우리는 JavaScript 이론은 배웠지만, 한번도 제대로 써먹은적이 없다.

또한, 지금까지 배운 기초 JavaScript 로는 제대로 된 프로그램 만들기엔 한계가 있을수밖에 없다. 이것은 아무리 강의를 많이 제공해도 부족할 것이다.

이에, 간단한 게임을 만들어보면서 이벤트와 돔 조작에 대해 익혀보는 시간을 가지도록 하겠다.

우리가 만들어 볼 게임은 메모리 게임이다.

이 게임의 특징은 다음과 같다.

1. 맨 처음, 3*4 칸의 모든 화면은 물음표로 채워진다.
2. 아무 칸이나 클릭을 처음 하면 숨겨진 그림이 나온다.
3. 다른 칸을 클릭하면 숨겨진 그림이 나오면서,
   틀렸을 경우 0.5초 후에 물음표로 돌아간다.
   맞았을 경우 두 개의 그림을 그대로 둔다.
4. 나머지 그림에 2, 3을 반복하며, 모든 그림이 오픈되면 다 맞춘 것이다.

시작하자.

HTML

일단 기본 HTML 파일을 만든다.

index.html > ...

```html
<!DOCTYPE html>
<html lang="en">
<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>memory game</title>
</head>
<body>
    <h1>메모리 게임</h1>
</body>
</html>
```

타이틀과 <h1> 태그만 달아두었다.

css 와 js 파일을 사용할 것이기에, index.css, index.js 빈 파일 하나씩 만들고 임포트하자.

```html
    <link rel="stylesheet" href="./index.css">
    <script src="./index.js"></script>
    <title>memory game</title>
</head>
```

그리고 3*4 의 박스를 만들어야한다. 다음과 같이 입력하자.

```html
<body>
    <h1>메모리 게임</h1>
    <div class="container">
        <div class="row">
            <div class="column"></div>
            <div class="column"></div>
            <div class="column"></div>
            <div class="column"></div>
        </div>
        <div class="row">
            <div class="column"></div>
            <div class="column"></div>
            <div class="column"></div>
            <div class="column"></div>
        </div>
        <div class="row">
            <div class="column"></div>
            <div class="column"></div>
            <div class="column"></div>
            <div class="column"></div>
        </div>
    </div>
</body>
```

3*4 를 만든다는건, row 가 3, column 이 4라는 뜻이므로 container 클래스 안에 3개의 row 클래스, 각각의 row 안에 4개의 column 클래스를 만들었다.

그리고 우리가 클릭했을 때 이 카드를 뒤집었으면 좋겠다. 따라서, 각각의 column 클래스, 총 12 개에 onclick 이벤트 핸들러를 달아줄것인데, 함수 이름은 flip() 으로 하고 파라미터를 0번부터 11번까지 번호별로 넘겨주겠다.

CSS

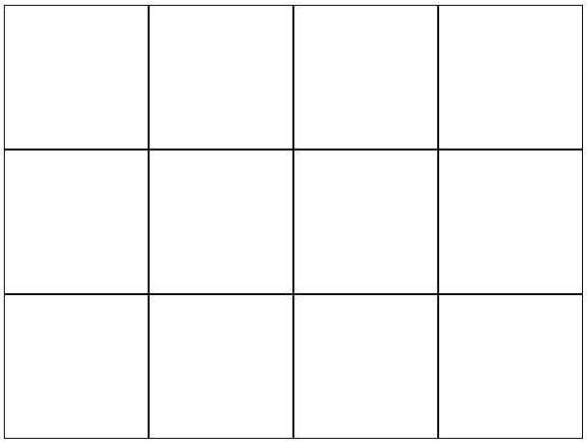
```css
.container {
  height: 300px;
  width: 400px;
  display: grid;
  grid-template-rows: 1fr 1fr 1fr;
}
.row {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr 1fr;
}
.column {
  border: 1px solid black;
}
```

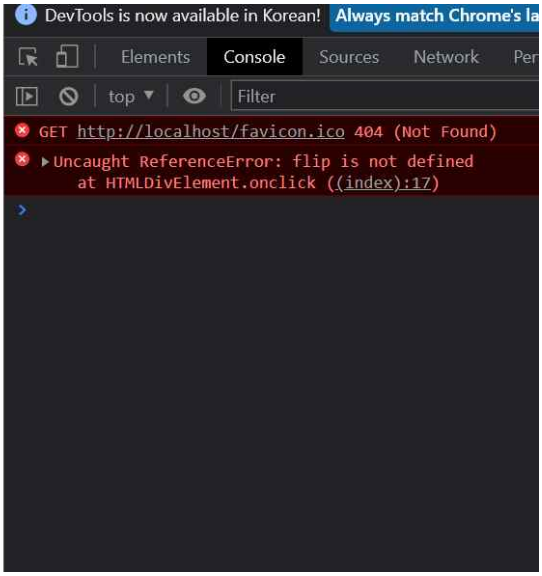
전체 칸들을 감싸는 컨테이너의 크기는 높이가 300px, 너비가 400px 이다. 100*100 의 정사각형을 총 12개 만들 것이기 때문이다.
display 는 grid 이고, 총 3 row 를 가지고 있으므로
grid-template-rows: 1fr 1fr 1fr;
이다.

반면, 각각의 row 역시 display 가 grid 이다. 자식이라도, display: grid 는 따로 지정해줘야한다. 기본적으로 div 태그는 block 이라 변경해줘야하기 때문이다.
여기서, 4개의 컬럼이니
grid-template-columns: 1fr 1fr 1fr 1fr;
이다.

각 컬럼은 눈에 보여야 하기 때문에, border 를 1px solid black 으로 주었다.

메모리 게임





아무 칸이나 클릭했을 때, flip is not defined 에러가 떴다.
이건 당연한데, 우린 flip 함수를 만든 적이 없기 때문이다.

이제 할 일은, 여러분들이 좋아하는 그림 6개를 인터넷에서 찾아, 그림판으로 100px*100px 사이즈로 바꿔서 프로젝트 디렉터리 안에 저장해두는것이다.
추가로, 아직 오픈되지 않은 칸을 표시하기 위해, 물음표 이미지 역시 100*100 으로 만들어서 담아두겠다.

