자바스크립트 let,const,var

이제 막 자바스크립트를 공부하신 분들이라면, let 과 const는 차이를 알기 쉽지만, let 과 var 사이에서 무슨 차이가 있는지 궁금하신 분들이 많을 것 같아요. 저 또한 그랬던 사람이고 쉽게 이해하는 방법을 터득해서 블로그에 기록해보려고 합니다. ㅎㅎ

그래서 사실 이번 글의 주제는 let와 const, var 를 다 아우르기 보단 **let과 var의 차이**에 대해 중점적으로 써보려고 합니다.

1. 타입추론형 언어

(타입추론을 알고계시는 분은 이 문단을 건너 뛰셔도 좋습니다.)

자바스크립트에는 데이터들의 고유한 타입이 존재합니다. 예를 들어 숫자는 정수든 실수 구분 없이 모두 number 타입이며, 문자열을 string, 배열은 array, true와false는 boolean, 객체는 object 입니다. 자바스크립트는 이러한 변수의 타입을 var, let ,const를 앞에 붙여주면 자동으로 리터럴(문자열이면 "문자열", 숫자면 1,2, ... 각 변수마다 고유한 형식이 있죠? 이걸 리터럴이라고 합니다.)을 확인하여 타입을 지정해줍니다. 그래서 자바스크립트나 파이썬을 **타입추론형 언어**라고 말하기도 한답니다.

cf). C++에는 auto 키워드가 있고, Java도 10부터 var를 지원합니다. C#에도 var가 존재합니다.

2. const

const는 constant 의 약자로써 우리 나라 말로는 **항상 같은 수**라는 뜻을 가진 **상수**가 되겠습니다.

즉, **한 번 정해지면, 프로그램이 끝나기 전까지 변경시킬 수 없다**는 의미입니다.

간단한 예제를 통해 알아볼까요?

```
const pi = 3.14 // 가능
```

```
const pi // 오류, 초기화가 반드시 필요합니다.
```

```
const pi = 3.14
pi = 3.141592 // 오류, 한 번 초기화 된 값은 바꿀 수 없습니다.
```

위에서 살펴볼 수 있듯이, const는 반드시 초기화가 필요하고 초기화 된 이후에는 그 값을 변경 시킬 수 없다는 성질을 갖고 있습니다.

이를 통해 얻게 된 **이점**이 뭘까요?

ES6(자바스크립트의 원래 이름인 ECMAscript 버전 6를 줄인 말) **이전의 자바스크립트**에는 변수 키워드로 var만 존재했었는데요. 이 때는 상수의 개념이 없어서 프로그래머가 값이 변경되지 않게 하려면 따로 코딩을 해줘야했습니다. 정말 번거로웠죠.

하지만, ES6에 const와 다음에 소개할 let이 등장하면서, 이러한 불편함이 다소 해결되었습니다. 특히, 변수명 앞에 const라고 명시함으로써 이 변수는 변하지 않는 값이라는 것을 **한 눈**에 알아볼 수 있게 되어 **가독성**도 좋아졌습니다.

마지막 특징으로 const는 블록 스코프를 갖는다란 말을 많이 쓰는데요. 이 특징은 let에 대해 소개하고 나서 같이 설명드리겠습니다.

3. let

let은 영어의 그 let 이 맞습니다. 영영사전에서 let의 뜻을 찾아보면, 다음과 같이

1. allow, permit

허락하다. 의 의미로 쓰이는 군요!

스택오버플로우의 표현을 빌리자면,

hey computer, can you please **let** this 'variable' **allow** to "String"? 이라는 의미에서 지어진 이름이라는 걸 알 수 있는데요.

```
let variable = "string"
```

정리하자면, 컴퓨터야 variable 이라는 변수를 앞으로 "string" 이라고 하자. 라는 의미로 해석할 수 있겠습니다. ㅎㅎ

let 키워드는 변수에 값을 대입하게 해줍니다. 그리고 const와 달리 반드시 값을 초기화 해줄 필요는 없으며, 언제든지 수정할 수 있습니다.

