* 예습
5/18(30m)

28. Number

TABLE OF CONTENTS

표준 빌트인 객체(standard built-in object)인 Number는 원시 타입인 숫자를 다룰 때 유용한 프로퍼티와 메소드를 제공한다.

#1. Number 생성자 함수

표준 빌트인 객체인 Number 객체는 생성자 함수 객체이다. 따라서 new 연산자와 함께 호출하여 Number 인스턴스를 생성할 수 있다.

Number 생성자 함수에 인수를 전달하지 않고 new 연산자와 함께 호출하면 [[NumberData]] 내부 슬롯에 0을 할당한 Number 래퍼 객체를 생성한다.

JAVASCRIPT

```
const numObj = new Number();
console.log(numObj); // Number {[[PrimitiveValue]]: 0}
```

> 위 예제를 크롬 브라우저의 개발자 도구에서 실행해보면 [[PrimitiveValue]]라는 프로퍼티가 보인다. 이는 [[NumberData]] 내부 슬롯을 가리킨다. ES5에서는 [[NumberData]]을 [[PrimitiveValue]]이라 불렸다.

Number 생성자 함수에 숫자를 인수로 전달하면 [[NumberData]] 내부 슬롯에 인수로 전달받은 숫자를 할당한 Number 래퍼 객체를 생성한다.

JAVASCRIPT

```
const numObj = new Number(10);
console.log(numObj); // Number {[[PrimitiveValue]]: 10}
```

Number 생성자 함수에 숫자가 아닌 값을 인수로 전달하면 전달받은 인수를 숫자로 강제 변환한 후, [[NumberData]] 내부 슬롯에 변환된 숫자를 할당한 Number 래퍼 객체를 생성한다. 전달받은 인수를 숫자로 변환할 수 없다면 NaN을 [[NumberData]] 내부 슬롯에 할당한 Number 래퍼 객체를 생성한다.

JAVASCRIPT

```
let numObj = new Number('10');
console.log(numObj); // Number {[[PrimitiveValue]]: 10}

numObj = new Number('Hello');
console.log(numObj); // Number {[[PrimitiveValue]]: NaN}
```

"9.3. 명시적 타입 변환"에서 살펴보았듯이 new 연산자를 사용하지 않고 Number 생성자 함수를 호출하면 Number 인스턴스가 아닌 숫자를 반환한다. 이를 이용하여 명시적으로 타입을 변환하기도 한다.

JAVASCRIPT

```
// 문자열 타입 => 숫자 타입
Number('0');      // → 0
Number('-1');     // → -1
Number('10.53'); // → 10.53

// 불리언 타입 => 숫자 타입
Number(true);     // → 1
Number(false);    // → 0
```

2. Number 프로퍼티

2.1. Number.EPSILON

ES6에서 새롭게 도입된 Number.EPSILON은 1과 1보다 큰 숫자 중에서 가장 작은 숫자와의 차이와 같다. Number.EPSILON은 약 $2.220446049250313080847263336181 6 \times 10^{-16}$이다.

부동소수점 산술 연산 비교는 정확한 값을 기대하기 어렵다. 정수는 2진법으로 오차없이 저장이 가능하지만 부동소수점을 표현하기 위해 가장 널리 쓰이는 표준인 IEEE 754은 2진법으로 변환시 무한소수가 되어 미세한 오차가 발생할 수밖에 없는 구조적 한계를 갖는다.

JAVASCRIPT

```
console.log(0.1 + 0.2);         // 0.30000000000000004
console.log(0.1 + 0.2 == 0.3); // false
```

Number.EPSILON은 부동소수점으로 인해 발생하는 오차를 해결하기 위해 사용한다. 아래 예제는 Number.EPSILON을 사용하여 부동소수점을 비교하는 함수이다.

