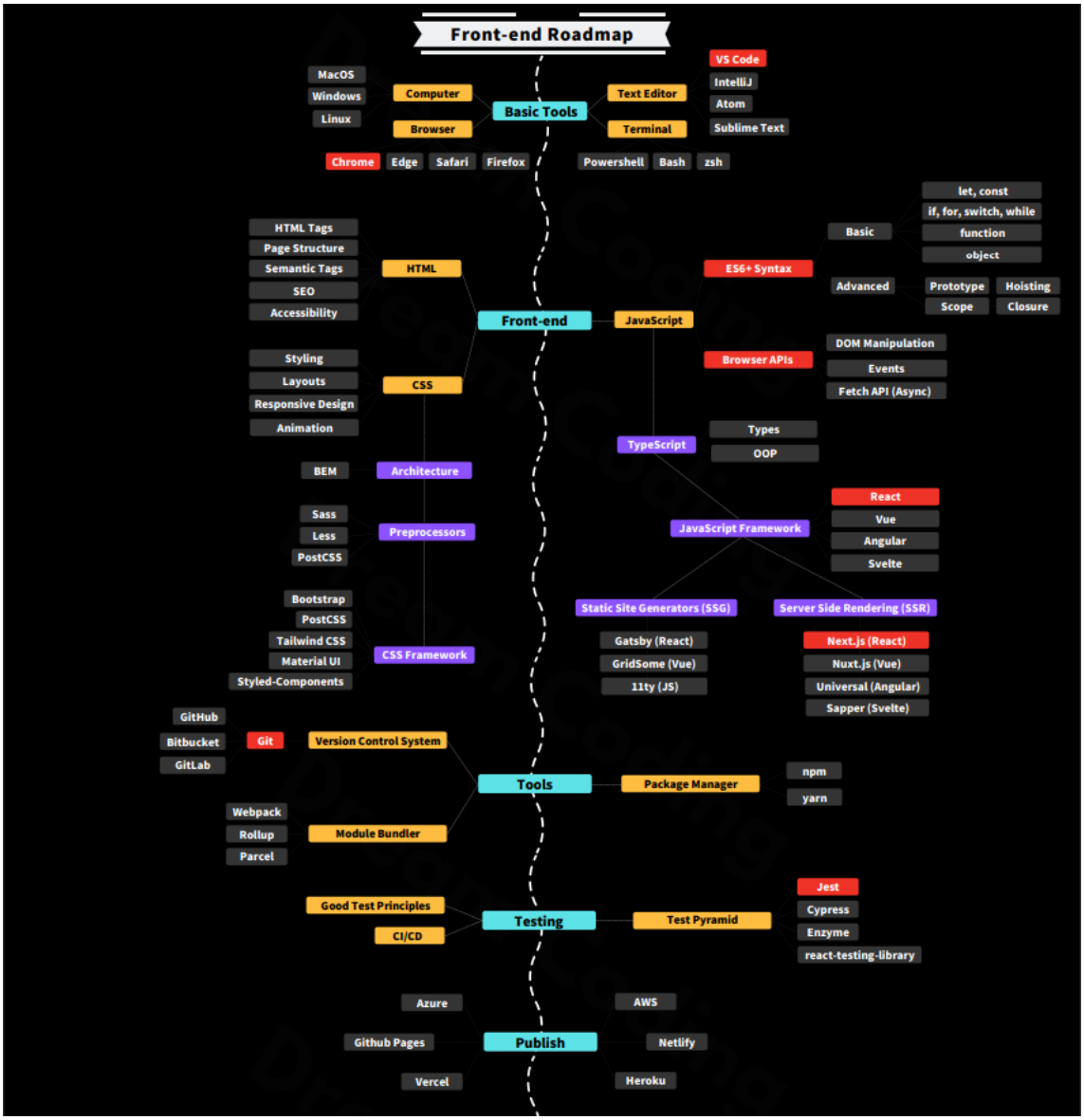
JavaScript / JQuery

작성일시	@2023년 4월 10일 오후 7:13
강의 번호	허지훈 강사님
유형	강의
자료	
복습	☐

[Front-end RoadMap]



[웹 컴바인 #3 : JavaScript]

- **console.log()**

==> 브라우저 검사도구(f12)에서 확인 가능한 영역

==> 콘솔창을 통해서 프로그램의 오류를 보거나 값을 테스트 한다

==> ;(세미콜론)은 명령이 끝나는 영역을 알려주는 마침표 역할을 한다

==> 자바스크립트는 매유 유한 언어라서 세미콜론 없이 실행 가능

==> 명령이 길어지면 구분이 어렵기 때문에 찍는 것을 권장

- **주석의 종류**

1) // : 행단위 주석

2) /* 블럭단위 주석 */

- **변수**

1. 변수 ?

- Variable
- let, var 라는 키워드를 통해서 선언
- let 변수의 이름 = 변수에 담을 내용
- 변수의 이름은 id와 같다. 유니크한 값이어야 한다. 중복이 안된다

2. 변수 상세

- 변수에는 어떠한 값이든 넣을 수 있다
- 변수의 이름을 호출해서 사용할 수 있다
- 변수 생성자의 이름과 호출시의 이름이 같아야 한다

3. 키워드의 차이점

- **let, var, const** : 변수가 최초에 선언될 때에 한번만 사용되는 키워드
- **let, var** : 단순 변수 선언시에 사용되는 키워드

  이 후 변수의 값을 바꿀때에는 '변수이름 = 바꿀 내용'으로 재할당하여 사용가능하다
- **const** : 상수의 선언에 사용되는 키워드

  let, var와 다르게 고정이 필요한 값을 선언할 때 사용된다

  한번 선언하면 재할당 불가

4. 자바스크립트에서만 사용?

- 변수는 모든 프로그래밍 언어에서 사용되는 개념
- 언어별 선언 및 사용 방식의 차이

5. 변수의 선언 방법

- **var 변수이름** = 변수에 저장할 내용
- **let 변수이름** = 변수에 저장할 내용

※ 통상적으로 var보다 let을 더 많이 사용

※ var는 2016년 이전에 주로 사용 (let은 var의 단점 보완)

```
1. 변수선언

let americano = 4500;      // 초기 선언
let latte = 4500;
let capucino = 4500;
```

```
let cookie = 3000;

americano = 4500;          // 재할당
※재할당 시 let을 사용하면 오류 발생

console.log(americano * 3 + cookie * 2);

2. 상수 선언

const CUP = 10000;
const cup = 5000;
console.log(CUP + americano);
console.log(cup + americano);
※대소문자 결과 다르게 출력
```

6. 변수와 상수의 차이점

- 변수는 값의 재할당 가능하나 상수는 불가능
- 상수는 초기 선언 시 무조건 값을 지정해주는 것이 좋다

7. 변수와 상수의 이름 규칙

- 시작은 영문 또는 언더바(_) 또는 $로 시작
- 두번째 부터 숫자 가능
- 대소문자 구분
- 고유로 지정된 키워드는 사용 불가능

7-1) 변수와 상수의 이름 관례

- 의미 없는 이름 사용하지 않기
- 너무 추상적인 이름은 좋지 않다
- 모든 변수의 이름은 **카멜케이스(camelCase)** 또는 **스네이크케이스(Snake_case)** 또는 **파스칼(VariableNameValue)**을 사용하기
- 상수의 이름은 대문자만 사용

- **자료형 (Date Type)**
- **변수 안에 저장할 수 있는 다양한 형태의 자료형**

1. String 타입

- 문자열, 따옴표안에 들어가있는 데이터
- 따옴표 안에 들어간 것들은 모두 문자열로 인식
- ex) "123" + 1 의 결과 값은 1231이 출력
- 큰따옴표("")와 작은따옴표('')를 구분하지 않는다. 무조건 문자열로 인식

2. Number 타입

- 숫자, 양수와 음수, 소수, 모두다 숫자 타입으로 들어간다
- 123 + 1 의 결과 값은 124가 출력

3. Boolean 타입

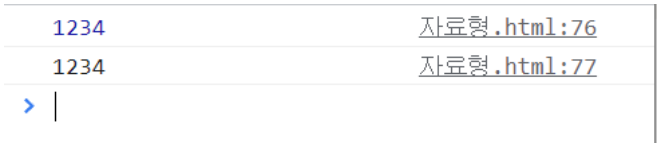
- 논리 연산에 쓰이는 데이터
- true(참) , false(거짓) 두개의 값으로 표현
- 불대수, 불타입, 블린형 등으로 불러진다
- 상수형 데이터의 가장 대표적인 데이터

4. 기타

- 배열 : [] 로 묶어두는 자료형
- 객체 : {} 로 묶어두는 자료형

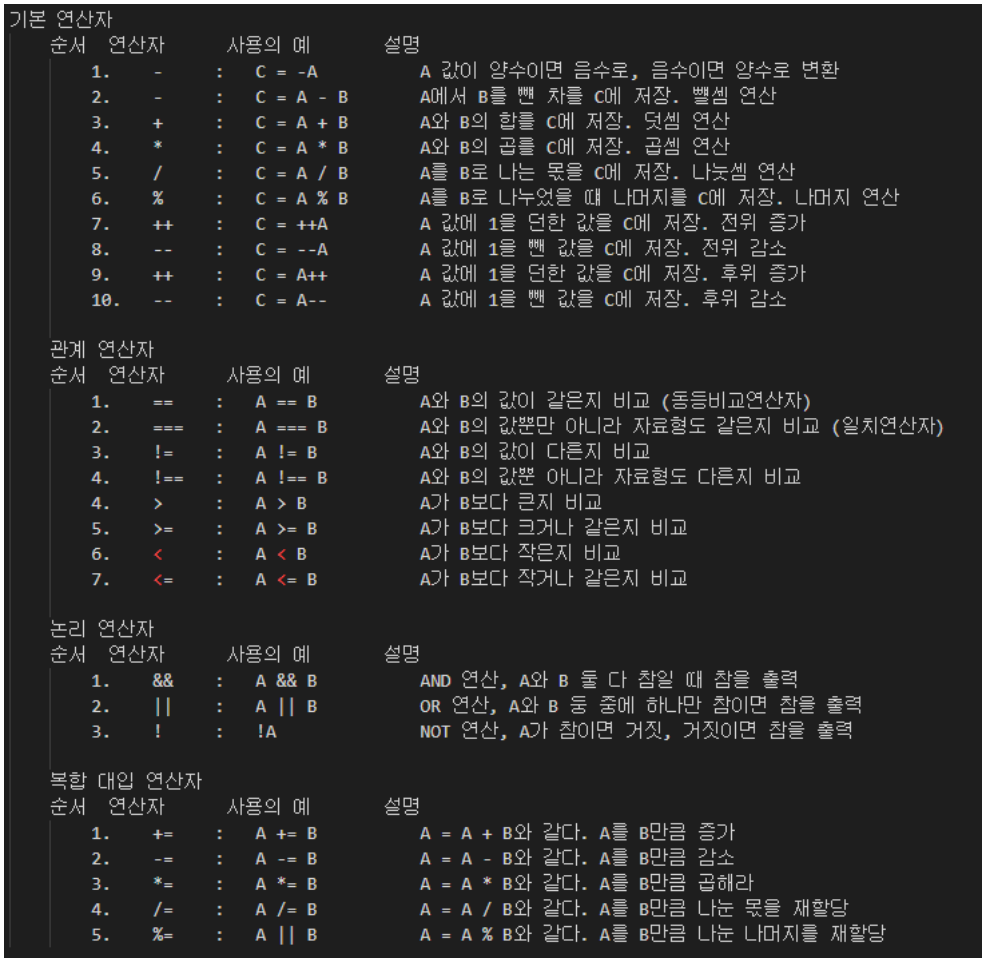
```
console.log(1234);
console.log("1234");
console.log(false);
console.log("1" + 1);
```

※ 문자열의 경우 검은색 , 숫자의 경우 파란색으로 출력



- 연산자(Operator)

```
let number = 10;
console.log(++number);

let a = 111;
let b = "111";
console.log(a == b);
console.log(a === b);      ※ a는 숫자, b는 문자열이라 false 결과 출력
```



- **호이스팅(Hoisting)**

1. 호이스팅 : 자바스크립트의 특이한 특징 ?

- 자바스크립트는 문서가 실행되면서 **코드를 실행하기 전에 변수와 함수를 먼저 메모리에 저장해두는 과정(단계)**
- 함수가 실행되기 전에 안에 있는 변수들을 범위의 최상단으로 올리는 개념이 호이스팅이다

2. let과 var의 차이

- 2015년도에 자바스크립트가 ES6 문법으로 업그레이드 되면서 생겨난 것이 let이다
- var의 문제점

```
var a = 1;
console.log(a);
※문제 없는 코드이다

console.log(a);
var a = 1;
console.log(a);
라고 입력되면 undefined라는 코드가 먼저 나오게 된다.
말이 안되는 코드 같지만, 콘솔창에 출력하기 전에 V8엔진이 선언된 변수 a를 먼저 읽어들이기 때문에
오류가 아니라 undefined(찾을 수 없음)라는 문구로 출력 된다.
자바스크립트에서 호이스팅을 할 때에 변수의 선언과 초기화를 같이 시켜버린다.
그리고서 할당은 나중에 실제 코드에서 변수가 선언되어질 때 할당하게 된다.

console.log(a);
a = 1;
var a;
console.log(a);
이것 또한 말도 안되는 코드 같지만, 자바스크립트는 이를 이해한다

결론 1) 자바스크립트는 매우 유한 언어이다. 허용범위가 크다.
```

```
        var의 문제점이 더 있다면 지역변수와 전역변수의 개념이 확실하지 않다는 점이다.
        var의 문제점 마지막으로는 중복선언을 허용한다는 점이 있다.

※ 전역변수 : 블럭 밖에서 아무것도 없이 그냥 선언되어진 변수, 어디에서나 접근 할 수 있고, 사용할 수 있는 변수
※ 지역변수 : 함수나 기타 등등 블럭 범위의 안에서 선언되어진 변수, 해당되는 지역 안에서만 쓸 수 있는 변수
```

- 위에서 언급된 var의 문제점들이 엄청나기 때문에 let을 만들게 되었다
- 실제 2015년 이후 var보다 let을 사용해 줄 것을 적극적으로 권장하고 있다
- 논리적이고 상식적으로 작성된 코드라면 var를 let으로 바꾸었을 때 아무런 문제가 없어야 한다
- let은 Temporal Death Zone(TDZ)라는 개념을 만들었다
- 호이스팅이 이루어진 후에 선언되어진 변수라는 것은 알고 있지만, 실제 변수가 선언되어지기 전이라면 오류를 일으킨다
- 호출되기 전이니까 **'호출되어지면 사용하겠다'** 라는 느낌이라고 이해하면 좋다

3.결론

- **var를 쓰지말고 let을 사용**

※ 2일차 과제 풀이

```
// 문제 1) 나이 계산기
const BIRTH_YEAR = 1985;
let year = 2023;
let age = year - BIRTH_YEAR + 1;

console.log(age);
// 한줄 공백
console.log('')

// 문제 2) 음료 칼로리 계산기
let expresso = 10;
let milk = 170;
let chocolate_syrup = 50;
let whipped_cream = 60;

※ 값이 변하지 않을 것 같은 변수는 상수로 선언
const AMERICANO = expresso;
const LATTE = expresso;
const MOCCA = expresso + chocolate_syrup + milk;
const WHIPPED_MOCCA = expresso + chocolate_syrup + milk + whipped_cream;

console.log(AMERICANO);
console.log(LATTE);
console.log(MOCCA);
console.log(WHIPPED_MOCCA);
```

- **배열**

1.배열이란 ?

