3장. 타입, 값, 변수

박유진

데이터 타입

- 원시타입
  - 숫자, 텍스트의 나열(문자, string), 불리언 진리값(boolean)
  - null, undefined는 원시값이지만 자기 자신만을 값으로 갖는 독립적인 타입
- 객체타입
  - Object 타입
  - 이름과 값을 갖는 프로퍼티의 집합(값은 어느 것도 될 수 있다)

상수와 리터럴

- 리터럴(literal)
  - 의미 : 있는 그대로의
  - 사전 : 소스 코드의 고정된 값을 대표하는 용어
  - 상수와 같은 의미인가?

상수와 리터럴

- 상수(constant)
    - 변하지 않는 변수

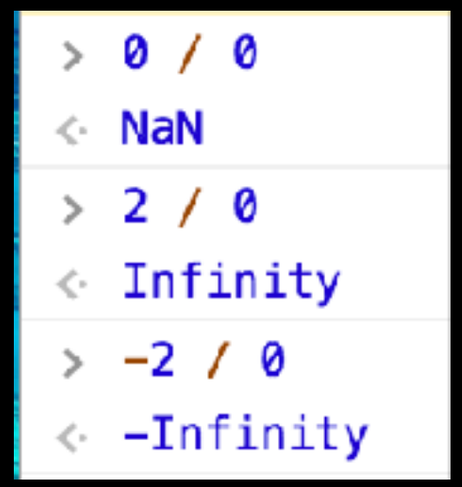
상수와 리터럴

- 차이점
- 상수 : 메모리에 할당된 공간(메모리의 주소값)이 변하지 않음
- 리터럴 : 이 공간에 저장된 값 자체가 변하지 않음
    - 상수와 마찬가지로 메모리 공간에 값이 변하지 않도록 저장되지만, 그 이름이 없다.

3.1 숫자

- 모든 숫자를 실수로 표현(64비트 실수 형태)
- 정수 / 부동소수점 리터럴로 표현 가능
- 산술 연산

- +, -, *, /, % 연산과 Math 객체의 다양한 프로퍼티 사용
- js에서 0으로 나누는 연산은 에러가 아니다.
  - 이런 경우 무한대가 발생
- 0을 0으로 나누면 NaN 출력



3.2 텍스트

- string : 16비트 값들이 연속적으로 나열된 변경이 불가능한 값
- 문자열은 0 기반의 인덱싱을 사용
- js에는 문자 하나를 표현하는 타입이 없다.

3.2 텍스트

이스케이프 시퀀스	유니코드	의미
\0	\u0000	널 문자
\b	\u0008	백스페이스
\t	\u0009	수평 탭
\n	\u000A	줄바꿈 문자
\v	\u000B	수직 탭
\f	\u000C	폼 피드 (새로운 페이지로 이동)
\r	\u000D	캐리지 리턴 (현재 행의 처음으로 이동)
\"	\u0022	큰 따옴표
\'	\u0027	작은 따옴표
\\	\u005C	역슬래시
\xXX		두 개의 16진수 XX에 의해 지정되는 Latin-1 문자
\uXXXX		네 개의 16진수 XXXX에 의해 지정되는 유니코드 문자

- 문자열을 포함하려면 ‘’나 “”로 둘러싸면 된다.
- 이스케이프 문자열
  - ‘\’는 뒤따라 나오는 문자와 결합될 경우 다른 방식으로는 표현할 수 없는 문자를 표현한다.
- ‘+’ 연산자를 문자열에 적용하면 이어 붙일 수 있다.
- 문자열 관련 메소드는 호출 시에 기존 문자열을 수정하지 않는다.

```
> temp = "hello " + "youjin"
<· "hello youjin"
> temp.length
<· 12
```

- ECMAScript5에서 문자열은 읽기 전용 배열처럼 취급될 수 있고 charAt() 메소드를 사용해서 문자 각각에 접근 가능하다.

```
> temp[0]
<· "h"
> temp[temp.length - 1]
<· "n"
```

3.3 불리언 값

- 참/거짓, on/off, yes/no 를 표현한다.
- 항상 예약어인 true와 false중 하나의 값으로 평가된다.
- 일반적으로 비교의 결과로 생성된다.

