6. 데이터 타입

TABLE OF CONTENTS

데이터 타입(data type. 줄여서 타입이라고도 부른다)은 값의 종류를 말한다. 자바스크립트의 모든 값은 데이터 타입을 갖는다.

자바스크립트(ES6)는 7개의 데이터 타입을 제공한다. 7개의 데이터 타입은 원시 타입(primitive type)과 객체 타입(object/reference type)으로 분류할 수 있다.

- 원시 타입(primitive type)
  - 숫자(number) 타입: 숫자. 정수와 실수 구분없이 하나의 숫자 타입만 존재
  - 문자열(string) 타입: 문자열
  - 불리언(boolean) 타입: 논리적 참(true)과 거짓(false)
  - undefined 타입: var 키워드로 선언된 변수에 암묵적으로 할당되는 값
  - null 타입: 값이 없다는 것을 의도적으로 명시할 때 사용하는 값
  - Symbol 타입: ES6에서 새롭게 추가된 7번째 타입
- 객체 타입 (object/reference type): 객체, 함수, 배열 등

예를 들어 숫자(number) 타입의 값 1과 문자열(string) 타입의 값 '1'은 비슷하게 보이지만 전혀 다른 값이다. 숫자 타입의 값 1과 문자열 타입의 값 '1'은 값을 생성한 목적과 용도가 다르다. 숫자 타입의 값은 주로 산술 연산을 위해 생성하지만 문자열 타입의 값은 주로 텍스트를 화면에 출력하기 위해 생성한다. 또한 확보해야 할 메모리 공간의 크기도 다르고 메모리에 저장되는 2진수도 다르며 읽어 들여 해석하는 방식도 다르다.

이처럼 개발자는 명확한 의도를 가지고 타입을 구별하여 값을 생성할 것이고 자바스크립트 엔진은 타입을 구별하여 값을 취급할 것이다. 자바스크립트가 제공하는 데이터 타입의 특징에 대해 살펴보도록 하자.

#1. 숫자 타입

C나 Java의 경우, 정수(소숫점 이하가 없는 숫자)와 실수(소숫점 이하가 있는 숫자)를 구분하여 int, long, float, double 등과 같은 다양한 숫자 타입이 존재한다. 하지만 자바스크립트는 독특하게 하나의 숫자 타입만 존재한다.

ECMAScript 사양에 따르면 숫자 타입의 값은 배정밀도 64비트 부동소수점 형식(double-precision 64-bit floating-point format)을 따른다. 즉, 모든 수를 실수로 처리하며 정수만을 표현하기 위한 데이터 타입(integer type)이 별도로 존재하지 않는다.

JAVASCRIPT

```
// 모두 숫자 타입이다.
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 10진수로 해석된다.

JAVASCRIPT

```
var binary = 0b01000001; // 2진수
var octal = 0o101;       // 8진수
var hex = 0x41;          // 16진수

// 표기법만 다를 뿐 모두 같은 값이다.
console.log(binary); // 65
console.log(octal);  // 65
console.log(hex);    // 65
console.log(binary === octal); // true
console.log(octal === hex);    // true
```

자바스크립트의 숫자 타입은 정수만을 위한 타입이 없고 모든 수를 실수로 처리한다고 했다. 정수로 표시된다 해도 사실은 실수다. 따라서 정수로 표시되는 수 끼리 나누더라도 실수가 나올 수 있다.

JAVASCRIPT

```
// 숫자 타입은 모두 실수로 처리된다.
console.log(1 === 1.0); // true
console.log(4 / 2);     // 2
console.log(3 / 2);     // 1.5
```

숫자 타입은 추가적으로 3가지 특별한 값들도 표현할 수 있다.

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

JAVASCRIPT

```
// 숫자 타입의 3가지 특별한 값
console.log(10 / 0);        // Infinity
console.log(10 / -0);       // -Infinity
console.log(1 * 'String'); // NaN
```

자바스크립트는 대소문자를 구별(case-sensitive)하므로 NaN을 NAN, Nan, nan과 같이 표현하면 에러가 발생하니 주의하기 바란다.

JAVASCRIPT

```
// 자바스크립트는 대소문자를 구별한다.
var x = nan; // ReferenceError: nan is not defined
```

2. 문자열 타입

문자열(string) 타입은 텍스트 데이터를 나타내는데 사용한다. 문자열은 0개 이상의 16bit 유니코드 문자(UTF-16) 들의 집합으로 전세계 대부분의 문자를 표현할 수 있다.

문자열은 작은 따옴표(‘’), 큰 따옴표(“”) 또는 백틱(``)으로 텍스트를 감싼다. 자바스크립트에서 가장 일반적인 표기법은 작은 따옴표를 사용하는 것이다.

JAVASCRIPT

