* **HTML/CSS**

#### **Flexbox와 Grid Layout의 차이점은 무엇인가요?**

**답변:** Flexbox는 1차원 레이아웃 모델로, 주로 한 방향(행 또는 열)으로 아이템을 배치하는 데 사용됩니다. 반면 Grid Layout은 2차원 레이아웃 모델로, 행과 열 모두를 이용하여 복잡한 레이아웃을 구성할 수 있습니다.

#### **CSS 그리드(Grid)를 사용하여 복잡한 레이아웃을 만드는 방법에 대해 설명해주세요.**

**답변:** CSS Grid를 사용하여 복잡한 레이아웃을 만드는 방법은 다음과 같습니다:

1. display: grid; 속성을 부모 요소에 설정합니다.
2. grid-template-columns와 grid-template-rows 속성을 사용하여 행과 열의 크기를 정의합니다.
3. grid-template-areas 속성을 사용하여 그리드 영역을 정의하고, 자식 요소에 grid-area 속성을 설정하여 배치합니다.

#### **미디어 쿼리(Media Queries)를 사용하여 반응형 웹 디자인을 구현한 경험이 있나요?**

**답변:** 네, 미디어 쿼리를 사용하여 다양한 화면 크기에 맞추어 반응형 웹 디자인을 구현한 경험이 있습니다. 주로 @media 규칙을 사용하여 화면 너비, 높이, 해상도에 따라 다른 스타일을 적용합니다.

#### **모바일 우선 디자인(Mobile-First Design)의 이점에 대해 설명해보세요.**

**답변:** 모바일 우선 디자인은 작은 화면부터 디자인을 시작하여 점차 큰 화면으로 확장하는 접근 방식입니다. 이 접근 방식은 성능 최적화에 도움이 되며, 모바일 사용자에게 우선적으로 최적화된 경험을 제공합니다.

#### **SASS 또는 LESS와 같은 CSS 전처리기를 사용한 경험이 있나요? 이들의 주요 기능에 대해서 언급해주세요.**

**답변:** 네, SASS와 LESS를 사용한 경험이 있습니다. 주요 기능으로는 변수, 중첩 규칙, 믹스인, 함수 등을 통해 CSS를 더욱 효율적이고 유지보수하기 쉽게 작성할 수 있습니다.

#### px, em, rem 의 차이에 대해 설명해주세요.

**답변:**

* px: 절대 단위로, 고정된 크기를 의미합니다.
* em: 상대 단위로, 부모 요소의 글꼴 크기를 기준으로 합니다.
* rem: 상대 단위로, 루트 요소(html)의 글꼴 크기를 기준으로 합니다.

#### **반응형 웹을 구현할 때는 이 단위 중에서 어떤 단위를 주로 사용하시나요?**

**답변:** 반응형 웹에서는 주로 rem과 em을 사용하여 요소의 크기를 설정합니다. 이는 화면 크기에 따라 글꼴 크기와 요소 크기를 유연하게 조절할 수 있기 때문입니다.

#### **vw, vh에 대해 설명해주세요.**

**답변:** vw와 vh는 각각 뷰포트의 너비와 높이를 기준으로 하는 상대 단위입니다. 1vw는 뷰포트 너비의 1%, 1vh는 뷰포트 높이의 1%를 의미합니다.

#### **반응형 브레이크 포인트는 보통 어떻게 잡으시나요?**

**답변:** 일반적으로 반응형 브레이크 포인트는 주요 디바이스 크기를 기준으로 설정합니다. 예를 들어, 모바일 디바이스는 576px 이하, 태블릿은 768px 이하, 데스크탑은 992px 이하로 설정할 수 있습니다.

#### **CSS 선택자의 우선순위에 대해 설명해주세요.**

**답변:** CSS 선택자의 우선순위는 다음과 같이 결정됩니다:

1. 인라인 스타일 (1000점)
2. ID 선택자 (100점)
3. 클래스, 속성, 가상 클래스 선택자 (10점)
4. 태그, 가상 요소 선택자 (1점)
5. 중요도(!important)가 설정된 스타일은 최우선 적용됩니다.

#### **페이지 크기가 변해도 항상 같은 비율을 유지하는 요소를 만들려면 CSS를 어떻게 설정해야될까요?**

**답변:** 페이지 크기가 변해도 같은 비율을 유지하려면 aspect-ratio 속성을 사용합니다. 예를 들어, aspect-ratio: 16 / 9;로 설정하여 16:9 비율을 유지할 수 있습니다.

#### **Flexbox에 대해 설명해주세요.**

**답변:** Flexbox는 1차원 레이아웃 모델로, 행 또는 열 방향으로 아이템을 배치하고 정렬할 수 있습니다. display: flex;를 부모 요소에 설정하고, justify-content, align-items, flex-direction 등의 속성을 사용하여 레이아웃을 조정합니다.

#### **float의 동작에 대해 설명해주세요.**

**답변:** float 속성은 요소를 왼쪽이나 오른쪽으로 떠오르게 하여 다른 텍스트나 요소들이 그 주변을 감싸도록 합니다. clear 속성을 사용하여 float 요소의 영향을 받지 않도록 할 수 있습니다.

#### **CSS에서 Cascading에 대해 설명해주세요.**

**답변:** Cascading은 여러 스타일 규칙이 동일한 요소에 적용될 때 우선순위에 따라 어떤 스타일이 적용될지를 결정하는 방식을 의미합니다. 우선순위는 인라인 스타일, ID 선택자, 클래스 선택자, 태그 선택자 순으로 결정됩니다.

#### **⭐ SCSS에 대해 설명해주세요.**

**답변:** SCSS는 SASS의 확장 문법으로, 기존 CSS 문법과 호환됩니다. SCSS는 중첩 규칙, 변수, 믹스인, 상속 등의 기능을 제공하여 CSS를 더 효율적으로 작성할 수 있게 해줍니다.

#### **⭐ position 속성에 대해 설명해주세요.**

**답변:** position 속성은 요소의 위치를 지정하는 데 사용됩니다. 주요 값으로는 static, relative, absolute, fixed, sticky가 있으며, 각 값에 따라 요소의 위치가 다르게 설정됩니다.

#### **margin과 padding에 대해 설명해주세요.**

**답변:** margin은 요소의 외부 여백을 설정하여 다른 요소와의 간격을 조절하고, padding은 요소의 내부 여백을 설정하여 콘텐츠와 요소 경계 사이의 간격을 조절합니다.

#### **HTML 렌더링 도중 JavaScript가 실행되면 렌더링이 멈추는 이유가 뭔가요?**

**답변:** JavaScript는 기본적으로 동기적으로 실행되기 때문에, HTML 렌더링 도중 JavaScript가 실행되면 렌더링이 멈추고 스크립트가 완료될 때까지 대기합니다. 이를 해결하기 위해 async 또는 defer 속성을 사용하여 비동기적으로 스크립트를 로드할 수 있습니다.

#### **Box model에 대해 설명해주세요.**

**답변:** Box model은 요소의 레이아웃을 정의하는 모델로, 요소는 content, padding, border, margin으로 구성됩니다. 이 모델을 통해 요소의 크기와 여백을 설정할 수 있습니다.

#### **Attribute와 Property의 차이에 대해 설명해주세요.**

**답변:** Attribute는 HTML 태그 내에 정의된 속성으로, HTML 문서가 처음 로드될 때 설정됩니다. Property는 JavaScript를 통해 접근할 수 있는 DOM 객체의 속성으로, 동적으로 변경할 수 있습니다.

#### **⭐ display 속성에 어떤 것들이 있는지 설명해 주세요.**

**답변:** display 속성은 요소의 레이아웃 방식과 표시 방식을 정의합니다. 주요 값으로는 block, inline, inline-block, flex, grid, none 등이 있습니다. 각 값은 요소의 표시와 배치에 영향을 줍니다.

#### **CSS 애니메이션과 JS 애니메이션의 차이에 대해서 설명해 주세요.**

**답변:** CSS 애니메이션은 주로 간단한 애니메이션 효과에 사용되며, @keyframes와 animation 속성을 사용하여 정의합니다. JS 애니메이션은 복잡한 애니메이션과 상호작용을 구현하는 데 적합하며, JavaScript를 사용하여 애니메이션을 제어할 수 있습니다.

#### **CSS in JS의 장단점에 대해 설명해주세요.**

**답변:**

* 장점: 스타일을 컴포넌트 단위로 관리할 수 있으며, 동적인 스타일을 적용하기 쉽습니다.
* 단점: 성능 저하의 우려가 있으며, CSS와 JS의 분리가 불명확해질 수 있습니다.

#### **⭐ 시맨틱 마크업에 대해 설명해주세요.**

**답변:** 시맨틱 마크업은 HTML 태그를 의미에 맞게 사용하는 것을 말합니다. 예를 들어, 제목은 <h1>~<h6>, 단락은 <p>, 섹션은 <section>, 내비게이션은 <nav> 등을 사용합니다. 이는 접근성 향상과 검색 엔진 최적화에 도움이 됩니다.

