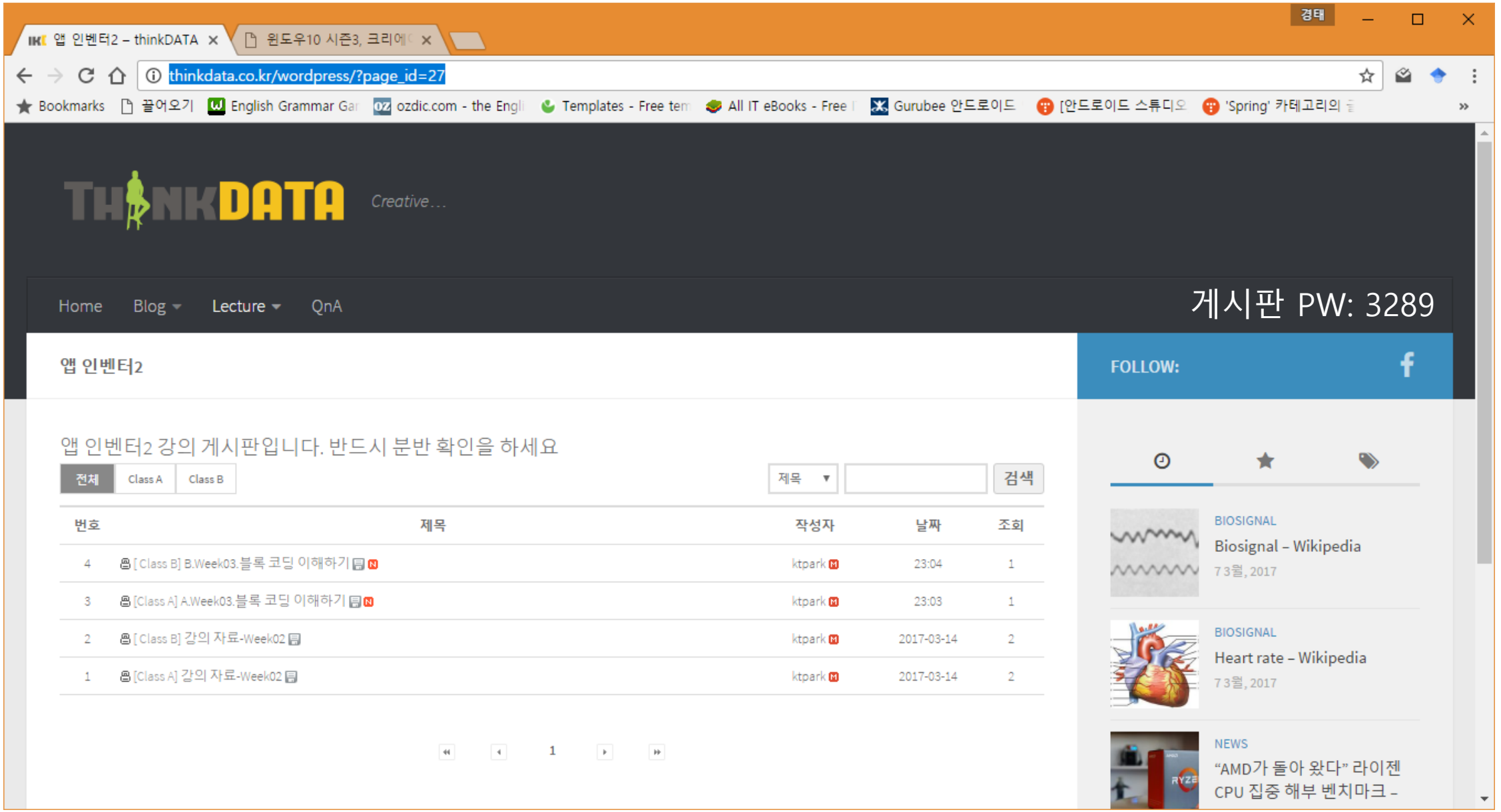
블록 코딩 이해하기

Contents Title

주 차	수 업 내 용
1	수업 소개
2	앱 인벤터 개발환경과 디자인 편집기
3	블록 코딩 이해하기(Hello 앱 인벤터)
4	안녕 야옹이
5	페인터 통
6	파리 관광
7	잡아라 두더지
8	**중간고사**
9	운전 중 문자 금지(or 바로 콜)
10	무당벌레 추적
11	내 차를 찾아줘
12	실로폰
13	서점에서 온라인 검색하기
14	메모장 만들기
15	기말고사

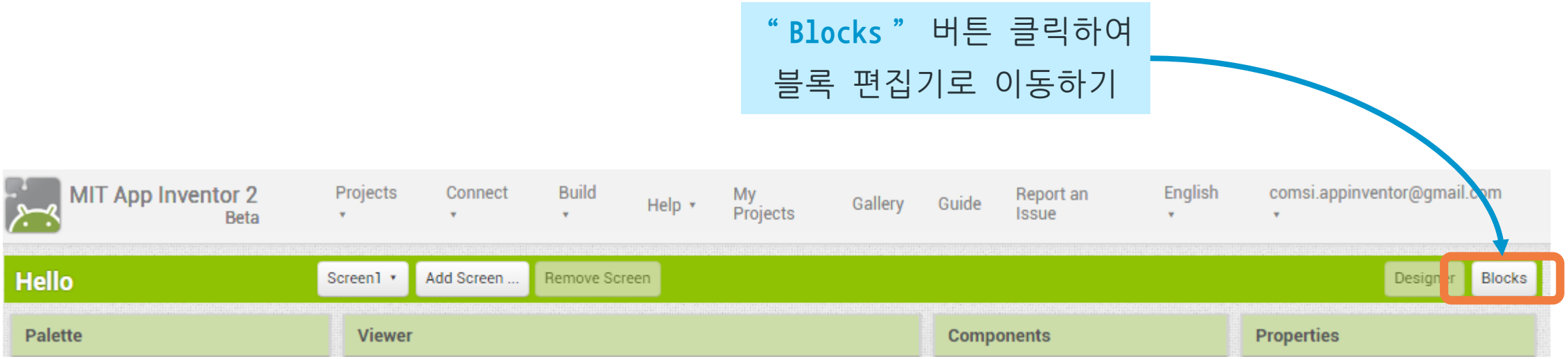
http://thinkdata.co.kr/wordpress/?page_id=27



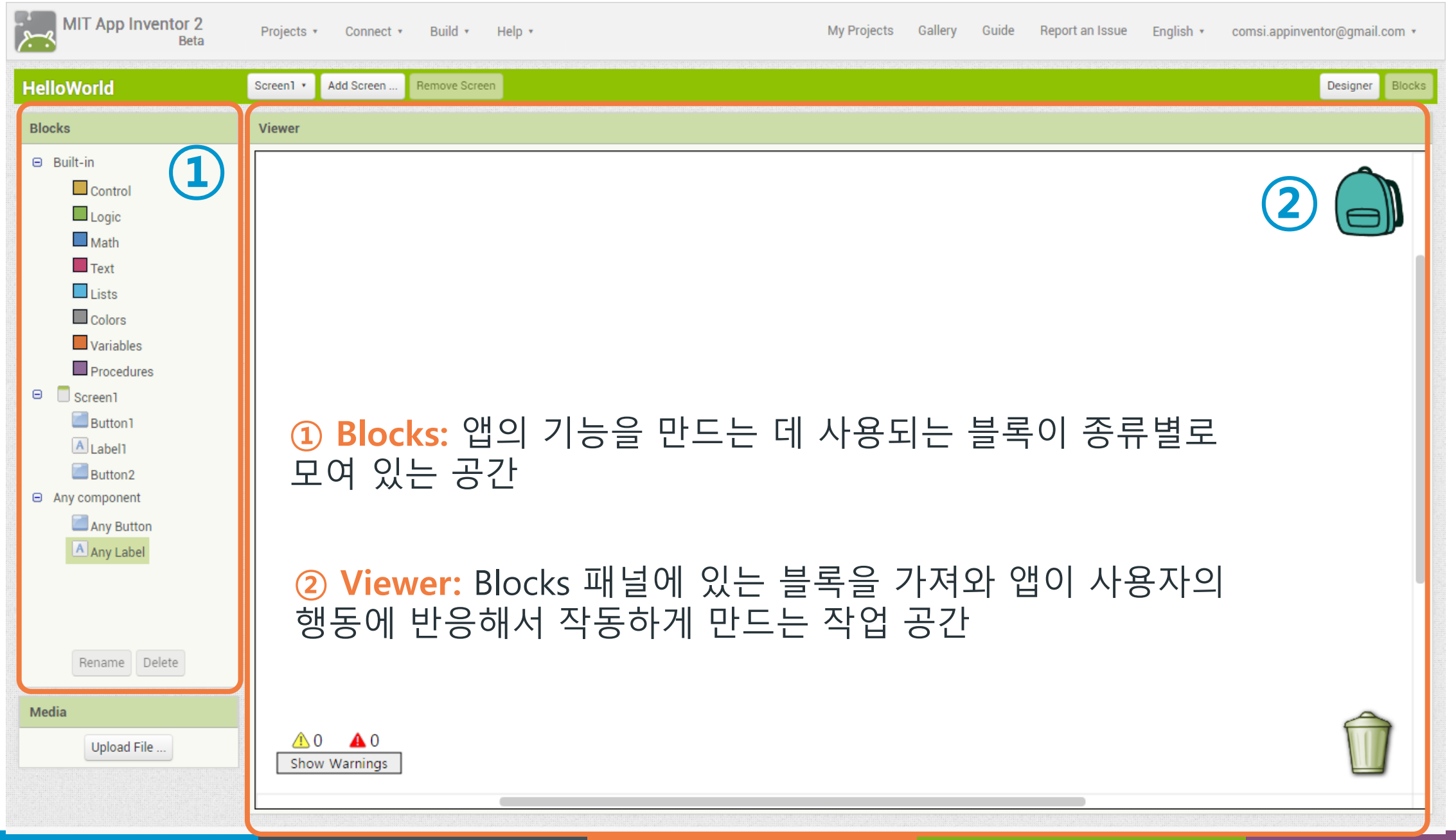
2. 블록 코딩 알아보기

2. 블록 코딩 알아보기

- 디자인 편집기에서 **[Blocks]버튼**을 클릭하여 블록 편집기로 이동
- 블록 편집기는 **Blocks** 패널과 **Viewer** 패널로 구성



• 블록 편집기 화면

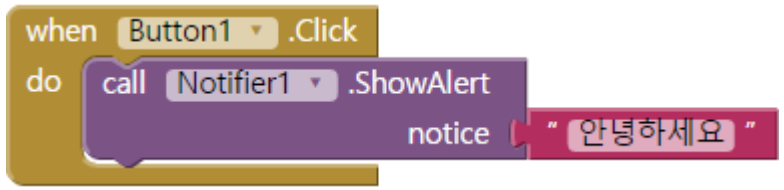

MIT App Inventor 2
Beta
Projects
Connect
Build
Help
My Projects
Gallery
Guide
Report an Issue
English
comsi.appinventor@gmail.com
HelloWorld
Screen1
Add Screen ...
Remove Screen
Designer
Blocks
Blocks
Built-in
Control
Logic
Math
Text
Lists
Colors
Variables
Procedures
Screen1
Button1
Label1
Button2
Any component
Any Button
Any Label
Rename
Delete
Media
Upload File ...
Viewer
① Blocks: 앱의 기능을 만드는 데 사용되는 블록이 종류별로 모여 있는 공간
② Viewer: Blocks 패널에 있는 블록을 가져와 앱이 사용자의 행동에 반응해서 작동하게 만드는 작업 공간
0
0
Show Warnings

1) 블록 코딩 vs. 텍스트 코딩

“ 안녕하세요 ” 팝업 출력

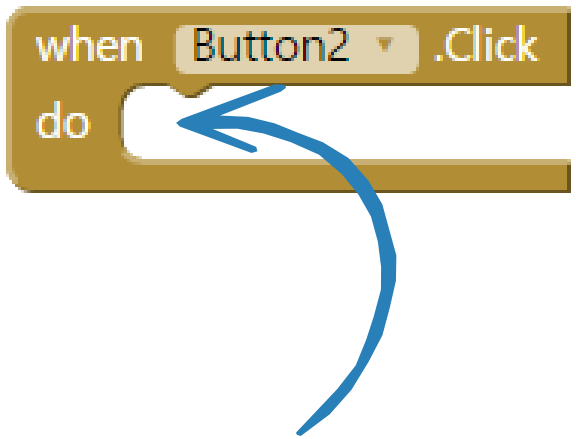
```
package com.example.park.myapplication;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button btnHelloWorld = (Button) findViewById(R.id.btnHelloWorld);
        btnHelloWorld.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Toast.makeText(getApplicationContext(), "안녕하세요", 1000).show();
            }
        });
    }
}
```

텍스트 코딩



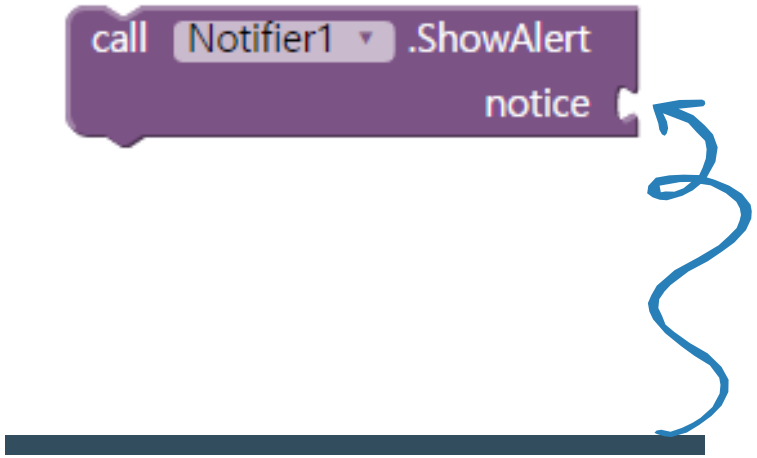
블록 코딩

블록의 종류와 명칭



섹션(Section)

