4장 변수

변수란 무엇인가?

컴퓨터는 CPU를 사용해 연산하고, 메모리를 사용해 데이터를 기억한다.
하지만 사람은 계산과 기억을 모두 두뇌에서 한다.

메모리는 데이터를 저장할 수 있는 메모리 셀의 집합체이다.
메모리 셀 하나의 크기는 1바이트 (8bit)이다.
데이터 버스의 구조에 따라 CPU가 하나의 instruction을 처리할 때 데이터를 읽어들이는 단위가 32bit, 64bit로 나뉜다.

자바스크립트는 개발자의 직접적인 메모리 제어를 허용하지 않는다.

만약 자바스크립트 개발자의 직접적인 메모리 제어를 허용하더라도 문제가 있다.

값이 저장될 메모리 주소는 코드가 실행될 때 메모리의 상황에 따라 임의로 결정된다. (운영체제에 의해)

변수(variable) : 하나의 값을 저장하기 위해 확보한 메모리 공간 자체 또는 그 메모리 공간을 식별하기 위해 붙인 이름

변수에 여러 값을 저장하는 방법

변수는 하나의 값을 저장하기 위한 메커니즘이다.
여러 개의 값을 저장하려면 여러 개의 변수를 사용해야 한다.

단, 배열이나 객체 같은 자료구조를 사용하면 관련이 있는 여러 개의 값을 그룹화 해서 하나의 값처럼 사용할 수 있다.

자바스크립트에도 파이썬의 딕셔너리와 리스트와 같은 자료구조가 존재한다. (object, array)

식별자

변수 이름을 식별자라고도 한다. 식별자는 어떤 값을 구별해서 식별할 수 있는 고유한 이름을 말한다.
값은 메모리 공간에 저장되어 있다. 식별자는 메모리 공간에 저장되어 있는 어떤 값을 구별해서 식별해낼 수 있어야 한다.
이를 위해 식별자는 값이 저장되어 있는 메모리 주소를 기억해야 한다.

변수 선언

변수 선언이란, 변수를 생성하는 것을 말한다.

자세히 말하면, 값을 저장하기 위한 메모리 공간을 확보하고 확보된 메모리 공간을 변수 이름과 연결해서 값을 저장할 수 있게 준비하는 것이다.

자바스크립트에서 변수를 사용하려면 반드시 선언해야 한다.

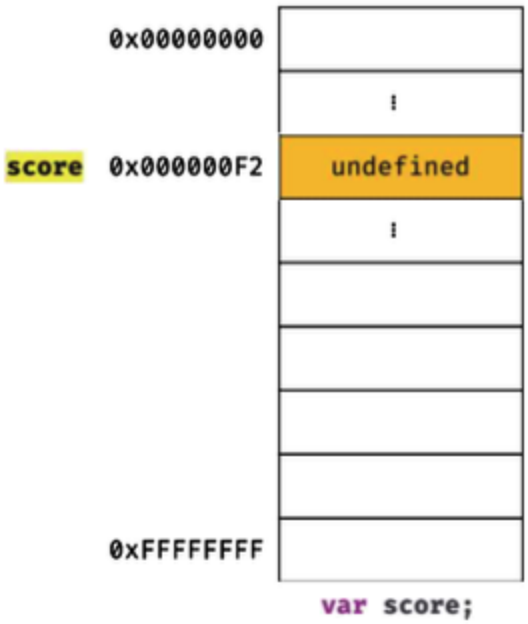
변수를 선언할 때는, var, let, const 키워드를 사용한다.
ES6에서 let, const 키워드가 도입되기 전까지 var키워드는 자바스크립트에서 변수를 선언할 수 있는 유일한 키워드였다.

var의 단점
var 키워드의 여러 단점 중에서 가장 대표적인 것이 블록 레벨 스코프를 지원하지 않고 함수 레벨 스코프를 지원하다는 것이다. 이로 인해 의도치 않게 전역 변수가 선언되어 심각한 부작용이 발생하기도 한다.
ES6에서 let과 const를 도입한 이유는 var 키워드의 여러 단점을 보완하기 위해서이다.
let과 const가 도입되었다고 해서, var이 폐기된 것은 아니다.
레거시 프로젝트에서 var을 이용해서 구현되어 있을 것이며, ES6 이후 사양을 따른다 하더라도 권장하지는 않지만 var 키워드를 사용할 수 있다.