```
// 문자열 타입
var string;
string = '문자열'; // 작은 따옴표
string = "문자열"; // 큰 따옴표
string = `문자열`; // 백틱 (ES6)

string = '작은 따옴표로 감싼 문자열 내의 "큰 따옴표"는 문자열로 인식된다.';
string = "큰 따옴표로 감싼 문자열 내의 '작은 따옴표'는 문자열로 인식된다.";
```

다른 타입의 값과는 달리 문자열을 따옴표로 감싸는 이유는 키워드나 식별자와 같은 토큰과 구분하기 위함이다. 만약 문자열을 따옴표로 감싸지 않으면 자바스크립트 엔진은 키워드나 식별자와 같은 토큰으로

인식한다.

JAVASCRIPT

```
// 따옴표로 감싸지 않은 hello를 식별자로 인식한다.
var string = hello; // ReferenceError: hello is not defined
```

그리고 만약 따옴표로 문자열을 감싸지 않는다면 스페이스와 같은 공백 문자도 포함시킬 수 없다.

C나 Java와 같은 언어와는 다르게 자바스크립트의 문자열은 원시 타입이며 변경 불가능한 값(immutable value)다. 이것은 문자열이 생성되면 그 문자열을 변경할 수 없다는 것을 의미한다. 이에 대해서는 "11.1.2. 문자열과 불변성"에서 살펴보기로 하자.

3. 템플릿 리터럴

ES6부터 템플릿 리터럴(template literal)이라고 불리는 새로운 문자열 표기법이 도입되었다. 템플릿 리터럴은 멀티라인 문자열(multi-line string), 표현식 삽입(expression interpolation), 태그드 템플릿(tagged template) 등 편리한 문자열 처리 기능을 제공한다. 템플릿 리터럴은 런타임에 일반 문자열로 변환되어 처리된다.

템플릿 리터럴은 일반 문자열과 비슷해 보이지만, 작은 따옴표(') 또는 큰 따옴표("") 같은 일반적인 따옴표 대신 백틱(backtick) `를 사용한다.

JAVASCRIPT

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

3.1. 멀티라인 문자열

일반 문자열 내에서 줄바꿈은 허용되지 않는다.

JAVASCRIPT

```
var str = 'Hello
world.';
// SyntaxError: Invalid or unexpected token
```

일반 문자열 내에서 줄바꿈(개행)을 하려면 백슬래시(\)로 시작하는 이스케이프 시퀀스(escape sequence)를 사용하여야 한다.

이스케이프 시퀀스	의미
\0	Null
\b	백스페이스
\f	폼 피드(Form Feed): 프린트 출력시 다음 페이지의 시작으로 이동한다.
\n	개행(LF, Line Feed): 다음 행으로 이동
\r	개행(CR, 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.
\'	작은 따옴표
\"	큰 따옴표
\\	백슬래시

라인 피드와 캐리지 리턴

개행(newline) 문자는 텍스트의 한 줄이 끝남을 표시하는 문자 또는 문자열이다. 개행 문자에는 라인 피드(LF, Line Feed)와 캐리지 리턴(CR, Carriage Return)이 있다. 이는 과거 타자기에서 커서를 제어하는 방식에서 비롯된 것이다. 라인 피드(\n)은 커서는 정지한 상태에서 종이를 한 줄 올리는 것이고, 캐리지 리턴(\r)은 종이는 움직이지 않고 커서를 맨 앞줄로 이동하는 것이다. 초창기 컴퓨터는 출력을 프린터로 수행했는데 이때 개행을 위해 라인 피드와 캐리지 리턴을 모두 사용하였다. 즉, CRLF(\r\n)로 커서를 맨 앞으로 이동시키고 종이를 한 줄 올리는 방식으로 개행하였다.

현대의 컴퓨터 운영 체제는 서로 다른 체계의 개행 방식을 사용한다. 윈도우는 CR+LF(ASCII 코드 13번과 10번)로 새줄을 나타내고 유닉스는 LF(ASCII 코드 10번)로 새줄을 나타낸다. 맥OS은 버전 9까지 CR로 새줄을 나타냈지만 버전 10부터 LF를 사용하고 있다. 따라서 다른 운영 체제에서 작성한 텍스트 파일은 서로 개행 문자를 인식하지 못한다. 다만 대부분의 텍스트 에디터는 운영 체제에 맞게 개행 코드를 자

동 변환해주므로 큰 문제는 없다. 자바스크립트에서 라인 피드와 캐리지 리턴은 모두 새줄을 위미한다. 하지만 캐리지 리턴(\r)으로 개행하는 경우는 거의 없고 일반적으로 라인 피드(\n)을 사용해 개행한다.

LF와 CR의 차이

예를 들어, 줄바꿈과 들여쓰기가 적용된 문자열은 아래와 같이 이스케이프 시퀀스를 사용하여 작성한다.

JAVASCRIPT

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

console.log(template);
/*
<ul>
  <li><a href="#">Home</a></li>
</ul>
*/
```

