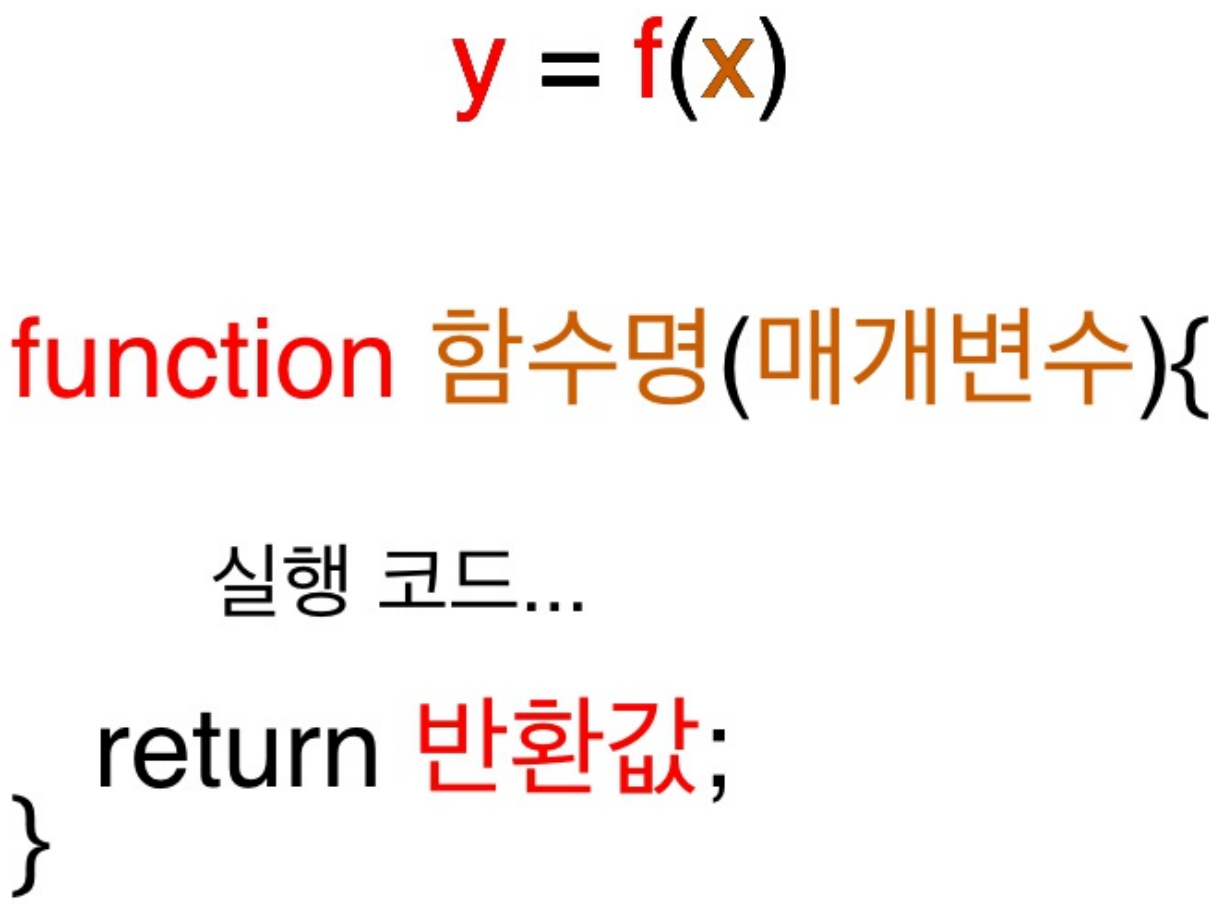
Javascript 함수

키워드 : 함수 (https://www.w3schools.com/js/js_functions.asp),
함수 선언 (https://www.w3schools.com/js/js_function_definition.asp),
함수의 스코프 (https://www.w3schools.com/js/js_scope.asp),
Const (https://www.w3schools.com/js/js_const.asp),
Let (https://www.w3schools.com/js/js_let.asp)

자바스크립트 코드를 작성할 때 논리적인 흐름을 제어하고, 하나의 기능적인 역할을 하는 코드를 코드 블록({})으로 묶어 개별적으로 관리하기 위해 함수(Function)라는 것을 선언하고 사용합니다.

