REPORT

휴먼컴퓨터인터페이스 과제 #3 직접 조작 계산기

광운대학교 KwangWoon University

개발환경: VScode

과목명	휴먼컴퓨터인터페이스
담당교수	이강훈 교수님
학과	전자융합공학과
학년	4학년
학번	2016742039
이름	정정현
제출일	21.06.12.

■개요

- 인터페이스 요구조건에 대한 구현 완성도

요구조건	요구조건 세부사항		완성도
기능적 요구조건	수식 입력	정수, 실수, 복소수의 표현과 그 기본 연산	100%
		벡터, 행렬의 표현과 그 기본 연산	
		자주 사용되는 상수 및 함수 지원	
	결과 출력	올바른 입력 => 수식 결과값 잘못된 입력 => 오류 메시지	100%
	변수, 함수 정의 및 사용	변수: 개수 제한 없음	100%
		함수: 개수 제한 없음	
인터페이스 요구조건	시각적 표현	수 (number)	100%
		연산자 (operator)	
		변수 (variable)	
		함수 (function)	
		수식 (expression)	
	대화형 상호작용	생성 (create)	100%
		이동 (move)	
		조립 (assemble)	
		실행 (execute)	
		분해 (disassemble)	
		복제 (duplicate)	
		소멸 (destroy)	
	예시 인터페이스와 차별성		100%

★주어진 요구조건을 모두 만족하였다.

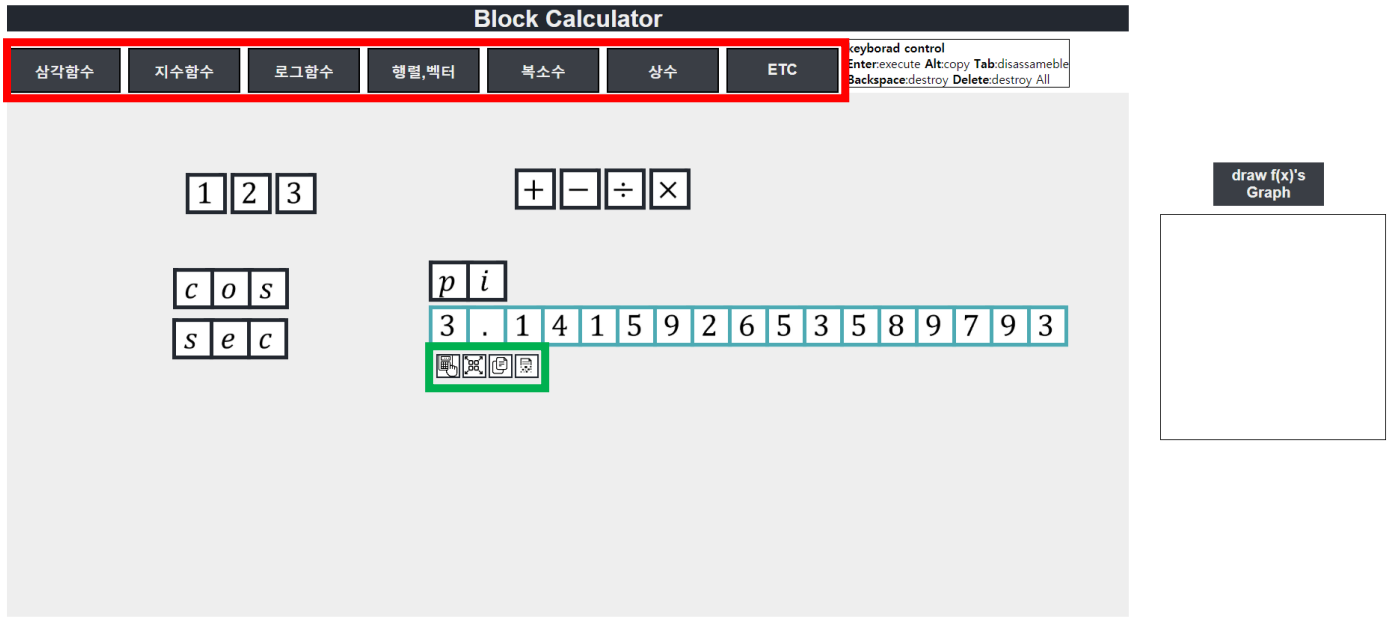
- 오픈소스 라이브러리 의존성

라이브러리 이름	사용한 이유
math.js	구버전인 math.js 에서 인식을 못하는 log2(x)와 같은 함수를 사용하기 위해 추가
plotly-latest.min.js	그래프를 그리기 위해 추가

■본문

-설계: 사용자 인터페이스의 구성 요소 및 사용 방법

설계한 직접 조작 계산기의 전체적인 사용자 인터페이스는 다음과 같습니다.



여러가지 수식의 시각적 표현을 위해 예시 인터페이스와 유사하게 블록을 사용했습니다. 블록을 생성하기 위한 방법은 **키보드를 통한 입력**과 **빨간색 네모로 둘러싸인 버튼 클릭**으로 가능합니다. 키보드 중 수식을 표현할 수 있는 키를 누른다면 canvas 를 마지막으로 클릭했던 위치에 입력한 키보드 key 값에 해당하는 블록들이 순차적으로 생성됩니다. 빨간색 네모로 둘러싸인 버튼에 마우스를 올리게 되면 버튼 이름과 관련이 있는 수식관련 함수들로 이뤄진 **드롭다운 메뉴**가 나오고 메뉴를 클릭하게 되면 해당하는 수식이 canvas 에 생성됩니다. 또한, 생성된 각각의 블록은 마우스 클릭으로 선택이 가능하고 Drag & Drop 을 통해 블록 이동이 가능합니다. 블록 이동을 통해 두개의 블록을 겹치게 되면 조합이 가능합니다.

대화형 상호작용은 **블록을 선택했을 때 나오는 팝업 메뉴**와 **키보드 입력**을 통해 상호작용이 가능합니다. 블록을 선택한 경우 선택한 블록은 경계선이 하늘색으로 변경되고 초록색 네모로 둘러싸인 부분과 같은 팝업 메뉴가 나오게 됩니다. 팝업 메뉴에 있는 각 버튼들의 기능은 다음과 같습니다.

각 팝업 버튼을 누르게 되면 선택된 블록에 대해서 각 버튼의 기능들을 수행합니다. 팝업 버튼에 있는 기능들은 키보드 입력을 통해서도 실행할 수 있습니다. 키보드 입력에 따른 기능들은 다음 표와 같습니다.

키보드 입력	수행기능
Enter	선택된 블록 계산 후 결과블록 생성
Tab	선택된 블록을 하나씩 분해
Alt	선택된 블록 복사
Backspace	선택된 블록 삭제
Del	Canvas 에 있는 모든 블록 삭제

추가로 f(x)로 정의한 함수의 경우 우측에 있는 'draw f(x)'s graph' 버튼을 통해 그래프를 그릴 수 있습니다.

-구현: 특징적인 상호작용 방식들에 대한 세부 구현 방법

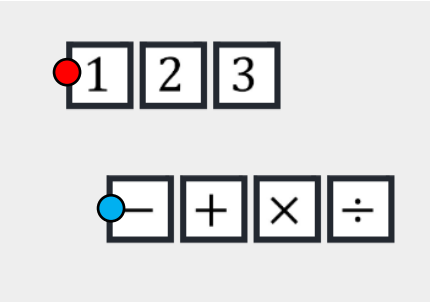
Math.js 의 기호, 수, 수식, 벡터, 행렬, 함수 등의 수식요소를 구성하기 위해서는 숫자와 알파벳뿐만 아니라 특수기호들도 표현을 해야 합니다. 수식 표현들은 각각 하나의 character 로 분리될 수 있기 때문에 Math.Block 객체를 상속하는 MathApp.Symbol 를 생성해 입력되는 character 를 resources 폴더에 있는 비트맵 이미지와 대응시켜 이미지를 canvas 에 띄우고 fabric 의 Rect 객체를 생성해 배경과 경계사각형도 생성해 하나의 블록으로 수학적 요소들을 표현했습니다.

