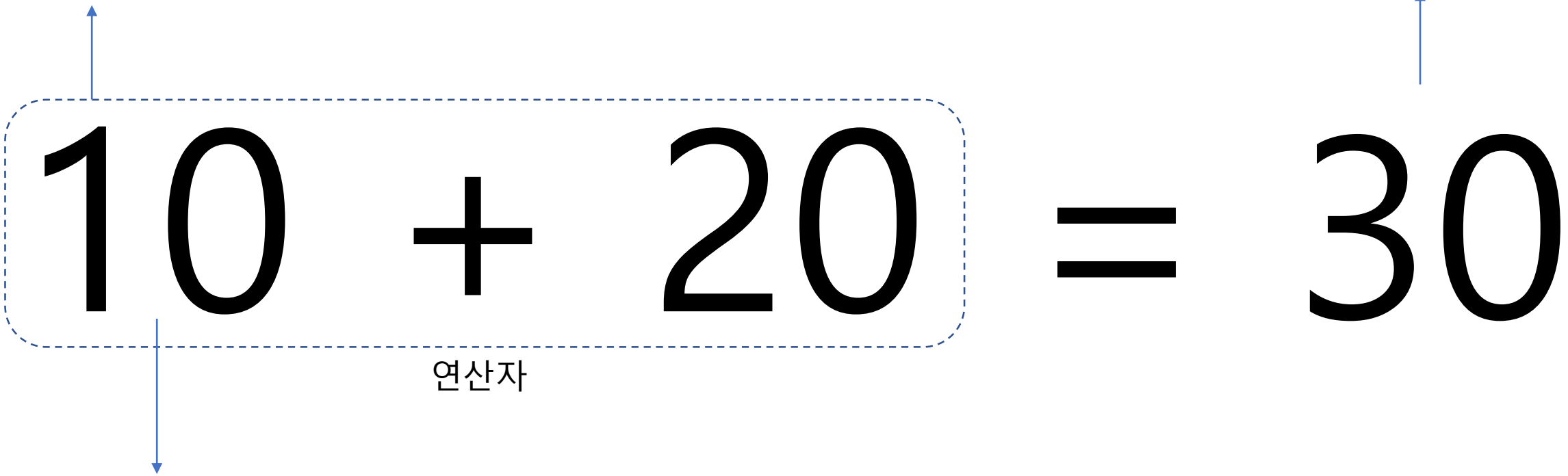
모던 자바스크립트
Deep Dive

5장_표현식과 문

용어 정리

표현식(혹은 식)
값으로 **평가**될 수 있는 문(statement)

값
식이 **평가**되어 생성된 결과



리터럴
사람이 이해할 수 있는 문자 또는 약속된 기호를 사용해 값을 생성(**평가된다**)하는 표기법
평가되기 때문에, **리터럴 표현식**이라고 적어도 무관

용어 정리

평가
'식'을 '해석'해서 값을 '생성'하거나 '참조'하는 것

예)
var sum = 10 + 20; // 변수에는 '10 + 20이 평가되어' 생성된 값 30이 변수에 할당됨
console.log(sum); // sum은 30으로 '평가'

표현식
값으로 평가될 수 있는 문(statement)
평가되면 새로운 값을 **생성**하거나 기존 값을 **참조**

예)
var score = 50 + 50; // 50 + 50은 평가되어 숫자 값 100을 생성하므로, 표현식
score; // 변수 식별자는 평가되어 기존 값 100을 참조

용어 정리

표현식
(1) 리터럴 표현식
10, ‘Hello’

(2) 식별자 표현식 (선언이 이미 존재한다고 가정)
sum, person.name, arr[1]

(3) 연산자 표현식
10 + 20 / sum = 10; / sum !== 10

(4) 함수/메서드 호출 표현식 (선언이 이미 존재한다고 가정)
square(); / person.getName();

