

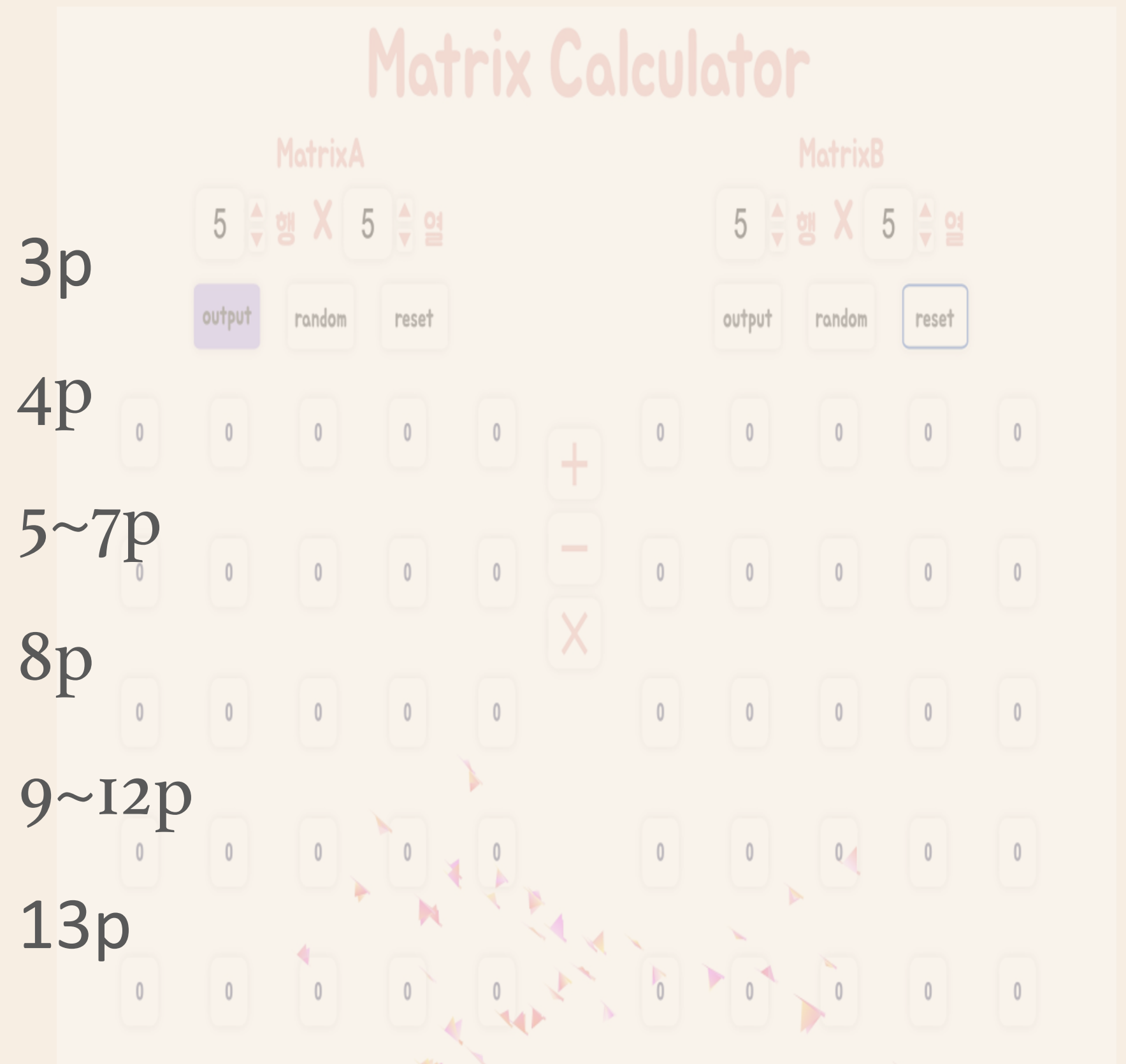
# MATRIX CALCULATOR



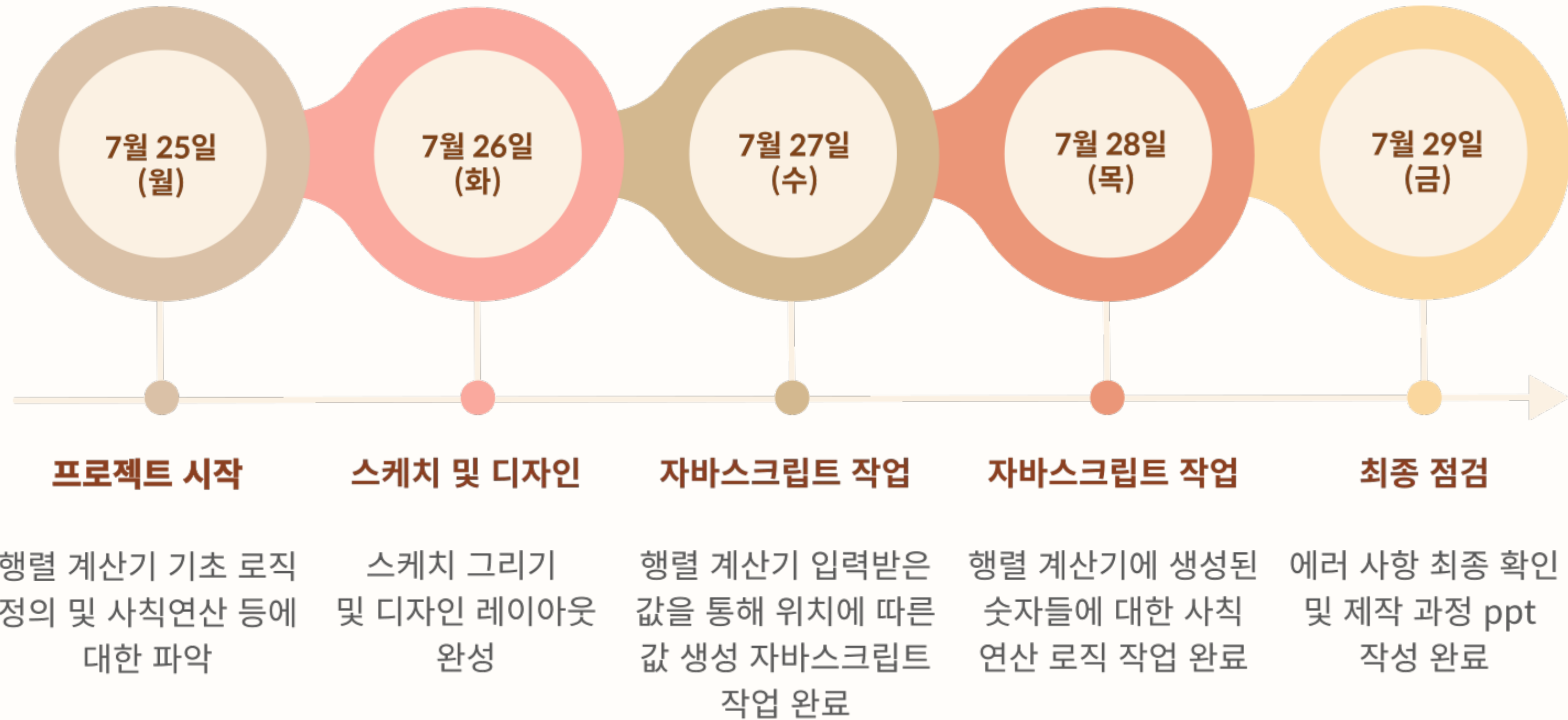
