6장 데이터 타입

데이터 타입은 값의 종류를 말한다.

자바스크립트의 모든 값은 데이터 타입을 갖는다. 자바스크립트 (ES6)는 7개의 데이터 타입을 제공한다.

7개의 데이터 타입은 **원시 타입**과 **객체 타입**으로 분류할 수 있다.

원시 타입

- number : 숫자, 정수와 실수 구분 없이 하나의 숫자 타입만 존재.
- string : 문자열
- boolean : 논리적 참과 거짓
- undefined : var 키워드로 선언된 변수에 암묵적으로 할당되는 값
- null : 값이 없다는 것을 의도적으로 명시할 때 사용하는 값
- symbol : ES6에서 추가된 7번째 타입

객체 타입

- 객체, 함수, 배열 등

숫자 타입

C나 자바의 경우, 정수와 실수를 구분해서 int, long, float, double 등과 같은 다양한 숫자 타입을 제공한다.

하지만 자바스크립트는 독특하게 하나의 숫자 타입만 존재한다.

ECMAScript 사양에 따르면 숫자 타입의 값은 **배정밀도 64비트 부동 소수점 형식**을 따른다.

즉, **모든 수를 실수로 처리하며, 정수만 표현하기 위한 데이터 타입이 별도로 존재하지 않는다.**

```
var integer = 10; // 정수
var double = 10.12; // 실수
var negative = -20; // 음의 정수
```

정수, 실수, 2진수, 8진수, 16진수 리터럴은 모두 메모리에 배정밀도 64비트 부동소수점 형식의 2진수로 저장된다.

자바스크립트는 2진수,8진수,16진수를 표현하기 위한 데이터 타입을 제공하지 않기 때문에 이들 값을 참조하면 **모두 10진수**로 해석된다.

`자바스크립트에서 number 타입은 정수 값으로 최대 2^53-1, 최소 -(2^53-1)(9007199254740991)까지 저장할 수 있다.`

`number 타입은 최대 1.797631348623157 * 10^308까지 저장할 수 있지만, 정수 범위 2^53-1 가 벗어`

`나면, 정밀도 오차가 발생한다. 그 이유는 모든 10진 숫자가 64비트 안에 저장될 수 없기 때문이다.`

`따라서 범위가 초과할 경우 대략적인 값이 저장된다.`

```
console.log(9007199254740991 + 1); // 9007199254740992
console.log(9007199254740991 + 2); // 9007199254740992
```

즉, `2^53-1` 보다 큰 모든 홀수 정수는 '숫자' 유형에 전혀 저장할 수 없다.

대부분의 경우 `+- (2^53-1)` 범위면 충분하지만, 암호화나 마이크로 초 단위의 정밀한 타임스탬프와 같이 매우 큰 정수의 전체 범위가 필요한 경우도 있다.

임의의 길이의 정수를 표현하기 위해 최근 언어에 BigInt 유형이 추가되었다. (ES6보다 다음 버전인 듯?)

BigInt 타입은 정수뒤에 n을 추가함으로써 만들 수 있다.

```
// the "n" at the end means it's a BigInt
const bigInt = 12345678901234567890123456789012345678901234567890123456789;
```

자바스크립트의 숫자 타입은 정수 만을 위한 타입이 없고 모든 수를 실수로 처리한다.

이는 정수로 표시된다고 해도 사실은 실수라는 것을 의미한다.

숫자 타입은 추가적으로 세 가지 특별한 값도 표현할 수 있다.

- Infinity : 양의 무한대
- -Infinity : 음의 무한대
- NaN : 산술 연산 불가 (not-a-number)

```
console.log(10 / 0); // Infinity
console.log(10 / -0); // -Infinity
console.log(1 * 'String'); // NaN
```

자바스크립트는 대소문자를 구별하므로 NaN을 NaN, Nan, nan과 같이 표현하면 에러가 발생하므로 주의해야 한다.

자바크스립트 엔진은 NAN, Nan, nan을 값이 아닌 식별자(변수 명)으로 해석한다.

문자열 타입

문자열 타입은 텍스트 데이터를 나타내는 데 사용한다. 문자열은 0개 이상의 16비트 유니코드 문자의 집합으로 전 세계 대부분의 문자를 표현할 수 있다.

