

휴먼컴퓨터인터페이스

HW2 - WebUI 계산기 구현

제출일 : 2021년 5월 14일

학과 : 컴퓨터소프트웨어학과

학번 : 2016726022

이름 : 이지원

요구조건과 제약조건 만족도 요약

	템플릿 프로젝트에 기초	위젯 관련 코드 입력	로그인 페이지 구성	
0 단계	○	○	○	
1 단계	템플릿 프로젝트에 기초	위젯 관련 코드 복사	레이아웃 관련 코드 입력	로그인 페이지 재구성
2 단계	1단계 결과 복사해 작업	수식 계산을 위한 객체 정의	CalcButton 클래스 정의	initWidgets() 함수 재정의
3 단계	2단계 결과 복사해 작업	보다 편리한 기능 / 인터페이스 제공		
제약 조건	클라이언트 측 스크립트만 사용	오픈소스 라이브러리 허용	추가 리소스 사용 권장	크롬 웹브라우저 호환
	○	- ○ - plotly.js , html2canvas 사용	○ - 효과음	○

0 단계

위젯 구현

0 단계 : 위젯 구현 - 위젯 기능 구현을 위한 코드 제시 및 의미 설명

```
WebUI.Widget = function() {
    this.type = WebUI.WidgetTypes.UNDEFINED;
    this.parent = null;
    this.children = [];
    this.position = {left: 0, top: 0};
    this.size = {width: 0, height: 0};

    this.is_draggable = false;
    this.is_movable = true;

    this.visual_items = [];
    this.is_resource_ready = false;

    WebUI.widgets.push(this);
}
```

Widget 객체

이 코드는 이 객체의 초기값들을 세팅하는 코드이다.
자세히 설명해보자면 type 으로 해당 위젯의 타입을 지정하고 있고, UNDEFINED 이다.
Parent 로 해당 위젯의 부모를 저장할 멤버를 null 로 세팅하고, 자식을 배열로 가진다.

Position 으로 위치를 초기화하고 있고, size 로 높이와 너비를 초기화 한다.

Is_draggable 와 moveable 변수는 위젯이 그레그와 이동이 가능한지 설정하고 있고,

Visual_items 속성으로 위젯을 구성하는 요소들을 배열로 가진다.

is_resource_ready 로 위젯의 요소가 준비되면 true 로 설정하며,

이 객체를 widget 에 추가한다.

0 단계 : 위젯 구현 - 위젯 기능 구현을 위한 코드 제시 및 의미 설명

```
//Text widget
WebUI.Text = function(label) {
    WebUI.Widget.call(this);

    this.type = WebUI.WidgetTypes.TEXT;
    this.label = label;

    this.font_family = 'System';
    this.font_size = 20;
    this.font_weight = 'bold';
    this.text_align = 'left';
    this.text_color = 'black';

}

WebUI.Text.prototype = Object.create(WebUI.Widget.prototype);
WebUI.Text.prototype.constructor = WebUI.Text;
```

Text 객체

앞의 코드와 마찬가지로 초기값들을 세팅해주고 있고, 스타일을 지정해주고 있다.

또한 안에서 Widget.call(this) 를 함으로서 위젯 생성자 를 호출한다.

0 단계 : 위젯 구현 - 위젯 기능 구현을 위한 코드 제시 및 의미 설명

```
//Text widget visualization
WebUI.Text.prototype.initVisualItems = function(){
    let text = new fabric.Text(this.label, {
        left: this.position.left,
        top: this.position.top,
        selectable: false,
        fontFamily: this.font_family,
        fontSize: this.font_size,
        fontWeight: this.font_weight,
        textAlign: this.text_align,
        stroke: this.text_color,
        fill: this.text_color
    });
    any
    let bound = text.getBoundingRect();
    this.position.left = bound.left;
    this.position.top = bound.top;
    this.size.width = bound.width;
    this.size.height = bound.height;

    this.visual_items.push(text);
    this.is_resource_ready = true;
}
//
```

Text 객체 - initVisualItems

이제 fabric.js 의 객체를 이용해 visual_items 를 구성한다. Label 을 받고 각종 속성들을 설정해주고, getBoundingClientRect() 를 이용해 텍스트의 사각형 bound 를 구하고 텍스트의 position 과 size 를 구한 후 객체의 visual_items에 저장한다.
그리고 is_resource_ready 를 true 로 해준다.

0 단계 : 위젯 구현 - 위젯 기능 구현을 위한 코드 제시 및 의미 설명

```
//Image widget
WebUI.Image = function(path, desired_size) {
    WebUI.Widget.call(this);

    this.type = WebUI.WidgetTypes.IMAGE;
    this.path = path;
    this.desired_size = desired_size;
}

WebUI.Image.prototype = Object.create(WebUI.Widget.prototype);
WebUI.Image.prototype.constructor = WebUI.Image;
```

Image 객체

앞과 마찬가지로 값을 받아 초기값을 설정해준다.

이미지 파일 경로를 받아야 하므로 이를 path에 저장해주는 것을 볼 수 있다.

0 단계 : 위젯 구현 - 위젯 기능 구현을 위한 코드 제시 및 의미 설명

```
WebUI.Image.prototype.initVisualItems = function() {
    let widget = this;

    fabric.Image.fromURL(this.path, function(img){
        if(widget.desired_size != undefined){
            img.scaleToWidth(widget.desired_size.width);
            img.scaleToHeight(widget.desired_size.height);
            widget.size = widget.desired_size;
        }
        else{
            widget.size = {width: img.width, height: img.height};
        }

        img.set({left:widget.position.left,
                 top: widget.position.top,
                 selectable: false});
        widget.visual_items.push(img);
        widget.is_resource_ready = true;
    });
}
```

Image 객체

첫 줄의 `let widget=this;` 를 통해 `this` 를 변수에 저장한다. `Fabric.js` 라이브러리로 이미지를 로딩하고, 이미지 로딩이 끝나면 콜백 함수가 호출된다. 이때 저장한 `this` 를 `callBack` 함수 내에서 지역변수 `widget`를 통해 접근 한다.

0 단계 : 위젯 구현 - 위젯 기능 구현을 위한 코드 제시 및 의미 설명

```
WebUI.PushButton.prototype.handleMouseDown = function() {  
    if(!this.is_pushed){  
        this.translate({x:0, y:5});  
        this.is_pushed = true;  
  
        if(this.onPushed != undefined){  
            this.onPushed.call(this);  
        }  
        return true;  
    }  
    else{  
        return false;  
    }  
  
WebUI.PushButton.prototype.handleMouseUp = function() {  
    if(this.is_pushed){  
        this.translate({x:0, y: -5});  
        this.is_pushed = false;  
  
        return true;  
    }  
    else{  
        return false;  
    }  
}
```

PushButton 객체

마우스로 해당 버튼을 클릭했다면, y 축으로 살짝 아래로 내려가도록 해서 시각적으로 눌린 모습을 구현한다.

마우스를 눌렀다가 떼었을 때 원래 모습으로 되돌리는 모습이다.

0 단계 : 위젯 구현 - 위젯 기능 구현을 위한 코드 제시 및 의미 설명

```
WebUI.PushButton.prototype.handleMouseDown = function() {
    if(!this.is_pushed){
        this.translate({x:0, y:5});
        this.is_pushed = true;

        if(this.onPushed != undefined){
            this.onPushed.call(this);
        }
        return true;
    }
}

WebUI.PushButton.prototype.handleMouseEnter = function() {
    this.visual_items[0].set('strokeWidth', 3);
    return true;
}

WebUI.PushButton.prototype.handleMouseExit = function() {
    this.visual_items[0].set('strokeWidth', 1);

    if(this.is_pushed){
        this.translate({x:0, y:-5});
        this.is_pushed = false;
    }
    return true;
}
```

PushButton 객체

마우스로 해당 버튼을 클릭했다면, y 축으로 살짝 아래로 내려가도록 해서 시각적으로 눌린 모습을 구현한다.

마우스가 해당 버튼 구역에 들어왔을 때 테두리를 두껍게 하여 효과를 주고, 나갔을 때는 다시 얇게 만들고, 버튼이 눌려있었다면 다시 처음으로 되돌린다.

