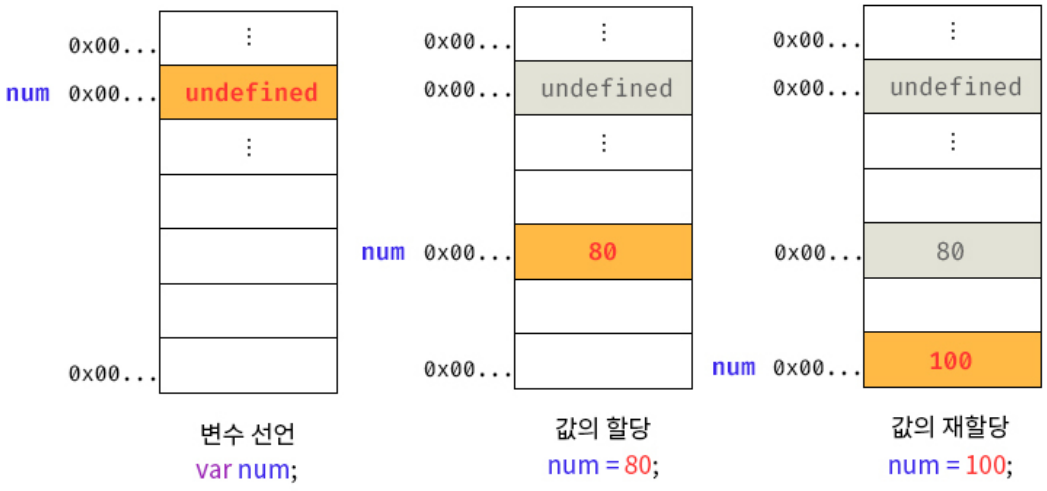
Chap4 변수

4-1 변수란 무엇인가? 왜 필요한가?

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

  ▼ 변수에 값이 저장되는 과정



1. 변수를 선언한다. (var, let, const 키워드 사용) → 키워드에 따라 값의 재할당. 재정의 가능 유무가 달라진다.

```
var num;
```

2. 값을 할당한다.

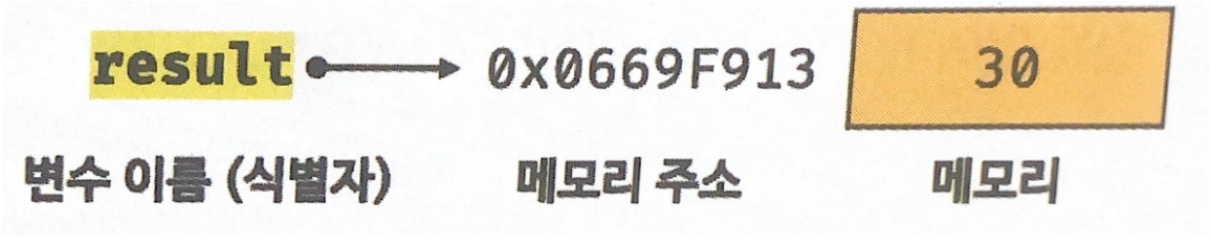
```
num=80;
```

값은 메모리에 저장되고 각각 메모리 주소를 갖게 된다. 그리고 변수는 이 메모리 주소를 참조하는 것이다.

메모리 주소는 메모리 공간의 위치를 나타내며 0부터 시작해서 메모리의 크기만큼 정수로 표현된다. 그리고 메모리에 저장되는 값은 값의 종류와 상관없이 2진수로 저장된다.

4-2 식별자

- 식별자 : 변수의 이름. 어떤 값을 구별해서 식별할 수 있는 고유의 이름



4-3,4 변수 선언

- 변수 선언 : 값을 저장하기 위한 메모리 공간을 확보하고 변수 이름과 확보된 메모리 공간의 주소를 연결해서 값을 저장할 수 있게 준비하는 것.
- var, let, const 키워드를 사용하여 변수를 선언한다. 선언하지 않고 식별자에 접근하면 ReferenceError(참조 에러)가 발생한다.

