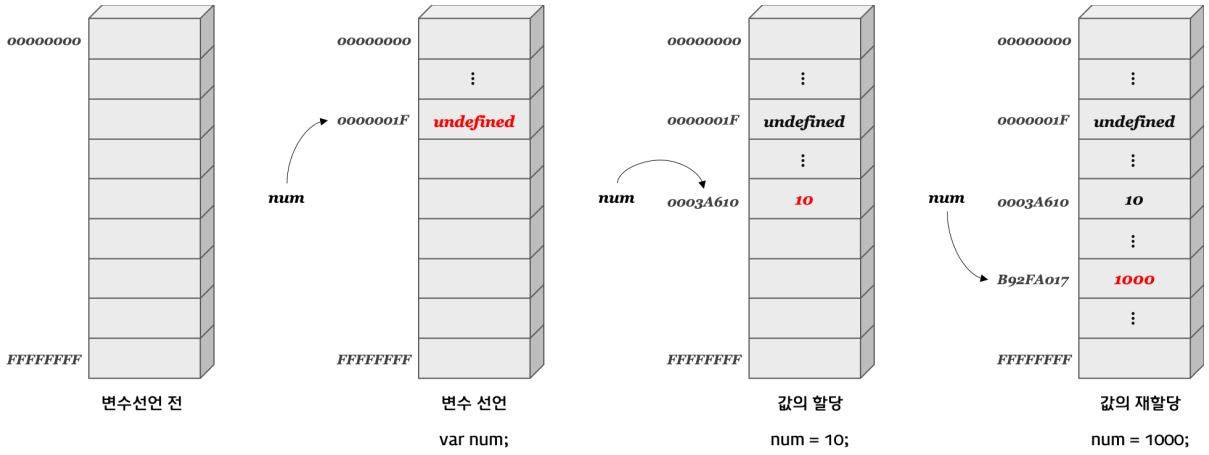
데이터 타입과 변수

프로그래밍은 변수를 통해 값을 저장하고 참조하며 연산자로 값을 연산, 평가하고 조건문과 반복문에 의한 흐름제어로 데이터의 흐름을 제어하고 함수로 재사용이 가능한 구문의 집합을 만들며 객체, 배열 등으로 자료를 구조화하는 것이다.

변수는 값의 위치(주소)를 기억하는 저장소이다. 값의 위치란 값이 위치하고 있는 메모리 상의 주소(address)를 의미한다. 즉, 변수란 값이 위치하고 있는 메모리 주소(Memory address)에 접근하기 위해 사람이 이해할 수 있는 언어로 명명한 식별자(identifier)이다.



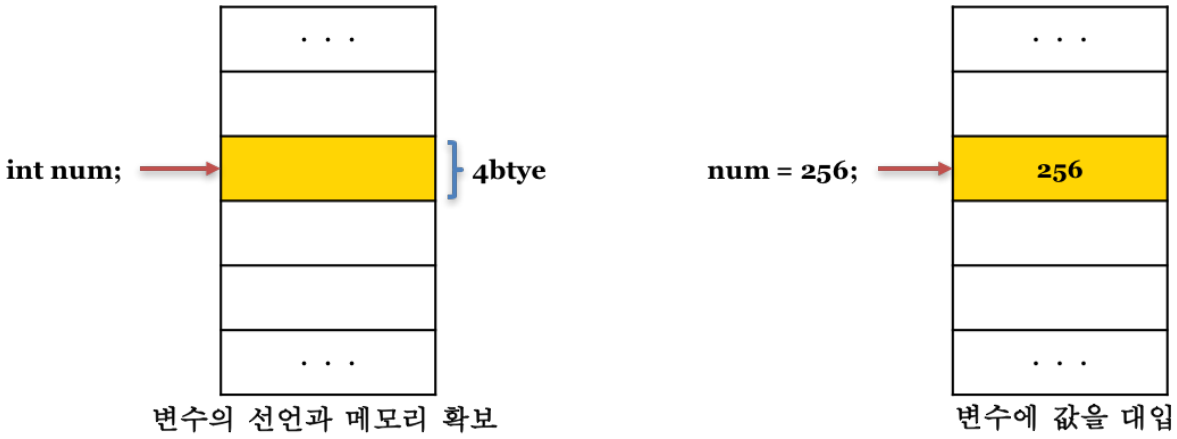
메모리에 값을 저장하기 위해서는 먼저 메모리 공간을 확보해야 할 메모리의 크기(byte)를 알아야한다. 이는 값의 종류에 따라 확보해야 할 메모리의 크기가 다르기 때문이다. 이때 값의 종류, 즉 데이터의 종류를 데이터 타입(Data Type)이라 한다.