0 단계 : 위젯 구현 - 위젯 기능 구현을 위한 코드 제시 및 의미 설명

```
WebUI.TextField.prototype.initVisualItems = function() {
    let boundary = new fabric.Rect({
        left: this.position.left,
        top: this.position.top,
        width: this.desired_size.width,
        height: this.desired_size.height,
        fill: this.fill_color,
        stroke: this.stroke_color,
        strokeWidth: this.stroke_width,
        selectable: false
    });
    let textbox = new fabric.Textbox(this.label,{
        left: this.position.left + this.margin,
        fontFamily: this.font_family,
        fontSize: this.font_size,
        fontWeight: this.font_weight,
        textAlign: this.text_align,
        stroke: this.text_color,
        fill: this.text_color,
        selectable: false
    });
    let bound = textbox.getBoundingRect();
    textbox.top = this.position.top + (this.desired_size.height - bound.height)/2;

    this.size = this.desired_size;

    this.visual_items.push(boundary);
    this.visual_items.push(textbox);

    this.is_resource_ready = true;
}
```

TextField 객체 initVisualItems

이 위젯은 전체를 구성하는 박스, 그리고 실제 텍스트가 쓰여지는 박스로 구성된다. 각각 Rect와 Textbox 를 그려준다. 각각의 위치를 조정해주고 visual_items 에 추가해준다.

0 단계 : 위젯 구현 - 위젯 기능 구현을 위한 코드 제시 및 의미 설명

```
WebUI.Switch = function(is_on, desired_size) {
    WebUI.Widget.call(this);
    this.type = WebUI.WidgetTypes.SWITCH;

    this.is_on = is_on;
    this.desired_size = desired_size;
    this.stroke_width = 1;
    this.radius = desired_size.width / 4;

    this.fill_color = 'rgb(142,142,147)';
    this.stroke_color = 'rgb(142,142,147)';
    this.offColor = 'rgb(142, 142, 147)';
    this.onColor = 'rgb(48, 209, 88)';
}
```

Switch 객체 - 생성자

생성자이다. 켜져있는지 여부를 초기화하고 여러가지 속성을 지정하고, 색깔을 회색으로 지정해둔다. onColor 와 offColor 를 둔다.

0 단계 : 위젯 구현 - 위젯 기능 구현을 위한 코드 제시 및 의미 설명

```
WebUI.Switch.prototype.initVisualItems = function() {
    let radius = this.desired_size.width/4;

    let leftRound = new fabric.Circle({
        left: this.position.left,
        top: this.position.top,
        radius: radius,
        fill: this.fill_color,
        strokeWidth: this.stroke_width,
        selectable: false
    });

    let rightRound = new fabric.Circle({
        left: this.position.left + radius*2,
        top: this.position.top,
        radius: radius,
        fill: this.fill_color,
        strokeWidth: this.stroke_width,
        selectable: false
    });

    let centerRect = new fabric.Rect({
        left: this.position.left + radius,
        top: this.position.top,
        width: radius*2,
        height: radius*2,
        fill: this.fill_color,
        strokeWidth: this.stroke_width,
        selectable: false
    });

    let circle = new fabric.Circle({
        left: this.position.left + radius*0.1,
        top: this.position.top + radius*0.1,
        radius: this.radius*0.9,
        fill: 'white',
        selectable: false
    });

    if (this.is_on) {
        circle.left += radius*2;
    }

    this.size = this.desired_size;
    this.visual_items.push(leftRound);
    this.visual_items.push(rightRound);
    this.visual_items.push(centerRect);
    this.visual_items.push(circle);
    this.is_resource_ready = true;
}
```

Switch 객체 - initVisualItems

Switch 위젯의 initVisualItems() 함수이다. Switch 위젯의 Canvas 요소 구성은 안에 있는 원과 반원 2개, 사각형으로 구성된다. 처음엔 반원을 그리려다가 생각해보니 그냥 원을 그려도 상관이 없었다. 요소들을 그린뒤 visual_items 에 이들을 넣어주고 is_resource_ready 를 true 로 변경해주었다.

0 단계 : 위젯 구현 - 위젯 기능 구현을 위한 코드 제시 및 의미 설명

```
WebUI.Switch.prototype.handleMouseDown = function() {
    var canvas = WebUI.canvas;

    if (this.is_on == true) {
        this.visual_items[0].set('fill', this.offColor);
        this.visual_items[1].set('fill', this.offColor);
        this.visual_items[2].set('fill', this.offColor);

        this.visual_items[3].animate('left', '-=50', {
            onChange: canvas.requestRenderAll.bind(canvas),
            duration: 120
        });
        this.is_on = false;
    } else {
        this.visual_items[0].set('fill', this.onColor);
        this.visual_items[1].set('fill', this.onColor);
        this.visual_items[2].set('fill', this.onColor);

        this.visual_items[3].animate('left', '+=50', {
            onChange: canvas.requestRenderAll.bind(canvas),
            duration: 120
        });
        this.is_on = true;
    }
    return true;
}
```

Switch 객체 - handleMouseDown

HandleMouseDown 함수이다. is_on 에 따라 색깔을 변경하고 좌우로 움직이게 해주었다.

0 단계 : 위젯 구현 - 로그인 페이지 구성을 위한 코드 제시 및 결과 분석

```
WebUI.initWidgets = function() {
    WebUI.title = new WebUI.Text("introduction to HCI");

    WebUI.img_html = new WebUI.Image("resources/HTML5.png", {width: 100, height: 80});
    WebUI.img_css = new WebUI.Image("resources/CSS3.png", {width:100, height:80});
    WebUI.img_js = new WebUI.Image("resources/Javascript.png", {width: 100, height: 80});

    WebUI.text_id = new WebUI.Text("ID");
    WebUI.text_pwd = new WebUI.Text("Password");

    WebUI.edit_id = new WebUI.TextField("", {width: 200, height:50});
    WebUI.edit_pwd = new WebUI.TextField("", {width:200, height:50});

    WebUI.btn_ok = new WebUI.PushButton("OK", {width:100, height:50});
    WebUI.btn_cancel = new WebUI.PushButton("Cancel", {width:100, height:50});
    //
    WebUI.text_blaah = new WebUI.Text();
    WebUI.switch = new WebUI.Switch();
}

WebUI.initVisualItems = function()
{
    WebUI.widgets.forEach(widget =>
        widget.initVisualItems();
    );
}
```

```
    WebUI.layoutWidgets = function() {
        WebUI.title.moveTo({left: 100, top: 10});

        WebUI.img_html.moveTo({left: 50, top: 50});
        WebUI.img_css.moveTo({left: 160, top:50});
        WebUI.img_js.moveTo({left: 270, top: 50});

        WebUI.text_id.moveTo({left: 50, top: 160});
        WebUI.text_pwd.moveTo({left: 50, top: 220});

        WebUI.edit_id.moveTo({left: 150, top: 140});
        WebUI.edit_pwd.moveTo({left: 150, top: 200});

        WebUI.text_blaah.moveTo({left: 50, top: 300});
        WebUI.switch.moveTo({left: 250, top: 280});

        WebUI.btn_ok.moveTo({left: 50, top: 350});
        WebUI.btn_cancel.moveTo({left: 160, top: 350});
    }
}
```

initWidgets & layoutWidgets

위젯들을 캔버스에 어떤 크기와 모습으로 그릴지 initWidgets에서 정의한다. 생성자들을 호출하면서 객체를 생성한다. 그리고 for 문을 돌면서 initVisualItem 함수를 호출하고 layoutWidget 함수를 이용해서 객체들을 배치한다.

0 단계 : 위젯 구현 - 로그인 페이지 구성을 위한 코드 제시 및 결과 분석

introduction to HCI

HTML CSS JS

ID

Password

I want to get A+!

OK Cancel

introduction to HCI

HTML CSS JS

ID

Password

I want to get A+!

OK Cancel

introduction to HCI

HTML CSS JS

ID

Password

I want to get A+!

OK Cancel

결과 분석

성공적으로 스위치가 움직이고 나타나며, 텍스트 박스에서도 글자가 박스를 넘어가지 않는 것을 볼 수 있다.
결과는 정상적으로 나오는 것을 볼 수 있지만 절대 레이아웃으로 그려져 변경이 어렵다는 큰 단점이 있다.