  ▼ var키워드

  (let과 const는 추후에 설명예정)

  - 함수 레벨 스코프이다. 이로 인해 전역변수로 선언되어 부작용을 일으킨다. 따라서 이를 보완하기 위해 ES6에 let과 const를 도입하였다.
  - 변수 선언과 동시에 초기화가 진행된다.

```
var score; //자동으로 undefined로 초기화 된다.
```

> 키워드란? 자바스크립트 코드를 해석하고 실행하는 자바스크립트 엔진이 수행할 동작을 규정한 일종의 명령어. 자바스크립트 엔지이 키워드를 만나면 자신이 수행해야 할 약속된 동작을 수행한다. 예를 들어 var키워드를 만나면 자바스크립트 엔진은 뒤에 오는 변수 이름으로 새로운 변수를 선언한다.

▼ ReferenceError

식별자를 통해 값을 참조하려 했지만 자바스크립트 엔진이 등록된 식별자를 찾을 수 없을 때 발생하는 에러

4-4 변수 선언의 실행 시점과 변수 호이스팅

```
console.log(score); //undefined
var score; //변수 선언문
```

변수 선언문보다 변수를 참조하는 코드가 앞에 있으면 위에서 설명한 것 과 같이 ReferenceError(참조 에러)가 발생할 것처럼 보인다. 그러나 참조 에러가 발생하지 않고 undefined가 출력된다.

Why? **호이스팅** 때문이다.

자바스크립 엔진은 소스 코드를 한 줄씩 순차적으로 실행하기에 앞서 변수 선언을 포함한 모든 선언문(변수 선언문, 함수 선언문 등)을 소스 코드에서 찾아내 먼저 실행한다. 따라서 변수 선언문이 코드의 맨 위로 끌어올려진 것처럼 느끼게 한다.

> 💡 **호이스팅**
> 변수 선언문이 코드의 선두로 끌어 올려진 것처럼 동작하는 자바스크립트 고유의 특징

변수 뿐만 아니라 var, elt const, function, function*, class 키워드를 사용한 모든 식별자는 호이스팅 된다.

▼ function* 란?

▼ **const/let 호이스팅**

var와 달리 let/const는 TDZ에 의해 제약을 받는다.

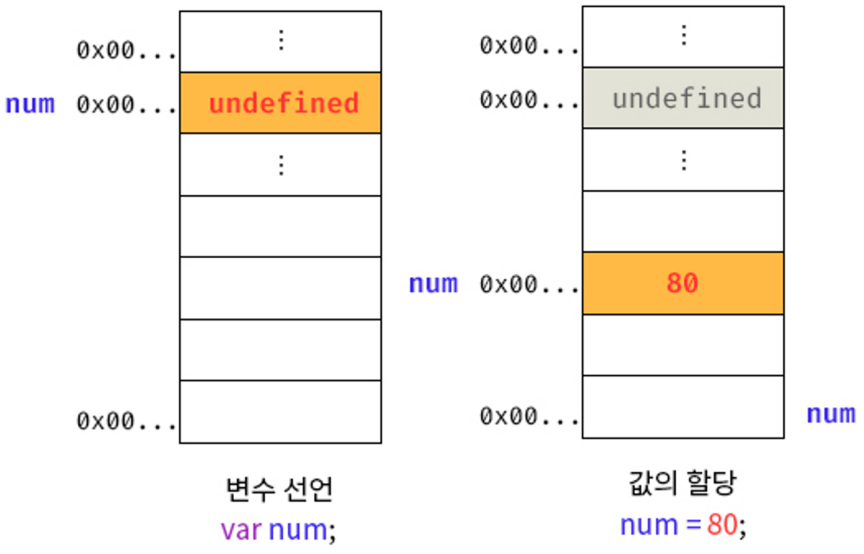
즉 변수가 초기화되기 전에 액세스하려고 하면 undefined를 반환하지 않고 ReferenceError가 발생한다.