예를 들어 1byte(8bit)로 표현할 수 있는 경우의 수, 즉 값의 총 개수는 256개(28)로 아스키 코드(ASCII)를 표현할 수 있으며, 4byte(32bit)로 표현할 수 있는 값의 총수는 4,294,967,296개(232)로 -2,147,483,648 ~ 2,147,483,647의 정수를 표현할 수 있다.

C나 Java같은 C-family 언어는 정적 타입(Static/Strong Type) 언어로 변수 선언 시 변수에 저장할 값의 종류를 사전에 타입 지정(Type annotation)하여야 한다. 다음은 C에서 정수형 변수를 선언하는 예이다.

```
// 1byte 정수형: -128 ~ 127
char c;

// 4byte 정수형: -2,124,483,648 ~ 2,124,483,647
int num;
```



C 언어의 경우, 4byte 정수형인 int형 변수 선언을 만나면 시스템은 이후 할당될 값과는 상관없이 4byte의 메모리 영역을 확보한다. 이후 int형 변수에 할당할 때에는 int형 값을 할당해야한다. 다음은 C에서 정수형 변수에 문자열을 잘못 할당한 예이다.

```
int main(void) {
  int num = 46;
  char * str = "String";

  num = "String"; // warning: incompatible pointer to integer
```

```
  return 0;
}
```

자바스크립트는 동적 타입(Dynamic/Weak Type) 언어이다. 변수의 타입 지정(Type annotation)없이 값이 할당되는 과정에서 자동으로 변수의 타입이 결정(타입 추론, Type Inference)된다. 즉, 변수는 고정된 타입이 없다. 따라서 같은 변수에 여러 타입의 값을 자유롭게 할당할 수 있다.

```
var str  = 'Hello';
var num  = 1;
var bool = true;

var foo = 'string';
console.log(typeof foo); // string
foo = 1;
console.log(typeof foo); // number
```

자바스크립트에는 어떠한 데이터 타입이 있는지 그리고 변수는 어떻게 사용하는지 알아보도록 하자.

데이터 타입

데이터 타입(Data Type)은 프로그래밍 언어에서 사용할 수 있는 데이터(숫자, 문자열, 불리언 등)의 종류를 말한다.

코드에서 사용되는 모든 데이터는 메모리에 저장하고 참조할 수 있어야 한다. 데이터 타입은 데이터를 메모리에 저장할 때 확보해야 하는 메모리 공간의 크기와 할당할 수 있는 유효한 값에 대한 정보, 그리고 메모리에 저장되어 있는 **2진수 데이터를 어떻게 해석**할 지에 대한 정보를 컴퓨터와 개발자에게 제공한다.

데이터 타입은 한정된 메모리 공간을 효율적으로 사용하기 위해서, 그리고 2진수 데이터로 메모리에 저장된 데이터를 다양한 형태로 사용하기 위해 존재한다.

자바스크립트의 모든 값은 데이터 타입을 갖는다. ECMAScript 표준(ECMAScript 2015 (6th Edition), 이하 ES6)은 7개의 데이터 타입을 제공한다

원시 타입 (primitive data type)

- `number`
- `string`
- `boolean`
- `null`
- `undefined`
- `symbol` (New in ECMAScript 6)

객체 타입 (Object data type)

- `object`

예를 들어 숫자(number) 타입 1과 문자열(string) 타입 '1'은 비슷하게 보이지만 다른 타입의 값이다. 숫자 타입의 값은 주로 산술 연산을 위해 만들지만 문자열 타입의 값은 주로 텍스트로 출력하기 위해 만든다. 이처럼 개발자는 명확한 의도를 가지고 타입을 구별하여 값을 만들 것이고 자바스크립트 엔진은 타입을 구별하여 값을 취급할 것이다.

자바스크립트에서 제공하는 7개의 데이터 타입은 크게 원시 타입(primitive data type)과 객체 타입(object/reference type)으로 구분할 수 있다.