일반 문자열과 달리 템플릿 리터럴 내에서는 이스케이프 시퀀스 사용 없이도 줄바꿈이 허용되며 모든 공백(white space)도 있는 그대로 적용된다.

JAVASCRIPT

```
var template = `<ul>
  <li><a href="#">Home</a></li>
</ul>`;

console.log(template);
/*
<ul>
  <li><a href="#">Home</a></li>
</ul>
*/
```

3.2. 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.

JAVASCRIPT

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

// ES5: 문자열 연결
console.log('My name is ' + first + ' ' + last + '.');
// My name is Ung-mo Lee.
```

템플릿 리터럴 내에서는 표현식 삽입(expression interpolation)을 통해 간단히 문자열을 삽입할 수 있다. 이를 통해 문자열 연산자보다 가독성 좋고 간편하게 문자열을 조합할 수 있다.

JAVASCRIPT

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

// ES6: 표현식 삽입
console.log(`My name is ${first} ${last}.`);
// My name is Ung-mo Lee.
```

표현식 삽입은 `${ }` 으로 표현식을 감싼다. 이때 표현식의 평가 결과가 문자열이 아니더라도 문자열로 강제 타입 변환되어 삽입된다.

JAVASCRIPT

```
console.log(`1 + 2 = ${1 + 2}`); // 1 + 2 = 3
```

표현식 삽입은 반드시 템플릿 리터럴 내에서 사용해야 한다. 템플릿 리터럴이 아닌 일반 문자열에서 표현식 삽입은 문자열 취급을 받는다.

JAVASCRIPT

```
console.log('1 + 2 = ${1 + 2}'); // 1 + 2 = ${1 + 2}
```

4. 불리언 타입

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

JAVASCRIPT

```
var foo = true;
console.log(foo); // true

foo = false;
console.log(foo); // false
```

불리언 타입의 값은 참과 거짓으로 구분되는 조건에 의해 프로그램의 흐름을 제어하는 조건문에서 자주 사용한다. 이에 대해서는 "8.2. 조건문"에서 살펴보기로 하자.

5. undefined 타입

undefined 타입의 값은 undefined가 유일하다.

var 키워드로 선언한 변수는 undefined로 초기화된다. 다시 말해, 변수 선언에 의해 확보된 메모리 공간을 처음 할당이 이루어질 때까지 빈 상태(대부분 비어있지 않고 쓰레기 값(Garbage value)이 들어 있다)로 내버려두지 않고 자바스크립트 엔진이 undefined로 초기화한다. 따라서 선언 이후 값을 할당하지 않은 변수에 접근하면 undefined가 반환된다.

JAVASCRIPT

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

이처럼 undefined는 개발자가 의도적으로 할당하기 위한 값이 아니라 자바스크립트 엔진이 변수를 초기화할 때 사용하는 값이다. 변수를 참조했을 때 undefined가 반환된다면 참조한 변수가 선언 이후 값이 할당된 적인 없는, 즉 초기화되지 않은 변수라는 것을 간파할 수 있다.

자바스크립트 엔진이 변수 초기화에 사용하는 undefined를 개발자가 의도적으로 변수에 할당한다면 undefined의 본래의 취지와 어긋날 뿐더러 혼란을 줄 수 있으므로 권장하지 않는다.

그렇다면 변수에 값이 없다는 것을 명시하고 싶은 경우 어떻게 하면 좋을까? 그런 경우는 undefined를 할당하는 것이 아니라 null을 할당한다.