라인컴퓨터아트학원  
C16ST15  
손성균

# 목차

1. 일정표
2. Sketch
3. Design Layout
4. File Directory
5. Code
6. 마무리

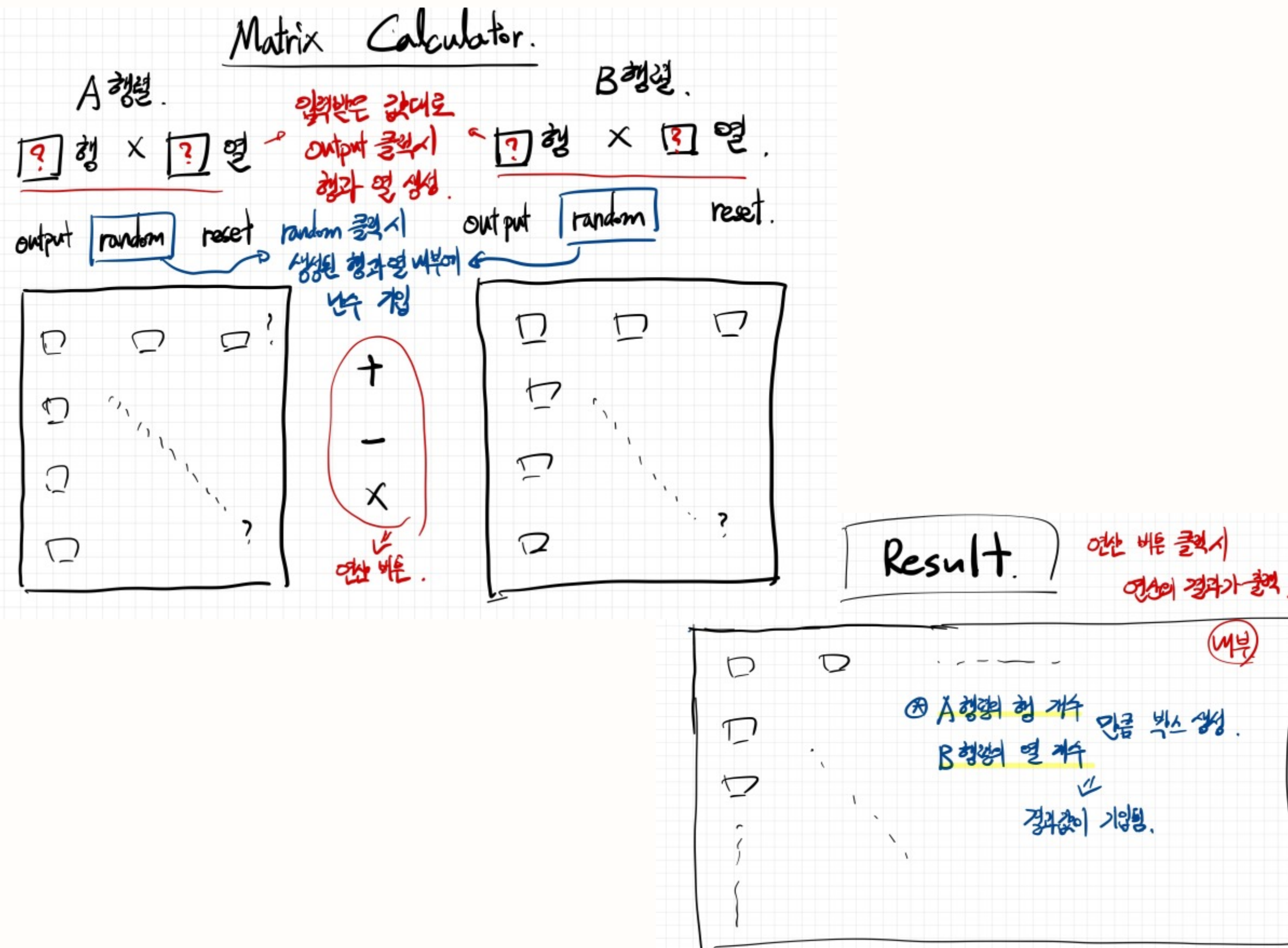


# 일정표



# Matrix Calculator Sketch

Sketch