원시 타입 (Primitive Data Type)

원시 타입의 값은 변경 불가능한 값(immutable value)이며 **pass-by-value(값에 의한 전달)** 이다.

number

C나 Java의 경우, 정수와 실수를 구분하여 int, long, float, double 등과 같은 다양한 숫자 타입이 존재한다. 하지만 자바스크립트는 독특하게 하나의 숫자 타입만 존재한다.

ECMAScript 표준에 따르면, 숫자 타입의 값은 배정밀도 64비트 부동소수점 형(double-precision 64-bit floating-point format : -(253 -1) 와 253 -1 사이의 숫자값)을 따른다. 즉, 모든 수를 실수를 처리하며 정수만을 표현하기 위한 특별한 데이터 타입(integer type)은 없다.

```
var integer = 10;         // 정수
var double = 10.12;       // 실수
var negative = -20;       // 음의 정수
var binary = 0b01000001; // 2진수
var octal = 0o101;        // 8진수
var hex = 0x41;           // 16진수
```

2진수, 8진수, 16진수 리터럴은 메모리에 동일한 배정밀도 64비트 부동소수점 형식의 2진수로 저장된다. 자바스크립트는 2진수, 8진수, 16진수 데이터 타입을 제공하지 않기 때문에 이들 값을 참조하면 모두 10진수로 해석된다.

```
console.log(binary); // 65
console.log(octal);  // 65
console.log(hex);    // 65

// 표기법만 다를뿐 같은 값이다.
console.log(binary === octal); // true
console.log(octal === hex);    // true
```

자바스크립트의 숫자 타입은 정수만을 위한 타입이 없고 모든 수를 실수를 처리한다. 정수로 표시된다해도 사실은 실수다. 따라서 정수로 표시되는 수 끼리 나누더라도 실수가 나올 수 있다.

```
console.log(1 === 1.0); // true

var result = 4 / 2;
console.log(result); // 2
```

```
result = 3 /2;
console.log(result); // 1.5
```

추가적으로 3가지 특별한 값들도 표현할 수 있다.

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

```
var pInf = 10 / 0;  // 양의 무한대
console.log(pInf);  // Infinity

var nInf = 10 / -0; // 음의 무한대
console.log(nInf);  // -Infinity

var nan = 1 * 'string'; // 산술 연산 불가
console.log(nan);       // NaN
```

수학적 의미의 실수(實數, real number)는 허수(虛數, imaginary number)가 아닌 유리수와 무리수를 통틀은 말이지만 프로그래밍 언어에서 실수는 일반적으로 소수를 가리킨다.

string

문자열(String) 타입은 텍스트 데이터를 나타내는데 사용한다. 문자열은 0개 이상의 16bit 유니코드 문자(UTF-16) 들의 집합으로 대부분의 전세계의 문자를 표현할 수 있다. 문자열은 작은 따옴표(‘’) 또는 큰 따옴표(“”) 안에 텍스트를 넣어 생성한다. 가장 일반적인 표기법은 작은 따옴표를 사용하는 것이다.

```
var str = "string"; // 큰 따옴표
str = 'string';     // 작은 따옴표
str = `string`;     // 백틱(ES6 템플릿 리터럴)
```

```
str = "큰 따옴표로 감싼 문자열 내의 '작은 따옴표'는 문자열이다.";
str = '작은 따옴표로 감싼 문자열 내의 "큰 따옴표"는 문자열이다.';
```

C와 같은 언어와는 다르게, 자바스크립트의 문자열은 원시 타입이며 변경 불가능(immutable)하다. 이것은 한 번 문자열이 생성되면, 그 문자열을 변경할 수 없다는 것을 의미한다. 아래의 코드를 살펴보자.

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

첫번째 구문이 실행되면 메모리에 문자열 'Hello'가 생성되고 식별자 str은 메모리에 생성된 문자열 'Hello'의 메모리 주소를 가리킨다. 그리고 두번째 구문이 실행되면 이전에 생성된 문자열 'Hello'을 수정하는 것이 아니라 새로운 문자열 'world'를 메모리에 생성하고 식별자 str은 이것을 가리킨다. 이때 문자열 'Hello'와 'world'는 모두 메모리에 존재하고 있다. 변수 str은 문자열 'Hello'를 가리키고 있다가 문자열 'world'를 가리키도록 변경되었을 뿐이다.

```
var str = 'string';
// 문자열은 유사배열이다.
for (var i = 0; i < str.length; i++) {
  console.log(str[i]);
}

// 문자열을 변경할 수 없다.
str[0] = 'S';
console.log(str); // string
```

문자열은 배열처럼 인덱스를 통해 접근할 수 있다. 이와 같은 특성을 갖는 데이터를 **유사 배열**이라 한다.