> **선언(Declaration)과 정의(Definition)**
> undefined를 직역하면 "정의되지 않은"이다. 일반적으로 정의란 개념은 어떤 대상을 명확하게 규정하는 것을 의미한다. 자바스크립트의 undefined에서 말하는 정의란 변수에 값을 할당하여 변수의 실체를 명확히 하는 것을 말한다.
> 다른 프로그래밍 언어에서는 선언과 정의를 엄격하게 구분하여 사용하는 경우가 있다. 예를 들어 C에서 선언과 정의는 "실제로 메모리 주소를 할당하는가"로 구분한다. 단순히 컴파일러에게 식별자의 존재 만을 알리는 것은 선언이고 실제로 컴파일러에게 변수를 생성하도록 하여 식별자와 메모리 주소가 연결되면 정의로 구분한다. 자바스크립트의 경우, 변수를 선언하면 암묵적으로 정의가 이루어지기 때문에 선언과 정의의 구분이 모호하다.
> ECMAScript 사양에서 변수는 선언한다(Declarations and the Variable Statement)고 표현하고, 함수는 정의한다(Function Definitions)고 표현한다. 따라서 이 책에서도 ECMAScript 사양에서 사용하는 용어를 최대한 반영하여 변수는 선언, 함수는 정의로 표현하도록 하겠다.

6. null 타입

null 타입의 값은 null이 유일하다. 자바스크립트는 대소문자를 구별(case-sensitive)하므로 null은 Null, NULL등과 다르다.

프로그래밍 언어에서 null은 변수에 값이 없다는 것을 의도적으로 명시(의도적 부재 Intentional absence)할 때 사용한다.

변수에 null을 할당하는 것은 변수가 이전에 참조하던 값을 더이상 참조하지 않겠다는 의미이다. 이는 이전에 할당되어 있던 값에 대한 참조를 명시적으로 제거하는 것을 의미하며 자바스크립트 엔진은 누구도 참조하지 않는 메모리 공간에 대해 가비지 콜렉션을 수행할 것이다.

JAVASCRIPT

```
var foo = 'Lee';

// 이전에 할당되어 있던 값에 대한 참조를 제거. 변수 foo는 더이상 'Lee'를 참조하지 않는다.
// 유용해 보이지는 않는다. 변수의 스코프를 좁게 만들어 변수 자체를 재빨리 소멸시키는 편이 낳
```

다.

```
foo = null;
```

함수가 유효한 값을 반환할 수 없는 경우, 명시적으로 null을 반환하기도 한다. 예를 들어, HTML 요소를 검색해 반환하는 document.querySelector 메소드는 조건에 부합하는 HTML 요소를 검색할 수 없는 경우, 에러 대신 null을 반환한다.

HTML

```
<!DOCTYPE html>
<html>
<body>
  <script>
    var element = document.querySelector('.myClass');

    // HTML 문서에 myClass 클래스를 갖는 요소가 없다면 null을 반환한다.
    console.log(element); // null
  </script>
</body>
</html>
```

7. symbol 타입

심볼(symbol)은 ES6에서 새롭게 추가된 7번째 타입으로 변경 불가능한 원시 타입의 값이다. 심볼은 주로 이름의 충돌 위험이 없는 객체의 유일한 프로퍼티 키를 만들기 위해 사용한다.

심볼 이외의 원시값은 리터럴을 통해 생성하지만 심볼은 Symbol 함수를 호출해 생성한다. 이때 생성된 심볼 값은 노출되지 않으며 다른 값과 절대 중복되지 않는 유일무이한 값이다.

JAVASCRIPT

```
// 심볼 값 생성
var key = Symbol('key');
console.log(typeof key); // symbol

// 객체 생성
var obj = {};
```

```
// 심볼 key를 이름의 충돌 위험이 없는 유일한 프로퍼티 키로 사용한다.
obj[key] = 'value';
console.log(obj[key]); // value
```

심볼에 대해서는 “33. 7번째 타입 Symbol”에서 자세히 살펴보도록 하자.

8. 객체 타입

자바스크립트의 데이터 타입은 원시 타입과 객체 타입으로 크게 분류한다고 했다. 그 이유는 무엇일까? 원시 타입과 객체 타입은 근본적으로 다르다는 의미일 것이다. 이에 대한 설명은 아직 객체에 대해 살펴보지 않았으므로 잠시 미루도록 하자. 이에 대해서는 “11. 원시 값 과 객체의 비교”에서 자세히 살펴볼 것이다.

중요한 것은 자바스크립트는 객체 기반의 언어이며 자바스크립트를 이루고 있는 거의 모든 것이 객체라는 것이다. 지금까지 살펴본 6가지의 데이터 타입 이외의 값은 모두 객체 타입이다.

9. 데이터 타입의 필요성

데이터 타입은 왜 필요한 것일까? 데이터 타입의 필요성에 대해 살펴보도록 하자.

