4. 변수

TABLE OF CONTENTS

#1. 변수란 무엇인가? 왜 필요한가?

애플리케이션은 데이터를 다룬다. 아무리 복잡한 애플리케이션이라 해도 데이터를 입력(input) 받아 처리하고 그 결과를 출력(output)하는 것이 전부다. 변수는 데이터를 관리하기 위한 핵심 개념이다. 변수란 무엇인지 그리고 왜 필요한지 살펴보도록 하자.

먼저 아래의 자바스크립트 코드를 실행하면 컴퓨터에서는 어떤 일이 일어날지 생각해 보자.

JAVASCRIPT

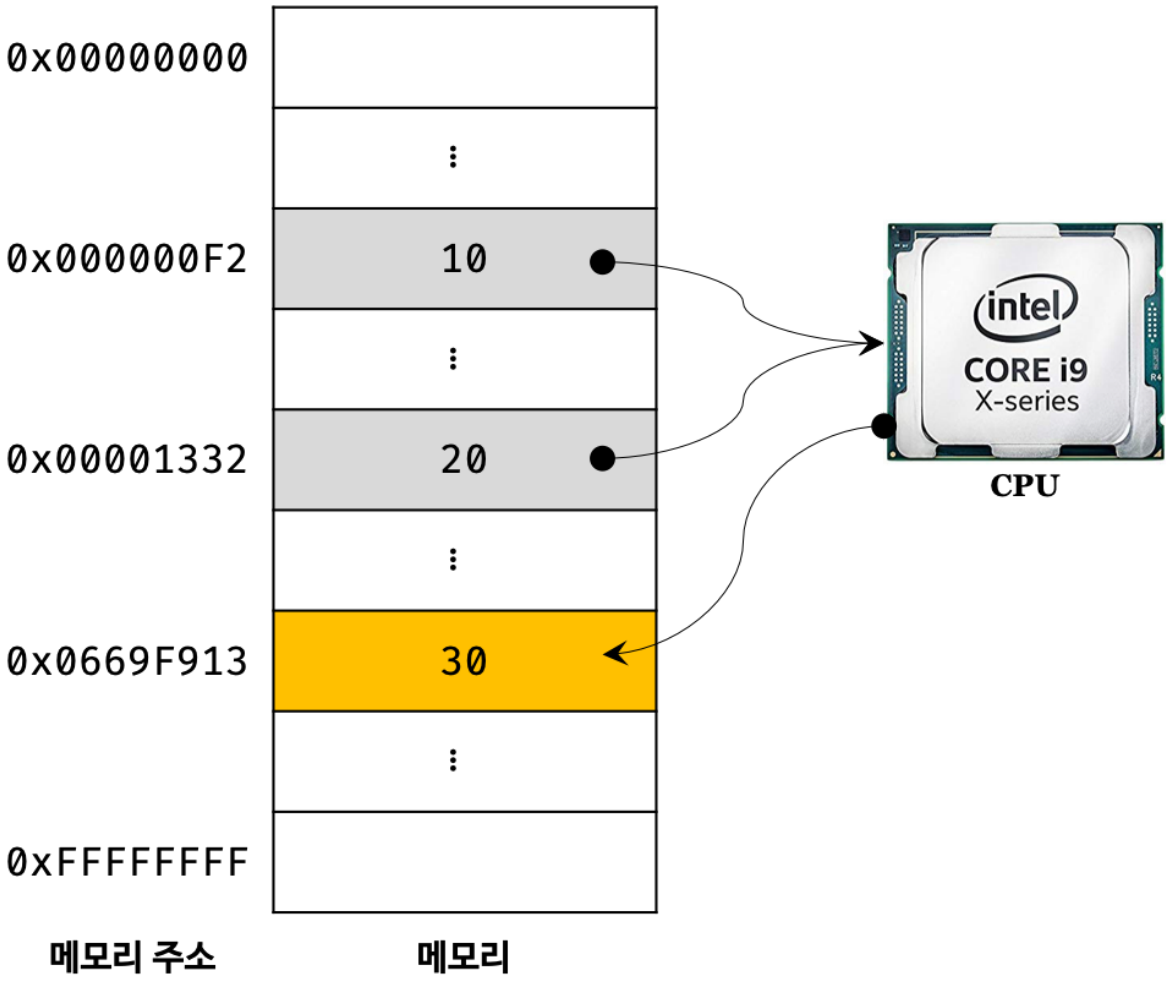
```
10 + 20
```

숫자 값 10과 20을 + 연산자(operator)로 합산하고 있다. CPU가 + 연산을 수행하기 위해서는 먼저 + 연산자의 좌변과 우변의 숫자 값, 즉 피연산자(operand)를 기억해야 한다. 컴퓨터는 메모리를 사용해 데이터를 기억한다.

메모리

메모리(memory)는 데이터를 저장할 수 있는 메모리 셀(memory cell) 들의 집합체이다. 셀 하나의 크기는 1byte(8bit)이며 컴퓨터는 셀의 크기, 즉 1byte 단위로 데이터를 저장(write)하거나 읽어(read) 들인다. 컴퓨터는 모든 데이터를 2진수로 처리한다. 따라서 메모리에 저장되는 데이터는 데이터의 종류(숫자, 텍스트, 이미지, 동영상 등)와 상관없이 2진수다. 각각의 셀은 고유의 메모리 주소(memory address)를 갖는다. 이 메모리 주소는 메모리 공간의 위치를 나타내며 0부터 시작하여 메모리의 크기만큼 정수로 표현된다. 예를 들어 4GB의 메모리는 0부터 4,294,967,295(0x00000000 ~ 0xFFFFFFFF)까지의 메모리 주소를 갖는다.