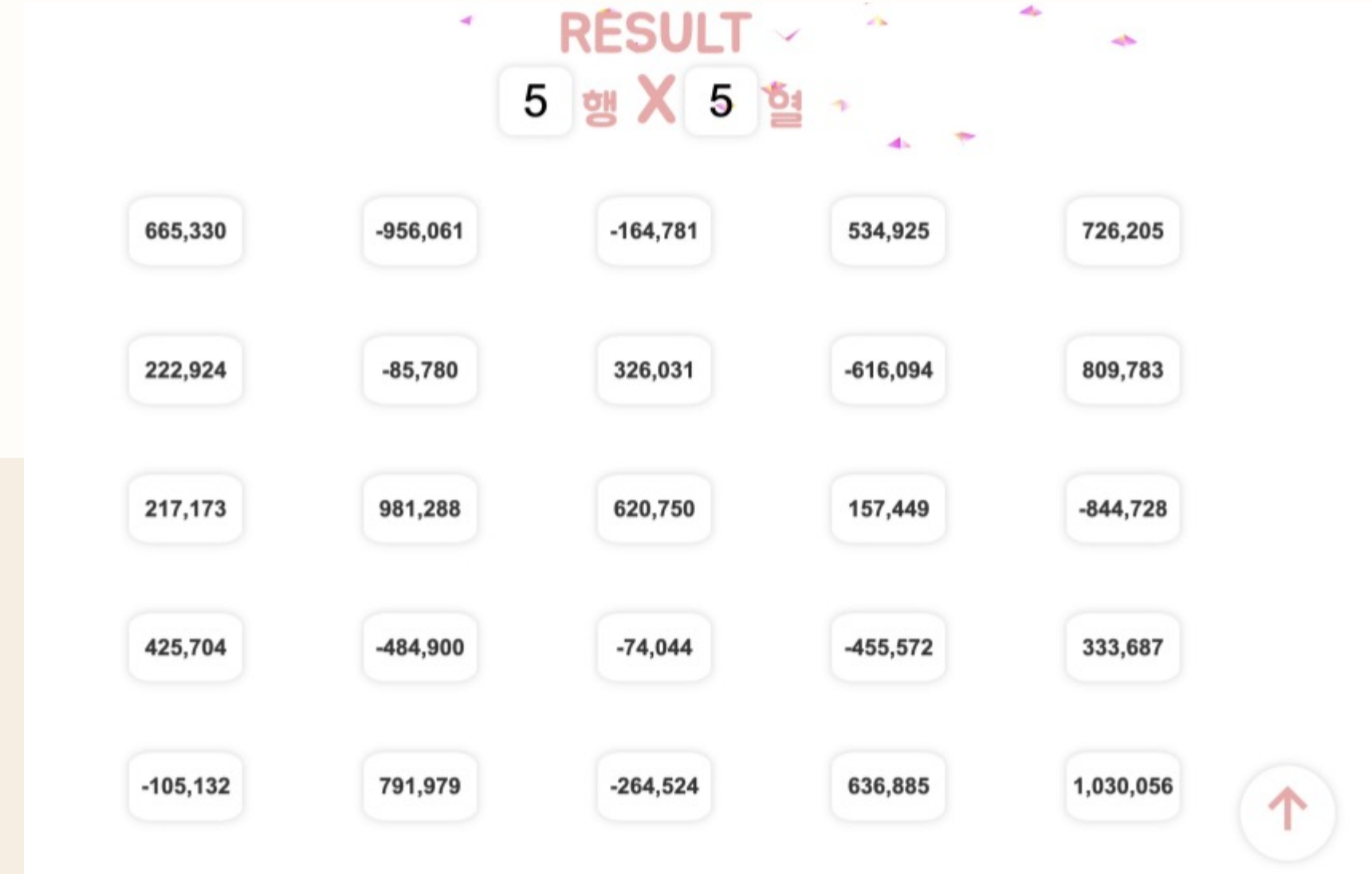
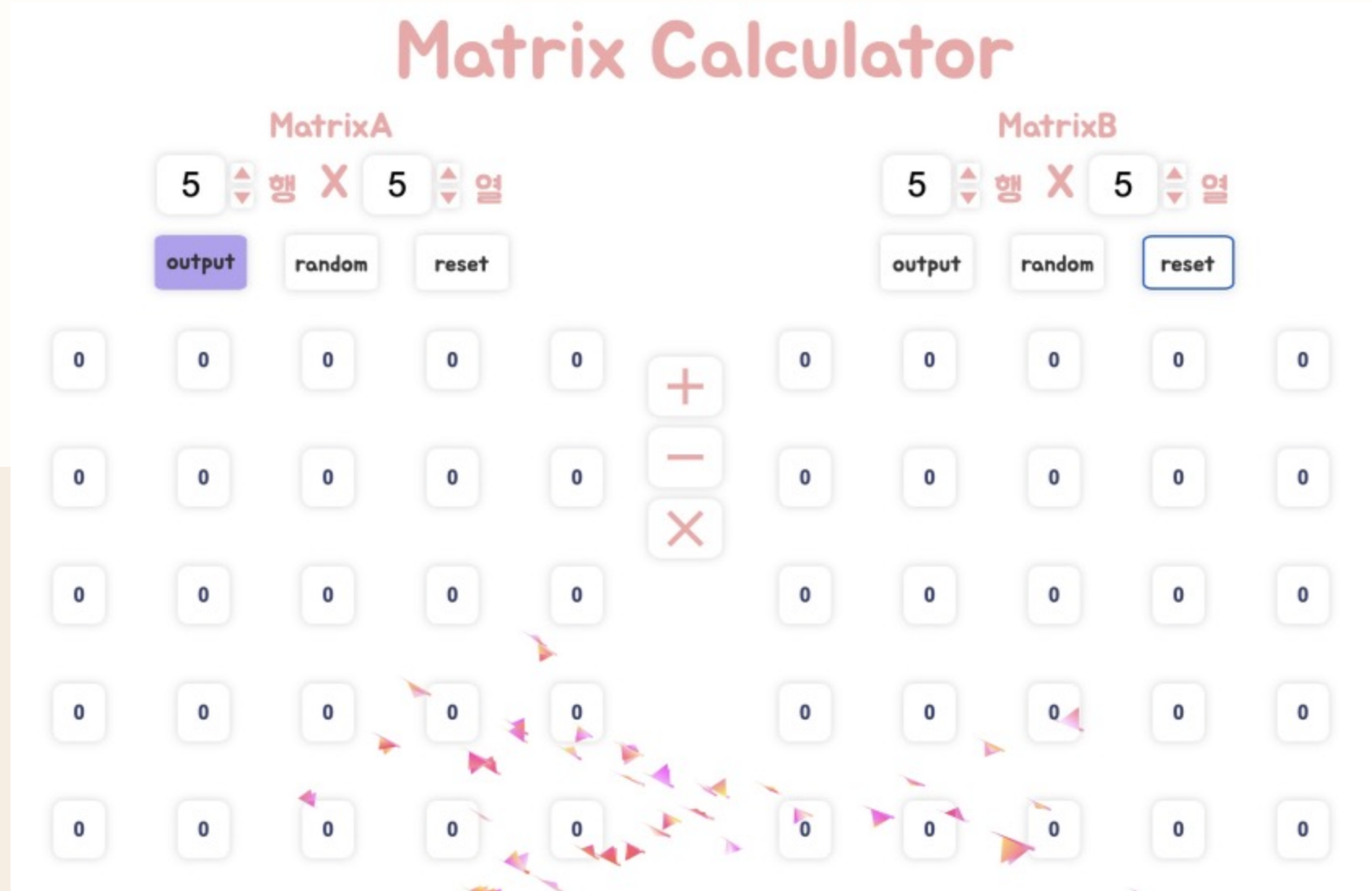
## 스케치 작성 시 고려한 부분