난 public 이라는 디렉터리를 따로 만들어서 동물친구들과 물음표를 담아두었다.

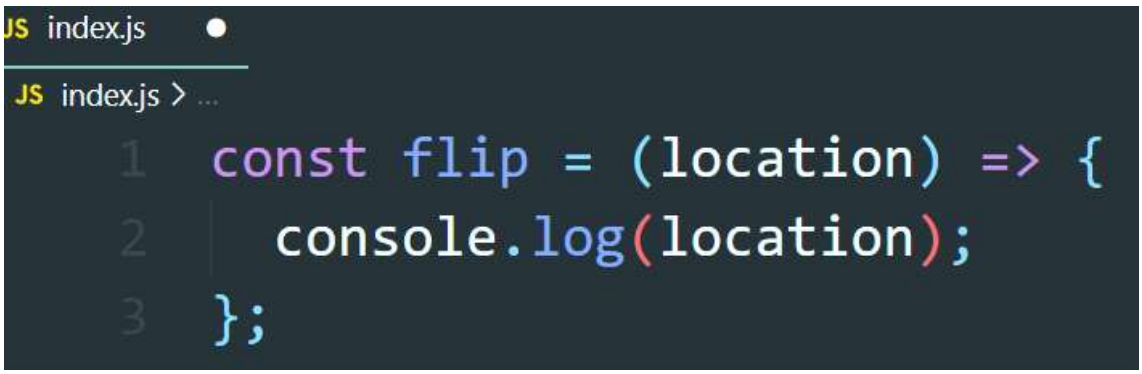
public 이라는 건 “공용” 이라는 뜻인데, 서버 내에서 공용으로 쓰는 디렉터리를 말한다. 이름을 바꾼다고해서 동작 안하는것은 아니고, image 등의 이름을 사용해도 좋다.

JavaScript

이제 주인공인 JavaScript 를 코딩해볼 시간이다.

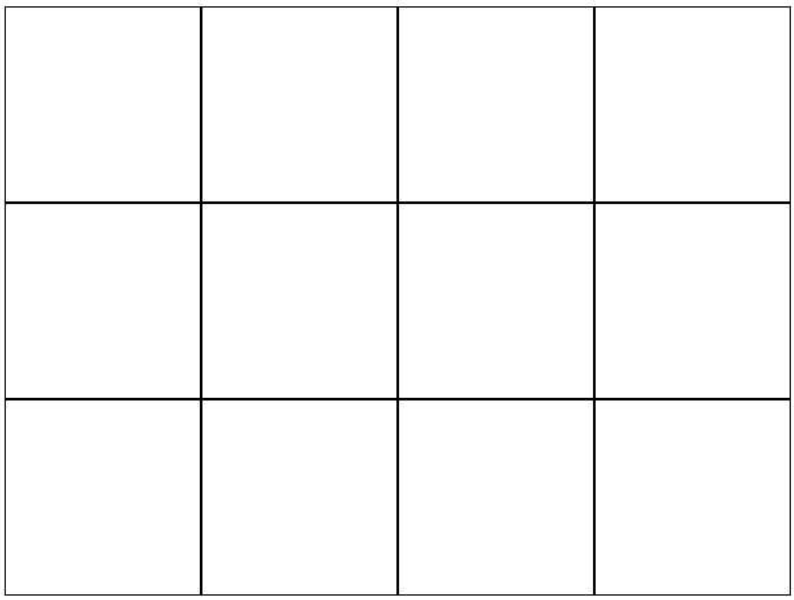
먼저, flip 함수부터 테스트해보자.

다음과 같이 쓴다.



우리가 onclick 이벤트핸들러에서 flip 이라는 함수를 달았을 때, 파라미터를 달았기 때문에, 해당 파라미터는 location 이라는 이름으로 받을 것이다.
각각을 찍어보면 사용자가 어느 칸에서 클릭했는지 위치가 출력된다.

메모리 게임



숫자는 0부터 11까지다. 그 뜻은 다음과 같다.

0	1	2	3
4	5	6	7
8	9	10	11

여러분이 누른 위치대로 콘솔에 출력되는것을 볼 수 있다.

전역변수 만들기

우리가 맨 처음 생각할 건, 우리가 다룰 데이터가 무엇인지부터 정의하는 일이다.
코딩은 아무리 크든 작든 데이터를 다루는 기술이다.

우리의 데이터 타입을 유리하게 정해놓고 (자료구조)
가진 데이터로 문제를 해결한다. (알고리즘)

그럼, 이렇게 생각해보자.
하나의 배열이 있다. 크기는 12 이다.

각각의 엘리먼트는 객체인데 다음으로 구성되어 있다.

이름

이미지경로

그리고 각각의 엘리먼트를 총 두번씩 반복한다.

무슨 말이냐면, 우린 클릭했을 때 카드 맞추는 게임을 만들 것이다.

그러면 같은 카드가 총 두개가 있어야 되기 때문에,

이미지가 6개 있다면 총 12개의 엘리먼트를 가진 배열을 만들어야한다.

```
// data
const cardArray = [
  {
    name: "cat",
    img: "./public/cat.png",
  },
  {
    name: "cat",
    img: "./public/cat.png",
  },
  {
    name: "dog",
    img: "./public/dog.png",
  },
  {
    name: "dog",
    img: "./public/dog.png",
  },
```

배열의 이름은 cardArray 이다. 프로퍼티는 일단 name 하고 img 만 둔다.
최종적으로는 4개의 프로퍼티가 필요한데, 그것은 코딩을 하면서 추가해나가도록 하자.

그리고, querySelector 를 배웠으므로 미리 쓰고자 하는 돔 정보를 다루기 쉬운 형태로 받아두는 배열을 만들자.
* 이렇게, 원래 데이터를 내가 처리하기 유리한 방식으로 처리하는것을 데이터 파싱(parsing) 이라고 한다.

```
  {
    name: "squirrel",
    img: "./public/squirrel.png",
  },
];
// 파싱한 DOM 정보
const gameDOM = [];
```

파싱한 돔 정보를 담는 하나의 배열을 gameDOM 이라고 이름지어놓고 빈 배열로 초기화시켰다.
즉, 쓰고자 하는 변수는 파일의 맨 위쪽에 배치하고, 아랫쪽엔 함수들을 배치할 것이다.

