WEB STUDY

3 주차 *javascript (Fundamentals)*

목차

- 자바스크립트란?
- Type, Values , and Variable
- Scope
- Expression and Operator
- Statements
- 실습

JAVASCRIPT란?

JAVASCRIPT 란?

- 객체 기반의 스크립트 프로그래밍 언어이다.
- 웹 브라우저 내에서 주로 사용한다.
- 다른 응용프로그램의 내장 객체에도 접근할 수 있다.
- Node.js와 같은 런타임 환경과 같이 사이드 네트워크 프로그램에도 사용된다.

웹 브라우저에서 어떤 일을 하는가?

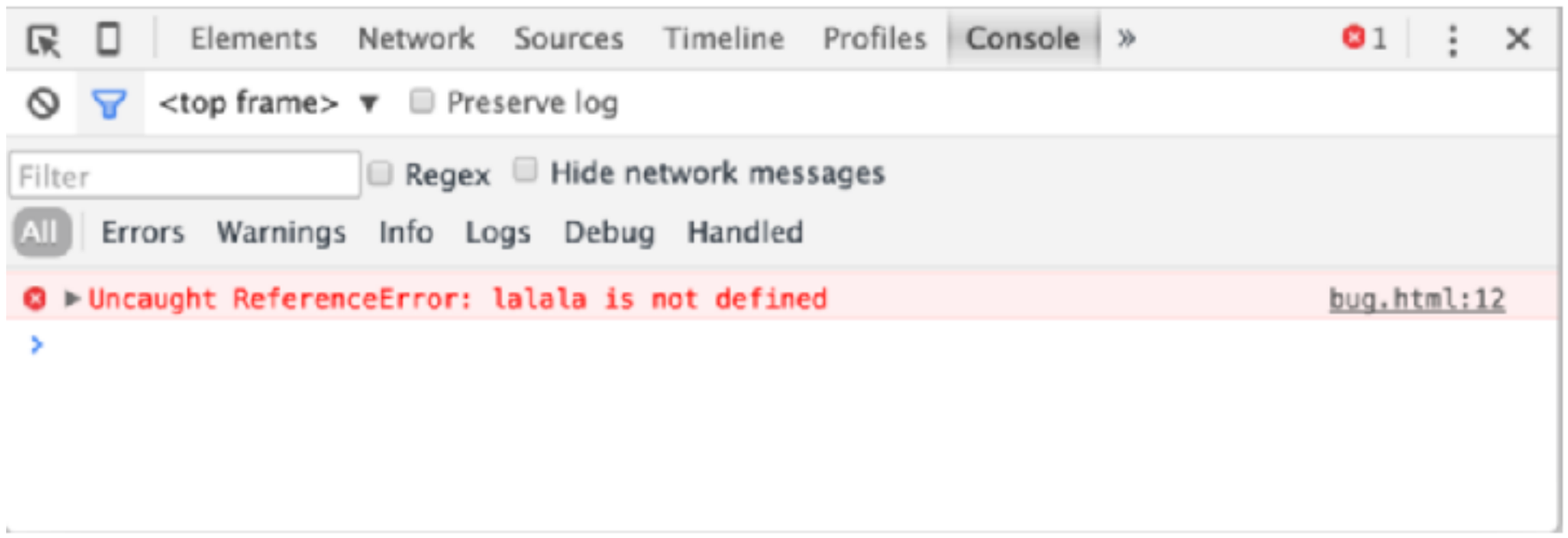
현재 자바스크립트는 편하고 간단한 프로그래밍 언어이다. 메모리나 cpu 같은 low-level접근을 제공하지 않는다. 브라우저 안에서 자바스크립트는 웹페이지를 다루는 일, 유저와 웹서버의 상호작용을 관련한 모든일을 할 수 있다.

예를 들어.

- HTML page 추가, content 변화, 스타일 수정
- 유저의 action의 반응, 마우스 클릭 또는 키보드 이벤트
- 서버의 request를 보내는 것, file 업로드, 다운로드
- 쿠키 생성
- Localstrage의 데이터를 기억

개발자 CONSOLE

- Chrome에서 지원하는 개발자 도구를 이용하여 javascript의 console을 사용할 수 있다.
- 만든 웹 페이지를 브라우저에 올릴 때, 버그가 있는 지도 확인할 수 있다.
- 현재 웹페이지의 **DOM** 객체에 접근도 가능하다.