용어 정리

표현식과 동치
표현식과 표현식이 평가된 값은 **동치**

```
1 + 2 = 3;
```

1 + 2와 3은 동치

```
var x = 1 + 2;
x + 3;
```

+의 좌항과 우항에는 '숫자' 값이 위치해야 함
x는 평가되어 '숫자' 값과 동치이므로, 사용할 수 있음

용어 정리

문(statement)
프로그램을 구성하는 기본 단위이자 최초 실행 단위
문의 집합은 프로그램이며, 문을 나열하는 것이 프로그래밍
명령문이라고도 부름. 즉, 컴퓨터에 내리는 명령

변수 선언문: var x;
할당문: x = 5;
함수 선언문: function foo() {}
조건문: if(x > 1) { console.log(x); }
반복문: for(var i = 0; i < 2; i++) { console.log(i); }

토큰
문법적인 의밀르 가지며, 문법적으로 더 이상 나눌 수 없는 코드의 기본 요소

var sum = 1 + 2;

statement: var sum = 1 + 2;
토큰: var, sum, =, 1, +, 2

용어 정리

리터럴
사람이 이해할 수 있는 문자(아라비아 숫자, 알파벳, 한글) or 약속된 기호([], {}, “”)를 사용해 값을 생성하는 표기법

3 // 숫자 리터럴

3은 아라비아 숫자가 아닌 ‘리터럴’
자바스크립트 엔진은 리터럴 3을 ‘평가’하여 숫자 ‘값’ 3을 생성

리터럴	예시	비고
정수 리터럴	100	
부동소수점 리터럴	10.5	
2진수 리터럴	0b01000001	0b로 시작
문자열 리터럴	‘Hello’, ‘World’	
불리언 리터럴	True, flase	
null 리터럴	null	
undefined		

용어 정리

세미콜론과 세미콜론 자동 삽입 기능
- 자바스크립트 엔진은 세미콜론으로 문이 종료한 위치를 파악하여 순차적으로 문을 실행

- 세미콜론은 옵션이다. 자바스크립트 엔진은 문의 끝으로 예측되는 지점에 세미콜론을 자동으로 붙여주는 ASI(Automatic Semicolon Insertion)를 수행하기 때문

- 하지만 개발자의 예측과 빗나가거나, ESLint같은 정적 분석 도구에서도 세미콜론 사용, TC39(ECMAScript 기술 위원회)도 세미콜론 사용을 권장하는 분위기 이므로, 사용하도록 하자

```
function foo() {
    return
    {}
    // ASI 예측  return; {};
    // 개발자 예측 return {};
}

console.log(foo()) // undefined;

var bar = function() {}
( function() {})(); // TypeError: (Intermediate value)(...) is not a function
// ASI 예측 var bar = function() {} (function() {})();
// 개발자 예측 var bar = function() {}; (function(){});
```

- 코드 블록은 문의 종료를 의미하는 자체 종결성(self-closing)을 갖고 있으므로, 세미콜론을 붙이지 않음
예) if문, for문, 함수 등

용어 정리

표현식인 문과 표현식이 아닌 문

```
var x; // 문이지만 표현식은 아님
x = 1 + 2; // 표현식이면서 문
```

구별하는 가장 간단하고 명료한 방법은 **변수에 할당해 보는 것**

표현식인 문은 **값으로 평가할 수 있으므로** 할당이 가능
표현식이 아닌 문은 **값으로 평가할 할 수 없으므로** 할당이 불가능

```
var foo = var x; // SyntaxError
```

```
var foo = x = 100; // x = 100은 100으로 평가
console.log(foo); // 100
```

용어 정리

완료 값(completion value)
크롬 개발자 도구에서 표현식이 아닌 문을 실행하면 언제나 undefined 출력
이를 완료 값이라 하며, 표현식의 평가 결과가 아니므로 변수에 할당할 수 없고 참조할 수 없음

```
> var foo = 10; // 변수 선언문
<- undefined
> if(true) {} // 조건문
<- undefined
```

표현식인 문을 실행하면 언제나 평가된 값을 반환

```
> var num = 10;
<- undefined
> 100 + num;
<- 110
> num = 100;
<- 100
```

4장_변수

애플리케이션

- 운영체제에서 실행되는 모든 소프트웨어
  소스코드가 아닌, 실행 파일 자체, 즉 **실제로 기능을 하는 최종 상태**

- 입력으로 데이터를 받고, 출력으로 결과물을 내놓음
  변수는 이러한 **데이터를 관리하기 위한 핵심 개념**

자바스크립트 엔진은 10 + 20을 어떻게 계산할까?