- 변수에 담을 수 있는 여러가지 데이터 중 하나
- 문자, 숫자, 블린 외 하나의 자료형

2.배열의 탄생배경

- 변수에는 일반적인 데이터를 하나씩만 담을 수 있게 되어 있다
- 하지만 배열에는 여러가지의 데이터들을 배열이라는 하나의 이름으로 담을 수 있다

3.선언

```
let fruit = ['banana' , 'apple' , 'grape' , 'mango'];
```

- 배열도 숫자나 문자처럼 일종의 자료형이므로 변수에 할당한다

```
<script>
    let fruit1 = 'banana';
    let fruit2 = 'apple';
    let fruit3 = 'grape';
    let fruit4 = 'mango';

    let fruit = ['banana' , 'apple' , 'grape' , 'mango'];

    console.log(fruit);
    console.log(fruit[0]);  // 배열값의 참조

    fruit[1] = 'cherry'     // 배열값의 수정
    console.log(fruit[3]);
</script>
```

```
▼(4) ['banana', 'apple', 'grape', 'mango']
   0: "banana"
   1: "cherry"
   2: "grape"
   3: "mango"
   length: 4
  ▶[[Prototype]]: Array(0)
banana                                  05_배열.html:36
mango                                   05_배열.html:39
>
```

※ 3일차 과제 풀이

```
<let animals= ["Aardvark","Albatross","Alligator","Alpaca","Ant","Ape","Armadillo","Donkey","Baboon","Badger","Barracuda","Bat","Bear"
               "Clam","Cobra","Cockroach","Cod","Cormorant","Dugong","Dunlin","Eagle","Echidna","Eel","Eland","Elephant","Elk","Emu","
               "Heron","Herring","Hippopotamus","Hornet","Horse","Kangaroo","Kingfisher","Koala","Kookabura","Moose","Narwhal","Newt",
               "Red deer","Sandpiper","Sardine","Sparrow","Spider","Spoonbill","Squid","Squirrel","Starling","Stingray","Tiger","Toad"

// 문제 1) 어레이에 마지막 아이템 “Zebra” 제거하기
animals.pop();
console.log(animals);

// 문제 2) 주어진 어레이에 “Dog” 추가하기
animals.push('Dog');
console.log(animals);

// 문제 3) 주어진 어레이에 “Mosquito”,“Mouse”,“Mule” 추가하기
animals.push('Mosquito','Mouse','Mule' );
console.log(animals);

// 문제 4) 해당 어레이에는 "Human"이 있는가?
console.log(animals.includes('Human'));

// 문제 5) 해당 어레이에는 “Cat” 이 있는가?
console.log(animals.includes('Cat'));

// 문제 6) "Red deer"을 "Deer"로 바꾸시오
// console.log(animals[77]);
console.log(animals.indexOf('Red deer')); // 'Red deer' 인덱스 위치(값) 확인
animals[animals.indexOf('Red deer')] = "Deer"
// animals[77] = 'Deer'
console.log(animals[77]);

// 문제 7) "Spider"부터 3개의 아이템을 기존 어레이에서 제거하시오
// console.log(animals[81], animals[82], animals[83], animals[84]);
animals.splice(animals.indexOf("Spider"),3);
// animals.splice(81, 3);
console.log(animals);

// 문제 8) "Tiger"이후의 값을 제거하시오
// console.log(animals[84], animals[85], animals[86], animals[87], animals[88], animals[89], animals[90], animals[91], animals[92], an
animals.splice(animals.indexOf("Spider"));
console.log(animals);
// animals.splice(84);
// console.log(animals[84], animals[85], animals[86], animals[87], animals[88], animals[89], animals[90], animals[91], animals[92], an

// 문제 9) "B"로 시작되는 아이템인 "Baboon"부터 "Bison"까지 가져와 새로운 어레이에 저장하시오
// console.log(animals[8],animals[9],animals[10],animals[11],animals[12],animals[13],animals[14],animals[15]);
// let new_animals = [animals[8],animals[9],animals[10],animals[11],animals[12],animals[13],animals[14],animals[15]];
// console.log(new_animals);
let newList = animals.slice(animals.indexOf("Baboon"), animals.indexOf("Bison") + 1);
// +1을 해주어야 마지막 요소인 'Bison' 까지 포함하는 것
console.log(newList)
```

- **객체(Object)**

1. 객체가 필요한 이유 ?

- 하나의 데이터에 많은 정보가 필요한 경우 객체라는 것으로 해결이 가능하다
- 사실 하나의 정보로 이루어진 데이터는 없다

- '나' 라는 사람에 대해 표현하기 위해서도 이름, 나이, 사는 곳 등등 많은 정보가 필요하다
- 객체는 관련있는 정보들을 묶어서 하나의 데이터로 표현하기 위해서 탄생하게 되었다

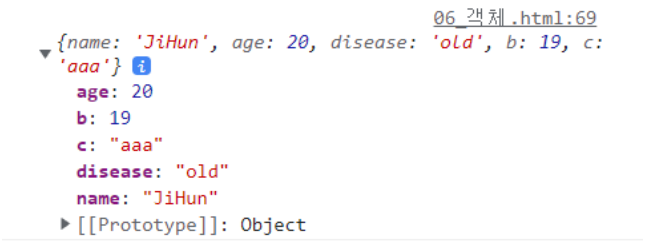
2. 객체의 생성방법

```
let patient = {
                name : "JiHun",
                age : 20,
                disease : 'old'
              }
```

- 위의 코드는 patient라는 객체를 생성하고, 거기에 name, age, disease 라는 정보를 입력하고 있다
- 이처럼 관련된 **정보들을 하나로 묶어서 데이터로 저장해주는게 바로 객체라는 개념**이라고 할 수 있다
- 배열 [] 과는 다른 개념

```
let 객체이름 = {
                 key : value,
                 key : value,
                 key : value
               }
```

```
let patient = {
                name: "JiHun",
                age : 20,
                disease : 'old',
                b : 19,
                c : "aaa",
                c : "ccc"

              }
console.log(patient);
```



※ 코드에 작성된 우선순위에 관계 없이 알파벳 순서로 정렬되어 출력된다

3.객체 활용 방법

- 2번에서 생성한 객체를 활용해서 그 안에 들어있는 정보들 중 한가지만 보고싶다면 직접 접근하는 방법이 있다
- 객체이름.key의 양식으로 사용되어진다
- key와 value 또는 키와 키 값의 이름으로 사용되어지는 객체는 흡사 사전과 같은 양식으로 이루어져있다
- 배열과 혼동할 수 있지만, 배열의 방식으로 이 객체의 key 값을 불러오는 방법은 다음과 같다

=> **객체이름['key'] == 객체이름.키**

- 객체의 내용을 수정하거나, 참조하는 방법은 배열과 유사한 방식으로 사용되어진다

=> **객체이름.key = 바꿀 value의 내용 or 객체이름['key'] = 바꿀 value의 내용**

- 객체로 만든 정보들을 배열에 넣을 수도 있는데, 다음과 같은 방식으로 사용되어진다

=> let 배열이름 = [{name:"JiHun", age:20}, {name:"JiSu" , age=19}, {name:"JiMin" , age=18}]

※ 여기서 배열에 저장된 첫번째 정보만 보려면?

=> **배열이름[0]** 이라고 하면 가능하다

※ 첫번째 정보에서 name 값만 보고 싶다면?

=> 배열이름[0].name or 배열이름[0]['name']

- 객체에는 단순한 값(데이터) 뿐만 아니라 함수(기능)도 넣을 수 있다
- **활용 예시**

```
let patient = {
                name: "JiHun",
                age : 20,
                disease : 'old'
              }
console.log(patient.name);
console.log(patient['name']);       // 배열처럼 사용
patient.name = "JiSu";              // 값 수정 #1
// patient['name'] = "JiSu";           값 수정 #2

※ 배열 안에 객체 선언 #1
let patientList = [{name: "JiHun", age : 20,}, {name: "JiSu", age : 19}, {name: "JiMin" , age : 18}];
console.log(patientList);

※ 객체 안에 배열 선언
let test1 = {
              name:["JiHun" , "JiSu" , "JiMin" , "JiYoung"],
              age:[20, 19, 18, 17]
            }
console.log(test1['name'][2])
// test1.name[2] == test1['name'][2]
// => test1의 이름 중 2번째 값(JiMin) 호출

※ 배열 안에 객체 선언 #2
let test2 = [{name:"MinSu", age:20},{name:"MinJi", age:19},{name:"MinYoung", age:18}, {name:"MinSik" , age:17}];
console.log(test2[3]['name']);
// test2[3]['name'] == test2[3].name;
// => test2의 3번째 값의 이름값(MinSik) 호출
```

4.주의 사항

- 배열(Array)과는 다른 존재이므로 혼동해서는 안된다
- 배열과 같은 방식으로 사용할 수 있다
- 객체 지향성 프로그래밍(Object Oriented Programming)

=> 위에서 언급한 객체라는 것 때문에 '객체'? 지향형 프로그램이리라는 것인가 오해할 수 있지만, 위에서 언급한 객체는 자료형으로서의 객체 인 것이고, 객체지향형 프로그램에서 말하는 개념과는 다르다

※ JavaScript는 프로토타입 객체지향 언어(Prototype Object Oriented Language)이다

- **조건문**

1. 조건문 if, if ~ else

- 영어 if의 뜻과 거의 흡사하다. '만약 ~라면 ~하겠다'
- if가 필요한 이유는 여러가지의 케이스마다 실행하고자 하는 내용을 달리하기 위해서 필요한 문법이다

2. if문의 사용

```
if(조건){
            조건이 참(true)이라면 실행할 코드
        }
```

- 조건이 true 라면 중괄호 안으로 들어와 코드를 실행한다. 이 중괄호 영역을 if블럭 또는 if절 이라고 한다
- 조건이 참(true)인지 거짓(false)인지에 따라서 실행이 될 수도 있고, 아닐 수도 있다

```
if(조건){
            조건이 참(true)일 경우 실행할 코드
        } else {
            조건이 거짓(false)일 경우 실행할 코드
        }
```

- 조건이 true 라면 if 다음의 중괄호 안으로 들어와서 코드를 실행하고, false 라면 else 다음의 중괄호 안으로 들어와 코드를 실행한다
- if 다음의 중괄호 영역을 if절 또는 if블럭이라고 하고, else 다음의 중괄호 영역을 else절 또는 else블럭 이라고 한다
- 조건이 참(true)이라면 if절을, 거짓(false)라면 else절을 실행한다. 즉, 둘 중 하나는 반드시 실행한다

```
if(조건1){
            조건 1이 참(true)이라면 실행할 코드
        } else if(조건2){
            조건 2가 참(true)이라면 실행할 코드
        } else {
            조건1, 조건2가 모두 거짓(false)이라면 실행할 코드
        }
```

- 조건 1이 true 라면 첫번째 중괄호 안으로 들어가서 코드를 실행하고, false 라면 다음 조건 2로 넘어간다
- 조건 2가 true 라면 두번째 중괄호 안으로 들어가서 코드를 실행하고, false 라면 넘어간다
- 조건 1, 2가 모두 false 라면 else 다음의 중괄호로 들어가 실행한다

3. 조건문의 중첩사용

```
if(조건1){
            if(조건1-1){
                조건 1과 1-1이 모두 참(true)일 때 실행할 코드
            } else {
                조건 1은 참(true), 조건 1-1은 거짓(false)일 때 실행할 코드
            }
        } else if(조건2){
            조건 2가 참(true)일 때 실행할 코드들
        } else {
            if(조건3){
                조건1, 조건2가 모두 거짓(false), 조건3은 참(true)일 때 실행할 코드
            } else {
                조건1, 조건2, 조건3이 모두 거짓(false)일 때 실행할 코드
            }
        }
```