이 정도면 데이터는 일단 준비가 되었다.

onload()

페이지가 맨 처음 생겼을 때 자동으로 실행되는 함수는 onload() 이다. 제공하는 내장함수 중 하나이다. 화면이 완전히 생긴 다음 실행된다. 이걸 안 쓰면 당연하게도 에러가 생길 수 있는데, 이미 있지도 않은 화면을 DOM 조작할 순 없기 때문이다.
우린 이 함수를 "재정의(overriding)" 할 것이다. 어렵게 생각하지 말고, 이미 있는 함수를 우리가 쓰는 목적에 맞게 새로 작성한다고 생각하면 된다. 이 경우, 앞에 const 를 붙이지 않는다.

```
onload = () => {
    console.log("DOM is ready");
};
```

결과:

```
DOM is ready
```

getGameDOM()

맨 처음에 생각할 건, 화면의 12개 칸의 돔 정보를 우리가 나중에 다루기 쉽도록 gameDOM 배열에 받아두는 것이다. 이 일을 하는 함수를 getGameDOM() 이라고 하자.

```
onload = () => {
  // DOM 정보를 작업하기쉽게 미리 가져옴
  getGameDOM();
};
```

이제 getGameDOM 함수를 만들자. 당연히, onload 보다 위에 작성해야 한다.
선언이 되기 전엔 사용을 못하기 때문이다.
전역변수 아래
onload 위에
작성하도록 하자. 이 공간에 함수들을 넣을 것이다.

```
const getGameDOM = () => {
    console.log("getGameDOM 함수 실행");
};
```

결과:

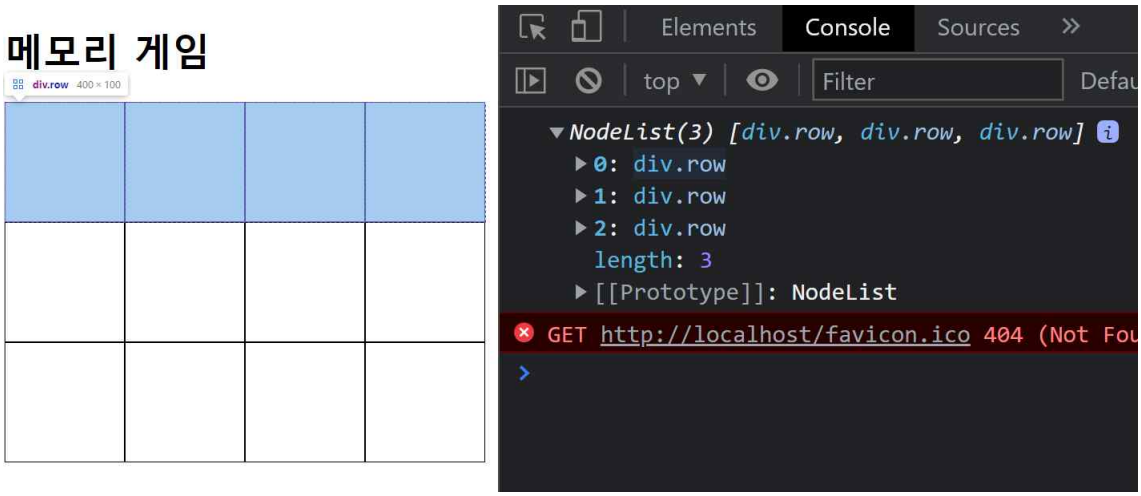
```
getGameDOM 함수 실행
```

화면이 로드됨과 동시에 콘솔에 찍혔으니 함수를 만들 준비가 되었다.

그럼, 생각해보자. 12개의 박스 돔 정보를 받아오려면 어떻게 하는 게 좋을까?
일단, 3개의 행(row) 에 대한 돔 정보를 임시로
반복문을 써서 각각에 대한 돔 정보를 gameDOM 배열에 받으면 될 것이다.
* querySelectorAll 로 받으면 배열(array) 형태가 되는 “것처럼” 보이지만, 사실 타입은 Array 가 아닌 NodeList 이다. Array 가 아니기 때문에 map 을 사용할 수 없으므로 for loop 를 사용한다.
동일한 클래스의 DOM 정보를 배열로 한꺼번에 받아오려면, querySelectorAll 을 사용한다. 다음과 같이 써보자.

```
const getGameDOM = () => {
  const rows = document.querySelectorAll(".container .row");
  console.log(rows);
};
```

결과:



콘솔의 div.row 각각에 마우스를 올려보면 각 줄에 해당하는 돔 정보를 받아왔기 때문에 파란색으로 연하게 블록처리 되는것을 볼 수 있다.
여기서,

querySelectorAll(“.container .row”)

라고 써줬다. 이건 무슨 뜻이냐면, container 클래스 안에 row 클래스를 가져오겠다는 뜻이다.
이렇게, 원하는 선택자 안에 한 칸 띄우고 선택자 식으로 쓸 수 있다.

그리고 다음 for loop 를 작성하자.

```
const getGameDOM = () => {
  const rows = document.querySelectorAll(".container .row");
  for (let i = 0; i < rows.length; i++) {
    gameDOM[i] = rows[i].querySelectorAll(".column");
  }
  console.log(gameDOM);
};
```

배열의 길이만큼(3) 반복하고, row[0] 에서 먼저 querySelectorAll 을 사용한 것을 볼 수 있다.