#### **HTML5의 태그에 대해 설명해주세요.**

**답변:** HTML5는 새로운 시맨틱 요소를 도입하여 문서 구조를 더 명확히 정의할 수 있게 했습니다. 주요 태그로는 <article>, <section>, <nav>, <aside>, <header>, <footer>, <main>, <figure>, <figcaption> 등이 있습니다.

#### **script 태그에서 Async와 Defer의 차이에 대해 설명해주세요.**

**답변:** async는 스크립트를 비동기적으로 로드하며, 로드가 완료되면 바로 실행합니다. defer는 스크립트를 비동기적으로 로드하지만, HTML 파싱이 완료된 후에 실행합니다.

#### **가상 클래스에 대해 설명해주세요.**

**답변:** 가상 클래스는 특정 상태나 동작에 따라 요소에 스타일을 적용하는 클래스입니다. 주요 가상 클래스로는 :hover, :focus, :active, :visited, :nth-child, :nth-of-type 등이 있습니다.

#### **margin 병합에 대해 설명해주세요.**

**답변:** margin 병합은 두 개 이상의 인접한 요소의 margin 값이 합쳐져 하나의 margin 값으로 계산되는 현상입니다. 상하 방향의 margin이 병합되며, 가장 큰 margin 값이 적용됩니다.

* **Javascript**

#### **자바스크립트는 무슨 언어인가요?**

**답변:** 자바스크립트는 웹 개발에서 주로 사용되는 스크립트 언어입니다. 동적이고 인터프리터 방식의 언어로, 브라우저에서 실행되며, 클라이언트와 서버 측 개발 모두에 사용됩니다. 자바스크립트는 객체 지향, 함수형, 그리고 이벤트 기반 프로그래밍을 지원합니다.

#### **변수 선언, 초기화, 할당의 차이점에 대해 설명해주세요.**

**답변:**

* **변수 선언:** 변수의 이름을 선언하고 메모리 공간을 확보하는 과정입니다. 예를 들어, var x;.
* **초기화:** 변수 선언 시 초기 값을 설정하는 것입니다. 예를 들어, var x = 5;.
* **할당:** 변수에 값을 할당하는 것으로, 선언과 초기화 후에 값을 변경하는 경우를 말합니다. 예를 들어, x = 10;.

#### **데이터 타입에 대해 설명해주세요.**

**답변:** 자바스크립트의 데이터 타입은 크게 원시 타입과 객체 타입으로 나뉩니다.

* **원시 타입:** 숫자(Number), 문자열(String), 불리언(Boolean), null, undefined, 심볼(Symbol), 빅인트(BigInt).
* **객체 타입:** 배열(Array), 함수(Function), 날짜(Date), 정규 표현식(RegExp) 등을 포함하는 객체(Object).

#### **생성자에 대해 설명해주세요.**

**답변:** 생성자는 객체를 생성하고 초기화하는 데 사용되는 함수입니다. new 키워드를 사용하여 생성자를 호출하면, 새로운 객체가 생성되고, 생성자 함수 내부의 this가 해당 객체를 가리키며 초기화 작업이 수행됩니다. 예를 들어, function Person(name) { this.name = name; }.

#### **⭐ this에 대해 설명해주세요.**

**답변:** this 키워드는 실행 컨텍스트에 따라 다른 객체를 참조합니다. 일반적으로 함수 내부에서 this는 해당 함수를 호출한 객체를 가리킵니다. 그러나 함수의 호출 방식에 따라 this의 값이 달라질 수 있습니다.

#### **call, apply, bind에 대해 설명해주세요.**

**답변:**

* **call:** 함수를 호출하면서 this와 인수를 전달합니다. function.call(thisArg, arg1, arg2, ...).
* **apply:** 함수를 호출하면서 this와 인수를 배열 형태로 전달합니다. function.apply(thisArg, [argsArray]).
* **bind:** this와 인수를 설정하여 새로운 함수를 반환합니다. 나중에 호출할 때 사용됩니다. function.bind(thisArg, arg1, arg2, ...).

#### **⭐ 콜백 함수에 대해 설명해주세요.**

**답변:** 콜백 함수는 다른 함수의 인수로 전달되어, 특정 작업이 완료된 후 호출되는 함수입니다. 주로 비동기 작업에서 사용되며, 이벤트 처리, 타이머, 네트워크 요청 등에 활용됩니다.

#### **⭐ 콜백 지옥을 해결하는 방법을 설명해주세요.**

**답변:** 콜백 지옥을 해결하기 위해 다음 방법을 사용할 수 있습니다:

1. **Promise:** 비동기 작업을 체인 형태로 작성할 수 있어 가독성이 높아집니다.
2. **Async/Await:** 비동기 코드를 동기식 코드처럼 작성할 수 있어 가독성과 유지보수성이 향상됩니다.

#### **⭐ Promise에 대해 설명해주세요.**

**답변:** Promise는 비동기 작업의 완료 또는 실패를 나타내는 객체입니다. then과 catch 메서드를 사용하여 성공과 실패를 처리할 수 있습니다. 예를 들어, new Promise((resolve, reject) => { /\* 비동기 작업 \*/ }).

#### **Promise.all() 에 대해 설명해주세요.**

**답변:** Promise.all()은 여러 개의 Promise를 병렬로 실행하고, 모두 완료될 때까지 기다린 후 결과를 배열로 반환합니다. 모든 Promise가 성공해야 결과를 반환하며, 하나라도 실패하면 catch로 처리됩니다.

#### **⭐ Promise와 Callback를 비교 설명해주세요.**

**답변:** Promise는 콜백 함수보다 가독성이 좋고, 비동기 작업의 체인 처리를 통해 코드의 깊이를 줄여줍니다. 또한, then, catch, finally 메서드를 통해 에러 처리와 후속 작업을 명확히 할 수 있습니다.

#### **⭐ Async, Await이 뭔지 그리고 사용 방법을 설명해주세요.**

**답변:** async와 await는 비동기 코드를 동기식 코드처럼 작성할 수 있게 해줍니다. async 키워드를 함수 앞에 붙여 해당 함수가 Promise를 반환하도록 하고, await 키워드를 사용하여 Promise가 해결될 때까지 기다립니다. 예를 들어, const result = await asyncFunction();.

#### **⭐ Promise와 Async, Await의 차이를 설명해주세요.**

**답변:** Promise는 비동기 작업의 성공과 실패를 처리하기 위한 객체이고, then과 catch 메서드를 사용하여 후속 작업을 연결합니다. Async/Await는 Promise를 기반으로 하며, 비동기 코드를 동기식 코드처럼 작성할 수 있게 해줍니다. 이는 가독성과 유지보수성을 높여줍니다.

#### **AJAX에 대해 설명해주세요.**

**답변:** AJAX는 비동기적으로 서버와 통신하여 페이지를 새로 고침하지 않고 데이터를 가져오거나 서버에 데이터를 보낼 수 있게 해주는 기술입니다. 주로 XMLHttpRequest 객체나 Fetch API를 사용합니다.

#### **⭐ var, let, const 차이를 설명해주세요.**

**답변:**

* **var:** 함수 스코프를 가지며, 중복 선언이 가능합니다. 호이스팅에 의해 선언이 최상단으로 끌어올려집니다.
* **let:** 블록 스코프를 가지며, 중복 선언이 불가능합니다. TDZ(Temporal Dead Zone)에 의해 선언 전에 접근할 수 없습니다.
* **const:** 블록 스코프를 가지며, 중복 선언이 불가능하고, 선언 시 반드시 초기화해야 합니다. 값이 재할당되지 않습니다.

#### **TDZ에 대해 설명해주세요**.

**답변:** TDZ(Temporal Dead Zone)는 변수가 선언되기 전까지 변수를 참조할 수 없는 구역을 의미합니다. let과 const로 선언된 변수는 TDZ에 영향을 받으며, 선언 전에 접근하려고 하면 ReferenceError가 발생합니다.

#### **함수 선언형과 함수 표현식의 차이에 대해 설명해주세요**.

**답변:**

* **함수 선언형:** function 함수명() {} 형태로 선언되며, 호이스팅에 의해 코드 상단으로 끌어올려집니다.
* **함수 표현식:** const 함수명 = function() {}; 형태로 선언되며, 호이스팅되지 않으며, 선언 후에만 사용할 수 있습니다.

#### **⭐ 이벤트 버블링과 캡처링에 대해 설명해주세요.**

**답변:**

* **이벤트 버블링:** 이벤트가 발생한 요소에서 상위 요소로 전파되는 현상입니다.
* **이벤트 캡처링:** 이벤트가 최상위 요소에서 발생하여 이벤트가 발생한 요소까지 전파되는 현상입니다. 이벤트 리스너를 addEventListener의 세 번째 인수로 true를 전달하면 캡처링 단계에서 실행됩니다.