- 조건문 안에 다른 조건문을 넣어서 여러가지 조건을 충족할 때 실행할 코드를 작성할 수 있다

※ 4일차 과제풀이

```
// 문제 1) 유저가 입력하는 숫자가 0인지, 음수인지 양수인지 판단하는 프로그램을 만들어보시오.

let num = 8;
if(num > 5){
    if(num < 10){
       console.log(num + "는 5보다는 크고 10보다는 작은 양수");
    }else {
       console.log(num + "는 10보다 큰 양수");
    }
}else if(num == 0){
    console.log(num + "는 0 입니다");
}else {
    if(num < 0){
        console.log(num + "는 음수입니다");
    }else {
        console.log(num + "는 숫자를 입력하세요")
```

```
    }
}

// 문제 2) 나는 대학교 교수다. 레포트 점수에 따라서 등급을 매기는 프로그램을 만들어보시오.

let score = 75;
let grade = '';                              // 등급에 대한 정보 저장용 변수

if(90 <= score && score <= 100){
    grade = "A"
} else if(80 <= score && score <= 89){
    grade = "B"
} else if(70 <= score && score <= 79){
    grade = "C"
} else if(60 <= score && score <= 69){
    grade = "D"
} else if(0 <= score && score <= 59){
    grade = "F"
} else {
    console.log("잘못된 범위의 점수입니다 ~!")
}
    console.log(grade);

// 문제 3) 한 지원자가 우리회사에 지원을 했다. 지원자가 사용가능한 스킬은 배열에 제공이 된다.
// JavaScript와 React 둘 다 할줄 안다면 "합격!" JavaScript와 React 둘 중 하나만 할 줄 안다면 "예비!"
// 둘 다 할 줄 모른다면 "탈락!"을 보여주는 프로그램을 짜보자.

let skills = ['HTML','CSS', 'JavaScript', 'React'];

if(skills.includes('JavaScript') && skills.includes('React')){
   console.log("합격");
}else if(skills.includes('JavaScript') || skills.includes('React')){
   console.log("예비");
}else if(skills != skills.includes('JavaScript') && skills != skills.includes('React')) {
   console.log("탈락!");
}else {
   console.log(skills);
}
```

- **유사 조건문**

=> 조건문과 비슷한 다른 방식들

=> 조건문 if, if ~ else, if ~ else 외 다른 표현

1. switch 문

- 문자가 주는 느낌대로 가 바꿀 것 같은 문법이다
- 조금 더 간결하고 의미가 명확해 보인다는 장점이 있음
- case 가 값으로 딱 정해진 경우에만 사용 가능
- 조건이 비교식일 경우 사용불가
- 범위를 표현하는데 있어서 불가능은 아니지만, 거의 불가능에 가깝고, 가능하다고 해도 매우 비효율적이다

```
let munu = 1;
if(menu == 1){
    console.log("물건 사기")
   } else if(menu == 2){
    console.log("잔고 확인")
   } else if(menu == 3){
    console.log("히스토리 확인")
   } else {
    console.log("홈으로 돌아가기")
   }

- 예를 들어 위와 같은 방식의 키오스크 프로그램이 있다고 하면 이를 switch 문으로 표현하면 다음과 같다

let menu = 1;
switch(menu){
     case 1:
            console.log("물건 사기")
            break
     case 2:
            console.log("잔고 확인")
            break
     case 3:
```

```
            console.log("히스토리 확인")
            break
     case 4:
            console.log("홈으로 돌아가기")

}
```

- 이와 같이 if문을 대체하여 사용할 수 있는 것이 switch 문이다
- default는 else처럼 매칭되는 case가 없을 때 실행된다
- case 마다 break를 넘겨 주는 이유는 그렇게 하지 않으면 코드의 전체를 실행하기 때문이다
- switch문으로 구현할 수 있는 코드는 모두 if문을 사용해서 구현이 가능하지만, 그 반대의 경우는 아니다
- 결론 : 지금은 잘 쓰여지지 않아 코드의 형태만 알고 넘어가도 충분함

2. 삼항연산자(조건연산자)

- switch와 다르게 제법 많이 사용되어지는 방식이다. if else를 대체할 수 있다
- 연산자이므로 연산의 결과값을 반환하게 된다. (즉, 저장할 변수와 같은 것들이 있어야 한다는 이야기)

=> **let 변수이름 = 조건 ? 조건이 true 일 때 실행 : 조건이 false일 때 실행**

- 삼항연산자가 사용되어 지는 경우는 다음과 같다

=> **조건이 많지 않거나, return 이 하나만 코드에 있을 때, 반환값이 딱 하나만 코드에 있을 때**

- 매우 복잡한 방식의 if문에서는 사용이 불편하다

```
let menu = 8;
if(menu <= 3){
     console.log('범위 안의 숫자입니다')
} else {
     console.log('범위 밖의 숫자입니다')
}

- 위의 if ~ else 문을 딱 한줄에 줄여서 표현이 가능하다

let answer = menu <= 3 ? '범위 안의 숫자입니다' : '범위 밖의 숫자입니다’

- 3항 연산자의 중첩

let answer = nemu <= 3 ? '0초과 3이하의 숫자' : menu <= 0 ? '0이하의 숫자' : '그 외의 숫자'
```

- 이렇게 사용하는 것을 삼항연산자 또는 조건연산자 라고 한다

- **반복문(for, while, do-while)**

1. 개요

- 프로그래밍을 하는 근본적인 이유 : 사람이 하기 귀찮은 것들을 컴퓨터에게 시키는 것
- **반복문의 종류 : for문, while문, do ~while문**
- **반복의 필수 요소 3가지** : 반복을 위한 변수선언, 반복의 범위, 반복의 종료를 위한 증감연산
- **반복의 종류**

=> **범위가 정해진 경우의 반복 : for문**

=> **범위가 정해지지 않은 경우의 반복 : while문**

2. while문

- **while문의 기본 구조**

```
반복을 위한 변수의 선언
while (반복의 조건) {
    반복될 내용
```

```
    증감연산
}

- 위의 양식처럼 사용되어 진다
```

- **while문**은 보통 **정확한 반복의 횟수를 알 수 없는 경우에 사용**되어 진다
- 반복의 조건이 true라면 {} 안에 들어가 있는 내용을 실행하는 구조이다

3. for문

- **for문의 기본 구조**

```
for(초기식; 조건식; 증감식){
    반복 내용
}

- 위의 양식처럼 사용되어 진다
```

- **for문**은 보통 정확한 **반복의 횟수를 알 수 있는 경우에 사용**되어 진다
- 반복의 조건이나 범위의 정보만큼 증감연산 하면서 실행되어지는 구조이다

4.반복문의 중첩

- **for문의 중첩**

```
for(let i = 0; i < 3; i++){
    for(let j = 0; j < 3; j++){
        console.log("[", i, j, "]")
    }
}
```

- **while문의 중첩**

```
let i = 0
let j = 0
while(i < 3){
     j = 0
     while(j < 3){
           console.log("[", i, j, "]")
           j++
     }
     i++
}
```

- 중첩으로 사용되어지는 반복문에서 바깥에 있는 반복문은 큰 톱니바퀴, 안쪽에 있는 반복문은 작은 톱니바퀴이다
- 작은 톱니바퀴가 한바퀴를 다 돌아야, 큰 톱니바퀴를 한칸 이도하는 방식으로 큰 톱니바퀴가 다 돌때까지 반복하는 것이 기본
- 통상적으로 3중 이상으로 중첩시켜서 사용하는 경우는 잘 없다고 봐도 무관하다
- 반복문의 중첩이 많을수록 코드를 읽어들이고 연산하는 시간이 오래 걸린다
- 그래서 반복이 중첩이 되어지면 '과연 이게 최선인가 ?' 라는 고민을 한번쯤은 해봐야 한다

5. 반복문의 활용

- **반복문은 배열과 아주 궁합이 좋다**
- 사실 for문의 존재이유는 배열때문이라고해도 좋을만큼 궁합이 좋다
- 배열은 순서가 중요한 자료형이고, 반복문은 그 순서들에 하나씩 순차적으로 접근하는게 가능하기 때문이다
- 배열에는 length라는 데이터가 존재하는데 이는 길이값을 저장하고 있으므로, for문의 반복범위를 length 만큼 해주면 배열의 길이만큼 반복하게 된다

※ 5일차 과제풀이

```
// 문제 1) 구구단 (for문)
for(let dan = 2 ; dan < 10 ; dan++){
    console.log("<" + dan + "단 시작!!" + ">")
    for(let num = 1 ; num < 10 ; num++){
        console.log(dan + " " + "x" + " " + num + " " + "=" + " " + dan * num)
        }
        console.log("-----------------------------")
        }

        console.log("");

// 문제 2) 구구단 (while문)
let dan = 2;
let num = 1;

while(dan < 10){
     num = 1
     console.log("<" + dan + "단 시작!!" + ">") ※log를 중간중간 찍어보면서 코드의 흐름을 이해하는게 중요함
     while(num < 10){
          console.log(dan + " " + "x" + " " + num + " " + "=" + " " + dan * num)
          num++
     }
     dan++
     console.log("-----------------------------")
     }


// 문제 3) 8 'X' 8 '=' 64 의 형식이 아니라 8X8=64 의 형식으로 출력
// 쉼표(,) 연산자를 사용하면 별도의 객체로 반환
// => console.log(dan, "+" , num , "=" , dan * num) ==> 8 'x' 8 '=' 64

// 플러스(+) 연산자를 사용하면 문자열끼리 연결하여 새로운 문자열로 반환
// => console.log(dan + "+" + num + "=" + dan * num) ==> 8x8=64
```

※ while문과 do~while문의 차이

```
let a = 0;
        while(a < 3){
          console.log(a)
          a++
        }

do{
   console.log(a)
} while(a > 3)

※최초 1회는 무조건 실행하고 while문 실행
```

- **중지(break) / 생략(continue)**

※ 제어문을 보다 효율적으로 사용하는 방법

1. break : 중지

- 현재 내가 속해 있는 반복문 {} 를 탈출한다
- 전체적인 반복 횟수에 영향을 미친다

2. continue : 생략

- 조건검사 or 증감연산으로 이동한다
- 전체적인 반복 횟수에 영향을 미치지 않는다

3. 반복문 무한루프

- for문, while문의 경우 무한루프가 존재한다

- 말 그대로 중지되는 것 없이 계속 반복하게 되는 것을 말한다
- 반복의 조건이 true라면 무한루프라고 한다
- break 키워드와 함께 사용되어진다
- for문보다는 while에서 더 많이 사용되어진다
- 예외처리에서도 사용되어진다. (적극적인 예외처리)

```
1. break문 예시
- 1 ~ 99까지 정수 중 5와 7로 나눴을 때 나머지가 0인 정수 출력

let num = 1
let search = false

while(num < 100){
     if(((num % 5) == 0) && ((num % 7) == 0)){
          search = true
          break          ※ if반복문 내부 {}에 존재하더라도 반복문 전체의 {}를 벗어난다
     }
     num++
}
if(search){
   console.log("찾는 정수 : " + num)
}else {
   console.log("찾는 정수가 없습니다")
}

2. continue문 예시
- 1부터 100까지 1씩 증감하면서 5와 7로 나누었을 때 나머지가 0이 아닐 경우 continue를 실행하면서
  조건 검사로 돌아가게 되고, 나머지가 0일 경우 반복문을 빠져 나오면서 count를 증감하고 출력

let number = 0
let count = 0

while(number < 100){
    number++
    if(((number % 5) != 0) || ((number % 7) != 0)){
        continue
    }
    count++
    console.log(number)
}

console.log("count : " + count)

3. 무한루프 예시
- 1부터 if의 조건을 충족할 때 까지 계속(무한) 반복

let natNum = 1
while(true){
    if(((natNum % 6) == 0) && ((natNum % 14) == 0)){
         break
    }
    natNum++
}
console.log(natNum)
```

- **함수(function)**

1. 함수란?

- 여러줄의 코드(명령)을 하나로 묶어놓은 주머니 역할
- 반복되는 기능을 주머니 안에 넣어두고 필요할 때 주머니 이름만 불러내면 기능이 실행되어진다

2. 함수 정의

```
function 함수이름(){
            함수안에 들어갈 내용
        }
```

3. 함수의 사용

- 함수를 정의만 한다고 해서 기능들이 실행되어지지 않는다
- 함수를 정의하고, 기능들이 실행되게 하려면 함수를 호출해야한다

4. 함수 호출

```
함수이름()
```

5. 주의 사항

- 함수는 호이스팅의 대상이 되어진다
- 호출을 먼저하고 나중에 선언해도 문제없이 실행되어 진다

6. 함수의 종류

- 입력이 없고, 반환도 없는 함수
- 입력은 있고, 반환은 없는 함수
- 입력이 없고, 반환은 있는 함수
- 입력이 있고, 반환도 있는 함수

※ 위 4가지 종류 정도로 나눌 수 있고, **함수는 통상 기능상자의 개념**이라고 이해하면 좋다

```
1. 입력 X, 반환 X
function noInNoOut(){
    console.log("입력 없고, 반환 없고")
}

noInNoOut()
console.log("")

2. 입력 O, 반환 X
2-1) 매개변수가 1개일 경우
function oneInNoOut(number){
   console.log("함수에 입력된 값 : " + number)
}

oneInNoOut(99)

2-2) 매개변수가 2개일 경우
function twoInNoOut(num1, num2){
    console.log("함수에 입력된 두 값의 합 : " + (num1 + num2))
}
twoInNoOut(25, 30)
console.log("")

3. 입력 X, 반환 O
function noInOneOut(){
    return "입력 없고, 반환 있고"
}

console.log(noInOneOut())   // == console.log("입력 없고, 반환 있고")

4. 입력 O, 반환 O
function oneInOneOut(number){
    return number * number
}

console.log("함수에 입력된 값의 제곱 : " + oneInOneOut(7))
console.log("")
```

7. 함수 관련 용어

- **입력** : **매개변수(Parameter)와 인자(Agument)**
- **반환** : **리턴(return)**

- **return 키워드의 2가지 기능**

1.return 키워드 오른쪽에 오는 값을 함수 호출위치로 반환

2. 함수의 실행 종료

※ break와 return의 차이

- break의 경우 내가 속해있는 반복문 {}를 빠져나오지만, return의 경우 함수의 사용이 종료되기 때문에 함수 내부에 존재하는 반복문을 빠져나오는 개념이 아닌 실행 자체를 하지 않는다
- 단, 함수 내부에 다른 함수가 존재할 경우 return을 만난 함수만 사용이 종료된다