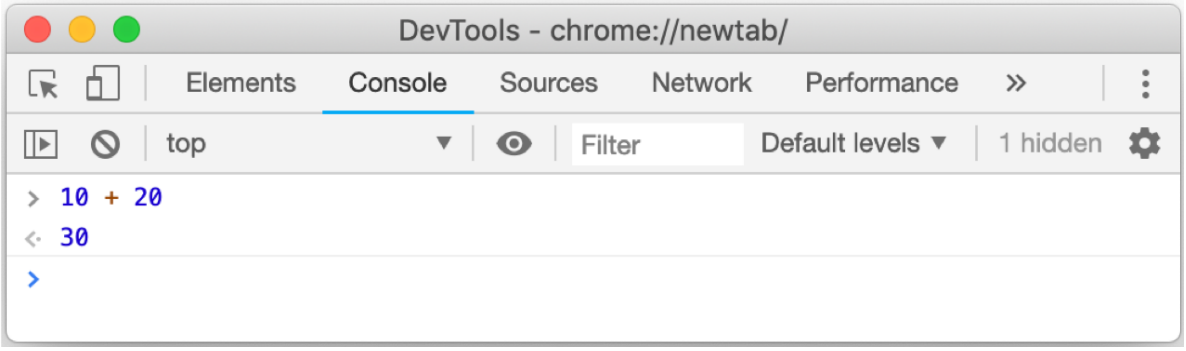
숫자 값 10과 20은 메모리 상의 임의의 위치(메모리 주소, memory address)에 기억(저장)되고 CPU는 이 값을 읽어들여 연산을 수행한다. 연산 결과로 생성된 숫자 값 30도 메모리 상의 임의의 위치에 저장된다.



메모리에 기억된 데이터와 CPU의 연산

위 그림에서는 메모리에 저장된 숫자 값을 편의상 10진수로 표기했다. 하지만 메모리에 저장되는 모든 값은 2진수로 저장된다는 것을 기억하기 바란다.

성공적으로 연산이 끝났고 연산 결과도 메모리에 저장됐지만 문제가 있다. CPU가 연산하여 만들어낸 숫자 값 30을 재사용할 수 없다는 것이다.



연산 결과를 재사용할 수 없다.

10 + 20이라는 연산을 했다는 것은 그 연산 결과가 필요하고 이를 사용해 무언가를 하겠다는 의도가 있었을 것이다. 연산 결과를 단 한번만 사용한다면 문제가 없겠지만 만약 연산 결과 30을 재사용하고 싶다면 메모리 주소(그림 4-1의 경우, 0x0669F913)를 통해 연산 결과 30이 저장된 메모리 공간에 직접 접근하는 것 이외에는 방법이 없다.

<메모리 공간에 직접 접근하는 것의 2가지 문제점>

하지만 메모리 주소를 통해 값에 직접 접근하는 것은 시스템을 멈추게 할 수도 있는 치명적 오류를 생산할 가능성이 높은 매우 위험한 일이다. 만약 실수로 OS가 사용하고 있는 값을 변경하면 시스템에 치명적인 오류가 발생할 수도 있다. 따라서 자바스크립트는 개발자의 직접적인 메모리 제어를 허용하지 않는다.

만약 자바스크립트가 개발자의 직접적인 메모리 제어를 허용하더라도 문제가 있다. 값이 저장될 메모리 공간의 주소는 코드가 실행될 때, 메모리의 상황에 따라 임의로 결정된다. 따라서 동일한 컴퓨터에서 동일한 코드를 실행해도 코드가 실행될 때마다 값이 저장될 메모리 주소는 변경된다. 이처럼 코드가 실행되기 이전에는 값이 저장된 메모리 주소를 알 수 없으며 알려 주지도 않는다. 따라서 메모리 주소를 통해 값에 직접 접근하려는 시도는 올바른 방법이 아니다.

프로그래밍 언어는 기억하고 싶은 값을 메모리에 저장하고 저장된 값을 읽어 들여 재사용하기 위해 변수라는 메커니즘을 제공한다. 변수의 정의를 내려보면 아래와 같다.

변수(Variable)는 하나의 값을 저장하기 위해 확보한 메모리 공간 자체 또는 그 메모리 공간을 식별하기 위해 붙인 이름을 말한다.

간단히 말하자면 변수는 프로그래밍 언어에서 값을 저장하고 참조하는 메커니즘으로 **값의 위치를 가리키는 상징적인 이름**이다. 상징적 이름인 변수는 프로그래밍 언어의 컴파일러 또는 인터프리터에 의해 값이 저장된 메모리 공간의 주소로 치환되어 실행된다. 따라서 개발자가 직접 메모리 주소를 통해 값에 저장하고 참조할 필요가 없고 변수를 통해 안전하게 값에 접근할 수 있다.

변수에 여러 개의 값을 저장하는 방법

변수는 하나의 값을 저장하기 위한 메커니즘이다. 여러 개의 값을 저장하려면 여러 개의 변수를 사용해야 한다. 단, 배열이나 객체와 같은 자료 구조를 사용하면 관련이 있는 여러 개의 값을 그룹화하여 하나의 값처럼 사용할 수 있다.

JAVASCRIPT

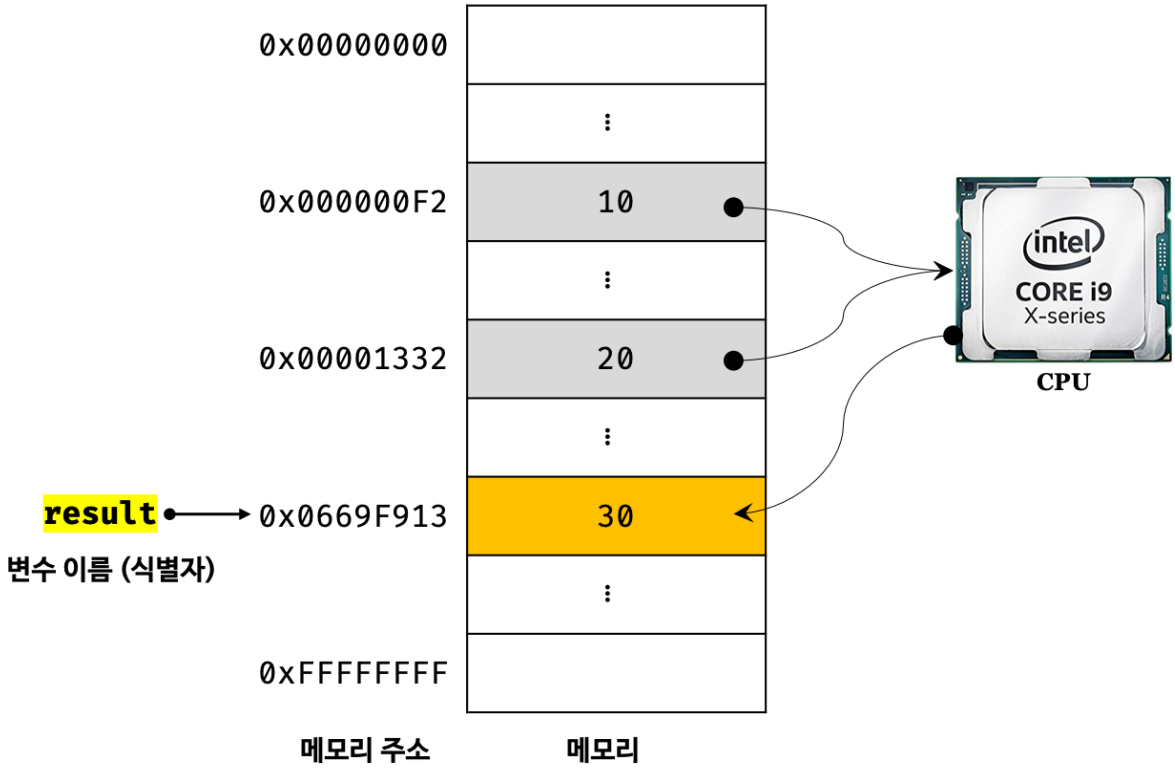
```
// 변수는 하나의 값을 저장하기 위한 수단이다.
var userId = 1;
var userName = 'Lee';

// 객체나 배열과 같은 자료 구조를 사용하면 여러 개의 값을 하나로 그룹화하여 하나의 값처럼 사용할 수 있다.
var user = { id: 1, name: 'Lee' };

var users = [
  { id: 1, name: 'Lee' },
  { id: 2, name: 'Kim' }
];
```