키워드
키워드는 자바스크립트 코드를 해석하고 실행하는 자바스크립트 엔진이 수행할 동작을 규정한 일종의 명령어다.
자바스크립트 엔진은 키워드를 만나면 자신이 수행해야 할 약속된 동작을 수행한다.

만약, 아래 코드처럼 var 키워드를 사용하여 변수를 선언했다고 하자.

```
var score;
```



변수를 선언한 이후, 아직 변수에 값을 할당하지 않았다. 따라서 변수 선언에 의해 확보된 메모리 공간은 비어 있을 것으로 생각할 수 있으나, 확보된 메모리 공간에는 자바스크립트 엔진에 의해 undefined 라는 값이 암묵적으로 할당되어 초기화 된다.

undefined

undefined는 자바스크립트에서 제공하는 원시 타입의 값(primitive value)이다.

자바스크립트 엔진은 변수 선언을 다음과 같은 2단계에 거쳐 수행한다.

- 선언 단계 : 이름을 등록해서 자바스크립트 엔진에 변수의 존재를 알린다.
- 초기화 단계 : 값을 저장하기 위한 메모리 공간을 확보하고 암묵적으로 undefined를 할당해 초기화 한다.

변수 이름은 어디에 등록되나?

변수 이름을 비롯한 모든 식별자는 실행 컨텍스트에 등록된다.

실행 컨텍스트는 자바스크립트 엔진이 소스코드를 평가하고 실행하기 위해 필요한 환경을 제공하고 코드의 실행 결과를 실제로 관리하는 영역이다.

자바스크립트 엔진은 실행 컨텍스트를 통해 식별자와 스코프를 관리한다.

`변수 이름과 변수 값은 실행 컨텍스트 내에 키/값 형식인 객체로 등록되어 관리된다.`

var 키워드를 사용한 변수 선언은 선언 단계와 초기화 단계가 동시에 진행된다.

일반적으로 초기화란 변수가 선언된 이후 최초로 값을 할당하는 것을 말한다.

var 키워드로 선언한 변수는 undefined로 암묵적인 초기화가 자동 수행된다.

따라서 var로 선언한 변수는 어떠한 값도 할당하지 않아도 undefined라는 값을 갖는다.

만약 초기화 단계를 거치지 않으면, 확보된 메모리 공간에는 이전에 다른 애플리케이션이 사용했던 값이 남아있을 수 있다. → 이러한 값을 쓰레기 값이라 한다.

변수를 사용하려면 반드시 선언이 필요하다.

변수뿐만 아니라 모든 식별자가 그렇다.

변수 선언의 실행 시점과 변수 호이스팅

```
console.log(score); // undefined

var score;
```

위 코드를 보면, 변수 선언문 보다 변수를 참조하는 코드가 앞에 있다. 자바스크립트 코드는 인터프리터에 의해 한 줄씩 순차적으로 실행되므로 console.log(score);가 가장 먼저 실행되고 순차적으로 다음 줄에 있는 코드를 실행한다.

따라서 `console.log(score);` 가 실행되는 시점에는 아직 score 변수의 선언이 실행되지 않았으므로 참조 에러가 발생할 것처럼 보인다.

하지만 참조 에러가 발생하지 않고, undefined가 출력된다.

그 이유는 변수 선언이 소스코드가 한 줄씩 순차적으로 실행되는 시점, 즉 런타임이 아니라 그 이전 단계에서 먼저 실행되기 때문이다.

