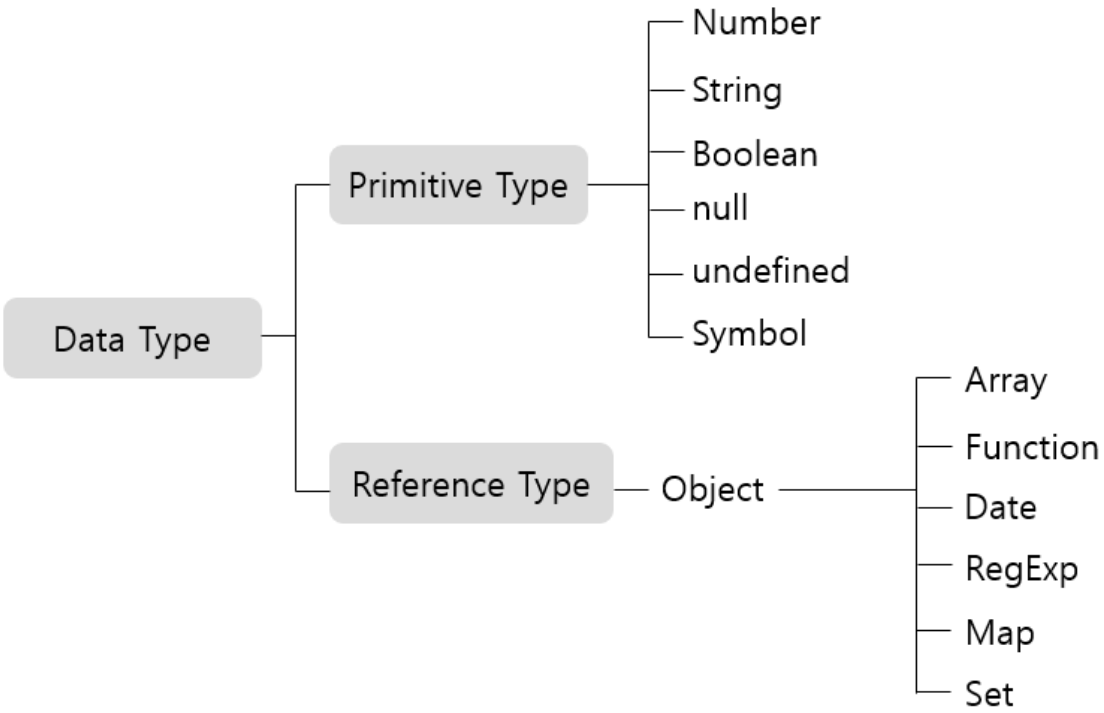
6장 데이터 타입



해당 그림은 JS에서 데이터 타입을 한눈에 설명함

데이터 타입

구분	타입	설명
원시	숫자	숫자, 정수와 실수 구분 없이 하나의 숫자 타입만 존재
원시	문자열	문자열
원시	불리언	논리적 참과 거짓
원시	undefined	var 키워드로 선언된 변수에 암묵적으로 할당되는 값
원시	null	값이 없다는 것을 의도적으로 명시할 때 사용하는 값
원시	심벌	ES6 에서 추가된 타입 7번째 타입
객체	객체	객체, 함수, 배열 등

숫자타입

- ECMAScript 기준으로 숫자 타입의 값은 배정밀도 64비트 부동소수점 형식을 따름
  - 이 말은 즉 정수,실수 그딴거 필요없고 다 실수 취급한다는 뜻

    → integer 값을 표현하는 변수 타입이 따로 없음(ex. C 의 int)

```
var a = 1; // -> 넘버
var b = -1; // -> 넘버
var c = 509.00000000001; // -> 넘버
```

그렇다면 위에 설명과 같이 다 실수로 측정한다면 2진수, 8진수, 16진수를 출력하면 어떤식으로 될까?

```
var binary = 0b01000001;
var octal = 0o101;
var hex = 0x41;

console.log(binary,octal,hex);

if(binary === hex) console.log(true);
if(binary === octal) console.log(true);
```

💡 다 실수라면 아래의 비교문의 결과는 어떻게 나올까?

```
console.log(1 === 1.0);
console.log(4 / 2);
console.log(3 / 2)
```

- Infinity : 양의 무한대
- -Infinity : 음의 무한대
- NaN(Not A Number) : 산술 연산 불가

💡 위에 세가지를 console을 이용하여 도출해보세요.