앞서 살펴본 코드를 변수를 사용해 다시 작성해보자.

JAVASCRIPT

```
var result = 10 + 20;
```

10 + 20은 연산을 통해 새로운 값 30을 생성한다. 그리고 연산을 통해 생성된 값 30은 메모리 공간에 저장된다. 이때 메모리 공간에 저장된 값 30을 다시 읽어 들여서 재사용할 수 있도록 값이 저장된 메모리 공간에 상징적인 이름을 붙인 것이 바로 변수이다.

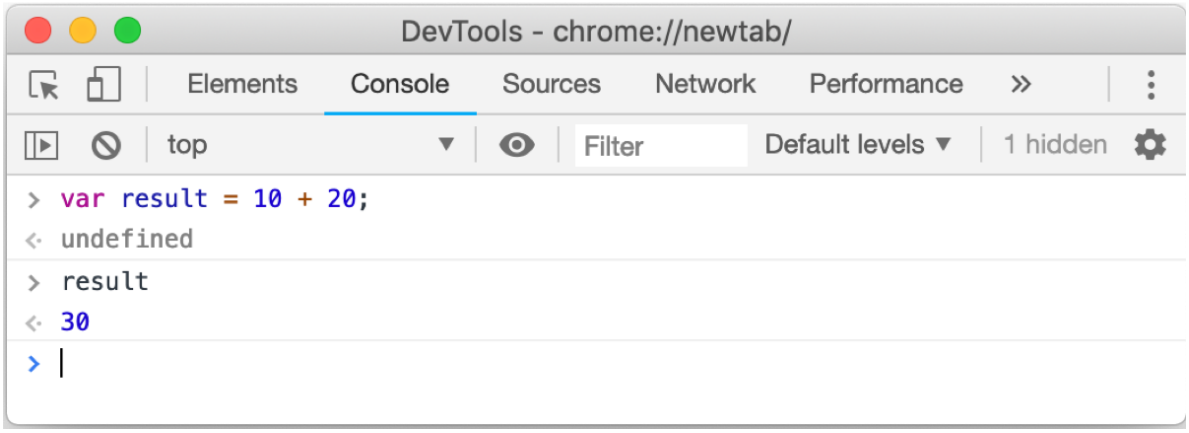


값의 생성과 변수에의 할당

메모리 공간에 저장된 값을 식별할 수 있는 고유한 이름(위 예제의 경우 result)을 변수 이름이라 한다. 그리고 변수에 저장된 값(위 예제의 경우 30)을 변수 값이라고 한다.

변수에 값을 저장하는 것을 **할당**(assignment **대입, 저장**)이라 하고, 변수에 저장된 값을 읽어 들이는 것을 **참조**(reference)라고 한다.

변수 이름은 사람을 위해 사람이 이해할 수 있는 언어로 값이 저장된 메모리 공간에 붙인 상징적인 이름이다. 변수 이름을 통해 참조를 요청하면 자바스크립트 엔진은 변수 이름과 매핑된 메모리 주소를 통해 메모리 공간에 접근하여 저장된 값을 반환해 준다.



변수 이름에 의한 값의 참조

사람이 이해할 수 있는 언어로 명명한 변수 이름을 통해 변수에 저장된 값의 의미를 명확히 할 수 있다. 따라서 좋은 이름, 즉 변수에 저장된 값의 의미를 파악할 수 있는 변수 이름은 가독성을 높여주는 부수적인 효과도 있다.

코드는 컴퓨터에게 내리는 명령이지만 개발자를 위한 문서이기도 하다. 개발자의 의도를 나타내는 명확한 네이밍은 코드를 이해하기 쉽게 만들며 이는 협업과 품질 향상에 도움을 준다. 변수 이름은 첫아이 이름을 짓듯이 심사숙고해서 지어야 한다.

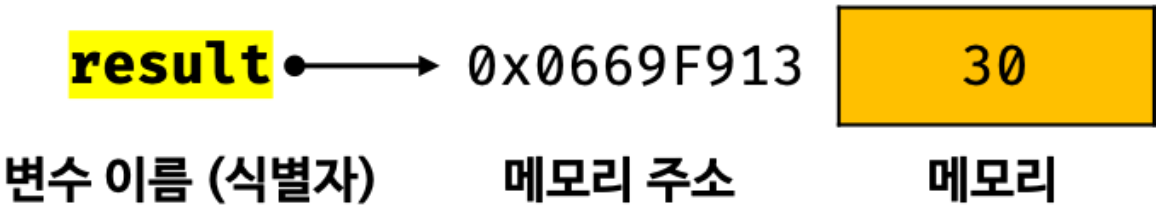
2. 식별자

변수 이름을 식별자(identifier)라고도 부른다. **식별자는 어떤 값을 구별하여 식별해낼 수 있는 고유한 이름**을 말한다. 사람을 이름으로 구별하여 식별하는 것처럼 값도 식별자로 구별하여 식별할 수 있다.

값은 메모리 공간에 저장되어 있다. 따라서 식별자는 메모리 공간에 저장되어 있는 어떤 값을 구별하여 식별해낼 수 있어야 한다. 이를 위해 식별자는 어떤 값이 저장되어 있는 메모리 주소를 기억(저장)해야 한다.