사람은 계산과 기억을 모두 두뇌에서 담당
컴퓨터는 계산은 **CPU**, 기억은 **메모리**에서 담당

메모리 특징
(1) 메모리는 데이터를 저장하는 셀의 집합체

(2) 셀 하나의 크기는 1바이트이며, 고유한 메모리 주소를 가짐
4GB 기준 0x00000000~0xFFFFFFFF까지의 메모리 주소 존재

(3) 컴퓨터는 1바이트 단위로 데이터를 저장하거나 읽음

(4) 컴퓨터는 모든 데이터를 2진수로 저장(편의상 10진수로 표시함)
모든 데이터(숫자, 텍스트, 이미지, 동영상 등)

0x00000000	
0x000000F2	10
0x00001332	20
0x669F913	
0xFFFFFFFF	

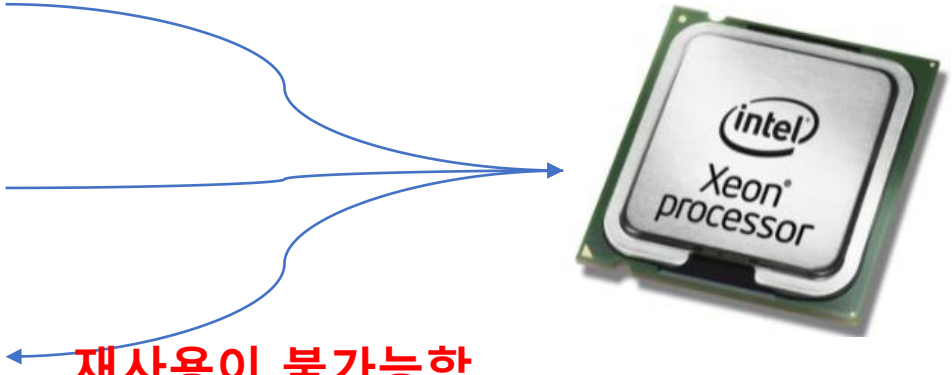
메모리 셀

메모리 주소 **메모리**

자바스크립트 엔진은 10 + 20을 어떻게 계산할까?

0x00000000	
0x000000F2	10
0x00001332	20
0x669F913	30
0xFFFFFFFF	

메모리 주소 **메모리**



재사용이 불가능함

CPU

자바스크립트 엔진은 10 + 20을 어떻게 계산할까?

0x00000000	
0x000000F2	10
0x00001332	20
0x669F913	30
0xFFFFFFFF	

메모리 주소 **메모리**

재사용이 불가능함

(1) 재사용하려면 메모리 공간에 직접 접근해야 함
이는 치명적인 오류 발생 가능성(운영체제 사용중인 값 변경)이 높음
자바스크립트는 직접적인 메모리 제어를 허용하지 않음

(2) 메모리 주소가 임의로 결정됨
코드가 실행되기 이전에는 값이 저장된 메모리 주소를 알 수 없음
코드가 실행된 후에도 알려주지 않음

이러한 문제를 **변수**를 통해서 해결이 가능함

자바스크립트 엔진은 10 + 20을 어떻게 계산할까?

프로그래밍 언어는 기억하고 싶은 값을 메모리에 저장하고, 이를 재사용하기 위해 변수 메커니즘 제공

변수는 하나의 값을 저장하기 위해 확보한 **메모리 공간 자체** 또는 **값의 위치를 가리키는 상징적인 이름**

이후 변수는 컴파일러 또는 인터프리터에 의해 값이 저장된 메모리 공간의 주소로 치환되어 실행

참고로 자바스크립트는 컴파일 과정과 인터프리트 과정이 모두 들어가 있는데, 아마도 컴파일러에 의해 실행되지 않을까?

