1주 2회차

1. 아래의 코드에서 표현식인 부분과 표현식이 아닌 부분에 대해서 구분하시오

```
var x;
x=100;
//*
----------------------------
*/
var a = y = 100;
console.log(a);
//*
----------------------------
*/
var foo = var x;
```

- 표현식 : 값으로 평가될 수 있는 문으로 평가되면 새로운 값을 생성하거나 기존 값을 참조하는 것이다.
- `var x;` : 변수 `x`를 선언하는 부분으로 표현식이 아니다.
- `x = 100;` : 변수 `x`에 값 100을 할당하는 부분이며, 할당 연산자 `=` 을 사용하고 있어 값을 반환하므로 표현식이다.
- `var a = y = 100;` : 변수 `a`와 `y`를 선언하고, 둘 모두에게 값 100을 할당하는 부분이며, 할당 연산자 `=`이 사용하고 있어 값을 반환하므로 표현식이다
- `console.log(a);` : 변수 `a` 의 값을 로그에 출력하는 부분이므로 표현식이다.
- `var foo = var x;` : `var foo`는 변수 `foo`를 선언하고, `var x` 부분은 변수 `x`를 선언하려고 시도하는데, 유효한 문법이 아니므로 표현식이 아니다.

2. 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);
```

- 출력결과

```
65 65 65
true
true
```

- 이유
  - 자바스크립트에선 모두 실수로 취급하며 내부적으로 값을 10진수로 처리한다.
  - 해당 값들은 모두 10진수로 65 이므로 `console.log()`를 통해 출력한 결과는 모두 65다.
  - `binary === hex`와 `binary === octal`를 비교하는데 값이 모두 65로 모두 참이므로 `true`가 출력되는 것이다.

3. 다 실수라면 아래의 비교문의 결과는 어떻게 나올까?

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

- `console.log(1 === 1.0);` : 값과 데이터 형식이 완전히 동일한지 확인하므로 `true`가 출력된다. (**1**은 정수이고 **1.0**은 부동 소수점 숫자이지만 값은 같다. `===` 연산자를 사용하여 비교하면 형 변환이 발생하지 않는다. 즉, 두 값이 다른 데이터 형식을 가지고 있어도 true가 반환될 수 있다.)
- `console.log(4 / 2);` : 나눗셈 연산이며, **4**를 **2**로 나눈 결과는 **2**이므로 **2**가 출력된다.
- `console.log(3 / 2);` : 나눗셈 연산이며, **3**을 **2**로 나눈 결과는 **1.5**이므로 **1.5**가 출력된다. (세미콜론 이 없어도 실행되는데 자바스크립트 엔진이 자동으로 문장을 해석하고 세미콜론을 추가해주기 때문에 코드가 잘 작동한다.)

4. 아래의 세가지를 console을 이용하여 도출해보세요.

```
Infinity
-Infinity
NaN
```

```
console.log(1 / 0);
console.log(-1 / 0);
console.log(0 / 0);
```

- 1 / 0 은 양의 무한대를 나타내므로 Infinity를 출력한다.

- -1 / 0 은 음의 무한대를 나타내므로 -Infinity를 출력한다.
- 0 / 0 은 산술 연산이 불가능하므로 NaN을 출력한다.

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

- 식별자로 인식해 ReferenceError가 발생한다.

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

- 더블쿼터 안의 싱글쿼터는 싱글쿼터가 문자열로 인식된다.
- 싱글쿼터 안의 더블쿼터는 더블쿼터가 문자열로 인식된다.
- 예시

```
console.log("This 'single quotes' inside");    // 출력 : This 'single quotes' inside
console.log('This "double quotes" inside');    // 출력 : This "double quotes" inside
```

7. 그렇다면 아래의 코드는 어떤식으로 다를까?

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

- 출력결과