#### **⭐ 이벤트 위임에 대해서 설명해주세요.**

**답변:** 이벤트 위임은 상위 요소에 이벤트 리스너를 설정하여, 하위 요소에서 발생한 이벤트를 처리하는 방법입니다. 이는 동적으로 생성된 요소에도 이벤트를 적용할 수 있어 효율적입니다.

#### **이벤트 위임의 동작 방식에 대해서 설명해주세요.**

**답변:** 이벤트 위임은 이벤트 버블링을 활용하여 상위 요소에 이벤트 리스너를 설정합니다. 이벤트가 발생하면 버블링을 통해 상위 요소로 전파되고, 상위 요소에서 이벤트를 처리합니다.

#### **⭐ 호이스팅과 발생하는 이유에 대해 설명해주세요.**

**답변:** 호이스팅은 변수와 함수 선언이 코드의 최상단으로 끌어올려지는 현상입니다. 이는 자바스크립트의 실행 컨텍스트 생성 단계에서 변수와 함수의 메모리가 미리 할당되기 때문에 발생합니다.

#### **스코프 (Scope)에 대해 설명해주세요.**

**답변:** 스코프는 변수나 함수가 유효한 범위를 의미합니다. 자바스크립트에서는 전역 스코프, 함수 스코프, 블록 스코프가 있으며, 스코프 체인을 통해 상위 스코프의 변수에 접근할 수 있습니다.

#### **스코프 체인에 대해 설명해주세요.**

**답변:** 스코프 체인은 현재 스코프에서 변수를 찾지 못하면 상위 스코프로 이동하여 변수를 찾는 메커니즘입니다. 전역 스코프까지 이동하며, 변수의 유효 범위를 결정합니다.

#### **⭐ 클로져(Closure)에 대해 설명해주세요.**

**답변:** 클로저는 함수가 선언될 때의 렉시컬 환경을 기억하여, 함수 외부에서 선언된 변수에 접근할 수 있는 기능을 의미합니다. 이는 데이터 은닉과 상태 유지를 위해 사용됩니다.

#### **실행 컨텍스트에 대해 설명해주세요.**

**답변:** 실행 컨텍스트는 함수 실행 시 생성되는 환경으로, 변수, 객체, 함수 선언과 같은 정보를 포함합니다. 콜 스택에 쌓이며, 함수 실행이 끝나면 제거됩니다.

#### **렉시컬 환경(Lexical Environment)에 대해 설명해주세요.**

**답변:** 렉시컬 환경은 변수와 함수 선언 정보를 저장하는 구조로, 실행 컨텍스트의 일부입니다. 현재 스코프와 외부 환경 참조를 포함하며, 클로저를 구현하는 데 사용됩니다.

#### **자바스크립트에서 일어나는 데이터 형변환에 대해 설명해주세요**.

**답변:** 자바스크립트에서는 명시적 형변환과 암시적 형변환이 발생합니다. 명시적 형변환은 개발자가 직접 변환하는 것이며, 암시적 형변환은 자바스크립트 엔진이 자동으로 변환하는 것입니다. 예를 들어, Number("123")은 명시적 형변환이고, 123 + "456"은 암시적 형변환입니다.

#### **자바스크립트가 동적 언어인 이유는 무엇인가요?**

**답변:** 자바스크립트는 동적 언어로, 런타임에 변수의 타입을 확인하고, 타입을 변경할 수 있습니다. 이는 변수의 선언 시 타입을 지정하지 않기 때문입니다.

#### **프로토타입에 대해 설명해주세요.**

**답변:** 프로토타입은 객체 지향 프로그래밍에서 상속을 구현하는 방법입니다. 자바스크립트에서는 객체가 프로토타입을 통해 다른 객체의 속성과 메서드를 상속받습니다. 모든 객체는 [[Prototype]] 링크를 가지며, 프로토타입 체인을 통해 상위 객체의 속성에 접근할 수 있습니다.

#### **깊은 복사와 얕은 복사에 대해 설명해주세요.**

**답변:**

* **깊은 복사:** 객체의 모든 속성을 재귀적으로 복사하여 완전히 새로운 객체를 생성합니다. 예를 들어, JSON.parse(JSON.stringify(obj)).
* **얕은 복사:** 객체의 1차원 속성만 복사하고, 중첩 객체는 참조를 복사합니다. 예를 들어, Object.assign({}, obj).

#### **불변성을 유지하려면 어떻게 해야하나요?**

**답변:** 불변성을 유지하려면 객체나 배열을 직접 변경하지 않고, 새로운 객체나 배열을 생성하여 변경 사항을 반영합니다. 예를 들어, 스프레드 연산자를 사용하여 새로운 객체를 생성합니다. const newObj = {...obj, newProp: value};.

#### **Blocking과 Non-Blocking에 대해 설명해주세요.**

**답변:**

* **Blocking:** 코드가 순차적으로 실행되며, 이전 작업이 완료될 때까지 다음 작업이 대기하는 방식입니다.
* **Non-Blocking:** 코드가 비동기적으로 실행되며, 이전 작업의 완료 여부와 상관없이 다음 작업이 실행되는 방식입니다. 예를 들어, 콜백 함수, Promise, Async/Await 등을 사용합니다.

#### **⭐ 동기와 비동기에 대해 설명해주세요.**

**답변:**

* **동기:** 작업이 순차적으로 실행되며, 이전 작업이 완료될 때까지 다음 작업이 대기합니다.
* **비동기:** 작업이 병렬로 실행되며, 이전 작업의 완료 여부와 상관없이 다음 작업이 실행됩니다. 비동기 작업은 주로 콜백 함수, Promise, Async/Await를 통해 처리됩니다.

#### **ES6에서 새로 생긴 기능을 아는대로 말씀해주세요.**

**답변:** ES6(ECMAScript 2015)에서 도입된 주요 기능은 다음과 같습니다:

* let과 const 키워드
* 화살표 함수
* 클래스
* 템플릿 리터럴
* 디스트럭처링 할당
* 기본 함수 매개변수
* 스프레드 연산자와 Rest 연산자
* Promise
* 모듈(Import/Export)
* Map과 Set
* 심볼(Symbol)

#### **undefined, null, undeclared를 비교해주세요.**

**답변:**

* **undefined:** 변수가 선언되었으나 초기화되지 않은 상태입니다.
* **null:** 변수가 명시적으로 "값이 없음"을 나타냅니다.
* **undeclared:** 변수가 선언되지 않은 상태입니다. 해당 변수를 참조하면 ReferenceError가 발생합니다.

#### **자바스크립트를 멀티 쓰레드처럼 사용하는 방법이 뭔가요? / 자바스크립트에서 비동기 로직이 어떻게 동작하는지 설명해주세요. (이벤트 루프)**

**답변:** 자바스크립트는 싱글 쓰레드 언어이지만, Web Workers를 사용하여 멀티 쓰레드처럼 사용할 수 있습니다. 비동기 로직은 이벤트 루프를 통해 동작합니다. 이벤트 루프는 콜 스택과 태스크 큐를 관리하여, 콜 스택이 비어 있을 때 태스크 큐의 작업을 실행합니다.

#### **태스크 큐와 마이크로 태스크 큐에는 어떤 함수가 들어가나요?**

**답변:**

* **태스크 큐:** setTimeout, setInterval, I/O 작업 등의 콜백 함수가 들어갑니다.
* **마이크로 태스크 큐:** Promise의 then 콜백, MutationObserver의 콜백 등이 들어갑니다.

#### requestAnimationFrame에 대해 설명해주세요.

**답변:** requestAnimationFrame은 브라우저의 다음 리페인트가 되기 전에 호출할 콜백을 등록합니다. 이는 애니메이션을 효율적으로 구현할 수 있게 해주며, 60fps로 최적화된 애니메이션을 구현할 수 있습니다.

#### **비동기적으로 실행되는 것을 동기적으로 코딩하는 방법이 있나요?**

**답변:** 비동기 작업을 동기식 코드처럼 작성하려면 async와 await를 사용합니다. async 키워드를 함수 앞에 붙이고, await를 사용하여 Promise가 해결될 때까지 기다립니다.

#### **map과 forEach, reduce에 대해 설명해주세요.**

**답변:**

* **map:** 배열의 각 요소에 대해 주어진 함수를 호출하고, 결과를 새로운 배열로 반환합니다. 원본 배열은 변경되지 않습니다.
* **forEach:** 배열의 각 요소에 대해 주어진 함수를 호출하지만, 반환값이 없습니다. 주로 반복 작업에 사용됩니다.
* **reduce:** 배열의 각 요소에 대해 주어진 함수를 호출하고, 하나의 누적된 결과값을 반환합니다. 주로 배열의 합계, 평균 등을 구할 때 사용됩니다.