Symbol 객체가 생성된 후에는 해당 객체를 키보드 입력을 통해 canvas 생성할 수 있게 하는 대화형 상호작용인 **'생성'**이 필요합니다. 코드는 다음과 같습니다.

```
264         MathApp.handleKeyPress = function(key) {
265             if(MathApp.selected_block != null){
266                 MathApp.selected_block.onDeselected();
267                 MathApp.selected_block = null;
268             }
269             if (key in this.symbol_paths)
270             {
271                 let size = {
272                     width : SYMBOL_WIDTH,
273                     height : SYMBOL_HEIGHT
274                 };
275                 let position
276                 if(nowXY!=lastXY){
277                     position = {
278                         x : nowXY.x,//클릭한 마우스 위치에 블럭 생성
279                         y : nowXY.y
280                     };
281                     lastXY = nowXY;
282                 }
283                 else{
284                     position = {
285                         x : nowXY.x+=60,//생성된 블럭 왼쪽에 블럭 생성
286                         y : nowXY.y
287                     };
288                 }
289                 let new_symbol = new MathApp.Symbol(position, size, key);
290             }
291         }
```

기존에 handleKeyPress 는 random 한 좌표에 symbol 들을 생성하도록 코드가 작성돼 있었습니다. 이는 사용자 입장에서 각각의 블록을 조작하기 불편할 것이라 판단해 nowXY 변수를 선언해 현재 좌표를 저장하도록 하고 lastXY 변수에는 이전에 클릭한 좌표를 저장하도록 하였습니다. 이를 통해 nowXY 가 lastXY 와 다르다면 이전에 클릭한 마우스 좌표와 현재 클릭한 마우스 좌표가 다르다는 의미이므로 nowXY 좌표에 블록을 생성하도록 하고

만약 좌표가 변경이 안되었다면 생성한 좌표 오른쪽에 블록들이 생성되도록 하였습니다. 여러 개의 블록을 클릭한 좌표에 생성시키게 되는 모습은 다음과 같습니다.



1. 빨간색 원이 있는 위치 클릭 후 키보드 1, 2, 3 입력
2. 파란색 원이 있는 위치 클릭 후 키보드 -, +, *, / 입력

생성한 블록들로 수식을 만들려면 각 블록들을 합쳐야 합니다. 블록들을 합치기 위해서는 대화형 상호작용 **'조합'**이 필요하고 블록들을 이동시킬 수 있는 대화형 상호작용인 **'이동'**도 필요합니다. 두 대화형 상호작용은 밀접한 관계를 갖고 있음으로 동시에 설명하겠습니다. '조합'과 '이동'은 마우스 조작을 통해 동작합니다. 코드는 다음과 같습니다.

```
441         MathApp.handleMouseMove = function(window_p) {
442             if(MathApp.is_mouse_dragging)
443             {
444                 let canvas_p = MathApp.transformToCanvasCoords(window_p);
445                 if(MathApp.selected_block != null)
446                 {
447                     let tx = canvas_p.x - MathApp.mouse_drag_prev.x;
448                     let ty = canvas_p.y - MathApp.mouse_drag_prev.y;
449                     MathApp.selected_block.translate({x: tx, y: ty});
450 
451                     let popupButtonTemp = [];
452 
453                     for (let i = MathApp.blocks.length - 1; i >= MathApp.blocks.length - 4; i--)
454                         popupButtonTemp.push(MathApp.blocks[i]);
455 
456                     popupButtonTemp.forEach(item => {
457                         item.translate({ x: tx, y: ty });
458                     });
459                 }
460 
461                 MathApp.mouse_drag_prev = canvas_p;
462 
463                 MathApp.canvas.requestRenderAll();
464             }
465         }
```

handleMouseMove 함수는 마우스가 눌려 있는 상태로 움직일 때 동작하는 함수입니다. 이때, 현재 drag 되고 있는 블록이 있다면 마우스의 좌표에 따라 블록을 translate 함수로 블록을 이동시키고 '이동'을 구현하게 됩니다. 여기서 451~458 라인 코드를 보면 popupButtonTemp 라는 객체를 생성하고 이를 다루는 내용이 있는데 이는 대화형 상호작용인 '실행', '분해', '복제', '소멸'을 위해 생성한 popupButton 으로 블록과 함께 움직여야 하기 때문에 작성한 코드입니다.

다음은 handleMouseUp 함수 코드입니다.

```
467         MathApp.handleMouseUp = function(window_p) {
468             if(MathApp.is_mouse_dragging)
469             {
470                 let canvas_p = MathApp.transformToCanvasCoords(window_p);
471                 //block assemble
472                 //is_intersect 함수로 경계 사각형간 중첩 검사
473                 MathApp.is_intersect();
474 
475                 if (MathApp.is_mouse_dragging) {
476                     MathApp.is_mouse_dragging = false;
477                     MathApp.mouse_drag_prev = { x: 0, y: 0 };
478                 }
479                 MathApp.canvas.requestRenderAll();
480             }
481         }
```

handleMouseUp 함수는 마우스의 버튼 클릭을 땠을 때 동작하는 함수입니다. 마우스 버튼을 땠을 때 블록이 겹친다면 '조합'을 하도록 MathApp.is_intersect() 함수를 구현했습니다. Is_intersect() 함수의 코드는 다음과 같습니다.

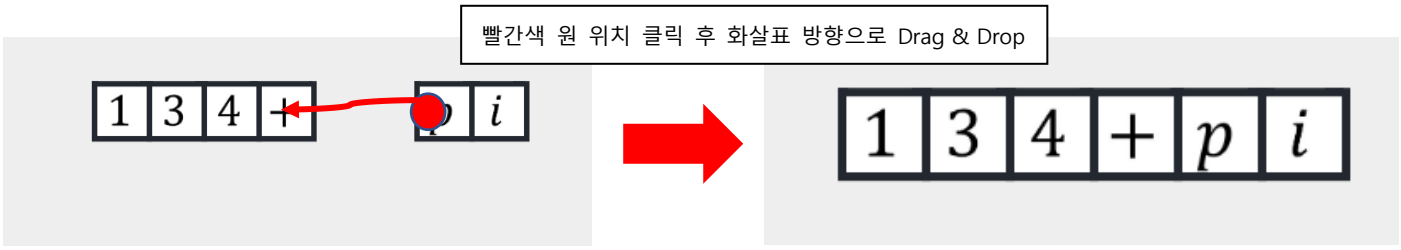
```
565         MathApp.is_intersect = function() {
566             if(MathApp.selected_block != null){
567                 let selected_block = MathApp.selected_block;
568 
569                 for(let i=0;i<this.blocks.length;i++){
570                     let check_block = this.blocks[i];
571                     let axisX = Math.abs(selected_block.position.x - check_block.position.x);
572                     let axisY = Math.abs(selected_block.position.y - check_block.position.y);
573                     if(axisX==0&&axisY==0)//원래 블럭의 경우
574                         continue;
575                     else if(axisX<=(check_block.size.width+5)&&axisY<=(check_block.size.height+5)){//두개의 축이 겹치는 경우
576                         MathApp.canvas.requestRenderAll();
577                         MathApp.assemble(i);
578                     }
579                 }
580             }
581         }
```