※ 함수 예시 코드

```
※ 햄버거 만들기

function makeHamburger(type){
    console.log("빵 깔기")
    console.log("상추 올리기")
    if(type == "불고기"){
        console.log("불고기 패티 올리기")
    } else if (type == "새우"){
        console.log("새우 패티 올리기")
    }
        console.log("토마토 올리기")
        console.log("치즈 올리기")
        console.log("빵 덮기")
    }

function serveCoke(){
    console.log("콜라통 선택")
    console.log("얼음 담기")
    console.log("콜라 담기")
}
function serveFrenchFries(){
    console.log("감튀통 선택")
    console.log("감튀 담기")
}
    makeHamburger("새우")
    serveCoke()
    serveFrenchFries()

※ 매개변수 여러개 사용
function make_hamburger(type, size, num){
     console.log("버거의 종류는 ?", type)
     console.log("빵 깔기")
     if(type == "불고기"){
         console.log("불고기 패티 올리기")
     } else if (type == "새우"){
         console.log("새우 패티 올리기")
     } else if (type == "치킨"){
         console.log("치킨 패티 올리기")
     }
     console.log("상추 올리기")
     console.log("토마토 올리기")
     console.log("치즈 올리기")
     console.log("빵 덮기")
     console.log(type+"버거", size+"사이즈", num+"개")
}

make_hamburger("치킨", "L", 3)

※ 다중 함수 이용하기
function makeSet(){
    makeHamburger("새우")
    return
    serveCoke()
    serveFrenchFries()
}

makeSet()
```

※ 6일차 과제 풀이

```
// 문제 1) 1부터 100까지 더하는 for문을 만들고 그 결과를 출력해보자.

1. for문
let sum = 0;

for(let num = 1 ; num < 101 ; num++){
    console.log(num + " " + "+" + " " + sum + " " + "=" + " " + (sum + num))
    sum += num
}
console.log("1부터 100까지의 합 : " + sum)

2. while문

let num1 = 1
let sum1 = 0
while(num1 < 101){
     sum1 += num1
     num1++
}
console.log(sum1)

// 문제 2) 1부터 100까지 홀수만 출력해보자.

1.
for(let num = 1 ; num < 101 ; num++){
    if((num % 2) == 1){
       console.log(num)
    }
}


2.
for(let num2 = 1 ; num2 < 101 ; num2 = num2 + 2){
    console.log(num2)
}

//  문제 3) 1부터 50까지 369의 결과를 프린트해보자.
for(let i = 1; i < 51 ; i++){
    let stringValue = i.toString()                    // 변수 stringValue에 대입되는 i(1~50) 값을 문자열로 변경
    let result = ""                                   // res에 공백 선언 (result.length = 0)
    for(let j = 0; j < stringValue.length; j++){
        if(stringValue[j] == "3" || stringValue[j] == "6" || stringValue[j] == "9"){
           result += "짝"                            // stringValue[0 ~ 1] 값이 "3", "6", "9" 일때 result에 "짝" 대입 ※33의 경우 0번째와 1
번째 문자열이 모두 "3"이기 때문에 "짝" + "짝" 결과 출력
        }
    console.log(result.length > 0 ? result : i)      // result = ""(공백)이 아니면 "짝" ""(공백)이면 i값을 넣어 출력 ※위 for문으로 result에 3,
6, 9가 들어간 경우 "짝" 을, 그 외 해당 문자열(i) 출력
    }
}

// 문제 4) 주어진 숫자가 소수이면 ture 아니면 false를 출력하는 프로그램을 짜세요.
let prime_num = 13
let isPrime = true

if(prime_num === 1){
   isPrime = false
}
for(let i = 2; i < prime_num; i++){
    if(prime_num % i ==0){
       isPrime = false
    }
}
if(isPrime){
   console.log("소수입니다")
}else {
   console.log("소수아닙니다")
}
```

※ Java 와 JavaScript 나눗셈(/) 연산 결과 비교

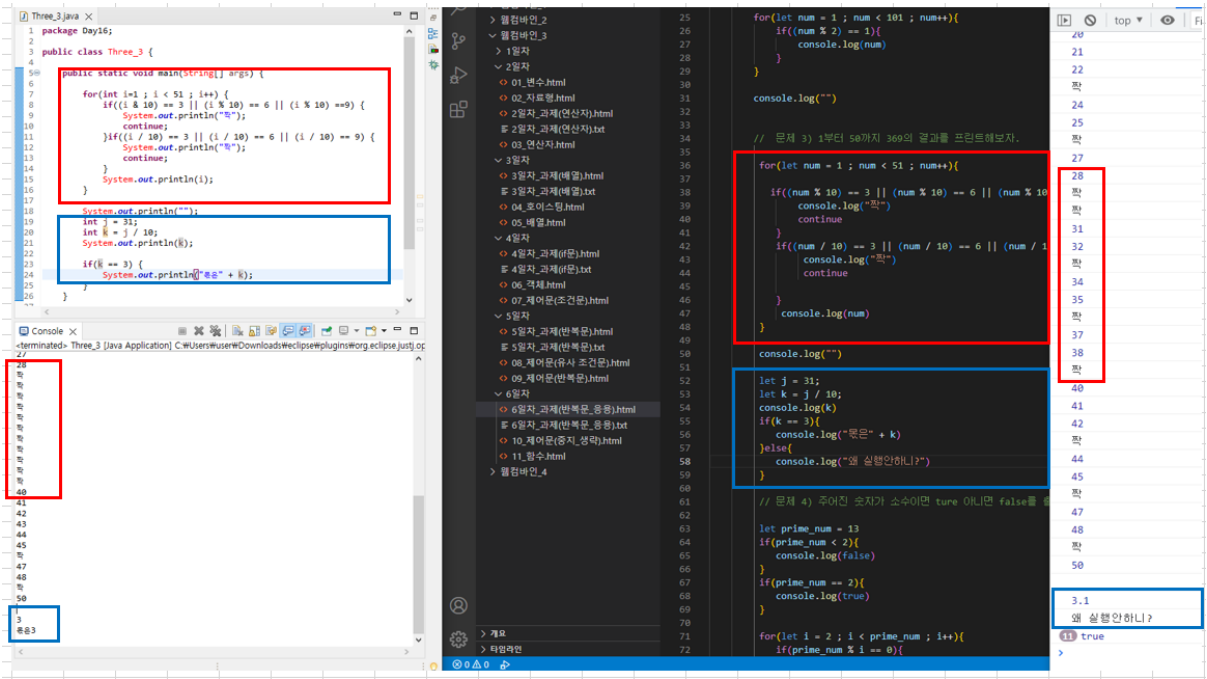
```
1. 369게임 알고리즘 코드 비교
- 아래의 코드에서 결과값 비교 시 Java 코드의 경우 30 ~ 39까지 "짝"이라는 결과가 출력되지만, JavaScript의 경우 30 ~ 39중 1의 자리가 3, 6, 9인 숫자와 해당
 숫자를 10으로 나눴을 때 3으로 떨어지는 숫자 30만 "짝"이라는 결과로 출력되는 것으로 확인

<Java>
public static void main(String[] args) {

    for(int i=1 ; i < 51 ; i++) {
      if((i & 10) == 3 || (i % 10) == 6 || (i % 10) ==9) {
        System.out.println("짝");
```

```
        continue;
      }if((i / 10) == 3 || (i / 10) == 6 || (i / 10) == 9) {
        System.out.println("짝");
        continue;
      }
      System.out.println(i);
    }

<JavaScript>
for(let num = 1 ; num < 51 ; num++){

    if((num % 10) == 3 || (num % 10) == 6 || (num % 10) == 9){
        console.log("짝")
        continue
    }
    if((num / 10) == 3 || (num / 10) == 6 || (num / 10) == 9){
        console.log("짝")
        continue
    }
    console.log(num)
}
```

2. 단순 나눗셈 연산 및 if문을 통한 결과값 비교
- 아래의 코드에서 Java 코드의 경우 int형(정수)으로 선언된 k의 연산결과는 3(정수)이지만, JavaScript 코드의 경우 k의 연산결과가 3.1(실수) 형태로 출력되는 것으로 확인

- 각각의 k의 값으로 if문을 통한 조건 결과 출력 시 Java 코드의 경우 조건이 true(참)임으로 "몫은 3" 이라는 결과가 출력되지만, JavaScript의 경우 조건이 false(거짓)임으로 else가 없을 경우 아무것도 실행하지 않는 것으로 확인되었으며, else가 있을 경우 else 안의 내용 "왜 실행안하니?" 가 출력되는 것으로 확인 (※K == 3.1 로 비교시 true(참)로 "몫은 3.1"이란 결과 출력되는 것으로 확인)

```
<Java>
int j = 31;
int k = j / 10;
System.out.println(k);

if(k == 3) {
  System.out.println("몫은" + k);

<JavaScript>
let j = 31;
let k = j / 10;
console.log(k)

if(k == 3){
   console.log("몫은" + k)
}else{
   console.log("왜 실행안하니?")
}
```



※ 별탑 세우기

```
1. 왼쪽정렬

console.log("별탑")
let star = ''
for(let i = 1; i < 6; i++){
    star += '*'
    console.log(star)
}
```

※ Template literals

```
※ ``(백틱)을 활용한 템플릿 리터럴

let year = 2023
let month = 4
let day = 21

console.log("오늘은" + year + "년" + month + "월" + day + "일")
console.log(`오늘은 ${year}년, ${month}월, ${day}일`)
```

```
const city = "seoul";
const name = "jim"
const age = 21;


// 일반적인 문자열
console.log("저는 " + city + " 에 살고 있구요, 이름은 " + name + ", 나이는 " + age + " 살 입니다.");

// () 템플릿 리터럴
console.log(`저는 ${city} 에 살고 있구요, 이름은 ${name}, 나이는 ${age} 살 입니다.`);
```

※ 7일차 과제풀이

```
// 문제 1) meetAt 함수를 만들어주세요.

// (조건) 인자 3개
//  1) 첫번째 인자는 년도에 해당하는 숫자입니다.
//  2) 두번째 인자는 월에 해당하는 숫자입니다.
//  3) 세번째 인자는 일에 해당하는 숫자입니다.

// (결과)
//  1) 년도 인자만 받았을 경우 → "1234년" 과 같은 형식의 문자열을 리턴 해주세요.
//  2) 년도,월 인자를 받았을 경우 → 년도와 월을 조합해서 "1234년 5월" 과 같은 형식의 문자열을 리턴 해주세요.
//  3) 년도,월,일 인자를 전부 받았을 경우 → 년도,월,일을 조합해서 "1234/5/6" 과 같은 형식의 문자열을 리턴 해주세요.

1.
function meeAt(year, month, day){
    if(day){
        return year + "/" + month + "/" + day
    }else if(month){
        return year + "년" + month + "월"
    }else if(year){
        return year + "년"
    }
}

console.log(meeAt(2023))
console.log(meeAt(2023, 4))
console.log(meeAt(2023, 4 , 20))


2.
function meeAt(year, month, day){
     let todayYear = year
     let todayMonth = month
     let todayDate = day

     if(todayDate){
```

```
          return `${todayYear}/${todayMonth}/${todayDate}`
     }
     if(todayMonth){
          return `${todayYear}년 ${todayMonth}월`
     }
     if(todayYear){
          return `${todayYear}년`
     }

}
console.log(meeAt(2021))
console.log(meeAt(2021, 11))
console.log(meeAt(2021 , 4, 21))

※ 1번과 2번의 차이점은 1번의 경우 else가 존재하기 때문에 else의 조건까지 모두 검사 하는 반면 (※ else는 단독실행 불가)
   2번의 경우 조건에 부합하는 if문만 단독으로 실행하기 때문에 불필요한 검사 과정을 생략할 수 있다


// 문제 2) findSmallestElement 함수를 구현해 주세요.

// (조건) findSmallestElement 의 arr 인자는 숫자 값으로만 이루어진 배열입니다.
//        이용되는 배열
//        [100,200,3,0,2,1]

// (결과)
//  1) arr 의 값들 중 가장 작은 값을 리턴 해주세요.
//  2) 만일 arr 가 비어있으면 0을 리턴 해주세요.
//  3) 예를 들어, 다음과 같은 배열이 인자(input)으로 들어왔다면 0이 리턴 되어야 합니다.

function findSmallestElement(arr){
    let min = arr[0]                            // arr[0]번째 값이 저장되어있기 때문에 for문에서 1번째부터 검사함
    for(let i = 0 ; i <= arr.length; i++){
        if(min > arr[i]){
           min = arr[i]
        }
    }
    return min
}

let arr = [100,200,3,0,2,1]
console.log(findSmallestElement(arr))

// 문제 3) 돈을 매개변수로 받으면 몇개의 지폐와 동전이 필요한지 최소 개수를 계산해주는 함수를 만드시오

// (결과 예시)
// 제공받은 돈이 12300인 경우
// 50000 X 0
// 10000 X 1
// 5000  X 0
// 1000 X 2
// 500 X 0
// 100 X 3

1.
function calMoney(don){
    console.log("받은 돈 : " + don + "원")

    let oman = don / 50000
    let man = (don % 50000) / 10000
    let ochun = (don % 10000) / 5000
    let chun = (don % 5000) / 1000
    let obak = (don % 1000) / 500
    let bak = (don % 500) / 100

    console.log("5만원" + "X" + " " + Math.floor(oman))
    console.log("1만원" + "X" + " " + Math.floor(man))
    console.log("5천원" + "X" + " " + Math.floor(ochun))
    console.log("1천원" + "X" + " " + Math.floor(chun))
    console.log("5백원" + "X" + " " + Math.floor(obak))
    console.log("백원" + "X" + " " + Math.floor(bak))
}

calMoney(12300)

2.
let unit = [50000, 10000, 5000, 1000, 500, 100]
function changeMoney(money){
    for(let i = 0; i < unit.length; i++){          // 1) 제공받은 돈을 unit의 0번째 값부터 순차적으로 나눈 몫을 소수점을 버리고 num에 대입
        let num = Math.floor(money / unit[i])      // 2) 제공받은 금액에서 1)에서 대입한 num의 값과 unit의 0번째 값부터 순차적으로 곱하기 연산결과를 빼
        money = money - unit[i] * num              // ex) num = 12300 / unit = [50000, 10000, 5000, 1000, 500, 100]
        console.log(`${unit[i]} X ${num}`)         //     money = 12300 - (unit = [50000, 10000, 5000, 1000, 500, 100] * 0 ~ 5(unit[0]
    }
}

changeMoney(24600)
```