#### **자바스크립트의 메모리 관리에 대해 아는 대로 설명해주세요.**

**답변:** 자바스크립트의 메모리 관리는 가비지 컬렉션을 통해 이루어집니다. 사용하지 않는 객체는 메모리에서 자동으로 해제됩니다. 참조 카운팅과 마크 앤 스윕 알고리즘이 사용됩니다.

#### **클래스에 대해 설명해주세요.**

**답변:** 클래스는 객체 지향 프로그래밍의 개념을 구현하기 위한 템플릿입니다. ES6에서 도입되었으며, constructor 메서드와 함께 사용됩니다. 상속을 통해 클래스 간의 관계를 정의할 수 있습니다.

#### **즉시 실행 함수 (IIFE)에 대해 설명해주세요.**

**답변:** 즉시 실행 함수 표현식(IIFE)은 정의되자마자 즉시 실행되는 함수입니다. 주로 전역 네임스페이스 오염을 방지하기 위해 사용됩니다. 예를 들어, (function() { /\* 코드 \*/ })();.

#### **엄격 모드에 대해 설명해주세요.**

**답변:** 엄격 모드("use strict";)는 자바스크립트의 오류를 더 빨리 발견할 수 있게 해주는 기능입니다. 엄격 모드에서는 암시적 전역 변수 선언이 금지되며, 안전하지 않은 행동들이 제한됩니다.

#### **콜 스택 (Call Stack)과 힙 (Heap)에 대해 설명해주세요.**

**답변:**

* **콜 스택:** 함수 호출이 쌓이는 자료 구조로, 현재 실행 중인 함수의 상태를 저장합니다. 함수 실행이 끝나면 스택에서 제거됩니다.
* **힙:** 동적으로 할당된 메모리 공간으로, 객체와 함수의 메모리가 저장됩니다. 가비지 컬렉션에 의해 관리됩니다.

#### **Rest 연산자와 Spread 연산자에 대해 설명해주세요.**

**답변:**

* **Rest 연산자:** 함수의 인수 또는 배열의 나머지 요소들을 하나의 변수로 묶는 데 사용됩니다. 예를 들어, function(...args) {}.
* **Spread 연산자:** 배열 또는 객체의 요소를 개별적으로 분해하여 함수 인수 또는 다른 배열/객체에 전달합니다. 예를 들어, [...array].

#### **제너레이터에 대해 설명해주세요.**

**답변:** 제너레이터는 function\* 키워드로 정의되며, 실행을 중간에 멈췄다가 재개할 수 있는 함수입니다. yield 키워드를 사용하여 값을 반환하고, next() 메서드로 실행을 재개할 수 있습니다.

#### **이터러블과 이터레이터 프로토콜에 대해 설명해주세요.**

**답변:**

* **이터러블 프로토콜:** 객체가 Symbol.iterator 메서드를 가지고 있어야 하며, 이 메서드는 이터레이터를 반환해야 합니다.
* **이터레이터 프로토콜:** 이터레이터는 next() 메서드를 가지고 있으며, 이 메서드는 { value: 값, done: 불리언 } 객체를 반환합니다.

#### **JavaScript에서의 화살표 함수(Arrow Function), 템플릿 리터럴(Template Literal), 클래스(Class), 및 비구조화 할당(Destructuring Assignment)에 대해 설명하세요.**

**답변:**

* **화살표 함수:** function 키워드 대신 =>를 사용하여 간결하게 함수를 작성합니다. this가 상위 스코프를 가리킵니다.
* **템플릿 리터럴:** 백틱(`)을 사용하여 문자열을 작성하며, ${}를 사용하여 변수나 표현식을 포함할 수 있습니다.
* **클래스:** 객체 지향 프로그래밍의 개념을 구현하는 템플릿으로, constructor 메서드와 함께 사용됩니다.
* **비구조화 할당:** 배열이나 객체의 값을 분해하여 변수에 할당하는 문법입니다. 예를 들어, const [a, b] = array; 또는 const { x, y } = obj;.

#### **const, let, var의 차이점과 각각의 사용 사례에 대해 설명해보세요.**

**답변:**

* **const:** 재할당이 불가능한 상수를 선언할 때 사용합니다. 블록 스코프를 가집니다.
* **let:** 재할당이 가능한 변수를 선언할 때 사용합니다. 블록 스코프를 가집니다.
* **var:** 함수 스코프를 가지며, 중복 선언이 가능하고, 호이스팅됩니다. 현재는 거의 사용되지 않습니다.

#### **JavaScript를 사용하여 이벤트 핸들링을 어떻게 구현하는지 설명해보세요.**

**답변:** 이벤트 핸들링은 addEventListener 메서드를 사용하여 특정 이벤트가 발생할 때 실행할 함수를 등록합니다. 예를 들어, element.addEventListener('click', function() { /\* 코드 \*/ });.

#### **이벤트 위임(Event Delegation)의 개념과 사용 이유에 대해 설명해주세요.**

**답변:** 이벤트 위임은 상위 요소에 이벤트 리스너를 설정하여, 하위 요소에서 발생한 이벤트를 처리하는 방법입니다. 이는 많은 하위 요소에 이벤트 리스너를 설정하지 않고도 효율적으로 이벤트를 처리할 수 있게 해줍니다.

#### **JavaScript에서 비동기 코드를 처리하는 방법과 콜백(callback) 함수의 문제점에 대해 설명해주세요.**

**답변:** 비동기 코드는 콜백 함수, Promise, Async/Await를 사용하여 처리할 수 있습니다. 콜백 함수의 문제점은 가독성이 떨어지고, 콜백 지옥이 발생할 수 있다는 것입니다. Promise와 Async/Await를 사용하여 이를 해결할 수 있습니다.

#### **Promise와 async/await의 차이점을 설명하고, 언제 어떤 것을 사용해야 할까요?**

**답변:** Promise는 비동기 작업의 성공과 실패를 처리하는 객체이며, then, catch, finally 메서드를 사용합니다. Async/Await는 Promise를 기반으로 하며, 비동기 코드를 동기식 코드처럼 작성할 수 있게 해줍니다. 복잡한 비동기 흐름을 처리할 때는 Async/Await를 사용하는 것이 가독성과 유지보수성 측면에서 유리합니다.

* **React**

#### **React에 대해 설명해주세요.**

**답변:** React는 Facebook에서 개발한 사용자 인터페이스를 구축하기 위한 JavaScript 라이브러리입니다. 컴포넌트 기반 아키텍처를 채택하여 재사용 가능한 UI 컴포넌트를 만들 수 있게 하며, 가상 DOM을 통해 효율적인 렌더링을 지원합니다.

#### **React의 원리, 특징, 장단점이 무엇인가요?**

**답변:**

* **원리:** React는 가상 DOM을 사용하여 UI의 변화를 감지하고, 최소한의 DOM 업데이트를 통해 효율적인 렌더링을 수행합니다.
* **특징:** 컴포넌트 기반 구조, 단방향 데이터 바인딩, 선언적 UI, 가상 DOM.
* **장점:** 재사용 가능한 컴포넌트, 빠른 렌더링, 커뮤니티와 생태계의 지원.
* **단점:** 러닝 커브, JSX 문법의 초기 적응 필요, 상태 관리의 복잡성.

#### **Virtual DOM 이 무엇인지 설명해주세요.**

**답변:** Virtual DOM은 실제 DOM의 가벼운 복사본입니다. React는 상태 변화 시 Virtual DOM을 업데이트하고, 실제 DOM과 비교하여 변경된 부분만 실제 DOM에 반영함으로써 성능을 최적화합니다.

#### **Virtual DOM 작동 원리에 대해 설명해주세요.**

**답변:** Virtual DOM은 상태 변화가 발생하면 전체 UI를 가상으로 다시 렌더링합니다. 이후 이전 Virtual DOM과 비교하여 차이점을 찾고, 실제 DOM에 최소한의 업데이트만 적용합니다. 이를 통해 성능을 높일 수 있습니다.

#### **JSX에 대해 설명해주세요.**

**답변:** JSX는 JavaScript XML의 약자로, JavaScript 코드 내에서 HTML을 작성할 수 있게 해줍니다. JSX는 React.createElement() 함수를 호출하는 JavaScript 코드로 변환되어 사용됩니다.

#### **엘리먼트와 컴포넌트의 차이에 대해 설명해주세요.**

**답변:**

* **엘리먼트:** React 엘리먼트는 UI의 가장 작은 단위로, DOM 요소나 컴포넌트를 나타냅니다.
* **컴포넌트:** 컴포넌트는 상태와 생명 주기 메서드를 가지며, 엘리먼트를 반환하는 함수 또는 클래스입니다. UI의 재사용 가능한 블록을 구성합니다.

#### **리액트에서 컴포넌트를 어떻게 생성하나요?**

**답변:** 컴포넌트는 함수형 또는 클래스형으로 생성할 수 있습니다.

* 함수형 컴포넌트: function MyComponent() { return <div>Hello</div>; }
* 클래스형 컴포넌트: class MyComponent extends React.Component { render() { return <div>Hello</div>; } }

#### **클래스형 컴포넌트를 사용해보셨나요?**

**답변:** 네, 클래스형 컴포넌트를 사용해본 경험이 있습니다. 클래스형 컴포넌트는 상태(state)와 생명 주기 메서드를 관리할 수 있어 복잡한 로직을 처리하는 데 유용합니다.

#### **클래스형 컴포넌트와 함수형 컴포넌트의 차이에 대해 설명해주세요.**

**답변:**

* **클래스형 컴포넌트:** 상태와 생명 주기 메서드를 가질 수 있으며, this 키워드를 사용합니다. ES6 클래스를 기반으로 합니다.
* **함수형 컴포넌트:** 단순히 props를 받아 JSX를 반환하는 함수입니다. React 16.8 이후로 Hooks를 사용하여 상태와 생명 주기 기능을 사용할 수 있게 되었습니다.