그림 4-3에서 식별자 result는 값 30을 식별할 수 있었다. 이를 위해 식별자 result는 값 30이 저장되어 있는 메모리 주소 0x0669F913을 기억해야 한다. 즉, 식별자는 값이 저장되어 있는 메모리 주소와 매핑 관계를 가지며 이 매핑 정보도 메모리에 저장되어야 한다.

이처럼 **식별자는 값이 아니라 메모리 주소를 기억하고 있다**. 식별자로 값을 구별하여 식별한다는 것은 식별자가 기억하고 있는 메모리 주소를 통해 메모리 공간에 저장된 값에 접근할 수 있다는 것을 의미한다. 즉, 식별자는 메모리 주소에 붙인 이름이라고 할 수 있다.



식별자

식별자라는 용어는 변수 이름에만 국한해서 사용하지 않는다. 예를 들어, 변수, 함수, 클래스 등의 이름은 모두 식별자다. 식별자인 변수 이름으로는 메모리 상에 존재하는 변수 값을 식별할 수 있고, 함수 이름으로는 메모리 상에 존재하는 함수(자바스크립트에서는 함수는 값이다)를 식별할 수 있다. 즉, 메모리 상에 존재하는 어떤 값을 식별할 수 있는 이름은 모두 식별자라고 부른다.

변수, 함수, 클래스 등의 이름과 같은 식별자는 네이밍 규칙("8. 식별자 네이밍 규칙" 참고)을 준수해야 하며, **선언**(declaration)에 의해 자바스크립트 엔진에 식별자의 존재를 알린다. 변수를 선언하는 방법에 대해 살펴보자.

3. 변수 선언

변수 선언(Variable declaration)이란 변수를 생성하는 것을 말한다. 좀 더 자세히 말하면 값을 저장하기 위한 메모리 공간을 확보(allocate, 할당)하고 변수 이름과 확보된 메모리 공간의 주소를 연결(name binding)하여 값을 저장할 수 있도록 준비하는 것이다. 변수 선언에 의해 확보된 메모리 공간은 확보가 해제(release)되기 이전까지는 누구도 확보된 메모리 공간을 사용할 수 없도록 보호되므로 안전하게 사용할 수 있다.

변수를 사용하려면 반드시 선언이 필요하다. 변수를 선언할 때는 var, let, const **키워드를 사용한다.**

ES6에서 let, const 키워드가 도입되기 이전까지 var 키워드는 자바스크립트에서 변수를 선언할 수 있는 유일한 키워드였다. 먼저 var 키워드를 사용해 변수를 선언하는 방법에 대해 살펴보고 let, const 키워드에 대해서는 나중에 자세히 알아보도록 하자.

> ES5 vs. ES6
> 아직 살펴보지 않았지만 var 키워드는 여러 단점("15.1. var 키워드로 선언한 변수의 문제점" 참고)이 있다. var의 여러 단점 중에서 가장 대표적인 것이 블록 레벨 스코프(Block-level scope)를 지원하지 않고 함수 레벨 스코프(Function-level scope)를 지원한다는 것이다. 이로 인해 의도치 않게 전역 변수가 선언되어 심각한 부작용이 발생되기도 한다.
> ES6에서 let과 const 키워드를 도입한 이유는 var 키워드의 여러 단점을 보완하기 위함이다. 따라서 let과 const 키워드가 도입된 이유를 정확히 파악하려면 먼저 var 키워드의 단점부터 이해해야 한다. var 키워드의 단점을 이해하려면 먼저 스코프와 같은 자바스크립트의 핵심 개념을 먼저 살펴보아야 한다.
> ES6에서 var 키워드가 폐지(deprecated)된 것은 아니다. ES6 이전 사양으로 작성된 코드는 var 키워드를 사용해 구현되어 있을 것이며, ES6 사양을 따른다 하더라도 권장하지는 않지만 var 키워드를 사용할 수도 있다.
> ES5와 ES6는 서로 상관없는 별개의 사양(Specification)이 아니다. ES6 이전 사양으로 구현된 코드는 ES6 기반의 자바스크립트 엔진에서 모두 정상 동작한다. 즉, ES6**는 기본적으로 하위 호환성을 유지하면서** ES5**의 기반 위에 새로운 기능을 추가한 것이다. 다시 말해,** ES6**는** ES5**의 상위 집합**(superset)**이다.** 따라서 ES6 사양을 기준으로 자바스크립트를 학습한다 하더라도 ES5 사양을 잘 알아둘 필요가 있다. ES5를 잘 이해하고 있으면 ES6를 보다 빠르고 명확하게 이해할 수 있기 때문이다.

아래의 코드를 살펴보자.