9.1. 데이터 타입에 의한 메모리 공간의 확보와 참조

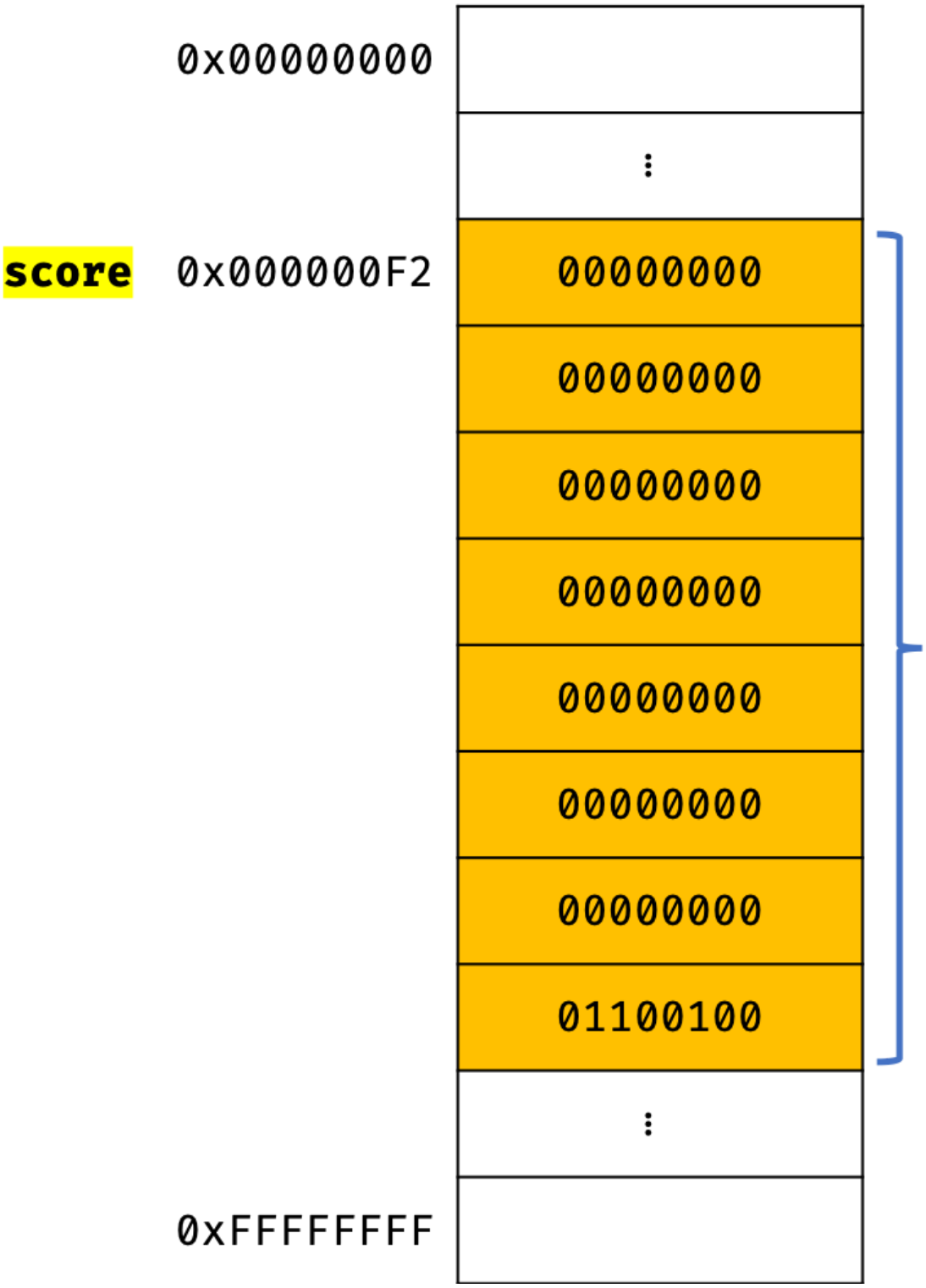
값은 메모리에 저장하고 참조할 수 있어야 한다. 메모리에 값을 저장하기 위해서는 먼저 확보해야 할 메모리 공간의 크기를 결정해야 한다. 다시 말해, 몇 byte의 메모리 공간을 사용해야 낭비와 손실없이 값을 저장할 수 있는지 알아야 한다. 예를 들어 아래와 같이 변수를 선언하고 숫자 값을 할당해 보자.

JAVASCRIPT

```
var score = 100;
```

위 코드가 실행되면 컴퓨터는 숫자 값 100을 저장하기 위해 메모리 공간을 확보한 다음, 확보된 메모리에 숫자 값 100을 2진수로 저장한다. 이러한 처리를 하려면 숫자 값을 저장할 때 확보해야 할 메모리 공간의 크기를 알아야 한다. 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.