IN-BROWSER JAVASCRIPT

- HTML file안에서 자바스크립트는 <script>…</script> 형식으로 자바스크립트를 명시해 줄 수 있다.

```
<!DOCTYPE HTML>
<html>

<body>

</body>

<script>
...
</script>

</html>
```

- 또한 자바스크립트는 코드 위에 ‘;’이 있어도 되고 없어도 된다. 하지만 같은 라인에 구별되는 코드라면 ‘;’를 이용하여 분할한다.

```
alert('hello'); alert('world');
```

< 같은 동작

```
alert('hello')
alert('world')
```

HELLO, WORLD!

➤ 아래 코드는 웹브라우저에 'Hello, World!'라는 알림을 보여준다.

```
<!DOCTYPE HTML>
<html>

<body>

  <p>Before the script...</p>

  <script>
    alert( 'Hello, world!' );
  </script>

  <p>...After the script.</p>

</body>

</html>
```

TYPE, VALUES, AND VARIABLE

TYPES, VALUES, AND VARIABLE

자바스크립트는 c언어와 java와는 다르게 모든 타입의 변수를 var 또는 let으로 선언할 수 있다. 변수 타입은 변수에 들어있는 데이터의 타입에 맞게 변환된다.

```
let message;

message = 'Hello' // store the string
```

➤ *'message'*라는 변수는 *string type*으로 변환된다.

*var*과 *let*말고도 *const*를 이용하여 변수를 선언할 수 있다. *Const* 같은 경우는 한번 초기화를 하면 안에 있는 변수의 값을 변경할 수 없게 된다.

```
const message = 'Hello'

message = 'World' // Error
```

SCOPE

자바스크립트에서 스코프란 어떤 변수들에 접근할 수 있는지를 정의한다. 크게 전역 스코프와 지역 스코프가 있다.

- Global Scope : 최상위 함수 밖에 선언된 변수는 전역 스코프로 정의되며, 모든 함수 안에서 이 변수를 접근할 수 있다.
- Local Scope: 함수 안에 선언된 변수는 지역 스코프로 정의되며, 선언된 함수 안에서만 변수 접근이 가능하다.

호이스팅 (HOISTING)

- 함수나 변수가 선언되면, 그 변수는 현재 스코프의 최상위 단으로 끌어 올라가게 된다.
- 예를 들어,

```
// This is the same as the one below
sayHello()
function sayHello () {
  console.log('Hello CSS-Tricks Reader!')
}
```

```
// This is the same as the code above
function sayHello () {
  console.log('Hello CSS-Tricks Reader!')
}
sayHello()
```

HOISTING

- 반면 함수가 함수 표현식(function expression)으로 선언되면, 함수는 호이스팅되지 않는다.

```
sayHello() // Error, sayHello is not defined
const sayHello = function(){
        console.log('function')
}
```

NESTED SCOPES

- 함수가 다른 함수 내부에서 정의되었다면, 내부 함수는 외부 함수의 변수에 접근할 수 있다.
- 반면, 외부 함수는 내부 함수의 변수에 접근할 수 없다.

```
function outerFunction () {
  const outer = 'I’m the outer function!'

  function innerFunction() {
     const inner = 'I’m the inner function!'
     console.log(outer) // I’m the outer function!
  }

  console.log(inner) // Error, inner is not defined
}
```

CLOSURES

- 함수 내부에 함수들을 클로저라고 부른다.
- 클로저는 차후에 외부 함수의 변수를 사용할 수 있기 때문에 대게 변환하여 사용한다.

```
function outerFunction () {
  const outer = 'I see the outer variable!'
  function innerFunction() {
    console.log(outer)
  }
  return innerFunction
}
outerFunction()() // I see the outer variable!
```

LEXICAL SCOPE

➤ 클로저를 사용하여 외부 변수의 life를 유지하여 사용할 수 있다.

➤ 이것을 Lexical Scope라고 부른다.

```
function outFunction(){
    let count=0
    return function innerFunction(){
        count++;
        console.log(count)
    }
}
let count = outFunction()
count() //1
count() //2
```

BLOCK SCOPE

Block Scope는 괄호 안에서의 변수 선언은 괄호 밖에 있는 변수에 영향을 끼치지 않는다.

var 과 let의 차이점

- let 은 Block Scope를 가진다.
- var은 Block Scope X

BLOCK SCOPE

```
let foo = 'bar1';
console.log(foo); // bar1

if (true) {
  console.log(foo); // bar1
  foo = 'bar2';
  console.log(foo) // bar2
}
console.log(foo); // bar2
```