- **Git-Hub**

※git-hub vscode 연결 순서

1) 구글 → **github** 검색 → 회원 가입 및 이메일 계정 로그인

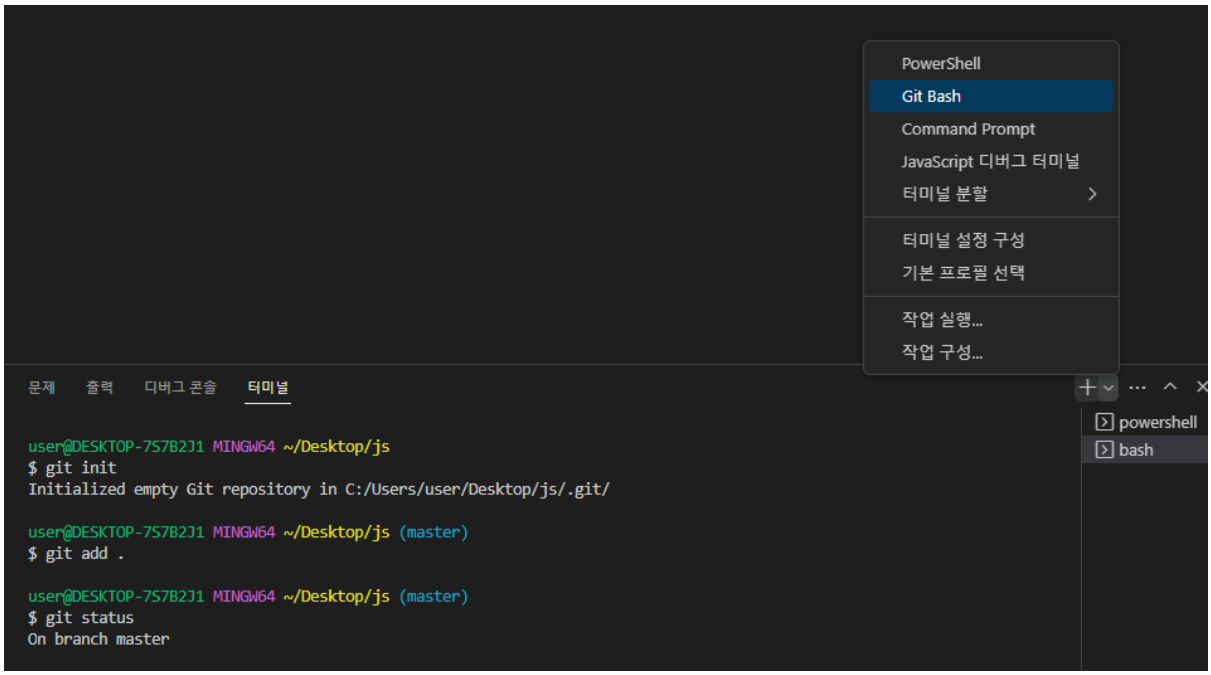
2) 'New' 버튼을 누르고 신규 **Repositories** 생성

3) 구글 → git 검색 → 다운로드 및 **git bash** 실행

4) 아래의 명령 순차적 실행 (이름 및 메일 주소 등록 상태 확인)

- git conifg —global user.name "BaeSangWon"
- git config —global user.email "sjqcl3310@naver.com
- git config —list

5) vs code → 터미널 → 새터미널 실행 후 우측 + 버튼 → Git Bash 추가



6) Git Bash 에서 아래의 명령 순차적 실행

- git init
- git add .
- git status
- git commit -m "First commit"
- git remote add origin https://github.com/BaeSangWon/test.git
- git remote -v
- git push origin master

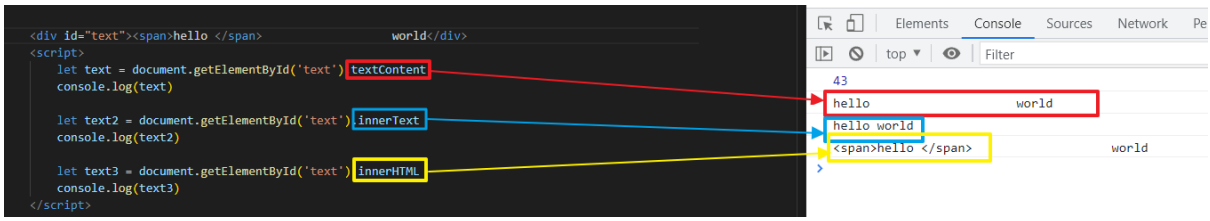
[Up & Down Game]

- **Math** : 자바스크립트에서 유용하게 사용되어지는 객체 중 하나. 수학적인 내용의 함수들을 가지고 있다
- **Math.random()** : 0 ~ 1 사이의 값을 반환(1은 미초함)
- **Math.floor()** : 소수점을 버린다
- **Math.ceil()** : 소수점을 올린다
- **Math.round()** : 소수점을 반올림한다

- **Math.max()** : 여러개의 값 중 제일 큰 값을 반환
- **Math.min()** : 여러개의 값 중 제일 작은 값을 반환

※ 노드의 속성값

- **textContent** : 노드(HTML문서)의 text 값을 반환
- **innerText** : 노드의 text 값을 반환, textContent와 비슷하지만 차이점이 있다
- **textContent** 는 모든 요소를 반환하는 반면 innerText는 사람이 읽을 수 있는 요소만 가져온다 (글자사이에 공백이 많다면 textContent는 그대로 가져오지만 innerText는 한칸만 가지고 온다)
- **innerHTML** : HTML 요소를 반환




```
<div id="text"><span>hello </span>                  world</div>


1. hello world 사이의 모든 공백을 포함하여 결과 출력
let text = document.getElementById('text').textContent
console.log(text)

2. hello world 사이의 한칸의 공백외 중복되는 공백은 제외하고 결과 출력
let text2 = document.getElementById('text').innerText
console.log(text2)

3. hello world 에 적용되어 있는 내부 태그 및 스타일 요소 모두 출력
let text3 = document.getElementById('text').innerHTML
console.log(text3)
```

[ProJect #1 : Up & Down 숫자 맞추기 게임]

※ 시작 화면(Cover Page)

```
<!DOCTYPE html>
<html lang="ko">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Up & Down Game</title>
    <script src="https://kit.fontawesome.com/8c36059e36.js" crossorigin="anonymous"></script>
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link href="https://fonts.googleapis.com/css2?family=Russo+One&display=swap" rel="stylesheet">
    <link href="https://fonts.googleapis.com/css2?family=Abril+Fatface&family=Russo+One&display=swap" rel="stylesheet">
    <link href="https://fonts.googleapis.com/css2?family=Abril+Fatface&family=Jua&family=Russo+One&display=swap" rel="stylesheet">
    <style>
        *{
            margin: 0;
            padding: 0;
            box-sizing: border-box;
        }

        a {
            text-decoration: none;
            color: initial;
            text-align: center;
            margin-top: 550px;
        }

        .splash{
            width: 100vw;
            height: 100vh;
            background-color: beige;
            position: absolute;
```

```
            top: 0;
            left: 0;
            display: flex;
            justify-content: center;
            align-items: center;
            font-size: 130px;
        }

        .splash span{
            position: relative;
            top: 20px;
            font-family: 'Abril Fatface', 'cursive';
            animation: bounce .3s ease infinite alternate;
            display: inline-block;
            text-shadow: 0 1px 0 #CCC,
                         0 2px 0 #CCC,
                         0 3px 0 #CCC,
                         0 4px 0 #CCC,
                         0 5px 0 #CCC,
                         0 6px 0 transparent,
                         0 7px 0 transparent,
                         0 8px 0 transparent,
                         0 9px 0 transparent,
                         0 10px 10px rgba(0, 0, 0, .4);
        }

        .splash .fa-comment{
            padding-left: 30px;
            margin-top: 550px;
        }

        .splash span:nth-child(2) {
            animation-delay: .1s;
        }

        .splash span:nth-child(3) {
            animation-delay: .2s;
        }

        .splash span:nth-child(4) {
            animation-delay: .3s;
        }
        .splash span:nth-child(5) {
            animation-delay: .4s;
        }
        .splash span:nth-child(6) {
            animation-delay: .5s;
        }

        .splash span:nth-child(7) {
            animation-delay: .6s;
        }

        .splash span:nth-child(8) {
            animation-delay: .7s;
        }

        .splash span:nth-child(9) {
            animation-delay: .8s;
        }

        .tab_hidden , .mo_hidden {
            display: none;
        }

        @keyframes bounce {
            100% {
                    top: -20px;
                    text-shadow: 0 1px 0 #CCC,
                                 0 2px 0 #CCC,
                                 0 3px 0 #CCC,
                                 0 4px 0 #CCC,
                                 0 5px 0 #CCC,
                                 0 6px 0 #CCC,
                                 0 7px 0 #CCC,
                                 0 8px 0 #CCC,
                                 0 9px 0 #CCC,
                                 0 50px 25px rgba(0, 0, 0, .2);

            }
        }

        .svg{
            width: 100%;
            height: 90vh;
            position: absolute;
            bottom: 50px;
```

```
            left: 50%;
            transform: translateX(-50%);
            margin-bottom: 100px;
        }

        .ears{
            animation: moveEars 1s linear infinite alternate;
        }

        /* 타원의 경우 cx="315" cy="90" 값을 기준점으로 지정해줘야 제자리에서 움직이는 애니메이션 적용 가능   */
        .ears:first-child{
            transform-origin: 315px 90px;
            /*
                transform-origin
                => 회전의 중심점(기준점) 지정
                => 기본값 : 50% 50%;
                => 사용단위 : %, 키워드, px

                [백분율과 대응 키워드]

                0% => left, top
                50% => center
                100% => right, bottom
            */
        }

        .ears:nth-child(2){
            transform-origin: 105px 90px;
        }

        .cheek{
            animation: cheek 1s linear infinite alternate;
        }

        @keyframes moveEars {
            to{
                transform: scale(1.1);
            }
        }

        @keyframes cheek{
            from{
                transform: translateY(0)
            }
            to{
                transform: translateY(-10px)
            }
        }

        /* tab */
        @media all and (min-width:768px) and (max-width:1023px){

            .fa-comment{
                display: none;
            }
            a{
                font-size: 100px;
            }
        }

        /* 모바일 */
        @media all and (max-width:767px){
            .mo_hidden{
                display: block;
            }
            .fa-comment{
                display: none;
            }
            a{
                font-size: 100px;
                margin-bottom: 50px;
            }
        }
    </style>
</head>
<body>

    <div class="splash">
        <a href="index.html">
            <svg class="svg" width="420" height="420" viewBox="0 0 420 420" fill="none" xmlns="http://www.w3.org/2000/svg">
                <!-- <rect width="420" height="420" fill="white"/> -->
                <circle class="ears" cx="315" cy="90" r="40" fill="#765049" stroke="black" stroke-width="10"/>
                <circle class="ears" cx="105" cy="90" r="40" fill="#765049" stroke="black" stroke-width="10"/>
                <circle cx="210" cy="210" r="150" fill="#B68278"/>
                <circle cx="210" cy="210" r="150" fill="#B68278"/>
                <circle cx="210" cy="210" r="150" fill="#B68278"/>
                <circle cx="210" cy="210" r="150" stroke="black" stroke-width="10"/>
```

```
                <mask id="path-4-inside-1_1_7" fill="white">
                    <path fill-rule="evenodd" clip-rule="evenodd" d="M186.693 238H233.522C231.866 228.918 231 219.56 231 210C231 15
7.078 257.522 110.351 298 82.3857C273.002 65.1152 242.682 55 210 55C178.793 55 149.74 64.2224 125.418 80.0905C164.531 108.228 190 1
54.14 190 206C190 216.97 188.86 227.674 186.693 238Z"/>
                </mask>
                <path fill-rule="evenodd" clip-rule="evenodd" d="M186.693 238H233.522C231.866 228.918 231 219.56 231 210C231 157.07
8 257.522 110.351 298 82.3857C273.002 65.1152 242.682 55 210 55C178.793 55 149.74 64.2224 125.418 80.0905C164.531 108.228 190 154.1
4 190 206C190 216.97 188.86 227.674 186.693 238Z" fill="#93665D"/>
                <path d="M186.693 238L176.906 235.946L174.376 248H186.693V238ZM233.522 238V248H245.512L243.36 236.205L233.522 238ZM
298 82.3857L303.684 90.6131L315.593 82.3857L303.684 74.1584L298 82.3857ZM125.418 80.0905L119.954 71.7153L107.734 79.6877L119.578 8
8.2082L125.418 80.0905ZM186.693 248H233.522V228H186.693V248ZM221 210C221 220.165 221.92 230.124 223.685 239.795L243.36 236.205C241.
811 227.713 241 218.955 241 210H221ZM292.316 74.1584C249.251 103.911 221 153.655 221 210H241C241 160.501 265.793 116.792 303.684 9
0.6131L292.316 74.1584ZM210 65C240.587 65 268.935 74.4598 292.316 90.6131L303.684 74.1584C277.069 55.7706 244.777 45 210 45V65ZM13
0.882 88.4657C153.629 73.6256 180.792 65 210 65V45C176.794 45 145.852 54.8193 119.954 71.7153L130.882 88.4657ZM200 206C200 150.786
 172.87 101.908 131.258 71.9728L119.578 88.2082C156.191 114.547 180 157.493 180 206H200ZM196.48 240.054C198.788 229.057 200 217.665
200 206H180C180 216.276 178.933 226.291 176.906 235.946L196.48 240.054Z" fill="black" mask="url(#path-4-inside-1_1_7)"/>
                <circle cx="210.5" cy="253.5" r="22.5" fill="#B68278" stroke="black" stroke-width="10"/>
                <path d="M171 232.5C171 215.655 184.655 202 201.5 202H209V244C209 254.493 200.493 263 190 263C179.507 263 171 254.4
93 171 244V232.5Z" fill="#D9D9D9" stroke="black" stroke-width="10"/>
                <path d="M250 232.5C250 215.655 236.345 202 219.5 202H212V244C212 254.493 220.507 263 231 263C241.493 263 250 254.4
93 250 244V232.5Z" fill="#D9D9D9" stroke="black" stroke-width="10"/>
                <path d="M191.5 200.5H228.5L217 223H204L191.5 200.5Z" fill="#0B0B0B"/>
                <circle class="cheek" cx="290.5" cy="235.5" r="35.5" fill="url(#paint0_linear_1_7)"/>
                <circle class="cheek" cx="130.5" cy="238.5" r="35.5" fill="url(#paint1_linear_1_7)"/>
                <rect x="243" y="177" width="24" height="10" rx="5" fill="black"/>
                <rect x="154" y="177" width="24" height="10" rx="5" fill="black"/>
                <defs>
                <linearGradient id="paint0_linear_1_7" x1="290.5" y1="200" x2="290.5" y2="271" gradientUnits="userSpaceOnUse">
                    <stop stop-color="#FF0000" stop-opacity="0.37"/>
                    <stop offset="1" stop-color="#D9D9D9" stop-opacity="0"/>
                </linearGradient>
                    <linearGradient id="paint1_linear_1_7" x1="130.5" y1="203" x2="130.5" y2="274" gradientUnits="userSpaceOnUse">
                    <stop stop-color="#FF0000" stop-opacity="0.37"/>
                    <stop offset="1" stop-color="#D9D9D9" stop-opacity="0"/>
                </linearGradient>
                </defs>
            </svg>
            <span>G</span>
            <span>a</span>
            <span>m</span>
            <span>e</span>
            <br class="tab_hidden mo_hidden">
                <span>S</span>
                <span>t</span>
                <span>a</span>
                <span>r</span>
                <span>t</span>
        </a>
        <span>
            <i class="fa-solid fa-comment"></i>
        </span>
    </div>

</body>
</html>
```