위 예제의 경우, 자바스크립트 엔진은 100을 숫자 타입의 값으로 해석하고 숫자 타입의 값 100을 저장하기 위해 8byte의 메모리 공간을 확보한다. 그리고 10진수 100을 2진수로 저장한다.



숫자 타입 값의 할당

자바스크립트는 숫자 타입의 값을 생성할 때 배정밀도 64비트 부동소수점 포맷(double-precision 64-bit floating-point format)을 사용한다. 따라서 실제로 메모리에 저장되는 2진수 값은 위 그림과 다르다. 지금은 간단히 양의 정수로 저장된다고 생각하자.

> **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 포맷을 사용한다고 명시되어 있고, 배정밀도 64비트 부동소수점 포맷은 8byte로 숫자를 표현하므로 이 책에서는 숫자 값의 크기를 8byte로 설명하고 있다.

이번에는 값을 참조하는 경우를 생각해보자. 식별자 score를 통해 숫자 타입의 값 100이 저장되어 있는 메모리 공간의 주소를 찾아갈 수 있다. 정확히 말하면 숫자 값 100이 저장되어 있는 메모리 공간의 선두 메모리 셀의 주소를 찾아갈 수 있다.

이때 값을 참조하려면 한번에 읽어 들여야 할 메모리 공간의 크기, 즉 메모리 셀의 개수(byte 수)를 알아야 한다. 변수 score의 경우, 저장되어 있는 값이 숫자 타입이므로 8byte 단위로 읽어 들이지 않으면 값이 훼손된다. 그렇다면 컴퓨터는 한번에 읽어 들여야 할 메모리 셀의 크기를 어떻게 알 수 있는 것일까? 변수 score에는 숫자 타입의 값이 할당되어 있으므로 자바스크립트 엔진은 변수 score를 숫자 타입으로 인식한다. 숫자 타입은 8byte 단위로 저장되어 있으므로 변수 score를 참조하면 8byte 단위로 메모리 공간에 저장된 값을 읽어 들인다.

9.2. 데이터 타입에 의한 값의 해석

그런데 아직 문제가 남아 있다. 메모리에서 읽어 들인 2진수를 어떻게 해석해야 하는지에 대한 것이다.

모든 값은 데이터 타입을 갖으며 메모리에 2진수, 즉 비트(bit)의 나열로 저장된다. 메모리에 저장된 값은 데이터 타입에 따라 다르게 해석될 수 있다. 예를 들어, 메모리에 저장된 값 0100 0001를 숫자로 해석하면 65이지만 문자열로 해석하면 'A'이다.

JAVASCRIPT

```
// 2진수 01000001을 10진수 숫자로 해석하면 65이다.
// parseInt는 문자열을 숫자로 변환한다.
console.log(parseInt('01000001', 2)); // 65
// 진수 01000001을 문자(Unicode)로 해석하면 'A'이다.
```

```
// String.fromCharCode은 UTF-16 코드 유닛의 시퀀스로부터 문자열을 생성한다.
console.log(String.fromCharCode(parseInt('01000001', 2))); // A
```

위에서 살펴본 예제의 변수 score에 할당된 값은 숫자 타입의 값이다. 따라서 변수 score를 참조하면 메모리 공간의 주소에서 읽어 들인 2진수를 숫자로 해석한다.

지금까지 살펴본 데이터 타입에 대해 정리해보자. 데이터 타입은 값의 종류를 말한다. 자바스크립트의 모든 값은 데이터 타입을 갖는다. 데이터 타입이 필요한 이유는 아래와 같다.

- 값을 저장할 때 확보해야 하는 **메모리 공간의 크기**를 결정하기 위해
- 값을 참조할 때 한번에 읽어 들여야 할 **메모리 공간의 크기**를 결정하기 위해
- 메모리에서 읽어 들인 2**진수를 어떻게 해석**할 지를 결정하기 위해

10. 동적 타이핑

10.1. 동적 타입 언어와 정적 타입 언어

자바스크립트의 모든 값은 데이터 타입을 갖는다고 했다. 그렇다면 변수는 데이터 타입을 갖을까?