실행할 블록을 여러 개 붙여 순차적 처리를 할 수 있는 블록



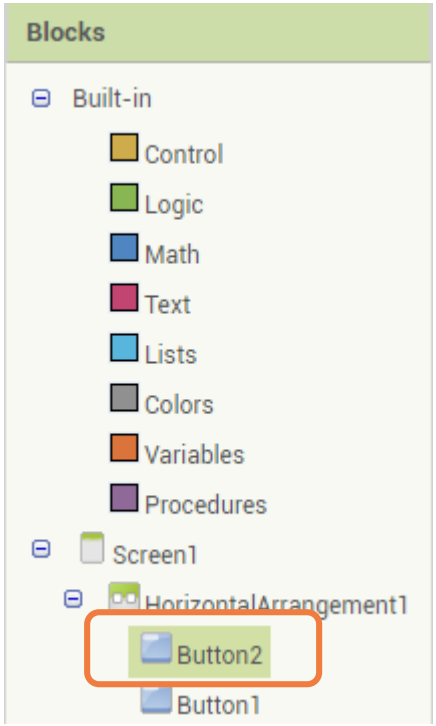
소켓(Socket)

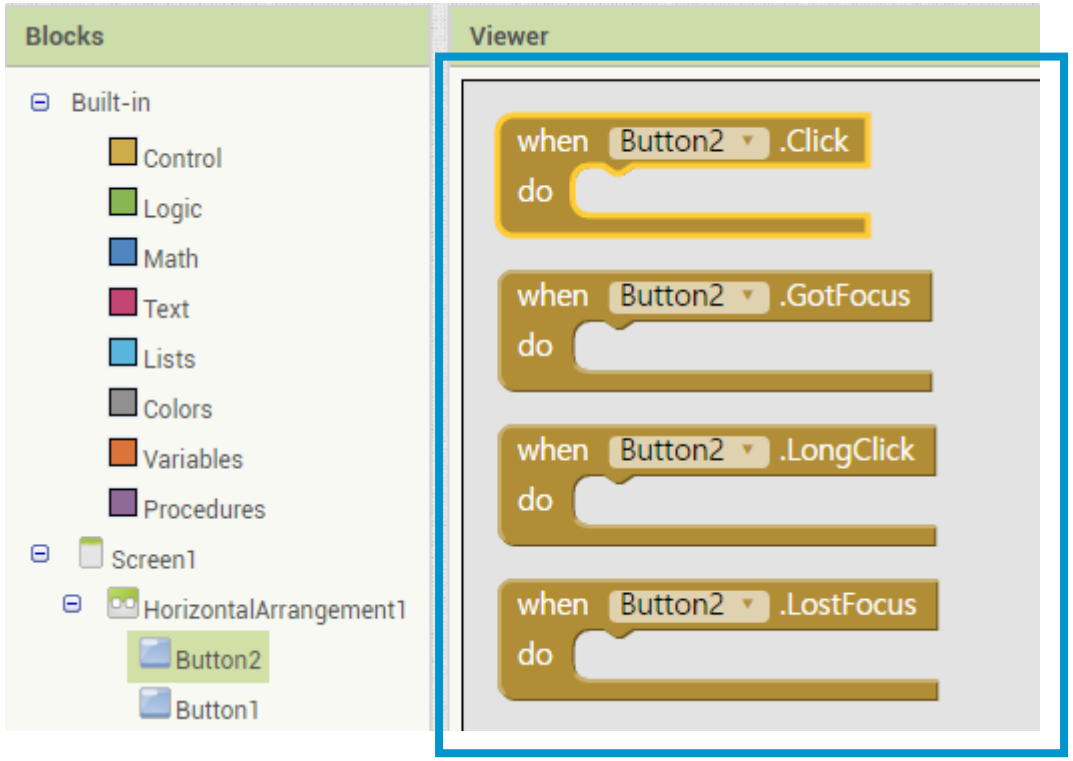
실행할 블록을 하나 끼워 처리할 수 있는 블록으로 소켓 블록을 끼운다.

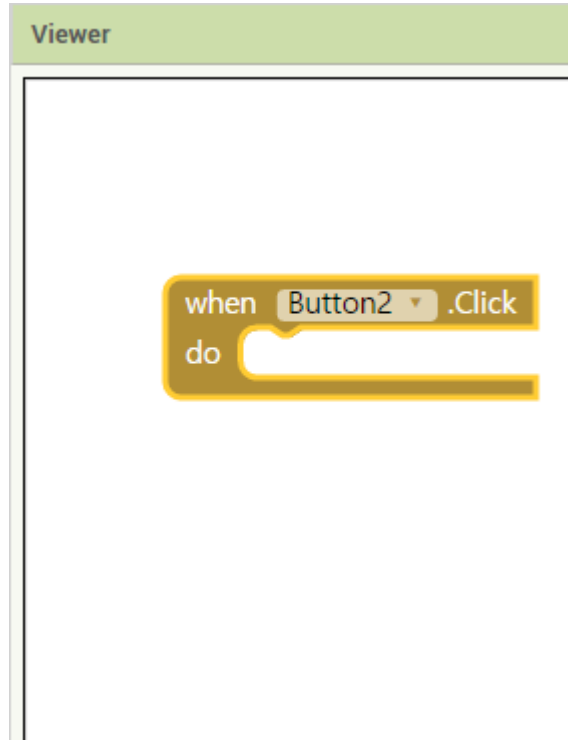
플러그(Plug)

소켓에 끼워 사용할 수 있는 블록

2) 블록 코딩 방법 – 3단계







Step 1

Blocks 패널에서 코딩을 위한 요소(**Button2**)를 클릭하여 사용 가능한 팝업 창(**step2 사각형**)을 띄운다.

Step 2

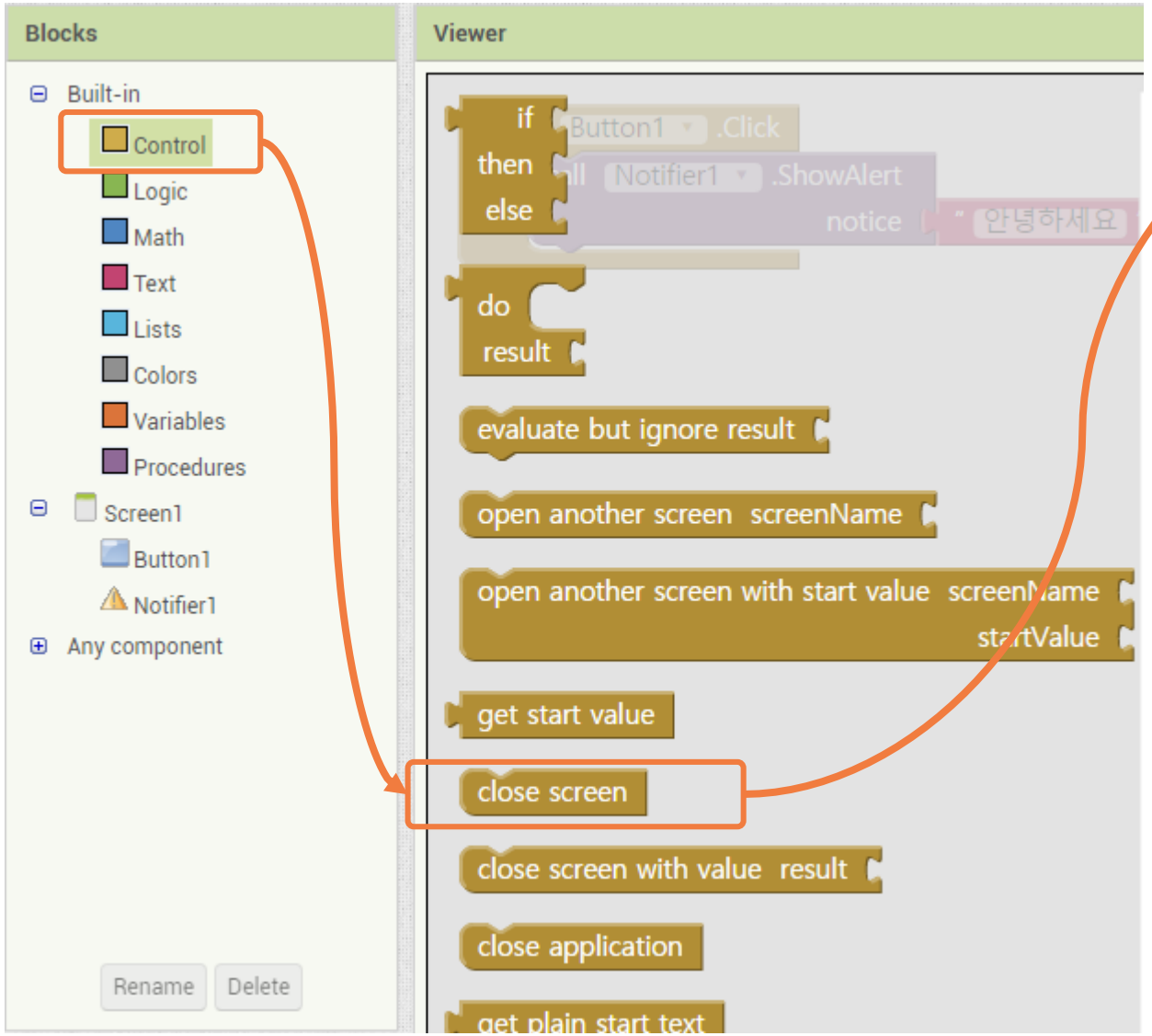
블록 창(**사각형**)에서 사용하고자 하는 블록을 선택하여 드래그 한다.

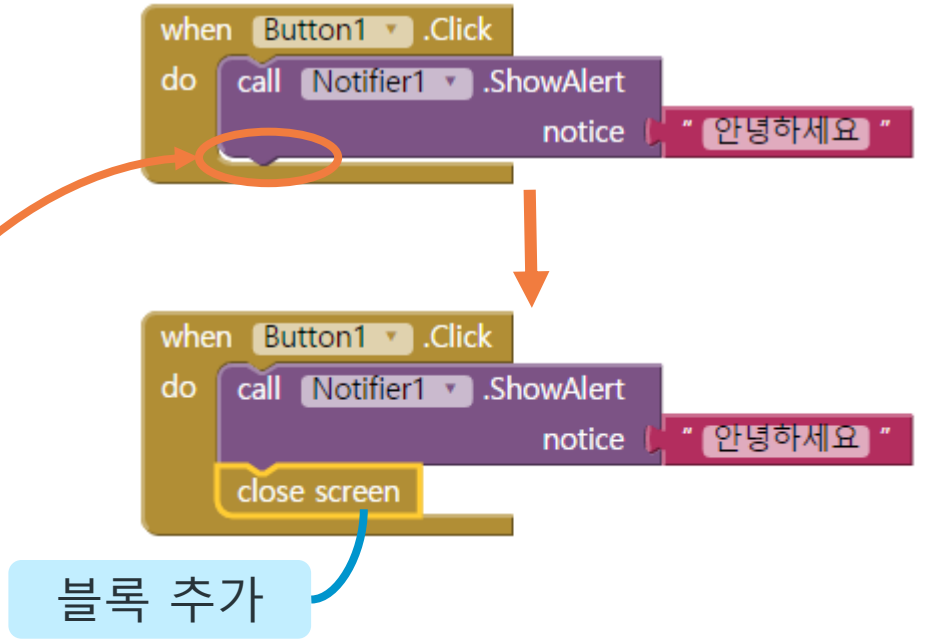
Step 3

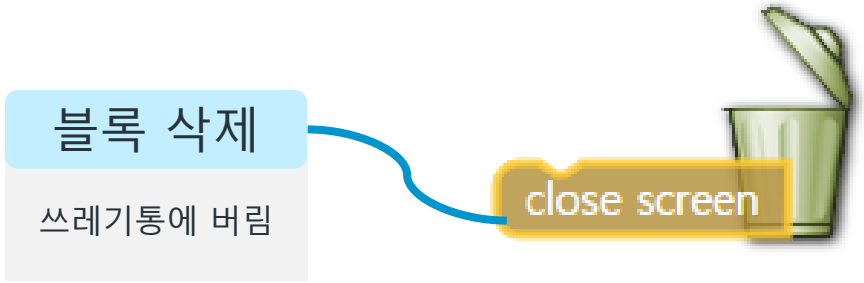
드래그 한 블록을 Viewer에 드롭한다.