#### ⭐ **라이프사이클에 대해 설명해주세요**.

**답변:** 컴포넌트의 라이프사이클은 컴포넌트가 생성되고 업데이트되며 소멸되는 과정을 말합니다. 주요 단계로는 마운트(Mount), 업데이트(Update), 언마운트(Unmount)가 있으며, 각 단계에서 호출되는 메서드가 있습니다.

#### ⭐ **라이프 사이클 메소드에 대해 설명해주세요.**

**답변:**

* **마운트:** constructor(), componentDidMount()
* **업데이트:** shouldComponentUpdate(), componentDidUpdate()
* **언마운트:** componentWillUnmount()

#### **함수형 컴포넌트의 장점에 대해 설명해주세요.**

**답변:** 함수형 컴포넌트는 작성이 간단하고, 코드가 더 깔끔합니다. React Hooks를 사용하여 상태와 생명 주기 메서드를 사용할 수 있어, 클래스형 컴포넌트보다 이해하기 쉽고 유지보수가 용이합니다.

#### ⭐ **React Hooks에 대해 설명해주세요.**

**답변:** React Hooks는 함수형 컴포넌트에서 상태와 생명 주기 기능을 사용할 수 있게 해주는 함수입니다. 주요 Hooks로는 useState, useEffect, useContext, useReducer, useMemo, useCallback 등이 있습니다.

#### ⭐ **함수형 컴포넌트에서 클래스형 컴포넌트의 라이프 사이클 메소드를 비슷하게 사용하는 방법에 대해 설명해주세요**. (useEffect => ComponentDidMount, ComponentDidUpdate, ComponentWillUnmount)

**답변:** useEffect Hook을 사용하여 함수형 컴포넌트에서 클래스형 컴포넌트의 라이프 사이클 메서드를 비슷하게 사용할 수 있습니다.

* componentDidMount와 componentDidUpdate는 의존성 배열 없이 useEffect에서 구현할 수 있습니다.
* componentWillUnmount는 useEffect의 정리(cleanup) 함수를 사용합니다.

javascript

코드 복사

useEffect(() => {

// componentDidMount, componentDidUpdate 역할

return () => {

// componentWillUnmount 역할

};

}, [/\* 의존성 배열 \*/]);

#### **useEffect와 useLayoutEffect의 차이점에 대해 설명해주세요.**

**답변:**

* **useEffect:** 비동기적으로 실행되며, DOM 업데이트 이후에 실행됩니다. 주로 사이드 이펙트를 처리하는 데 사용됩니다.
* **useLayoutEffect:** 동기적으로 실행되며, 브라우저가 페인트하기 전에 실행됩니다. 레이아웃 측정을 하거나 DOM 변경을 바로 반영해야 할 때 사용됩니다.

#### **Pure Component에 대해 설명해주세요.**

**답변:** Pure Component는 shouldComponentUpdate 메서드를 자동으로 구현하여, props와 state의 얕은 비교를 통해 불필요한 렌더링을 방지하는 클래스형 컴포넌트입니다.

#### **shouldComponentUpdate에 대해 설명해주세요.**

**답변:** shouldComponentUpdate는 컴포넌트가 업데이트될 때 호출되며, 반환 값에 따라 렌더링 여부를 결정합니다. true를 반환하면 렌더링이 수행되고, false를 반환하면 렌더링을 건너뜁니다.

#### **State에 대해 설명해주세요.**

**답변:** State는 컴포넌트의 동적인 데이터를 저장하는 객체입니다. State가 변경되면 컴포넌트는 다시 렌더링됩니다. 클래스형 컴포넌트에서는 this.state로 접근하며, 함수형 컴포넌트에서는 useState Hook을 사용합니다.

#### **React에서 State를 어떻게 관리하나요?**

**답변:**

* **로컬 상태 관리:** useState Hook을 사용하여 개별 컴포넌트 내에서 상태를 관리합니다.
* **전역 상태 관리:** Context API, Redux, MobX 등을 사용하여 애플리케이션 전반에 걸친 상태를 관리합니다.

#### **Props에 대해 설명해주세요.**

**답변:** Props는 부모 컴포넌트가 자식 컴포넌트에 데이터를 전달하기 위해 사용하는 읽기 전용 객체입니다. 자식 컴포넌트는 전달된 props를 사용하여 UI를 렌더링합니다.

#### **⭐ Props Drilling에 대해 설명해주세요.**

**답변:** Props Drilling은 상위 컴포넌트에서 하위 컴포넌트로 props를 계속 전달해야 하는 상황을 말합니다. 중간에 필요하지 않은 컴포넌트들도 데이터를 전달받게 되어 복잡해질 수 있습니다.

#### **⭐ Props Drilling을 어떻게 해결할 수 있나요?**

**답변:** Props Drilling 문제를 해결하기 위해 Context API를 사용하여 전역적으로 상태를 관리할 수 있습니다. 이를 통해 필요할 때마다 데이터를 직접 접근할 수 있습니다.

#### **데이터를 자식에서 부모로도 전달할 수 있나요?**

**답변:** 네, 자식 컴포넌트에서 부모 컴포넌트로 데이터를 전달하려면 부모 컴포넌트에 콜백 함수를 props로 전달하고, 자식 컴포넌트에서 해당 콜백 함수를 호출하여 데이터를 전달할 수 있습니다.

#### **Props와 State의 차이에 대해 설명해주세요.**

**답변:**

* **Props:** 부모 컴포넌트에서 자식 컴포넌트로 전달되는 읽기 전용 데이터입니다.
* **State:** 컴포넌트 내부에서 관리되는 동적인 데이터로, 변경될 수 있습니다.

#### **왜 state를 직접 바꾸지 않고 setState (useState)를 사용해야 하나요?**

**답변:** setState 또는 useState를 사용해야 React가 상태 변경을 감지하고, 컴포넌트를 다시 렌더링하기 때문입니다. 직접 state를 변경하면 React는 이를 감지하지 못해 UI가 업데이트되지 않습니다.

#### **React 에서 상태 변화가 생겼을 때, 변화를 어떻게 알아채는지에 대해 설명해주세요.**

**답변:** React는 setState 또는 useState가 호출될 때마다 Virtual DOM을 사용하여 이전 상태와 현재 상태를 비교합니다. 차이점을 감지하고, 필요한 부분만 실제 DOM에 업데이트합니다.

#### **React에서 State의 불변성은 어떻게 유지할 수 있나요?**

**답변:** 불변성을 유지하려면 상태를 직접 수정하지 않고, 새로운 객체나 배열을 생성하여 업데이트합니다. 예를 들어, 스프레드 연산자를 사용하여 새로운 상태를 생성합니다. const newState = { ...oldState, newValue };

#### **setState는 동기적으로 동작하나요? 아니면 비동기적으로 동작하나요?**

**답변:** setState는 비동기적으로 동작합니다. 여러 상태 변경이 동시에 일어날 때 성능 최적화를 위해 이를 일괄 처리(batch processing)합니다.

#### **왜 비동기적으로 동작하나요?**

**답변:** 비동기적으로 동작함으로써 여러 상태 변경이 동시에 일어날 때 성능을 최적화하고, 불필요한 렌더링을 방지할 수 있습니다.

#### **HTML과 React의 이벤트 처리 차이점에 대해 설명해주세요.**

**답변:**

* **HTML:** 이벤트 핸들러는 문자열로 지정하며, 전역 범위에서 호출됩니다.
* **React:** 이벤트 핸들러는 함수로 지정하며, JSX 문법을 사용합니다. 이벤트가 SyntheticEvent 객체로 래핑되어 브라우저 간 호환성을 제공합니다.

#### **⭐ Key Props를 사용하는 이유에 대해 설명해주세요.**

**답변:** Key props는 React가 리스트를 효율적으로 업데이트하는 데 사용됩니다. Key를 사용하여 각 엘리먼트를 식별하고, 변경, 추가, 제거 시 최적의 방식으로 업데이트를 수행합니다.

#### **Ref의 용도에 대해 설명해주세요.**

**답변:** Ref는 React 요소에 직접 접근하여 DOM 요소나 컴포넌트 인스턴스를 참조할 수 있게 합니다. 주로 포커스 설정, 텍스트 선택, 애니메이션 등을 제어할 때 사용됩니다.

#### **제어 컴포넌트와 비제어 컴포넌트의 차이에 대해 설명해주세요.**

**답변:**

* **제어 컴포넌트:** 상태를 통해 폼 입력값을 제어합니다. 값이 state에 저장되고, state를 통해 업데이트됩니다.
* **비제어 컴포넌트:** 직접 DOM 요소에 접근하여 값을 제어합니다. ref를 사용하여 폼 데이터를 가져옵니다.