💡 만약 NaN이 아닌 nan, NAN 같이 변수에 대입하면 어떤 식으로 나올까요?

문자열 타입

- string은 0개 이상의 16비트 유니코드 문자의 집합으로 대부분 표현 가능
- ‘’(single quote), “”(big quotes), ``(백틱)
  - JS 표준은 ‘’ (single quote)
- JS에서 문자열은 원시 타입이며 변경 불가능 값(immutable value)으로 저장됨
  - 문자열이 생성되면 해당 문자열을 변경할 수 없다는 것을 의미

💡 “” 안의 ‘’(single quote)은 뭘로 인식되고 ‘’(single quote) 안의 “”은 뭘로 인식될까

```
var a = "이거랑'이거랑'";
var b = '꽃게랑"꽃게랑"';
```

- 일반 문자열 내에서는 개행이 허용되지 않음
  - 개행을 위해서는 이스케이프 시퀀스(escape sequence) 를 사용해야함

```
var str = 'Hello From The
Other Siiiiiiiiiiide'; // 바아아아로 Syntax Error : Invalid or unexpected token 발생
```

이스케이프 시퀀스	의미
\0	Null
\f	백스페이스
\b	폼 피드, 프린트 기준으로 다음페이지 시작 지점으로 이동
\n	개행, 다음 행으로 이동
\r	개행, 커서를 처음으로 이동
\t	탭 (수평)
\v	탭 (수직)
\uXXXX	유니코드
\’	작은따옴표
\”	큰따옴표
\\	백슬래시

- 파충류가 겪었던 **최악의 ESLint Error**
  - 개행 방식 CRLF, LF
    - 윈도우 기반의 OS는 CR+LF(ASCII 기준 13 + 10 번, \r\n)을 개행으로 사용함
    - 유닉스 기반의 OS는 LF(ASCII 기준 10번)을 개행으로 사용함
    - MacOS는 9 버전까지는 CR을 사용하고 10버전 부터는 LF를 사용함
  - **Mac에서 사용하다가 윈도우에서 깃을 불러오고 빌드를 진행하면…?**

    → 거지같은 ESLint는 나에게 모든 파일이 LF로 작성되었으니 다 CRLF로 바꾸라고 한다.

템플릿 리터럴

- ES6부터 추가된 문자열 표기법이며 멀티라인 문자열, 표현식 삽입, 태그드 탬플릿 등 편리한 문자열 처리 기능을 제공함
  - 런타임시 일반 문자열로 변환되어 처리되는 형식을 가짐

멀티라인 문자열

- 템플릿 안에서는 공백도 다 출력되며 굳이 escape sequence를 사용하지 않아도 줄바꿈이 됨

```
var str = `Hello From the
Other Siiiiiiiiiiiiiiiiiide~
`;
console.log(str); // 콘솔에 정상적으로 개행과 공백이 함께 출력됨

var tem = `<ul>\n\t<li><span>Fxxx</span></li>\n</ul>`;
console.log(tem); // 콘솔에 마치 html로 작성한것과 같은 모양으로 코드가 나열됨
```

표현식 삽입

- 문자열은 문자열 연산자 + 를 통해서 연결이 가능함
    - 다만 피연산자 중 하나 이상이 문자열일 경우에만 작동하며 이 외의 경우에는 덧셈 연산자로 동작함

```
var a = "turt1e is";
var b = "turtle";

console.log(a+b); // turt1e is turtle로 출력된다
```

- 이때 표현식을 삽입 하려면 ${ }을 이용함

```
var a = "turt1e is";
var b = "turtle";

console.log(`Did you know that ${a} ${b}`); // Did you know that turt1e is turtle로 출력
```

그렇다면 아래의 코드는 어떤식으로 다를까?