행렬 계산기 작동과 가독성을 위해 공간 활용 최대화.

결과 창을 한 화면에 띄우기 보다 페이지 스크롤을 통해 결과를 산출.

행렬 값을 직접 입력하는 것과 랜덤한 숫자를 나타내어 계산기가 작동하도록 버튼 개수 및 명칭 고려.

# Design Layout







## 디자인 설명

1. 상자들의 행과 열의 값은 1~9까지의 값을 입력받으면서 출력되며 상자 간의 간격은 일정하도록 유지하였습니다.
2. 글자색과 배경이미지의 조화를 위해 색을 통일시켰습니다.
3. 결과 창 값이 1,000단위를 넘어가면 “,” 표시를 추가하도록 설계하였습니다.
4. Result 우측하단에 버튼을 생성하여 결과 확인 후 초기화면으로 돌아갈 수 있도록 설계하였습니다.



# Matrix Calculator

MatrixA

5   행 X 5   열

output

random

reset

0

0

0

0

0

+

-

X

0





0

0

0

0

MatrixB

5   행 X 5   열

output

random

reset

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

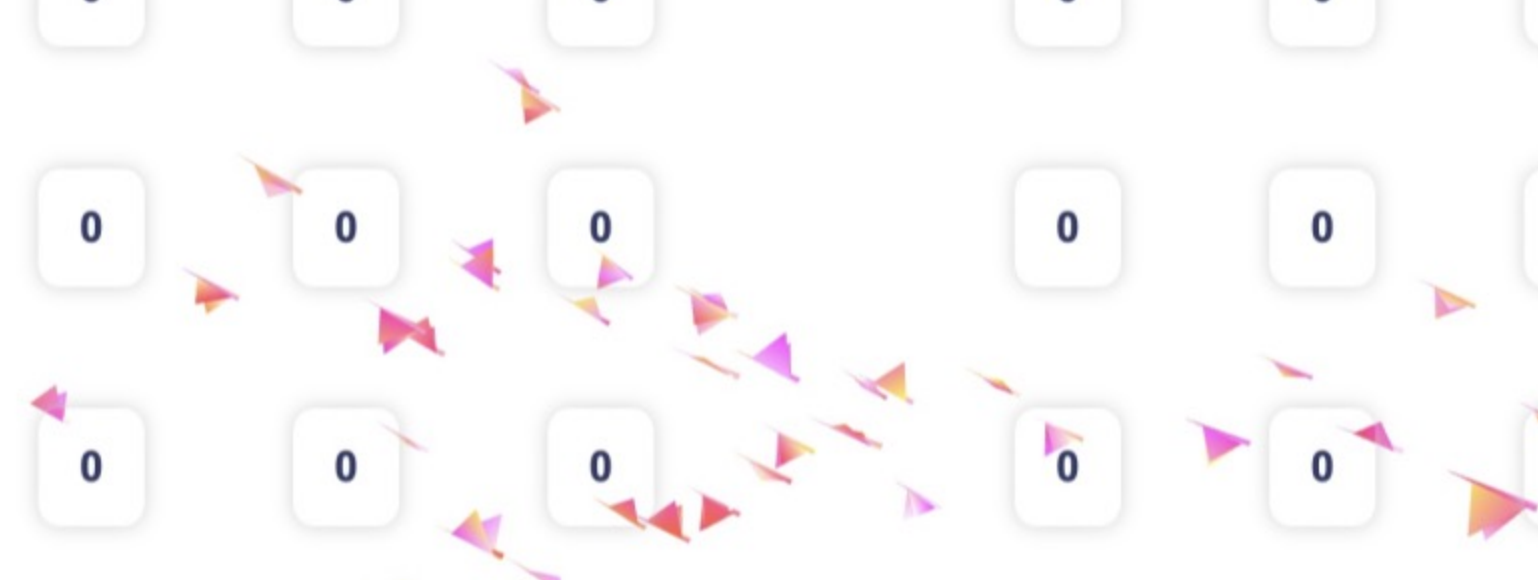
0

0

0

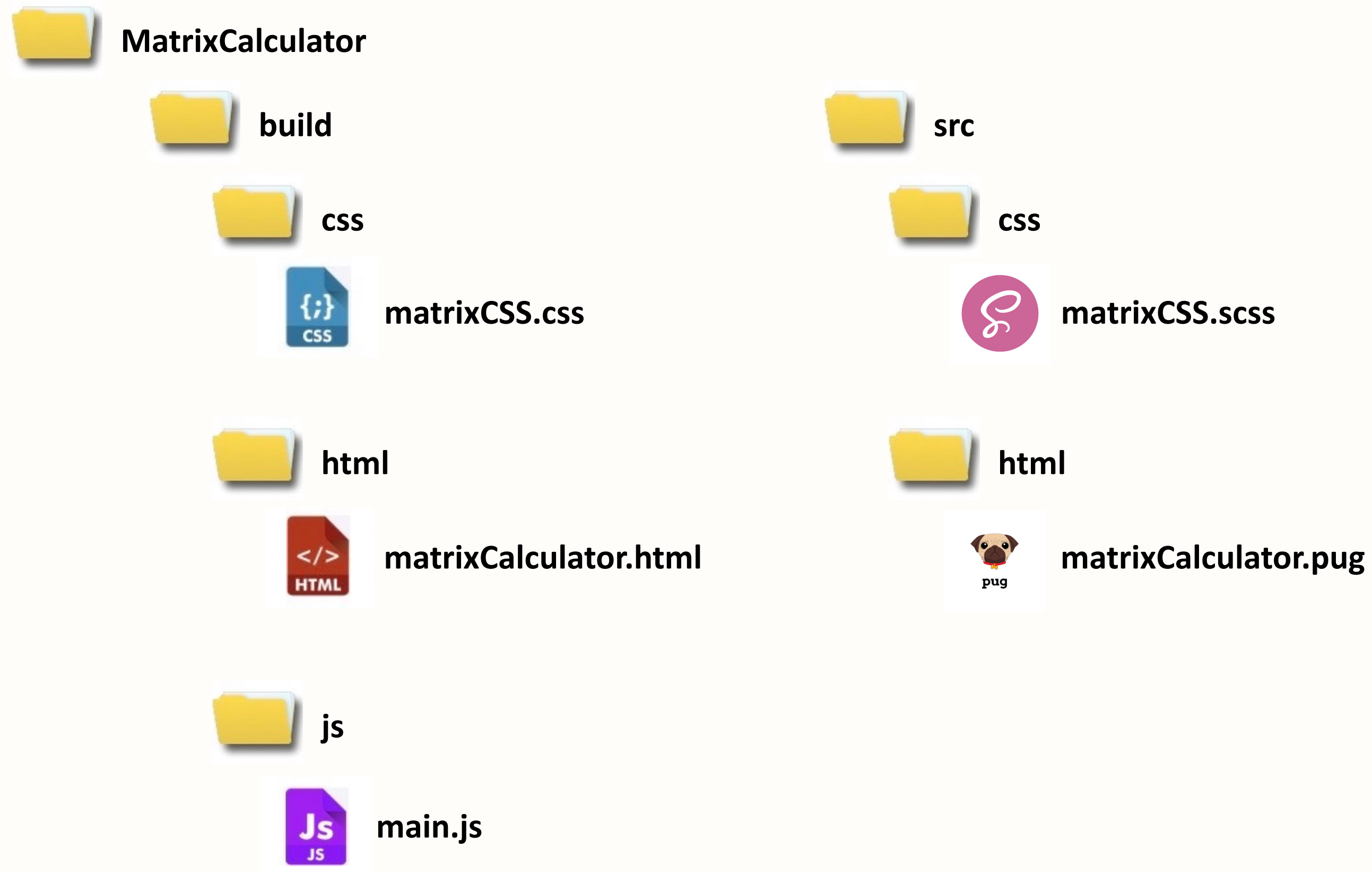
0

0



RESULT					
5	행	X	5	열	
665,330	-956,061	-164,781	534,925	726,205	
222,924	-85,780	326,031	-616,094	809,783	
217,173	981,288	620,750	157,449	-844,728	
425,704	-484,900	-74,044	-455,572	333,687	
-105,132	791,979	-264,524	636,885	1,030,056	↑