3) 블록 추가하기와 삭제하기

- 앱 종료 블록을 추가해 보자

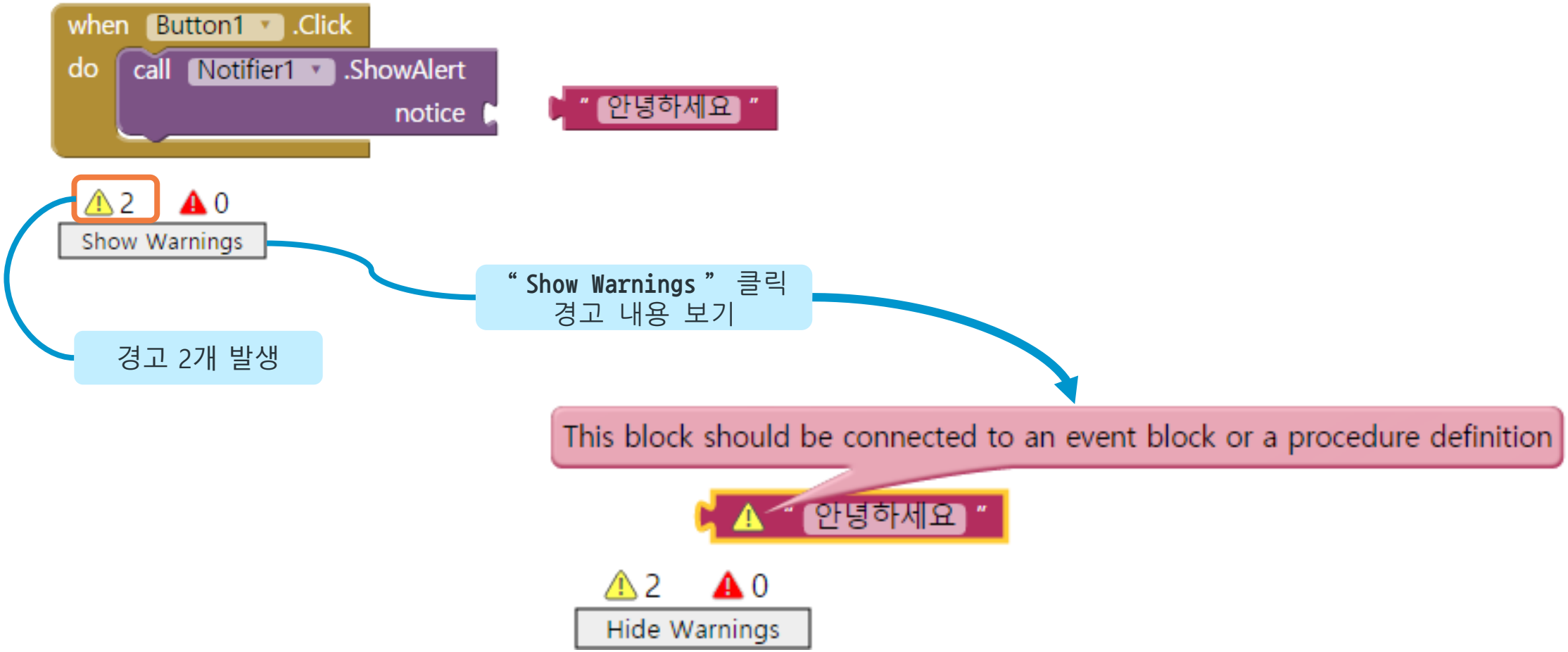




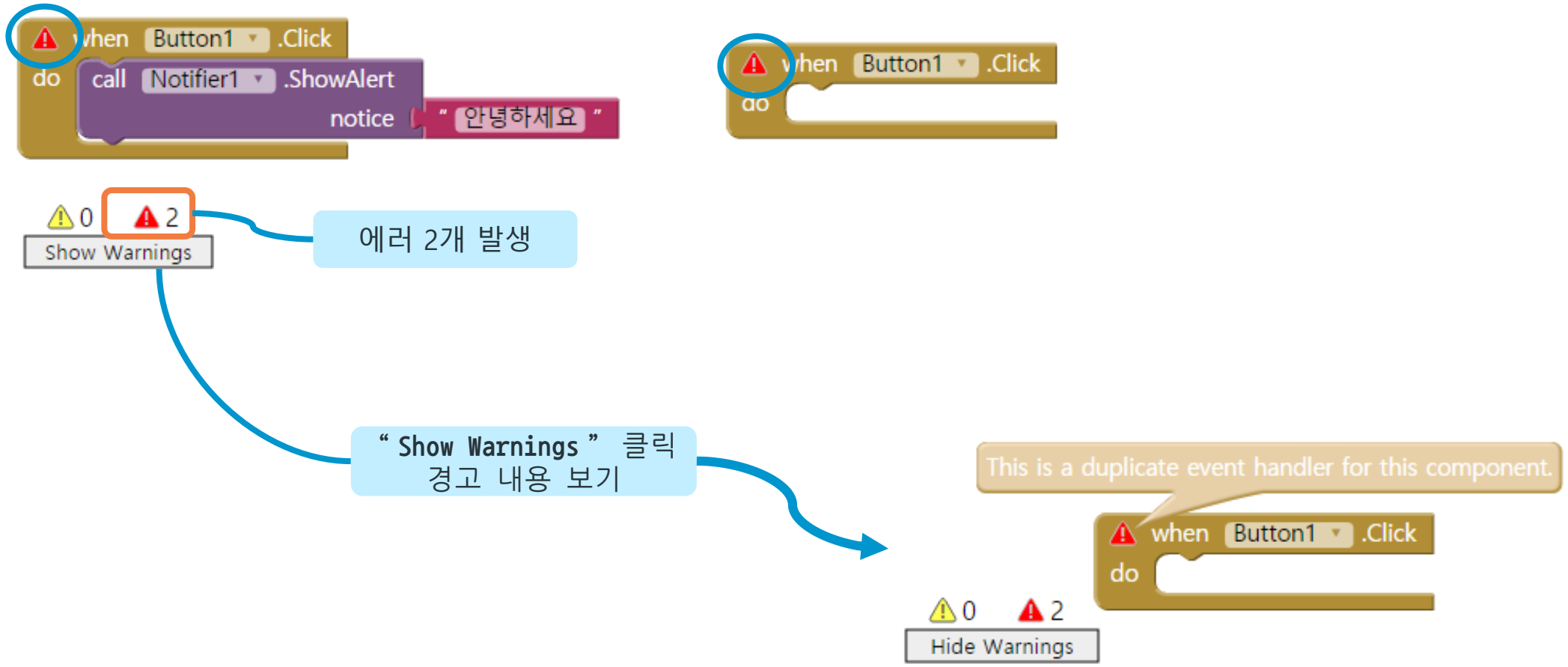


4) 오류 처리 – 경고와 에러

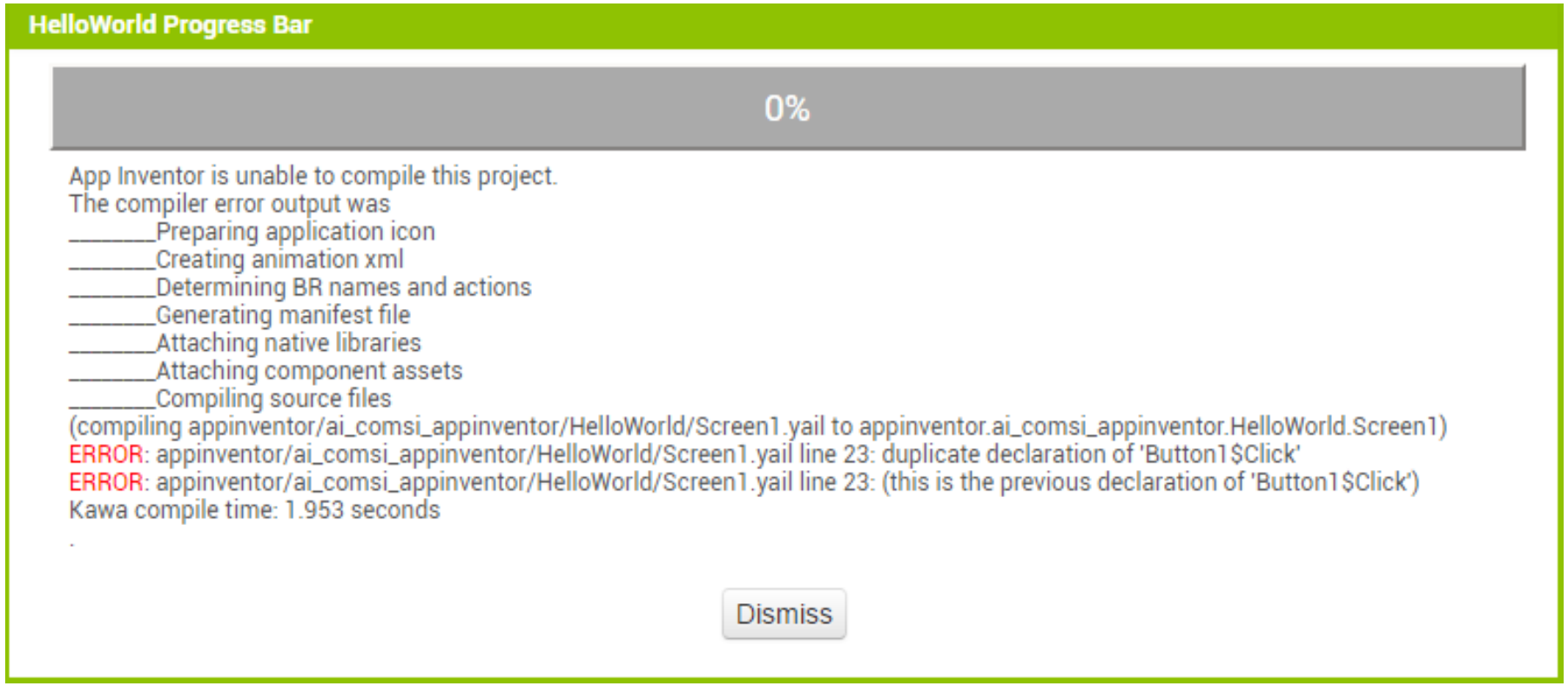
☞ 오류 - **경고 메시지**(앱을 빌드하여 스마트폰에 정상적으로 설치하여 실행할 수 있다)



오류 - 에러 메시지(앱을 빌드할 수 없다)



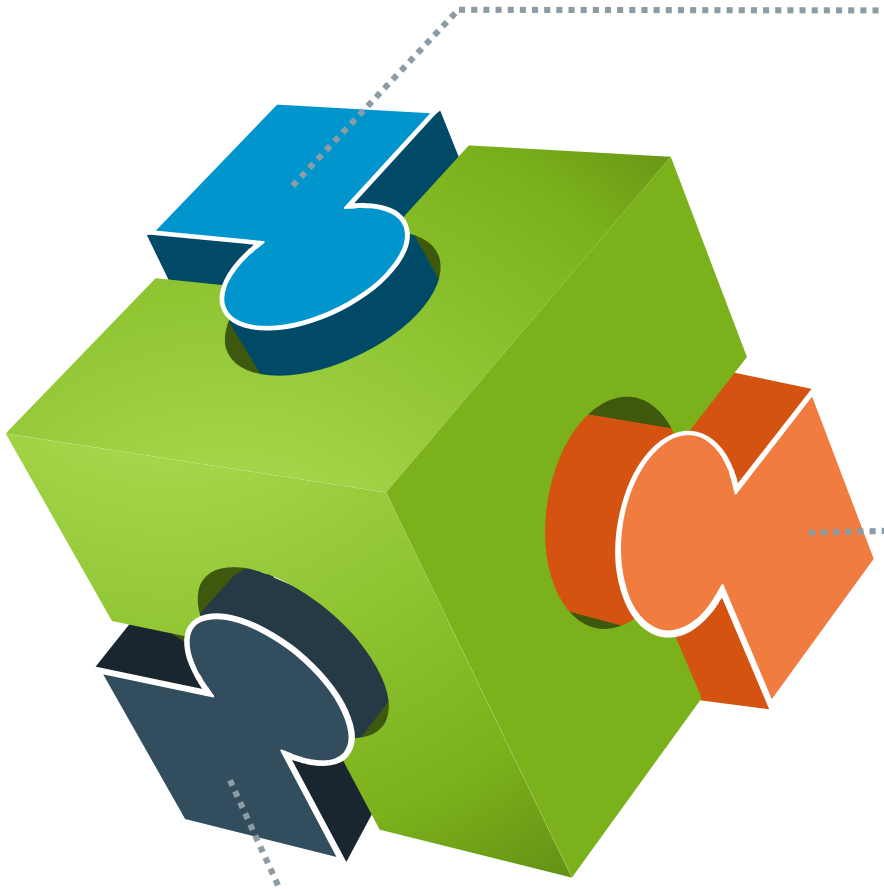
에러 상태에서 빌드(Build → App(provide QR code for .apk) 메뉴 실행) 할 경우 오류



- 첫번째 빨간색 ERROR: duplicate declaration of ‘Button1$Click’
- 해결: 중복된 [when Button1.Click] 블록을 쓰레기통에 버리면 오류가 제거되고 빌드가 정상적으로 진행된다.

3. 블록의 구분

3. Blocks의 구분



Built-in

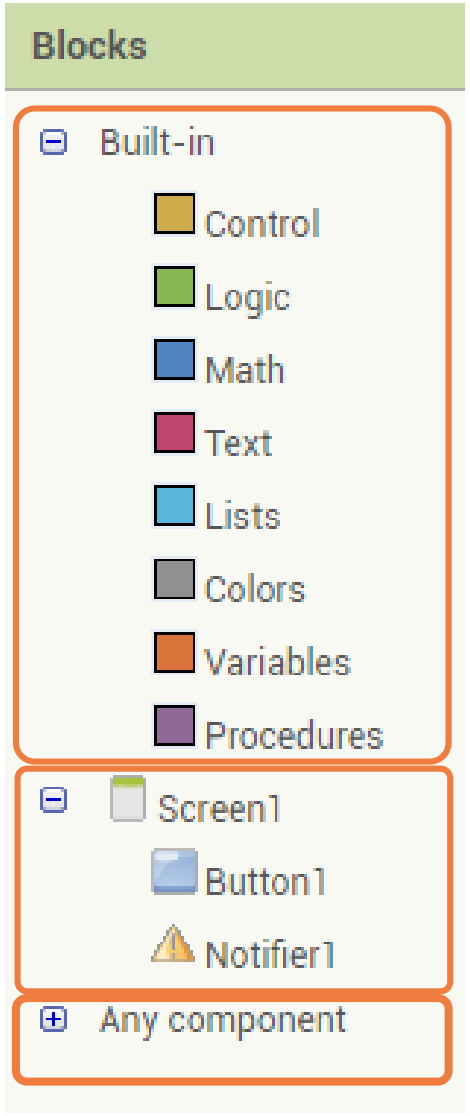
기능에 따라
[Control], [Logic], [Math],
[Text], [List], [Colors],
[Variables], [Procedures]
8가지로 분류.