4. 스코프(scope, 유효, 참조범위)

자 이제 const와 let의 공통점을 설명해드릴게요!

const는 값을 변경할 수 없고 let은 값을 수정할 수 있다고 말했습니다. 그런데 이 둘 사이에 공통점이 존재하는데.

이를 공부하기 전에 먼저, **스코프**란 무엇인지 부터 짚고 넘어가겠습니다. 아마 제 블로그 글을 찾으신 분들은, 이 스코프 때문에 헷갈려서 찾아오신 분들이 많을 거라 생각되네요. 저 또한 그랬기 때문에 ㅎㅎ.. 자 이제 공부하러 가시죠!

> 스코프란 {} 이 중괄호를 의미합니다.여러분이 많이 하시는 슈팅게임의 총에 달려있는 동그란 망원경과도 같은 뜻인데요. 스코프를 보면, 동그란 부분 바깥은 전부 검게 되어 있어서 안 보이잖아요? 자바스크립트에서도 이 스코프 안에 선언된 변수들은 스코프 밖에서 참조할 수가 없습니다. 다시 말해서 접근이 불가능하다는 말이에요.
>
> 예제를 통해 확인해봅시다.

```
let test1 = "test1"
{
    let test2 = "test2"
}
console.log(test1, test2)
```

위의 문장에서 중괄호{} 를 스코프라고 한다고 했죠? 이 스코프 안에 있는 친구들은 스코프 바깥 영역에서는 참조할 수 없다고 했습니다. 그래서 console 창에 test2 를 출력하려고 하면 오류가 나게 됩니다!

자 이제 스코프에 대해서 알게 되었는데. 이건 사실 다음에 나올 function scope와 block scope 이해하기 위해서 필요한 추진력을 얻기 위함이었습니다. ㅋㅋ 다시 가시죠

1. Block Scope

   위에서 봤던 중괄호 {} 안의 범위를 스코프라고 했었는데. 다음에 소개할 Function Scope 와의 차이를 비교하기 위해서 앞으로 블록 스코프라고 표현하겠습니다. 그리고 사실, 블록 스코프는 함수 스코프를 포함하는 용어랍니다.

   블록 스코프 ⊃ 함수 스코프

2. Function Scope

   함수 스코프는 {} 앞에 function() 이 붙었을 때, 함수 스코프라고 부르게 됩니다. 그냥 중괄호만 있을 때는 블록 스코프, {} 앞에 function()이 붙으면 함수 스코프. 아시겠죠?

자, 그럼 이제 코드로 비교해볼까요?

```
// 1. 함수 스코프
function funcScope() {
    console.log("여기 안에 들어오는 건 함수 스코프 영역!")
}

{
```

```
    console.log("여기 안에 들어오는 건 블록 스코프 영역1!")
}

for(let i = 0; i < 10; i++) { // 블록스코프!
    if(true) { // 나도 블록스코프!
        consol.log(i)
    }
}
```

어떠신가요? function() 이 중괄호 앞에 붙지 않으면 나머지 중괄호는 모두 block scope라고 부른다는 사실! 아시겠쥬?

5. let과 const의 Scope

let과 const는 다음에 소개할 var가 함수 스코프인 것과 달리, **블록 스코프** 입니다! (**주의**: 블록스코프와 함수스코프는 반대가아니라 **포함관계**라는 사실에 유의하세요!)

블록 스코프를 다른 말로 **지역변수**라고 표현하기도 합니다.

그래서 {} 밖을 벗어나면 참조할 수 없습니다. 즉 자기 울타리 안에서만 놀고 바깥에선 참견할 수 없다는 것이죠.