※ index.html(main page)

```
<!DOCTYPE html>
<html lang="ko">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>게임화면</title>
    <link rel="stylesheet" href="style.css">
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link href="https://fonts.googleapis.com/css2?family=Russo+One&display=swap" rel="stylesheet">
    <link href="https://fonts.googleapis.com/css2?family=Abril+Fatface&family=Russo+One&display=swap" rel="stylesheet">
    <link href="https://fonts.googleapis.com/css2?family=Abril+Fatface&family=Jua&family=Russo+One&display=swap" rel="stylesheet">
</head>
<body>
    <div class="wrap">
        <div class="in-box">
            <h1>Up & Down 숫자맞추기 게임!</h1>
            <div id="result-area">결과 출력</div>
            <div id="chance-area">남은 찬스 : 7번</div>
            <input id="user-input" type="number" placeholder="1 에서 100까지 숫자 중 입력하세요">
            <div class="buttom_box">
                <button id="play-button">Go!</button>
                <button id="reset-button">Reset!</button>
                <a href="javascript:window.history.back()">
                    <button class="back-button">Back!</button>
```

```
                </a>
            </div>
        </div>
    </div>
    <script src="main.js"></script>

    <!-- <div id="text"><span>hello </span>                  world</div>
    <script>
        let text = document.getElementById('text').textContent
        console.log(text)

        let text2 = document.getElementById('text').innerText
        console.log(text2)

        let text3 = document.getElementById('text').innerHTML
        console.log(text3)
    </script> -->
</body>
</html>
```

※ main.js

```
let computerNumber = 0
let chances = 7
let gameOver = false

// 중복된 숫자 입력 방지를 위한 배열 선언
let userValueList = []

let playButton = document.getElementById('play-button')   // id(ById), class(ByClassName), tag(ByTagName)에 따라 다르게 사용된다
// let userInput = document.querySelector('#user-input')  // querySelector : 제공한 선택자와 일치하는 요소 반환 (id = # , class = .)
let userInput = document.getElementById('user-input')
let ressultArea = document.getElementById('result-area')
let resetButton = document.getElementById('reset-button')
let chanceArea = document.getElementById('chance-area')


// addEventListener('이벤트이름' , 이벤트 발생시 실행시킬 함수)
playButton.addEventListener('click', play)
// 함수의 이름만 전달하는 이유는 click 했을 때 이벤트가 발생하여야 하기 때문이다
// 함수() 로 적을 경우 클릭 여부와 상관없이 호출로 인해 바로 실행되어 진다
resetButton.addEventListener('click' , reset)

// 간단한 기능만 구현하는 함수의 경우 익명으로 선언 후 사용 가능
// 1회성으로 사용 시 활용!!
userInput.addEventListener('focus' , function(){
    userInput.value = ""
})


// 난수 생성 함수
function pickRandomNumber(){
    computerNumber = Math.floor(Math.random() * 100) + 1         // Math.random : 0 ~ 1사이의 숫자 랜덤하게 생성 , 1 ~ 100 안의 숫자 지정을 위해
    console.log(computerNumber)
}

// go 버튼 눌렀을 때 이벤트 발생하는 함수
function play(){
    // input 에 입력되는 값은 value에 저장된다
    const USER_VALUE = userInput.value      // value : input 태그가 가지고 있는 속성
    console.log(USER_VALUE)
    if(USER_VALUE < computerNumber){
        // console.log("up!!")
        ressultArea.textContent = "upup!!"
    }else if(USER_VALUE > computerNumber){
        // console.log("down!!")
        ressultArea.textContent = "down!!"
    }else {
        // console.log("딩동댕!!")
        ressultArea.textContent = "딩동댕!!"
        // 정답일 경우 go 버튼 비활성화
        gameOver = true
    }

    // 1부터 100사이 외 숫자 입력 시 기회가 감소되지 않고 종료
    if(USER_VALUE < 1 || USER_VALUE > 100){
        ressultArea.textContent = "1부터 100사이의 숫자를 입력해주세요"
        return
    }
    if(userValueList.includes(USER_VALUE)){
        ressultArea.textContent ="이미 입력된 숫자입니다"
        return
```

```
    }

    // go버튼이 실행될 때마다 기회가 1회씩 감소
    chances--

    // 입력한 값을 배열에 추가할 때 '입력된 숫자' 여부를 먼저 점검 후 해당 조건이 false 일때 그 값을 배열에 추가하는 로직
    userValueList.push(USER_VALUE)

    // console.log("남은 기회" + chances + "번")
    chanceArea.textContent = `남은 기회 : ${chances}번`
    if(chances == 0){
        gameOver = true
    }
    // gameOver == true라는것은 위 조건에서 chances == 0이라는 조건이 참이기 때문에
    if(gameOver){
        // disabled : input 태그의 속성
        playButton.disabled = true
    }
}

// rest 버튼 눌렀을 때 이벤트 발생하는 함수
function reset(){
    // reset 버튼을 누르면 난수 초기화
    pickRandomNumber()
    userInput.value = ''
    ressultArea.textContent = "결과출력"
    // console.log("리셋")
}
```

※ style.css

```
@charset "UTF-8";
@import url("https://fonts.googleapis.com/css2?family=Noto+Sans+KR&display=swap");

*{
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

button{
    -webkit-appearance: none;
    -moz-appearance: none;
    appearance: none;
}
/* reset */

.wrap{
    width: 100%;
    height: 100vh;
    background: url(img/bg_02.jpg) no-repeat center / cover;
    position: relative;
}

.in-box{
    max-width: 700px;
    width: 90%;
    height: 500px;
    background-color: skyblue;
    border-radius: 20px;
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: space-around;
}

.in-box > h1{
    width: 90%;
    height: 70px;
    background-color: beige;
    text-align: center;
    line-height: 70px;
    margin: 20px auto;
    border-radius: 20px;
    font-family: 'Jua', sans-serif;
    font-size: 45px;
}
```

```css
.in-box > .select-area , #result-area , #chance-area{
    font-size: 35px;
    font-weight: 500;
    font-family: 'Jua', sans-serif;
}

input{
    margin: 0;
    padding: 0.5rem 1rem;
    font-family: "Noto Sans KR", sans-serif;
    font-size: 1rem;
    font-weight: 400;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    width: auto;
    border: none;
    border-radius: 4px;
    box-shadow: 0 4px 6px -1px rgba(0, 0, 0, 0.1), 0 2px 4px -1px rgba(0, 0, 0, 0.06);
    transition: 0.5s;
}

input{
    width: 300px;
    height: 50px;
    font-family: 'Jua', sans-serif;
    font-size: 18px;
}

button{
    font-family: 'Abril Fatface', 'cursive';
    font-size: 25px;
    background-color: lightgreen;
    border: none;
    cursor: pointer;
    display: inline-block;
    font-size: 22px;
    margin: 15px;
    outline: none;
    padding: 12px 40px 10px;
    position: relative;
    text-transform: uppercase;
    font-weight: 700;
    transition: .5s;
    border-radius: 5px;
}

button::before, button::after{
    border-color: transparent;
    -webkit-transition: all 0.25s;
    transition: all 0.25s;
    border-style: solid;
    border-width: 0;
    content: "";
    height: 24px;
    position: absolute;
    width: 24px;
}

button::before{
    border-color: lightgreen;
    border-right-width: 3px;
    border-top-width: 3px;
    right: -5px;
    top: -5px;
}

button::after{
    border-bottom-width: 3px;
    border-color: lightgreen;
    border-left-width: 3px;
    bottom: -5px;
    left: -5px;
}

button:hover , button.hover {
    background-color: lightgreen;
    transform: scale(1.2);
}

button:hover::before, button.hover::before, button:hover::after, button.hover::after{
    width: 100%;
    height: 100%;
}
```

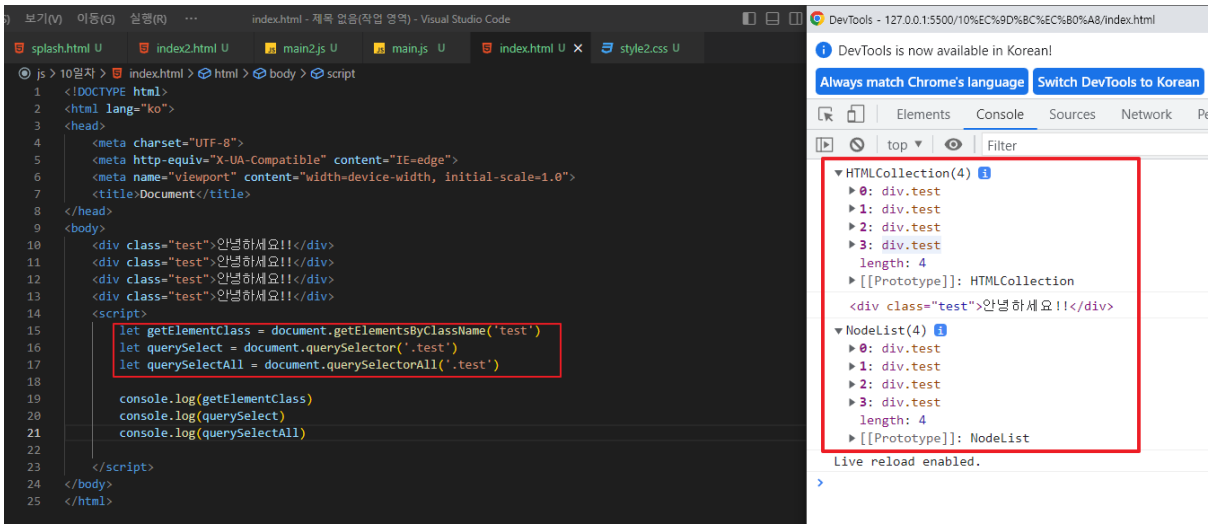
※ Up & Down 숫자게임 함수 및 이벤트

```
1. 난수 생성
- 게임 시작과 동시에 자동으로 1 ~ 100 중 랜덤 숫자 생성
function pickRandomNumber(){
    computerNumber = Math.floor(Math.random() * 100) + 1
    console.log(computerNumber)
}
pickRandomNumber()

2. Go 버튼 눌렀을 때 이벤트
- 사용자가 입력한 숫자와 컴퓨터가 생성한 난수(정답) 비교 후 조건에 따라 up, down, 딩동댕 출력
- 사용자가 입력한 숫자가 정답일 경우 Go 버튼 비활성화
- 1 ~ 100 사이 숫자를 입력하지 않았을 경우 기회가 감소되지 않고 "1에서 100사이 숫자 입력" 출력 후 종료
- 사용자가 입력한 숫자를 순차적으로 배열에 저장하여 중복 입력 방지
- 숫자 1회 입력할 때 마다 찬스 1회씩 감소
- 찬스가 0일 경우 게임 종료 및 Go 버튼 비활성화
function play(){

    const USER_VALUE = userInput.value
    console.log(USER_VALUE)
    if(USER_VALUE < computerNumber){
        ressultArea.textContent = "upup!!"
    }else if(USER_VALUE > computerNumber){
        ressultArea.textContent = "down!!"
    }else {
        ressultArea.textContent = "딩동댕!!"
        gameOver = true
    }

    if(USER_VALUE < 1 || USER_VALUE > 100){
        ressultArea.textContent = "1부터 100사이의 숫자를 입력해주세요"
        return
    }
    if(userValueList.includes(USER_VALUE)){
        ressultArea.textContent ="이미 입력된 숫자입니다"
        return
    }
    chances--
    userValueList.push(USER_VALUE)

    chanceArea.textContent = `남은 기회 : ${chances}번`
    if(chances == 0){
        gameOver = true
    }
    if(gameOver){
        // disabled : input 태그의 속성
        playButton.disabled = true
    }
}

3. 숫자 입력 후 Enter key를 눌렀을 때 Go 버튼을 누른것과 동일한 효과 주기
- 키보드 Enter Key 입력 시 입력창 내용 삭제
userInput.addEventListener('keyup', function(){
    if(window.event.keyCode == 13){
        play()
        userInput.value =''
    }
})

4. 숫자 입력 후 입력창에 마우스 커서 팝업 시
userInput.addEventListener('focus' , function(){
    userInput.value = ""
})

5. reset 버튼 눌렀을 때 이벤트
- 난수 초기화
- 사용자가 입력창에 입력한 숫자 삭제
- 남은 찬스 7회로 초기화
function reset(){
    pickRandomNumber()
    userInput.value = ''
    ressultArea.textContent = "결과출력"
    chances = 7
    chanceArea.textContent = `남은 기회 : ${chances}번`
}
```