함수 선언 및 호출(Definitions and Call)



우선 자바스크립트 함수(Function)는 수학에서의 함수와 닮아 있는 부분이 굉장히 많습니다. 예를 들어, y = f(x)와 같은 함수가 있을 경우 일반적으로 y라는 종속변수는 f라는 함수의 독립변수 x 값에 따라 다른 결과 값을 돌려받게 됩니다.

조금 더 이야기하면, y = ax + b라는 1차 함수는 x의 값에 따라 직선의 모양이 달라집니다. 또한, a의 값에 따라 직선의 기울기가 달라지고, b의 값에 따라 y 절편의 값이 달라지게 됩니다.

여기서 함수의 이름은 f로 대응됩니다. 또한, x의 값을 함수의 매개변수, 인자 혹은 파라미터(Parameter) 등의 용어로 지칭할 수 있고, a와 b의 값은 함수 코드 블록의 실행코드에서 사용되는 임시 변수(지역 변수)로 대응할 수 있습니다. 함수의 결과인 y의 값은 return 구문이라고 할 수 있습니다.

단, 수학에서의 함수는 정의역 집합(X)이 공역 집합(Y)에 무조건 대응해야 하지만, 프로그램 코드 상에서는 반환 값(return 값)이 없는 빈 반환 값(void)형태의 함수를 작성할 수 있습니다.

함수 선언 기본 형식

아래 코드는 함수를 선언하는 기본적인 코드 예시를 나타냅니다.

```
//코드 선언 방식
//(주의)함수의 인자는 var 키워드를 붙이지 않습니다!
function 함수명(인자, 인자...){
    실행코드
}

//1번 (Definition)
function sum(x, y){
    return x + y;
}
//2번 (Call)
var result = sum(1, 1);
console.log(result);//2
```

위 코드 예시에서 1번은 함수를 선언하는 코드입니다. 우리가 어떤 함수를 1번처럼 선언만 하고 호출하지 않으면, 해당 함수는 사용되지 않습니다. 2번 코드의 내용처럼 함수를 호출하는 작업이 진행되어야 해당 함수가 코드 블록({})의 실행 코드를 실행하고, 결과를 반환(return) 할 수 있습니다. (단, 선언과 동시에 자동으로 호출되는 함수가 있습니다. 그 내용은 아래에서 더 다루도록 하겠습니다.)

또 다른 기본 함수 선언 방식은 강의에서 살펴본 익명 함수(Anonymous function) 선언 방식입니다. 이름이 없는 함수이기 때문에 익명 함수라고 불리우며, 함수를 변수에 할당하고 변수의 이름으로 호출할 수 있었습니다.

```
var sum = function(x, y){
    return x + y;
};
console.log(sum(1, 1), typeof sum); //2, function
```

함수의 스코프(Scope)

함수를 이야기할 때 꼭 빠지지 않고 등장하는 굉장히 중요한 내용이 있습니다. 바로 함수의 스코프(활동 범위)입니다.

함수는 크게 두 가지 스코프 영역을 갖습니다.

> Local scope, Global scope

스코프는 우리가 함수의 실행 코드를 정의할 때 함수에서 사용된 변수의 사용 범위를 말합니다. 함수 안에서 사용된 변수는 지역 변수(Local scope)는 함수 안에서만 사용 가능하며, 다른 곳에서 사용하려고 하면 에러를 발생시킵니다. 반대로 전역 변수(Global scope)로 선언된 변수는 함수 안, 바깥에서 모두 사용 가능한 활동 범위를 갖습니다.

아래 코드는 지역 스코프와 전역 스코프의 차이를 보여주는 예시입니다.

```
var x = 1; //전역 변수 선언
function sum(y){ //(주의)인자(Parameter)도 지역 변수 취급!
    var z = 1; //지역 변수 선언
    return x + y + z;
}
console.log(sum(1)); //3 에러 없음!
console.log(y, z); //에러 발생!
```

위의 코드에서 변수 x는 함수 바깥쪽에서 선언되었습니다. 때문에 전역 변수로 취급됩니다. 반대로, 변수 y와 z는 함수 안쪽에서 선언되었습니다. 이런 경우 함수 안에서만 사용 가능하며, 변수 자체의 이름으로 값에 접근하려고 하면 에러가 발생하게 됩니다.