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

불리언 타입

- 말이 필요한가… **true false**

undefined 타입

- JS 엔진에서 개발자가 var 키워드로 선언한 변수는 자동으로 undefined로 초기화함
    - 선언에 의해 확보된 메모리 공간을 처음 할당이 이뤄질 때까지 빈 상태로 내버려두지 않고 초기화함

```
var code1;
console.log(code1); // undefined
```

> ***그럼 앤 진짜 무슨 용도인가요?***
> 개발자가 선언해 놓고 사용하지 않은 변수에는 고의적으로 넣지 않는 이상 무조건 undefined으로 초기화되니 해당 타입이 검출되면 사용한 적 없는 변수라는 뜻으로 해석 가능함

- 만약 억지로 undefined을 변수에 할당한다면 원래 취지와 어긋나며 코드에 혼란을 주니 매우 권장하지 않음
    - 값이 아직 없다라는 걸 표시하고 싶으면 null을 사용하면 됨
        - 근데 이것도 어떨 때는 그렇게 좋은 방법은 아님 → 추후 null 타입 설명 때 서술

*** 해당 문구는 한번 쯤은 생각할 수 있는 부분입니다. 반드시 읽고 넘어 가주세요.**
선언 declaration 과 정의 definition
우선 undefined은 '정의되지 않은' 이라는 뜻의 영어이다. 여기서 JS에서 **정의**는 변수에 값을 할당하고 해당 변수의 실체를 정확히 하는 행위를 뜻한다. 다른 프로그래밍 언어에서는 선언과 정의를 구분하는 경향이 있다.
대표적으로 C 같은 경우에는 메모리 할당을 기준으로 구분하는데, **선언**은 컴파일러에게 식별자의 존재를 알리는 것을 뜻하고 **정의**는 컴파일러가 해당 식별자에 맞는 변수를 생성하고 식별자와 메모리를 연결했을 때를 뜻한다.
그에 반해 JS는? 선언과 정의의 경계가 매우 모호한 것이 선언과 동시에 암묵적으로 undefined로 정의 되어버린다.
그래서 표준인 ECMAScript에서는 '변수를 선언한다' 라 하며 '함수를 정의한다'라고 표현한다.

null 타입

- null 은 null 이 유일함
  - JS는 대소문자를 구별하니 이상한 바리에이션 Null, NULL … 등등 유사한 String 은 null로 인식하지 않음
    - ~~이상한 곳에서 깐깐하다~~
- 프로그래밍 언어에서는 null을 변수에 값이 없다라는 것을 의도적으로 명시(의도적 부재 intentional absence)할 때 사용함
  - JS 엔진의 GC를 어떤것도 참조하지 않는 메모리 공간에 대해서 수행하는 경향이 있음
  - 변수에 null이 선언되어 있으면 이는 이전에 있던 참조값을 참조하지 않겠다는 뜻으로 받아들임

의도적 부재를 왜 사용할까?