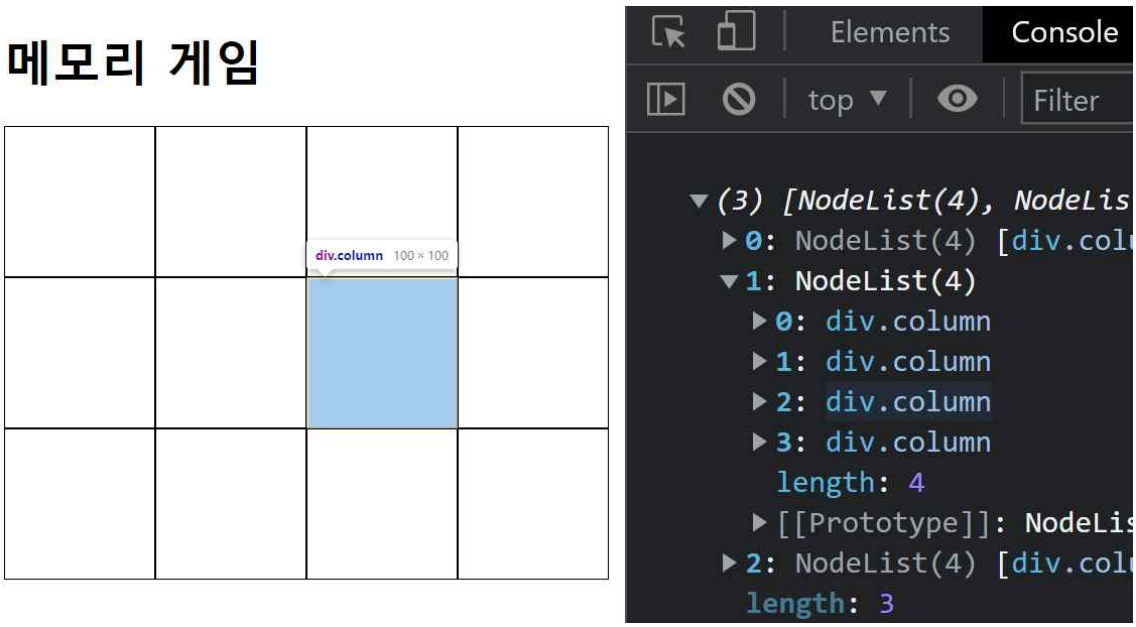
아니, querySelector 는 document 객체 안에서만 쓸 수 있지 않은가? 맞다. 그러나

우린 rows[0], rows[1], rows[2] 변수에 담을 때 document 를 포함시켜 담은 것이나 마찬가지다. 그래서 rows[0].querySelector 가 가능한 것이다.

이렇게 우리가 원하는 DOM, 즉 12칸을 파싱한 결과인 gameDOM 은 다음과 같다.

```
(3) [NodeList(4), NodeList(4), NodeList(4)]
  0: NodeList(4) [div.column, div.column, div.column, div.column]
  1: NodeList(4) [div.column, div.column, div.column, div.column]
  2: NodeList(4) [div.column, div.column, div.column, div.column]
  length: 3
  [[Prototype]]: Array(0)
```

즉, NodeList 안에 NodeList, 이차원 NodeList 이다.



1-2 에 커서를 올렸을 땐, 두번째 행(인덱스1) 에 세번째 열(인덱스2) 를 가리킴을 알 수 있다.
즉, gameDOM 을 getGameDOM() 함수로 미리 받아둠으로서, 나중에 작업하기 좋게, querySelector() 를 끝도 없이 반복하지 않도록 파싱해둔 것이다. 만약 각각의 칸에 대해 변경할 일이 있을때는 document 에 접근하는 것이 아니라, gameDOM 변수에 접근할 것이다.

배열의 순서 섞기

다시 onload() 로 돌아오자.
우리가 가진 배열을 섞지 않으면, 매번 같은 게임이 될 것이다. 그래서 gameDOM 으로 document 를 파싱한 다음 해야할 일은, 우리가 가진 배열의 순서를 섞는 것이다.
배열을 섞는 건 JavaScript 에서 기본으로 제공하는 기능이 아니다. 정렬을 하는 sort() 내장함수를 사용하되, sort 의 기준을 바꾸면 된다.
보통은 다음과 같이 한다.

```
onload = () => {
  // DOM 정보를 작업하기쉽게 미리 가져옴
  getGameDOM();
  cardArray.sort(() => 0.5 - Math.random());
};
```

cardArray 배열을 정렬(sort) 할 것인데, 정렬의 기준으로 콜백이 들어간다.
관습적으로, 파라미터는 없고 0.5를 난수 Math.random() 에서 빼주는 방식을 쓴다.
이 구문에 대해 깊게 알려면 설명할 게 많기 때문에, 다음 검색을 추천한다.

JavaScript array sort 0.5-Math.random()

sort 내장함수를 사용하면 원래의 cardArray 가 바뀐다는건 알고있어야 한다.
console.log(cardArray) 를 해보자.

원래의 배열이 섞인 것을 알 수 있다.

setIDtoCardArray()

이렇게 섞인 배열에 나중에 작업하기 좋게 아이디를 부여해주자.
무슨 말이냐, 원래 우리의 cardArray 배열은 인덱스로만 접근한다.
그러나, 1-2, 2-3 등의 형태로 접근할 수 있도록 하면 나중에 편할것이다.
그래서, cardArray 배열 각각의 엘리먼트 객체에 id 라는 프로퍼티를 하나씩 추가하고 그 값을 null 로 둘 것이다. 내가 일부러 비워두기 때문이다.

```
{
  name: "pig",
  img: "./public/pig.png",
  id: null,
},
{
  name: "squirrel",
  img: "./public/squirrel.png",
  id: null,
},
{
  name: "squirrel",
  img: "./public/squirrel.png",
  id: null,
},
```

이런 식으로.

그리고 setIDtoCardArray() 함수를 하나 만들자. 이 함수의 기능은, “섞여진” 배열에 아이디를 부여하는 것이다.

실행 부분만 먼저 적으면 다음과 같다. 맨 마지막 라인이다.

```
onload = () => {
  getGameDOM();
  cardArray.sort(() => 0.5 - Math.random());
  console.log(cardArray);
  setIDtoCardArray();
};
```

함수 정의 부분은 다음과 같다.

```
const setIDtoCardArray = () => {
  cardArray[0].id = "0-0";
  cardArray[1].id = "0-1";
  cardArray[2].id = "0-2";
  cardArray[3].id = "0-3";
  cardArray[4].id = "1-0";
  cardArray[5].id = "1-1";
  cardArray[6].id = "1-2";
  cardArray[7].id = "1-3";
  cardArray[8].id = "2-0";
  cardArray[9].id = "2-1";
  cardArray[10].id = "2-2";
  cardArray[11].id = "2-3";
};
```

cardArray 각각의 엘리먼트 객체에 id 라는 프로퍼티를 입력해주는 것이다.
왜 이런식으로 다 써줬을까? 개발자답지 않다.