자바스크립트 엔진은 소스코드를 한 줄씩 순차적으로 실행하기 전에, 먼저 소스코드의 평가 과정을 거치면서 소스코드를 실행하기 위한 준비를 한다. 이때 소스코드 실행을 위한 준비 단계인 소스코드의 평가 과정에서 자바스크립트 엔진은 변수 선언을 포함한 모든 선언문(변수 선언문, 함수 선언문 등)을 소스코드에서 찾아내 먼저 실행한다.

즉, 자바스크립트 엔진은 변수 선언이 소스코드의 어디에 있든 상관없이 다른 코드보다 먼저 실행한다.

이처럼 변수 선언문이 코드의 선두로 끌어 올려진 것처럼 동작하는 자바스크립트 고유의 특징을 변수 호이스팅(variable hoisting)이라고 한다.

값의 할당

변수에 값을 할당할 때는 할당 연산자 =을 사용한다. 할당 연산자는 우변의 값을 좌변의 변수에 할당한다.

자바스크립트 엔진은 변수 선언과 값의 할당을 하나의 문으로 단축 표현해도 변수 선언과 값의 할당을 2개의 문으로 나누어 각각 실행한다.

이때 주의할 점은 변수 선언과 값의 할당의 실행 시점이 다르다는 것이다.

변수 선언은 소스코드가 순차적으로 실행되는 시점인 런타임 이전에 먼저 실행되지만 값의 할당은 소스코드가 순차적으로 실행되는 시점은 런타임에 실행된다.

```
console.log(score); // undefined

var score; // 1 변수 선언
```

```
score = 80; // 2 값 할당

console.log(score); // 80
```

변수 선언 1은 런타임 이전에 먼저 실행되고 값의 할당은 런타임에 실행된다. 따라서 score 변수에 값을 할당하는 시점에는 이미 변수 선언이 완료된 상태이며, 이미 undefined로 초기화되어 있다.

따라서 score 변수에 값을 할당하면 score 변수의 값은 undefined에서 새롭게 할당한 숫자 값 80으로 변경된다.

```
console.log(score);

var score = 80;

console.log(score);
```

변수의 선언과 값의 할당을 하나의 문장으로 단축 표현해도 자바스크립트 엔진은 변수의 선언과 값의 할당을 2개의 문으로 나눠서 각각 실행한다.

따라서 변수에 undefined 가 할당되어 초기화되는 것은 변함이 없다.

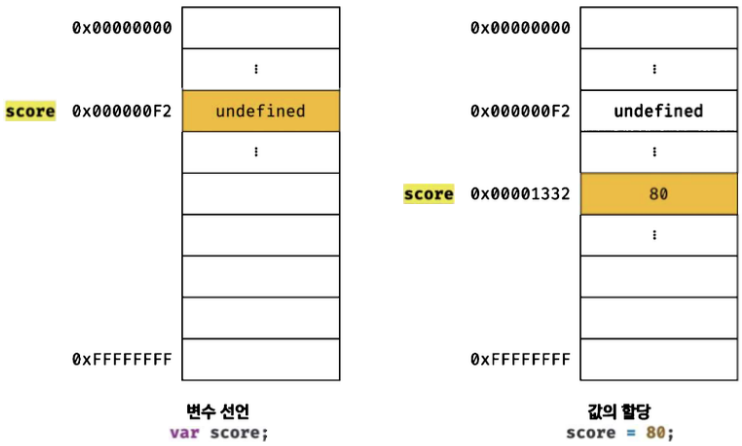


그림 4-8 값의 할당

위 그림처럼 변수에 값을 할당할 때는 이전 값 undefined가 저장되어 있던 메모리 공간을 지우고 그 메모리 공간에 할당 값 80을 새롭게 저장하는 게 아니라,

새로운 메모리 공간을 확보하고 그곳에 할당 값 80을 저장한다.

```
console.log(score);

score = 80;
var score;

console.log(score);
```

위 코드의 출력 결과는 아래와 같다.