현재 선택한 블록이 다른 블록과 겹치는지 확인하게 위해서는 canvas 에 생성된 모든 블록의 좌표들과 비교를 해야 합니다. 생성되어 있는 블록이 저장되는 배열인 blocks 의 길이만큼 반복하는 반복문을 만들고 현재 선택한 블록의 좌표와 생성된 블록들의 좌표를 계산해 겹치는지를 검사했습니다. 계산내용은 먼저 blocks 배열에서 검사를 수행할 check_block 을 정합니다. 그리고 axisX 변수에 선택한 블록과 check_block 의 x 축 값을 빼고 절대값을 취한 수를 저장합니다. axisY 도 마찬가지로 선택한 블록과 check_block 의 y 축에 대해서 x 축에 대해 계산한 과정을 똑같이 적용해 수를 저장합니다. 이렇게 되면 두 블록 사이 간의 x, y 간격 axisX, axisY 를 알 수 있게 됩니다. 이때 axisX 가 check_block 의 너비 보다 작으면 현재 겹치는지를 확인하는 두 블록의 x 축이 겹쳐 있다는 뜻이고 axisY 가 check_block 의 높이 보다 작으면 현재 겹치는지를 확인하는 두 블록의 y 축이 겹쳐 있다는 뜻이므로 이 두개의 조건을 모두 만족한다면 두 블록이 겹쳐 있다는 것을 알 수 있습니다. 이때 주의해야 할 점은 blocks 배열을 접근하게 되면 현재 선택한 블록도 검사하게 되므로 자기자신에 대한 검사는 제외하기 위해 573 번 라인 조건문을 넣었습니다. 이를 통해 조건문을 작성하고 Math.assemble()함수를 통해 블록을 합치도록 만들었습니다. Math.assemble 함수의 내용은 다음과 같습니다.

```
MathApp.assemble = function (mainBlockIndex){
    if(MathApp.selected_block != null){
        let selected_block = MathApp.selected_block;
        let mainBlock = this.blocks[mainBlockIndex];

        let position= {
            x: mainBlock.position.x + mainBlock.size.width,
            y: mainBlock.position.y,
        };
        selected_block.moveTo(position);
        selected_block.visual_items.forEach(item => {
            mainBlock.visual_items.push(item);
        });

        mainBlock.name += selected_block.name;
        mainBlock.size.width += selected_block.size.width;

        // 골랐던 block destroy
        selected_block.destroy();

        mainBlock.visual_items.forEach(item => {
            MathApp.canvas.bringToFront(item);
        });
        MathApp.selected_block = null;
        console.log("너비 : "+mainBlock.size.width);
    }
}
```

assemble 함수는 parameter 로 blocks 배열의 인덱스 값을 받게 했습니다. 현재 선택되어 있는 블록은 selected_block 에 저장하고 parameter 로 받은 인덱스 값을 이용해 조합의 주체가 되는 blocks 배열에 있는 블록은 mainBlock 에 저장했습니다. Selected_block 은 mainBlock 의 오른쪽에 조합되야 하기 때문에 position 은 mainBlock 의 x 축 값에 mainBlock 의 width 값을 더해 x 축 값을 정하고 y 축은 mainBlock 과 같은 y 축 값을 정해 이동시켰습니다. 그리고 두 블록은 조합하게 되면 하나의 블록이 되어야 하므로 selected_block 의 모든 블록을 mainBlock 에 push 시켜주었습니다. 그런 뒤 mainBlock 에 합쳐지는 블록인 selected_block 의 name 과 width 를 mainBlock 에 더해주게 되면 두 블록이 조합되는 과정을 마치게 되고 기존 selected_block 을 삭제합니다. 상호작용의 방식에 대한 예시는 다음과 같습니다.

"134+" 블록과 pi 블록을 합쳐 "134+pi"블록을 생성하려면 pi 블록을 클릭하고 Drag & Drop을 아래 그림과 같이 수행하면 됩니다.



이렇게 '조합'을 통해 블록들을 합쳐 수식을 만들 수 있게 되었습니다.

각 블록들에 대해 수행 가능한 대화형 상호작용 '실행', '복제', '소멸', '분해'를 한 번에 다루기 위해 popupButton 클래스를 MathApp 에 생성해 주었습니다. popupButton 은 기존에 있던 코드인 symbol 을 생성하는 코드를 재사용해 정의했으므로 생성자 함수 코드 설명은 생략하겠습니다. popupButton 은 현재 선택한 블록에 대해 정의된 대화형 상호작용을 할 수 있는 버튼들입니다. 그러므로 수식 블록이 선택되었을 때 해당 블록 아래쪽에 popupButton 이 나타나야 하고 선택이 안된 블록들은 popupButton 이 나타나지 않아야 하므로 다음과 같이 코드를 작성했습니다.

```
632         MathApp.Block.prototype.onDeselected = function() {
633             if (this.visual_items.length <2) {//block 1개
634                 this.visual_items[this.visual_items.length - 1].set({
635                     stroke: "#222831"
636                 });
637             }
638             else {//block 2개 이상
639                 for (let i = this.visual_items.length - 1; i >= 0; i--)
640                     this.visual_items[i].set({
641                         stroke: "#222831"
642                     });
643             }
644             let lengthV = MathApp.blocks.length
645             for (let i = lengthV - 1; i >= lengthV - 4; i--)
646                 MathApp.blocks[i].destroy();
647         }
648 
649         MathApp.Block.prototype.onSelected = function() {
650             if(this.visual_items.length<2){//block 1개
651                 this.visual_items[this.visual_items.length-1].set({
652                     stroke: "#00adb5"
653                 });
654             }
655             else{//block 2개 이상
656                 for (let i = this.visual_items.length - 1; i >= 0; i--)
657                 this.visual_items[i].set({
658                     stroke: "#00adb5"
659                 });
660             }
661             let funcButton = ["execute","disassemble","duplicate","destroy"];
662             for(let i=0;i<4;i++){
663                 let size = {
664                     width:POPUP_WIDTH,
665                     height:POPUP_HEIGHT,
666                 }
667                 let position = {
668                     x: this.position.x+((POPUP_WIDTH+5)*i),
669                     y: this.position.y+(SYMBOL_HEIGHT+5),
670                 }
671                 let popup_btn = new MathApp.popupButton(position,size,funcButton[i]);
672             }
673             this.visual_items.forEach(item => {
674                 MathApp.canvas.bringToFront(item);
675             });
676         }
```

onSelected 함수는 블록이 선택되었을 때 선택한 블록에 대한 상호작용을 구현한 함수입니다. 기존에 작성된 함수에 여러 개의 블록이 합쳐진 블록이 선택되었을 경우에도 상호작용을 하기위해if else문을 통해 655~660라인의 코드를 추가했습니다. 그리고 앞서 블록이 선택되었을 때 popupButton이 선택한 블록 아래쪽에 나타나야 한다 하였음으로 각 버튼의 기능 이름이 저장된 funcButton 배열을 선언하고 반복문을 통해 선택한 블록 아래쪽에 popupButton들이 생성되도록 함수를 작성했습니다. onDeselected 함수는 블록의 선택의 해제되었을 때 상호작용을 나타내는 함수이므로 onSelected 함수 동작의 반대로 함수를 작성했습니다.

이렇게 시각화를 한 popupButton 의 기능을 구현하기 전에 popupButton 은 기존 Symbol 과 크기가 다르므로 이에 대해 마우스가 버튼 위에 있는지 아닌지를 판단하기 위한 findBlockOn 함수에 if else 문을 통해 다음과 같은 코드를 추가했습니다.

```
else if(block.type=='popupButton'){
    if (//경계 조건 변경
    x >= block.position.x - 15 &&
    x <= block.position.x + (15 * (2 * block.visual_items.length / 3 ) - 5) &&
    y >= block.position.y - block.size.height / 2 &&
    y <= block.position.y + block.size.height / 2) {
        return block;
    }
}
```

Block.type을 확인하고 popupButton에 대한 경계조건을 x, y 축 중심으로 width와 height를 이용하여 계산했습니다.

대화형 상호작용 '실행', '복제', '소멸', '분해'는 팝업 버튼 클릭 외에도 키보드 입력을 통해 상호작용 수행이 가능하게 하기위해 마우스를 클릭했을 때 호출되는 handleMouseDown 함수와 키보드의 키가 down 될 때 호출되는 keyControl 함수 내부에 각 상호작용에 해당되는 코드들을 조건문만 다르게 해 작성하였음으로 상호작용에 해당되는 코드에 대해서만 설명하겠습니다. 먼저, 대화형 상호작용 '실행'에 대한 코드입니다.