1 단계

레이아웃 구현

1 단계 : 레이아웃 구현 - 레이아웃 관련 코드 입력

```
WebUI.WidgetTypes = {  
    UNDEFINED: "undefind",  
    TEXT: "text",  
    IMAGE: "image",  
    PUSH_BUTTON: "push_button",  
    TEXT_FIELD: "text_field",  
    SWITCH: "switch",  
    CONTAINER: "container",  
    ROW: "row",  
    COLUMN: "column"  
};  
  
WebUI.Alignment = {  
    CENTER: "center",  
    LEFT: "left",  
    RIGHT: "right",  
    TOP: "top",  
    BOTTOM: "bottom"  
};
```

WidgetType 과 Alignment

레이아웃 구현을 위해 Container, Row, Column 타입을
추가정의하고, 위치를 지정하기 위해 Alignment 도 정의하였
다.

1 단계 : 레이아웃 구현 - 레이아웃 관련 코드 입력

```
WebUI.maxSize = function(size1, size2) {
    let max_size = {width: 0, height: 0};
    max_size.width = (size1.width > size2.width) ?
        size1.width : size2.width;
    max_size.height = (size1.height > size2.height) ?
        size1.height : size2.height;
    return max_size;
}

WebUI.minSize = function(size1, size2) {
    let min_size = {width: 0, height: 0};
    min_size.width = (size1.width < size2.width) ?
        size1.width : size2.width;
    min_size.height = (size1.height < size2.height) ?
        size1.height : size2.height;
    return min_size;
}
```

maxSize 함수과 minSize 함수 구현

위젯의 크기를 측정할 때 사용될 maxSize 함수와 minSize 함수를 구현하였다. maxSize는 높이와 너비 중 큰 크기를 반환하고 minSize는 작은 크기를 반환한다.

1 단계 : 레이아웃 구현 - 레이아웃 관련 코드 입력

```
if (properties != undefined) {
  for (let name in properties) {
    let value = properties[name];
    if (name == 'children') {
      value.forEach(child => {
        child.parent = this;
        this.children.push(child);
      });
    } else {
      this[name] = value;
    }
  }
}
```

Widget 객체 정의 확장

위젯 객체 생성시 프로퍼티를 인자로 받아서 원하는 속성을 설정하게 하는 코드이다. 속성들이 전달되었을 때 속성 이름이 children 이면 자식들과의 관계를 지어주고 아니면 속성을 전달해서 적용시킨다.

1 단계 : 레이아웃 구현 - 레이아웃 관련 코드 입력

```
WebUI.Widget.prototype.measure = function() {
  if (this.children.length > 0) {
    this.size_children = {width: 0, height:0};
    this.children.forEach(child => {
      let size_child = child.measure();
      this.size_children = this.extendSizeChildren(this.size_children, size_child);
    });
    this.size = WebUI.maxSize(this.desired_size, this.size_children);
  }
  else {
    this.size.width += this.padding * 2;
    this.size.height += this.padding * 2;
  }
  return this.size;
}
```

measure 함수

이 함수는 위젯의 크기를 측정해서 리턴한다.
자식이 없을 때는 위젯 크기를 기본 크기와 양쪽 패딩을 더한
값을 리턴하고 자식이 있을 때는 자식의 크기를 재귀적으로 측
정해서 큰 값을 리턴한다.

1 단계 : 레이아웃 구현 - 레이아웃 관련 코드 입력

```
WebUI.Widget.prototype.arrange = function(position) {
    this.moveTo(position);
    this.visual_items.forEach(item => { WebUI.canvas.add(item); });

    //arrange children
    if(this.children.length > 0) {
        let left_spacing = 0, top_spacing = 0;

        if(this.size.width > this.size_children.width) {
            let room_width = this.size.width - this.size_children.width;

            if (this.horizontal_alignment == WebUI.Alignment.LEFT)
                left_spacing = this.padding;
            else if (this.horizontal_alignment == WebUI.Alignment.CENTER)
                left_spacing = this.padding + room_width /2.0;
            else if (this.horizontal_alignment == WebUI.Alignment.RIGHT)
                left_spacing = this.padding + room_width;
        }

        if(this.size.height > this.size_children.height) {
            let room_height = this.size.height - this.size_children.height;

            if (this.vertical_alignment == WebUI.Alignment.TOP)
                top_spacing = this.padding;
            else if (this.vertical_alignment == WebUI.Alignment.CENTER)
                top_spacing = this.padding + room_height /2.0;
            else if (this.vertical_alignment == WebUI.Alignment.BOTTOM)
                top_spacing = this.padding + room_height;
        }

        let next_position = {left: position.left + left_spacing,
                            top : position.top + top_spacing};
        this.children.forEach(child => {
            child.arrange(next_position);
            next_position = this.calcNextPosition(next_position, child.size);
        });
    }

    WebUI.Widget.prototype.layout = function() {
        this.measure();
        this.arrange(this.position);
    }
}
```

Arrange 함수

위젯과 자식을 위치에 그려주는 함수이다. 먼저 위젯을 이동시킨 후 visual_item 을 순차적으로 그려간다. 좌표들을 자식의 유무와 위젯의 사이즈를 정의하고 재귀적으로 자식들에게도 이 함수를 호출하게 한다.

1 단계 : 레이아웃 구현 - 레이아웃 관련 코드 입력

```
WebUI.Container.prototype.extendSizeChildren = function(size, child_size) {
    if (size.width < child_size.width) size.width = child_size.width;
    if (size.height < child_size.height) size.height = child_size.height;
    return size;
}

WebUI.Container.prototype.calcNextPosition = function(position, size) {
    let next_left = position.left;
    let next_top = position.top;
    return {left: next_left, top: next_top};
}

WebUI.Column.prototype.extendSizeChildren = function(size, child_size) {
    size.width += child_size.width;
    if (size.height < child_size.height) size.height = child_size.height;
    return size;
}

WebUI.Column.prototype.calcNextPosition = function(position, size) {
    let next_left = position.left + size.width;
    let next_top = position.top;
    return {left: next_left, top: next_top};
}

WebUI.Row.prototype.extendSizeChildren = function(size, child_size) {
    if (size.width < child_size.width) size.width = child_size.width;
    size.height += child_size.height;
    return size;
}

WebUI.Row.prototype.calcNextPosition = function(position, size) {
    let next_left = position.left;
    let next_top = position.top + size.height;
    return {left: next_left, top: next_top};
}
```