#### **HOC (Higher-Order Components)에 대해 설명해주세요.**

**답변:** HOC는 컴포넌트를 인수로 받아 새로운 컴포넌트를 반환하는 함수입니다. 주로 코드 재사용, 로직 추상화, 부가 기능 추가에 사용됩니다. 예를 들어, 권한 검사, 로깅, 데이터 가져오기 등이 있습니다.

#### **Context API에 대해 설명해주세요.**

**답변:** Context API는 전역적으로 데이터를 공유할 수 있게 해주는 React의 기능입니다. 주로 테마, 사용자 인증 정보, 설정 등을 전역적으로 관리할 때 사용됩니다.

#### **React.Fragment에 대해 설명해주세요.**

**답변:** React.Fragment는 여러 요소를 하나의 부모 요소로 그룹화할 때 사용되며, 추가적인 DOM 노드를 생성하지 않습니다. JSX에서 <></> 구문으로 사용할 수 있습니다.

#### **Portal에 대해 설명해주세요.**

**답변:** Portal은 컴포넌트를 현재 DOM 계층 구조 외부에 렌더링할 수 있게 합니다. 주로 모달, 팝업, 툴팁 등 최상위 요소로 렌더링해야 하는 UI를 구현할 때 사용됩니다.

#### **에러 바운더리에 대해 설명해주세요.**

**답변:** 에러 바운더리는 컴포넌트 트리의 에러를 포착하여 UI를 안정적으로 유지하는 컴포넌트입니다. componentDidCatch와 getDerivedStateFromError 메서드를 사용하여 에러를 처리합니다.

#### **메모이제이션에 대해 설명해주세요.**

**답변:** 메모이제이션은 함수의 실행 결과를 저장하여 동일한 입력값으로 함수가 다시 호출될 때 저장된 결과를 반환하는 최적화 기술입니다. React에서는 useMemo와 useCallback Hook을 사용하여 메모이제이션을 구현합니다.

#### ⭐ **리액트에서 메모이제이션을 어떤 방식으로 활용할 수 있나요?**

**답변:**

* **useMemo:** 컴포넌트의 렌더링 성능을 최적화하기 위해 연산 비용이 높은 계산 결과를 메모이제이션합니다.
* **useCallback:** 함수형 컴포넌트에서 이벤트 핸들러 함수를 메모이제이션하여, 불필요한 재생성을 방지합니다.

#### **⭐ useMemo와 useCallback에 대해 설명해주세요.**

**답변:**

* **useMemo:** 연산 비용이 높은 값을 메모이제이션합니다. useMemo(() => computeExpensiveValue(a, b), [a, b]).
* **useCallback:** 함수를 메모이제이션합니다. useCallback(() => { doSomething(a); }, [a]).

#### **React.memo와 useMemo의 차이에 대해 설명해주세요.**

**답변:**

* **React.memo:** 고차 컴포넌트로, 컴포넌트를 메모이제이션하여 props가 변경되지 않으면 컴포넌트를 다시 렌더링하지 않습니다.
* **useMemo:** 값의 계산 결과를 메모이제이션하여 의존성이 변경되지 않으면 계산을 다시 수행하지 않습니다.

#### **리액트 관련 패키지 중에 제일 좋다고 생각한 것은 무엇인가요?**

**답변:** 개인적으로 Redux와 React Router를 유용하게 사용했습니다. Redux는 전역 상태 관리를 효과적으로 도와주며, React Router는 SPA(Single Page Application) 내에서의 라우팅을 쉽게 구현할 수 있게 해줍니다.

#### **리액트의 렌더링 성능 향상을 위해 어떻게 해야 하나요?**

**답변:**

* 불필요한 렌더링 방지 (React.memo, shouldComponentUpdate)
* 코드 스플리팅과 동적 임포트
* Lazy loading을 통해 컴포넌트 지연 로딩
* useMemo, useCallback을 사용한 메모이제이션
* 가상화된 리스트 (React Virtualized) 사용

#### **CSS in JS를 사용해 본 적이 있나요? 있다면 소감을 말해주세요.**

**답변:** 네, 사용해 본 적이 있습니다. CSS in JS는 스타일을 컴포넌트에 국한시키고, 동적 스타일링을 가능하게 해주어 편리합니다. 그러나 큰 프로젝트에서는 스타일 관리가 복잡해질 수 있습니다.

#### **React 18 버전 업데이트 내용에 대해 말씀해주세요.**

**답변:** React 18에서는 Concurrent Rendering, Automatic Batching, Transitions API, 새로운 Hooks (useId, useTransition), Suspense 개선 등이 추가되었습니다. 이는 애플리케이션의 성능과 사용자 경험을 크게 향상시킵니다.

#### **Client Side Routing에 대해 설명해주세요. (CSR)**

**답변:** Client Side Routing은 브라우저의 히스토리 API를 사용하여 클라이언트 측에서 페이지 간 이동을 처리합니다. 전체 페이지 리로드 없이 콘텐츠를 동적으로 업데이트하여 빠른 사용자 경험을 제공합니다.

#### **Next.js 를 사용해보셨나요?**

**답변:** 네, Next.js를 사용해본 경험이 있습니다. Next.js는 React 기반의 프레임워크로, 서버 사이드 렌더링(SSR), 정적 사이트 생성(SSG), API 라우팅 등을 제공하여 성능과 SEO를 향상시킵니다.

#### **React에서 Form 이벤트는 어떻게 제어하셨나요?**

**답변:** Form 이벤트는 onChange 이벤트 핸들러를 사용하여 입력값을 상태에 저장하고, onSubmit 이벤트 핸들러를 사용하여 폼 제출을 처리합니다.

#### **React Hook Form 를 사용해보셨나요?**

**답변:** 네, 사용해 본 적이 있습니다. React Hook Form은 성능 최적화된 폼 관리를 제공하며, 간단한 API로 폼 유효성 검사와 상태 관리를 쉽게 구현할 수 있습니다.

#### **기억나는 Hook에 대해 전부 설명해주세요.**

**답변:** 주요 Hook으로는 useState, useEffect, useContext, useReducer, useMemo, useCallback, useRef, useLayoutEffect, useImperativeHandle 등이 있습니다. 각각 상태 관리, 사이드 이펙트 처리, 컨텍스트 접근, 성능 최적화 등을 위해 사용됩니다.

#### React에서 컴포넌트 A가 사용하는 CSS파일과 컴포넌트 B가 사용하는 CSS파일의 선택자가 겹치는 이유가 뭘까요?

**답변:** CSS 파일의 전역 범위 때문입니다. CSS는 전역으로 적용되므로 동일한 선택자가 있으면 충돌이 발생합니다. 이를 해결하기 위해 CSS 모듈이나 CSS-in-JS를 사용할 수 있습니다.

#### **⭐ Redux에 대해 설명해주세요.**

**답변:** Redux는 JavaScript 애플리케이션을 위한 상태 관리 라이브러리입니다. 단일 스토어에서 상태를 관리하고, 액션과 리듀서를 통해 상태 변경을 처리합니다.

#### **⭐ Redux를 사용하는 이유에 대해 설명해주세요.**

**답변:** Redux는 복잡한 애플리케이션에서 상태 관리를 일관되게 유지할 수 있게 해줍니다. 예측 가능한 상태 관리, 시간 여행 디버깅, 미들웨어를 통한 확장성 등이 주요 이유입니다.

#### **⭐ Redux의 장단점에 대해 설명해주세요.**

**답변:**

* **장점:** 예측 가능한 상태 관리, 시간 여행 디버깅, 중앙 집중식 상태 관리, 미들웨어를 통한 확장성.
* **단점:** 보일러플레이트 코드 증가, 러닝 커브, 작은 애플리케이션에서의 과도한 사용.

#### **Flux 패턴에 대해 설명해주세요.**

**답변:** Flux는 Facebook에서 개발한 애플리케이션 아키텍처로, 단방향 데이터 흐름을 강조합니다. 액션, 디스패처, 스토어, 뷰로 구성되어 있으며, Redux의 설계에 영향을 주었습니다.

#### **Context API와 Redux를 비교해주세요.**

**답변:** Context API는 전역 상태 관리를 위해 React에서 제공하는 간단한 방법입니다. Redux는 복잡한 상태 관리를 위해 설계된 라이브러리입니다. Context API는 간단한 상태 관리에 적합하고, Redux는 복잡한 상태와 비즈니스 로직을 처리하는 데 유리합니다.

#### **Redux의 3대 원칙에 대해 설명해주세요.**

**답변:**

1. **단일 진실의 원천:** 모든 상태는 하나의 스토어에서 관리됩니다.
2. **상태는 읽기 전용:** 상태는 불변성을 유지하며, 오직 액션을 통해서만 변경될 수 있습니다.
3. **변경은 순수 함수로 작성:** 리듀서는 순수 함수로 작성되어야 하며, 이전 상태와 액션을 받아 새로운 상태를 반환합니다.