과연 아래의 사용법이 옳은 선택일까? 다른 방법으로 변수를 소멸시키는게 좋지 않을까?

```
var night = 'Turtle';
// 밑의 선언으로 인해 night는 더이상 터틀이라는 값을 참조하지 않으며 언젠가 gc에 없어져버린다.
night = null;
```

- 또 null은 함수가 유효한 값을 반환할 수 없는 경우 명시적으로 반환하는 경우도 존재함
  - document.querySelector 메서드는 조건에 부합하는 HTML 요소를 검색할 수 없는 경우 Error대신 null을 반환함
    - 아니 없으면 undefined가 아닌가? 생각되면 해당 논쟁에 대한 키워드로 검색해보면 실제로 그렇게 싸우고 있다.
- undefined 타입 설명 중 … 근데 이것도 어떨 때는 그렇게 좋은 방법은 아님 → 추후 null 타입 설명 때 서술
  - 예를 들어 변수의 타입 검사를 하기 위해 조건에 `if(a === typeof null) console.log("hello")` 와 같은 코드를 적용한다면 undefined가 뜰 것이다. 왜 그럴까?

심벌 타입

- 해당 타입은 ES6에서 추가된 7번째 타입이며 변경 불가능한 원시 타입의 값
  - 심지어 심벌은 다른 값과 중복되지 않은 유니크 값임

    → 중복될 일 없는 이름으로 유일한 프로퍼티 키를 만들기 위해 사용함
- 생성은 Symbol 함수를 호출해서 생성되며 이 때 생성된 값은 외부에 노출되지 않으며 다른 값과 중복되지 않음

```
var key = Symbol('key');
console.log(typeof key); //Symbol 로 출력

var object1 = {};

object1[key] = 'value';
console.log(obj[key]); // value 라는 값을 출력
```

객체 타입

- JS는 데이터 타입을 크게 원시와 객체로 분류하며 JS는 객체 기반 언어라는 사실을 안다면 JS를 이루고 있는 거의 모든 것이 객체로 이루어져 있음
  - 위 6개의 데이터 타입(리터럴 제외)이외에는 모두 객체 타입임

필요성

메모리 공간의 확보와 참조

- 모든 값은 메모리에 저장하고 참조할 수 있어야 함
  - 메모리에 값을 저장하려면 첫 번째로 해당 변수 값에 맞는 낭비와 손실 없는 적절한 메모리 크기를 결정해야함
  - JS엔진은 데이터 타입, 값의 종류에 따라 정해진 크기의 메모리 공간을 확보함

```
var score = 100;
```

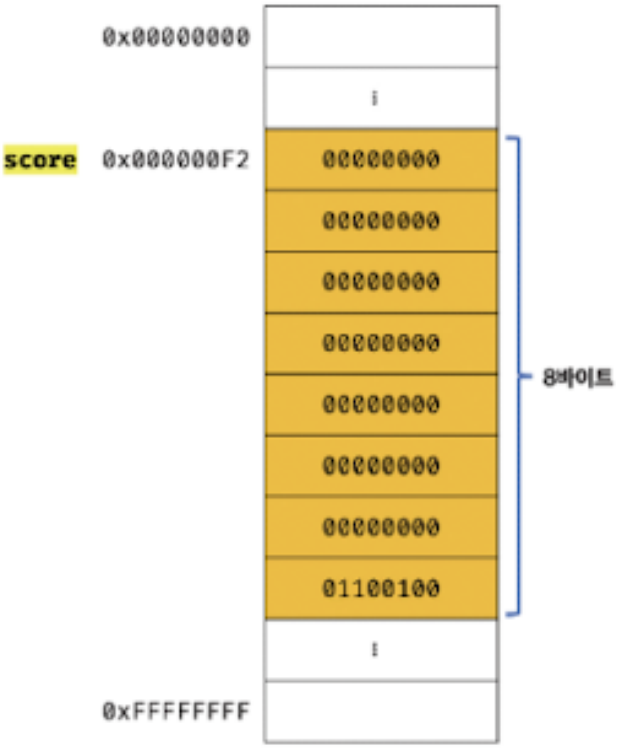


그림 6-1 숫자 타입 값의 할당

ECMAScript 사양은 문자열과 숫자 타입 외에는 명시적으로 규정하고 있지 않은데 그렇다면 해당 데이터 타입들 외에는 어떤 식으로 계산되고 있는가?