JAVASCRIPT

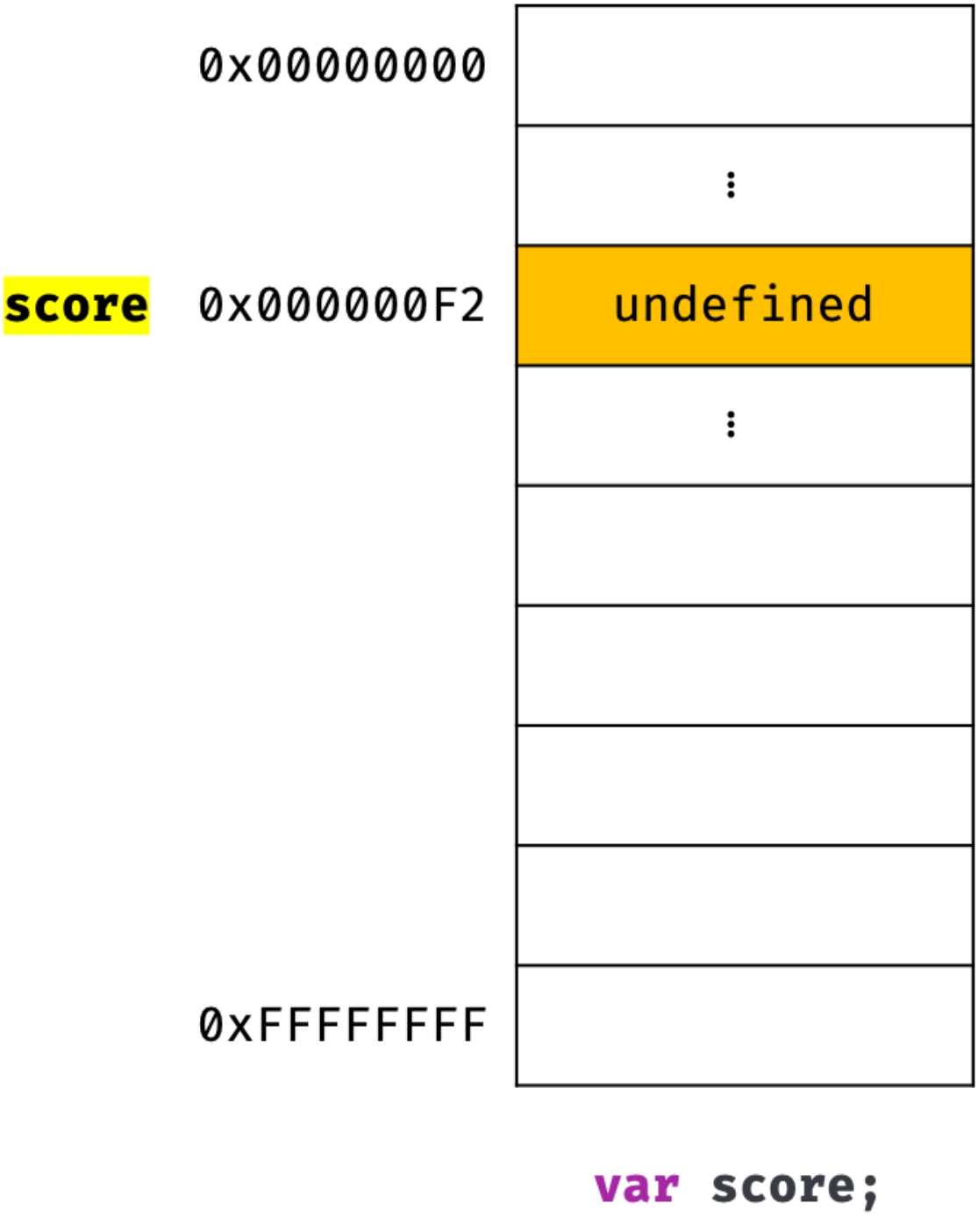
```
var score; // 변수 선언(변수 선언문)
```

var 키워드는 뒤에 오는 변수 이름으로 새로운 변수를 선언할 것을 지시하는 키워드이다.

키워드

키워드(keyword)는 자바스크립트 코드를 해석하고 실행하는 자바스크립트 엔진이 수행할 동작을 규정한 일종의 명령어이다. 자바스크립트 엔진은 키워드를 만나면 자신이 수행해야 할 약속된 동작을 수행한다. 예를 들어 var 키워드가 실행되면 자바스크립트 엔진은 뒤에 오는 변수 이름으로 새로운 변수를 선언한다.

위 변수 선언문은 아래와 같이 변수 이름을 등록(변수 이름은 어디에 등록되는가? 참고)하고 값을 저장할 메모리 공간을 확보한다.



변수 선언

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 초기화된다. 이것은 자바스크립트의 독특한 특징이다. (참고: Variable Statement)

> undefined
> undefined는 자바스크립트가 제공하는 원시 타입의 값(primitive value)이다. 이에 대해서는 "6. 데이터 타입"에서 살펴볼 것이다.

자바스크립트 엔진은 변수 선언을 아래의 2단계를 거쳐 수행한다.

- 선언 단계(Declaration phase) : 변수 이름을 등록하여 자바스크립트 엔진에 변수의 존재를 알린다.
- 초기화 단계(Initialization phase) : 값을 저장하기 위한 메모리 공간을 확보하고 암묵적으로 `undefined` 를 할당한다.

> **변수 이름은 어디에 등록되는가?**
> 변수 이름을 비롯한 모든 식별자는 실행 컨텍스트에 등록된다. 실행 컨텍스트(execution context)는 자바스크립트 엔진이 소스 코드를 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 key/value 형식인 객체로 등록되어 관리된다.
> 자바스크립트 엔진이 변수를 관리하는 메커니즘은 "13. 스코프"와 "23. 실행 컨텍스트"에서 자세히 살펴볼 것이다. 지금은 단순히 자바스크립트 엔진이 변수를 관리할 수 있도록 변수의 존재를 알린다는 정도로만 알아 두자.

var 키워드를 사용한 변수 선언은 선언 단계와 초기화 단계가 동시에 진행된다. `var score;` 는 선언 단계를 통해 변수 이름 score를 등록하고, 초기화 단계를 통해 변수 score에 암묵적으로 `undefined` 를 할당하여 초기화한다.

일반적으로 초기화(initialization)란 변수가 선언된 이후 최초로 값을 할당하는 것을 말한다. var 키워드로 선언한 변수는 `undefined` 로 암묵적인 초기화가 자동 수행된다. 따라서 var 키워드로 변수는 선언 이후 어떠한 값도 할당하지 않아도 `undefined` 라는 값을 갖는다.

만약 초기화 단계를 거치지 않으면 확보된 메모리 공간에는 이전에 다른 애플리케이션이 사용했던 값이 남아 있을 수 있다. 이러한 값을 쓰레기 값(garbage value)이라 한다. 따라서 메모리 공간을 확보한 다음 값을 할당하지 않은 상태에서 곧바로 변수 값을 참조하면 쓰레기 값이 나올 수 있다.

변수를 사용하려면 반드시 선언이 필요하다. 변수 뿐만이 아니라 모든 식별자(함수, 클래스 등)가 그렇다. 만약 선언하지 않은 식별자에 접근하면 ReferenceError(참조 에러)가 발생한다. ReferenceError는

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

ReferenceError(참조 에러)

4. 변수 선언의 실행 시점과 변수 호이스팅

아래 예제를 살펴보자.