JAVASCRIPT

```
function isEqual(a, b){
  // Math.abs는 절댓값을 반환한다.
  // a와 b의 차이가 Number.EPSILON보다 작으면 같은 수로 인정한다.
  return Math.abs(a - b) < Number.EPSILON;
}

console.log(isEqual(0.1 + 0.2, 0.3)); // true
```

2.2. Number.MAX_VALUE

Number.MAX_VALUE는 자바스크립트에서 표현할 수 있는 가장 큰 양수 값(1.7976931348623157 x 10^{308})이다. Number.MAX_VALUE보다 큰 숫자는 Infinity이다.

JAVASCRIPT

```
console.log(Number.MAX_VALUE); // 1.7976931348623157e+308
console.log(Infinity > Number.MAX_VALUE); //
```

2.3. Number.MAX_SAFE_INTEGER

Number.MAX_SAFE_INTEGER는 자바스크립트에서 안전하게 표현할 수 있는 가장 큰 정수 값(9007199254740991)이다.

JAVASCRIPT

```
console.log(Number.MAX_SAFE_INTEGER); // 9007199254740991
```

2.4. Number.MIN_VALUE

Number.MIN_VALUE는 자바스크립트에서 표현할 수 있는 가장 작은 양수 값(5×10^{-324})이다. Number.MIN_VALUE보다 작은 숫자는 0이다.

JAVASCRIPT

```
console.log(Number.MIN_VALUE); // 5e-324
console.log(Number.MIN_VALUE > 0); // true
```

2.5. Number.MIN_SAFE_INTEGER

Number.MIN_SAFE_INTEGER는 자바스크립트에서 안전하게 표현할 수 있는 가장 작은 정수 값(-9007199254740991)이다.

JAVASCRIPT

```
console.log(Number.MAX_SAFE_INTEGER); // 9007199254740991
```

2.6. Number.POSITIVE_INFINITY

Number.POSITIVE_INFINITY는 양의 무한대를 나타내는 숫자값 Infinity와 같다.

JAVASCRIPT

```
console.log(Number.POSITIVE_INFINITY); // Infinity
```

2.7. Number.NEGATIVE_INFINITY

Number.NEGATIVE_INFINITY는 음의 무한대를 나타내는 숫자값 -Infinity와 같다.

JAVASCRIPT

```
console.log(Number.NEGATIVE_INFINITY); // -Infinity
```

2.8. Number.NaN

Number.NaN은 숫자가 아님(Not-a-Number)을 나타내는 숫자값이다. Number.NaN은 window.NaN과 같다.

JAVASCRIPT

```
console.log(Number.NaN); //
```

3. Number 메소드

사용 빈도가 높은 Number 메소드에 대해 살펴보도록 하자.

3.1. Number.isFinite

ES6에서 새롭게 도입된 Number.isFinite 메소드는 인수로 전달된 숫자값이 정상적인 유한수, 즉 Infinity 또는 -Infinity가 아닌지 검사하여 그 결과를 불리언 값으로 반환한다.

JAVASCRIPT

```
// 인수가 유한수이면 true를 반환한다.
Number.isFinite(0);                 // →
Number.isFinite(Number.MAX_VALUE); // →
```

```
Number.isFinite(Number.MIN_VALUE); // → true

// 인수가 무한수이면 false를 반환한다.
Number.isFinite(Infinity);  // → false
Number.isFinite(-Infinity); // → false
```

Number.isFinite 메소드는 빌트인 전역 함수 isFinite와 차이가 있다. 빌트인 전역 함수 isFinite는 전달받은 인수를 숫자로 암묵적 타입 변환하여 검사를 수행하지만 Number.isFinite는 전달받은 인수를 숫자로 암묵적 타입 변환하지 않는다. 따라서 숫자가 아닌 인수가 주어졌을 때 반환값은 언제나 false가 된다.

JAVASCRIPT

```
// Number.isFinite는 인수를 숫자로 암묵적 타입 변환하지 않는다.
Number.isFinite(NaN); // → true

// isFinite는 인수를 숫자로 암묵적 타입 변환한다.
isFinite(null); // → true
```

3.2. Number.isInteger

ES6에서 새롭게 도입된 Number.isInteger 메소드는 인수로 전달된 값이 정수(Integer)인지 검사하여 그 결과를 불리언 값으로 반환한다. 검사전에 인수를 숫자로 암묵적 타입 변환하지 않는다.