extendSizeChildren & CalcNextPosition

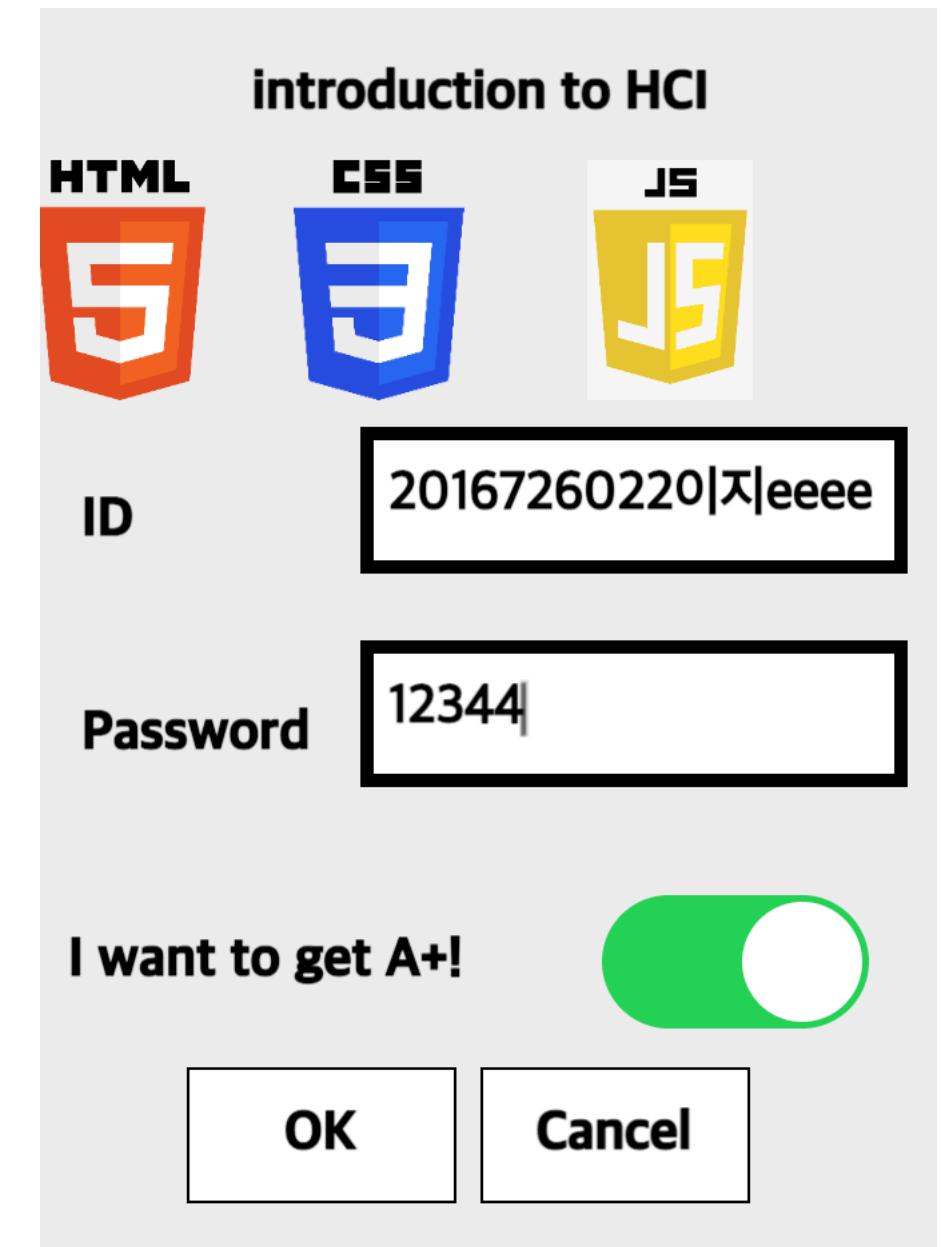
레이아웃이 자식들의 크기가 레이아웃 크기를 벗어나지 않는지를 확인하는 함수인 extendSizeChildren 함수와 레이아웃 안에서 다음 자식이 어디에 위치되어야 하는지 지정해주는 함수인 CalcNextPosition를 각 객체에 맞게 구현한다.

1 단계 : 레이아웃 구현 - 로그인 페이지 재구성

```
WebUI.initWidgets = function() {
    WebUI.app = new WebUI.Row({
        children: [
            new WebUI.Container({
                desired_size: {width: 350, height: 60},
                horizontal_alignment: WebUI.Alignment.CENTER,
                vertical_alignment: WebUI.Alignment.CENTER,
                padding: 10,
                children: [ new WebUI.Text("introduction to HCI") ]
            }),
            new WebUI.Column({
                children: [
                    new WebUI.Column({
                        children: [
                            new WebUI.Image("img/html5.png"),
                            new WebUI.Image("img/css3.png"),
                            new WebUI.Image("img/js.png")
                        ]
                    }),
                    new WebUI.Column({
                        children: [
                            new WebUI.Container({
                                desired_size: {width: 20, height: 80},
                                children: [ new WebUI.Text("", {width: 100, height: 90}) ]
                            }),
                            new WebUI.Container({
                                desired_size: {width: 110, height: 60},
                                horizontal_alignment: WebUI.Alignment.LEFT,
                                vertical_alignment: WebUI.Alignment.CENTER,
                                padding: 10,
                                children: [ new WebUI.Text("Password", {width: 100, height: 80}) ]
                            }),
                            new WebUI.Container({
                                desired_size: {width: 300, height: 50},
                                horizontal_alignment: WebUI.Alignment.LEFT,
                                vertical_alignment: WebUI.Alignment.CENTER,
                                children: [ new WebUI.TextField("", {width: 200, height: 50}) ]
                            })
                        ]
                    })
                ],
                desired_size: {width: 350, height: 120},
                horizontal_alignment: WebUI.Alignment.CENTER,
                vertical_alignment: WebUI.Alignment.CENTER,
                padding: 10,
                children: [
                    new WebUI.Column({
                        children: [
                            new WebUI.Container({
                                desired_size: {width: 20, height: 80},
                                children: [ new WebUI.Text("", {width: 100, height: 90}) ]
                            }),
                            new WebUI.Container({
                                desired_size: {width: 200, height: 80},
                                horizontal_alignment: WebUI.Alignment.RIGHT,
                                vertical_alignment: WebUI.Alignment.CENTER,
                                children: [ new WebUI.PushButton("OK", {width: 100, height: 50}) ]
                            }),
                            new WebUI.Container({
                                desired_size: {width: 175, height: 60},
                                horizontal_alignment: WebUI.Alignment.LEFT,
                                vertical_alignment: WebUI.Alignment.CENTER,
                                children: [ new WebUI.PushButton("Cancel", {width: 100, height: 50}) ]
                            })
                        ]
                    })
                ]
            })
        ]
    })
}
```

로그인 화면 재구성

앞서 구현한 위젯을 이용하여 앞서 절대 레이아웃으로 구현했던 코드를 변경하였다. 다음은 결과 화면이다.



2 단계

기본 계산기 구현

2 단계 : 기본 계산기 구현 - CalcButton 클래스 정의

결과 분석

```
WebUI.CalcButton = function(label, desired_size, properties) {
    WebUI.PushButton.call (this, label, desired_size, properties);

    this.type = WebUI.WidgetTypes.CalcBUTTON;
    this.onPushed = this.handleButtonPushed;
}

WebUI.CalcButton.prototype = Object.create(WebUI.PushButton.prototype);
WebUI.CalcButton.prototype.constructor = WebUI.CalcButton;
```

성공적으로 스위치가 움직이고 나타나며, 텍스트 박스에서도 글자가 박스를 넘어가지 않는 것을 볼 수 있다.
결과는 정상적으로 나오는 것을 볼 수 있지만 절대 레이아웃으로 그려져 변경이 어렵다는 큰 단점이 있다.

2 단계 : 기본 계산기 구현 - CalcButton 클래스 정의

```
WebUI.parser = math.parser();
```

```
WebUI.CalcButton = function(label, desired_size, properties) {
    WebUI.PushButton.call(this, label, desired_size, properties);

    this.type = WebUI.WidgetTypes.CalcBUTTON;
    this.onPushed = this.handleButtonPushed;
}

WebUI.CalcButton.prototype = Object.create(WebUI.PushButton.prototype);
WebUI.CalcButton.prototype.constructor = WebUI.CalcButton;
```

CalcButton 클래스 정의

Parser 를 정의하고

PushButton 을 상속받는 CalcButton 클래스이다.

2 단계 : 기본 계산기 구현 - CalcButton 클래스 정의

```
WebUI.CalcButton.prototype.handleButtonPushed = function() {
    var btn_text = this.visual_items[1].text;

    if (expression == '0') {
        expression = '';
    }

    switch(btn_text) {
        case 'CL':
            expression = '0';
            WebUI.initialize();
            break;

        case 'EV':
            try {
                let result = WebUI.parser.eval(expression).toString();
                expression = result;
                var tokens = result.split(' ');

                if(tokens[0] == 'function') {
                    expression = tokens[0];
                }

                WebUI.initialize();
                expression = '0';

            } catch(e) {
                expression = '0';
                if(expression != 'function') {
                    expression = e.message;
                    WebUI.initialize();
                }
            }
            break;

        default:
            expression += btn_text;
            WebUI.initialize();
            break;
    }
}
```

handleButtonPushed

CalcButton handleButtonPushed 함수를 정의해서 사용할 수 있게 한다. 함수의 큰 틀을 설명하자면 parser 를 사용해서 결과값이 에러라면 에러문을 출력해주고 에러가 아니라면 결과 값을 텍스트 박스에 표시해준다. 만약 CL 이나 EV 둘 중 아무것도 아니면 텍스트 뒤에 이어 붙여준다.