문자열은 작은 따옴표(''), 큰 따옴표("") 또는 백틱으로 텍스트를 감싼다.

```
var string;
string = '문자열'; // 작은 따옴표
string = "문자열"; // 큰 따옴표
string = `문자열`; // 백틱(ES6)
string = '작은 따옴표로 감싼 문자열 내의 "큰 따옴표는" 문자열로 인식된다.';
string = "큰따옴표로 감싼 문자열 내의 '작은 따옴표는' 문자열로 인식된다.";
```

다른 타입의 값과 달리 문자열을 따옴표로 감싸는 이유는 키워드나 식별자 같은 토큰과 구분하기 위해서이다.

만약 문자열을 따옴표로 감싸지 않으면 자바스크립트 엔진은 키워드나 식별자 같은 토큰으로 인식한다.

템플릿 리터럴

ES6부터 템플릿 리터럴이라고 하는 새로운 문자열 표기법이 도입됐다.

템플릿 리터럴은 **멀티라인 문자열, 표현식 삽입, 태그드 템플릿** 등 편리한 문자열 처리 기능을 제공한다.

템플릿 리터럴은 일반 문자열과 비슷해 보이지만 작은 따옴표 (''), 큰 따옴표 ("") 같은 일반적인 따옴표 대신 백틱 (``)을 사용해 표현한다.

```
var template = `Template literal`;
console.log(template); // Template literal
```

멀티라인 문자열

일반 문자열 내에서는 줄바꿈(개행)이 허용되지 않는다.

```
var str = 'Hello
world.';
```

따라서 일반 문자열 내에서 줄바꿈 등의 공백을 표현하려면 백슬래시(\)로 시작하는 이스케이프 시퀀스를 사용해야 한다.

이스케이프 시퀀스	의미
\0	Null
\b	백스페이스
\f	폼 피드Form Feed: 프린터로 출력할 경우 다음 페이지의 시작 지점으로 이동한다.
\n	개행LF, Line Feed: 다음 행으로 이동
\r	개행CR, Carriage Return: 커서를 처음으로 이동
\t	탭(수평)
\v	탭(수직)
\uXXXX	유니코드. 예를 들어 '\u0041'은 'A', '\uD55C'는 '한', '\u{1F600}'는 😀이다.
\'	작은따옴표
\"	큰따옴표
\\	백슬래시

❗⭐라인 피드와 캐리지 리턴⭐❗

개행 문자는 텍스트의 한 줄이 끝남을 표시하는 문자 또는 문자열이다.

개행 문자에는 라인 피드와 캐리지 리턴이 있다. 이는 과거 타자기에서 커서를 제어하는 방식에서 비롯된 것이다.

라인 피드(\n)는 커서를 정지한 상태에서 종이를 한 줄 올리는 것이고, 캐리지 리턴(\r)은 종이를 움직이지 않고 커서를 맨 앞 줄로 이동하는 것이다.

초창기 컴퓨터는 출력을 프린터로 수행했는데, 이때 개행을 위해 라인 피드와 캐리지 리턴을 모두 사용했다. 즉, **CRLF(\r\n)로 커서를 맨 앞으로 이동시키고** 종이를 한 줄 올리는 방식으로 개행했다.

현대의 컴퓨터 운영체제는 서로 다른 체계의 개행 방식을 사용한다.

윈도우는 CR+LF(ASCII 코드 13번 과 10번)로 새 줄을 나타내고 유닉스는 LF(ASCII 코드 10번), macOS는 버전 9까지 CR로 새 줄을 나타냈지만 버전 10부터 LF를 사용한다.

따라서 다른 운영체제에서 작성한 텍스트 파일은 서로 개행 문자를 인식하지 못한다. 다만 대부분의 텍스트 에디터는 운영체제에 맞게 개행 문자를 자도응로 변환해주므로 큰 문제는 없다.

자바스크립트에서 라인 피드와 캐리지 리턴은 모두 개행을 의미한다.