`str[0] = 'S'`처럼 이미 생성된 문자열의 일부 문자를 변경해도 반영되지 않는다(이때 에러가 발생하지 않는다). 한번 생성된 문자열은 read only로서 변경할 수 없다. 이것을 변경 불

가능(immutable)이라 한다.

그러나 새로운 문자열을 재할당하는 것은 물론 가능하다. 이는 기존 문자열을 변경하는 것이 아니라 새로운 문자열을 새롭게 할당하는 것이기 때문이다.

```
var str = 'string';
console.log(str); // string

str = 'String';
console.log(str); // String

str += ' test';
console.log(str); // String test

str = str.substring(0, 3);
console.log(str); // Str

str = str.toUpperCase();
console.log(str); // STR
```

boolean

불리언(boolean) 타입의 값은 논리적 참, 거짓을 나타내는 `true`와 `false` 뿐이다.

```
var foo = true;
var bar = false;

// typeof 연산자는 타입을 나타내는 문자열을 반환한다.
console.log(typeof foo); // boolean
console.log(typeof bar); // boolean
```

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

비어있는 문자열과 `null`, `undefined`, 숫자 0은 `false`로 간주된다.

undefined

undefined 타입의 값은 `undefined`가 유일하다. 선언 이후 값을 할당하지 않은 변수는 `undefined` 값을 가진다. 즉, 선언은 되었지만 값을 할당하지 않은 변수에 접근하거나 존재하지 않는 객체 프로퍼티에 접근할 경우 undefined가 반환된다.

이는 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 초기화하기 때문이다.

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

이처럼 undefined는 개발자가 의도적으로 할당한 값이 아니라 자바스크립트 엔진에 의해 초기화된 값이다. 변수를 참조했을 때 undefined가 반환된다면 참조한 변수가 선언 이후 값이 할당된 적인 없는 변수라는 것을 개발자는 간파할 수 있다. 그렇다면 개발자가 의도적으로 undefined를 할당해야하는 경우가 있을까? 자바스크립트 엔진이 변수 초기화에 사용하는 이 값을 만약 개발자가 마음대로 할당한다면 undefined의 본래의 취지와 어긋날 뿐더러 혼란을 줄 수 있으므로 권장하지 않는다. 그럼 변수의 값이 없다는 것을 명시하고 싶은 경우 어떻게 하면 좋을까? 그런 경우는 undefined를 할당하는 것이 아니라 null을 할당한다.

null

null 타입의 값은 `null`이 유일하다. 자바스크립트는 대소문자를 구별(case-sensitive)하므로 `null`은 Null, NULL등과 다르다.

프로그래밍 언어에서 `null`은 의도적으로 변수에 값이 없다는 것을 명시할 때 사용한다. 이는 변수가 기억하는 메모리 어드레스의 참조 정보를 제거하는 것을 의미하며 자바스크립트 엔진은 누구도 참조하지 않는 메모리 영역에 대해 가비지 콜렉션을 수행할 것이다.

```
var foo = 'Lee';
foo = null;  // 참조 정보가 제거됨
```

또는 함수가 호출되었으나 유효한 값을 반환할 수 없는 경우, 명시적으로 null을 반환하기도 한다. 예를 들어, 조건에 부합하는 HTML 요소를 검색해 반환하는 Document.querySelector()는 조건에 부합하는 HTML 요소를 검색할 수 없는 경우, null을 반환한다.

```
var element = document.querySelector('.myElem');
// HTML 문서에 myElem 클래스를 갖는 요소가 없다면 null을 반환한다.
console.log(element); // null
```

타입을 나타내는 문자열을 반환하는 `typeof` 연산자로 null 값을 연산해 보면 null이 아닌 object가 나온다. 이는 자바스크립트의 설계상의 오류이다.

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

따라서 null 타입을 확인할 때 typeof 연산자를 사용하면 안되고 일치 연산자(===)를 사용하여야 한다.