# File Directory





# Code

```
if(document.querySelectorAll("#innerBox_aRow").length == 0 && document.querySelectorAll('.innerBox_aRow_1 #innerBox_aColume').length == 0){
    let i=0;
    while(i<inputValue_aRow){
        $('#matrixA_output').append('<div class="innerBox_aRow_${i+1}" id="innerBox_aRow"></div>');
        let j=0;
        while(j<inputValue_aColume){
            $('.innerBox_aRow_${i+1}').append('<input class="innerBox_aColume_${j+1}" id="innerBox_aColume" type="text" value="0" oninput="handle_input(this, 4)">');
            j++;
        }
        i++;
    }
} else {
    //행 감소에 따른 로직
    if(inputValue_row < 0){
        let rowCount = Math.abs(inputValue_row);
        console.log(rowCount)
        let i=0;
        while(i<rowCount){
            $('#innerBox_bRow: last-child').remove();
            console.log(1)
            i++;
        }
    }
    //열 감소에 따른 로직
    if(inputValue_colume < 0){
        let columeCount = Math.abs(inputValue_colume);
        console.log(columeCount)
        let j=0;
        while(j<columeCount){
            $('#innerBox_bColume: last-child').remove();
            console.log(1)
            j++;
        }
    }
}
```

## 1. Output 버튼 실행

Output 버튼을 클릭하면 발생하는 이벤트로서 행과 열의 값을 입력받고 해당 값에 해당하는 행의 개수에 해당하는 div박스를 먼저 출력합니다.

이후 각 행마다 반복문을 통하여 열의 개수를 입력받은 만큼 행렬 각 요소에 입력할 수 있는 상자를 생성하는 로직입니다.

```

$("#matrixA_random").on("click",outputCal_random);
function outputCal_random(){
  $("#matrixA_output").empty();
  const inputValue_rowR = $("#matrixA_row").val();
  const inputValue_columeR = $("#matrixA_colume").val();
  let i=0;
  while(i<inputValue_rowR){
    $('#matrixA_output').append('<div class="innerBox_aRow_${i+1}" id="innerBox_aRow"></div>');
    let j=0;
    while(j<inputValue_columeR){
      $('#.innerBox_aRow_${i+1}').append('<input class="innerBox_aColume_${j+1}" id="innerBox_aColume" type="text" min="1" max="999" oninput="handle_input(this, 4)">');
      $('#.innerBox_aRow_${i+1}.innerBox_aColume_${j+1}').attr('value',`${Math.floor(Math.random()*(999-(-999)+1))+(-999)}`);
      j++;
    }
    i++;
  }
}

```

## 2. Random 버튼 실행

Random버튼은 행과 열의 값을 입력 받아 output 버튼 등을 이용하여 생성된 행과 열에 임의의 숫자를 넣어 계산할 수 있는 기능을 추가하였으며, 해당 값은 1~999사이의 값이 임의로 들어가게 됩니다.

```

$("#plusButton").on("click",plusCal);
function plusCal(){
  if($('.matrixA_row').val() == $('.matrixB_row').val() && $('.matrixA_colume').val() == $('.matrixB_colume').val()){
    if(document.querySelectorAll('.innerBox_aRow_1 #innerBox_aColume').length == $('.matrixA_colume').val() && document.querySelectorAll('.innerBox_bRow_1 #innerBox_bColume').length == $('.matrixB_colume')
    && document.querySelectorAll('#innerBox_aRow').length == $('.matrixA_row').val() && document.querySelectorAll('#innerBox_bRow').length == $('.matrixB_row').val()){
      location.href="#matrixRTitle";
      $("#matrixR_output").empty();
      const inputValue_row = $("#matrixA_row").val();
      const inputValue_colume = $("#matrixB_colume").val();
      let matrixA_rowR = $('.matrixA_row').val();
      let matrixB_columeR = $('.matrixB_colume').val();
      $('.matrixR_row').attr('value',`${matrixA_rowR}`);
      $('.matrixR_colume').attr('value',`${matrixB_columeR}`);
      let i=0;
      while(i<inputValue_row){
        $('#matrixR_output').append(`<div class="innerBox_rRow_${i+1}" id="innerBox_rRow"></div>`);
        let j=0;
        while(j<inputValue_colume){
          $('.innerBox_rRow_${i+1}`).append(`<input class="innerBox_rColume_${j+1}" id="innerBox_rColume" type="text" oninput="handle_input(this, ${i})" readonly>`);
          let matrixA_resultCal = $('.innerBox_aRow_${i+1} .innerBox_aColume_${j+1}`).val();
          let matrixB_resultCal = $('.innerBox_bRow_${i+1} .innerBox_bColume_${j+1}`).val();
          let matrixR_plusResult = Number(matrixA_resultCal)+Number(matrixB_resultCal);
          let result = matrixR_plusResult.toString().replace(/\B(?=(\d{3})+(?! \d))/g, ',');
          $('.innerBox_rRow_${i+1} .innerBox_rColume_${j+1}`).attr('value',`${result}`);
          j++;
        }
        i++;
      }
    }
  }
}