하지만 캐리지 리턴(\r)으로 개행하는 경우는 거의 없고 일반적으로 라인 피드(\n)를 사용해 개행한다.

```
var template = '<ul>\n\t<li><a href="#">Home</a><li>\n<ul>';

console.log(template);
```

표현식 삽입

문자열은 문자열 연산자 +를 사용해 연결할 수 있다. + 연산자는 피연산자 중 하나 이상이 문자열인 경우 문자열 연결 연산자로 동작한다. 그 외의 경우는 덧셈 연산자로 동작한다.

```
var first = 'Ung-mo';
var last = 'Lee';

console.log('My name is' + first + ' ' + last + '.');
// My name is Ung-mo Lee.
```

템플릿 리터럴 내에서는 표현식 삽입을 통해 간단히 문자열을 삽입할 수 있다.

이를 통해 문자열 연산자보다 가독성 좋고 간편하게 문자열을 조합할 수 있다.

```
console.log(`My name is ${first} ${last}`);
// My name is Ung-mo Lee.
```

불리언 타입

불리언 타입의 값은 논리적 참, 거짓을 나타내는 true와 false 뿐이다.

불리언 타입의 값은 참과 거짓으로 구분되는 조건에 의해 프로그램의 흐름을 제어하는 조건문에서 자주 사용한다.

undefined

undefined 타입의 값은 undefined가 유일하다.

var 키워드로 선언한 변수는 암묵적으로 undefined로 초기화된다.

변수 선언에 의해 확보된 메모리 공간을 처음 할당이 이뤄질 때까지 빈 상태로 내버려 두지 않고 자바스크립트 엔진이 undefined로 초기화 한다.

undefined는 자바스크립트 엔진이 초기화되지 않는 변수라는 것을 의미하는 것이므로, 개발자가 직접 할당하는 것은 권장하지 않는다.

선언(declare) 과 정의(definition)

undefined를 직역하면 “정의되지 않은”이다.

자바스크립트에서 undefined가 말하는 정의란 변수에 값을 할당하여 변수의 실체를 명확히 하는 것을 말한다.

C언어에서 선언과 정의는 “실제로 메모리 주소를 할당하는가”로 구분한다. 단순히 컴파일러에게 식별자의 존재만 알리는 것은 선언이고, 실제로 컴파일러가 변수를 생성해서 식별자와 메모리 주소가 연결되는 것은 정의로 구분한다.

자바스크립트의 경우에는 변수를 선언하면 암묵적으로 정의가 이루어지므로 경계가 애매하다.

ECMAScript 에서는 변수는 '선언한다'라고 표현하고, 함수는 '정의한다'라고 표현한다.

Null 타입

프로그래밍 언어에서 null은 변수에 값이 없다는 것을 의도적으로 명시할 때 사용한다.

변수에 null을 할당하는 것은 변수가 이전에 참조하던 값을 더 이상 참조하지 않겠다는 의미다.

자바스크립트 엔진은 누구도 참조하지 않는 메모리 공간에 대해 가비지 컬렉션을 수행한다.

또한 함수가 유효한 값을 반환할 수 없는 경우 명시적으로 null을 반환하기도 한다.

HTML요소를 검색해 반환하는 `document.querySelector` 메소드는 조건에 부합하는 HTML 요소를 검색할 수 없는 경우 에러 대신 null을 반환한다.

심벌 타입

심벌은 ES6에서 추가된 7번째 타입으로, 변경 불가능한 원시 타입의 값이다.

심벌 값은 다른 값과 중복되지 않는 유일 무이한 값이다.

따라서 주로 이름이 충돌할 위험이 없는 객체의 유일한 프로퍼티 키를 만들기 위해 사용한다.

심벌 이외의 원시 값은 리터럴을 통해 생성하지만 심벌은 Symbol 함수를 호출해 생성한다. 이때 생성된 심벌 값은 외부에 노출되지 않으며, 다른 값과 절대 중복되지 않는 유일 무이 한 값이다.

```
var key = Symbol("key");

var obj = {};
```

```
// 이름이 충돌할 위험이 없는 유일무이한 값인 심벌을 프로퍼티 키로 사용한다.
obj[key] = 'value';
console.log(obj[key]); // value
```

객체 타입

자바스크립트의 데이터 타입은 크게 원시형 타입과 객체 타입으로 분류한다고 했다.

이는 원시 타입과 객체 타입은 근본적으로 다르다는 의미이다.