2 단계 : 기본 계산기 구현 - initWidget()

```
WebUI.initWidgets = function() {
    WebUI.app = new WebUI.Row({
        children: [
            new WebUI.Container({
                desired_size: {width: 800, height: 80},
                horizontal_alignment: WebUI.Alignment.CENTER,
                vertical_alignment: WebUI.Alignment.CENTER,
                children: [ new WebUI.Text("WebUI Calculator", 'rgb(1,0,255)') ]
            }),
            new WebUI.Container({
                desired_size: {width: 800, height: 80},
                horizontal_alignment: WebUI.Alignment.CENTER,
                children: [
                    new WebUI.BackGround_Container('white',{
                        desired_size: {width: 550, height: 50},
                        horizontal_alignment: WebUI.Alignment.LEFT,
                        vertical_alignment: WebUI.Alignment.CENTER,
                        children: [
                            new WebUI.Text('expression.toString() "black"')
                        ]
                    })
                ]
            })
        ],
        children: [
            new WebUI.Container({
                desired_size: {width: 800, height: 600},
                horizontal_alignment: WebUI.Alignment.CENTER,
                vertical_alignment: WebUI.Alignment.TOP,
                children: [
                    new WebUI.Row({
                        children: [
                            new WebUI.Column({
                                children: [
                                    new WebUI.CalcButton("1", { width: 50, height: 50 }),
                                    new WebUI.CalcButton("2", { width: 50, height: 50 }),
                                    new WebUI.CalcButton("3", { width: 50, height: 50 }),
                                    new WebUI.CalcButton("4", { width: 50, height: 50 }),
                                    new WebUI.CalcButton("5", { width: 50, height: 50 }),
                                    new WebUI.CalcButton("6", { width: 50, height: 50 }),
                                    new WebUI.CalcButton("7", { width: 50, height: 50 }),
                                    new WebUI.CalcButton("8", { width: 50, height: 50 }),
                                    new WebUI.CalcButton("9", { width: 50, height: 50 }),
                                    new WebUI.CalcButton("0", { width: 50, height: 50 })
                                ]
                            })
                        ]
                    })
                ]
            })
        ]
    })
}
```

```
new WebUI.Column({
    children: [
        new WebUI.CalcButton("(", { width: 50, height: 50 }),
        new WebUI.CalcButton(")", { width: 50, height: 50 }),
        new WebUI.CalcButton("[", { width: 50, height: 50 }),
        new WebUI.CalcButton("]", { width: 50, height: 50 }),
        new WebUI.CalcButton(".", { width: 50, height: 50 }),
        new WebUI.CalcButton(",", { width: 50, height: 50 }),
        new WebUI.CalcButton(":", { width: 50, height: 50 }),
        new WebUI.CalcButton(";", { width: 50, height: 50 }),
        new WebUI.CalcButton("==", { width: 50, height: 50 }),
        new WebUI.CalcButton("!=", { width: 50, height: 50 })
    ]
}),
new WebUI.Column({
    children: [
        new WebUI.CalcButton("i", { width: 50, height: 50 }),
        new WebUI.CalcButton("e", { width: 50, height: 50 }),
        new WebUI.CalcButton("pi", { width: 50, height: 50 }),
        new WebUI.CalcButton("w", { width: 50, height: 50 }),
        new WebUI.CalcButton("x", { width: 50, height: 50 }),
        new WebUI.CalcButton("y", { width: 50, height: 50 }),
        new WebUI.CalcButton("z", { width: 50, height: 50 }),
        new WebUI.CalcButton("f", { width: 50, height: 50 }),
        new WebUI.CalcButton("g", { width: 50, height: 50 }),
        new WebUI.CalcButton("=", { width: 50, height: 50 })
    ]
}),
new WebUI.Column({
    children: [
        new WebUI.CalcButton("exp", { width: 50, height: 50 }),
        new WebUI.CalcButton("log", { width: 50, height: 50 }),
        new WebUI.CalcButton("sqrt", { width: 50, height: 50 }),
        new WebUI.CalcButton("sin", { width: 50, height: 50 }),
        new WebUI.CalcButton("cos", { width: 50, height: 50 }),
        new WebUI.CalcButton("tan", { width: 50, height: 50 }),
        new WebUI.CalcButton("cross", { width: 50, height: 50 }),
        new WebUI.CalcButton("det", { width: 50, height: 50 }),
        new WebUI.CalcButton("CL", { width: 50, height: 50 }),
        new WebUI.CalcButton("EV", { width: 50, height: 50 })
    ]
}),
```

InitWidget() 함수를 정의하였다. 여기서 큰 틀을 보자면 큰 Row 안에 Container들을 담아 정렬이 맞도록 한다. 또한 텍스트 박스의 BackgroundContainer 를 정의하여 사용하였다. Container 안에는 Column 속에 CalcButton 들을 넣어 표시되도록 하였다.

2 단계 : 기본 계산기 구현 - 결과화면

WebUI Calculator

0

1	2	3	4	5	6	7	8	9	0
+	-	*	/	%	^	<	>	<=	>=
()	[]	.	,	:	;	==	!=
i	e	pi	w	x	y	z	f	g	=
exp	log	sqrt	si						

WebUI Calculator

WebUI Calculator

34+20

1	2	3	4	5	6	7	8	9	0
+	-	*	/	%	^	<	>	<=	>=
()	[]	.	,	:	;	==	!=
i	e	pi	w	x	y	z	f	g	=
		sqrt	sin	cos	tan	cross	det		

WebUI Calculator

값 입력

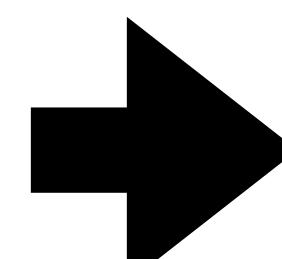
초기화면

54

1	2	3	4	5	6	7	8	9	0
+	-	*	/	%	^	<	>	<=	>=
()	[]	.	,	:	;	==	!=
i	e	pi	w	x	y	z	f	g	=
exp	log	sqrt	sin	cos	tan	cross	det	CL	EV