#### **React-Query를 사용해보셨나요?**

**답변:** 네, 사용해 본 적이 있습니다. React-Query는 서버 상태 관리를 위한 라이브러리로, 데이터 페칭, 캐싱, 동기화, 업데이트를 쉽게 처리할 수 있게 해줍니다.

#### **React-Query에 대해 설명해주세요.**

**답변:** React-Query는 서버 상태를 관리하고, 데이터 페칭, 캐싱, 동기화 및 업데이트를 쉽게 처리할 수 있게 해주는 라이브러리입니다. 복잡한 비동기 로직을 간단하게 관리할 수 있습니다.

#### **React-Query의 등장 배경과 장점에 대해 설명해주세요.**

**답변:** 서버 상태 관리는 클라이언트 상태 관리와 달리 비동기 데이터 페칭, 캐싱, 동기화 등이 필요합니다. React-Query는 이러한 요구 사항을 효율적으로 처리하여, 개발자가 비동기 로직에 집중할 수 있게 해줍니다. 주요 장점으로는 자동 리페칭, 캐싱, 쿼리 무효화, 쿼리 상태 관리 등이 있습니다.

#### **Recoil을 사용해보셨나요?**

**답변:** 네, 사용해 본 적이 있습니다. Recoil은 Facebook에서 개발한 상태 관리 라이브러리로, React의 상태 관리를 단순화하고 확장성을 제공합니다.

#### **Recoil에 대해 설명해주세요.**

**답변:** Recoil은 React 애플리케이션의 상태 관리를 단순화하고, 비동기 데이터 흐름을 쉽게 처리할 수 있게 해주는 상태 관리 라이브러리입니다. Atom과 Selector를 사용하여 상태와 파생 상태를 관리합니다.

#### **Recoil에서 Loadable의 개념에 대해 설명해주세요.**

**답변:** Loadable은 비동기 상태를 관리하기 위한 Recoil의 유틸리티입니다. 비동기 작업의 로딩 상태, 성공, 실패를 쉽게 관리할 수 있습니다.

#### **Recoil에서 비동기로 데이터를 받아올 때 State를 어떻게 관리하셨나요?**

**답변:** Recoil의 Atom과 Selector를 사용하여 비동기 데이터를 관리합니다. Selector에서 비동기 함수를 사용하여 데이터를 페칭하고, Loadable을 통해 상태를 관리합니다.

#### **Recoil에서 로딩, 성공, 에러와 관련된 처리를 어떻게 하셨나요?**

**답변:** Recoil의 Loadable을 사용하여 비동기 상태를 관리합니다. loadable.state를 통해 로딩, 성공, 에러 상태를 확인하고, 이에 따라 UI를 업데이트합니다.

#### **⭐ Redux와 Recoil에 대해 비교 설명해주세요. (개념, 장단점)**

* **Redux:** 중앙 집중식 상태 관리, 미들웨어 지원, 복잡한 상태 관리에 유리.
  + **장점:** 예측 가능한 상태 관리, 풍부한 생태계.
  + **단점:** 보일러플레이트 코드 증가, 러닝 커브.
* **Recoil:** 분산된 상태 관리, 비동기 상태 관리에 유리, React와의 높은 호환성.
  + **장점:** 간단한 상태 관리, 비동기 데이터 흐름 처리 용이.
  + **단점:** 생태계가 비교적 작음, 새로운 개념 학습 필요.

### **프론트엔드 전반**

#### **⭐ 브라우저 렌더링 과정을 설명해주세요.**

**답변:** 브라우저는 HTML, CSS, JavaScript를 받아들여 다음 과정을 통해 렌더링합니다:

1. **HTML 파싱:** HTML 문서를 파싱하여 DOM 트리를 생성합니다.
2. **CSS 파싱:** CSS를 파싱하여 CSSOM(CSS Object Model)을 생성합니다.
3. **렌더 트리 구성:** DOM과 CSSOM을 결합하여 렌더 트리를 생성합니다.
4. **레이아웃:** 각 요소의 크기와 위치를 계산합니다.
5. **페인트:** 화면에 픽셀을 그립니다.
6. **컴포지팅:** 여러 레이어를 결합하여 최종 화면을 구성합니다.

#### **브라우저는 어떻게 동작 하나요?**

**답변:** 브라우저는 사용자가 입력한 URL을 통해 웹 서버에 요청을 보내고, 서버로부터 HTML, CSS, JavaScript 파일을 응답받습니다. 브라우저는 이를 파싱하여 DOM, CSSOM을 생성하고, 렌더 트리를 구성하여 화면에 렌더링합니다.

#### **Webpack, Babel, Polyfill에 대해 설명해주세요.**

**답변:**

* **Webpack:** 모듈 번들러로, 여러 파일을 하나의 번들로 결합하고, 최적화 및 로드 속도를 향상시킵니다.
* **Babel:** JavaScript 컴파일러로, 최신 문법을 구 브라우저에서도 동작할 수 있게 트랜스파일합니다.
* **Polyfill:** 최신 브라우저 기능을 지원하지 않는 브라우저에서도 사용할 수 있게 해주는 코드입니다.

#### **ESLint와 Prettier에 대해 설명해주세요.**

**답변:**

* **ESLint:** 코드 품질을 유지하기 위한 정적 코드 분석 도구로, 코드의 문제점을 찾아내고 규칙을 설정하여 일관성을 유지합니다.
* **Prettier:** 코드 포맷터로, 코드 스타일을 일관되게 유지하고, 자동으로 코드 형식을 맞춰줍니다.

#### **SPA와 MPA에 대해 설명해주세요.**

**답변:**

* **SPA(Single Page Application):** 단일 HTML 페이지로 구성된 애플리케이션으로, 페이지 간 이동 시 전체 페이지 리로드 없이 동적으로 콘텐츠를 업데이트합니다.
* **MPA(Multi-Page Application):** 여러 HTML 페이지로 구성된 애플리케이션으로, 페이지 간 이동 시마다 전체 페이지가 새로 로드됩니다.

#### **⭐ CSR과 SSR의 차이는 무엇인가요?**

**답변:**

* **CSR(Client-Side Rendering):** 클라이언트 측에서 JavaScript를 사용하여 렌더링을 처리하는 방식으로, 초기 로딩 속도가 느릴 수 있지만, 이후 상호작용이 빠릅니다.
* **SSR(Server-Side Rendering):** 서버 측에서 렌더링을 처리하여 완성된 HTML을 클라이언트에 전달하는 방식으로, 초기 로딩 속도가 빠르고, SEO에 유리합니다.

#### **⭐ CORS와 해결 경험을 말씀해주세요.**

**답변:** CORS(Cross-Origin Resource Sharing)는 도메인 간 리소스 요청을 허용하도록 설정하는 방식입니다. 서버에서 특정 도메인에 대해 허용 헤더(Access-Control-Allow-Origin)를 설정하여 해결할 수 있습니다. 예를 들어, Express.js 서버에서 CORS 미들웨어를 사용하여 해결한 경험이 있습니다.

#### **bundle의 사이즈를 줄이려면 어떻게 해야 하나요?**

**답변:**

* 코드 스플리팅
* 트리 쉐이킹
* 압축 (Webpack의 TerserPlugin 사용)
* 이미지 및 리소스 최적화
* 불필요한 라이브러리 제거

#### **⭐ 쿠키, 세션, 웹스토리지의 차이에 대해 설명해주세요.**

**답변:**

* **쿠키:** 클라이언트와 서버 간에 작은 데이터를 저장하고 전송하기 위한 방법입니다. 만료 시간이 있으며, 모든 요청에 포함됩니다.
* **세션:** 서버에 사용자 데이터를 저장하고, 클라이언트는 세션 ID를 통해 데이터를 참조합니다.
* **웹스토리지:** 브라우저에 데이터를 저장하는 방법으로, 로컬 스토리지와 세션 스토리지가 있습니다. 로컬 스토리지는 만료되지 않고, 세션 스토리지는 탭을 닫을 때 삭제됩니다.

#### **브라우저에서 탭 이동 혹은 탭 종료 시에는 세션 스토리지에 어떤 영향을 끼치나요?**

**답변:** 세션 스토리지는 탭이 닫히면 데이터가 삭제됩니다. 탭 이동 시에는 유지되지만, 탭을 종료하면 데이터가 사라집니다.

#### **⭐ 로그인 처리를 할 때 쿠키와 세션을 어떻게 사용하시나요?**

**답변:** 쿠키를 사용하여 세션 ID를 저장하고, 서버에서 세션 ID를 통해 사용자를 인증합니다. 쿠키에는 세션 ID와 같은 민감한 정보를 저장하며, 서버는 세션 스토리지에 사용자 정보를 저장합니다.

#### **⭐ 토큰 기반 인증 방식에 대해 설명해주세요.**

**답변:** 토큰 기반 인증은 서버에서 클라이언트를 인증하고, JWT(JSON Web Token)와 같은 토큰을 발급하여 클라이언트가 이를 저장합니다. 이후 요청 시마다 토큰을 포함하여 서버에 인증을 요청합니다.

#### **⭐ JWT 토큰을 쿠키에 저장했을 때 취약점에 대해 설명해주세요.**

**답변:** JWT 토큰을 쿠키에 저장하면 CSRF 공격에 취약할 수 있습니다. 이를 방지하기 위해 SameSite 속성을 설정하거나, 쿠키를 HttpOnly로 설정하여 JavaScript로 접근하지 못하게 해야 합니다.