C나 Java와 같은 **정적 타입**(static/strong type) **언어**는 변수를 선언할 때 변수에 할당할 수 있는 값의 종류, 즉 데이터 타입을 사전에 선언해야 한다. 이를 명시적 타입 선언(explicit type declaration)이라 한다. 다음은 C에서 정수 타입의 변수를 선언하는 예이다.

```c
// 변수 c에는 1byte 정수 타입의 값(-128 ~ 127)만을 할당할 수 있다.
char c;

// 변수 num에는 4byte 정수 타입의 값(-2,124,483,648 ~ 2,124,483,647)만을 할당할 수 있다.
int num;
```

정적 타입 언어는 변수의 타입을 변경할 수 없으며, 변수에 선언한 타입에 맞는 값만을 할당할 수 있다. 정적 타입 언어는 컴파일 시점에 **타입 체크**(선언한 데이터 타입에 맞는 값을 할당했는지 검사하는 처리)를

수행한다. 만약 타입 체크를 통과하지 못했다면 에러를 발생시키고 프로그램의 실행 자체를 막는다. 이를 통해 타입의 일관성을 강제하여 보다 안정적인 코드의 구현을 통해 런타임에 발생하는 에러를 줄인다. 대표적인 정적 타입 언어는 C, C++, Java, Kotlin, Go, Haskell, Rust, Scala 등이 있다.

자바스크립트는 정적 타입 언어와는 다르게 변수를 선언할 때 타입을 선언하지 않는다. 다만 var, let, const 키워드를 사용해 변수를 선언할 뿐이다. 자바스크립트의 변수는 정적 타입 언어와 같이 미리 선언한 데이터 타입의 값만을 할당할 수 있는 것이 아니다. 어떠한 데이터 타입의 값이라도 자유롭게 할당할 수 있다.

하나의 변수를 선언하고 지금까지 살펴본 다양한 데이터 타입의 값을 할당한 다음, typeof 연산자로 변수의 데이터 타입을 조사해 보자. typeof 연산자는 자신의 뒤에 위치한 피연산자의 데이터 타입을 문자열로 반환한다.

JAVASCRIPT

```
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

foo = null;
console.log(typeof foo);  // object

foo = Symbol(); // 심볼
console.log(typeof foo);  // symbol

foo = {}; // 객체
console.log(typeof foo);  // object

foo = []; // 배열
console.log(typeof foo);  // object

foo = function () {}; // 함수
console.log(typeof foo);  // function
```

typeof 연산자로 변수를 연산해 보면 변수의 데이터 타입을 반환한다. 정확히 말하면 변수의 데이터 타입을 반환하는 것이 아니라 변수에 할당된 값의 데이터 타입을 반환한 것이다.

자바스크립트의 변수는 어떤 데이터 타입의 값이라도 자유롭게 할당할 수 있으므로 정적 타입 언어에서 말하는 데이터 타입과 개념이 다르다. 정적 타입 언어는 변수 선언 시점에 변수의 타입이 결정되고 변수의 타입을 변경할 수 없다. 자바스크립트는 값을 할당하는 시점에 변수의 타입이 동적으로 결정되고 변수의 타입을 언제든지 자유롭게 변경할 수 있다.

다시 말해 **자바스크립트 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 하며 자바스크립트를 정적 타입 언어와 구별하기 위해 **동적 타입**(dynamic/weak type) **언어**라 부른다. 대표적인 동적 타입 언어는 자바스크립트, Python, PHP, Ruby, Lisp, Perl 등이 있다.

처음의 질문으로 돌아가 보자. 변수는 타입을 갖을까? 기본적으로 변수는 타입을 갖지 않는다. 하지만 값은 타입을 갖는다. 따라서 현재 변수에 할당되어 있는 값에 의해 변수의 타입이 동적으로 결정된다고 표현하는 것이 보다 적절하다. 변수는 값에 묶여 있는 값에 대한 별명이기 때문이다.

10.2. 동적 타입 언어와 변수

동적 타입 언어는 변수에 어떤 데이터 타입의 값이라도 자유롭게 할당할 수 있다. 이러한 동적 타입 언어의 특징은 데이터 타입에 대해 무감각해질 정도로 편리하다. 하지만 언제나 그렇듯 편리함의 이면에는 위험도 도사리고 있다.

모든 소프트웨어 아키텍처에는 트레이드오프(trade-off)가 존재하며 모든 애플리케이션에 적합한 은 탄환(Silver bullet)은 없듯이, 동적 타입 언어 또한 구조적인 단점이 있다.

> **트레이드오프**(trade-off)
> 두 개의 정책 목표 가운데 하나를 달성하려고 하면 다른 목표의 달성이 늦어지거나 희생되는 모순적 관계를 의미한다. 예를 들어, 실업률을 줄이면 물가가 상승하고, 물가를 안정시키면 실업률이 높아진다.