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

symbol

심볼(symbol)은 ES6에서 새롭게 추가된 7번째 타입으로 변경 불가능한 원시 타입의 값이다. 심볼은 주로 이름의 충돌 위험이 없는 유일한 객체의 프로퍼티 키(property key)를 만

들기 위해 사용한다. 심볼은 Symbol 함수를 호출해 생성한다. 이때 생성된 심볼 값은 다른 심볼 값들과 다른 유일한 심볼 값이다.

```
// 심볼 key는 이름의 충돌 위험이 없는 유일한 객체의 프로퍼티 키
var key = Symbol('key');
console.log(typeof key); // symbol

var obj = {};
obj[key] = 'value';
console.log(obj[key]); // value
```

객체 타입 (Object type, Reference type)

객체는 데이터와 그 데이터에 관련한 동작(절차, 방법, 기능)을 모두 포함할 수 있는 개념적 존재이다. 달리 말해, 이름과 값을 가지는 데이터를 의미하는 프로퍼티(property)와 동작을 의미하는 메소드(method)를 포함할 수 있는 독립적 주체이다.

자바스크립트는 객체(object) 기반의 스크립트 언어로서 자바스크립트를 이루고 있는 거의 "모든 것"이 객체이다. 원시 타입(Primitives)을 제외한 나머지 값들(배열, 함수, 정규표현식 등)은 모두 객체이다. 또한 객체는 **pass-by-reference(참조에 의한 전달)** 방식으로 전달된다.

변수

변수(Variable)는 프로그램에서 사용되는 데이터를 일정 기간 동안 기억하여 필요한 때에 다시 사용하기 위해 데이터에 고유의 이름인 식별자(identifier)를 명시한 것이다. 변수에 명시한 고유한 식별자를 변수명이라 하고 변수로 참조할 수 있는 데이터를 변수값이라 한다.

> 식별자는 어떤 대상을 유일하게 식별할 수 있는 이름을 말한다. 식별자에는 변수명, 함수명, 프로퍼티명, 클래스명 등이 있다.

변수는 `var`, `let`, `const` 키워드를 사용하여 **선언**하고 할당 연산자를 사용해 값을 **할당**한다. 그리고 식별자인 변수명을 사용해 변수에 저장된 값을 **참조**한다.

```
var score;  // 변수의 선언
score = 80; // 값의 할당
score = 90; // 값의 재할당
score;      // 변수의 참조

// 변수의 선언과 할당
var average = (50 + 100) / 2;
```

사람을 고유한 이름으로 구별하듯이 변수도 사람이 이해할 수 있는 언어로 지정한 고유한 식별자(변수명)에 의해 구별하여 참조할 수 있다. 데이터는 메모리에 저장되어 있다. 메모리에 저장된 데이터를 참조하려면 데이터가 저장된 메모리 상의 주소(address)를 알아야 한다. 식별자는 데이터가 저장된 메모리 상의 주소를 기억한다. 따라서 식별자를 통해 메모리에 저장된 값을 참조할 수 있다. 또한 변수명을 통해 데이터의 의미를 명확히 할 수 있어 코드의 가독성이 좋아지는 효과도 있다.

변수의 선언

예를 들어 반지름의 길이가 2인 원의 넓이를 구해보자. 이때 원주율은 3.141592653589793라고 하자.

```
3.141592653589793 * 2 * 2; // 12.566370614359172
```

원의 넓이를 구하였으나 그 결과를 기억할 수 없다. 만약에 원의 넓이를 다시 사용해야 한다면 다시 구해야 한다. 변수를 사용하여 원의 넓이를 기억(캐싱)하고 기억된 원의 넓이를 재사용하여 높이가 5인 원기둥의 부피를 구해보자.

```
var circleArea = 3.141592653589793 * 2 * 2;
var cylinderVolume = circleArea * 5;
```

원주율 3.1415926535897 93은 재사용할 가능성이 크므로 변수에 저장하도록 하자. 원주율은 변하지 않는 상수이지만 자바스크립트는 상수를 별도 지원하지 않으므로 변수 이름을 대문자로 하여 상수임을 암시하도록 하자. 그리고 반지름과 원기둥의 높이도 값의 의미를 명확히하고 변화에 대처하기 쉽도록 변수에 저장하도록 하자.

> 원주율은 자바스크립트 빌트인 상수인 Math.PI을 통해 참조할 수 있다. 상수는 ES6의 const 키워드를 사용해 표현할 수 있다. 이것에 대해서는 나중에 자세히 살펴볼 것이다.

```
var PI = 3.141592653589793; // 상수
var radius = 2; // 변수
var circleArea = PI * radius * radius;
var cylinderHeight = 5;
var cylinderVolume = circleArea * cylinderHeight;
```

이처럼 변수는 애플리케이션에서 한번 쓰고 버리는 값이 아닌 값이 아닌 일정 기간 유지할 필요가 있는 값에 사용한다. 또한 변수를 사용하면 값의 의미가 명확해져서 코드의 가독성이 좋아진다.