예제를 통해 알아볼까요?

```
if(true) {
    let name = "Hong"
    {
        console.log(name) // "Hong"
    }
}
console.log(name) // Error
```

위의 명령어에서 마지막 줄 log는 에러가 난다고 했는데. 그 이유는 if문 안에 있는 {} 블록 스코프 안에서 if 문의 } 가 아직 닫히지 않은 상태이기 때문입니다.

하지막 마지막줄은, } 가 닫혀버려서 접근할 수가 없는 것이죠.

const도 마찬가지입니다.

자 여기서 퀴즈, 다음 for문은 에러가 날까요 안 날까요~?

```
for(let i = 0; i < 2; i++) {
    const j = i
    console.log(j)
}
```

const 가 있어서 새로운 값이 할당되니까 에러가 날 것이다! 라고 착각하기 쉽습니다.

하지만 위의 문장은 사실 아래와 같은 문장이죠?

```
{
    let i = 0
    const j = i
    console.log(j)
}
{
    let i = 1
    const j = i
    console.log(j)
}
```

스코프가 다릅니다. let과 const는 스코프가 끝나면, 내부 변수들은 지워진다고 했죠? 그래서 새로 정의할 수 있는 겁니닷!

그런데, let의 경우 한 가지 중요한 성질이 더 있어요!

```
if(true) {
    let a = 1
    console.log(a) // 1
    let a = 2
    console.log(a) // 오류!
}
```

위의 경우, if 문 안에 a 라는 이름으로 2번 선언 되어버렸군요.

이럴 경우 자바스크립트는 에러가 납니다!

하지만 다음에 소개할 var에서는 그렇지 않은데요.

이점 기억하시고 다시 가봅시다.

6. var

ES6가 나와서 let과 const가 등장하기 전까지 자바스크립트는 변수 키워드가 var뿐이었습니다. 이 친구는 위의 두 친구들과 달리 함수 스코프라는 특징이 있습니다. 그래서 함수 안에서만 참조할 수 있고 함수 바깥에서 참조하려고 하면 오류가 납니다.

여기까진 이해하셨죠?

자 이제 많은 분들이 헛갈려 하시는 내용입니다.

함수스코프의 경우 함수가 아닌 블록 스코프에 선언 되었을 경우 호이스팅 이라는 성질 때문에 전역변수로 작용하게 됩니다.

> 호이스팅(Hoisting: 감아 올리다.)
>
> 스코프 안에서 선언 된 변수는 어느 위치에 있든간에 결국 스코프의 시작지로 선언 코드가 옮겨가게 됩니다. 다음과 같이 말이죠. 그리고 **자바스크립트의 모든 변수는 호이스팅됩니다.** 감아올린다는 표현이 맞는 것 같기도 하네요 ㅎㅎ
>
> ```
> for(var i = 1; i <= 10; i++) {
>     console.log(i)
> }
> console.log(i)
> ```
>
> 자 여러분, 마지막 줄의 결과가 어떻게 될 것 같나요?

```
var i
...
for(i = 1; i <=10; i++) {
    console.log(i)
}
console.log(i)
```

> var는 좀전에 **함수 스코프**라고 했습니다. for문은 함수가 아니죠? 그래서 바깥으로 나가서 선언되는 겁니다. 그래서 **마지막 줄에서는 i 는 11로 출력이 됩니다.**

이처럼 함수가 아닌 곳에 선언 되어버리면, 전역변수로 작동하니 상당히 골치가 아팠습니다. 그래서 let과 const가 등장한것이죠! 그러니 되도록이면 var 대신 let과 const를 사용하도록 합시다!

정리

ES6에 var의 모호함을 해결하기 위해 const와 let이 등장했다.

모든 변수는 Hoisting 된다.

const는 선언과 정의를 반드시 동시에 해줘야한다.

var 대신 let을 쓰자.

정도로 정리할 수 있을 것 같습니다.

궁금하시거나 잘못 된 부분, 오타 지적은 언제든 환영합니다!

긴 글 읽어주셔서 감사합니다. ㅎㅎ