Screen1

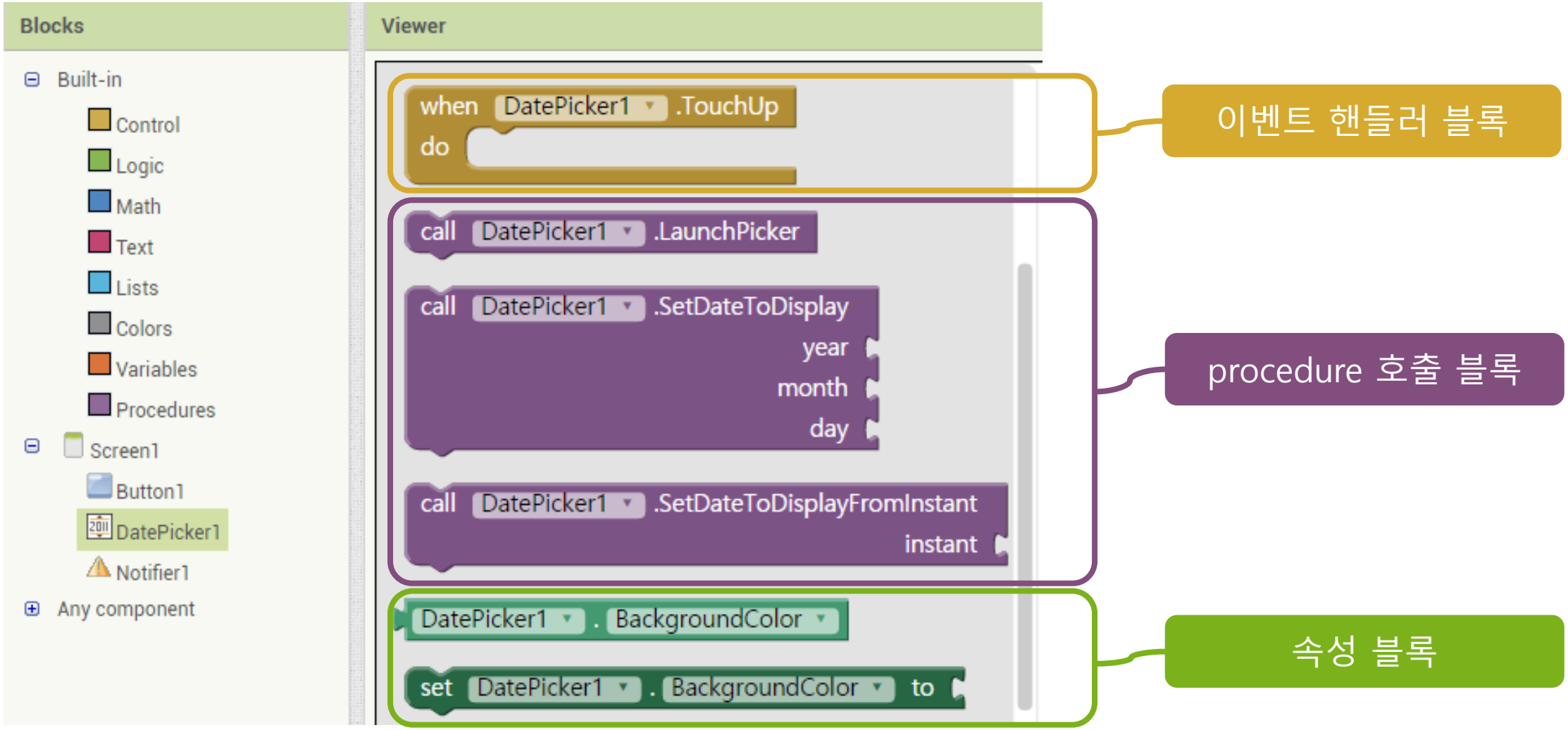
디자인 편집에서
사용한
컴포넌트들이
트리구조로 나열

Any component

Screen1에 있는 같은
종류의 컴포넌트들을
한꺼번에 제어할 때 사용

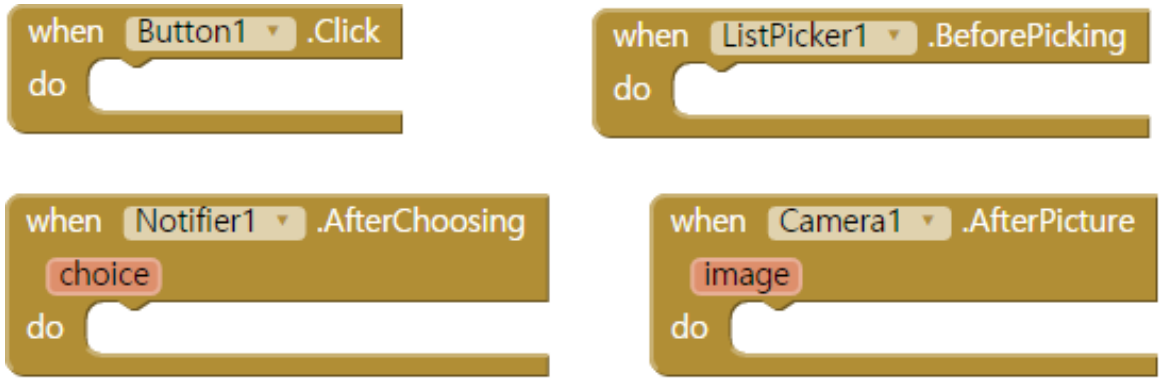


DatePicker 컴포넌트의 블록 목록(블록의 기능에 따라 분류-색상구분)

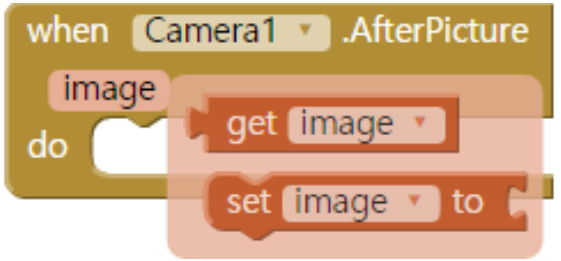


1) 이벤트 핸들러 블록

- 사용자가 특정 행동을 했을 때 **지정된 동작을 하도록 사용하는 블록**
- 이벤트는 **사용자가 발생시키는 경우**와 **내부적으로 특정한 일을 수행하기 전이나 후에 발생**하기도 한다.
- **황토색**으로 되어 있고, 시작 부분에 "**when**"이라는 단어로 시작하고 이벤트가 발생했을 때 수행하고자 하는 블록은 "**do**" 섹션에 삽입



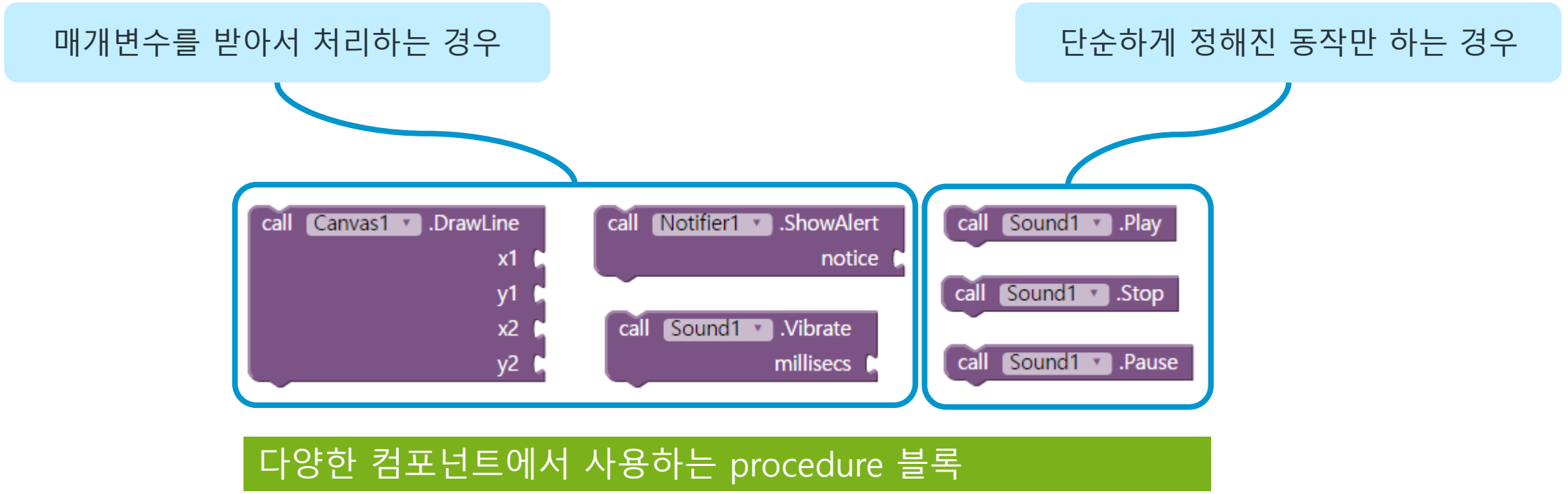
다양한 컴포넌트에서 발생하는 이벤트 블록



이벤트 핸들러 블록에서 매개변수 받기

2) procedure 블록

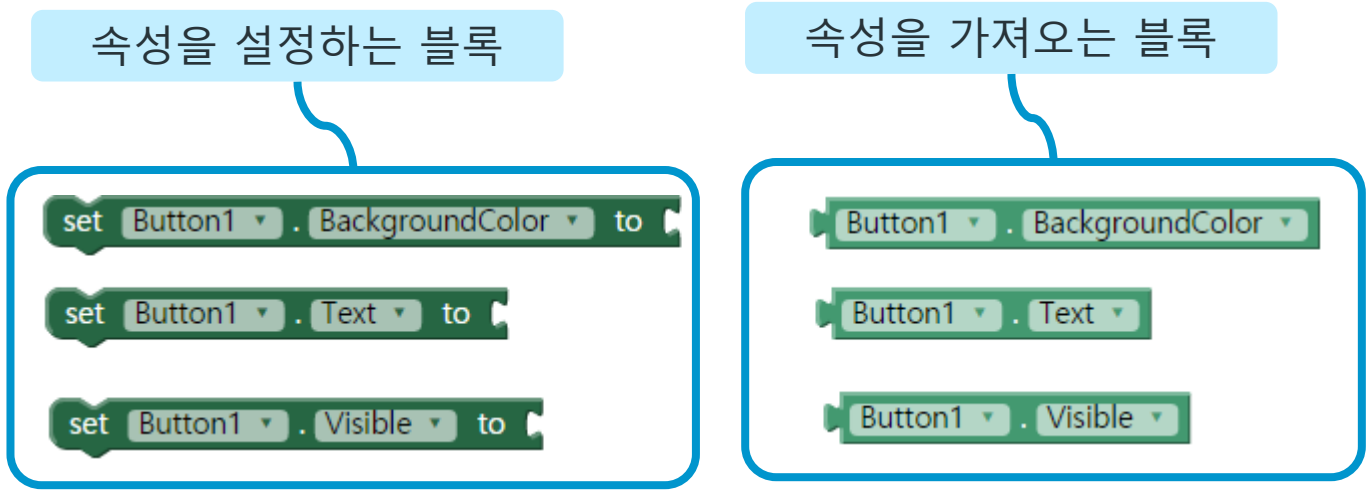
- **procedure**는 컴포넌트가 수행해야 할 동작을 미리 만들어 놓은 **보라색** 블록
- 소켓이 있어서 **매개변수를 받아서 처리하는 경우**와 **단순하게 정해진 동작만 하는 경우**



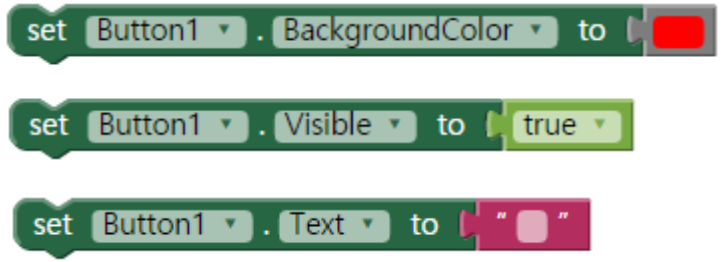
다양한 컴포넌트에서 사용하는 procedure 블록

3) 속성 블록

- 앱 인벤터에 사용되는 컴포넌트의 속성을 설정하거나 가져오는 **초록색** 블록
- 속성을 설정하는 블록은 오른쪽에 소켓이 있고, 가져오는 블록은 왼쪽에 플러그가 구성되어 있음
- 속성을 설정하기 위한 블록은 "set"으로 시작하고 "to" 소켓으로 끝나는 블록