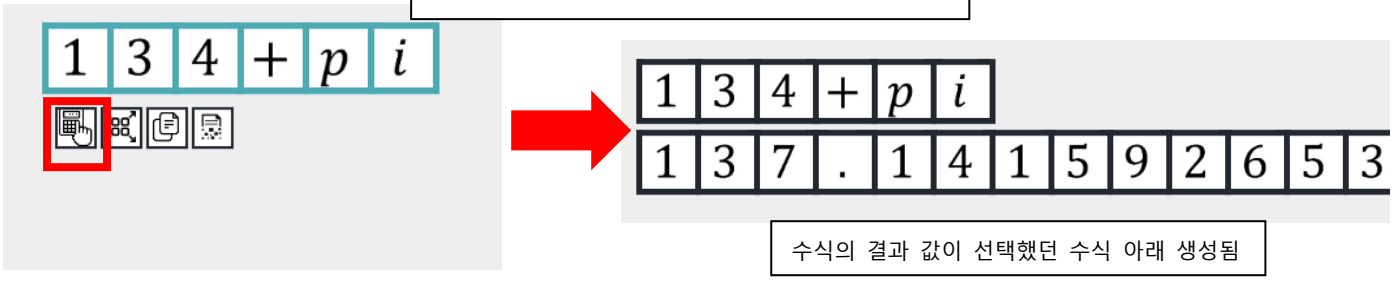
```
370                 let result
371                 if(block.type == 'popupButton'){
372                     if(block.label == 'execute'){
373                         try{
374                             let temp
375                             temp = lastBlock.name.substring(5,lastBlock.name.length);
376                             result = parser.evaluate(lastBlock.name).toString();
377                             let subBlock;
378                             let tokens = result.substring(0,8);
379                             if (tokens == 'function') {
380                                 result = tokens;
381                                 fxValue = temp;
382                             }
383                             let position={
384                                 x:lastBlock.position.x,
385                                 y:lastBlock.position.y+60,
386                             }
387                             let size={
388                                 width:SYMBOL_WIDTH,
389                                 height:SYMBOL_HEIGHT,
390                             }
391                             this.makeSeveralSymbols(result, position);
392                         }
393                         catch(e){
394                             alert('Error : '+e.message);
395                         }
396                     }
```

마우스 입력의 경우 현재 사용자가 클릭한 블록의 타입이 popupButton 인지 if 문을 통해 확인하고 해당 block 의 label 값이 'execute' 이면 '실행' 상호작용을 수행합니다. 키보드 입력의 경우 현재 사용자가 누른 key 의 값이 'Enter'이면 '실행' 상호작용을 수행합니다. 수식을 계산하는 상호작용 과정은 WebUI 계산기에서 수식을 계산하던 과정 유사하게 현재 선택한 블록의 name 값을 가져와 math.parser()로 입력된 수식을 계산하고 result 에 저장합니다. 이때 result 에 저장된 값은 여러 개의 기호가 조합된 블록이 나올 수 있으므로 여러 개의 블록을 생성하고 기존에 만들어 놓은 assemble() 함수를 사용해 합치려고 하였으나 image 객체는 바로 로드되지 않는 특성이 있어 블록들이 분리되어 생성하는 문제가 발생했습니다. 이를 해결하기 위해 makeSeveralSymbols() 함수를 만들어 결과 값을 여러 개의 기호가 조합된 블록이 출력이 되도록 하였습니다. 함수 코드는 다음과 같습니다.

```
589     MathApp.makeSeveralSymbols = function (name, position) {
590         for (let i = 0; i < name.length; i++) {
591             let newPosition = {
592                 x: position.x + SYMBOL_WIDTH * i,
593                 y: position.y,
594             }
595             this.makeOneSymbol(name[i], newPosition);
596         }
597         //setTimeout() => 일정 시간 후에 특정 함수를 의도적으로 지연한 뒤 실행할때 사용
598         // callback 함수로 visual_item 로딩 시간 대기
599         setTimeout(function () {
600             //assemble code
601             for (let i = 0; i < name.length - 1; i++) {
602                 let mainBlock = MathApp.blocks[MathApp.blocks.length - 2];//한칸 앞 블럭
603                 let subBlock = MathApp.blocks[MathApp.blocks.length - 1];//맨 뒤 블럭
604 
605                 subBlock.visual_items.forEach(item => {
606                     mainBlock.visual_items.push(item);
607                 });
608 
609                 mainBlock.name += subBlock.name;
610                 mainBlock.size.width += subBlock.size.width;
611                 subBlock.destroy();
612 
613                 mainBlock.visual_items.forEach(item => {
614                     MathApp.canvas.bringToFront(item);
615                 });
616             }
617         }, 100)
618     }
```

Parameter 로 name 과 position 을 받아 name 에 저장된 값을 한 블록 씩 생성해 준 뒤 image 객체의 로딩 시간을 대기하기위해 callback 함수 setTimeout()을 선언해 주고 setTimeout() 내부에 assemble()함수에서 사용했던 코드를 작성해 생성된 블록들을 조합했습니다. '실행'에 대한 예시는 다음과 같습니다.

'실행'을 수행할 블록을 클릭하고 블록 하단에 있는 첫번째 버튼 클릭 or Enter 키 누르기



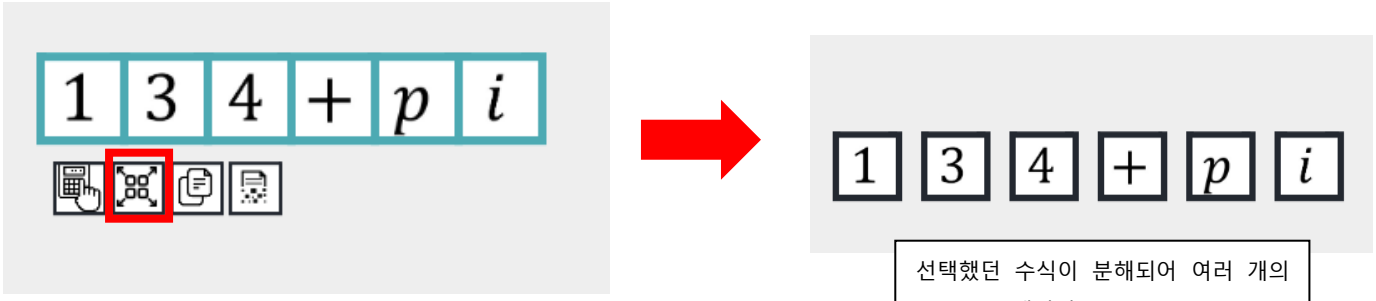
수식의 결과 값이 선택했던 수식 아래 생성됨

'분해'의 실행코드는 다음과 같습니다.

```
397     else if(block.label == 'disassemble'){
398         for(let i=0;i<lastBlock.name.length;i++){
399             let position={
400                 x:lastBlock.position.x+70*i,
401                 y:lastBlock.position.y+50
402             }
403             this.makeOneSymbol(lastBlock.name[i],position);
404         }
405         lastBlock.destroy();
406     }
```

마우스의 경우 현재 마우스가 클릭한 블록의 label 이 'disassemble'인지 확인하고 키보드 입력의 경우 현재 눌린 key 값이 'Tab'인지 확인합니다. '분해' 상호작용을 수행하기 위해 현재 선택된 블록의 name 을 가져와 한 글자 씩 분리하면서 x 좌표를 증가시켜 각 글자의 블록을 생성하고 현재 선택된 블록은 삭제하는 코드를 작성했습니다. 예시는 다음과 같습니다.