JAVASCRIPT

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

var score; // 변수 선언문
```

변수 선언문보다 변수를 참조하는 코드가 앞에 있다. 자바스크립트 코드는 인터프리터("2.5. 자바스크립트의 특징" 참고)에 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 변수 score의 선언이 실행되지 않았으므로 참조 에러(ReferenceError)가 발생할 것처럼 보인다. 하지만 참조 에러가 발생하지 않고 `undefined` 가 출력된다.

그 이유는 변수 선언이 소스 코드가 한 줄씩 순차적으로 실행되는 시점, 즉 런타임(run-time)이 아니라 그 이전 단계에서 먼저 실행되기 때문이다.

자바스크립트 엔진은 코드를 한 줄씩 순차적으로 실행하기에 앞서 먼저 코드의 평가 과정을 거치면서 코드 실행을 위한 준비를 한다. 이때, 즉 코드 실행을 위한 준비 단계인 코드의 평가 과정에서 자바스크립트 엔진은 변수 선언을 포함한 모든 선언문(변수 선언문, 함수 선언문 등)을 코드에서 찾아내어 먼저 실행한다. 그리고 코드의 평가 과정이 끝나면 비로소 변수 선언을 포함한 모든 선언문을 제외하고 코드가 한 줄씩 순차적으로 실행한다.

> 변수가 선언되는 위 과정에 대해서는 "23. 실행 컨텍스트"에서 자세히 살펴볼 것이다. 지금은 변수 선언이 코드가 순차적으로 실행되는 런타임 이전에 먼저 실행된다는 것에 주목하자

즉, 자바스크립트 엔진은 변수 선언이 소스 코드의 어디에 있던지 상관없이 다른 코드보다 먼저 실행한다. 따라서 변수 선언이 소스 코드에 어디에 있던지 상관없이 변수를 참조할 수 있다.

위 예제를 다시 살펴보자. 변수 선언문 `var score;` 보다 변수를 참조하는 코드 `console.log(score);` 가 앞에 있다. 만약 코드가 순차적으로 실행되는 런타임에 변수 선언이 실행된다면 `console.log(score);` 가 실행되는 시점에는 아직 변수가 선언되기 이전이므로 위 코드의 실행하면 참조 에러(ReferenceError)가 발생해야 한다. 하지만 `undefined` 가 출력된다.

이는 변수 선언(선언 단계와 초기화 단계)이 소스 코드가 순차적으로 실행되는 런타임 이전 단계에서 먼저 실행된다는 증거이다. 이처럼 **변수 선언문이 코드의 선두로 끌어 올려진 것처럼 동작하는 자바스크립트 고유의 특징을 변수 호이스팅**(Variable Hoisting)이라 한다.

사실 호이스팅은 변수 선언 뿐만이 아니라 var, let, const, function, function*, class 키워드를 사용하여 선언된 모든 식별자(변수, 함수, 클래스 등)는 호이스팅된다. 모든 선언문은 런타임 이전 단계에서 먼저 실행되기 때문이다.

5. 값의 할당

변수에 값을 할당(assignment 대입, 저장)할 때는 할당 연산자(=)를 사용한다. 할당 연산자는 우변의 값을 좌변의 변수에 할당한다.

JAVASCRIPT

```
var score;  // 변수 선언
score = 80; // 값의 할당
```

변수 선언과 값의 할당을 아래와 같이 하나의 문(statement)으로 단축 표현할 수도 있다.

JAVASCRIPT

```
var score = 80; // 변수 선언과 값의 할당
```

자바스크립트 엔진은 위 예제를 변수 선언과 값의 할당으로 구분하여 실행한다. 따라서 변수 선언과 값의 할당을 구분한 코드와 위 코드는 정확히 동일하게 동작한다. 이때 주의할 것은 **변수 선언은 소스 코드가 순차적으로 실행되기 이전, 즉 런타임 이전에 먼저 실행되지만 값의 할당은 소스 코드가 순차적으로 실행되는 시점인 런타임에 실행된다.**

JAVASCRIPT

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

var score = 80;     // 변수 선언과 값의 할당

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

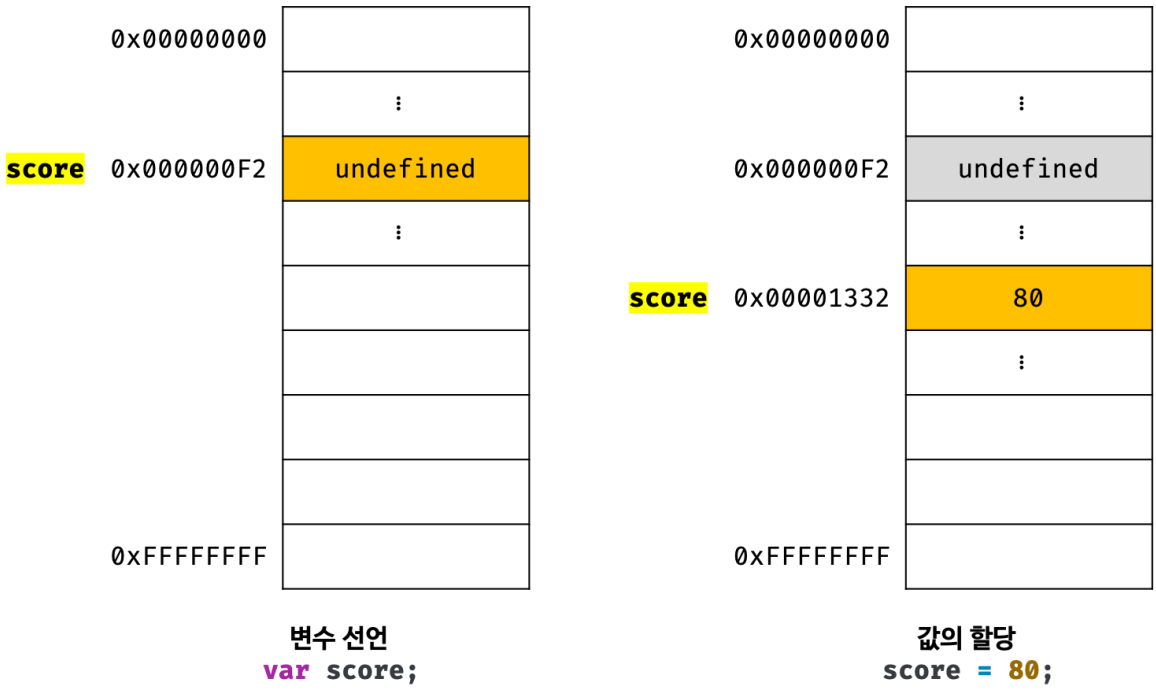
변수 score에 값을 할당하는 시점에는 이미 변수 선언이 완료된 상태이고 변수는 `undefined`로 초기화되어 있다. 따라서 변수 score에 값을 할당하면 변수 score의 값은 `undefined`에서 새롭게 할당한 숫자 값 80으로 변경(재할당)된다.

이것은 변수 선언과 값의 할당을 각각 실행했을 때와 변수의 선언과 값의 할당을 하나의 문으로 단축 표현했을 때 모두 동일하다.

JAVASCRIPT

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

var score;  // 변수 선언
score = 80; // 값의 할당

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

즉, 변수의 선언과 값의 할당을 하나의 문장으로 단축 표현해도 자바스크립트 엔진은 변수의 선언과 값의 할당을 분리해서 각각 실행한다는 의미이다. 따라서 변수에 `undefined` 가 할당되어 초기화되는 것은 변함이 없다.



값의 할당

위 그림처럼 변수에 값을 할당할 때는 이전 값 `undefined` 가 저장되어 있던 메모리 공간을 지우고 그 메모리 공간에 할당 값 80을 새롭게 저장하는 것이 아니라 새로운 메모리 공간을 확보하고 그 메모리 공간에 할당 값 80을 저장하는 것에 주의하자.

그렇다면 아래 예제의 실행 결과는 무엇일지 생각해 보자. 그리고 그 이유에 대해 자신에게 설명해보고 실제로 그렇게 동작하는지 예제를 실행하여 "컴퓨터에게 물어보라."("Ask the computer." Spring의 아버지, 로드 존슨(Rod Johnson))

JAVASCRIPT

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

score = 80; // 값의 할당
var score;  // 변수 선언

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

6. 값의 재할당

이번에는 아래와 같이 변수 score에 새로운 값을 재할당해보자. 재할당이란 이미 값이 할당되어 있는 변수에 새로운 값을 또다시 할당하는 것을 말한다.

JAVASCRIPT

