Сжатие файлов (gzip/brotli) – чтобы меньше данных передавалось.

HTTP/2 и HTTP/3 – позволяют параллельно загружать несколько ресурсов.

Возможные проблемы:

CORS – нельзя просто так запросить данные с чужого сайта без разрешения.

Безопасность – XSS, CSRF, MITM. Нужно HTTPS и проверка данных.

Недоступность ресурсов – если сервер упал или заблокирован, страница может не загрузиться.