결과값 출력

결과가 출력된 후
다른 숫자를 눌렀을 때 결과값에
이어지는 것이 아닌 새롭게
다시 시작



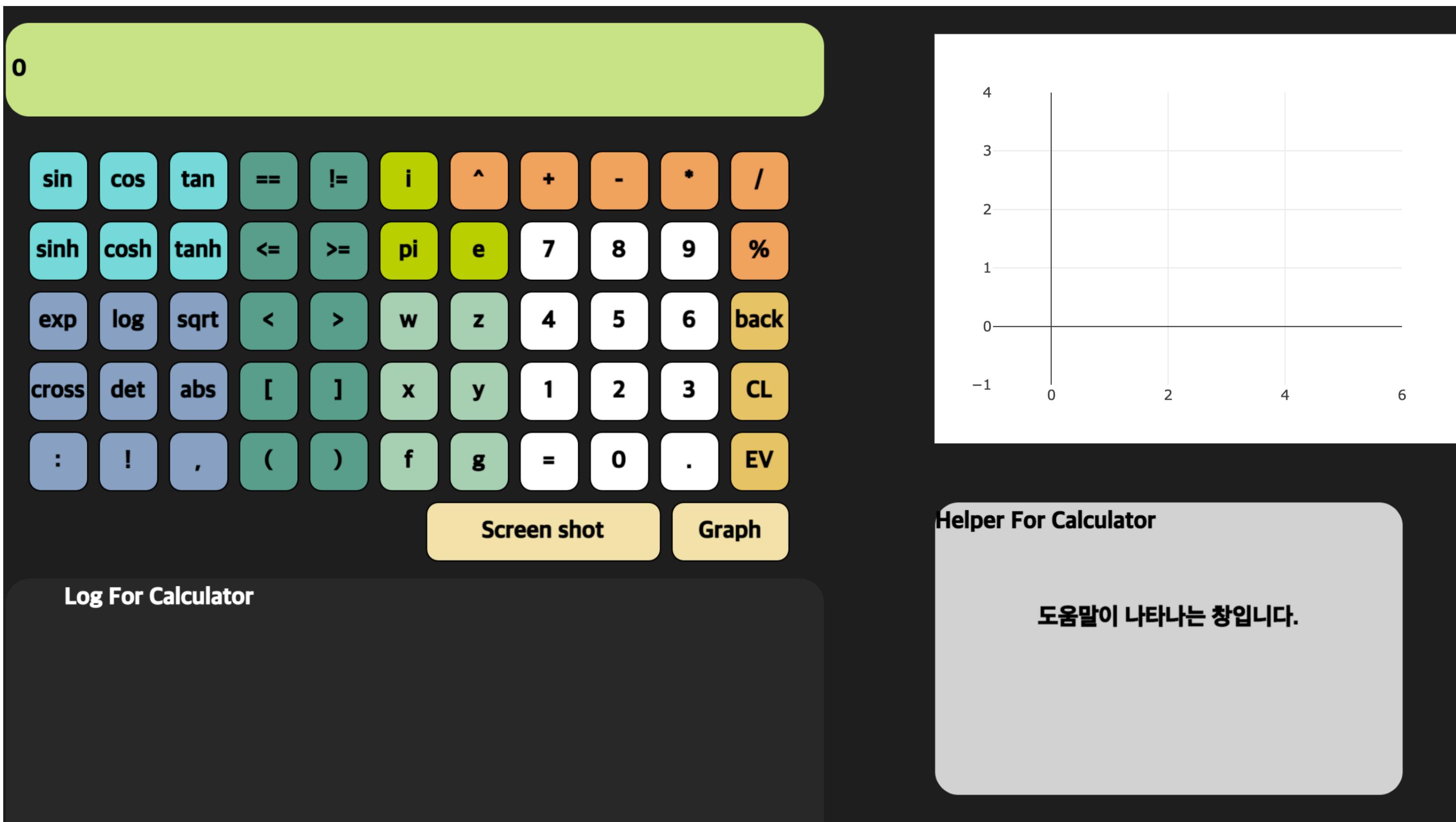
7

1	2	3	4	5	6	7	8	9	0
+	-	*	/	%	^	<	>	<=	>=
()	[]	.	,	:	;	==	!=
i	e	pi	w	x	y	z	f	g	=
exp	log	sqrt	sin	cos	tan	cross	det	CL	EV

3 단계

확장 계산기 구현

3단계 : 확장 계산기 구현 - 새로운 인터페이스 / 기능 개요



사용자의 편리성을 염두하며 계산기 [인터페이스](#)를 새로 재구성하였습니다. 같이 자주 쓰이는 버튼들을 신경써서 배치를 하였고 종류별로 색을 다르게 하여 [시각적으로 분리](#)되게 하였습니다. 또한 [계산 결과 로그](#)는 아래방향으로 내려가며 출력되게 하였고, 모든 버튼 위에 마우스를 올리면 [도움말](#)이 도움말 창에 나타납니다. [스크린샷](#) 버튼을 눌러 스크린샷을 저장할 수 있고, [그래프](#) 버튼을 눌러 그래프를 확인할 수 있습니다. 또한 버튼을 클릭할 때마다 [버튼클릭음](#)이 출력되며 계산에 실패하면 에러 메세지가 뜨면서 [에러음](#)이 출력됩니다.

3단계 : 확장 계산기 구현 - 새롭게 추가된 위젯들의 구현 방법

```
WebUI.BackGround.Container = function(color, properties) {
    WebUI.Widget.call(this, properties);

    this.type = WebUI.WidgetTypes.BACKGROUND_CONTAINER;
    this.fill_color = color;
}

WebUI.BackGround.Container.prototype = Object.create(WebUI.Widget.prototype);
WebUI.BackGround.Container.prototype.constructor = WebUI.BackGround.Container;

WebUI.BackGround.Container.prototype.initVisualItems = function() {
    let boundary = new fabric.Rect({
        left: this.position.left,
        top: this.position.top,
        width: this.desired_size.width,
        height: this.desired_size.height,
        fill: this.fill_color,
        rx: 20,
        ry: 20,
        selectable: false,
    });

    this.size = this.desired_size;
    this.visual_items.push(boundary);
    this.is_resource_ready = true;
}
```

Background Container 구현

기존의 컨테이너에 색을 입히고 둥근 모서리 효과를 주기 위해 Background Container라는 위젯을 선언했다. 이는 Container를 상속받고, color라는 개인 속성을 가진다. 둥근 모서리 효과를 주기 위해 initVisualItems 함수 내에서 rx, ry를 20으로 정의하였다.

3단계 : 확장 계산기 구현 - 새롭게 추가된 위젯들의 구현 방법

```
WebUI.HistoryField = function(label, desired_size) {
    WebUI.Widget.call(this);

    this.type = WebUI.WidgetTypes.HISTORY_FIELD;
    this.label = label;
    this.desired_size = desired_size;

    this.stroke_color = '#282828';
    this.fill_color = '#282828';

    this.font_family = 'System';
    this.font_size = 20;
    this.font_weight = 'normal';
    this.text_align = 'left';
    this.text_color = 'white';
}

WebUI.HistoryField.prototype = Object.create(WebUI.TextField.prototype);
WebUI.HistoryField.prototype.constructor = WebUI.HistoryField;
```

HistoryField 구현

TextField 를 상속받는 HistoryField 를 구현하였다. 이 필드는 배경색과 안에 속한 글자의 속성이 기존 TextField 와 다르다.

3단계 : 확장 계산기 구현 - 새롭게 추가된 위젯들의 구현 방법

```
new WebUI.Text("Log For Calculator", 'white'),  
WebUI.historyView[0] = new WebUI.HistoryField(`#${history[0]}`, {width: 600, height: 40}),  
WebUI.historyView[1] = new WebUI.HistoryField(`#${history[1]}`, {width: 600, height: 40}),  
WebUI.historyView[2] = new WebUI.HistoryField(`#${history[2]}`, {width: 600, height: 40}),  
WebUI.historyView[3] = new WebUI.HistoryField(`#${history[3]}`, {width: 600, height: 40}),  
WebUI.historyView[4] = new WebUI.HistoryField(`#${history[4]}`, {width: 600, height: 40}),
```

```
WebUI.addHistory(saveForlog, result);
```

```
WebUI.addHistory = function(expression, result){  
    let temp = expression + "=" + result;  
  
    history.unshift(temp);  
    history.pop();  
    WebUI.initialize();  
}
```

History 기능 구현

initWidgets 함수에 HistoryField 를 추가한다. 이들은 각각
label 로 미리 선언해둔 history 배열의 값들을 가지며,
WebUI.initialize 함수가 계산 버튼을 누를 때마다 호출되기 때
문에 기능을 효율적으로 구성하기 위해 배열을 따로 만들었다.
계산 버튼을 누르면 addHistory 함수가 호출이 되고, 계산식과
결과를 string 으로 저장해 배열의 앞에 삽입하고 제일 뒷부분
은 pop 한다.

3단계 : 확장 계산기 구현 - 새롭게 추가된 위젯들의 구현 방법

```
WebUI.Graph = function (id, desired_size) {
    WebUI.Widget.call(this);

    let graph = document.getElementById(id);
    this.id = id;
    this.desired_size = desired_size;
    this.type = WebUI.WidgetTypes.GRAPH;

    Plotly.newPlot(graph, [], {margin: {t: 50, l: 50, b: 50, r: 50}});
}
```

Graph 위젯 구현

Graph 를 이용하기 위해 위젯을 추가한다. 원래는 WebUI.canvas 상에서 직접 그리려고 했으나 구현에 실패하여 html 문서 상에서 id 를 가져와 변수로 설정하였다. 초기화면에 Plotly.js 라이브러리를 이용하여 새로운 그래프를 그린다. 이때 처음에는 값이 없기에 [] 을 넣어준다.

3단계 : 확장 계산기 구현 - 새롭게 추가된 위젯들의 구현 방법

```
var drawGraph = function (input, id) {  
    const graph = document.getElementById(id);  
  
    var text = input.replace('f(x)=', '');  
    text = text.replace('g(x)=', '');  
  
    const expr = math.compile(text);  
    const x = math.range(-20, 20, 0.5).toArray();  
    const y = x.map(function (x) {  
        return expr.eval({ x: x });  
    });  
  
    const trace = {  
        x: x,  
        y: y,  
        type: "scatter"  
    };  
  
    Plotly.newPlot(graph, [trace], {margin: {t: 50, l: 50, b: 50, r: 50}});  
}
```

Graph 위젯 함수 구현

버튼이 눌렸을 때 새로운 그래프를 그리기 위하여 함수를 구현하였다. 처음에 input에서 $f(x)=$ 와 같은 텍스트가 있다면 이를 제거하고, 컴파일하여 수식으로 만든다. 이어 x,y 축의 범위를 설정하고 Plotly.newPlot 함수를 이용하여 이 데이터의 배열을 그리게 한다. 이후 그래프 그리기 버튼이 눌렸을 때 이 함수를 호출하여 그래프를 그리게 한다.

