React

JavaScript 주요 개념 (ES6)

JavaScript 변수 선언

- var
  - Var로 선언된 변수는 기존 변수값을 대체
  - 함수 스코프(변수 선언한 함수 내에서만 접근 가능)를 기준으로 동작
  - 기존에 선언된 변수의 값을 덮어쓸 경우다른 코드에 영향 가능
    - 반복 변수 i의 경우, 함수 스코프를 가지므로 함수 내에 다른 i가 선언되어 있다면 값이 덮어쓰여질 가능성
- let(ES2015, ES6)
  - 재선언을 허용하지 않음
  - 재할당은 가능
  - 블록 스코프를 가짐
    - 변수를 둘러싸고 있는 블록 ({ }) 내 에서만 변수에 접근 가능
    - 반복 변수 i의 경우, 반복문 블록 스코프에 묶여 외부 변수 값을 덮어쓰거나 불필요한 참조가 되는 것 방지.
- const(ES2015, ES6)
  - let과 달리 값의 면경을 허용하지 않음
  - 상수

JavaScript 함수

- 함수 정의
- 호출
- This 바인딩
- 클로저(Closure)

- 일급 함수(first-class function)
  - 변수에 함수를 할당 가능
  - 함수를 다른 함수의 인자로 전달 가능
  - 함수를 반환값으로 사용 가능
- 콜백 함수
  - 다른 함수의 인자로 넘어가는 함수
  - Button.addEventListener('click', callback)
- 함수 선언문
  - 함수의 이름과 동일한 변수를 정의하여 함수 객체를 변수에 할당
  - 호이스팅(hoisting)으로 인해 함수가 선언된 위치에서 코드의 최상단으로 끌어올려짐. 이 선언된 위치보다 상단에서 호출 가능
- 함수 표현식
  - Const multiply = function (a, b) { return a * b; }
  - Multiply(2, 3);

JavaScript 함수

- 화살표 함수(arrow function)
  - Function 키워드 생략
  - 매개변수가 하나인 경우 괄호 생략
  - 함수 body에 문장이 하나인 경우 중괄호{ } 나 return 키워드 생략
  - Arguments 객체와 this를 바인딩하지 않기 때문에 기존 함수와 다르게 동작

JavaScript 함수

- this
  - 읽기 전용 값으로 런타임시에 설정할 수 없으며 함수를 호출한 방법에 따라 값이 달라짐
  - 전역 객체(브라우저 환경인 경우 window 객체)를 참조
    - Console.log(this === window); // true
- 메서드
  - 객체의 프로퍼티인 함수를 일반 함수와 구분하여 메서드라 부름.
  - 메서드를 호출하면 this는 해당 메서드를 소유하는 객체로 바인딩됨

Prototype과 Scope

- Scope
  - 변수나 매개변수에 접근할 수 있는 범위를 결정
- 함수 단위 scope
  - var
  - 선언된 함수 단위로 생성되는 scope
- 블록 단위 scope
  - let, const
- 렉시컬 스코프
  - 변수, 함수를 어디에 작성하였는가에 따라 스코프가 결정됨을 의미
  - 함수 스코프와 블록 스코프의 범위를 결정하는 단위
  - 중첩된 스코프 내에서 코드가 실행된 경우, 가장 안쪽의 스코프부터 시작하여 상위 스코프로 올라가며 원하는 대상(변수, 객체)를 검색하고 대상을 찾는 즉시 검색 중단.
  - 상위 방향으로 검색하므로 상위에서 아래쪽(안쪽) 스코프에 있는 변수나 함수에는 접근할 수 없음
  - 스코프 체이닝
    - 스코프 체인를 따라 검색하는 과정

Prototype과 Scope

- 호이스팅(hoisting)
  - 선언문이 스코프 내의 가장 최상단으로 끌어올려지는 것
  - console.log(a); // not Reference error. Undefined로 정상 실행
    var a = 1;
  - var a; 가 전역 스코프의 최상단으로 끌어올려지기(hoisting) 때문에 선언되기 전에도 참조될 수 있음
- 자바스크립트 변수는 세 단계로 나누어 생성됨
  - 선언: 스코프에 변수 선언
  - 초기화: 변수값을 undefined로 초기화. 실제 변수 접근 가능 단계
  - 할당: 할당문을 만나면 실제값 할당
  - var로 선언한 변수는 선언과 초기화를 한번에 실행하고, 두 단계는 스코프의 최상단으로 끌어올려져서 실행됨. 따라서 선언하기도 전에 접근하여도 이미 초기화가 되어 있기 때문에 접근이 가능한 것(호이스팅)
  - 호이스팅은 스코프 별로 동작
    - 함수 내에서 선언된 변수는 함수 스코프 내에서 호이스팅 발생

