JavaScript 핵심 치트시트

변수와 데이터 타입

- **변수 선언**:
  - `const`: 블록 스코프이며, 재할당이 불가능합니다. 기본적으로 `const` 사용을 권장합니다.
  - `let`: 블록 스코프이며, 재할당이 가능합니다.
  - `var`: 함수 스코프이며 호이스팅이 발생합니다. (모던 개발에서는 사용을 지양합니다)
- **데이터 타입**:
  - 원시 타입: `String`, `Number`, `Boolean`, `null`, `undefined`, `Symbol`, `BigInt`
  - 객체 타입: `Object` (배열, 함수, 날짜 등 모두 포함)
- **연산자**:
  - 일치 비교 (`===`): 값과 타입을 모두 비교합니다. (`==`보다 권장됨)
  - Nullish 병합 (`??`): 왼쪽 피연산자가 `null` 또는 `undefined`일 때만 오른쪽을 반환합니다.
  - 옵셔널 체이닝 (`?.`): 체인 중간에 `null`이나 `undefined`가 있어도 에러 없이 `undefined`를 반환합니다.

제어 흐름 및 반복문

- **조건문**: `if...else`, `switch`
- **반복문**:
  - `for`: 전통적인 인덱스 기반 반복
  - `for...of`: 배열 등 이터러블 객체의 값을 순회
  - `for...in`: 객체의 열거 가능한 속성 이름을 순회

함수와 화살표 함수

- **함수 선언문**: `function add(a, b) { return a + b; }` (호이스팅 가능)
- **함수 표현식**: `const add = function(a, b) { return a + b; };`
- **화살표 함수**: `const add = (a, b) => a + b;`
  - 자신만의 `this`를 가지지 않고 상위 스코프의 `this`를 유지합니다.
- **나머지 매개변수** (`...args`): 여러 인수를 배열로 한꺼번에 받습니다.

객체와 클래스

- **객체 구조 분해 할당**: `const { name, age } = user;`
- **스프레드 연산자** (`...`): 객체나 배열의 복사 및 결합에 사용합니다.
- **클래스 (ES6+)**: 프로토타입 기반 상속을 더 직관적으로 표현하는 문법적 설탕(Syntactic Sugar)입니다.

```
class Person {
  constructor(name) { this.name = name; }
  greet() { console.log(`안녕하세요, ${this.name}입니다.`); }
}
```

배열 고급 메서드

- `map()`: 각 요소를 가공하여 새로운 배열 생성
- `filter()`: 조건을 만족하는 요소만 추출하여 새로운 배열 생성
- `reduce()`: 요소를 순회하며 단일 값으로 누적 결과 계산
- `forEach()`: 단순 반복 실행
- `find()`, `findIndex()`: 특정 조건의 요소나 인덱스 검색

비동기 프로그래밍

- **Promise**: 비동기 작업의 미래 결과를 나타내는 객체 (`then`, `catch`, `finally`)
- **Async / Await**: 비동기 코드를 동기 코드처럼 읽기 쉽게 작성하는 최신 문법

```
async function getData() {
  try {
    const response = await fetch('/api/data');
    const data = await response.json();
    return data;
  } catch (err) {
    console.error(err);
  }
}
```

모듈 시스템 (ESM)

- **export**: 변수나 함수를 외부로 공개 (`export const myVar = 1;` 또는 `export default App;`)
- **import**: 외부 모듈 가져오기 (`import { myVar } from './module.js';`)

DOM 조작 및 이벤트

- **요소 선택**: `document.querySelector()`, `document.getElementById()`
- **내용 변경**: `element.textContent`, `element.innerHTML`
- **이벤트 리스너**: `element.addEventListener('click', (e) => { ... })`

클로저 (Closure)

함수가 선언된 당시의 렉시컬 환경(Lexical Environment)을 기억하여, 외부 함수가 종료된 후에도 그 환경에 접근할 수 있는 기능입니다.

최신 문법 및 유용한 팁

- **Template Literals**: `Hello ${name}` 형식을 사용하여 문자열 내 변수 삽입
- **Array.from()**: 유사 배열 객체를 실제 배열로 변환
- **Object.entries() / fromEntries()**: 객체를 엔트리 배열로 변환하거나 그 반대 작업 수행

Google Style Guide

명명 규칙 (Naming)

- 함수/변수/속성: `lowerCamelCase`
- 클래스/인터페이스: `UpperCamelCase`
- 상수: `UPPER_SNAKE_CASE`
- 비공개 필드: 언더스코어 접미사 (`_private_`)
- 파일 이름: `my_file.js` (소문자, 언더스코어 또는 대시)

포매팅 (Formatting)

- 들여쓰기: 공백 2개 (탭 금지)
- 줄 길이: 최대 80자
- 세미콜론: 반드시 사용 (자동 삽입 의존 금지)
- 따옴표: 작은따옴표(') 선호
- 중괄호: K&R 스타일 (Egyptian brackets)

프로그래밍 관례

- 변수 선언: `const`와 `let` 사용 필수 (`var` 사용 금지).
- 모듈: `goog.module` 또는 ES6 모듈 (`import`/`export`) 사용.
- Default Exports: 사용 금지 (명명된 내보내기만 사용).
- This: 클래스 생성자나 메서드 내에서만 사용 권장.
- Getters/Setters: 지양 (일반 메서드 선호, 필요 시 `getFoo()`/`setFoo()`).

언어 기능 활용

- 배열 리터럴: `new Array()` 대신 `[]` 사용. 트레일링 콤마(,) 권장.
- 객체 리터럴: `quoted`와 `unquoted` 키 혼용 금지. 메서드 단축 표기(`method() {}`) 권장.
- Arrow Functions: 익명 함수와 `this` 바인딩 시 선호.
- Type Casting: `Number()`, `String()` 등 명시적 함수 사용.
- JSDoc: 타입 정보를 포함하여 충실히 작성.