3.4 null과 undefined

- null / undefined

```
> null == undefined
< true
> null === undefined
< false
```

  - null : 아무 값도 갖지 않음을 나타냄(값의 부재 표현)

```
> typeof null
< "object"
```

  - undefined : 초기화되어 있지 않은 변수나 존재하지 않는 객체나 배열의 원소 값에 접근하려고 할 때 얻는 값

```
> typeof undefined
< "undefined"
```

    - 정의되지 않은 값
    - 시스템 상 오류성 값 부재 표현

변수와 프로퍼티

- js에서의 변수 vs 프로퍼티

```
> var a = 'test'
<- undefined
> this.b = 'test'
<- "test"
```

  - 변수 : var(private)로 선언한 것. 컨텍스트(실행 영역)의 일부
  - 프로퍼티 : this(public)로 선언한 것. 객체의 일부

- 예제)

```
> var setMyName = function( value ){
    var name = value;
    this.name = value;
  };
<- undefined
```

```
> var t2 = new setMyName("hiiiii")
<- undefined
> console.log(t2.name)
  hiiiii
<- undefined
> typeof t2
<- "object"
> t2.name
<- "hiiiii"
```

  - 여기서 name은 var로 선언한 name이 아닌 this로 선언한 name
  - t2는 setMyName을 통해 만들어진 객체이며, name은 객체의 프로퍼티가 된다.

3.4 전역객체

- 전역 객체의 프로퍼티는 js 프로그램 전역에서 사용할 수 있게 정의된 심벌
- 클라이언트 측 js에서 window 객체는 브라우저 창에 포함된 모든 js 코드를 위한 전역 객체이다. 이 전역 window 객체는 자기 자신을 참조하는 window 프로퍼티를 갖고 있는데, 전역 객체를 참조할 때 this 대신 사용 가능하다.
- 최초 전역 객체가 생성될 때, 그 안에서 js에서 사용하는 모든 전역 값이 정의된다.
    - 내가 만약 전역 변수를 선언한다면, 이 변수는 전역 객체의 프로퍼티가 된다.

3.6 wrapper 객체

- 문자열, 숫자, 불리언의 프로퍼티에 접근하려 할 때, 생성되는 임시 객체를 래퍼 객체라고 알려져 있다.

- String(), Number(), Boolean()

```
> typeof s
<· "string"
> var S = new String(s)
<· undefined
> typeof S
<· "object"
```

3.7 변경 불가능한 원시 타입 값과 변경 가능 객체 참조

- 원시 타입(undefined, null, boolean, number, string…)값은 수정할 수 없다.

- 참조 타입인 객체는 자신의 값을 변경할 수 있다.

3.8 타입 변환

- 변환과 동치
  - 동치 연산자(==)를 수행할 때, 타입을 유연하게 변화시켜서 판단한다.
- 자료형 변한을 수행하지 않는 '===' 동치 연산자를 사용하면 서로 변환 가능한 값이라고 해서 동치인것이 아닌 걸 확인 가능

```
> null == undefined
< true
> "0" == 0
< true
> 0 ==false
< true
> "0" == false
< true
```

```
> null === undefined
< false
> "0" === 0
< false
> 0 === false
< false
> "0" === false
< false
```

3.8 타입 변환

- 명시적 변환

```
> Number("3")
<· 3
> String(false)
<· "false"
> Boolean([])
<· true
> Object(3)
<· ▶ Number {3}
```

- 객체에서 원시 타입으로 변환
  - 객체 -> 불리언
  - 객체 -> 숫자
  - 객체 -> 문자열

```
> Boolean(new Boolean(false))
<· true
> Boolean(new Boolean(false))
<· true
> b = [1,2,3].toString()
<· "1,2,3"
> typeof(b)
<· "string"
> var d = new Date(2010, 0, 1)
<· undefined
> d.valueOf()
<· 1262271600000
> typeof(d.valueOf())
<· "number"
```

3.9 변수 선언

- var 키워드를 통해 변수를 선언한다.
- 변수 선언에 타입을 명시하지 않는다.
- 초기값을 지정하지 않으면 undefined 값을 가진다.

3.10 변수의 유효 범위

- 전역 변수
  - 선언시, var가 필요없다.
- 지역 변수
  - 함수 내부에서만 정의
  - 함수의 매개변수도 지역변수
  - 반드시 var 키워드로 선언해야 함

3.10 변수의 유효 범위

- 블록 유효범위(block scope)
  - 블록 안에 있는 코드는 자신만의 유효 범위를 가지며, 변수는 해당 블록 내에서만 유효하다.
- js에서는 이 개념이 없는 대신 ‘함수 유효범위’를 사용한다.
  - 변수는 해당 변수가 정의된 함수 안에서 보이고, 그 함수 안에 중첩된 함수 안에서도 보인다.