```
var score = 80; // 변수 선언과 값의 할당
score = 90;     // 값의 재할당
```

var 키워드로 선언한 변수는 값을 재할당할 수 있다. 재할당은 현재 변수가 저장하고 있는 값을 버리고 새로운 값을 저장하는 것이다. var 키워드로 선언한 변수는 선언과 동시에 `undefined` 로 초기화되기 때문에 엄밀히 말하자면 변수에 처음으로 값을 할당하는 것도 사실은 재할당이다.

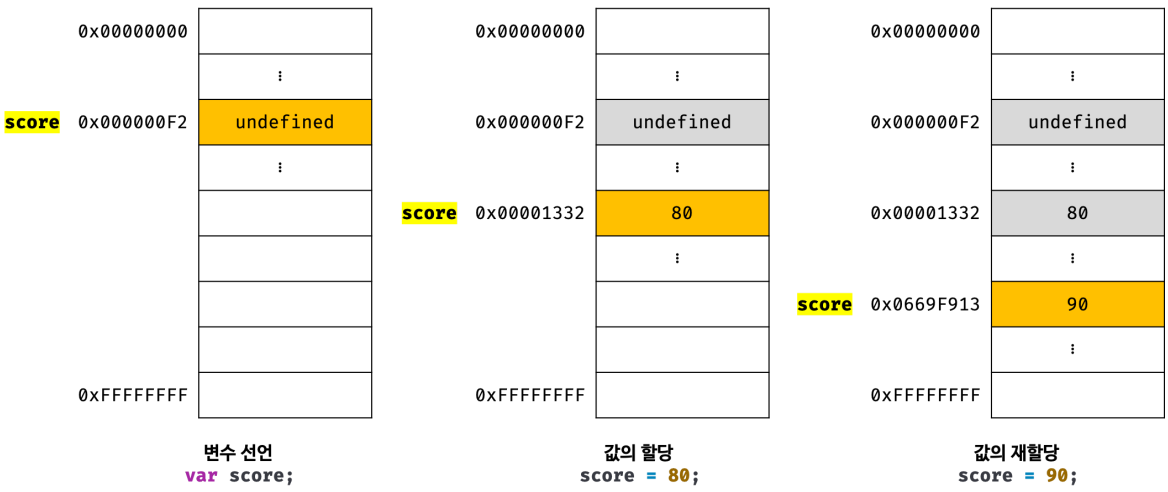
재할당은 변수에 저장된 값을 다른 값으로 변경한다. 따라서 변수라 부른다. 만약 **재할당을 할 수 없어서 변수에 저장된 값을 변경할 수 없다면 변수가 아니라 상수(Constant)라 부른다**. 상수는 한번 정해지면 변하지 않는 값이다. 다시 말해 상수는 단 한번만 할당할 수 있는 변수이다.

> const 키워드
> ES6에서 도입된 const 키워드로 선언한 변수는 재할당이 금지된다. 따라서 const 키워드를 사용하면 상수를 표현할 수 있다. 하지만 const 키워드는 반드시 상수 만을 위해 사용하지는 않는다. 이에 대해서는 "15.3. const 키워드"에서 살펴볼 것이다.