`undefined`

`80`

값의 재할당

```
var score = 80;
score = 90;
```

var 키워드로 선언한 변수는 값을 재할당 할 수 있다.

var 키워드로 선언한 변수는 선언과 동시에 undefined로 초기화되기 때문에 엄밀히 말하자면 변수에 처음으로 값을 할당하는 것도 재할당에 해당한다.

값을 재할당 할 수 없어서 변수에 저장된 값을 변경할 수 없다면 변수가 아니라 상수라고 칭한다.

`const 키워드`

ES6에서 도입된 const 키워드를 사용해 선언한 변수는 재할당이 금지된다.

즉 const 키워드는 단 한 번만 할당할 수 있는 변수를 선언한다. 따라서 const 키워드를 사용하면 상수를 표현할 수 있다.

변수에 값을 재할당하면 score 변수의 값은 이전 값 80에서 재할당한 값 90으로 변경된다.

처음 값을 할당했을 때와 마찬가지로 **이전 값 80이 저장되어 있던 메모리 공간 안의 데이터를 지우고 그 메모리 공간에 재할당 값 90을 새롭게 저장하는 것이 아니라 새로운 메모리 공간을 확보하고 그 메모리 공간에 숫자 값 90을 저장한다.**

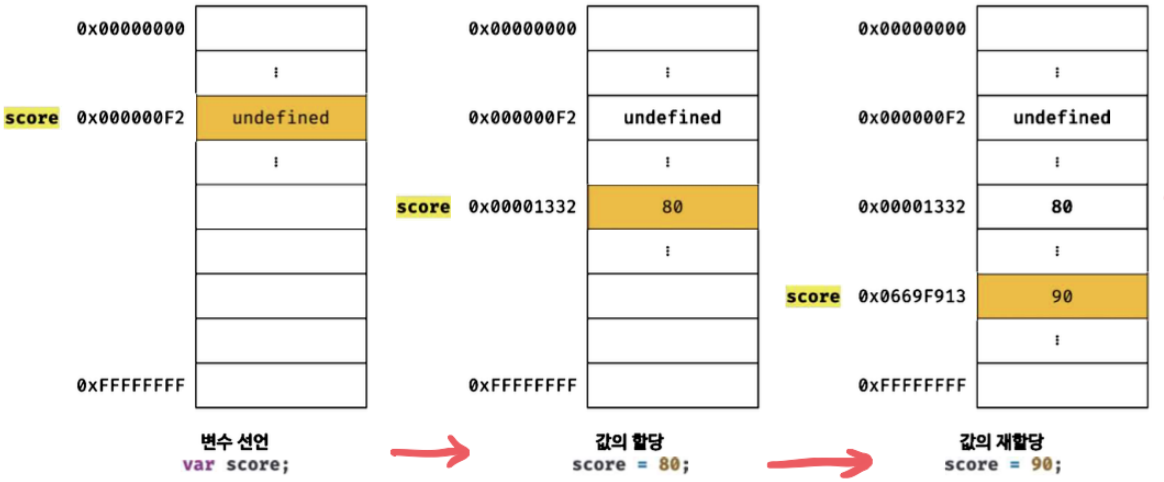


그림 4-10 값의 재할당

변수에 새로운 값이 저장되면, 이전 값 80, undefined 은 더 이상 필요하지 않은 값이므로, **가비지 콜렉터에의해 메모리에서 해제된다.**

단, 메모리에서 언제 해제될지는 예측할 수 없다.

가비지 콜렉터

GC는 애플리케이션이 할당한 메모리 공간을 주기적으로 검사하여 더 이상 사용되지 않는 메모리를 해제하는 기능을 말한다.

더 이상 사용되지 않는 메모리란, 어떤 식별자도 참조하지 않는 메모리 공간을 의미한다.

자바스크립트는 가비지 콜렉터를 내장하고 있는 매니지드 언어로, GC를 통해 메모리 누수를 방지한다.