그러나, 생각해보자.

0 0-0
1 0-1
2 0-2
3 0-3
4 1-0
4 1-1

무슨 규칙이 보이는가? 코딩 좀 하는 사람은 나머지(%) 값을 계산해서 처리할 수도 있겠지만, 익숙하지 않은 사람들은 규칙이 단번에 생각나지 않는다.
게다가, 12개면 일일히 쓴다고해서 힘든 일도 아니다. 비효율적이지만 당장 쉬운 코딩을 위해 쭉 다 써줬다.
setIDtoCardArray() 함수 실행 후, cardArray 를 콘솔로그로 찍어보면 다음과 같다.

자세히보면, 섞였고, 그 상태에서 순차적으로 아이디가 부여된것을 볼 수 있다.

0-0
0-1
0-2
0-3
1-0

이런 식으로.

createBoard()

onload 에서 마지막으로 할 일은, 물음표로 가득 찬 게임판을 생성하는 것이다.

```
onload = () => {
  getGameDOM();
  cardArray.sort(() => 0.5 - Math.random());
  setIDtoCardArray();
  createBoard();
};
```

createBoard() 라는 함수를 사용한다고 써놓고, 이제 만들어보자.

이 함수에선 for loop 안에 for loop 를 쓸 것이다.

```
const createBoard = () => {
  for (let i = 0; i < gameDOM.length; i++) {
    for (let j = 0; j < gameDOM[i].length; j++) {
      const card = document.createElement("img");
      card.setAttribute("src", "./public/Question-Mark.png");
      card.classList.add("eachImage");
      gameDOM[i][j].appendChild(card);
    }
  }
};
```

for loop 안에 for loop 를 쓸 때는 다음과 같이 i, j 의 기준, 즉 기준을 2개 잡는다.
여기서 여러분들이 모르는 개념이 나오므로 설명하자면,

document.createElement("img") <img> 태그 생성
생성된 img 태그는 일단 card 라는 변수에 담는다.

card.setAttribute("src", "./public/Question-Mark.png")
card는 하나의 <img> 태그다. <img> 태그에 src 애트리뷰트를 추가하며, 두번째 파라미터는 src의 값, 즉 이미지의 경로다. 우리는 물음표로 가득 채울 것이기 때문에, 반복하면서 모든 칸을 물음표 이미지로 채운다.

card.classList.add("eachImage")
<img> 태그에 eachImage 라는 클래스 이름을 추가한다.

gameDOM[i][j].appendChild(card)
appendChild() 는 자식태그를 만들 때 사용한다. card 라는 src 와 class 가 갖추어진 <img> 태그가 만들어졌으므로, 반복문으로 인덱스가 바뀌는 gameDOM 이차원 NodeList 에 <img> 태그를 자식태그로 넣는다.

반복을 마쳤을 때 최종적으로 다음 결과가 나온다.

결과:

이로써, onload() 에서 해야할 일은 끝났다. 이제, 각각을 클릭했을 때 실행할 flip 함수를 만들자.

flip()

이미 만들어둔 flip 함수는 다음과 같다.

```
const flip = (location) => {
  console.log(location);
};
```

화면에 각각의 칸을 클릭했을 때, 위치에 해당하는 일차원배열 인덱스, 즉 0부터 11까지 찍힐 것이다.

이 게임의 진행방식을 생각해보면,

1. 클릭을 하면 뒤집어진다.
2. 다른걸 클릭하면 뒤집어지고,
   맞으면 둘 다 뒤집힌채로 둔다.
   틀리면 0.5초 후에 둘 다 다시 뒤집어 물음표가 보이게 한다.

그렇다면, flip 함수는 “이미 맞춘” 칸에선 실행되면 안되는 함수다.
이미 맞췄다는 걸 어떻게 판단할까? cardArray 배열에 done 이라는 프로퍼티를 하나 추가하고, boolean 값을 담으면 될 것이다. 맨 처음엔 배열 각각의 엘리먼트의 done 프로퍼티가 모두 false 일 것이다. 작성해보도록 하자.

```
const cardArray = [
  {
    name: "cat",
    img: "./public/cat.png",
    id: null,
    done: false,
  },
  {
    name: "cat",
    img: "./public/cat.png",
    id: null,
    done: false,
  },
  {
    name: "dog",
    img: "./public/dog.png",
    id: null,
    done: false,
  },
```

이런 식으로, 각각에 done: false 를 달아주자.

그렇다면, 우리 flip 함수는 done 이 false 일때만 실행되면 된다.

```
const flip = (location) => {
  if (!cardArray[location].done) {
    console.log(location);
  }
};
```

이렇게 해두고 콘솔로그 찍어보면, 아까와 결과는 다를 바 없는것을 알 수 있지만, 앞에 느낌표를 떼버리고 테스트해보면 flip 이 작동 안되는것을 알 수 있다.
현재 cardArray 배열에 있는 모든 엘리먼트의 done 프로퍼티가 false 이기 때문이다.

즉, 모든 로직은 위의 if 조건문 안에서만 작성하면 된다. 중간에 done 이 true 로 변경되면 flip 은 실행되지 않을 것이다.

다음 할 일은 사용자의 첫번째 클릭, 두번째 클릭의 정보를 전역변수에 저장해두는 것이다. 이걸 저장해두었다가, gameDOM 을 교체할 때 사용할 계획이다.

먼저, 전역변수 두 개를 let 으로 선언한다. clickFirst, clickSecond 라는걸 만들고 -1로 각각 초기화시켜주자.

```
let clickFirst = -1;
let clickSecond = -1;
```

다음, flip() 함수 안에 setClickHistory() 라는 함수를 달아두자.

```
const flip = (location) => {
  if (!cardArray[location].done) {
    setClickHistory(location);
  }
};
```

함수의 파라미터로 클릭을 했을 때 받은 location 을 넘겨준다.