그렇다면, 변수의 이름이 똑같은 두 개의 변수가 전역 변수, 지역 변수로 각각 선언되어 있다면 어떻게 될까요?

```
var x = 0, y = 0;
function sum(x){
    var y = 10;
    return x + y;
}
console.log(sum(10)); //20
```

결론적으로, 전역 변수보다 함수 안에서 선언한 지역 변수가 더 높은 우선순위를 갖습니다. 때문에 같은 이름을 갖는 변수가 중복 선언되어 있다고 해도 함수 안에서 선언된 변수를 가지고 실행코드가 동작하게 됩니다.

전역 변수의 사용시 의도적으로 전역 변수를 선언하여 사용하는 경우가 아니라면, 전역 변수의 스코프를 잘 이해하며 사용해야 합니다.

마지막으로, 우리가 선언한 변수는 생명 주기라는 것을 갖습니다. 함수 안에서 선언한 지역 변수(Local scope) 같은 경우에는 함수가 호출되고 종료하는 시점에 메모리에서 함께 소멸됩니다.

반대로, 함수 바깥쪽에서 선언한 전역 변수(Global scope)의 경우에는 브라우저가 닫히는 경우에 함께 소멸됩니다. (단, 간혹 같은 페이지에서 새로운 페이지를 로딩하는 경우에 소멸되지 않고 메모리에 남아있을 수 있습니다.)

다양한 함수 선언 및 호출 방법

위에서 우리는 함수의 선언 방법과 변수의 활동 범위에 대해서 살펴보았습니다. 사실, 자바스크립트 코드를 작성할 때 위의 규칙대로 동작하지 않는 경우가 꽤나 많습니다. 이번 절에서는 그러한 부분에 대해서 살펴보도록 하겠습니다.

우선, 함수의 선언 위치입니다. 예를 들어, 아래와 같은 코드가 있다고 가정해보겠습니다.

```
init();
console.log(x); //10

function init(){
    x = 10;
}
```

위 코드가 이상하게 생각되신다면 위에서 이야기한 함수의 스코프에 대해서 잘 이해하셨다고 할 수 있습니다. 위 코드는 우리가 기존에 알고 있던 두 가지 정의에 위배됩니다.

첫 째는 변수를 선언할 때 사용했던 var 키워드 없이 변수 x를 선언함과 동시에 10을 할당하고 있습니다.

둘 째는 함수 안에서 선언된 x 변수를 함수 바깥쪽에서 접근했지만 10이라는 값이 잘 나오고 있습니다.

위 코드처럼 선언되지 않은(var 키워드 없이) 변수를 사용하는 경우 해당 변수는 곧바로 전역 변수 취급을 받게 됩니다. 이는 코드 작성 시 찾아내기 어려운 오류를 발생시킬 우려가 있습니다. 때문에 우리가 잘 알고 있는 형식대로 변수 선언 시에는 var 키워드를 사용하고 초기화 후 사용해주세요!

자바스크립트는 비교적 엄격한 문법 체계를 갖는 java, C와 같은 프로그래밍 언어와 다르게 생각보다 오류를 발생시키지 않는? 느슨한 문법 체계를 갖고 있습니다. 다른 프로그래밍 언어에서 위와 같은 코드를 작성하면 바로 문법 오류(Syntax Error)를 발생시킬 것입니다.

우리는 우리를 믿지 못하기 때문에? 코드 작성 시 엄격하게 문법 오류 등을 체크해서 코드를 작성하고 싶은 생각이 들곤 합니다. 상대적으로 자바스크립트는 우리에게 너무 관대한 프로그래밍 언어입니다.

이러한 점을 개선할 수 있는 방법은 바로 `Strict Mode`를 사용하는 것입니다. 실행 코드 작성 전에 `"use strict";` 혹은 `'use strict';` 형태로 작성하면 엄격하게 문법 체크를 진행할 수 있습니다.

```
'use strict';
init();
console.log(x);

function init(){
    x = 10; //x is not defined 에러 발생!
}
```

우리는 위에서 함수를 선언(Definition)하고 호출(Call)하는 두 가지 기본 과정을 통해서 함수를 사용할 수 있다고 알아보았습니다. 자바스크립트에선 즉시 실행(Self-invoking) 함수가 존재합니다. 즉시 실행 함수는 이름 그대로 따로 호출하지 않더라도 코드 구문을 만나는 순간 바로 실행되는 함수를 말합니다.