중요한 것은 자바스크립트는 객체 기반의 언어이며, 자바스크립트를 이루고 있는 거의 모든 것이 객체라는 것이다.

데이터 타입의 필요성

데이터 타입에 의한 메모리 공간의 확보와 참조

값의 저장

값은 메모리에 저장하고 참조할 수 있어야 한다. 메모리에 값을 저장하기 위해서는 먼저 확보해야 할 메모리 공간의 크기를 결정해야 한다.

`var score = 100;`

위와 같이 변수를 선언하고 값을 할당했다고 하자. 위 코드가 실행되면 컴퓨터는 아래와 같은 작업을 수행한다.

1. 숫자 값 100을 저장하기 위해 메모리 공간 확보
2. 확보된 메모리에 값 100을 2진수로 저장

자바스크립트 엔진은 데이터 타입, 즉 값의 종류에 따라 정해진 크기의 메모리 공간을 확보한다.

즉, 변수에 할당되는 값의 데이터 타입에 따라 확보해야 할 메모리 공간의 크기가 결정된다.

Number 타입의 경우에는, 숫자 리터럴을 저장하기 위해 8바이트의 공간을 확보한다. (자바스크립트 에서는 정수, 실수 구분 없이 모든 숫자를 64비트 배정밀도 부동 소수점 방식을 사용해 표현하므로 8바이트가 맞다.)

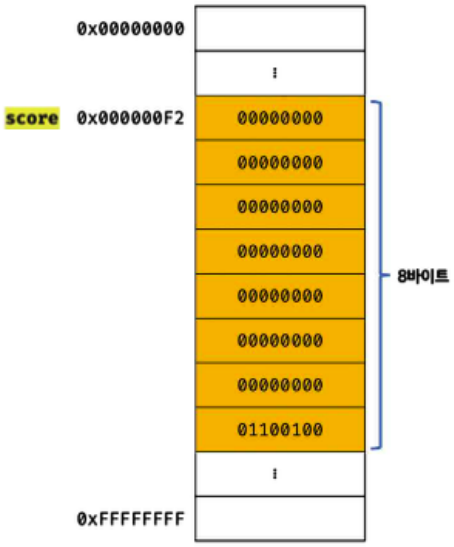

그림 6-1 숫자 타입 값의 할당

데이터 타입에 따라 확보되는 메모리 공간의 크기

ECMAScript 사양은 문자열과 숫자 타입 이외의 데이터 타입의 크기를 명시적으로 규정하고 있지는 않다.

따라서 문자열과 숫자 타입을 제외하고 데이터 타입에 따라 확보되는 메모리 공간의 크기는 자바스크립트 엔진 제조사의 구현에 따라 다를 수 있다.

단, ECMAScript 사양에 숫자 타입은 배정밀도 64비트 부동소수점 형식을 사용한다고 명시되어 있고, 배정밀도 64비트 부동소수점 형식은 8바이트로 숫자를 표현한다.

값의 참조

값을 참조하는 경우, 식별자 score를 통해 숫자 타입의 값 100이 저장되어 있는 메모리 공간의 주소를 찾아갈 수 있다.

정확히 말하면, 값 100이 저장되어 있는 메모리 공간의 선두 메모리 셀의 주소를 찾아갈 수 있다.

이때 값을 참조하기 위해서는 한 번에 읽어 들여야 할 메모리 공간의 크기, 메모리 셀의 개수(바이트 수)를 알아야 한다.

score 변수에는 숫자 타입의 값이 할당되어 있어 자바스크립트 엔진은 score 변수를 숫자 타입으로 인식한다.

숫자 타입은 8바이트 단위로 저장되므로 score 변수를 참조하면 8바이트 단위로 메모리 공간에 저장된 값을 읽어 들인다.

심벌 테이블

컴파일러 또는 인터프리터는 심벌 테이블이라고 부르는 자료 구조를 통해 식별자를 키로 바인딩된 값의 메모리 주소, 데이터 타입, 스코프 등을 관리한다.

데이터 타입에 의한 값의 해석

아직 문제가 남아있다. 메모리에서 읽어 들인 2진수를 어떻게 해석해야 하느냐이다.

모든 값은 데이터 타입을 가지며, 메모리에 2진수, 즉 비트의 나열로 저장된다.