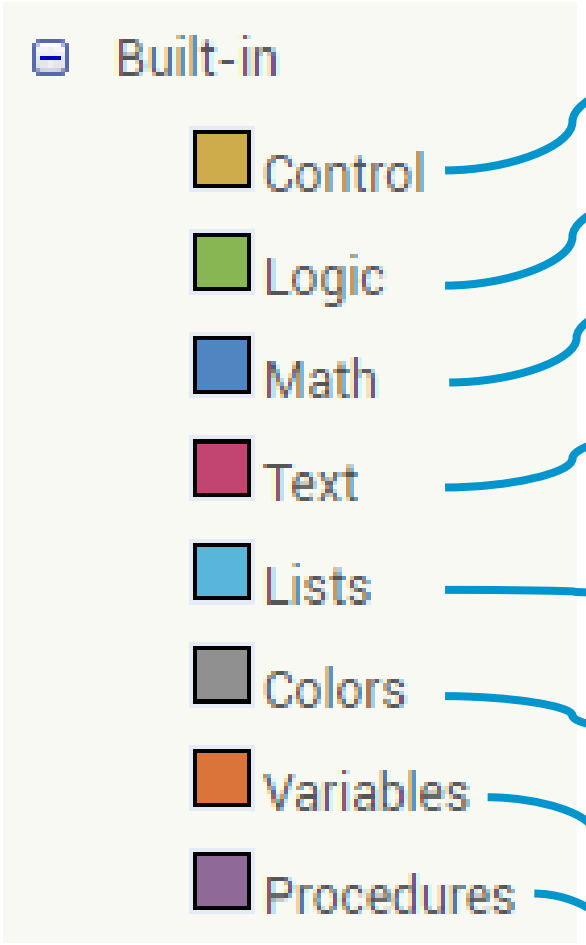
버튼 컴포넌트의 다양한 속성 블록들



버튼 컴포넌트의 속성 설정

4) Built-in 블록

- 일반적인 프로그래밍 언어에서 제공하는 명령어와 함수를 제공



조건에 따라 프로그램의 흐름을 제어하는 **황토색** 블록

흐름 제어에 필요한 논리 조건을 처리하는 **초록색** 블록

기본적인 숫자자체, 비교연산, 사칙연산, 난수 발생 등의 **파란색** 블록

문자열 자체, 문자열의 공백 제거나 길이, 문자열을 연결하거나 자르거나 대체하는 **자주색** 블록

여러 개의 값을 저장하는 변수 공간으로 **청록색** 블록

컴포넌트의 배경이나 글자색을 설정할 수 있는 **회색**의 색상블록
(기본 13가지 색상과 Red, Green, Blue의 농도 조절로 색상 조절 가능)

값을 임시로 저장할 수 있는 변수를 만드는 **주황색** 블록

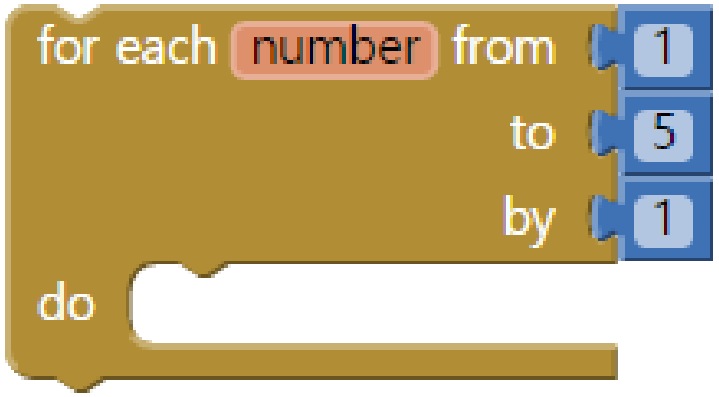
반복적으로 사용되는 기능을 모아 만든 **보라색** 블록
(기본 내장 procedure와 사용자 정의 procedure를 생성 가능)

(1) Control 서랍

- 조건에 따라 프로그램의 흐름을 제어하는 **황토색** 블록

① if then

② for each ~ from to by



③ while test

④ for each ~ in list

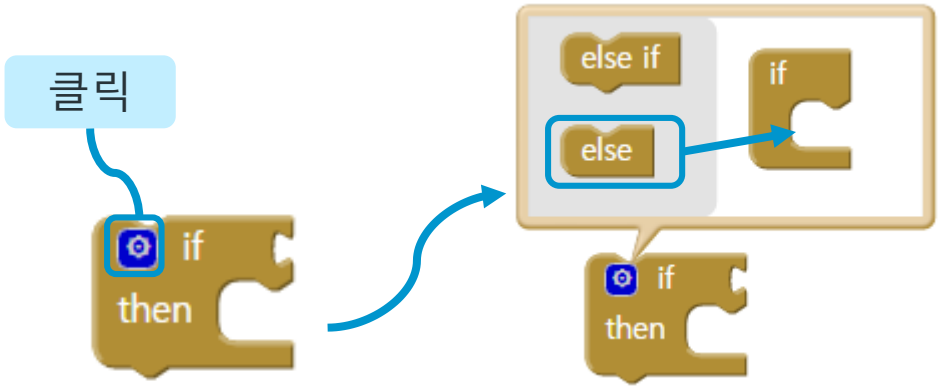
① if 소켓의 조건이 참일 경우에만 then 섹션의 블록을 실행

② by 소켓의 값을 증가양으로 하여 from 소켓의 값부터 to 소켓의 값까지 do 섹션의 블록을 반복적으로 수행. 이때, number에는 현재 값이 저장됨

③ test 소켓의 조건이 참일 동안 do 섹션을 반복 실행

④ list 소켓의 내용을 하나씩 가져와 모두 사용될 때까지 do 섹션 실행. item에서 list에서 가져온 내용이 저장됨

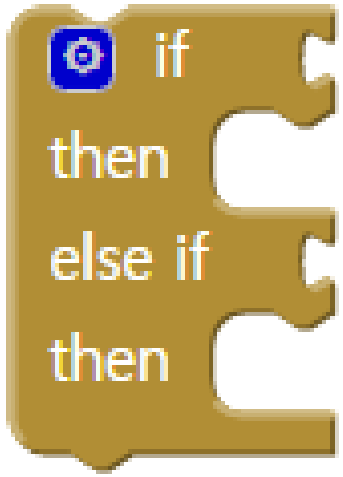
[if then] 블록으로 [if then else] 블록으로 확장하기



[if then] 블록

[if then else] 블록

- 같은 방법으로 [else] 블록 대신에 [else if] 블록을 끼우면 [if then else if then] 블록도 만들 수 있다.



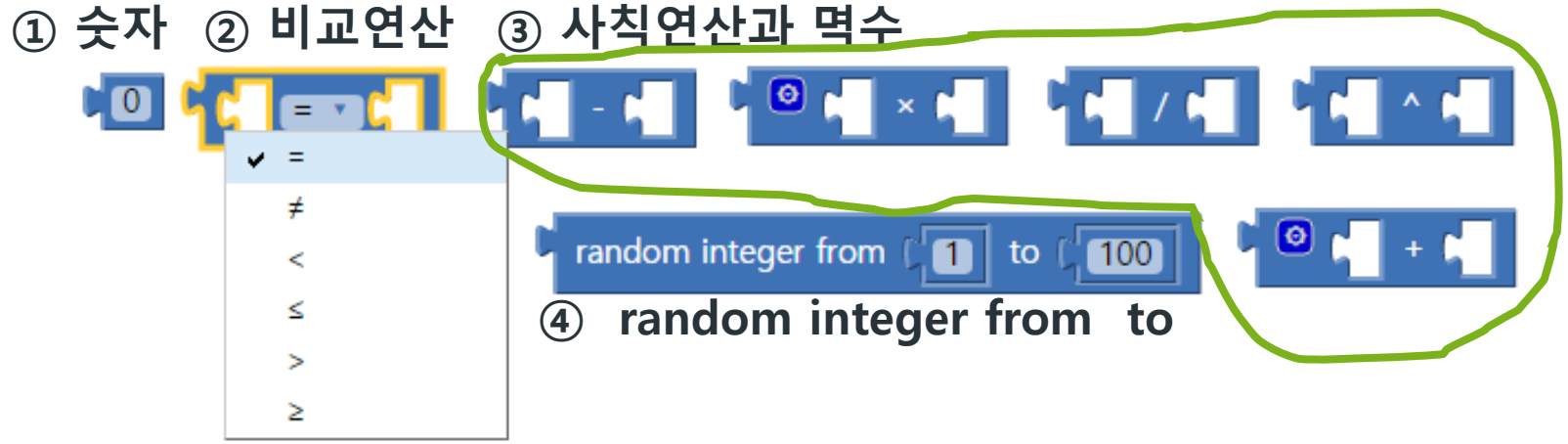
(2) Logic 서랍

- 흐름 제어에 필요한 논리 조건에 사용되는 **초록색** 블록

① “true” 값 블록
② “false” 값 블록
③ “not” 블록. 뒤쪽 소켓에 대한 not값을 돌려줌
④ 양쪽의 값을 비교하여 같으면 “true”, 다르면 “false”
⑤ 양쪽의 논리값이 둘 다 “true”이면 “true”,다르면 “false”
⑥ 양쪽의 논리값이 둘 중 하나가 “true”이면 “true”,다르면 “false”

(3) Math 서랍

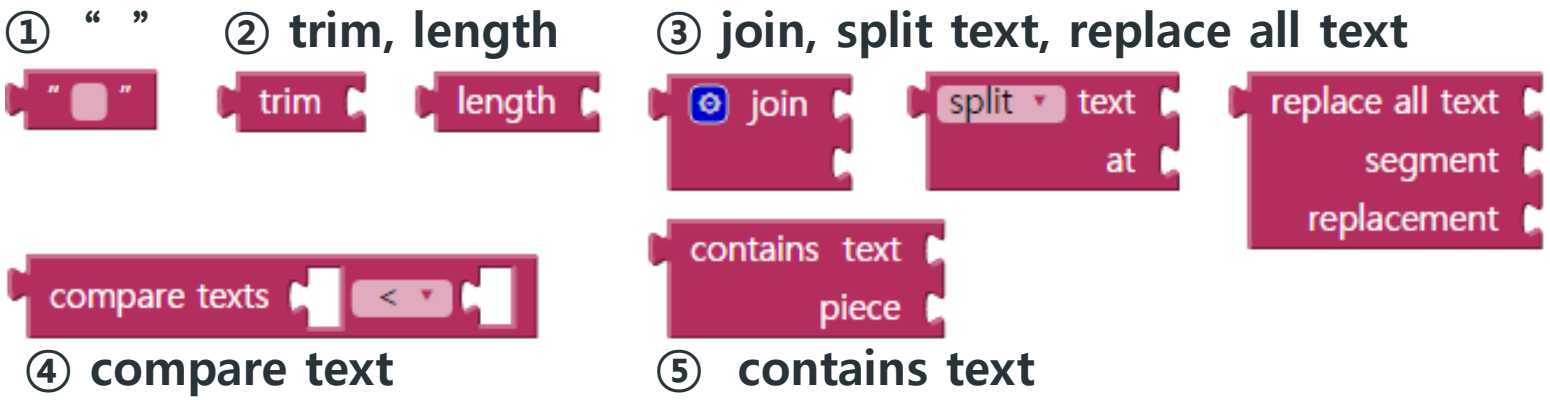
- 기본적인 숫자자체, 비교연산, 사칙연산, 난수 발생 등의 **파란색** 블록



① 기본적인 숫자자체
② 비교연산(=, ≠, <, ≤, >, ≥)
③ 사칙연산과 멱수(+, -, x, /, ^)
④ 난수 발생(정수 1부터 100까지 중에서 무작위 수 1개)

(4) Text 서랍

- 문자열 처리를 위한 **자주색** 블록



① 문자열 자체
② 문자열의 공백 제거나 길이
③ 문자열을 연결하거나 자르거나 대체
④ 문자열 비교
⑤ 문자열 포함 비교

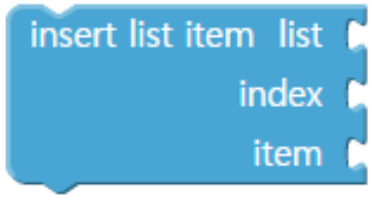
(5) List(목록) 서랍

- 여러 개의 값을 저장하는 변수 공간으로 **청록색** 블록

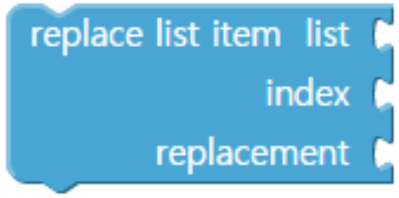
① length of list

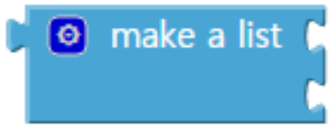
② create empty

③ insert list item

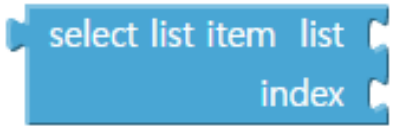


④ replace list item





⑤ make a list



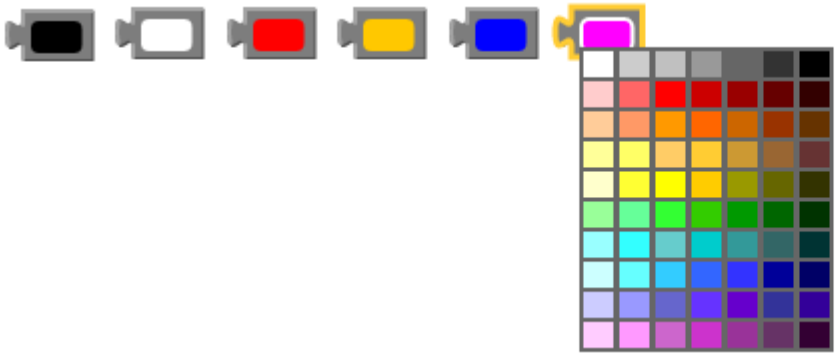
⑥ select list item

① list의 크기(length)를 가져온다.
② 비어 있는 list를 만든다.
③ list에서 정해진 위치(index)에 값(item)을 넣는다.
④ list에서 정해진 위치(index)의 값을 다른 것(replacement)으로 대체한다.
⑤ 소켓에 추가된 값으로 list를 만든다.
⑥ list에서 정해진 위치(index)의 값을 가져온다.

(6) Colors 서랍

- 컴포넌트의 배경이나 글자색을 설정할 수 있는 **회색**의 색상블록

① default colors



② make color

① 기본 13가지 색상의 블록을 가지고 있으며, 색상 블록을 클릭하면 색상 팝업창을 띄워 색상을 선택할 수 있다.