변수의 존재 목적을 쉽게 이해할 수 있도록 의미있는 변수명을 지정하여야한다.

```
var x = 3;         // NG
var score = 100;   // OK
```

변수명은 식별자(identifier)로 불리기도 하며 명명 규칙이 존재한다.

- 반드시 영문자(특수문자 제외), underscore (_), 또는 달러 기호($)로 시작하여야 한다. 이어지는 문자에는 숫자(0~9)도 사용할 수 있다.

- 자바스크립트는 대/소문자를 구별하므로 사용할 수 있는 문자는 “A” ~ “Z” (대문자)와 “a” ~ “z” (소문자)이다.

변수를 선언할 때는 `var` 키워드를 사용한다. 등호(=, equal sign)는 변수에 값을 할당하는 할당 연산자이다.

```
var name;     // 선언
name = 'Lee'; // 할당

var age = 30; // 선언과 할당

var person = 'Lee',
    address = 'Seoul',
    price = 200;

var price = 10;
var tax   = 1;
var total = price + tax;
```

값을 할당하지 않은 변수 즉 선언만 되어 있는 변수는 `undefined`로 초기값을 갖는다. 선언하지 않은 변수에 접근하면 `ReferenceError`가 발생한다.

```
var x;
console.log(x); // undefined
console.log(y); // ReferenceError
```

변수의 중복 선언

var 키워드로 선언한 변수는 중복 선언이 가능하다. 다시 말해 변수명이 같은 변수를 중복해 선언해도 에러가 발생하지 않는다.

```
var x = 1;
console.log(x); // 1

// 변수의 중복 선언
var x = 100;
console.log(x); // 100
```

위 예제의 변수 x는 중복 선언되었다. 이처럼 변수를 중복 선언하면 에러없이 이전 변수의 값을 덮어쓴다. 만약 동일한 변수명이 선언되어 있는 것을 모르고 변수를 중복 선언했다면 의도치 않게 변수의 값을 변경하는 부작용을 발생시킨다. 따라서 변수의 중복 선언은 문법적으로 허용되지만 사용하지 않는 것이 좋다.

동적 타이핑 (Dynamic Typing)

자바스크립트는 동적 타입(dynamic/weak type) 언어이다. 이것은 변수의 타입 지정(Type annotation)없이 값이 할당되는 과정에서 값의 타입에 의해 자동으로 타입이 결정(Type Inference)될 것이라는 뜻이다. 따라서 같은 변수에 여러 타입의 값을 할당할 수 있다. 이를 동적 타이핑(Dynamic Typing)이라 한다.

```
var foo;

console.log(typeof foo);  // undefined

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

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

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

foo = 3.14;
console.log(typeof foo);  // number
```

```
foo = 'Hi';
console.log(typeof foo);  // string

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

변수 호이스팅(Variable Hoisting)

아래의 예제를 살펴보자.