3단계 : 확장 계산기 구현 - 새롭게 추가된 위젯들의 구현 방법

```
this.helpView = new WebUI.Text("도움말이 나타나는 창입니다.", "black");

WebUI.PushButton.prototype.handleMouseEnter = function() {
    this.visual_items[0].set('strokeWidth', 3);
    let on_button = this.label

    switch(on_button) {
        case 'sin':
        case 'cos':
        case 'tan':
        case 'sinh':
        case 'cosh':
        case 'tanh':
            WebUI.helpView.visual_items[0].text = "삼각함수를 계산합니다. \n ex) sin(pi) = 0 \n"
            break;

        case '+':
        case '-':
        case '*':
        case '/':
            WebUI.helpView.visual_items[0].text = "사칙 연산 기호입니다. \n ex) 1 + 4 = 5 \n"
            break;

        case '%':
            WebUI.helpView.visual_items[0].text = "나머지 연산을 합니다. \n ex) 6 % 3 = 0";
            break;

        case '^':
            WebUI.helpView.visual_items[0].text = "제곱 연산을 합니다. \n ex) 2^4 = 8";
            break;

        case 'x':
        case 'y':
        case 'z':
        case 'w':
            WebUI.helpView.visual_items[0].text = "변수입니다. \nex) f(x) = x*x\n          y=f(x)"
            break;

        case 'f':
        case 'g':
            WebUI.helpView.visual_items[0].text = "함수입니다. \nex) f(x) = x*x\n          g(2) = "
            break;
    }
}
```

도움말 창 기능 구현

앞서 구현한 Background_Container 컨테이너를 깔아놓은 뒤, helpView라는 Text 객체를 생성하고 initWidgets에 추가하였다. PushButton handleMouseEnter 함수에서 마우스 커서가 버튼 안에 들어갔을 때 버튼의 label로 switch 문으로 분기해 도움말을 출력해준다. 원래 버튼의 상단에 도움말을 띄우려고 했지만, 다른 버튼을 잠시 가린다는 단점이 있어 도움말 창을 따로 만들게 되었다.

3단계 : 확장 계산기 구현 - 새롭게 추가된 위젯들의 구현 방법

```
<script src="https://html2canvas.hertzen.com/dist/html2canvas.min.js"></script>
```

```
<div id="capture" style="padding: 10px; background: #eee">
  <canvas id="c">
    Canvas not supported
  </canvas>
  <div id="graph" style="position: absolute; left: 900px; top: 50px; width: 450px; height: 350px"></div>
</div>
<body>
</body>
```

```
WebUI.saveScreenShot = function() {
  html2canvas(document.querySelector("#capture")).then(canvas => {
    var link = document.createElement('a');
    if(typeof link.download == 'string') {
      link.href = canvas.toDataURL('image/png');
      link.download = `캡쳐화면_${index_capture}`;

      document.body.appendChild(link);
      link.click();
      document.body.removeChild(link);
    }
    else {
      window.open(uri);
    }

    index_capture += 1;
  });
}
```

스크린샷 기능 구현

맨 위에서 **html2canvas** 라이브러리를 import 하는 것을 볼 수 있다. 두번째는 html 문서로 capture라는 아이디를 가진 div 안에 canvas를 정의하여 모든 화면이 스크린샷에 담길 수 있게 하였다. 아래의 함수는 스크린샷을 저장하는 함수로 html2canvas 라이브러리를 사용하여 uri와 이미지 파일 이름을 지정하여 캡쳐화면을 다운받는다.

3단계 : 확장 계산기 구현 - 새롭게 추가된 위젯들의 구현 방법

```
if(this.onPushed != undefined){  
    this.onPushed.call(this);  
    var beep= new Audio('resources/type.mp3');  
    beep.play();  
}  
}
```

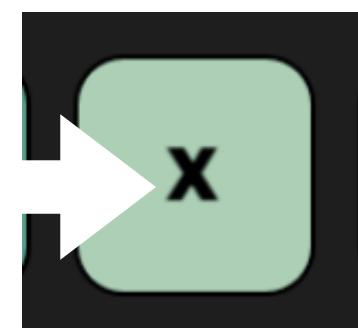
버튼 클릭음 기능 구현

버튼이 눌릴 때마다 소리가 나야 했기에 QPushButton의 mouseDown 함수에 미리 넣어둔 리소스에 있는 음성 파일을 가져와서 재생했다. 또한 에러가 났을 경우에는 'EV' CalcButton의 handlemousedown 함수 내 에러 catch 문에서 에러음을 출력하도록 하였다.

```
} catch(e) {  
    expression = '0';  
    if(expression != 'function') {  
        var beep= new Audio('resources/beep-1.wav');  
        beep.play();  
        expression = e.message;  
        WebUI.initialize();  
    }  
}
```

3단계 : 확장 계산기 구현 - 실행 과정 제시 및 결과 분석

0



디자인적 측면

기능을 다소 찾기 힘들고 단색이었던 기본 계산기에 비해, 기능별로 버튼들을 색으로 구분하기 쉽게 구성하였고 자주 쓰이는 기능들을 비슷한 공간에 배치하려고 노력하였다.

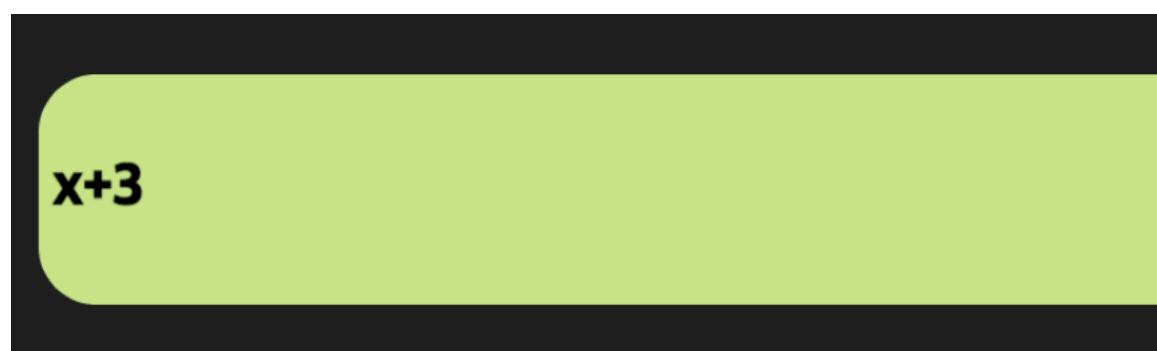
추가된 함수들

기본 계산기에 함수들을 추가하였고, BackSpace 기능을 하는 버튼도 추가하였다.

버튼 클릭음, 에러음 출력

계산기 버튼을 클릭할 때와 에러가 났을 때 이에 맞는 소리를 출력한다.