```
> const foo = 10;
< undefined
> foo = 100;
✖ ▶ Uncaught TypeError: Assignment to constant variable.
      at <anonymous>:1:5
```

const 키워드와 상수

변수에 값을 재할당하면 변수 score는 이전 값 80에서 재할당 값 90으로 변경된다. 처음 값을 할당했을 때와 마찬가지로 이전 값 80이 저장되어 있던 메모리 공간을 지우고 그 메모리 공간에 재할당 값 90을 새롭게 저장하는 것이 아니라 새로운 메모리 공간을 확보하고 그 메모리 공간에 숫자 값 90을 저장한다.



값의 재할당

현재 변수 score의 값은 90이다. 변수 score의 이전 값인 undefined와 80은 어떤 변수도 값으로 갖고 있지 않다. 이것은 undefined과 80이 더이상 필요하지 않다는 것을 의미한다. 아무도 사용하고 있지 않으니 필요하지 않은 것이다. 이러한 불필요한 값들은 가비지 컬렉터(garbage collector)에 의해 메모리에서 자동 해제된다. 단, 메모리에서 언제 해제될 지는 예측할 수 없다.

가비지 컬렉터(garbage collector)

가비지 컬렉터는 애플리케이션이 할당(allocate)한 메모리 공간을 주기적으로 검사하여 더 이상 사용되고 있지 않는 메모리를 해제(release)하는 기능을 말한다. 더 이상 사용되고 있지 않는 메모리란 간단히 말하자면 어떤 식별자도 참조하고 있지 않는 메모리 공간을 의미한다. 자바스크립트는 가비지 컬렉터를 내장하고 있는 매니지드 언어이다. 이를 통해 메모리 누수(memory leak)를 방지한다.

언매니지드 언어와 매니지드 언어

프로그래밍 언어는 메모리 관리 방식에 의해 언매니지드 언어(unmanaged language)와 매니지드 언어(managed language)로 분류할 수 있다.

C 언어와 같은 언매니지드 언어는 개발자가 명시적으로 메모리를 할당하고 해제하기 위해 malloc()과 free()와 같은 저수준(low-level) 메모리 제어 기능을 제공한다. 언매니지드 언어는 메모리 제어를 개발자가 주도할 수 있으므로 개발자의 역량에 의해 최적의 퍼포먼스를 확보할 수 있지만 그 반대의 경우, 치명적 오류를 생산할 가능성도 동시에 존재한다.

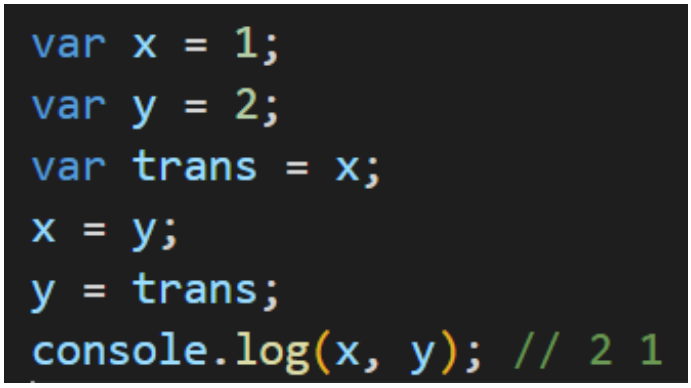
자바스크립트와 같은 매니지드 언어는 메모리의 할당 및 해제를 위한 메모리 관리 기능을 언어 차원에서 담당하고 개발자의 직접적인 메모리 제어를 허용하지 않는다. 즉, 개발자가 명시적으로 메모리를 할당하고 해제할 수 없다. 더 이상 사용하지 않는 메모리의 해제는 가비지 컬렉터가 수행하며 이 또한 개발자가 관여할 수 없다. 매니지드 언어는 개발자의 역량에 의존하는 부분이 상대적으로 작아져 어느 정도 일정한 생산성을 확보할 수 있는 장점이 있지만 퍼포먼스 면에서의 손실은 감수할 수 밖에 없다.

7. 값의 교환

두 변수의 값을 교환하는 코드를 작성하라.

JAVASCRIPT

```
var x = 1;
var y = 2;

// do something

console.log(x, y); // 2 1
```



8. 식별자 네이밍 규칙

앞에서 언급했듯이 식별자(identifier)는 어떤 값을 구별하여 식별해낼 수 있는 이름을 말한다. 식별자는 아래와 같은 네이밍 규칙을 준수해야 한다.

- 식별자는 특수문자를 제외한 문자, 숫자, underscore(_), 달러 기호($)를 포함할 수 있다.
- 단, 식별자는 특수문자를 제외한 문자, underscore(_), 달러 기호($)로 시작해야 한다. 숫자로 시작하는 것은 허용하지 않는다.
- 예약어는 식별자로 사용할 수 없다.

예약어

예약어(reserved word)는 프로그래밍 언어에서 사용되고 있거나 사용될 예정인 단어들을 말한다. 자바스크립트의 예약어는 아래와 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

변수 이름도 식별자이므로 위 네이밍 규칙을 따라야 한다. 예를 들어 아래와 같은 식별자는 변수 이름으로 사용할 수 있다. 참고로 변수는 쉼표(,)로 구분해 하나의 문에서 여러 개를 한번에 선언할 수 있다. 하지만 가독성이 나빠지므로 권장하지는 않는다.

JAVASCRIPT

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

ES5부터 식별자를 만들 때 유니코드 문자를 허용하므로 알파벳 이외의 한글이나 일본어 식별자도 사용할 수 있다. 하지만 알파벳 이외의 유니코드 문자로 명명된 식별자를 사용하는 것은 바람직하지 않으므로 권장하지 않는다.

JAVASCRIPT

```
var 이름, なまえ;
```

아래의 식별자는 명명 규칙에 위배되므로 변수 이름으로 사용할 수 없다.

JAVASCRIPT

```
var first-name; // SyntaxError: Unexpected token -
var 1st;        // SyntaxError: Invalid or unexpected token
var this;       // SyntaxError: Unexpected token this
```

자바스크립트는 대소문자를 구별하므로 아래의 변수는 각각 별개의 변수이다.

JAVASCRIPT

```
var firstname;
var firstName;
var FIRSTNAME;
```

변수 이름은 변수의 존재 목적을 쉽게 이해할 수 있도록 의미를 명확히 표현해야 한다. 좋은 변수 이름은 코드의 가독성이 높인다.

JAVASCRIPT

```
var x = 3;        // NG. 변수 x가 의미하는 바를 알 수 없다.
var score = 100; // OK. 변수 score는 점수를 의미한다.
```

변수 선언에 별도의 주석이 필요하다면 변수의 존재 목적을 명확히 드러내지 못하는 것이다.

JAVASCRIPT

```
// 경과 시간. 단위는 날짜이다
var d;                  // NG

var elapsedTimeInDays; // OK
```

하나 이상의 영어 단어로 구성된 식별자를 네이밍할 때 단어를 한눈에 구분하기 위해 자주 사용되는 네이밍 컨벤션은 아래와 같이 4가지가 있다.

JAVASCRIPT

```
// 카멜 케이스 (camelCase)
var firstName;

// 스네이크 케이스 (snake_case)
var first_name;

// 파스칼 케이스 (PascalCase)
var FirstName;

// 헝가리언 케이스 (typeHungarianCase)
var strFirstName;          // type + identifier
var $elem = $('.myClass'); // jQuery
```

일관성을 유지한다면 어떤 네이밍 컨벤션을 사용하여도 좋다. 가장 일반적인 것은 **변수나 함수의 이름에는 카멜 케이스**를 사용하고 **생성자 함수, 클래스의 이름에는 파스칼 케이스**를 사용하는 것이다.

ECMAScript 사양에 정의된 표준 빌트인 객체(String, Number, Array, Function 등)와 전역 함수들도 카멜 케이스와 파스칼 케이스 네이밍 컨벤션을 사용하고 있다. 따라서 코드 전체의 가독성을 높이려면 카멜 케이스와 파스칼 케이스 네이밍 컨벤션을 따르는 것이 유리하다.