'분해'를 수행할 블록을 클릭하고 블록 하단에 있는 두번째 버튼 클릭 or Tab 키 누르기



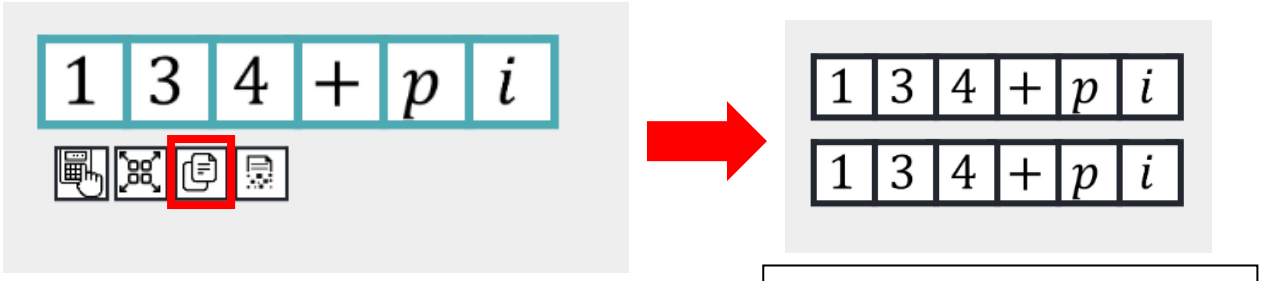
선택했던 수식이 분해되어 여러 개의 블록으로 생성됨

'복제'와 '소멸'의 실행코드는 다음과 같습니다.

```
407     else if(block.label == 'duplicate'){
408         let position={
409             x:lastBlock.position.x,
410             y:lastBlock.position.y+70,
411         }
412         this.makeSeveralSymbols(lastBlock.name,position);
413     }
414     else if(block.label=='destroy'){
415         lastBlock.destroy();
416     }
```

두 대화형 상호작용 모두 현재 마우스가 클릭한 블록의 label 을 확인하고 'duplicate'의 경우 현재 선택된 블록아래 좌표에 makeSeveralSymbols 함수로 같은 name 을 가진 블록을 만들어 선택된 블록을 복제합니다. 'destroy'의 경우 현재 선택한 블록은 destroy() 함수로 삭제합니다. 키보드 입력은 key 값이 'Alt' 이면 '복제'에 해당되는 상호작용을 수행하고 'BackSpace' 이면 '소멸'에 해당되는 상호작용을 수행합니다. 각각의 예시는 다음과 같습니다.

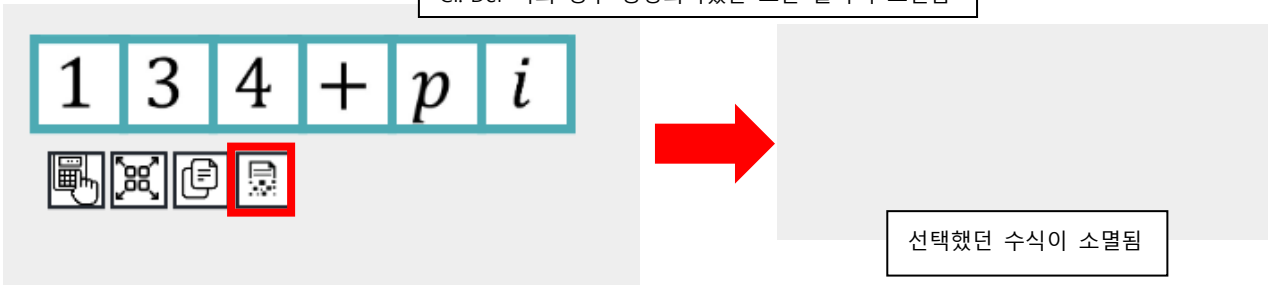
'복제'를 수행할 블록을 클릭하고 블록 하단에 있는 세번째 버튼 클릭 or Alt 키 누르기



선택했던 수식 아래에 복제된 내용이 생성됨

'소멸'을 수행할 블록을 클릭하고 블록 하단에 있는 네번째 버튼 클릭 or Backspace 키 누르기

Cf. Del 키의 경우 생성되어있는 모든 블록이 소멸됨



선택했던 수식이 소멸됨

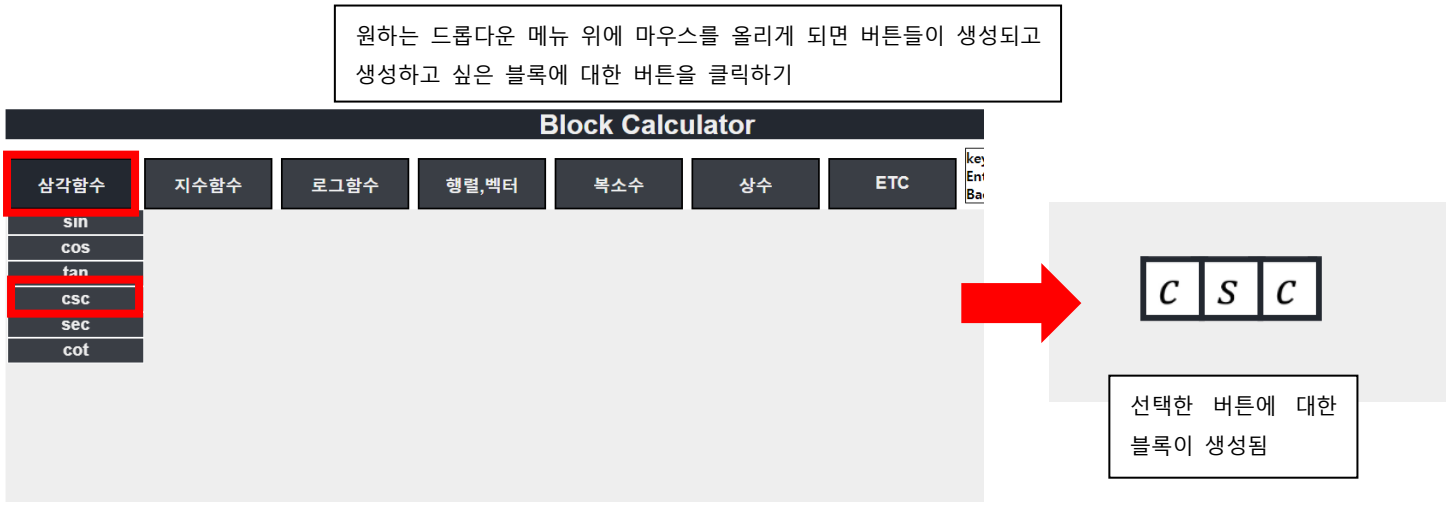
현재까지 생성한 대화형 상호작용 방식으로는 사용자 입장에서 복잡한 함수를 사용할 때 불편함이 있을 수도 있다 생각해 HTML GUI 로 자주 사용할 것 같은 함수들을 모아 놓은 드롭다운 메뉴로 만들었습니다. 드롭다운 메뉴를 클릭할 경우 메뉴의 이름과 관련 있는 함수들이 메뉴로 나오게 되고 메뉴를 클릭하면 canvas 에 해당 함수가 바로 생성되도록 했습니다. 관련 코드는 다음과 같습니다.

```
75      <!--함수 버튼 정의-->
76      <div style="align-items: center;width:1500px;height: 60px;background-color: white" >
77          <div class="dropdown" style="left: 10px;">
78              <button class="btn" style="width: 150px;height:60px">
79                  삼각함수
80              </button>
81              <div class="dropdown-content">
82                  <button class="button" onclick="functionbtnOnclick('sin')">sin</button>
83                  <button class="button" onclick="functionbtnOnclick('cos')">cos</button>
84                  <button class="button" onclick="functionbtnOnclick('tan')">tan</button>
85                  <button class="button" onclick="functionbtnOnclick('csc')">csc</button>
86                  <button class="button" onclick="functionbtnOnclick('sec')">sec</button>
87                  <button class="button" onclick="functionbtnOnclick('cot')">cot</button>
88              </div>
89          </div>
90          <div class="dropdown" style="left: 170px;">
91              <button class="btn" style="width:150px;height:60px">
92                  지수함수
```

드롭다운 메뉴의 본체가 될 div tag 에 "dropdown"(style tag 에 정의한 객체)이라는 class 를 부여하고 해당 div 의 자식 tag 로 버튼을 생성하는 과정을 반복해 드롭다운 메뉴를 구성했습니다. 각 버튼들은 onclick 함수가 functionbtnOnclick 으로 대응되어 버튼이 클릭될 경우 다음과 같은 함수를 실행하게 됩니다.

```
839     function functionbtnOnclick(name){
840         let position={
841             x: nowXY.x+200,
842             y: nowXY.y+100,
843         }
844         MathApp.makeSeveralSymbols(name,position);
845     }
```

함수는 button 으로부터 button 에 쓰여 있는 이름을 parameter 로 받아 앞서 정의한 makeSeveralSymbols 함수를 통해 canvas 에 블록을 생성합니다. 예시는 다음과 같습니다.