Prototype과 Scope

- Let 과 const로 선언된 변수는 var와 달리 선언과 초기화가 분리되어 실행됨
- 선언 단계는 스코프 최상단으로 끌어올려져서 실행되지만, 초기화 단계는 선언문을 만날때 실행됨. 따라서 초기화 단계 이전에 접근하려면 Reference error 가 발생됨. 즉 선언 단계가 실행되는 최상단 부터 초기화 단계를 실행하는 선언문이 나오기 전까지는 변수에 접근할 수 없음
- 함수선언문의 호이스팅
  - 함수 선언, 초기화, 할당의 세단계가 모두 스코프 최상단에서 실행됨
  - 따라서 어느 위치에서든 함수 호출 가능
  -

Prototype과 Scope

- 클로저(Closure)
  - 함수의 렉시칼 스코프를 기억하여 함수가 렉시칼 스코프를 벗어난 외부 스코프에서 실행될 때에도 자신의 렉시칼 스코프에 접근할 수 있도록 해 주는 것.
  - 클로저를 사용하여 특정한 상태를 기억하고 캡슐화하거나 하나의 모듈을 정의하는 패턴으로도 확장 가능
  - 클로저를 사용하면 외부에서도 얼마든지 원래의 렉시칼 스코프에 접근 가능

모듈 패턴

- 클로저를 활용하면 모듈을 정의하여 원하는 프로퍼티나 메서드를 캡슐화 할 수 있음
- 모듈 안에 포함된 함수들은 기억되어 있는 렉시칼 스코프 체인에 의해 함수의 스코프에 접근 가능
- 모듈함수가 반환한 객체는 함수들에 대한 참조만 가지며 내부 변수에 대한 접근은 불가능. 즉 내부변수는 캡슐화되어서 외부에서 접근할 수 없으며, 접근하고 싶으면 외부로 반환한 클로저 함수를 통해서만 접근 가능(클로저를 활용한 **모듈 패턴**)

JavaScript 모듈

- export, import
  - 모듈 안에 선언한 식별자(변수, 함수, 클래스)를 다른 모듈에서 접근할 수 있도록 하며, 모듈의 최상의 위치에 존재해야 함
  - 다른 모듈에서 내보낸 식별자(변수, 함수, 클래스)를 가져오며, 반드시 파일의 최상단에 위치해야 함.

실행 컨텍스트(Execution Context)

- 자바스크리트 코드를 실행할 때 필요한 정보를 저장하고 제공하는 환경(스코프의 정보를 담은 환경)
  - Lexical environment
  - Variable environment

HOF, Currying

- HOF(Higher Order Function)
  - 함수적 프로그래밍(Functional programming)시에 사용
  - 함수를 인자로 받아서 새로운 함수를 반환하는 함수
  - 유연하고 반복을 줄일 수 있는 코드 작성 가능
  - 하나 이상의 함수를 인자로 받아 함수를 결과로 리턴한다.
- Currying
  - 다중 인수 (혹은 여러 인수의 튜플)을 갖는 함수를 단일 인수를 갖는 함수들의 함수열로 바꾸는 것
  - 여러 개의 인자를 받는 함수를 단일 인자를 받는 함수의 체인을 이용하는 방식으로 바꾸는 것을 의미

```
const sum = function(x) {
  return function(y) {
    return x + y;
  }
}

console.log(sum(2)(5)); // 7
```

- 단일 인자를 받는 함수들을 연결. 인자의 수가 많으면 많아질수록 반환해야 하는 함수의 수도 많아짐(콜백 지옥)

HOF, Currying

- Curring 적용
  - 함수가 필요로 하는 인자 갯수에 도달하기 전까지는 함수를 반환하고 인자 갯수를 모두 충족시켰을 때 비로소 값을 반환
- Currying 사용 이유
  - 함수의 확장 용이
    - 곱하는 기능을 가진 함수를 사용하여 2배를 곱하는 함수와 3배를 곱하는 함수로 확장 가능
  - 중복 인자 사용 회피
    - 아래 100과 같이 중복 인자가 있을 경우,

```
volume(200,30,100) // 2003000l
volume(32,45,100); //144000l
volume(2322,232,100) // 53870400l
```

- 우측과 같이 currying으로 중복 인자 해결 가능

```
function volume(h) {
    return (w) => {
        return (l) => {
            return l * w * h
        }
    }
}

const hCylinderHeight = volume(100);
hCylinderHeight(200)(30); // 600,000l
hCylinderHeight(2322)(232); // 53,870,400l
```

HOC

- HOC(High Order Component)
  - 컴포넌트(react에서 코드 재사용의 기본 단위)를 재사용하기 위한 React 기술
  - 컴포넌트를 가져와 새 컴포넌트를 반환하는 함수
  - Redux의 connect 와 같은 React의 3-party 라이브러리
  - HOC는 입력된 컴포넌트를 수정하지 않으며 상속을 사용하여 동작을 복사하지도 않음.
  - HOC는 원본 컴포넌트를 컨테이너 컴포넌트로 *포장(Wrapping)*하여 *조합(compose)*함. HOC는 사이드 이펙트가 전혀 없는 순수 함수
  - 포장된 컴포넌트 새로운 props, data와 함께 컨테이너의 모든 props를 전달받으며 이 데이터들은 출력을 렌더링하는데 사용됨.