② Red, Green, Blue 3가지 색상으로 혼합하여 색상을 만든다. 각각 256단계의 색을 가지고 있으므로 256x256x256(16,777,216)가지의 색상을 표현할 수 있다.

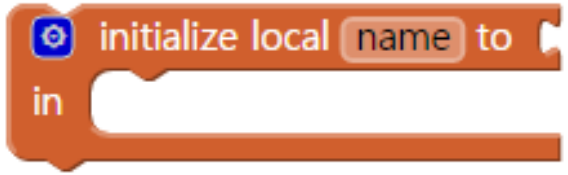
(7) Variables 서랍

- 값을 임시로 저장할 수 있는 변수를 만드는 **주황색** 블록

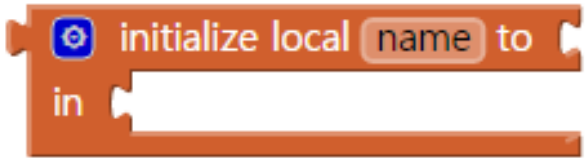
① initialize global ~ to

② get

③ set to



④ initialize local ~ to

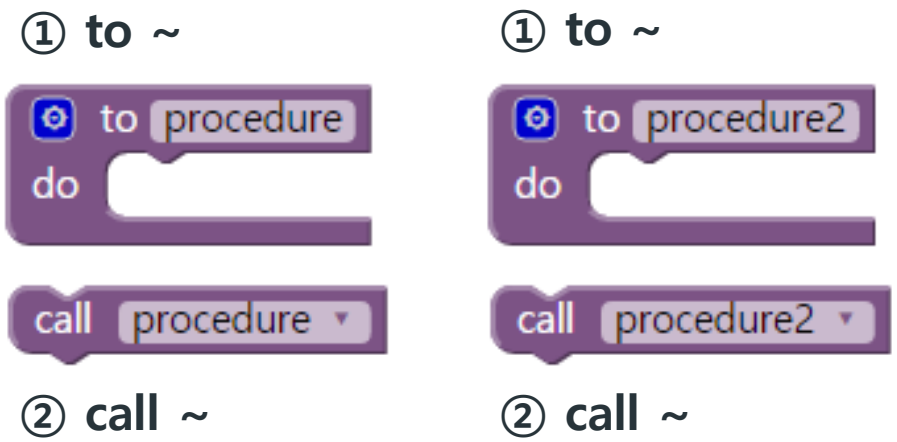


⑤ initialize local ~ to

① “name”이라는 전역 저장 변수를 만든다. “name”은 변경 가능하다.
② get 뒤의 목록 변수 이름에서 저장된 값을 가져온다.
③ to 소켓의 값을 set 뒤의 목록 변수 이름에 저장한다.
④ “name”이라는 지역 저장 변수를 만들어 사용한다. 소켓을 끼울 수 있는 형태로 만든다. “name”은 변경 가능하다.
⑤ “name”이라는 지역 저장 변수를 만들어 사용한다. 플러그를 끼울 수 있는 형태로 만든다. “name”은 변경가능하다.

(8) Procedures 서랍

- 반복적으로 사용되는 기능을 모아 만든 **보라색** 블록
- 기본 내장 procedure와 사용자 정의 procedure를 생성 가능



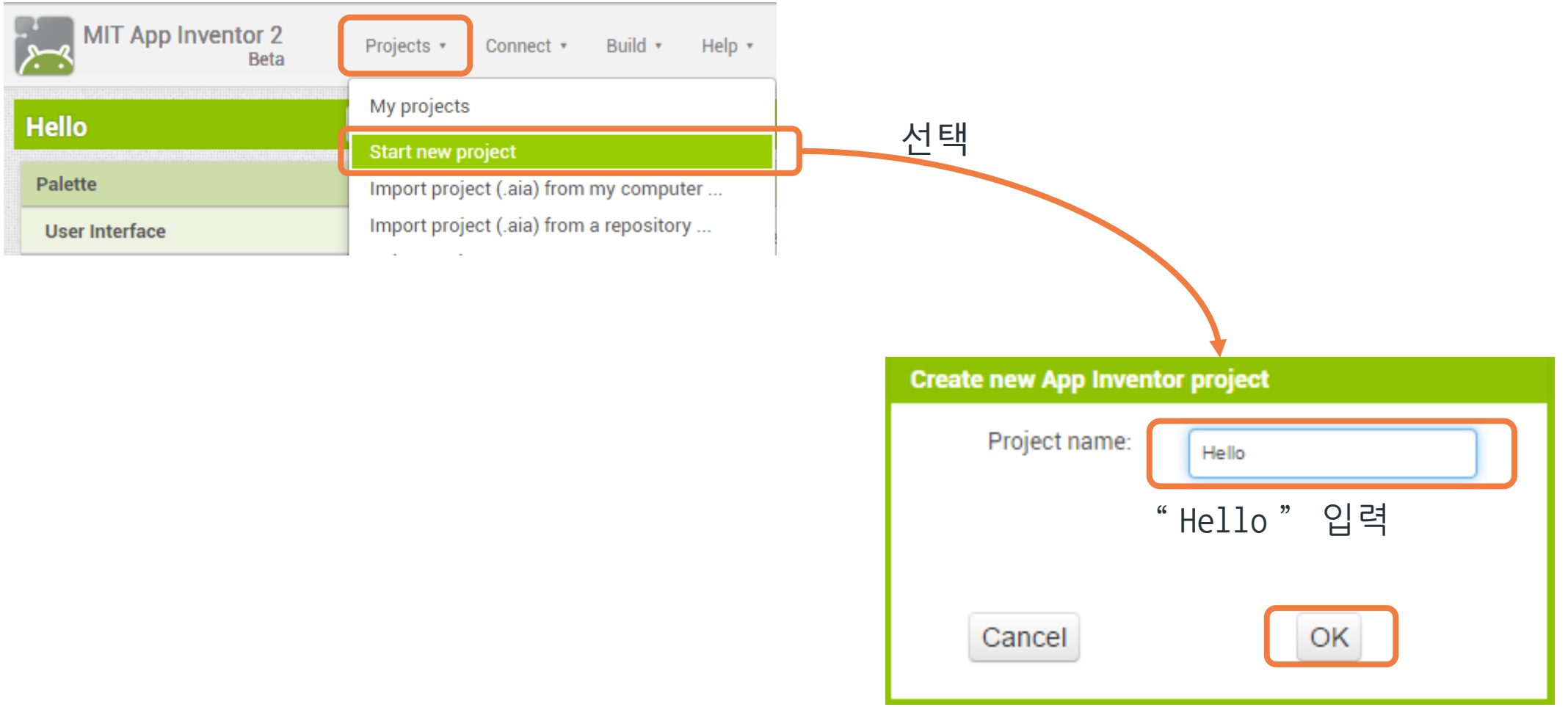
① to 뒤의 “ procedure ” 이름으로 프로시저를 만든다.
“ procedure ” 이름은 변경 가능하다.

② call 뒤의 “ procedure ” 이름의 프로시저를 호출하여 실행한다.

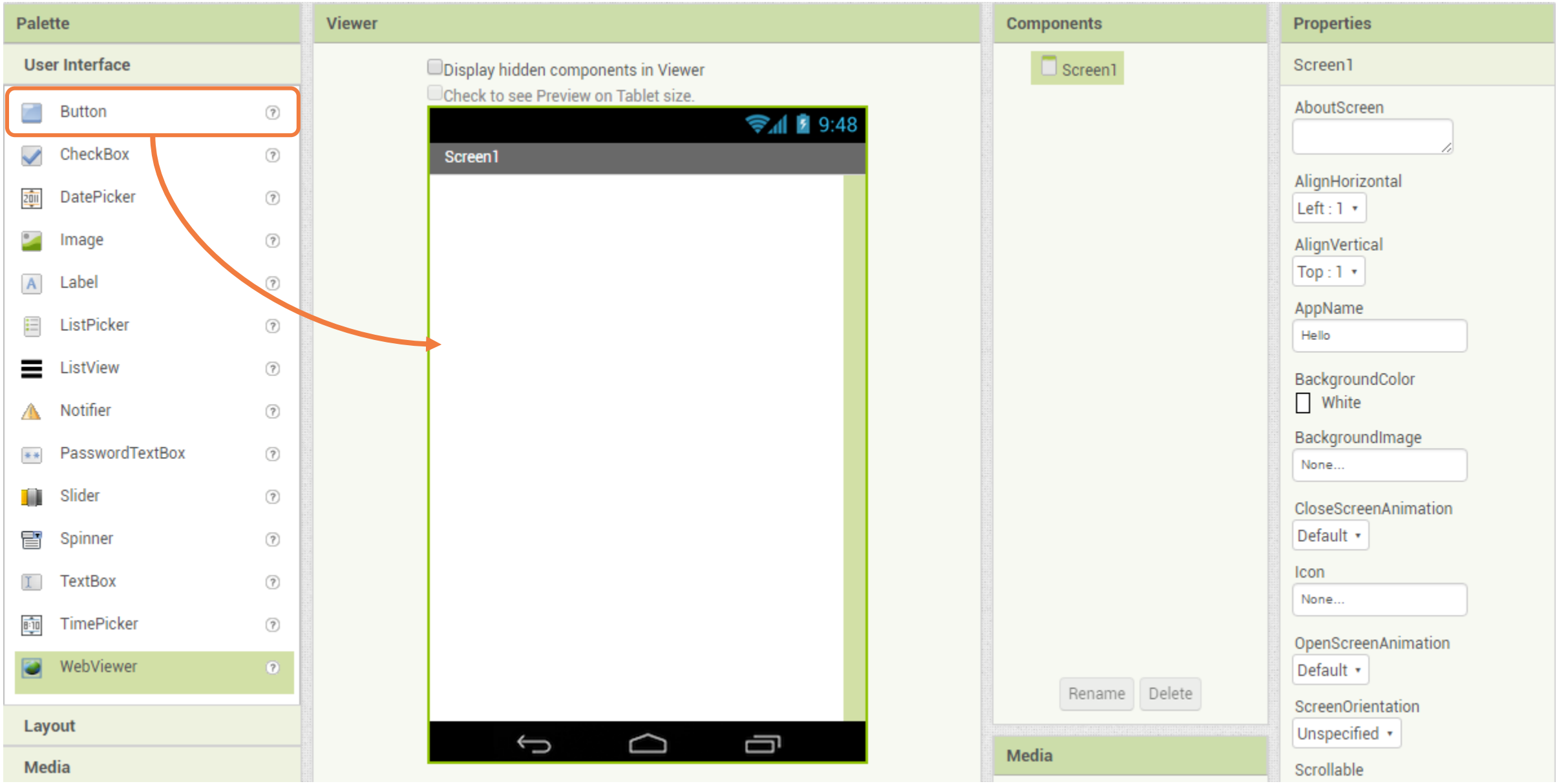
“ 안녕하세요 ”
인사말 띄우기

“안녕하세요” 인사말 띄우기 프로젝트

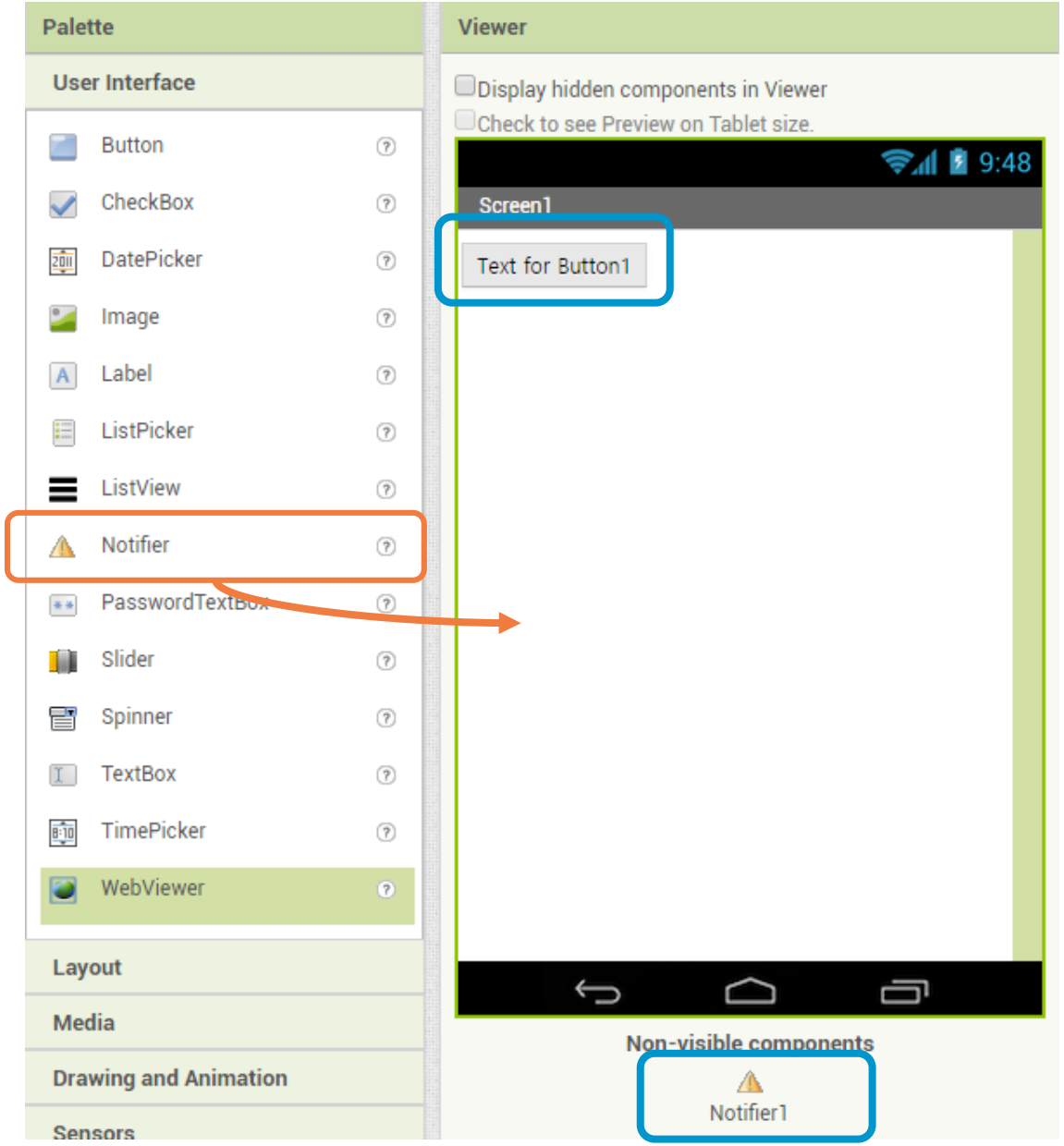
- “Hello”라는 프로젝트를 만든다.