```
a + b = 3
a + b = ${1 + 2}
```

- 첫 번째 줄 코드는 벡틱(``)을 사용하여 문자열을 생성하고 `${1 + 2}` 표현식을 문자열 내에서 사용하고 있다. 이런 방식으로 벡틱을 사용하면 템플릿 리터럴 이라는 문자열 표기법을 사용할 수 있고, `${}` 안에 자바스크립트 표현식을 삽입하여 계산 결과를 문자열로 포함 시킬 수 있어 출력결과가 `a + b = 3`이 된다.
- 두 번째 줄 코드는 작은 따옴표('')를 사용하여 문자열을 생성하고 `${1 + 2}`와 같은 표현식을 사용하고 있지만 작은 따옴표로 둘러싸인 문자열에서는 템플릿 리터럴 구문이 동작하지 않는다. 즉, `${1 + 2}`는 일반 문자열로 처리되고 변수 또는 표현식의 값으로 대체되지 않아 `a + b = ${1 + 2}`와 같은 출력결과가 나온다.

8. 의도적 부재를 왜 사용할까?

- `null`은 변수에 값이 없음을 나타내는 의미이므로 코드를 읽는 사람들은 해당 변수가 명시적으로 값이 없음을 이해할 수 있다.
- 변수가 값이 없음을 나타내기 위해 `null`을 사용하면, 의도치 않은 오류를 방지할 수 있다.
- 디버깅을할 때 변수의 상태를 파악하기 쉽다.
- GC 가 쓰지 않는 메모리 참조를 해제할 때 유용하다.
    - `null`을 사용하면 메모리를 효율적으로 관리할 수 있다.
    - 예를들어 클로저에서 참조하는 외부 변수가 더 이상 필요하지 않을 때 해당 변수에 `null`을 할당해 불필요한 참조를 제거할 수 있다.

9. 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;
```

- 메모리 관리와 가비지 컬렉션을 고려한 옳은 선택이라고 생각한다.
- 다른 방법
    - 변수에 `undefined`를 할당한다. (추천하지 않는다.)
    - 변수를 `var` 키워드로 재선언하면 이전 변수를 덮어쓸 수 있다. (이전 변수와 연결된 값은 더 이상 참조하지 않는다.)
    - 변수의 범위를 함수 내부로 제한하여 함수 외부에서 변수에 접근할 수 없게 만든다. (함수가 종료되면 해당 변수는 소멸된다.)

- 목적에 따라 변수를 소멸시키는 방법이 다를 것이다. 예를들어 다른 부분과 연관되는 변수인 경우 변수 재선언 또는 범위 제한 같은 방법을 사용하면 조금 더 좋을 것이다.

10. 변수의 타입 검사를 하기 위해 조건에 아래 와 같은 코드를 적용한다면 undefined가 뜰 것이다. 왜 그럴까?

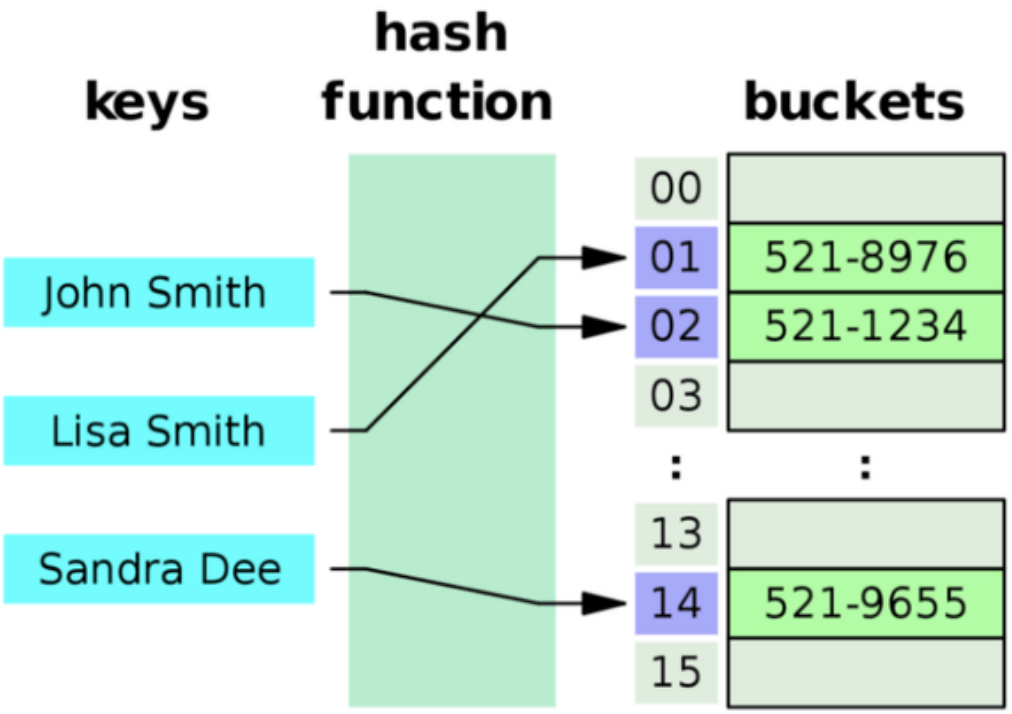
```
if(a === typeof null)
  console.log("hello")
```

- `typeof null`은 자바스크립트 초기 버전의 버그로 결과는 `object`다. 즉, `null`은 객체라는 것으로 잘못 알려주고 있는 것이다. 해당 버그는 수정될 수 없는 버그다. 왜냐하면 이게 수정되면 기존 코드에 부정적인 영향을 미칠 수 있기 때문이다.