자바스크립트 엔진은 10 + 20을 어떻게 계산할까?

var result = 10 + 20;

할당

0x00000000	
0x000000F2	10
0x00001332	20
0x669F913	30
0xFFFFFFFF	

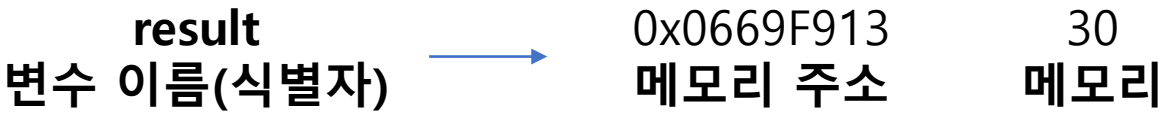
result
변수 이름(식별자)

console.log(result)

참조

자바스크립트 엔진은 10 + 20을 어떻게 계산할까?

이처럼 식별자(변수 이름)는 값이 아니라 메모리 주소를 기억



식별자라는 이름은 변수에만 국한하지 않음. **메모리 상에 존재하는 값을 식별할 수 있는 이름**을 식별자라고 함
여기에는 변수, 함수, 클래스 등의 이름도 포함

자바스크립트 엔진은 10 + 20을 어떻게 계산할까?

식별자 네이밍 규칙
(1) 특수 문자를 제외한 문자, 숫자, 언더 스코어(_), 달러 기호($) 사용 가능
(2) 단, 숫자로 시작하는 것은 허용하지 않음
(3) 예약어는 식별자로 사용 불가

※예약어
프로그래밍 언어에서 사용되고 있거나, 사용될 예정인 단어
implements, package, private 등의 단어는 strict mode에서는 사용 불가

In JavaScript you cannot use these reserved words as variables, labels, or function names:

abstract	arguments	await*	boolean
break	byte	case	catch
char	class*	const	continue
debugger	default	delete	do
double	else	enum*	eval
export*	extends*	false	final
finally	float	for	function
goto	if	implements	import*
in	instanceof	int	interface
let*	long	native	new
null	package	private	protected
public	return	short	static
super*	switch	synchronized	this
throw	throws	transient	true
try	typeof	var	void
volatile	while	with	yield

Words marked with* are new in ECMAScript 5 and 6.

자바스크립트 엔진은 10 + 20을 어떻게 계산할까?

식별자 네이밍 규칙
명명 규칙에 위배되는 케이스로는

```
var first-name;
var 1st;
var this;
```

하나의 문에 여러 개를 선언 가능

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

자바스크립트는 대소문자를 구별하므로 다음 변수는 각각 별개의 변수

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

변수 이름은 **변수의 존재 목적을 쉽게 이해할 수 있도록 의미를 명확히 표현해야 함**
변수 선언에 **별도의 주석이 필요하다면, 변수의 존재 목적을 명확히 드러내지 못하는 것**

자바스크립트 엔진은 10 + 20을 어떻게 계산할까?

네이밍 컨벤션
하나 이상의 영어 단어로 구성된 식별자를 만들 때, 가독성 좋게 단어를 구분하기 위해 규정한 명명 규칙

EMCAScript 사양에 정의되어 있는 객체와 함수들도 카멜 케이스와 파스칼 케이스를 따름

(1) 카멜 케이스 (변수나 함수)
```
var firstName;
```

(2) 스네이크 케이스
```
var first_name;
```

(3) 파스칼 케이스 (생성자 함수, 클래스 이름)
```
var FirstName;
```

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

자바스크립트 엔진은 10 + 20을 어떻게 계산할까?