Step 1. 디자이너 편집하기



Step 1. 디자이너 편집하기

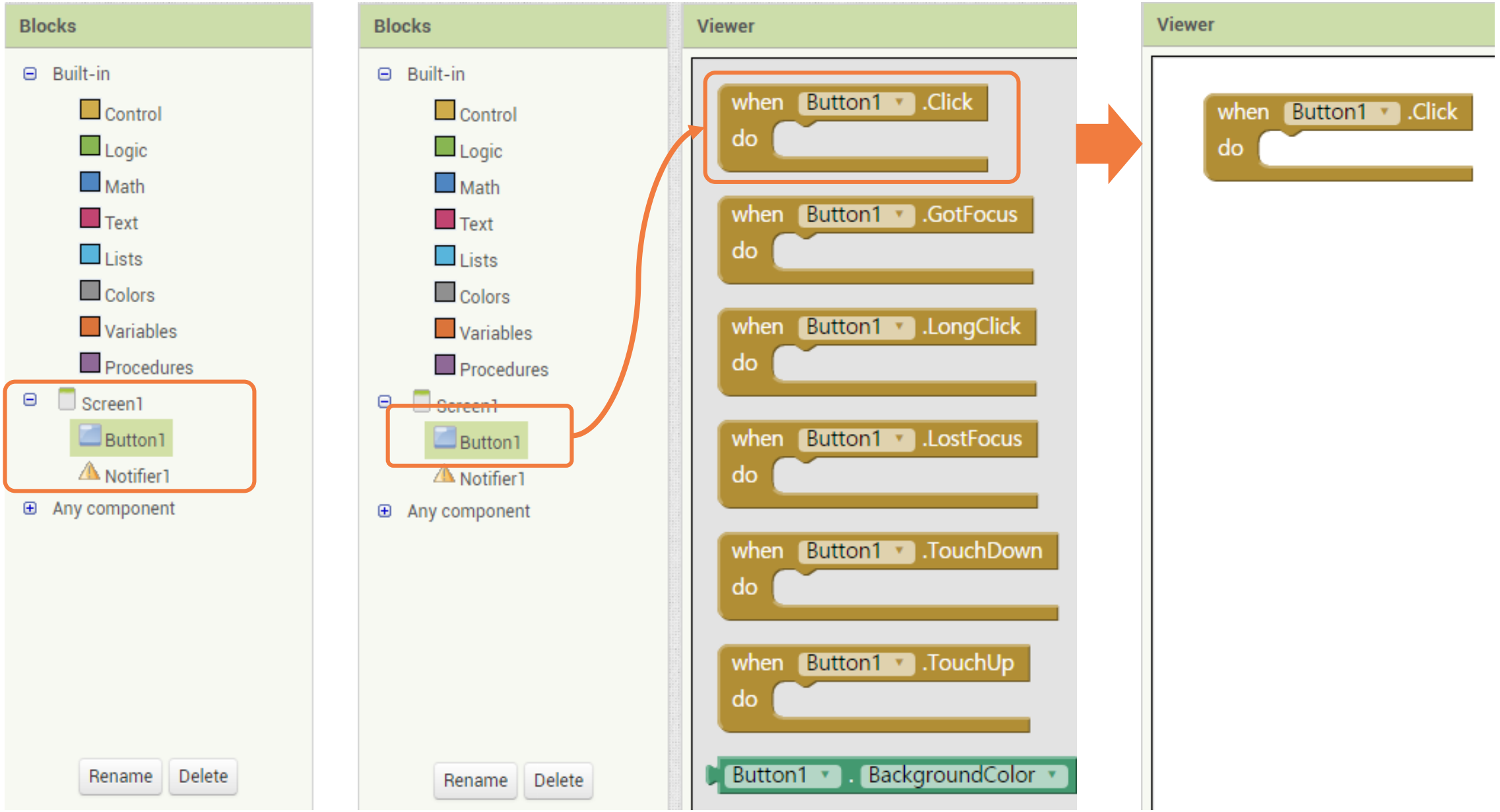


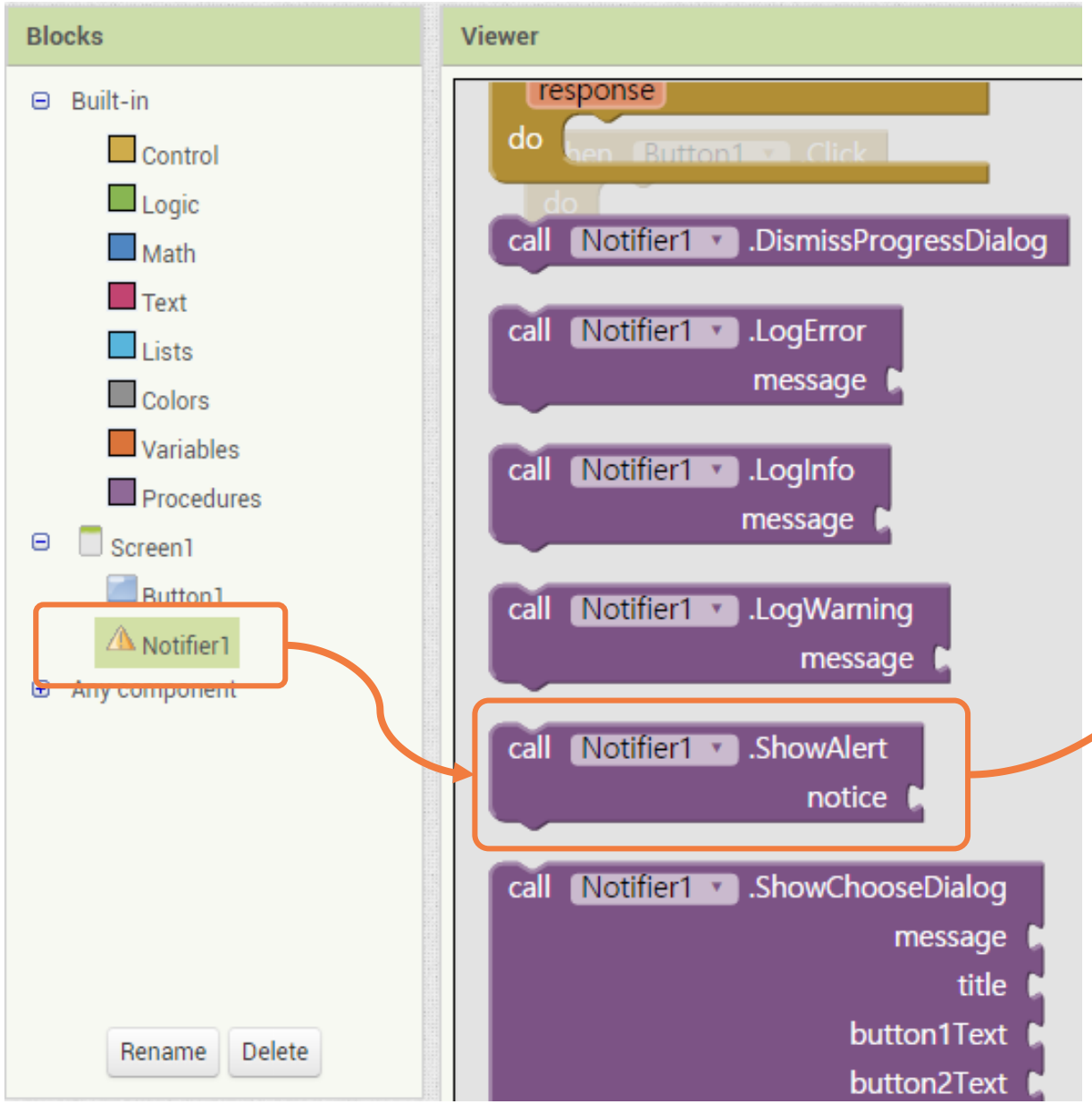
① 디자이너 편집기에서 Button와 Notifier 컴포넌트를 끌어와 Screen1 영역에 삽입
➔ 컴포넌트들은 화면 위에서 순차적으로 표시

② Notifer 컴포넌트는 보이지 않는 요소이므로 화면 아래에 표시

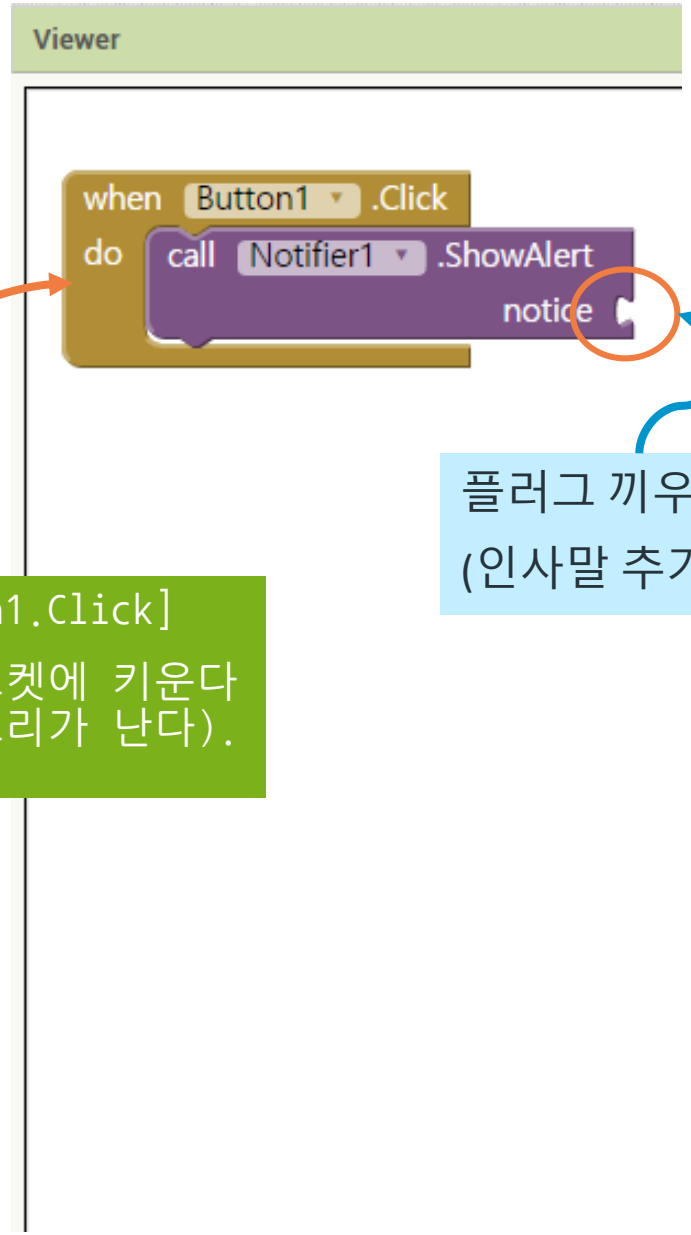
③ [Blocks] 버튼을 눌러 블록 편집기로 이동한다.