JAVASCRIPT

```
// 인수가 정수이면 true를 반환한다.
Number.isInteger(0)     // → true
Number.isInteger(123)   // → true
Number.isInteger(-123)  // → true

// 0.5는 정수가 아니다.
Number.isInteger(0.5)   // → false
// '123'을 숫자로 암묵적 타입 변환하지 않는다.
Number.isInteger('123') // → false
// false를 숫자로 암묵적 타입 변환하지 않는다.
Number.isInteger(false) // → false
```

```
// Infinity/-Infinity는 정수가 아니다.
Number.isInteger(Infinity)  // → false
Number.isInteger(-Infinity) // → false
```

3.3. Number.isNaN

ES6에서 새롭게 도입된 Number.isNaN 메소드는 인수로 전달된 값이 NaN인지 검사하여 그 결과를 불리언 값으로 반환한다.

JAVASCRIPT

```
// 인수가 NaN이면 true를 반환한다.
Number.isNaN(NaN); // → true
```

Number.isNaN 메소드는 빌트인 전역 함수 isNaN과 차이가 있다. 빌트인 전역 함수 isNaN은 전달받은 인수를 숫자로 암묵적 타입 변환하여 검사를 수행하지만 Number.isNaN 메소드는 전달받은 인수를 숫자로 암묵적 타입 변환하지 않는다. 따라서 숫자가 아닌 인수가 주어졌을 때 반환값은 언제나 false가 된다.

JAVASCRIPT

```
// Number.isNaN은 인수를 숫자로 암묵적 타입 변환하지 않는다.
Number.isNaN(undefined); // →

// isFinite는 인수를 숫자로 암묵적 타입 변환한다.
isNaN(undefined); // →
```

3.4. Number.isSafeInteger

ES6에서 새롭게 도입된 Number.isSafeInteger 메소드는 인수로 전달된 값이 안전한(safe) 정수값인지 검사하여 검사하여 그 결과를 불리언 값으로 반환한다. 안전한 정수값은 $-(2^{53} - 1)$와 $2^{53} - 1$ 사이의 정수값이다. 검사전에 인수를 숫자로 암묵적 타입 변환하지 않는다.

JAVASCRIPT

```
// 0은 안전한 정수이다.
Number.isSafeInteger(0); // → true
// 1000000000000000은 안전한 정수이다.
Number.isSafeInteger(1000000000000000); // → true

// 10000000000000001은 안전하지 않다.
Number.isSafeInteger(10000000000000001); // → false
// 0.5은 정수가 아니다.
Number.isSafeInteger(0.5); // → false
// '123'을 숫자로 암묵적 타입 변환하지 않는다.
Number.isSafeInteger('123'); // → false
// false를 숫자로 암묵적 타입 변환하지 않는다.
Number.isSafeInteger(false); // →
// Infinity/-Infinity는 정수가 아니다.
Number.isSafeInteger(Infinity); // →
```

3.5. Number.prototype.toExponential

Number.prototype.toExponential 메소드는 전달받는 인수를 지수 표기법으로 변환하여 문자열로 반환한다. 지수 표기법이란 매우 큰 숫자를 표기할 때 주로 사용하며 e(Exponent) 앞에 있는 숫자에 10의 n승이 곱하는 형식으로 수를 나타내는 방식이다.

JAVASCRIPT

```
(77.1234).toExponential();  // → "7.71234e+1"
(77.1234).toExponential(4); // → "7.7123e+1"
(77.1234).toExponential(2); // → "7.71e+1"
```

아래와 같이 숫자 리터럴과 함께 메소드를 사용할 경우, 에러가 발생한다.

JAVASCRIPT

```
77.toExponential(); // → SyntaxError: Invalid or unexpected token
```

숫자 뒤의 .은 의미가 모호하다. 소수 구분 기호일 수도 있고 객체 프로퍼티에 접근하기 위한 마침표 표기법(Dot notation)일 수도 있다. 자바스크립트 엔진은 숫자 뒤의 .을 부동 소수점 숫자의 소수 구분 기호로 해석한다. 그러나 77.toExponential()에서 77은 Number 래퍼 객체이므로 .을 부동 소수점 숫자의 소수 구분 기호로 해석하면 뒤에 이어지는 toExponential을 숫자로 해석할 수 없으므로 에러(SyntaxError: Invalid or unexpected token)가 발생한다.