```

### 3. 연산 버튼 실행

+연산과 -연산을 실행할 때는 2개 행렬의 행과 열의 개수가 같아야함을 이용하여 행렬 덧셈,뺄셈 규칙을 이용한 코드입니다.



```

//배열구조 잡기(A행렬, B행렬)
let matrixA_totalArray = [];
let matrixB_totalArray = [];
// A 매트릭스 배열
let i=0;
while(i<inputValue_Arow){
  let matrixA_columeArray = [];
  let j=0;
  while(j<inputValue_Acolume){
    let matrixA_resultCal = `${.innerHTML_aRow_${i+1}}.innerHTML_aColume_${j+1}`.val();
    matrixA_columeArray.push(`${matrixA_resultCal}`);
    j++;
  }
  i++;
  matrixA_totalArray.push(matrixA_columeArray);
}
// B 매트릭스 배열
i=0;
while(i<inputValue_Brow){
  let matrixB_columeArray = [];
  let j=0;
  while(j<inputValue_Bcolume){
    let matrixB_resultCal = `${.innerHTML_bRow_${i+1}}.innerHTML_bColume_${j+1}`.val();
    //console.log(matrixB_resultCal);
    matrixB_columeArray.push(`${matrixB_resultCal}`);
    j++;
  }
  i++;
  matrixB_totalArray.push(matrixB_columeArray);
}
let matrixMulti = 0;
let matrixMultiSum = 0;
i=0;
while(i<matrixA_totalArray.length){
  j=0;
  while(j<matrixB_totalArray[0].length){
    k=0;
    while(k<matrixB_totalArray.length){
      matrixMulti = matrixA_totalArray[i][k] * matrixB_totalArray[k][j];
      matrixMultiSum += matrixMulti;
      k++;
    }
    j++;
  }
  //여기서 각각 innerbox 안에 넣어주는 작업이 필요하다.
  let result = matrixMultiSum.toString().replace(/\\B(?=(\\d{3})+(?!\\d))/g, ',');
  `${.innerHTML_rRow_${i+1}}.innerHTML_rColume_${j}`.attr('value', `${result}`);
  matrixMulti = 0;
  matrixMultiSum = 0;
}
i++;

```

### 3. 연산 버튼 실행

X연산을 실행할 때는 2개 행렬 중  
첫번째 행렬의 열과 두번째  
행렬의 행의 개수가 같아야 함을  
이용하였습니다.

이후 행렬 곱셈법칙을 적용하여  
작성한 코드입니다.

# 마무리

Matrix Calculator을 디자인하고, 기능을 설계 후 실제로 코딩을 통해 구현해보면서 느낀 것이 많았습니다. 먼저 기능을 구현하기에 앞서 페이지를 접하는 사람들에게 제대로 표현하기 위해서는 사전 디자인 작업이 중요하다는 것을 알게 되었습니다.

제대로 기능을 구현하더라도 가독성이 떨어지는 디자인은 정보전달이라는 측면에서 부합하지 않다는 것을 알게되었습니다. 그러므로 사전 스케치 작업 등을 통해 실질적인 코딩 작업에 앞서 필요한 내용들을 정리하고 구상하는 작업을 해야함을 깨닫게 되었습니다.

다음으로 하나의 기능을 구현하는데에는 접근 방법이 여러가지며 정답은 없지만, 최적화된 접근 방법이 존재한다는 것을 알게 되었습니다. 최적화된 방법을 구현하기 위해서는 많은 연습과 논리적인 사고를 함양하는 노력이 필요하다는 것을 느꼈고, 이번 행렬 계산기 포트폴리오 작업을 통해 한 단계 성장할 수 있었습니다.



Thank you