메모리에 저장된 값은 데이터 타입에 따라 다르게 해석될 수 있다. 예를 들어, 메모리에 저장된 값 0100 0001을 숫자로 해석하면 65지만 문자열로 해석하면 'A' 이다.

데이터 타입은 값의 종류를 말한다. 자바스크립트의 모든 값은 데이터 타입을 갖는다.

- 값을 저장할 때 확보해야 하는 메모리 공간의 크기를 결정하기 위해
- 값을 참조할 때 한 번에 읽어들여야 할 메모리 공간의 크기를 결정하기 위해
- 메모리에서 읽어 들인 2진수를 어떻게 해석할지 결정하기 위해

동적 타이핑

동적 타입 언어와 정적 타입 언어

자바스크립트의 모든 값은 데이터 타입을 갖는다고 했다.

그렇다면 변수는 데이터 타입을 가질까? C나 자바 같은 정적 타입 언어는 변수를 선언할 때 변수에 할당할 수 있는 값의 종류, 즉 데이터 타입을 사전에 선언해야 한다. 이를 명시적 타입 선언(explicit type declareation)이라 한다.

정적 타입 언어는 변수의 타입을 변경할 수 없으며, 변수에 선언한 타입에 맞는 값만 할당할 수 있다.

정적타입 언어는 컴파일 시점에 **타입 체크**(선언한 데이터 타입에 맞는 값을 할당했는지 검사하는 처리)를 수행한다.

자바스크립트의 변수는 어떠한 데이터 타입의 값이라도 자유롭게 할당할 수 있다.

변수를 하나 선언하고 어떤 값을 할당한 다음, `typeof` 연산자로 변수의 데이터 타입을 조사해 보자.

```
var foo;
console.log(typeof foo); // undefined

foo = 3;
console.log(typeof foo); // number

foo = "Hello";
console.log(typeof foo); // string

foo = true;
console.log(typeof foo); // boolean
```

`typeof` 연산자로 변수를 연산하면 변수의 데이터 타입을 반환한다.

정확히 말하면 변수의 데이터 타입을 반환하는 것이 아니라, 변수에 할당된 값의 데이터 타입을 반환한 것이다.

정적 타입 언어는 변수 선언 시점에 변수의 타입이 결정되고 변수의 타입을 변경할 수 없다.

하지만 자바스크립트의 변수는 선언이 아닌 할당에 의해 타입이 결정(타입 추론(type inference))된다. 그리고 재할당에 의해 변수의 타입은 언제든지 동적으로 변할 수 있다. 이러한 특징을 동적 타이핑(dynamic typing)이라 한다.

자바스크립트는 이러한 성질을 가지며 정적 타입 언어와 구별하기 위해 동적 타입 언어(dynamic/weak type)언어라 한다. (python, php, Lisp, perl 등)

동적 타입 언어와 변수

동적 타입 언어는 변수에 어떤 데이터 타입의 값이라도 자유롭게 할당할 수 있다.

모든 소프트웨어 아키텍처에는 트레이드 오프가 존재하며, 모든 애플리케이션에 적합한 silver bullet은 없듯이 동적 타입 언어 또한 구조적인 단점이있다.

변수 값은 언제든지 변경될 수 있으므로 복잡한 프로그램에서는 값의 변화를 추적하기 어려울 수 있다.

자바스크립트는 개발자의 의도와는 상관없이 자바스크립트 엔진에 의해 암묵적으로 타입이 자동으로 변환되기도 한다.

결국 동적 타입 언어는 유연성(flexibility)은 높지만 신뢰성(reliablity)은 떨어진다.

💡 **변수 사용 주의사항**

-

변수는 꼭 필요한 경우에 한해 제한적으로 사용한다.

- 변수 값은 재할당에 의해 언제든지 변경될 수 있다.
  → 이로 인해 타입을 잘못 예측하여 오류가 발생할 가능성이 크다.

- 변수의 유효 범위(스코프)는 최대한 좁게 만들어 변수의 부작용을 억제한다.

- 전역 변수는 최대한 사용을 지양한다.

- 변수보다는 상수를 사용해 값의 변경을 억제한다.

- 변수 이름은 변수의 목적이나 의미를 파악할 수 있도록 네이밍 한다.