함수 정의는 다음과 같다.

```
const setClickHistory = (location) => {
  if (clickFirst === -1) {
    clickFirst = location;
  } else {
    clickSecond = location;
  }
};
```

clickFirst 가
  -1 이라는 건 처음 클릭했다는 뜻이므로 location을 clickFirst에 넣는다.
  -1 이 아니라는 건 두번째 클릭했다는 뜻이므로 location을 clickSecond 에 넣는다.

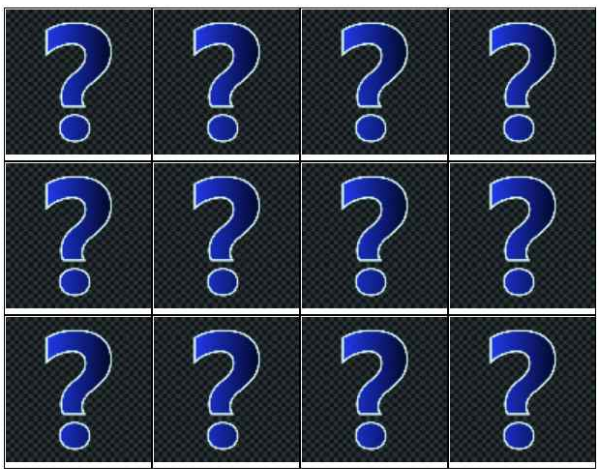
그리고 flip() 에서 clickFirst와 clickSecond 가 둘 다 -1이 아니면 초기화를 시켜야한다.

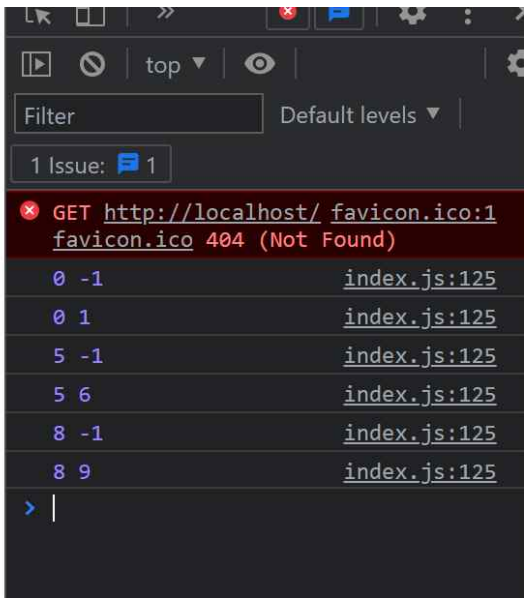
console.log(clickFirst, clickSecond) 를 flip() 함수 안에 초기화되기 전에 찍어보면 다음과 같다.

```
const flip = (location) => {
  if (!cardArray[location].done) {
    setClickHistory(location);
    console.log(clickFirst, clickSecond);
    if (clickFirst !== -1 && clickSecond !== -1) {
      clickFirst = -1;
      clickSecond = -1;
    }
  }
};
```

결과:

메모리 게임





여섯개 로그를 분석해보면
첫번째줄 0(0-0) 을 처음 클릭
두번째줄 1(0-1) 을 두번째로 클릭
세번째줄 -1로 초기화 된 상태에서 5(1-1)를 처음 클릭
네번째줄 6(1-2) 를 두번째로 클릭
다섯번째줄 -1 로 초기화 된 상태에서 8(2-0) 을 처음 클릭
여섯번째줄 9(2-1) 을 두번째로 클릭

원하던 결과대로 찍힘을 알 수 있다.

생각해보면 우리가 결국 하고자 하는 건, 클릭했을 때 gameDOM 에 접근해 물음표 이미지를 동물 사진으로 바꾸는 것이다.
그런데, gameDOM 은 이차원 NodeList 로 되어있어서, 0부터 11까지의 인덱스 정보인 location 으로는 접근할 수 없다.
이런 상황이 미리 올 것을 생각해, 우린 setIDtoCardArray() 함수를 이용해 셔플된 cardArray 배열에 0-0 부터 2-3까지의 아이디를 부여했다. 이 각각의 아이디는 1차원배열 인덱스인 0부터 11까지로 접근 가능하다.

```
const flip = (location) => {
  if (!cardArray[location].done) {
    setClickHistory(location);

    console.log(cardArray[location].id);

    if (clickFirst !== -1 && clickSecond !== -1) {
      clickFirst = -1;
      clickSecond = -1;
    }
  }
};
```

결과:

성공적이다. 그러나 하나의 문제가 있다.

“1-2” 를 예로 들어보면, 이 자체로 하나의 string 이라서 1과 2를 분리해야만 gameDOM 에 이차원 NodeList 인덱스로 접근 가능하다.

이 경우, split() 내장함수를 사용해서 분리해주면 된다. split() 함수는 파라미터로 분리할 기준을 string 으로 넣으면 분리해서 배열로 담는다.
그러니깐, split("-") 라고 쓰면 “-” 를 기준으로 둘을 분리해 배열로 만드는 것이다.

콘솔로그를 다음과 같이 고쳐 써보자.

```
const flip = (location) => {
  if (!cardArray[location].done) {
    setClickHistory(location);

    console.log(cardArray[location].id.split("-"));

    if (clickFirst !== -1 && clickSecond !== -1) {
      clickFirst = -1;
      clickSecond = -1;
    }
  }
};
```

결과:

성공적이다. 이제 분리된 인덱스 두 개로 내가 원하는 위치의 gameDOM 에 접근할 수 있다. 일단, 분리한 두 인덱스를 parseId 라는 변수에 담고, 다음과 같이 써보자.

```
const flip = (location) => {
  if (!cardArray[location].done) {
    setClickHistory(location);

    const parsedId = cardArray[location].id.split("-");
    gameDOM[parsedId[0]][parsedId[1]].querySelector("img").src =
      cardArray[location].img;

    if (clickFirst !== -1 && clickSecond !== -1) {
      clickFirst = -1;
      clickSecond = -1;
    }
  }
};
```

일단, cardArray 에 원하는 이미지 경로에 접근하려면 인덱스 location 에서 .img 로 접근해야 한다. 이것이 우리가 바꾸고 싶은 이미지 경로다.
그것을 어디에 넣는가? gameDOM[parseId[0]parseId[1]] 의, 우리가 createElement() 함수를 사용해 만든 <img> 태그에 접근해야하기 때문에 querySelector("img") 라고 쓰고, <img> 태그의 src 애트리뷰트에 넣어주는 것이다.