아래 코드는 즉시 실행 함수를 작성하는 코드 예시입니다.

```
//코드 선언 방식
(function(){
    실행코드...
})();

(function () {
    for(var i = 0; i < 5; ++i){
        console.log(i); //0, 1, 2, 3, 4
    }
})();
```

문법이 조금 생소할 수 있습니다. 우리가 익명 함수를 선언할 때 처럼 이름이 없는 `function(){}` 키워드를 소괄호`()`로 묶고 마지막에 `();`를 붙여준 형태입니다. 즉, 더 엄밀히 말하면 위 코드는 익명 즉시 실행 함수라고 할 수 있습니다.

이처럼 따로 호출하지 않고 바로 실행되어야 할 코드가 있을 경우 즉시 실행 함수를 사용 할 수 있습니다.

Const, Let 키워드

우리가 HTML에서 버전이 있다는 것을 알아보았듯, 자바스크립트에도 버전이 있습니다. 자바스크립트를 공부하다 보면 "ES5", "ES6" 등의 이름을 보신 적이 있을 겁니다. 이번 절에선 "ES6"(정식 명칭 ECMAScript 2015)에서 등장한 새로운 키워드인 const, let에 대해서 간단히 알아보겠습니다.

우리는 위에서 함수의 스코프에 대해서 이야기했습니다. `const`와 `let` 키워드는 변수의 스코프와 관련이 깊은 내용이라고 할 수 있습니다.

우선, var 키워드를 통한 변수 선언 방법에 대해서 살펴보겠습니다.

```
//1번 코드
var x = 0;
var x = 10;
console.log(x); //10

//2번 코드
var y = 0;
var y;
console.log(y); //10
```

위의 코드는 var 키워드를 이용한 자바스크립트 변수 선언의 문제점 아닌 문제점을 보여줍니다. var 키워드를 이용해서 변수를 선언하면 똑같은 이름의 변수를 중복 선언해도 아무런 문제가 없습니다. 그렇다면, const와 let 키워드는 어떤 차이가 있을까요?

아래는 const 키워드와 let 키워드를 이용한 선언 예시입니다.

```
//1번 코드
let x = 0;
x = 10;
let x = 10; //'x' has already been declared 에러 발생!
var x = 10; //'x' has already been declared 에러 발생!

//2번 코드
const x; //Missing initializer in const declaration 에러 발생!
const x = 0;
x = 10; //Assignment to constant variable 에러 발생!
```

우선, 1번 코드부터 살펴보겠습니다. let 키워드를 통해 변수를 선언하면, 처음 변수를 선언하고 같은 이름의 변수를 var, let 키워드를 통해 재 선언 하는 것이 불가능합니다.

2번 코드인 const 키워드는 상수를 나타내는 키워드라고 생각하면 이해하기 쉽습니다. const 키워드로 선언된 변수는 선언과 동시에 할당(초기화)이 이루어져야 합니다. 변수를 먼저 선언하고 중복으로 같은 이름의 변수를 선언하는 것이 불가능합니다. 또한, 한 번 값이 초기화되면 값을 변경할 수 없습니다.

위에서 이야기한 const, let 뿐만 아니라, 강의에선 다루지 않은 ES6에서 새롭게 등장한 여러 가지 내용들이 더 있습니다. 추후에 따로 내용을 할애해서 그 부분에 대해서도 정리해드릴 수 있도록 하겠습니다. :)

생각보다 글이 길어졌네요..^^ 그만큼 자바스크립트 코드를 작성할 때 함수와 스코프에 대한 개념은 굉장히 중요한 내용입니다.

프로그램 코드가 길어짐에 따라 수월한 유지 보수를 위해서라도 코드를 스크립트 형식으로 한 줄 한 줄 작성하는 것보다는 똑같은 실행 코드를 갖는 함수를 정의하고, 인자만 바꿔주며 결과를 반환하는 형태로 작성하는 것이 효율적인 경우가 많습니다.

단, 모든 코드를 함수로 선언하고 작성하려고 하다 보면 오히려 코드 분석이 어려워질 우려가 있습니다. (적절한 완급 조절이 필요하겠죠..? ^^)

처음 공부할 때는 함수를 선언하고 호출하는 것 자체에 어려움이 있을 수 있습니다. 특히 자바스크립트 언어는 함수 관련 내용도 굉장히 방대합니다. (위의 내용만으로도 부족한 부분이 많습니다.) 위에서 장황하게 작성한 모든 글이 처음 자바스크립트를 접하신 분들에게 지금 당장은 도움이 될지 모르겠지만, 추후에 더 깊은 학습을 위한 초석이 될 수 있으면 좋겠습니다! :)