#### **⭐ 크로스 브라우징에 대해 설명해주세요.**

**답변:** 크로스 브라우징은 다양한 브라우저에서 웹 애플리케이션이 동일하게 동작하도록 하는 것입니다. 이를 위해 폴리필, CSS 리셋, 브라우저 테스트 등을 사용합니다.

#### **크로스 브라우징 이슈를 해결했던 경험을 말씀해주세요.**

**답변:** 특정 브라우저에서 CSS가 올바르게 렌더링되지 않는 문제를 해결하기 위해, 브라우저별 CSS 프리픽스를 추가하고, Polyfill을 사용하여 최신 JavaScript 기능을 지원하지 않는 브라우저에서도 동작하도록 했습니다.

#### **웹사이트 성능 최적화에는 어떤 방법이 있나요?**

**답변:**

* 코드 스플리팅
* 이미지 최적화
* 지연 로딩 (Lazy Loading)
* 캐싱
* 서버 측 렌더링
* 불필요한 코드 제거

#### **객체 지향 프로그래밍이란 무엇인가요?**

**답변:** 객체 지향 프로그래밍은 데이터와 해당 데이터를 조작하는 메서드를 하나의 객체로 묶어, 재사용성과 확장성을 높이는 프로그래밍 패러다임입니다.

#### **객체 지향 프로그래밍의 특징에 대해 말씀해주세요.**

**답변:** 주요 특징으로는 캡슐화, 상속, 다형성, 추상화가 있습니다. 이는 코드의 재사용성을 높이고, 유지보수를 용이하게 합니다.

#### **객체 지향 프로그래밍의 장단점에 대해 말씀해주세요.**

**답변:**

* **장점:** 재사용성, 확장성, 유지보수 용이성.
* **단점:** 복잡성 증가, 성능 저하 가능성.

#### **⭐ REST API에 대해 설명해주세요.**

**답변:** REST API는 Representational State Transfer의 약자로, HTTP 프로토콜을 기반으로 리소스를 정의하고 접근하는 방식을 말합니다. 주요 HTTP 메서드로는 GET, POST, PUT, DELETE가 있습니다.

#### **Git Flow에 대해 설명해주세요.**

**답변:** Git Flow는 Git을 활용한 브랜치 전략으로, 기능 개발, 릴리즈 준비, 버그 수정 등을 위한 브랜치를 체계적으로 관리합니다. 주요 브랜치로는 master, develop, feature, release, hotfix가 있습니다.

#### **⭐ SEO에 대해 설명해주세요.**

**답변:** SEO(Search Engine Optimization)는 검색 엔진에서 웹 페이지의 순위를 높이기 위한 최적화 작업입니다. 주요 요소로는 키워드 최적화, 메타 태그, 모바일 최적화, 페이지 속도, 링크 빌딩 등이 있습니다.

#### **BOM과 DOM에 대해 설명해주세요.**

**답변:**

* **BOM (Browser Object Model):** 브라우저 창 및 프레임을 제어하는 객체 모델입니다. 주요 객체로는 window, navigator, screen, location, history 등이 있습니다.
* **DOM (Document Object Model):** HTML 및 XML 문서의 구조화된 표현으로, 문서의 콘텐츠를 조작할 수 있는 API를 제공합니다. 주요 객체로는 document, element, node 등이 있습니다.

#### **⭐ 리플로우와 리페인트에 대해 설명해주세요.**

**답변:**

* **리플로우:** DOM의 레이아웃이 변경될 때 발생하며, 브라우저가 요소의 크기와 위치를 다시 계산합니다.
* **리페인트:** 요소의 시각적 속성(색상, 배경 등)이 변경될 때 발생하며, 레이아웃 계산 없이 화면을 다시 그립니다.

#### **리플로우와 리페인트의 발생 조건에 대해 설명해주세요.**

**답변:**

* **리플로우:** DOM 요소의 추가, 제거, 크기 변경, 윈도우 리사이즈 등 레이아웃에 영향을 주는 변경이 있을 때 발생합니다.
* **리페인트:** 색상, 배경, 그림자 등 시각적 속성이 변경될 때 발생합니다.

#### **visibility 속성은 리플로우를 발생시킬까요?**

**답변:** visibility 속성은 리플로우를 발생시키지 않습니다. 요소를 숨기지만, 레이아웃은 그대로 유지되기 때문입니다. 반면, display 속성은 리플로우를 발생시킵니다.

#### **Agile에 대해 설명해주세요.**

**답변:** Agile은 소프트웨어 개발 방법론으로, 빠른 피드백과 적응을 통해 지속적으로 개선하는 것을 목표로 합니다. 짧은 개발 주기(스프린트)를 통해 점진적으로 기능을 추가하고, 팀 간 협업을 강조합니다.

#### **dependencies 와 devDependencies 차이에 대해 설명해주세요**.

**답변:**

* **dependencies:** 애플리케이션이 실행될 때 필요한 패키지입니다.
* **devDependencies:** 개발 과정에서만 필요한 패키지입니다. 예를 들어, 테스트 도구, 빌드 도구 등이 있습니다.

#### **XSS와 CSRF에 대해 설명해주세요.**

**답변:**

* **XSS (Cross-Site Scripting):** 악성 스크립트를 삽입하여 사용자 정보를 탈취하거나, 악의적인 동작을 수행하는 공격입니다. 방어 방법으로는 입력 검증, 출력 인코딩, Content Security Policy(CSP) 설정이 있습니다.
* **CSRF (Cross-Site Request Forgery):** 사용자가 의도하지 않은 요청을 보내도록 유도하여, 사용자의 권한으로 악의적인 동작을 수행하는 공격입니다. 방어 방법으로는 CSRF 토큰 사용, SameSite 속성 설정이 있습니다.

#### **인증과 인가에 대해 설명해주세요.**

**답변:**

* **인증 (Authentication):** 사용자의 신원을 확인하는 과정입니다. 주로 사용자명과 비밀번호, OTP, 생체 인식 등을 사용합니다.
* **인가 (Authorization):** 인증된 사용자가 특정 리소스에 접근할 수 있는 권한을 확인하는 과정입니다. 역할 기반 접근 제어(RBAC) 등을 사용합니다.

#### **package.json에서 private에 대해 설명해주세요.**

**답변:** private 속성이 true로 설정되면, 해당 패키지가 npm에 실수로 게시되는 것을 방지할 수 있습니다. 개인 프로젝트나 내부 패키지에 주로 사용됩니다.

#### **프로젝트를 npm에 배포하려면 어떤 설정이 필요할까요?**

**답변:** package.json 파일에 name, version, description, main, scripts, keywords, author, license 등의 필드를 설정하고, npm publish 명령어를 사용하여 배포합니다. 사전에는 npm 로그인(npm login)이 필요합니다.

#### **⭐ SEO는 어떤 식으로 신경쓰셨나요?**

**답변:** SEO를 위해 메타 태그, 키워드 최적화, 이미지 대체 텍스트, 모바일 최적화, 빠른 페이지 로딩 속도, 구조화된 데이터(Schema Markup) 등을 적용했습니다.

#### **⭐ 어플리케이션의 성능은 어떤식으로 측정하셨나요?**

**답변:** 주로 Lighthouse, Web Vitals, Chrome DevTools 등의 도구를 사용하여 어플리케이션의 성능을 측정합니다. 페이지 로딩 시간, 렌더링 속도, 인터랙티브 성능 등을 분석합니다.

#### **LightHouse 결과가 좋지 않을 때, 프로젝트에서 병목 현상은 어떻게 해결할 수 있을까요?**

**답변:** 성능이 저하된 부분을 파악하고, 코드 스플리팅, 이미지 최적화, Lazy Loading, 캐싱, 불필요한 코드 제거 등을 통해 병목 현상을 해결합니다.

#### **⭐ 주소창에 주소를 입력했을 때의 흐름을 설명해주세요.**

**답변:**

1. **DNS 조회:** 도메인 이름을 IP 주소로 변환합니다.
2. **TCP 연결:** 클라이언트와 서버 간에 TCP 연결을 수립합니다. (3-way handshake)
3. **HTTP 요청:** 클라이언트가 서버에 HTTP 요청을 보냅니다.
4. **서버 처리:** 서버가 요청을 처리하고, 응답을 생성합니다.
5. **HTTP 응답:** 서버가 클라이언트에 HTTP 응답을 보냅니다.
6. **렌더링:** 브라우저가 HTML, CSS, JavaScript를 파싱하고 렌더링하여 화면에 표시합니다.