언매니지드 언어, 매니지드 언어

프로그래밍 언어는 메모리 관리 방식에 따라 언매니지드 언어와 매니지드 언어로 분류할 수 있다.

C언어 같은 언매니지드 언어는 개발자가 명시적으로 메모리를 할당하고 해제하기 위해 malloc()과 free()같은 저수준 메모리 제어 기능을 제공한다.

언매니지드 언어는 개발자의 역량에 따라 최적의 성능을 확보할 수 있지만 그 반대의 경우 치명적 오류를 생산할 가능성도 있다.

매니지드 언어는 메모리의 할당 및 해제를 위한 메모리 관리 기능을 언어차원에서 담당하고 개발자의 직접적인 메모리 제어를 허용하지 않는다.

사용하지 않는 메모리의 해제는 GC가 담당하며, 이 또한 개발자가 관여할 수 없다.

식별자 네이밍 규칙

식별자는 어떤 값을 구별해서 식별해낼 수 있는 고유한 이름을 말한다.

식별자는 다음과 같은 네이밍 규칙을 준수해야 한다.

- 식별자는 특수문자를 제외한 문자,숫자,언더스코어(_), 달러 기호($)를 포함할 수 있다.
- 단, 식별자는 특수문자를 제외한 문자, 언더스코어(_), 달러 기호($)로 시작해야 한다. 숫자로 시작하는 것은 허용하지 않는다.
- 자바스크립트는 대소문자를 구분한다.
- 변수 선언에 별도의 주석이 필요하다면 변수 존재 목적을 명확히 드러내지 않는다는 것이다.

예약어

예약어는 프로그래밍 언어에서 사용되고 있거나 사용될 예정인 단어를 말한다.

js의 예약어는 다음과 같다.

await	break	case	catch	class	const
continue	debugger	default	delete	do	else
enum	export	extends	false	finally	for
function	if	implements*	import	in	instanceof
interface*	let*	new	null	package*	private*
protected*	public*	return	super	static*	switch
this	throw	true	try	typeof	var
void	while	with	yield*		

* 식별자로 사용 가능하나 strict mode에서는 사용 불가

표 4-1 예약어

변수 이름도 식별자이므로 위 네이밍 규칙을 따라야 한다.

예를 들어 아래와 같이 변수를 선언할 수 있다.

```
var person, $elem, _name, first_name, val1;
```

하지만 가독성이 나빠지므로 권장하지 않는다.

ES5부터 식별자를 만들 때 유니코드 문자를 허용하므로 알파벳 외의 한글이나, 일본어 식별자도 사용할 수 있다.

하지만 알파벳 외의 유니코드 문자로 명명된 식별자를 사용하는 것은 바람직하지 않으므로 권장하지 않는다.

네이밍 컨벤션은 하나 이상의 영어 단어로 구성된 식별자를 만들 때 가독성 좋게 단어를 한눈에 구분하기 위해 규정한 명명 규칙이다. 네이밍 컨벤션을 잘 지키면 읽기 좋은 이름을 만들 수 있다.

```
var fistName; // 카멜 케이스

var first_name; // 스네이크 케이스

var FistName; // 파스칼 케이스

// 헝가리안 케이스
var strFirstName; // type + identifier
var $elem = document.getElementById('myId'); // DOM 노드
var observable$ = fromEvent(document, 'click'); // RxJS 옵저버블
```

일관성을 유지한다면 어떤 네이밍 컨벤션을 사용해도 좋지만 자바스크립트에서는 일반적으로 변수나 함수의 이름에는 카멜 케이스를 사용하고, 생성자 함수, 클래스 이름에는 파스칼 케이스를 사용한다.

따라서 코드 전체의 가독성을 높이려면 카멜 케이스와 파스칼 케이스를 따르는 것이 유리하다.

- `변수, 함수 → 카멜 케이스`
- `생성자 함수, 클래스 → 파스칼 케이스`