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

- 자바스크립트 엔진 제조사 구현에 따라 다를 수 있다.

- V8 엔진 (객체)



- 프로퍼티의 개수가 정해져 있지 않고, 동적으로 추가되고 삭제될 수 있다.
- 프로퍼티의 값에도 제약이 없기에, 객체는 원시 값과 같이 확보해야 할 메모리 공간 크기를 사전에 확보할 수가 없다.
- 원시 값에 비해 관리하는 방법이 복잡하고 비용이 크다. 메모리 소비도 커질 수 있고 객체 생성 후 프로퍼티에 접근하는 방식도 비용이 많이 든다.
- 자바스크립트 객체는 프로퍼티 키를 인덱스로 하는 해시 테이블이다. 대부분의 자바스크립트 엔진은 해시 테이블과 유사하다.
- 프로퍼티에 접근하기 위해 동적 탐색이 아닌 히든 클래스 방식을 사용해 C++ 객체 프로퍼티에 접근하는 정도의 성능을 보장한다.

- ECMAScript 사양에 따르면 숫자 타입은 8비트 숫자 타입을 표현하는 배정밀도 64비트 부동소수점 형식을 사용한다고 명시되어 있다. 모든 수를 실수로 처리하며, 정수만 표현하기 위한 데이터 타입이 별도로 존재하지 않는다.
- 문자열 타입은 텍스트 데이터를 나타내는 데 사용한다. 문자열은 작은따옴표(''), 큰따옴표(""), 백틱(``)으로 감싼다. 일반적인 표기법은 작은 따옴표를 사용한다.

12. 심벌 테이블 이라는 뜻을 알아보시오

- 컴파일러 또는 인터프리터 같은 언어 변환기에서 사용되는 데이터 구조로 패키지나 함수, 클래스 정보를 포함한 코드 내에 정의된 여러 변수들에 대한 정보가 담긴 테이블이다.
- 심볼 테이블에는 소스코드에서 참조되는 심볼들의 이름과 주소가 정의되어 있다.
- 해당 오브젝트 파일의 심볼 정보만 가지고 있어 다른 파일에 참조되고 있는 심볼의 경우 저장할 수 없다.
- 함수를 실행하면 함수의 로컬 심볼 테이블이 생기고, 함수 내의 모든 변수 할당은 이 곳에 저장되어 인터프리터가 코드를 해석할 때 변수의 실제 값을 찾기위해 로컬 심볼 테이블을 먼저 찾아보고 그 후로, 그 다음으로 함수를 둘러싸고 있는 로컬 심볼 테이블에서 찾아보고, 글로벌 심볼 테이블, 내장 심볼 테이블 순으로 찾는다.
- 각 모듈 별로 글로벌 심볼 테이블을 갖고 있는데, 그래서 사용자의 전역 변수와 충돌이 발생하지 않고 모듈에서 전역 변수를 사용할 수 있다.

13. 대표적인 동적/정적 언어를 조사해보시오

- 동적언어

    - JavaScript : 브라우저에서 실행되며, 동적 타입 언어로 변수의 데이터 타입이 실행 시간에 결정된다.
    - Python : 데이터 분석, 웹 개발, 인공 지능 및 기계 학습 분야에서 널리 사용된다.
    - Ruby : 객체 지향 언어로, 웹 개발 프레임워크인 Ruby on Rails와 함께 사용되며 메타 프로그래밍을 지원한다.
    - SmallTalk : 객체 지향 프로그래밍 언어의 선구자 중 하나로 최초로 그래픽 사용자 인터페이스, GUI를 제공한 언어다.
- 정적 언어
    - Java : 강력한 타입 체크와 컴파일 시 타입 검사를 수행한다. 자바는 대규모 응용 프로그램 개발에 많이 사용된다.
    - C : 프로그래밍 언어에 직간접적으로 많은 영향을 준 언어로 주로 메모리와 하드웨어를 직접 제어하는데 사용된다.
    - C++ : C++은 C 언어의 확장으로, 정적 언어로 변수 및 데이터 타입이 컴파일 시간에 결정된다. 시스템 프로그래밍, 게임 개발, 임베디드 시스템에서 사용된다.
    - C# : Microsoft의 개발 플랫폼인 .NET Framework와 연동되는 언어로 Windows 애플리케이션 및 게임 개발에 사용된다.
    - Swift : Apple의 iOS 및 macOS 애플리케이션 개발을 위한 언어로 변수와 함수의 데이터 타입이 명시적으로 선언된다.
    - Rust : 메모리 안전성을 강조하는 언어로, 정적 언어의 장점을 가지고 있으며, 시스템 프로그래밍 및 웹 개발에서 사용된다.