3단계 : 확장 계산기 구현 - 실행 과정 제시 및 결과 분석



Log For Calculator

$\log(e)==1=true$

$6\%9=6$

$7^8=5764801$

$5-8=-3$

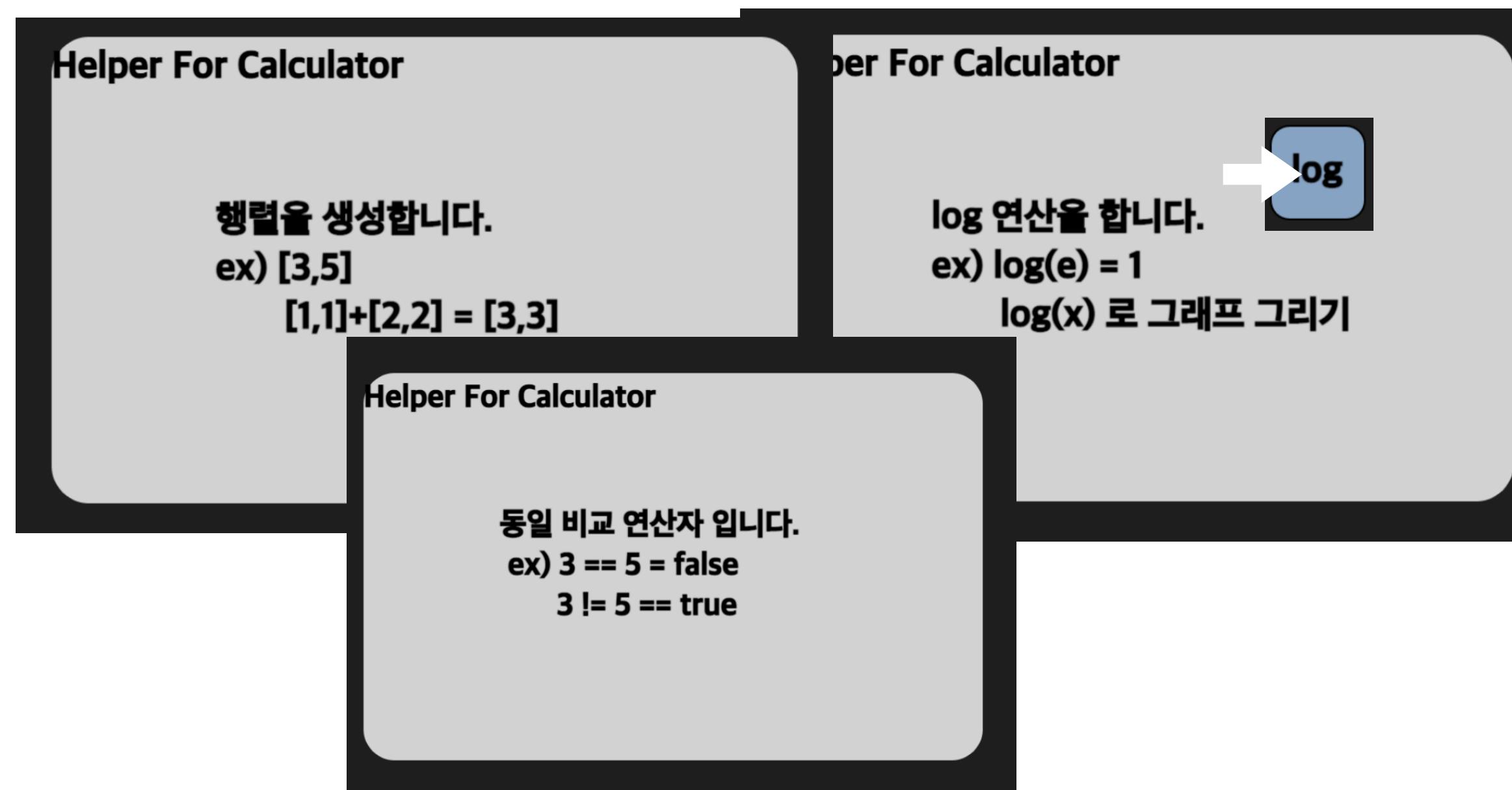
그래프 창

성공적으로 그래프가 출력되는 것을 볼 수 있다.
앞에 $f(x)$ 나 $g(x)$ 가 있는 경우도 처리한다.

과거 실행 로그

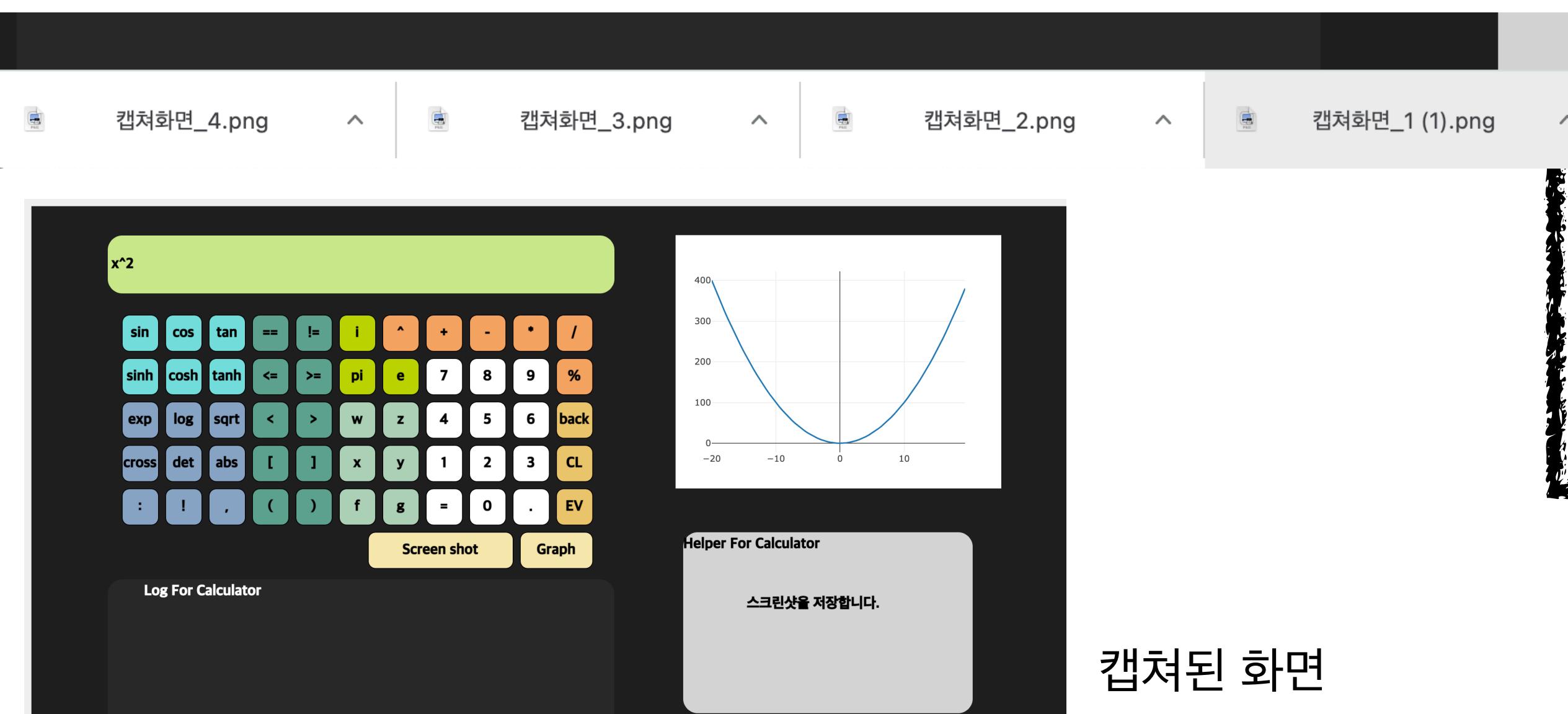
하단 창에 과거에 실행했던 결과들이 차례대로 저장되어 위에서 내려가는 형태로 나타난다.

3단계 : 확장 계산기 구현 - 실행 과정 제시 및 결과 분석



도움말 창

마우스 커서를 버튼 위에 올리면 도움말과 사용예시가 출력된다.



스크린샷 기능

현재 화면을 스크린샷으로 저장한다.

캡쳐된 화면

논의

성공적이었던 부분, 전반적 자체 평가

웹 프로그래밍이 처음이지만 두번째 과제에서 요구하는 요구 사항과 스크린샷 기능, 그래프, 도움말 기능, 로그 기능 등 목표로 했던 기능을 모두 만족시켰다는 점에서 만족할만 하다. 또한 기본 계산기에서 시각적으로 많이 발전한 모습이 보여진다. 이 과정에서 많이 헤매고 좌절하기도 했지만 배운 점이 많았다.

부족한 부분

끝까지 고민했던 부분은 `plotly.js` 를 이용해서 그래프를 구현할 때 `html` 을 이용하지 않고 `WebUI.canvas` 상에서 그려보려고 했지만 구현을 완성시키지 못해 많이 아쉽다. 또한 계산 `history` 부분에서 스크롤바를 이용해 더 예전 로그를 보여주는 기능을 추가하려고 했지만 계속 실패했고, 실패 요인은 `WebUI.initialize` 를 반복적으로 호출하는 코드때문이다. 하지만 이 부분을 고치려면 모든 로직이 변경되어야 함을 의미했고, 시간상 불가능하다고 판단되어 미루게 되었다.

향후 개선점

향후 개선점으로는 구현에 실패했던 부분을 고치려고 계속 시도해보고, 개인적으로 코드 구조에 대한 아쉬움이 많이 남아있다. 이는 자바 스크립트의 기본을 더욱 탄탄히 다져 지속적으로 고민해봐야 할 듯하다.