※ getElementById 와 querySelector 차이점

- **getElementBy** : 버튼과 같이 단일 항목 하나를 가져올 때 사용, 동일항목이 존재하면 최상단에 1개만 출력
- **querySelector** : 여러개의 항목을 가져올 때 사용

```
<!DOCTYPE html>
<html lang="ko">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div class="test">안녕하세요!!</div>
    <div class="test">안녕하세요!!</div>
    <div class="test">안녕하세요!!</div>
    <div class="test">안녕하세요!!</div>
    <script>
        let getElementClass = document.getElementsByClassName('test')
        let querySelect = document.querySelector('.test')
        let querySelectAll = document.querySelectorAll('.test')

        console.log(getElementClass)
        console.log(querySelect)
        console.log(querySelectAll)
    </script>
</body>
</html>
```



[ProJect #2 : ToDoList 만들기]

1) 참고 : https://non-stop.tistory.com/400

※ **index.html #1**

```
<!DOCTYPE html>
<html lang="ko">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>ToDoList</title>
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link href="https://fonts.googleapis.com/css2?family=Abril+Fatface&family=Bruno+Ace&family=Jua&family=Russo+One&display=swap" r
el="stylesheet">
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="wrap">
        <div class="in-box">
            <div class="left_wrap">
                <div class="bgimg-box">
                    <div class="bg_img"></div>
                    <p class="name">Do It<br class="mo_hidden tab_hidden">Bae Sang Won</p>
                    <p class="tab_name">Bae Sang Won</p>
                </div>
                <div class="task-box">
                    <p class="today">Month Task</p>
                    <div>
```

```
                        <input style="zoom: 2.0" class="check" type="checkbox" id="work1">
                        <label for="work1"><span>WORK 01</span></label>
                        <button>삭제</button>
                    </div>
                    <div>
                        <input style="zoom: 2.0" class="check" type="checkbox" id="work2">
                        <label for="work2"><span>WORK 02</span></label>
                        <button>삭제</button>
                    </div>
                    <div>
                        <input style="zoom: 2.0" class="check" type="checkbox" id="work3">
                        <label for="work3"><span>WORK 03</span></label>
                        <button>삭제</button>
                    </div>
                </div>
            </div>
            <div class="right_wrap">
                <h1>To Do List</h1>
                <input id="toDo" type="text" placeholder="할일을 입력하세요">
                <button id="addButton">추가</button>
                <div id="toDoList">
                    <div>
                        <input type="checkbox" id="check-box">
                        <label for="check-box"><span>example</span></label>
                        <button>삭제</button>
                    </div>
                </div>
            </div>
        </div>
    </div>
    <script src="main.js"></script>
</body>
</html>
```

※ main.js #1

```
// DOMContentLoaded : HTML문서를 완전히 불러오고 분석했을 때 이벤트 발생
/*
    ※람다식 함수
    - 코드간략화를 위해 사용
    - 선언한 시점에 this를 확보하기 때문에, 한번 확보된 값은 고정으로 설정되어 변경되지 않는다
      즉, 다른 곳에서 함수를 호출하여도 처음에 호출되었을 때 설정된 값이 계속 출력된다
*/
document.addEventListener('DOMContentLoaded', () => {
    const toDo = document.querySelector('#toDo')
    const addButton = document.querySelector('#addButton')
    const toDoList = document.querySelector('#toDoList')

    addButton.addEventListener('click', (event) => {
        // createElement : 지정한 태그네임(div) 생성
        const item = document.createElement('div')
        const checkBox = document.createElement('input')
        // element.setAttribute( 'attributename(속성이름)', 'attributevalue(속성값)' )
        checkBox.setAttribute('type','checkbox')

        const text = document.createElement('span')
        text.textContent = toDo.value

        const removeButton = document.createElement('button')
        removeButton.textContent = '삭제'

        removeButton.addEventListener('click', (event) => {
            // currentTarget : 이벤트 핸들러가 부착된 것을 가리킨다
            // - event.target 는 부모로부터 이벤트가 위임되어 발생하는 자식의 위치, 내가 클릭한 자식 요소 반환
            // - event.currentTarget 는 이벤트가 부착된 부모의 위치 반환
            // parentNode : 부모노드, childNodes : 자식노드 리스트, nextSibling : 다음 형제노드, previousSibling : 이전 형제노드
            // removeChild : 부모에서 포함된 자식 노드가 존재할 경우 일치하는 ID, Class 등과 같은 속성을 통해 자식 노드를 제거
            event.currentTarget.parentNode.parentNode.removeChild(event.currentTarget.parentNode)
})

        checkBox.addEventListener('change', (event) => {
            if(checkBox.checked){
                // 체크박스에 체크될 경우 텍스트 중간에 선을 만든다
                text.style.textDecorationLine = "line-through"
            }else{
                text.style.textDecorationLine = "none"
            }
        })
        // append 메서드 : 노드객체(Node object)나 DOMString(text) 사용가능
        // append 메서드는 return 값 반환 x
        // appendChild 메서드는 오직 Node 객체만 받을 수 있으며 한번에 하나의 노드만 추가 가능
        /*
```

```
            ※append 와 appendChild 사용예시
            - append : item.append(div,span,p)
            - appendChild : item.appendChild(text)
        */
        item.appendChild(checkBox)
        item.appendChild(text)
        item.appendChild(removeButton)
        toDoList.appendChild(item)
        toDo.value = ''
        })
})

// 키보드 Enter Key를 눌렀을 때 add버튼 클릭과 동일한 이벤트 적용
// click() 함수 : 마우스 왼쪽버튼으로 선택 요소를 클릭하면 발생
// ※addButton의 이벤트 발생 조건이 'click'이기 때문에 입력창에 내용 입력 후 Enter을 입력하면 addButton을 마우스 왼쪽버튼으로 클릭 실행
// 즉, Enter key 입력 = addButton.addEventListener('click', event)와 같은 내용
toDo.addEventListener('keyup', function(){
    if(window.event.keyCode == 13){
        addButton.click()
    }
})
```

※ style.css #1

```
@charset "utf-8";

*{
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

.wrap{
    width: 100%;
    height: 100vh;
    background-color: #C5a4fa;
    position: relative;
}

.in-box{
    max-width: 900px;
    width: 90%;
    height: 700px;
    border-radius: 20px;
    position: absolute;
    top: 50%;
    left: 56%;
    transform: translate(-50%, -50%);
    display: flex;
    border: 3px solid #d070fb;

}

.left_wrap{
    width: 100%;
    background-color: #fff;
    border-radius: 20px 0 0 20px;
}

.bgimg-box{
    width: 100%;
    height: 150px;
    display: flex;
    justify-content: space-around;
    padding: 20px;
}

.bg_img{
    width: 100px;
    height: 100px;
    border-radius: 50%;
    border: 1px solid gray;
    background: url('img/mimoticon_4.png') no-repeat center / cover;
}

.name{
    width: calc(100% - 150px);
    height: 100px;
    font-size: 30px;
    text-align: center;
    line-height: 50px;
    font-weight: bold;
```

```
    font-family: 'Bruno Ace', 'cursive';
}

.tab_name{
    width: calc(100% - 70px);
    height: 100px;
    text-align: center;
    line-height: 100px;
    display: none;
    font-family: 'Bruno Ace', 'cursive';
    font-size: 23px;
}

.task-box{
    width: 100%;
    height: calc(100% - 200px);
}

.task-box::before{
    content: "";
    display: block;
    width: 80%;
    border-bottom: 3px solid #C5a4fa;
    margin: 0 auto;
}

.today{
    width: 100%;
    height: 50px;
    font-size: 25px;
    margin-top: 20px;
    font-weight: bold;
    font-family: 'Bruno Ace', 'cursive';
    text-align: center;
    line-height: 50px;
}


.task-box div{
    width: 80%;
    height: 50px;
    display: flex;
    justify-content: space-around;
    align-items: center;
    background-color: rgba(208, 112, 251, .7);
    color: white;
    border-radius: 20px;
    margin: 10px auto;
    line-height: 50px;
    font-size: 20px;
    font-family: 'Bruno Ace', 'cursive';
}

.task-box div button{
    width: 50px;
    height: 35px;
    font-size: 17px;
    font-family: 'Jua', sans-serif;
    border-radius: 5px;
    border: none;
    cursor: pointer;
}

.right_wrap{
    max-width: 700px;
    width: 90%;
    height: 100%;
    background-color: blueviolet;
    border-radius: 0 20px 20px 0px;
}

.right_wrap > h1{
    width: 90%;
    height: 80px;
    color: #fff;
    font-family: 'Bruno Ace', 'cursive';
    font-size: 50px;
    text-align: center;
    line-height: 80px;
    margin: 0 auto;
}

#toDo, #addButton{
    width: 80%;
    height: 50px;
    font-family: 'Jua', sans-serif;
    font-size: 20px;
```

```
    font-weight: 400;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    border: none;
    border-radius: 5px;
    box-shadow: 0 4px 6px -1px rgba(0, 0, 0, 0.1), 0 2px 4px -1px rgba(0, 0, 0, 0.06);
    transition: 0.5s;
    margin-left: 20px;
    cursor: pointer;
}

#addButton{
    width: 50px;
    margin: 0;
}

#toDoList{
    width: 100%;
    height: calc(100% - 150px);
}

#toDoList > div{
    width: 90%;
    height: 40px;
    font-family: 'Bruno Ace', 'cursive' , 'Jua', sans-serif;
    font-weight: 500;
    font-size: 25px;
    text-decoration: none;
    margin: 20px auto;
    text-transform: uppercase;
    color: #fff;
    display: flex;
    justify-content: space-between;
}

#toDoList > div > #check-box{
    padding: 10px;
}

#toDoList > div > input[type=checkbox]{
    transform: scale(2);
}

#toDoList > div > span{
    border-bottom: 3px solid salmon;
}

#toDoList > div > button{
    width: 50px;
    font-size: 17px;
    font-family: 'Jua', sans-serif;
    text-align: center;
    border-radius: 5px;
    border: none;
    cursor: pointer;
}

#toDoList > div > input, span, button{
    margin-left: 10px;
}


/* tab */
@media all and (min-width:768px) and (max-width:1023px){
   .name{
       display: none;
   }

   .tab_name{
        display: block;

   }

   .right_wrap > h1{
        font-size: 35px;
   }

   #toDo{
        width: 200px;
   }

}

/* 모바일 */
@media all and (max-width:767px){
   .name{
```

```
        display: none;
    }

    .task-box div{
        width: 90%;
    }

    .task-box div span{
      font-size: 13px;
}

    .right_wrap{
        width: 500px;
    }

    .right_wrap > h1{
        font-size: 30px;
    }

    #toDo{
        width: 130px;
    }
    #toDo::placeholder {
        font-size: 15px;
    }

    #toDoList > div{
        height: 30px;
        display: flex;
        justify-content: space-between
    }

    input[type=checkbox]{
        transform: scale(.7);
    }

    #toDoList > div > span{
        font-size: 18px;
        margin: 0;
    }

    #toDoList > div > input, span, button{
        margin-left: 3px;
     }

    #toDoList > div > button, .task-box div button{
        width: 30px;
        font-size: 12px;
    }

}
```

※ JavaScript 일반 함수와 람다 함수 비교

1) 일반함수

일반 함수 this 예제

```
param = 'global param';

function printParam(){
        console.log(this.param);
}

let object = {
        param: 'object param',
        func: printParam
}
let object2 = {
        param: 'object2 param',
        func: printParam
}

object.func();
object2.func();
```

결과

```
object param
object2 param
```

2) 람다식 함수

- 코드간략화를 위해 사용
- **선언한 시점에 this를 확보하기 때문에, 한번 확보된 값은 고정으로 설정되어 변경되지 않는다**
- 즉, **다른 곳에서 함수를 호출하여도 처음에 호출되었을 때 설정된 값이 계속 출력**된다
- 아래의 코드에서 처럼 **JavaScript에서 var나 let을 사용하지 않고 선언한 변수는 전역 변수됨**
- 함수를 호출한 오브젝트가 존재하지 않는 상태에서는 this는 전역변수를 가르킴
- printParam 함수 안의 this도 전역변수를 가르키게됨

람다식 함수 this 예제

```
param = 'global param';

let printParam = () => {
        console.log(this.param);
}

let object = {
        param: 'object param',
        func: printParam
}
let object2 = {
        param: 'object2 param',
        func: printParam
}

object.func();
object2.func();
```

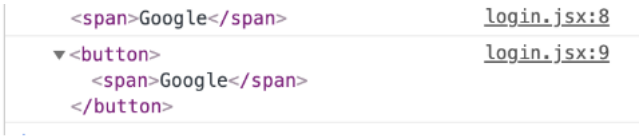
결과

```
global param
global param
```

※ target 과 currentTarget 비교

```
<!DOCTYPE html>
<html lang="ko">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>target 과 currentTarget 비교</title>
</head>
<body>
    <li>
        <button onClick={onLogin}>
            <span>Button</span>
        </button>
    </li>

    <script src="taget.js"></script>
</body>
</html>
```

```
const onLogin = (event) => {
  console.log(event.target);
    console.log(event.currentTarget);
};
```



정리

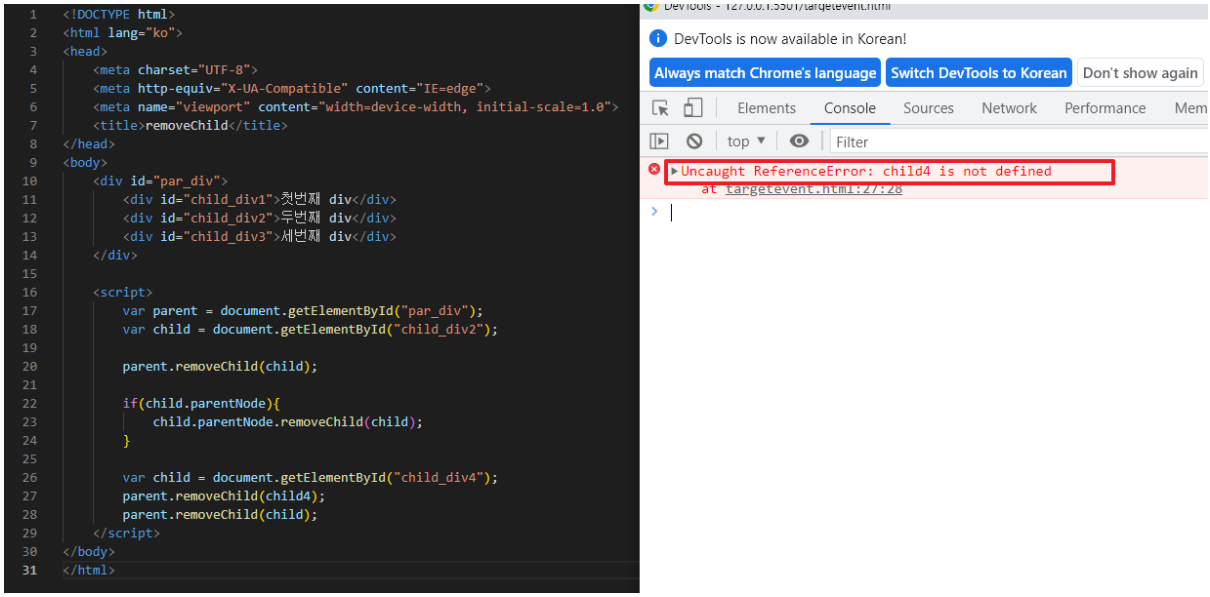
`event.target` 은 자식 요소인 `span` 을 리턴하고, `event.currentTarget` 은 부모 요소인 `button` 을 반환하는 것을 알 수 있다.