> **은 탄환**(Silver bullet)
> 고질적인 문제를 단번에 해결할 수 있는 명쾌한 해결책

변수 값은 언제든지 변경될 수 있기 때문에 복잡한 프로그램에서는 변화하는 변수 값을 추적하기 어려울 수 있다. 뿐만 아니라, 변수의 타입이 고정되어 있지 않고 동적으로 변하는 동적 타입 언어의 변수는 값의 변경에 의해 타입도 언제든지 변경될 수 있다. 따라서 **동적 타입 언어의 변수는 값을 확인하기 전에는 타입을 확신할 수 없다**.

더욱이 자바스크립트는 개발자의 의도와는 상관없이 자바스크립트 엔진에 의해 암묵적으로 타입이 자동 변환되기도 한다. 즉, 숫자 타입의 변수일 것이라고 예측했지만 사실은 문자열 타입의 변수일 수도 있다는 말이다. 잘못된 예측에 의해 작성된 프로그램은 당연히 오류를 뿜어낼 것이다. 결국 동적 타입 언어는 유연성(flexibility)은 높지만, 신뢰성(reliability)은 떨어진다.

이러한 이유로 안정적인 프로그램을 만들기 위해 변수를 사용하기 이전에 데이터 타입을 체크해야 하는 경우가 있는데 이는 매우 번거로울 뿐만 아니라 코드량도 증가한다. 코드량이 증가하면 버그가 발생할 확률도 높아지며 테스트 분량도 증가한다. 따라서 변수를 사용할 때 주의할 사항은 아래와 같다.

- 변수는 꼭 필요한 경우에 한해 제한적으로 사용한다. 변수값은 재할당에 의해 언제든지 변경될 수 있다. 이로 인해 동적 타입 언어인 자바스크립트는 타입을 잘못 예측해 오류가 발생할 가능성이 크다. 변수의 개수가 많으면 많을수록 오류가 발생할 확률도 높아진다. 따라서 변수의 무분별한 남발은 금물이며 필요 최소한으로 유지하도록 주의해야 한다.
- 변수의 유효 범위(스코프)는 최대한 좁게 만들어 변수의 부작용을 억제해야 한다. 변수의 유효 범위가 넓으면 넓을수록 변수로 인해 오류가 발생할 확률은 높아진다. 변수의 유효 범위에 대해서는 "13. 스코프"에서 자세히 살펴보도록 하자.
- 전역 변수는 최대한 사용하지 않도록 한다. 어디서든지 참조/변경 가능한 전역 변수는 의도치 않게 값이 변경될 가능성이 높고 다른 코드에 영향을 줄 가능성도 높다. 따라서 전역 변수는 프로그램의 복잡성을 증가시키고 처리의 흐름을 추적하기 어렵게 만들고, 오류가 발생했을 경우, 오류의 원인을 특정하기 어렵게 만든다. 전역 변수의 문제점과 전역 변수의 사용을 억제하는 방법에 대해서는 "14. 전역 변수의 문제점"에서 자세히 살펴보도록 하자.
- 변수보다는 상수를 사용해 값의 변경을 억제한다. 상수를 사용하는 방법에 대해서는 "15.3. const 키워드"에서 살펴보도록 하자.
- 변수 이름은 변수의 목적이나 의미를 파악할 수 있도록 네이밍한다. 변수 이름 뿐만이 아니라 모든 식별자(변수, 함수, 클래스 이름 등)는 존재 이유를 파악할 수 있는 적절한 이름을 네이밍해야 한다. 특히 식별자의 유효 범위가 넓을수록 보다 명확한 이름을 명명하도록 노력하도록 하자. 개발자의 의도를 나타내는 명확한 네이밍은 코드를 이해하기 쉽게 만들며 이는 협업과 생산성 향상에 도움을 준다. 변수 이름은 첫아이 이름을 짓듯이 심사숙고해서 지어야 한다.

코드는 오해하지 않도록 작성해야 한다. 오해는 커뮤니케이션을 어렵게 하는 대표적인 원인으로 생산성을 떨어뜨리는 것은 물론 팀의 사기까지 저하시킨다. 코드는 동작하는 것만이 존재 목적은 아니다. 코드는 개발자를 위한 문서이기도 하다. 따라서 사람이 이해할 수 있는 코드, 즉 **가독성이 좋은 코드가 좋은 코드다**.

> 컴퓨터가 이해하는 코드는 어떤 바보도 쓸 수 있다. 하지만 훌륭한 프로그래머는 사람이 이해할 수 있는 코드를 쓴다.
> Any fool can write code that a computer can understand. Good programmers write code that humans can understand.
> - 마틴 파울러(Martin Fowler), “리팩토링”의 저자