여기까지 하면 어떤 결과가 나오는지 보자.

결과:

클릭하면 이미지가 물음표에서 해당 동물사진으로 변경된다.
새로고침을 하고 다시 시도하면, 그림의 배치가 완전히 달라짐을 알 수 있다.
새로고침을 할 때, cardArray 배열을 섞는 코드를 작성해두었기 때문이다.

이제 우리가 할 건, 사용자가 두 번째 클릭을 했을 때 맞았는지 틀렸는지를 판단하는 것이다. 그렇다면 사용자가 두 번 클릭했다는것을 알 수 있어야 한다.
이를 위해 전역변수 clickCount 라는걸 만들고 0으로 초기화시키자.

```
    done: false,
  },
];
// 파싱한 DOM 정보
const gameDOM = [];
let clickFirst = -1;
let clickSecond = -1;
let clickCount = 0;
```

그리고 사용자가 클릭하면 clickCount를 증가시키는데, 만약 사용자가 두 번 클릭했다면 0으로 초기화시켜야할 것이다.
콘솔로그로 clickCount를 찍어보자.

```
const parsedId = cardArray[location].id.split("-");
gameDOM[parsedId[0]][parsedId[1]].querySelector("img").src =
  cardArray[location].img;

clickCount++;
if (clickCount === 2) {
  clickCount = 0;
}
console.log(clickCount);

if (clickFirst !== -1 && clickSecond !== -1) {
  clickFirst = -1;
  clickSecond = -1;
}
```

결과:

두번째 클릭했을 때 정확히 0으로 돌아감을 알 수 있다.
그러면, clickCount 를 0으로 초기화시키고 나서, 두 개의 사진이 일치하는지 판단하는 함수를 사용해

일치하면 그대로 두고

일치하지 않으면 0.5초 후에 다시 물음표로 뒤집는다.

이것을 하는 함수를 isCorrect() 라고 하고 일단 사용하겠다고 if 안에 쓰자.

```
clickCount++;
if (clickCount === 2) {
  clickCount = 0;
  isCorrect();
}
```

isCorrect()

항상 시작은 console.log() 이다. flip() 위에다 함수를 정의하자.

```
const isCorrect = () => {
  console.log("isCorrect 실행");
}

const flip = (location) => {
  if (!cardArray[location].done)
    setClickHistory(location);

    const parsedId = cardArray[lo
```

결과:

사용자가 두번째 클릭을 할 때, isCorrect() 함수가 실행됨을 알 수 있다.
이제 판단을 해야한다. 두번째 클릭을 했을 땐 clickFirst 와 clickSecond 둘 다 -1 이 아닌, 각각의 일차원 인덱스가 들어온 상태이다.

이 둘을 사용해 cardArray[clickFirst].name 식으로 접근하면, 내가 클릭한 동물 이름을 받을 수 있을 것이다.

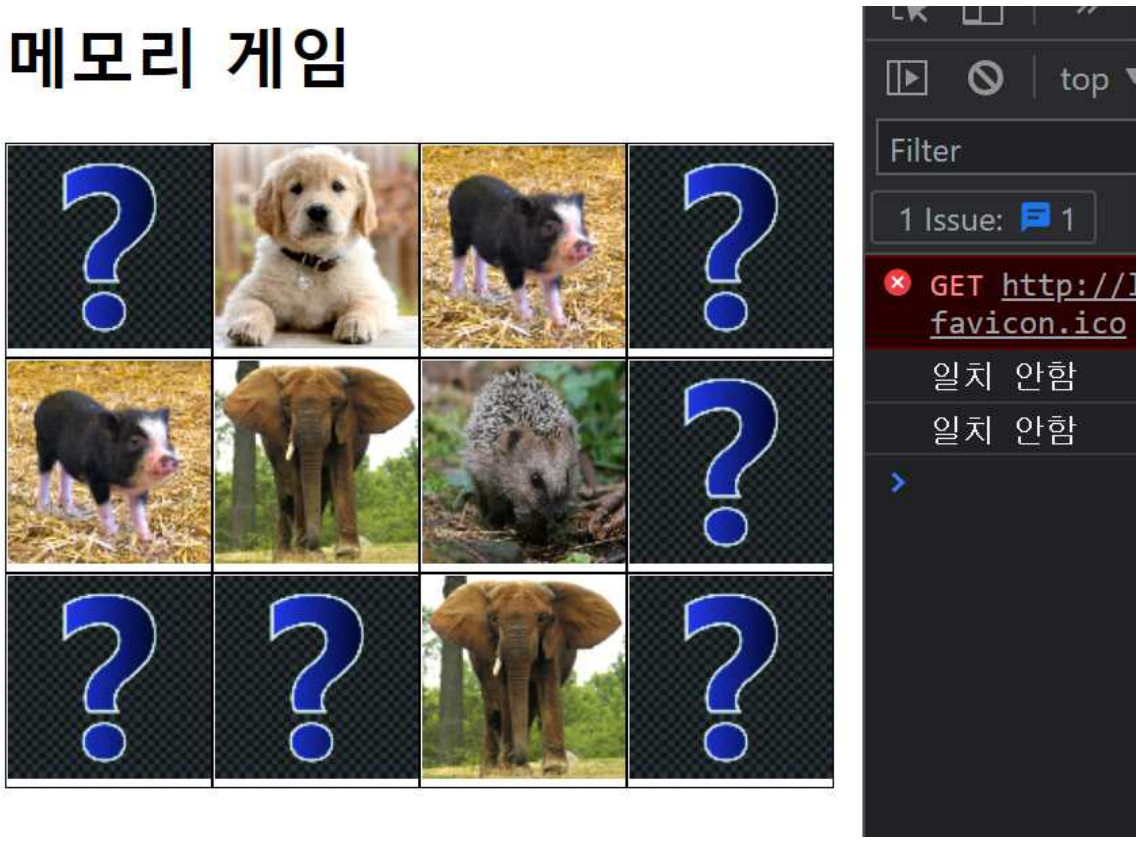
그리고 일치하면, cardArray 의 해당 엘리먼트의 done 프로퍼티를 false 에서 true 로 바꾸어 flip() 함수가 실행되지 못하게 할 것이다.