변수 선언은 **변수를 생성하는 것**
즉, 값을 저장하기 위해 (1) 메모리 공간을 확보, (2) 변수 이름과 메모리 공간의 주소를 연결
변수를 선언할 때는 var, let, const 키워드 이용

※ 키워드
JS엔진이 수행할 동작을 규정한 일종의 명령으로, JS엔진은 키워드를 만나면 약속된 동작을 수행
예를들어 var 키워드를 만나면, 뒤에 오는 변수 이름으로 새로운 변수를 선언

중요한 것은 값의 할당과 선언은 분리된다는 점
자바스크립트 엔진은 소스 코드를 두 가지 과정으로 나누어서 실행

(1) 평가 (2) 실행

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

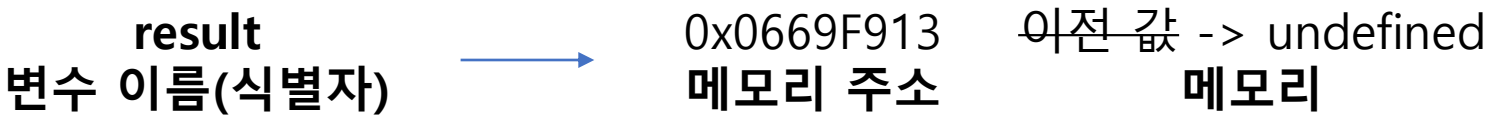
평가(선언)

실행(할당)

자바스크립트 엔진은 10 + 20을 어떻게 계산할까?

변수 선언은 다시 두 단계로 나뉨
(1) 선언 단계 변수 이름을 실행 컨텍스트의 전역 환경 레코드에 등록
(2) 초기화 단계 메모리 공간 확보 및 연결, 암묵적인 undefined 할당

초기화 단계를 통해서 이전에 다른 애플리케이션이 사용했던 값인 쓰레기값(garbage value)를 참조할 가능성 있음



※ 자바스크립트 엔진은 실행 컨텍스트를 통해서
(1) 소스코드 평가하고 실행하기 위한 환경
(2) 코드 실행 결과 관리
(3) 식별자와 스코프를 관리

자바스크립트 엔진은 10 + 20을 어떻게 계산할까?

변수 선언과 값의 할당 두 가지 방법
(1) 변수 선언과 값의 할당을 하나의 문으로 표현
(2) 변수 선언과 값의 할당을 나누어 표현하는 방법
(3) 값의 재할당

(1) 변수 선언과 값의 할당을 하나의 문으로 표현

```
1   console.log(score); // undefined
2
3   var score = 80; // 변수 선언과 값의 할당
4
5   console.log(score); // 80
```

자바스크립트 엔진은 10 + 20을 어떻게 계산할까?

변수 선언과 값의 할당 세 가지 방법
(2) 변수 선언과 값의 할당을 나누어 표현하는 방법

```
 9   console.log(score); // undefined
10
11   var score; // 1. 변수 선언
12   score = 80; // 2. 값의 할당
13
14   console.log(score); // 80
```

score

0x00000000	
0x669F913	undefined
0x728F918	
0xFFFFFFFF	

var score;