```
console.log(foo); // ① undefined
var foo = 123;
console.log(foo); // ② 123
{
  var foo = 456;
}
console.log(foo); // ③ 456
```

var 키워드를 사용하여 선언한 변수는 중복 선언이 가능하기 때문에 위의 코드는 문법적으로 문제가 없다.

①에서 변수 foo는 아직 선언되지 않았으므로 ReferenceError: foo is not defined가 발생할 것을 기대했겠지만 콘솔에는 undefined가 출력된다.

이것은 다른 C-family 언어와는 차별되는 자바스크립트의 특징으로 **모든 선언문은 호이스팅(Hoisting)되기 때문**이다.

호이스팅이란 var 선언문이나 function 선언문 등 모든 선언문이 해당 Scope의 선두로 옮겨진 것처럼 동작하는 특성을 말한다. 즉, 자바스크립트는 모든 선언문(var, let, const, function, function*, class)이 선언되기 이전에 참조 가능하다.

변수가 어떻게 생성되며 호이스팅은 어떻게 이루어지는지 좀더 자세히 살펴보자. 변수는 3 단계에 걸쳐 생성된다.

> 💡 **선언 단계(Declaration phase)**
>
> 변수 객체(Variable Object)에 변수를 등록한다. 이 변수 객체는 스코프가 참조하는 대상이 된다.

> 💡 **초기화 단계(Initialization phase)**
>
> 변수 객체(Variable Object)에 등록된 변수를 메모리에 할당한다. 이 단계에서 변수는 undefined로 초기화된다.

> 💡 **할당 단계(Assignment phase)**
>
> undefined로 초기화된 변수에 실제값을 할당한다.

var 키워드로 선언된 변수는 선언 단계와 초기화 단계가 한번에 이루어진다. 즉, 스코프에 변수가 등록되고 변수는 메모리에 공간을 확보한 후 undefined로 초기화된다. 따라서 변수 선언문 이전에 변수에 접근하여도 Variable Object에 변수가 존재하기 때문에 에러가 발생하지 않는다. 다만 undefined를 반환한다. 이러한 현상을 변수 호이스팅(Variable Hoisting)이라한다.

이후 변수 할당문에 도달하면 비로소 값의 할당이 이루어진다.

앞에서 살펴본 예제를 호이스팅 관점에서 다시 한번 알아보도록 하자.

①이 실행되기 이전에 `var foo = 123;`이 호이스팅되어 ①구문 앞에 `var foo;`가 옮겨진다. (실제로 변수 선언이 코드 레벨로 옮겨진 것은 아니고 변수 객체(Variable object)에 등록되고 undefined로 초기화된 것이다.) 하지만 변수 선언 단계와 초기화 단계가 할당 단계와 분리되어 진행되기 때문에 이 단계에서는 foo에는 undefined가 할당되어 있다. 변수 foo에 값이 할당되는 것은 2행에서 실시된다.

②에서는 변수에 값이 할당되었기 때문에 123이 출력된다.

자바스크립트의 변수는 다른 C-family와는 달리 **블록 레벨 스코프(block-level scope)**를 가지지 않고 **함수 레벨 스코프(function-level scope)**를 갖는다. 단, ECMAScript 6에서 도입된 let, const 키워드를 사용하면 블록 레벨 스코프를 사용할 수 있다.

> 💡 **함수 레벨 스코프(Function-level scope)**
>
> 함수 내에서 선언된 변수는 함수 내에서만 유효하며 함수 외부에서는 참조할 수 없다. 즉, 함수 내부에서 선언한 변수는 지역 변수이며 함수 외부에서 선언한 변수는 모두 전역 변수이다.

> 💡 **블록 레벨 스코프(Block-level scope)**
>
> 코드 블록 내에서 선언된 변수는 코드 블록 내에서만 유효하며 코드 블록 외부에서는 참조할 수 없다.

따라서 코드 블록 내의 변수 foo는 전역변수이므로 전역에 선언된 변수 foo에 할당된 값을 재할당하기 때문에 ③의 결과는 456이 된다.

var 키워드로 선언된 변수의 문제점

ES5에서 변수를 선언할 수 있는 유일한 방법은 var 키워드를 사용하는 것이다. var 키워드로 선언된 변수는 아래와 같은 특징을 갖는다. 이는 다른 C-family 언어와는 차별되는 특징(설계상 오류)으로 주의를 기울이지 않으면 심각한 문제를 발생시킨다.

1. 함수 레벨 스코프(Function-level scope)

- 전역 변수의 남발

- for loop 초기화식에서 사용한 변수를 for loop 외부 또는 전역에서 참조할 수 있다.

2. var 키워드 생략 허용

- 의도하지 않은 변수의 전역화

3. 중복 선언 허용

- 의도하지 않은 변수값 변경

4. 변수 호이스팅

- 변수를 선언하기 전에 참조가 가능하다.

대부분의 문제는 전역 변수로 인해 발생한다. 전역 변수는 간단한 애플리케이션의 경우, 사용이 편리한 면이 있지만 불가피한 상황을 제외하고 사용을 억제해야 한다. 전역 변수는 유효 범위(scope)가 넓어서 어디에서 어떻게 사용될 지 파악하기 힘들다. 이는 의도치 않은 변수의 변경이 발생할 수 있는 가능성이 증가한다. 또한 여러 함수와 상호 의존하는 등 부수 효과(side effect)가 있을 수 있어서 복잡성이 증가한다.

변수의 유효 범위(scope)는 좁을수록 좋다.

ES6는 이러한 var의 단점을 보완하기 위해 let과 const 키워드를 도입하였다.