14. 아래의 코드를 실행하시면 됩니다

```
var a = '1';

console.log(+a, typeof +a);
console.log(a, typeof a);

a = true;
console.log(+a, typeof +a);
console.log(a, typeof a);

a = false;
console.log(+a, typeof +a);
console.log(a, typeof a);

a = 'Hi';
console.log(+a, typeof +a);
console.log(a, typeof a);
```

- 출력결과

```
1 'number'
1 string
1 'number'
true 'boolean'
0 'number'
false 'boolean'
NaN 'number'
Hi string
```

- typeof는 피연산자의 평가 전 자료형을 나타내는 문자열을 반환하는 명령어다.
- `var a = '1' ;` : 문자열 `1`을 변수 `a`에 할당한다.
- `console.log(+a, typeof +a);` : `+a`는 문자열 `1`을 숫자로 변환하고, 결과는 `1`이다. `typeof +a`는 숫자 타입인 `number`를 반환한다
- `console.log(a, typeof a);` : `a`는 문자열 `1`이다. `typeof a`는 문자열 타입인 `string`을 반환합니다.
- `a = true;` : 불리언 값 `true`를 변수 `a`에 할당한다.
- `console.log(+a, typeof +a);` : `+a`는 불리언 `true`를 숫자로 변환하고, 결과는 `1` 이다. `typeof +a`는 숫자 타입인 `number`를 반환한다.
- `console.log(a, typeof a);` : `a`는 불리언 값 `true`이다. `typeof a`는 불리언 타입인 `boolean`을 반환한다.

- `a = false;` : 불리언 값 `false`를 변수 `a`에 할당한다.
- `console.log(+a, typeof +a);` : `+a`는 불리언 `false`를 숫자로 변환하고, 결과는 `0` 이다. `typeof +a`는 숫자 타입인 `number`를 반환한다.
- `console.log(a, typeof a);` : `a`는 불리언 값 `false`이다. `typeof a`는 불리언 타입인 `boolean`을 반환한다.
- `a = 'Hi';` : 문자열 `'Hi'`를 변수 `a`에 할당한다.
- `console.log(+a, typeof +a);` : `+a`는 문자열 `Hi`를 숫자로 변환할 수 없기 때문에 `NaN` 이 된다. `typeof +a`는 숫자 타입인 `number`를 반환한다.
- `console.log(a, typeof a);` : `a`는 문자열 `Hi`이며, `typeof a`는 문자열 타입인 `string`을 반환한다.

15. 암묵적 타입 변환 또는 타입 강제 변환에 대해서 알아보시오

- 프로그래밍 언어에서 데이터 타입을 자동으로 변환하는 과정을 말한다.
- 주로 연산자와 표현식에서 다른 데이터 타입 간에 발생하며, 데이터 타입이 서로 다른 경우에 이루어진다.
- 자바스크립트와 같은 동적 타입 언어에서는 암묵적 타입 변환이 빈번하게 발생한다.
- 예를들어 자바스크립트에서 `+` 연산자로 숫자와 문자열을 함께 사용할 때 암묵적 타입 변환이 발생한다. (숫자와 문자열 간 변환)

```
var num = 1;
var str = "2";
var result = num + str; // 결과: 12
```

- 숫자 `1`이 문자열로 변환되어 문자열 `2`와 연결되어 `12`가 반환된다.

16. 아래의 비교가 뭐가 다른지 알아보시오.

```
5 == 5;
5 == '5';
// ================================================
5 === 5;
5 === '5';
// ================================================
'0' == '';
0 == '';
0 == '0';
// ================================================
false == 'false';
false == '0';
false == null;
false == undefined;
// ---
NaN === NaN
0 == -0
0 === -0
```

- `==` 연산자는 두 피연산자의 값의 타입이 다를 경우 자동으로 일부 피연산자의 타입을 변환 후 값을 비교해서, 같으면 `true`, 다르면 `false` 라고 한다.
- `===` 연산자는 값과 타입이 모두 같은지를 비교해서, 같으면 `true`, 다르면 `false` 라고 한다. `==`연산자에 비해 비교하는 방식이 엄격하다.