이전 과제 2 의 WebUI 계산기에서 그래프를 그릴 수 있게 제작한 것처럼 f(x) 함수에 대한 그래프를 그리는 기능을 추가하기 위해 plotly.js 를 추가 라이브러리로 사용하였습니다. 그래프를 그리는 코드의 내용은 다음과 같습니다.

```
164     <button class="button" style="width:150px;height:60px; position: absolute; top: 230px; left:1620px" onclick="draw()">draw f(x)'s Graph</button>
165     <div id="plot" style="border: 1px solid black; width:300px; height:300px; position:absolute; top: 300px;left: 1550px;"></div>
```

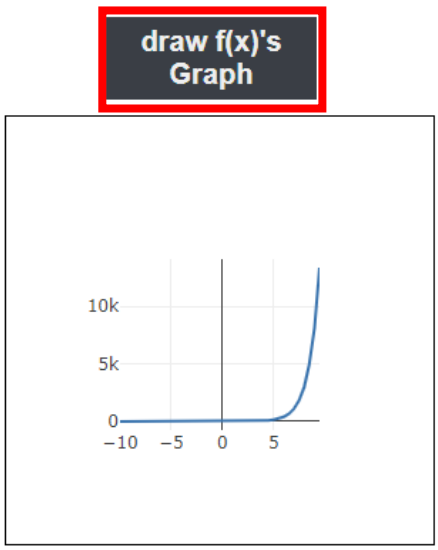
그래프를 그릴 div 와 button 을 html 요소로 생성했습니다. Button 은 onclick 함수로 draw()를 지정했습니다.

```
847         function draw(){
848             try {
849                 // compile the expression once
850                 const expression = fxValue;
851                 const expr = math.compile(expression)
852 
853                 // evaluate the expression repeatedly for different values of x
854                 const xValues = math.range(-10, 10, 0.5).toArray()
855                 const yValues = xValues.map(function (x) {
856                   return expr.evaluate({x: x})
857                 })
858 
859                 // render the plot using plotly
860                 const trace1 = {
861                   x: xValues,
862                   y: yValues,
863                   type: 'scatter'
864                 }
865                 const data = [trace1]
866                 Plotly.newPlot('plot', data);
867               }
868               catch (err) {
869                 console.error(err)
870                 alert(err)
871               }
872         }
```

draw 함수는 그래프를 그리는 함수로 저장되어있는 fxValue 를 math.js 로 계산을 하는데 x 값을 -10 부터 10 까지 증가시켜 가면서 x 값과 y 값을 얻고 Ploty 의 newPlot member function 으로 그래프를 그리도록 합니다. 예시는 다음과 같습니다.

F(x)에 대한 수식 생성해 대화형 상호작용 '실행'으로 함수 지정 후 draw f(x)'s Graph 버튼 클릭

f	(	x	)	=	e	^	x
f	u	n	c	t	i	o	n



-평가: 실제 문제에 대한 사용 예시

실제 문제에 대한 사용 예시를 보여주기 위해 5 개의 수학 문제를 만들었습니다.

1. 10 + 12 / 15 * 16 = ?

2. f(x) = 2x^3+x^2+6x+4

f(10) = ?

draw graph f(x)

3. argument of 28 + 13i & Real part of 28 + 13i

4. determinant of

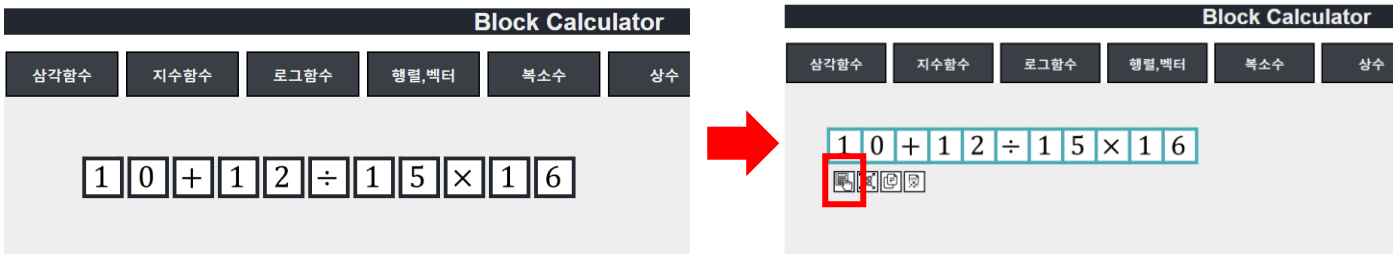
[[2, 4], [-1, 3]]

5. a=pi, b=e, c=5!

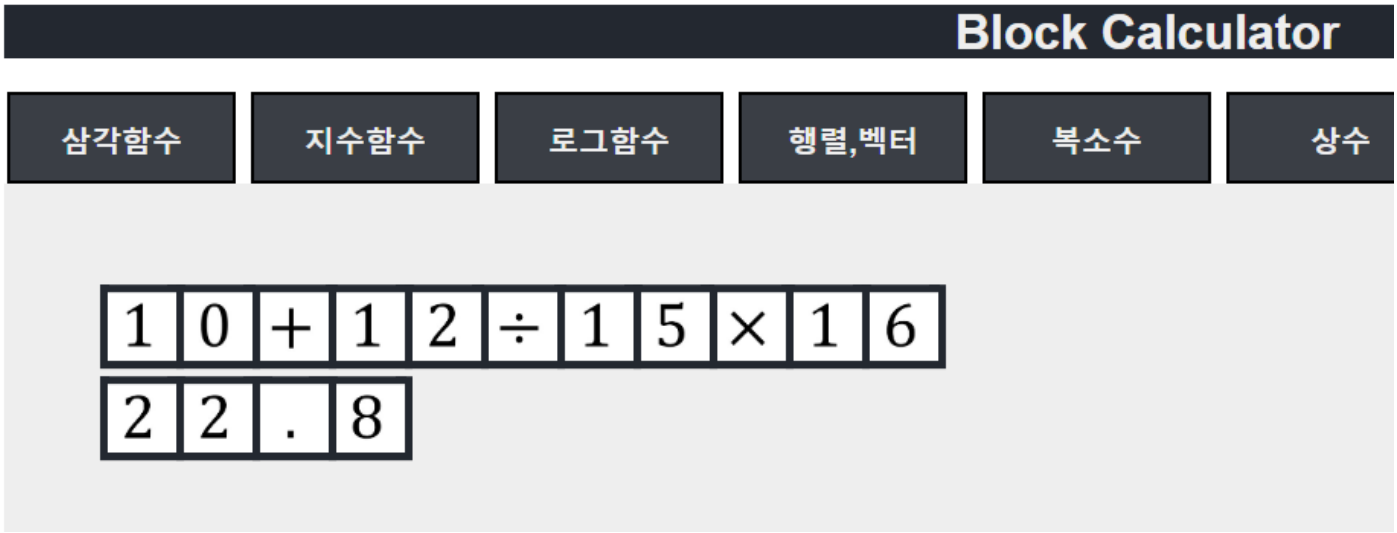
a+b+c = ?