score

0x00000000	
0x669F913	Undefined
0x728F918	80
0xFFFFFFFF	

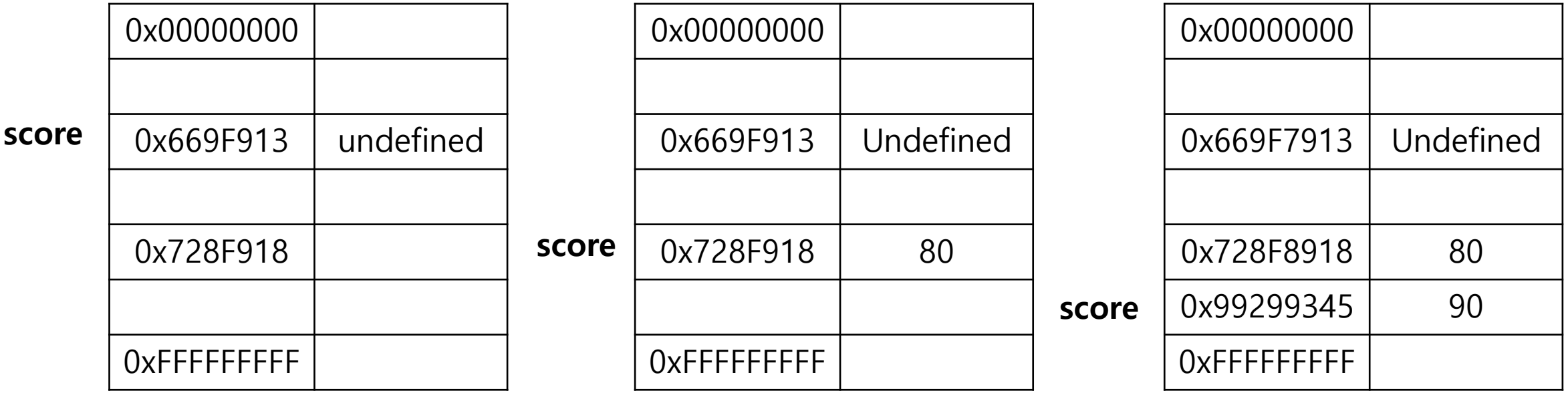
score = 80;

자바스크립트 엔진은 10 + 20을 어떻게 계산할까?

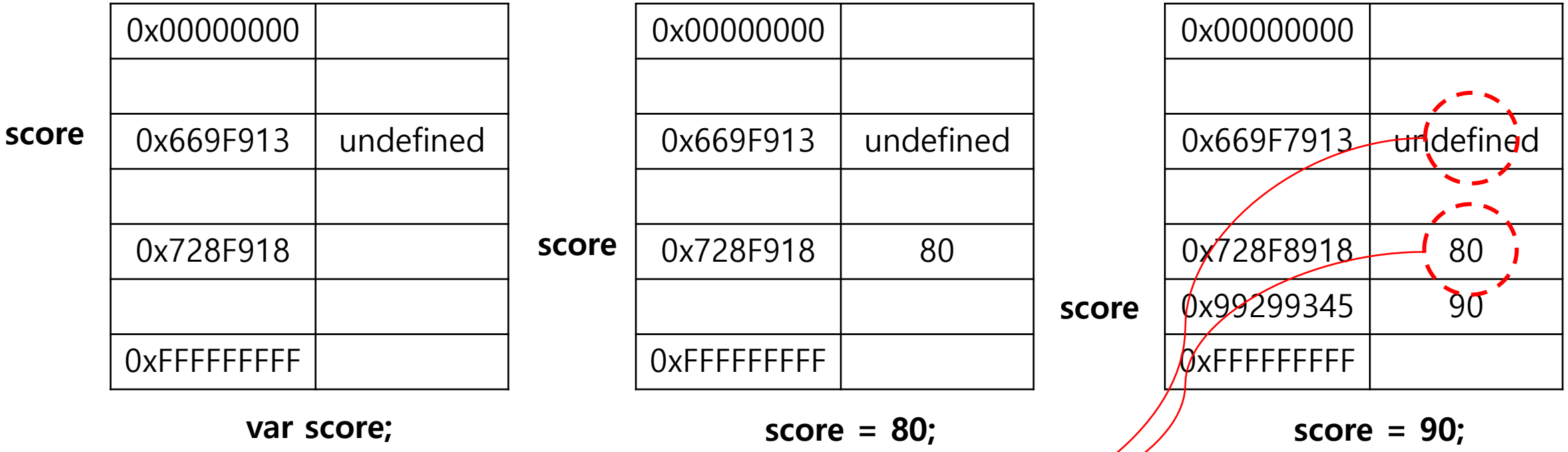
변수 선언과 값의 할당 세 가지 방법
(3) 값의 재할당
재할당은 이미 값이 할당되어 있는 변수에 새로운 값을 할당하는 것(값을 재할당 할 수 없는 경우 상수라고 함)

참고로, var키워드로 선언한 변수는 선언과 동시에 undefined로 초기화되기 때문에, 처음 변수에 할당하는 값은 엄밀히 말하면, 재할당된 값임

var score;
score = 80
score = 90;



자바스크립트 엔진은 10 + 20을 어떻게 계산할까?