그렇다면 호이스팅 하지않는걸끼? no

let/const 변수의 경우 var 변수와 다르게 변수의 선언 단계와 초기화 단계 사이에 일시적 사각 지대(TDZ)가 존재하고, TDZ에서 관리 중일 때(let 변수의 선언 또는 const 변수의 선언 및 할당 코드가 나오기 전)에 사용하려 한다면 RferenceError를 발생 시킴

4-5. 값의 할당

- 변수 선언과 값의 할당을 하나의 문으로 표현할 수 있다.
- 런타임 시점에 실행된다.
- 하나의 문장으로 단축 표현해도 자바스크립트 엔진은 변수의 선언과 값의 할당을 2개의 문으로 나누어 각각 실행한다.



4-6 값의 재할당

- 재할당 : 이미 값이 할당되어 있는 변수에 새로운 값을 또다시 할당하는 것을 말한다. (var, let 키워드 이용)

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

- 재할당으로 인해 메모리 공간에 식별자 없이 남은 값들은 가비지 콜렉터에 의해 메모리에서 자동 해제 된다. 단, 메모리에서 언제 해제 될지는 예측할 수 없다.

  > 💡 **가비지 콜렉터(garbage collector)**
  > 할당 한 메모리 공간을 주기적으로 검사하여 더 이상 사용되지 않는 메모리를 해제하는 기능. 자바스크립트는 가비지 콜렉터를 내장하고 있는 매니지드 언어로서 가비지 콜렉터를 통해 메모리 누수를 방지한다.

  ▼ 언매니지드 언어(unmanaged leanguage)

  C언어 같은 언매니지드 언어는 개발자가 명시적으로 메모리를 할당하고 해제하기 위한 malloc()과 free() 같은 저수준 메모리 제어 기능을 제공한다. 이렇듯 언매니지드 언어는 개발자가 직접 메모리 제어를 주도할 수 있으므로 개발자의 역량에 따라 최적의 성능을 확보할 수 있지만 능숙하지 않다면 오히려 치명적인 오류를 발생할 가능성이 있다.

  ▼ 매니지드 언어(unmanaged leanguage)

  자바스크립트 같은 매니지드 언어는 메모리의 할당 및 해제를 위한 메모리 관리 기능을 언어 차원에서 담당하고 개발자의 직접적인 메모리 제어를 허용하지 않는다. 따라서 개발자가 직접 메모리를 할당하고 해제할 수 없다. 재할당에 의해 더 이상 사용하지 않는 메모리의 해제는 가비지 콜렉터가 수행한다. 매니지드 언어는 개발자의 역량에 의존하는 부분이 상대적으로 작아져 어느 정도 일정한 생산성을 확보할 수 있다는 장점이 있지만 성능 면에서는 어느 정도 손실은 감수할 수밖에 없다.

- 재할당 x → 상수 (const 키워드 이용)

  상수는 한번 정해지면 변하지 않는 값이다.

▼ 참고

비교/대조 요약부터 👇

	var	const	let
변수 선언 및 초기값 할당	초기값 없이 변수 선언 가능	초기값 없이 변수 선언 불가	초기값 없이 변수 선언 가능
재선언 (re-declaration)	가능	불가	불가
재할당 (update)	가능	불가	가능
유효범위 (Scope)	**globally / locally blocked**	**globally / block scoped**	**globally / block scoped**
호이스팅 (hoisting)	scope 최상단으로 hoisting 되고, 값은 undefined로 초기화됨	scope 최상단으로 hoisting 되고, 값은 초기화되지 않음 (TDZ 발생)	scope 최상단으로 hoisting 되고, 값은 초기화되지 않음 (TDZ 발생)

4-7 식별자 네이밍 규칙

- 식별자는 특수문자를 제외한 문자, 숫자, 언더스코어(_), 달러 기호($)를 포함할 수 있다.
- 식별자는 특수문자를 제외한 문자, 언더스코어(_), 달러 기호($),로 시작해야한다. 숫자로 시작하는 것은 허용되지 않는다.
- 예약어는 식별자로 사용할 수 없다.

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*		