1. 10 + 12 / 15 * 16 = ?

문제에 해당하는 요소들을 키보드로 입력해 준 뒤 Drag & Drop 을 통해 각각의 요소들을 합쳐줍니다.

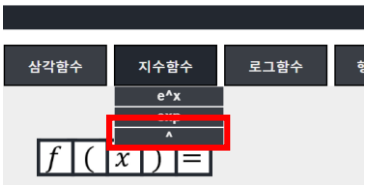


합쳐진 블록을 선택해 빨간색 사각형에 있는 버튼을 누르거나 Enter 키를 눌러 수식을 계산합니다.

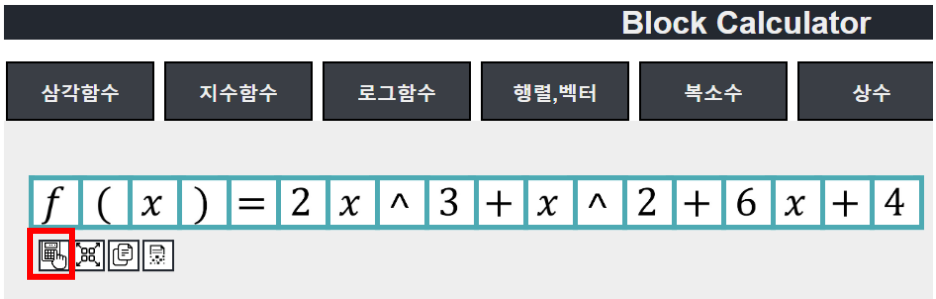


2. f(x) = 2x^3+x^2+6x+4, f(10) = ?, draw graph f(x)

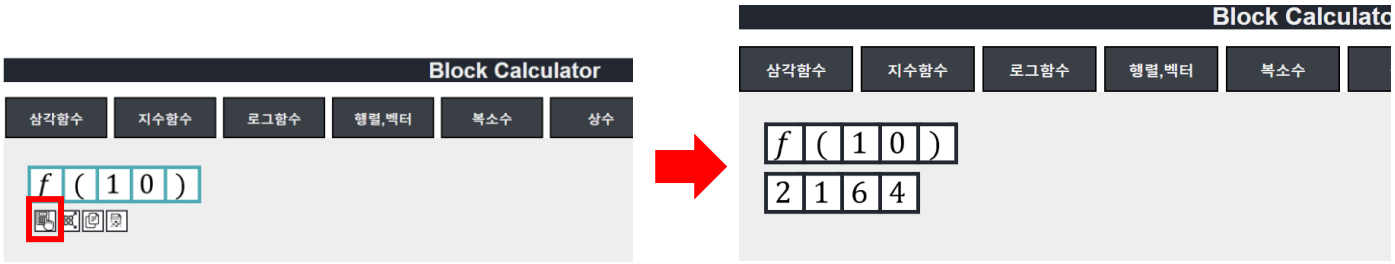
드롭다운 메뉴에 있는 f(x) 버튼을 클릭해 'f(x)=' 블록을 생성하고 주어진 수식을 키보드로 입력합니다. 제곱을 나타내는 '^'는 지수함수 드롭다운 메뉴의 버튼을 이용해 생성합니다.



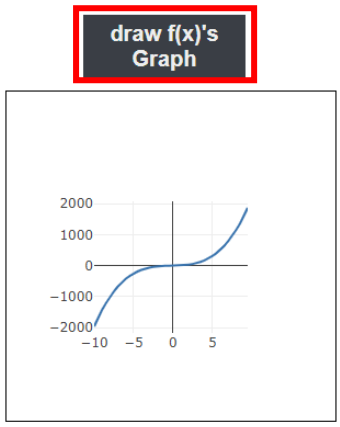
생성된 블록들을 합쳐 수식을 만들고 합쳐진 블록을 선택해 빨간색 사각형에 있는 버튼을 누르거나 Enter 키를 눌러 함수 f(x)를 정의합니다.



f(10)블록을 만들고 빨간색 사각형에 있는 버튼을 누르거나 Enter 키를 눌러 계산합니다.

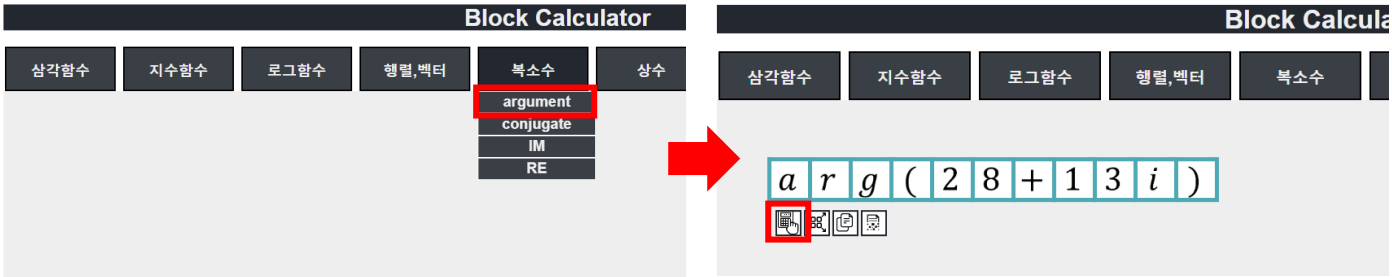


Draw f(x)'s Graph 버튼을 눌러 f(x) 그래프를 그립니다.



3. argument of 28 + 13i & Real part of 28 + 13i

복소수 드롭다운 메뉴에 있는 argument 버튼을 클릭해 'arg' 블록을 생성한 뒤 나머지 수식들을 키보드로 입력하고 합쳐줍니다.

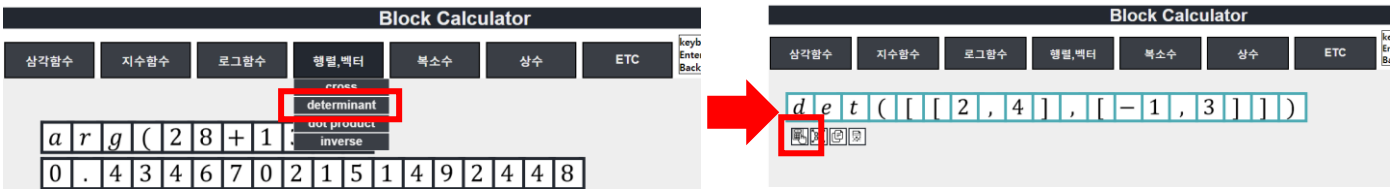


합쳐진 블록을 선택해 빨간색 사각형에 있는 버튼을 누르거나 Enter 키를 눌러 수식을 계산합니다.

Real part에 대한 문제도 동일한 과정을 수행하므로 과정설명은 생략하겠습니다.

4. determinant of [[2, 4], [-1, 3]]

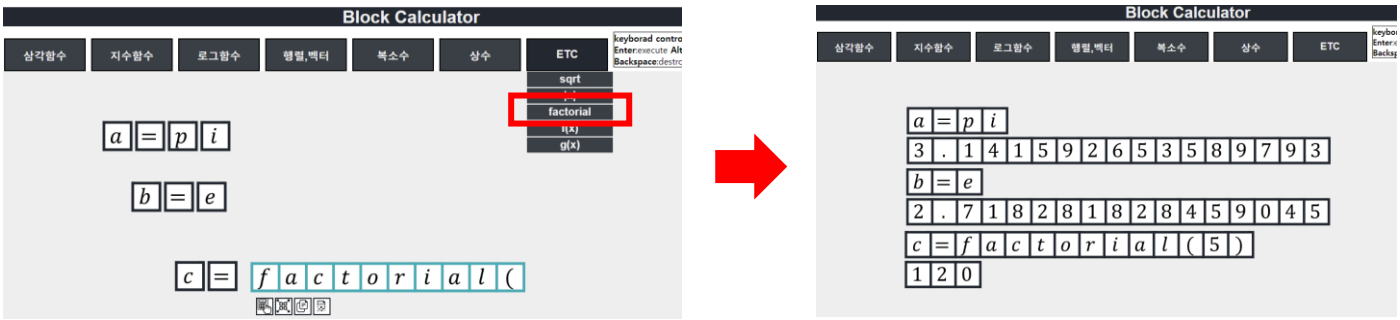
행렬, 벡터 드롭다운 메뉴에 있는 determinant 버튼을 클릭해 'det' 블록을 생성한 뒤 나머지 수식들을 키보드로 입력하고 합쳐 줍니다.



