EliceAcademy

javascript 실행원리

javascript 내장객체

- javascript 에는 개발에 활용할 수 있는 객체들을 지원합니다
- 아래의 사이트를 통해 javascript 의 내장 객체들이 어떤것들이 있는지 확인할 수 있습니다.
- https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects
- 그 중 시간을 관리하는 Date 를 통해 내장객체를 어떻게 활용할 수 있는지 알아봅시다

EliceAcademy

시간을 반환해 주는 내장객체 Date

```
const now = new Date()

console.log(now)
console.log(typeof now)   ....1) 객체 타입 확인.

console.log('getFullYear:', now.getFullYear())
console.log('getMonth:', now.getMonth())
console.log('getDate:', now.getDate())
console.log('getDay:', now.getDay())
console.log('getHours:', now.getHours())
console.log('getMinutes:', now.getMinutes())
console.log('getSeconds:', now.getSeconds())

console.log(now.toDateString())
```

1) typeof 는 뒤의 변수가 어떤 타입의 자료인지를 확인해 주는 명령어이다

2)
Date 를 받아와서 각각의 함수에 접근을 해 전체의 년, 달, 시간등의 단위로 결과를 확인해 볼 수 있다

ElieAcademy

new 생성자

```
const now = new Date()   => DateConstructor

console.log(typeof now)  => object 타입으로 확인
```

1) Date 를 보면 DateConstructor 라고 알려주는 것을 확인할 수 있다
2) 타입 역시 object 즉 객체로 인식되는 것을 확인할 수 있다
3) new Date() 처럼 접근하는 것을 ' 생성자' 라고 부른다
4) 생성자는 코드 생산의 효율성을 높여준다. 마치 공장처럼

ElieAcademy

객체와 생성자의 비교

```
let user = {
    name: '홍길동',
    age: 30,
}
```

user 라는 객체를 생성.
오른쪽의 생성자와 사실상 같다.

```
function User() {
    this.name = '홍길동';
    this.age = 30;
}
```

생성자의 기본 형태 모습.

ElieAcademy

객체와 생성자의 비교

```
let park = {
    name: '박',
    age: 30,
}

let jung = {
    name: '정',
    age: 40,
}
```

단순 객체로 다양한 값을 표현하기 위한 준비형태

```
let park = new User('park', 30);
let kim  = new User('kim', 40);
```

생성자로 좀 더 간결하고 효율적으로 코드를 만들어 낼 수 있다

closure

- 디버깅을 통해 scope 에 다시 한번 살펴볼 수 있었다.
- closure 는 함수 안에서 다른 자식 함수를 선언하는 형태를 말한다.
- closure 를 활용하면 전역(global) 변수를 줄이고, 코드 재사용율 역시 높일 수 있다.
- 자식 함수에서 부모 함수의 변수에 접근할 수 있기 때문이다.
- 말은 너무 어려우니 디버깅을 통해 파악해 보자.

EliceAcademy

closure

```
<script>
        let a = 'a';

    function B(){
        let c = 'c';
        console.log(a,b,c);   ==> b 는 참조할 수 없는 값이라는 reference error 발생.
    }

    function A(){
        let b = 'b';
        console.log(a,b);
        B();
    }

    A();
</script>
```

closure

```
<script>

    let a = 'a';

    function A(){
        function B(){
            let c = 'c';
            console.log(a,b,c);
        }
        let b = 'b';
        console.log(a,b);
        B();
    }


    A();
</script>
```

함수안에 함수를 선언 함으로써, 정상적인 출력이 가능한 것을 확인할 수 있다.
이를 통해 함수의 호출 시점이 아닌 함수의 선언 시점이 scope 에 영향을 미친다는 것을 확인할 수 있다.

callback

- 함수가 온전히 실행된 뒤, 실행되는 함수.
- 자바스크립트의 이벤트 응답 체계를 명확히 하기 위해 필요하다.

EliceAcademy

callback – 의도하지 않은 호출

```
function normal(){
    let val = 2+3;
    console.log(val);
}

function addCallback(){
    setTimeout(()=>{
        let val = 1+1;
        console.log(val);
    },3000);
}

    addCallback();  ===> addCallback 의 결과인 2가 계산된 뒤 아래의 end 로그가 찍히 길 기대
    console.log('end'); ==> 하지만 실행하면 'end' 가 먼저 출력 후 2 가 출력
```

EliceAcademy

callback – 의도한 대로 순차적 호출

```
function normal(){
    let val = 2+3;
    console.log(val);
}

function addCallback(fn){
    setTimeout(()=>{
        let val = 1+1;
        console.log(val);
        fn();  ==> 전달받은 인자함수를 호출.
    },3000);
}

addCallback(function(){    => 인자로 함수를 전달하고 있다.
    console.log('end');
})
```