심벌 테이블 이라는 뜻을 알아보시오

- 그래서 이유가 뭐죠?
  - 값을 저장할 때 확보해야 하는 메모리 공간의 크기를 결정하기 위함
  - 값을 참조할 때 한 번에 읽어 들여야 할 메모리 공간의 크기를 결정하기 위해
  - 메모리에서 읽어 들인 2진수를 어떻게 해석할지 결정하기 위해

동적 타이핑

동적 타입 언어 vs 정적 타입 언어

- C나 Java와 같은 정적 타입언어는 변수를 선언할 때 변수에 할당할 수 있는 데이터 타입을 미리 선언해야함
  - 이를 명시적 타입 선언 이라 지칭함
  - 이를 통해 컴파일 시점 때 타입 체크를 수행하며 해당 작업을 통과하지 못하면 에러를 발생시킴
- JS의 경우에는 명시적 선언을 하지 않으며 var, let, const를 이용한 변수 선언을 함
  - 이 경우 정수를 선언했다가 해당 변수에 문자열을 선언하면 타입이 변함
  - JS의 경우에는 변수는 선언이 아닌 할당에 의해 타입이 결정(타입 추론)됨, 재할당에 의해 변수의 타입은 언제든지 동적으로 변할 수 있음

대표적인 동적/정적 언어를 조사해보시오

동적 타입 언어와 변수에 부작용

이 때 되돌아보는 명언

모든 SW 아키텍처에는 Trade-off가 존재하며 모든 Application 적합한 Silver Bullet은 없다.

- Trade-off

  두 개의 정책이나 목표 중 하나를 달성하려고 하면 다른 목표의 달성이 늦어지거나 희생되는 모순적 관계를 의미

- Silver Bullet

  고질적인 문제를 단번에 해결할 수 있는 명쾌한 해결책

- 부작용이 뭐가 있을까
  1. 변수 값은 언제든 변경될 수 있기에 복잡한 프로그램에서는 변화하는 값을 추적하기 어려울 수 있음
  2. 타입 마저 고정되어 있지 않고 동적으로 변하기에 이 또한 추적에 어려움을 줄 수 있음
  3. JS는 개발자의 의도와 다르게 엔진에 의해 암묵적으로 타입이 자동 변환되기도 함. 예를 들어 a라는 변수에 1을 넣었다고 생각해보면 해당 변수가 숫자형이 될 수도 있고 문자형이 될 수도 있음
  - 유연성은 높지만 신뢰성이 떨어짐
- 변수를 선언할 때

1. 꼭 필요할 경우에 한해 제한적으로 사용함
    - 변수의 개수가 많을 수록 오류가 발생할 확률도 많아짐
2. 변수의 유효 범위(스코프)는 최대한 좁게 만들어 변수의 부작용을 억제해야함
    - 변수의 유효범위가 넓으면 넓을수록 변수로 인한 오류가 발생할 확률이 높아짐
3. 전역 변수사용을 삼가함
    - 어디에서든 참조/변경 가능한 전역 변수는 의도치 않게 값이 변경될 가능성이 높고 다른 코드에 영향을 줄 가능성도 높음
    - 심지어 오류가 발생할 경우 오류의 원인을 특정하기 어렵게 만듬
4. 변수보다는 상수를 사용해 값의 변경을 억제함
    - const 쓰시오
5. 변수 이름은 변수의 목적이나 의미를 파악할 수 있도록 네이밍하는게 좋음
    - 암만 잘 만들면 뭐하나 본인만 알면 다른사람과 협업하기 힘듦
    - https://velog.io/@humonnom/네이밍-컨벤션과-변수이름-짓기 참고하자

이 때 또 보는 명언

- 컴퓨터가 이해하는 코드는 어떤 바보도 쓸 수 있다. 하지만 훌륭한 프로그래머는 사람이 이해할 수 있는 코드를 쓴다.
    - 마틴 파울러 (리팩토링 저자)