- `5 == 5;` : 두 개의 숫자 5가 동일한 값을 가지기 때문에 `true` 다.
- `5 == '5';` : `==` 연산자는 타입 간의 암묵적 타입 변환을 수행한다. 문자열 `5`를 숫자로 변환하여 두 값이 같다고 판단하기 때문에 `true` 다.

- `5 === 5` : `===` 연산자는 값과 데이터 타입이 모두 같아야 `true`이다. 두 개의 숫자 5가 같기 때문에 `true`다.
- `5 === '5';` : `===` 연산자는 데이터 타입이 다르기 때문에 `false`다.
- `'0' == '';` : `'0'`과 `''`은 다르기 때문에 `false`다.
- `0 == '';` : `==` 연산자는 문자열을 숫자로 변환한다. 비어 있는 문자열은 숫자 0으로 변환되므로 `0`과 `0`은 같기 때문에 `true` 다.
- `0 == '0';` : `==` 연산자는 문자열을 숫자로 변환한다 `'0'`은 숫자 0으로 변환되므로. `0`과 `0`은 같기 때문에 `true`다.
- `false == 'false';` : `==` 연산자는 문자열 `'false'`를 불리언 값 `false`로 변환하지 않는다. 불리언 `false`와 문자열 `false`는 다르기 때문에 `false` 다.
- `false == '0';` : `==` 연산자는 문자열 `'0'`을 불리언 값 `false`로 변환한다. 불리언 `false`와 문자열 `false`는 같기 때문에 `true`다.
- `false == null;` : `==` 연산자는 `null`과 `undefined`를 제외한 다른 값과 비교하기 때문에 `false`를 반환한다.
- `false == undefined;` : 위의 경우와 마찬가지로 `==` 연산자는 `null`과 `undefined`를 제외한 다른 값과 비교하기 때문에 `false`를 반환한다.
- `NaN === NaN` : `===` 연산자는 `NaN` 값도 자기 자신과 일치하지 않는다고 판단하기 때문에 `false` 다.
- `0 == -0` : `==` 연산자는 `0`과 `0`을 같은 값으로 간주하기 때문에 `true` 다.

1. `0 === -0` : `===` 연산자는 `0`과 `0`의 데이터 타입과 값이 일치하기 때문에 `true`다.

17. 아래 코드의 결과가 다른 이유

```
- 0 === 0 ;
Object.is(-0,0)
```

```
NaN === NaN;
Object.is(NaN,NaN);
```

- 출력결과

```
true
false
false
true
```

- `Object.is();` 명령어는 `===` 연산자와 비교하는 방식이 비슷하지만 +0, -0 와 NaN 비교 비교가 가능하기 때문에 결과가 다르다.

18. 위에 있는 반환 값을 다 나타내보시오. (string, number, boolean, underfined, symbol, object, function)

예시)

```
typeof 1
```

- string (문자열)

```
typeof "string";
```

- number

```
typeof 1;
```

- boolean

```
typeof true;
```

- underfined

```
typeof undefined;
```

- symbol

```
typeof Symbol("symbol");
```

- object (배열도 객체로 간주된다)

```
typeof {};
typeof [];
```

- function

```
typeof function() {};
```

19. 알고리즘 문제를 자바스크립트를 이용해서 풀어보기

- 백준 10828번 (스택)

```
// 입력
const fs = require("fs");
const filePath = process.platform === "linux" ? "/dev/stdin" : "./input.txt";
let input = fs.readFileSync(filePath).toString().split("\n");

// 스택이 저장되는 배열
stack = [];
// 반복문을 돌때마다 출력하면 시간 초과가 나서 한번에 출력시키기 위한 배열
result = [];

// 입력한 값의 첫 번째 값 제거 후 길이
const len = input.shift();

// len의 크기만큼 반복 (입력한 명령의 수 만큼 반복)
for (let i = 0; i < len; i++) {
    switch(input[i]) {
        // 입력한 값이 pop일때
        case 'pop':
            // 스택이 비어있을 경우
            if(stack.length === 0)
                result.push("-1");
            else
                result.push(stack.pop());
            break;

        // 입력한 값이 size일때
        case 'size':
            result.push(stack.length);
            break;

        // 입력한 값이 empty일때
        case 'empty':
            // 스택이 비어있을 경우
            if (stack.length === 0)
                result.push(1);
            else
                result.push(0);
            break;

        // 입력한 값이 top일때
        case 'top':
            // 스택이 비어있을 경우
            if (stack.length === 0)
                result.push(-1);
            else
                result.push(stack[stack.length - 1]);
            break;

        // 입력한 값이 push일때
        default:
            stack.push(input[i].split(" ")[1]);
            break;
```

```
    }
}



console.log(result.join('\n'));
```