> Use var

```
let foo = 'bar1';
console.log(foo); // bar1

if (true) {
  let foo = 'bar2';
  console.log(foo) // bar2
}
console.log(foo);  // bar1
```

> User let

EXPRESSIONS AND OPERATOR

EXPRESSIONS

- Array : var a = [1, 1+2, 4];
- Object : var p = { x: 2.3 , y : -1.2};
- Function : var square = function(x) { return x*x };
- Property Access Expression : expression.identifier or expression[identifier]
- Object Creation Expression : new Point(2,3)

ARITHMETIC OPERATORS

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus (division remainder)
++	Increment
--	Decrement

Operator	Example	Same as
=	x=y	
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y

COMPARISONS OPERATORS

Operator	Description
==	is equal to
===	is exactly equal to (value and type)
!=	is not equal
>	is greater than
<	is less than
>=	is greater than or equal to
<=	is less than or equal to

STRING OPERATORS

String 끼리 '+' 연산자를 통해 이어붙이기가 가능하다.

```
txt1 = "What a ";
txt2 = "wonderful world";
txt3 = txt1 + txt2 // "What a wonderful world"
```

STATEMENTS

CONDITIONAL STATEMENTS

- If
- If .. else
- If .. else if .. else
- Switch

LOOPS

- For
- While

```
var txt = "";
var person = {fname: "john", lname:"Done", age:25);

for (var x in person){
        txt = txt + person[x];
}
console.log(txt) // johnDone25
```

LOOPS

```
cars = ["BMW", "Volvo","Saab","Ford"];
for (var i=0; i<cars.length;i++)
{
        console.log(cars[i])
}
/*
BMW
Volvo
Saab
Ford
*/
```

실습

목표 : Scope에 대한 이해

Javascript를 통해 유저 추가, 삭제, 수정을 하는 시스템 만들어보기

- 유저의 정보는 이름만 가지고 있다.
- 추가할 때는 유저이름만 입력받는다.
- 수정할 때 바꿀 이름과 수정 이름을 입력받는다.
- 삭제할 때 삭제하고 싶은 이름을 입력받는다.
- 시스템이 끝나면 유저 이름으로 정렬하여 보여준다.

실습

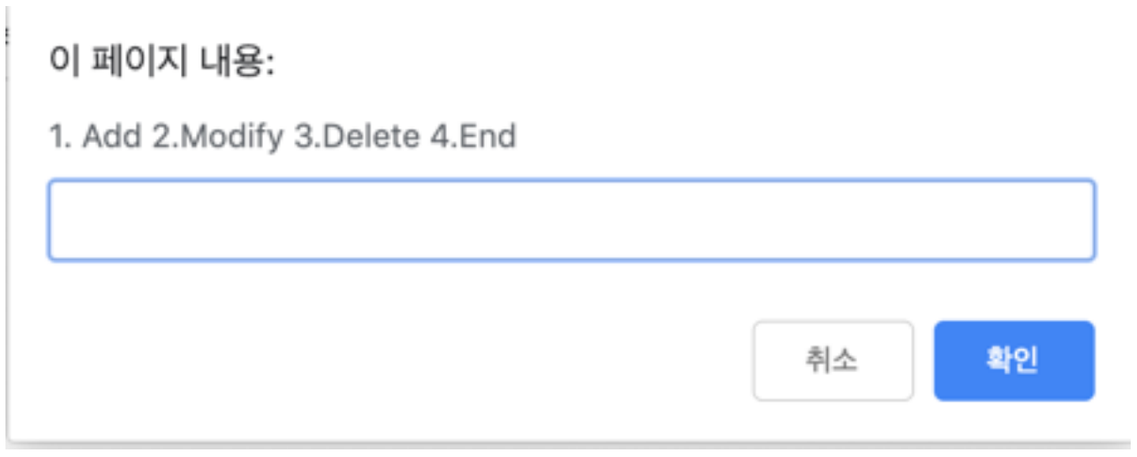
- 단 for문은 출력할 때 빼고는 사용하지 않는다.
- 유저를 찾을 때 : find method
- 유저를 삭제할 때 : splice method
- 유저를 정렬할 때 : sort method
- Lexical Scope를 적극 사용한다.

힌트

➤ 유저 :

```
function User(name){
    return {
        getName(){
            return name;
        },
        setName(newName){
            return name=newName
        }
    };
}
```

➤ 유저 인풋 : prompt 사용



결과

- console.log()를 사용해서 출력
- Add jiwon -> Add jinwoo