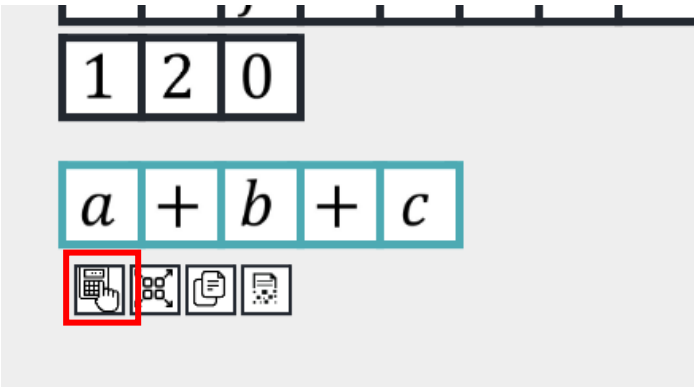
합쳐진 블록을 선택해 빨간색 사각형에 있는 버튼을 누르거나 Enter 키를 눌러 수식을 계산합니다.

5. a=pi, b=e, c=5!, a+b+c = ?

키보드로 a, b, c를 입력하고 수식을 입력하고 '실행' 팝업 버튼을 누르거나 Enter 키를 눌러 a, b, c에 값을 정해 줍니다. Factorial의 경우 ETC 드롭다운 메뉴에 있는 factorial을 클릭해 생성해 줍니다.



a, b, c에 문제에 주어진 대로 값을 주었음으로 a+b+c를 키보드로 입력하고 합쳐줍니다.



합쳐진 블록을 선택해 빨간색 사각형에 있는 버튼을 누르거나 Enter 키를 눌러 수식을 계산합니다.

5개의 예시문제를 큰 어려움 없이 풀 수 있는 것을 확인할 수 있었고 정답도 정상적으로 나오는 것을 알 수 있었습니다.

-Youtube 시연 동영상 링크

https://youtu.be/PL2O8K2HBhA

■논의

이번 과제는 직접조작 인터페이스를 이용해 Block Calculator 를 제작하는 과제였습니다. 구현 측면에서 성공적이라고 생각되는 부분은 각 블록에 대한 대화형 상호작용을 강의에서 배운 공간계산을 이용해 구현을 한 것과 HTML GUI 요소를 생성해 자주 사용하는 함수들을 편리하게 사용할 수 있도록 구현한 것입니다. 대화형 상호작용을 처음 구현할 때 배운 이론을 어떻게 코드에 적용해야 하는지 혼란스럽고 어려웠지만 '생성'을 시작으로 차근차근 구현해 나가면서 이론적인 부분과 실제적인 부분을 잘 이해할 수 있었습니다. 또한, 과제 2 에서 입력 수식에 대한 그래프를 구현할 때 plotly.js 로 그래프 이미지를 생성해 fabric.Image 에 업로드 하는 형식으로 그래프를 구현하려 했으나 이미지가 잘 업로드 되지 않아 정확한 문제점을 파악하지 못하고 다른 방식으로 구현했었습니다. 이번에도 수식에 대한 결과를 보이는 블록들이 서로 합쳐지지 않고 분리되어 나타나는 비슷한 문제가 발생하였는데 callBack 함수인 setTimeout()를 통해 블록들이 로드될 시간을 벌어 이를 해결하였고 이전 과제에서 해결하지 못했던 문제를 이번 과제에서 해결했다는 부분도 굉장히 성공적이라는 생각이 듭니다.

실패적인 부분은 원래 구현하려 했던 자동완성 기능을 구현하지 못한 것입니다. 초기에 Block Calculator 를 설계할 때 'c' 블록을 생성하면 c 로 시작하는 함수들인 'cos', 'csc', 'cross' 등... 과 같은 함수들을 아래 출력시키고 함수들을 클릭할 경우 해당하는 블록이 생성되도록 구현하려고 하였으나 정보를 찾으면서 부족한 부분이 많았고 시간이 부족한 면도 있어 구현하지 못했던 점이 아쉽습니다.

Block Calculator 의 사용성 측면에서 긍정적인 부분과 부정적인 부분을 설명하기 위해 이전에 제작한 WebUI Calculator 와 Block Calculator 의 사용성을 비교한 다음 표를 보겠습니다. 각 평가 항목의 점수는 10 점 만점입니다.

평가 항목	Block Calculator	WebUI Calculator
수식 입력 과정	7	9
활용성	8	7
정확성	6	8
유연성	10	8
학습 용이성	6	9

Block Calculator 의 부정적인 사용성 측면에 대해 먼저 설명하겠습니다. WebUI Calculator 의 경우 수식을 입력하는 과정은 원하는 입력을 버튼을 클릭해 입력하면 됩니다. 하지만, Block Calculator 의 경우 수식을 입력하는 과정이 원하는 입력에 대한 블록을 생성하고 각 블록을 조합해 줘야하는 과정이 추가되어 수식을 입력하는 측면에서 다소 불편하다는 점이 있습니다. 그리고 Block Calculator 의 경우 블록들을 합치는 과정에서 블록을 잘못 합치게 되면 모든 블록을 분해해 다시 조합하거나 새로 블록을 생성해야 하므로 사용자가 특정 목표를 완료할 때까지의 에러양인 정확성이 WebUI Calculator 보다 떨어집니다. 또한, 각 버튼에 대한 설명을 나오게 해 학습 용이성을 높인 WebUI Calculator 와 다르게 처음 사용하는 사용자 입장에서 객체를 보는 즉시 사용하는 방법을 알기 힘들어 학습 용이성이 다소 낮다고 생각합니다.

Block Calculator 의 긍정적인 사용성 측면으로는 수식을 계산하게 되면 결과 블록이 생성되고 결과블록을 계속 응용할 수 있다는 활용성 부분에서 WebUI Calculator 보다 우세하다고 생각합니다. 또한, WebUI Calculator 는 계산기에 주어진 버튼만 입력가능 하지만 Block Calculator 는 변수와 함수의 개수 제한이 없고 HTML GUI 로 주어진 특정 기능을 하는 함수 외에도 다른 함수들을 만들어 사용할 수 있음으로 유연성 부분이 긍정적인 사용성 측면이라고 생각됩니다.

이를 바탕으로 이번 과제의 결과는 다소 아쉬운 부분들이 있었지만 성공적이라고 생각합니다. 먼저, 주어진 요구조건을 모두 만족하였고 이외에도 예시 인터페이스보다 더 편리한 사용을 위해 HTML GUI 이용해 드롭다운 메뉴를 만들고 plotly.js 를 이용해 그래프를 그리는 특징적인 인터페이스와 기능을 제공하였기 때문입니다. 향후 개선 계획으로는 Block Calculator 의 부정적인 사용성 측면이었던 수식을 입력하는 과정이나 정확성 부분을 개선할 수 있는 입력 UI 를 만들고 학습용이성을 높이기 위해 WebUI Calculator 에서 각 버튼에 대한 기능이 나왔던 것과 비슷한 방식으로 블록에 대한 기능을 명시하는 부분을 구현할 것입니다. 또한, 생각하고 구현하지 못하였던 블록 생성에 따른 자동완성 기능도 추가할 계획입니다.