※ removeChild

- 부모에서 포함된 자식 노드가 존재할 경우 일치하는 ID, Class 등과 같은 속성을 통해 자식 노드를 제거

```
<!DOCTYPE html>
<html lang="ko">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>removeChild</title>
</head>
<body>
    <div id="par_div">
        <div id="child_div1">첫번째 div</div>
        <div id="child_div2">두번째 div</div>
        <div id="child_div3">세번째 div</div>
    </div>
</body>
</html>
```

```
var parent = document.getElementById("par_div");
var child = document.getElementById("child_div2");

parent.removeChild(child);

if(child.parentNode){
   child.parentNode.removeChild(child);
}

var child = document.getElementById("child_div4");
parent.removeChild(child4);
parent.removeChild(child);
```



- **click() 함수 : 마우스 왼쪽버튼으로 선택 요소를 클릭하면 발생**

2) SBS 아카데미 - 웹컴바인3 (with. 허지훈 강사님)

※ index.html #2

```
<!DOCTYPE html>
<html lang="ko">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>todolist</title>
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link href="https://fonts.googleapis.com/css2?family=Abril+Fatface&family=Bruno+Ace&family=Jua&family=Russo+One&display=swap" r
el="stylesheet">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-rbsA2VB
KQhggwzxH7pPCaAqO46MgnOM80zW1RWuH61DGLwZJEdK2Kadq2F9CUG65" crossorigin="anonymous">
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <!-- container : bootstrap 에서 제공하는 class name -->
    <div class="container">
        <h1>To Do List</h1>
        <section class="input-area">
            <input type="text" id="task-input" placeholder="할일을 입력하세요">
            <button id="add-button">
                <i class="fa-solid fa-square-plus"></i>
            </button>
        </section>
        <section class="task-area">
            <div class="task-tabs">
                <!-- js에서 가져와 사용하기 위해 ID로 부여 -->
                <div id="under-line"></div>
                <div id="all">모두</div>
                <div id="ongoing">진행중</div>
                <div id="done">완료</div>
            </div>
            <!-- 할일을 추가할 때 task class 내부 전체 요소가 같이 추가되도록 하기 -->
            <div id="task-board">
                <!-- <div class="task">
                    <div>치과가기</div>
                    <div>
                        <button>체크</button>
                        <button>삭제</button>
                    </div>
                </div> -->
            </div>

        </section>
    </div>
```

```
    <script src="https://kit.fontawesome.com/8c36059e36.js" crossorigin="anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-kenU1KFdBIe4zVF0s0
G1M5b4hcpxyD9F7jL+jjXkk+Q2h455rYXK/7HAuoJl+0I4" crossorigin="anonymous"></script>
    <script src="main.js"></script>
</body>
</html>
```

※ main.js #2

```
/*
[기본 Logic]
1. 유저가 값을 input에 입력한다.
2. + 버튼을 클릭하면 아이템이 더해진다. 할일이 추가된다.
3. 삭제 버튼을 클릭하면 할일이 삭제된다.
4. 체크 버튼을 누르면 할일이 끝나면서, 줄이 그어진다
5. 진행 중, 완료 탭을 누르면 언더바가 이동한다.
6. 완료 탭에는 완료 아이템만, 진행 중 탭에는 진행중인 아이템만 보여진다. 모두 탭을 누르면 전체 아이템을 보여준다
*/

let taskList = []

let taskInput = document.getElementById('task-input')
let addButton = document.getElementById('add-button')
let tabs = document.querySelectorAll('.task-tabs div')
console.log(tabs)

addButton.addEventListener('click' , addTask)

// 모두, 진행중, 완료 tab 기능 구현
for(let i = 1 ; i < tabs.length; i++){
    tabs[i].addEventListener('click' , function(event){filter(event)})
}

// 할일 추가 함수
// 1) input에 입력한 값을 가지고와 taskList 변수에 배열로 저장
function addTask(){
    // let taskContent = taskInput.value
    // task 객체에 input에 입력된 내용 + 진행중 or 완료 여부까지 포함된 값 저장
    let task = {
        id: randomIDGenerate(),             // 입력된 할일에 각각 유일한 ID값을 부여
        taskContent: taskInput.value,       // input에 입력된 내용을 저장하는 요소
        isComplete : false                  // 진행중 or 완료여부를 판단할 수 있는 요소
    }
    taskList.push(task)
    console.log(task)
    rander()
}

// 추가한 할일 출력 함수
// 1) for문 : taskList 배열에 길이값을 순차적으로 비교
// 2) if문 : 할일이 완료 상태라면 task-done을 호출하여 가운데 줄을 긋는 상태로 만든다
// 3) task-board의 전체 요소(태그포함)를 resultHTML에 저장
// 4) 체크 버튼 클릭 시 할일의 체크 여부 상태에 따라 "진행 중" "완료" 상태를 구분하는 이벤트 실행
function rander(){
    let resultHTML = ''
    for(let i = 0; i < taskList.length; i++){
        if(taskList[i].isComplete == true){
            // task-done class에 가운데 줄을 긋은 css 반영
            resultHTML += `
            <div class="task">
                <div class="task-done">${taskList[i].taskContent}</div>
                <div>
                    <button onclick="toggleComplete('${taskList[i].id}')">체크</button>
                    <button onclick="deleteTask('${taskList[i].id}')">삭제</button>
                </div>
            </div>`
        } else {
            //task 내부 요소 전부를 resultHTML에 누적시켜서 저장
            // - taskList 배열에 있는 객체 요소중 입력된 내용 요소만 resultHTML에 저장
            resultHTML += `
            <div class="task">
            <div>${taskList[i].taskContent}</div>
            <div>
                <button onclick="toggleComplete('${taskList[i].id}')">체크</button>
                <button onclick="deleteTask('${taskList[i].id}')">삭제</button>
            </div>
            </div>`
        }
        // button onclick : addEventListener을 대신해 HTML요소에 바로 동적 이벤트 적용 -> onclick="toggleComplete()"
    }
    // ${taskList[i].taskContent} : taskList 배열에 저장된 값중 내용 요소 , '${taskList[i].id : taskList 배열에 저장된 값중 내용의 고유ID 요소
    document.getElementById('task-board').innerHTML = resultHTML
}
```

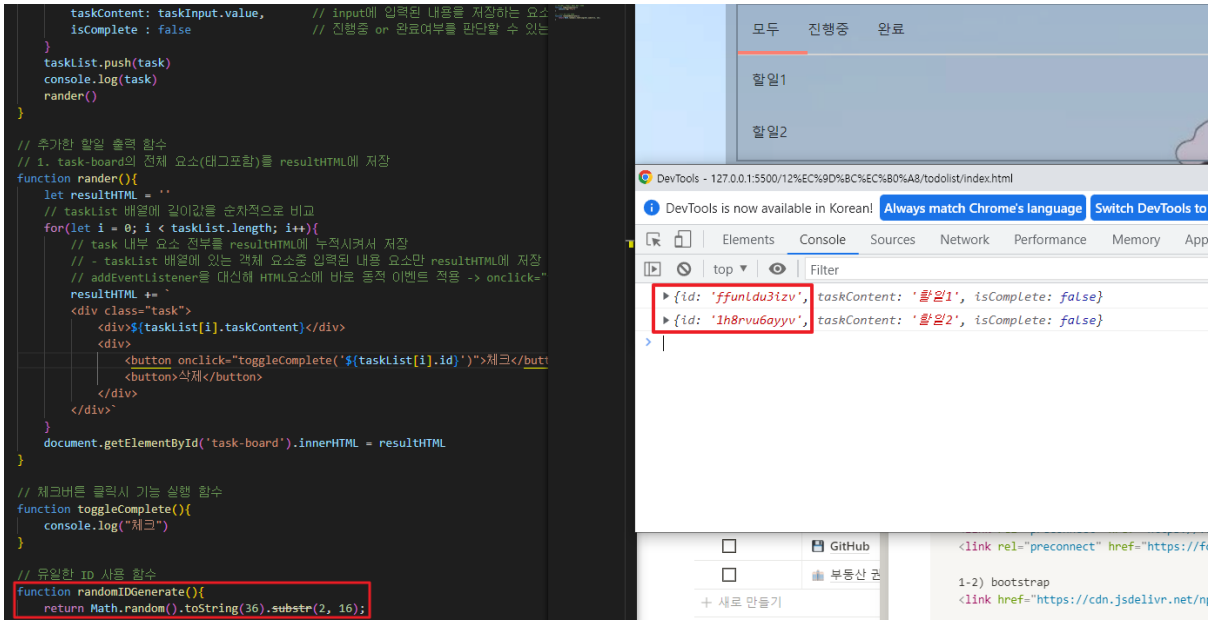
```
// 체크버튼 클릭시 기능 실행 함수
// 1) 할일에 체크를 하면 완료상태로 변경
function toggleComplete(id){
    console.log(id)
    // taskList 배열의 값의 id가 전달받은 id와 같다면 해당 할일을 완료 상태로 변경
    for(let i = 0 ; i < taskList.length ; i++){
        if(taskList[i].id == id){
            // 체크박스 체크 해제시 '완료' 상태를 '진행중' 상태로 변경
            // not 연산자를 이용하여 결과를 반대로 바꾼다
            taskList[i].isComplete = !taskList[i].isComplete
            break
        }
    }
    rander()
    console.log(taskList)
}

// 삭제버튼 클릭시 기능 실행 함수
// 1) 할일 삭제 버튼 누르면 삭제
function deleteTask(id){
    console.log(id)
    // taskList 배열의 값의 id가 전달받은 id와 같다면 삭제를 누른 배열 요소 1개 삭제
    for(let i = 0 ; i < taskList.length ; i++){
        if(taskList[i].id == id){
            taskList.splice(i, 1)
            break
        }
    }
    rander
}

// 모두, 진행중, 완료 tab 기능 실행 함수
// filter : 선택한 요소의 좌표값, 위치정보 등 모든 정보를 보여준다
function filter(event){
    console.log(event.target.id)
}


// 유일한 ID 값을 반환해주는 함수
function randomIDGenerate(){
    return Math.random().toString(36).substr(2, 16);
}
```

※ style.css #2

```

```

※JavaScript 유일한 ID 만들기

```
let newID = function () {
  // toString(36) : 36진수, 1~Z까지
  return Math.random().toString(36).substr(2, 16);
}
console.log(newID());
```



※ CSS를 활용하여 콘텐츠를 숨기는 방법

method	Visible	Accessible
`.sr-only`	X	O
`aria-hidden="true"`	O	X
`hidden=""`	X	X
`display: none`	X	X
`visibility: hidden`	X (Space)	X
`opacity: 0`	X (Space)	Depends?
`clip-path: circle(0)`	X (Space)	Depends?
`transform: scale(0)`	X (Space)	O
`width: 0` + `height: 0`	X	X
`content-visibility: hidden`	X	X

※ google font & bootstrap HTML , JS Link

```
1. HTML

1-1) google font
<link rel="preconnect" href="https://fonts.googleapis.com">
<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>

1-2) bootstrap
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-rbsA2VBKQhggwz

2. JavaScript

2-1) bootstrap
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-kenU1KFdBIe4zVF0s0G1M5b4h
```

※ input type = “checkbox” check box 네모 사이즈 조정

```
1-1. CSS 적용 #1
input[type=checkbox] {

-ms-transform: scale(2); /* IE */

-moz-transform: scale(2); /* FF */

-webkit-transform: scale(2); /* Safari and Chrome */

-o-transform: scale(2); /* Opera */

padding: 10px;

}

1-2. CSS 적용 #2
input[type="checkbox"]{

width: 30px; /*Desired width*/

height: 30px; /*Desired height*/

cursor: pointer;

-webkit-appearance: none;

appearance: none;

}

1-3. CSS 적용 #3
input[type=checkbox] {

zoom: 1.5;

}

1-4. CSS 적용 #4
input[type=checkbox] {

transform : scale(1.5);

}
```

```
2. HTML 적용
<input style="zoom: 2.0;" type="checkbox" id="checkbox">
```

※ HTML & CSS 참고 : https://www.coreasur.com/k-coding/ko/HTML/form-input-reset-tag/

https://getbootstrap.kr/

※ 바로 사용할 수 있는 애니메이션 효과 : https://wsss.tistory.com/1851

※ CSS 버튼 스타일 : https://velog.io/@whdnjsdyd111/CSS-버튼-이쁘게-꾸미기

※ rgba 색상표 사이트 : https://www.hexcolortool.com/#6bc2c1

[웹 컴바인 #4 : JQuery]