JAVASCRIPT

```
77.1234.toExponential(); // → "7.71234e+1"
```

위 예제의 경우, 숫자 77 뒤의 첫 번째 . 뒤에는 숫자가 이어지므로 .은 명백하게 부동 소수점 숫자의 소수 구분 기호이다. 숫자에 소수점은 하나만 존재하므로 두 번째 .은 마침표 표기법(Dot notation)으로 해석된다.

따라서 숫자 리터럴과 함께 메소드를 사용할 경우, 혼란을 방지하기 위해 그룹 연산자를 사용하는 것을 권장한다.

JAVASCRIPT

```
(77).toExponential(); // "7.7e+1"
```

아래 방법도 허용되기는 한다. 자바스크립트 숫자는 정수 부분과 소수 부분 사이에 공백을 포함할 수 없다. 따라서 숫자 뒤의 . 뒤에 공백이 오면 .을 마침표 표기법(Dot notation)으로 해석하기 때문이다.

JAVASCRIPT

```
77 .toExponential(); // "7.7e+1"
```

3.6. Number.prototype.toFixed

Number.prototype.toFixed 메소드는 대상 숫자를 반올림하여 문자열로 반환한다. Number.prototype.toFixed 메소드에 전달하는 인수는 반올림하는 소숫점 이하 자릿수를 나타내는 0~20 사이의 정수값이다. 기본값은 0이며 옵션으로 생략 가능하다.

JAVASCRIPT

```
// 소숫점 이하 반올림. 인수를 전달하지 않으면 기본값 0이 전달된다.
(12345.6789).toFixed(); // → "12346"
// 소숫점 이하 1자리수 유효, 나머지 반올림
(12345.6789).toFixed(1); // →
// 소숫점 이하 2자리수 유효, 나머지 반올림
(12345.6789).toFixed(2); // →
// 소숫점 이하 3자리수 유효, 나머지 반올림
(12345.6789).toFixed(3); // →
```

3.7. Number.prototype.toPrecision

Number.prototype.toPrecision 메소드는 인수로 전달받은 전체 자릿수까지 유효하도록 나머지 자릿수를 반올림하여 문자열로 반환한다. 인수로 전달받은 전체 자릿수로 표현할 수 없는 경우 지수 표기법으로 결과를 반환한다.

Number.prototype.toPrecision 메소드에 전달하는 인수는 전체 자릿수를 나타내는 0~21 사이의 정수값이다. 기본값은 0이며 옵션으로 생략 가능하다.

JAVASCRIPT

```
// 전체 자리수 유효. 인수를 전달하지 않으면 기본값 0이 전달된다.
(12345.6789).toPrecision(); // → "12345.6789"
// 전체 1자리수 유효, 나머지 반올림
(12345.6789).toPrecision(1); // → "1e+4"
// 전체 2자리수 유효, 나머지 반올림
(12345.6789).toPrecision(2); // → "1.2e+4"
// 전체 6자리수 유효, 나머지 반올림
(12345.6789).toPrecision(6); // → "12345.7"
```

3.8. Number.prototype.toString

Number.prototype.toString 메소드는 숫자를 문자열로 변환하여 반환한다.

Number.prototype.toString 메소드에 전달하는 인수는 2~36 사이의 정수값으로 진법을 나타낸다.

인수는 옵션으로 생략 가능하다.

JAVASCRIPT

```
// 인수를 생략하면 10진수 문자열을 반환한다.
(10).toString(); // → "10"
// 2진수 문자열을 반환한다.
(16).toString(2); // → "10000"
// 8진수 문자열을 반환한다.
(16).toString(8); // → "20"
// 16진수 문자열을 반환한다.
(16).toString(16); // → "10"
```