어떤 식별자도 참조하지 않는 값들은 **가비지 콜렉터**에 의해 제거

자바스크립트 엔진은 10 + 20을 어떻게 계산할까?

가비지 콜렉터(garbage collector)
메모리 공간을 주기적으로 검사하여 더 이상 사용되지 않는 메모리를 해제하는 기능
더 이상 사용되지 않는 메모리는 어떤 식별자도 참조하지 않는 메모리 공간을 의미
자바스크립트는 매니지드 언어로서, 이를 통해 메모리 누수 방지

매니지드 언어와 언매니지드 언어
프로그래밍 언어는 메모리 관리 방식에 따라 매니지드 언어와 언매니지드 언어로 나뉨

C언어는 개발자가 명시적으로 메모리를 할당하고 해제하기 위한 malloc과 free 같은 저수준 메모리 제어 기능 제공
메모리 제어를 개발자가 주도할 수 있으므로, 최적의 성능을 확보할 수 있지만, 치명적 오류 생산 가능성도 존재

자바스크립트는 메모리의 할당 및 해제등의 메모리 관리 기능을 언어 차원에서 담당
더 이상 사용하지 않는 메모리의 해제는 가비지 콜렉터가 수행하므로, 생산성을 확보할 수 있지만, 성능 면에서 어느 정도의 손실은 감수해야 함

비동기 통신

AJAX(Asynchronous Javascript and XML)

- 브라우저가 서버에게 **비동기 방식으로 데이터를 요청**하여 웹페이지를 **동적으로 갱신**하는 프로그래밍 방식
- WebAPI인 XMLHttpRequest 객체를 기반으로 동작. 이 객체는 HTTP 비동기 통신을 위한 메서드와 프로퍼티 제공

JSON(JavaScript Object Notation)

- 클라이언트와 서버간 HTTP 통신을 위한 **텍스트 데이터 포맷**
- 대부분의 프로그래밍 언어에서 사용 가능

fetch API

- fetch API는 비동기 방식
- fetch().then(function()) fetch()해서 가져온 데이터에 대해서 then의 인자에 있는 함수 실행
- fetch().then(function(response){})에서 response는, 서버가 응답한 결과를 담고 있는 객체 데이터
- fetch()로는 데이터를 바로 사용할 수 없음. response.json()을 통해서 JSON 형태로 파싱

```
const cities = [];

// JSON 형식의 파일을 '비동기 방식'으로 받아옴
fetch('https://gist.githubusercontent.com/Miserlou/c5cd8364bf9b2420bb29/raw/2bf258763cdddd704f8ffd3ea9a3e81d25e2c6f6/cities.json')
.then(response => response.json())
.then(data => cities.push(...data));
```

Synchronous vs Asychronous

1. 동기

```
// synchronous는 정해진 순서에 맞게 코드가 실행된다
console.log('1');
console.log('2');
console.log('3');
// output : 1, 2, 3
```

2. 비동기

```
// asynchronous는 언제 코드가 실행될지 예측할 수 없다
console.log('1');
setTimeout(() => { // 브라우저에 요청
    console.log('2');
}, 1000);
console.log('3');
// output 1, 3, 2

// Synchronous callback
function printImmediately(print) {
    print();
}

printImmediately(() => console.log('hello'));
// output 1, 3, hello, 2

// Asynchronous callback
function printWithDelay(print, timeout) {
    setTimeout(print,timeout);
}
printWithDelay(()=>console.log('async callback'), 2000);
// output 1, 3, hello, 2, async callback
```

Promise

ㅈㅂㄷㄱㅈㄱㅂㅈㄱ