일치하지 않으면, 둘 다 다시 뒤집어버리면 된다. 이 일을 할 함수를 backFlip() 이라는 함수라고 이름짓고, 다음과 같이 코드를 써보자.

코드:

```
const isCorrect = () => {
  if (cardArray[clickFirst].name === cardArray[clickSecond].name) {
    cardArray[clickFirst].done = true;
    cardArray[clickSecond].done = true;
  } else {
    console.log("일치 안함");
    // backFlip();
  }
};
```

결과:



일치하는 두 짝이 나왔을때는 일치 안함이 뜨지 않는다.
일치하지 않는 두 짝이 나왔을때는 일치 안함이 뜬다.
그러나 아직 일치하지 않았을 때 물음표로 뒤집어지지는 않는다.

주석으로 쓴 backFlip() 을 풀고, 마지막 함수인 backFlip() 을 만들어보자.

backFlip()

틀렸을 때 뒤집는 함수다. 생각을 먼저 해보면,

처음 클릭한 카드의 인덱스는 어디에 저장되어 있는가? clickFirst
두번째 클릭한 카드의 인덱스는 어디에 저장되어 있는가? clickSecond
이 둘의 타입은 무엇인가? 0부터 11까지의 number
어디에 접근해야하는가? gameDOM
gameDOM 에 접근하려면? 0부터 11까지 받았으면
1-3 형태로 파싱해야

파싱부터 먼저 하자. parsedIdFirst, parsedIdSecond 두 개의 지역변수를 선언한다.

```
const backFlip = () => {
  const parsedIdFirst = cardArray[clickFirst].id.split("-");
  const parsedIdSecond = cardArray[clickSecond].id.split("-");
};
```

그리고 querySelector() 를 사용해 <img> 태그의 src 프로퍼티에 접근 후, 물음표 경로로 넣어주면 될 것이다.

```
const backFlip = () => {
  const parsedIdFirst = cardArray[clickFirst].id.split("-");
  const parsedIdSecond = cardArray[clickSecond].id.split("-");
  gameDOM[parsedIdFirst[0]][parsedIdFirst[1]].querySelector("img").src =
    "./public/Question-Mark.png";
  gameDOM[parsedIdSecond[0]][parsedIdSecond[1]].querySelector("img").src =
    "./public/Question-Mark.png";
};
```

결과:

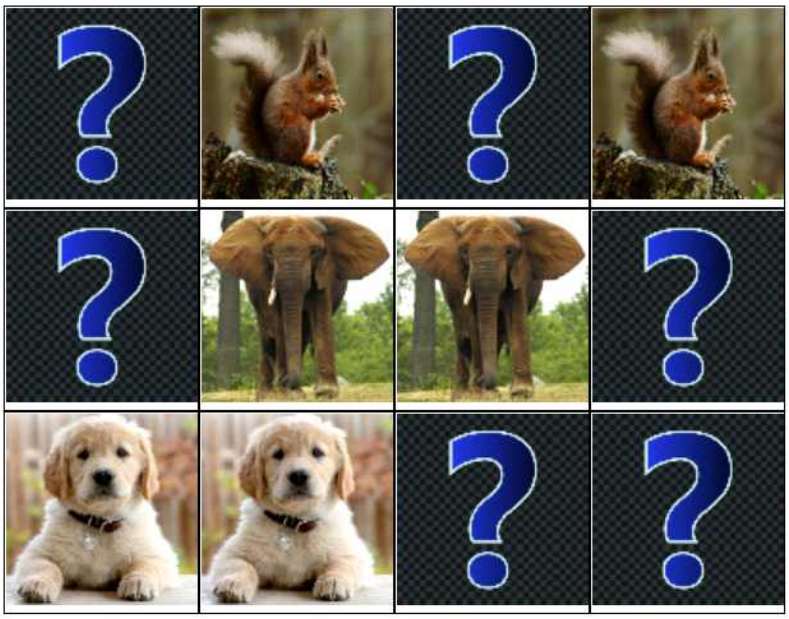
이 경우, 테스트를 해보면 알겠지만 두번째 카드가 아예 나오지 않아서 맞았는지 틀렸는지 여부를 알 수 없다.
사용자가 두번째를 클릭하자마자 일치하는지를 판단하고, 바로 물음표로 뒤집어버렸기 때문이다.
그래서, setTimeout() 을 사용해 0.5 초, 500 밀리초의 딜레이를 주면 해결될 것이다.

다음과 같이 쓰자.

```
const backFlip = () => {
  const parsedIdFirst = cardArray[clickFirst].id.split("-");
  const parsedIdSecond = cardArray[clickSecond].id.split("-");
  setTimeout(() => {
    gameDOM[parsedIdFirst[0]][parsedIdFirst[1]].querySelector("img").src =
      "./public/Question-Mark.png";
    gameDOM[parsedIdSecond[0]][parsedIdSecond[1]].querySelector("img").src =
      "./public/Question-Mark.png";
  }, 500);
};
```

이제 게임을 테스트해보자!

메모리 게임



마치며

해보니깐 알았겠지만, JavaScript 가 쉽지만은 않았을 것이다.
단순한 게임 만들기 강의가 아니다. 브라우저에서 JavaScript 를 어떻게 사용하는지 보여준 강의이다.
물론, 이벤트와 돔 조작에 대해 모든 것을 가르쳐주진 않았다. 차후 여러분들이 직접 개발하면서, 검색해보고 익혀보도록 하자.