Step 2. 블록 편집하기

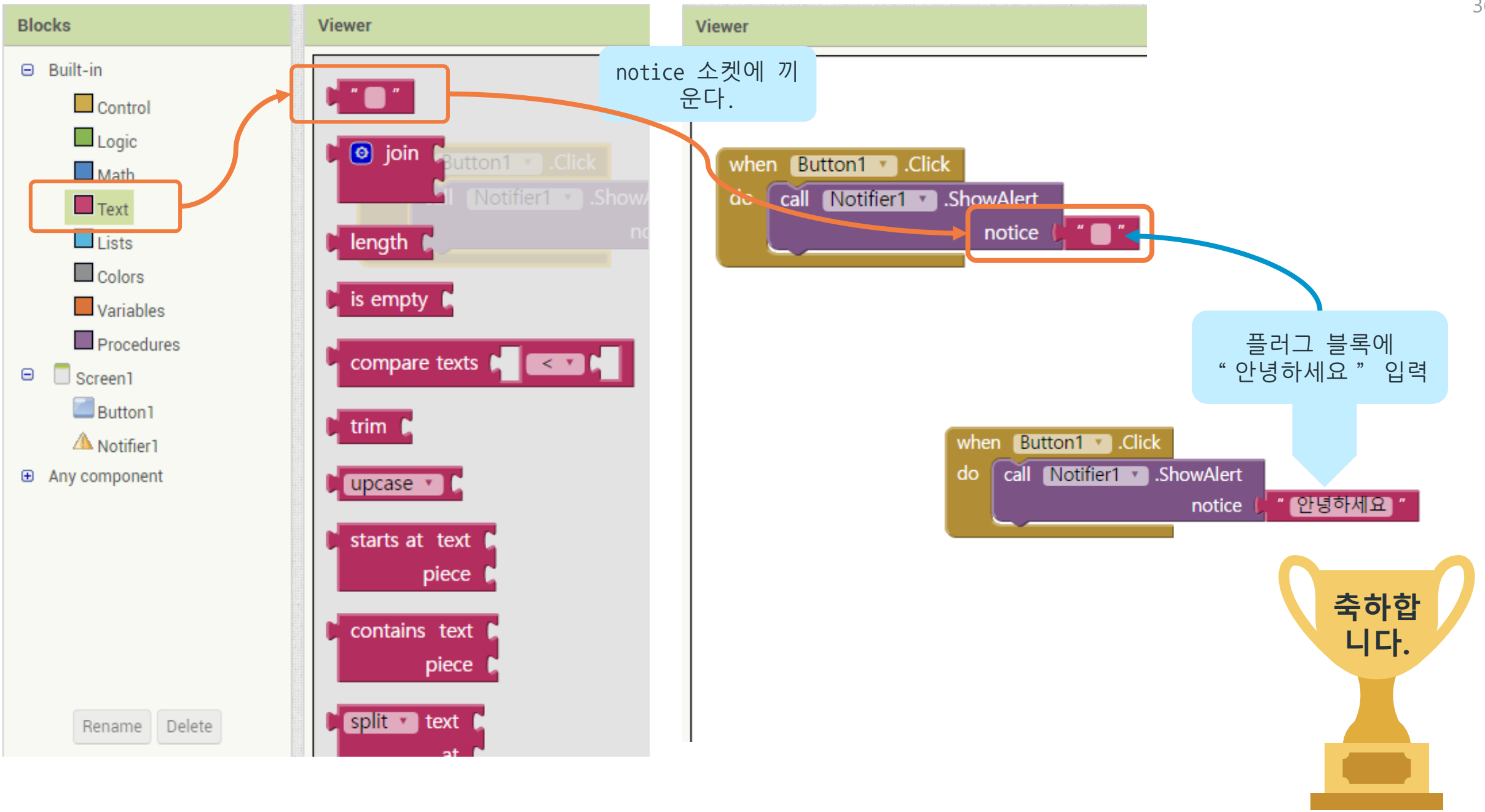




[when Button1.Click]
블록의 do 소켓에 키운다
(“ 딸깍 ” 소리가 난다).



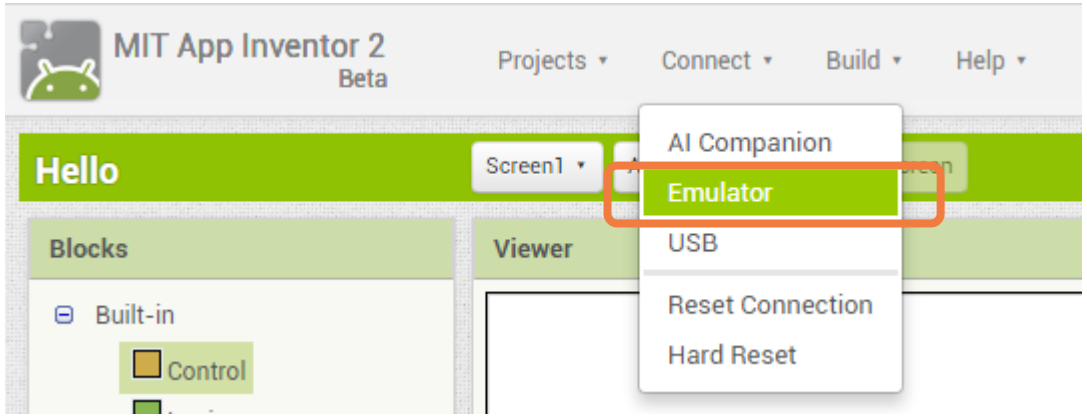
플러그 끼우기
(인사말 추가하기)

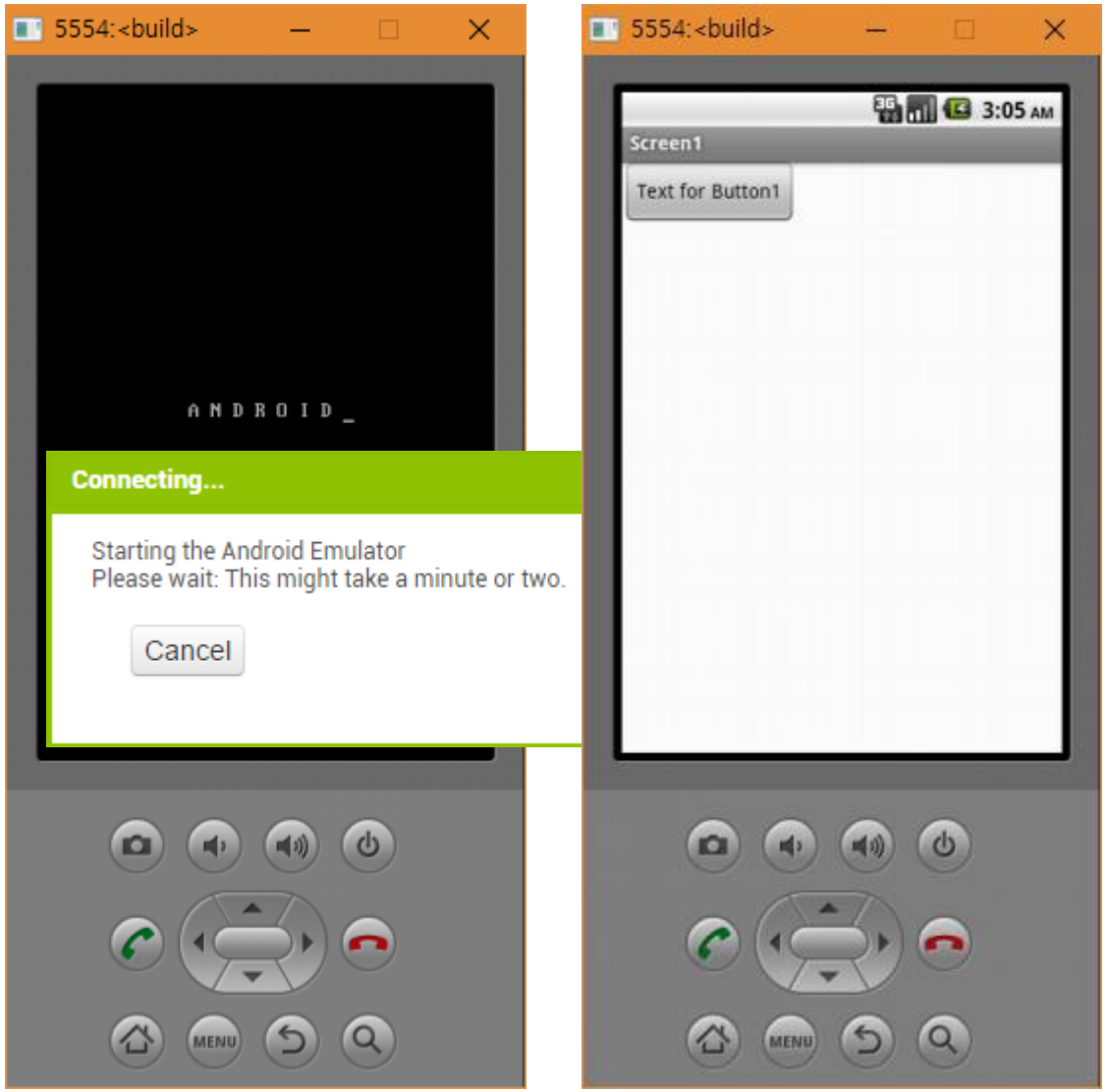

Blocks
Built-in
Control
Logic
Math
Text
Lists
Colors
Variables
Procedures
Screen1
Button1
Notifier1
Any component
Rename
Delete
Viewer
" "
join
length
is empty
compare texts
<
trim
upcase
starts at text
piece
contains text
piece
split text
Viewer
when Button1 .Click
do call Notifier1 .ShowAlert
notice " "
notice 소켓에 끼운다.
플러그 블록에
" 안녕하세요 " 입력
when Button1 .Click
do call Notifier1 .ShowAlert
notice " 안녕하세요 "
축하합니다.

Step 3. 결과 보기 – 에뮬레이터 실행

aiStarter

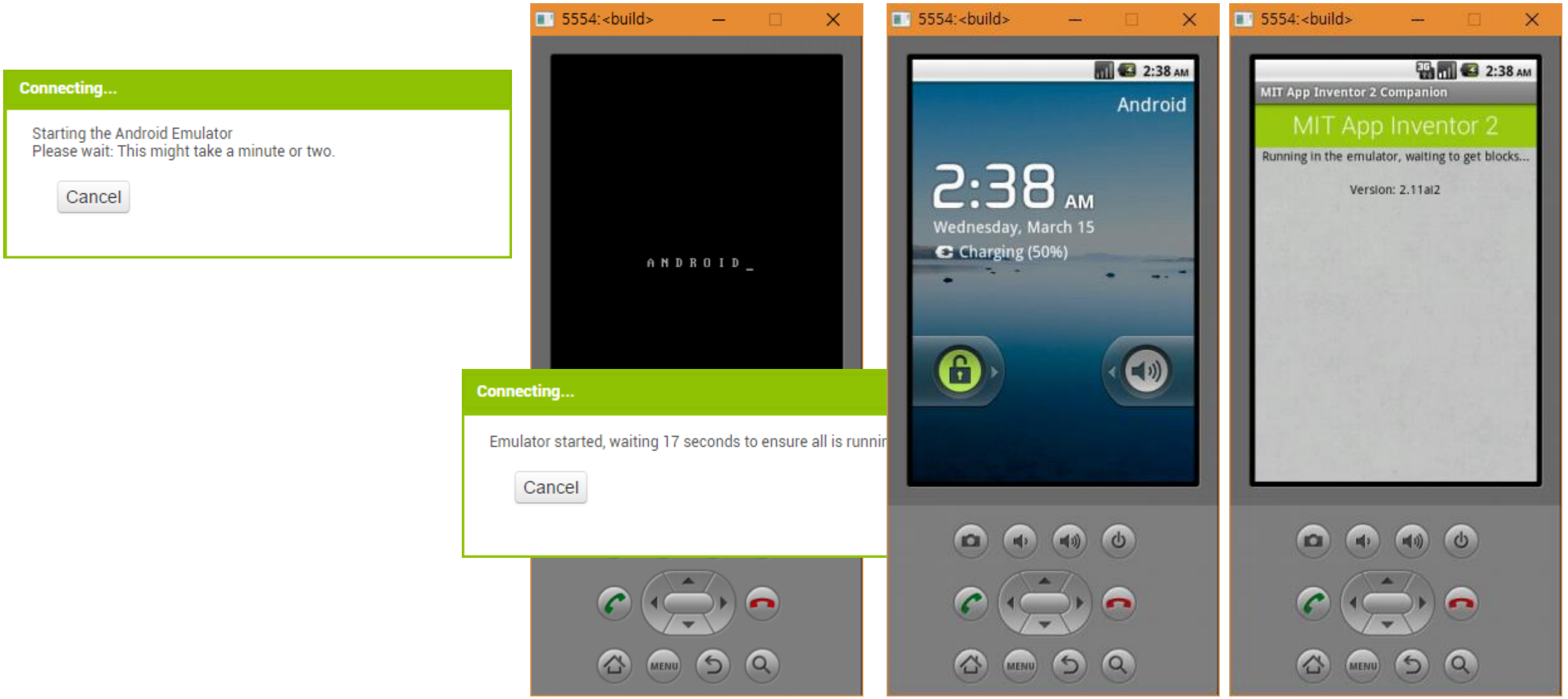
Emulator 실행



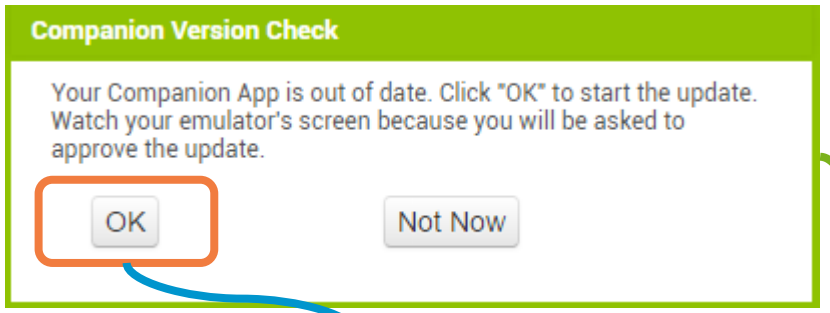


에뮬레이터의
컴패니언 앱 업데이트

컴패니언 앱의 업데이트가 발생하는 경우

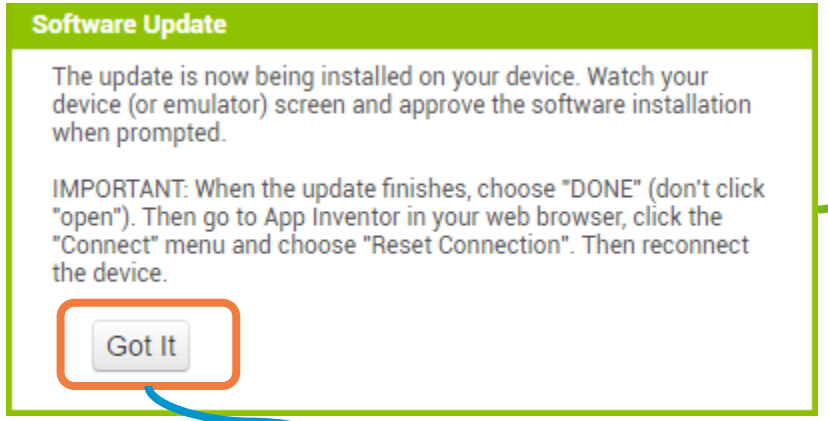


컴패니언 버전 체크



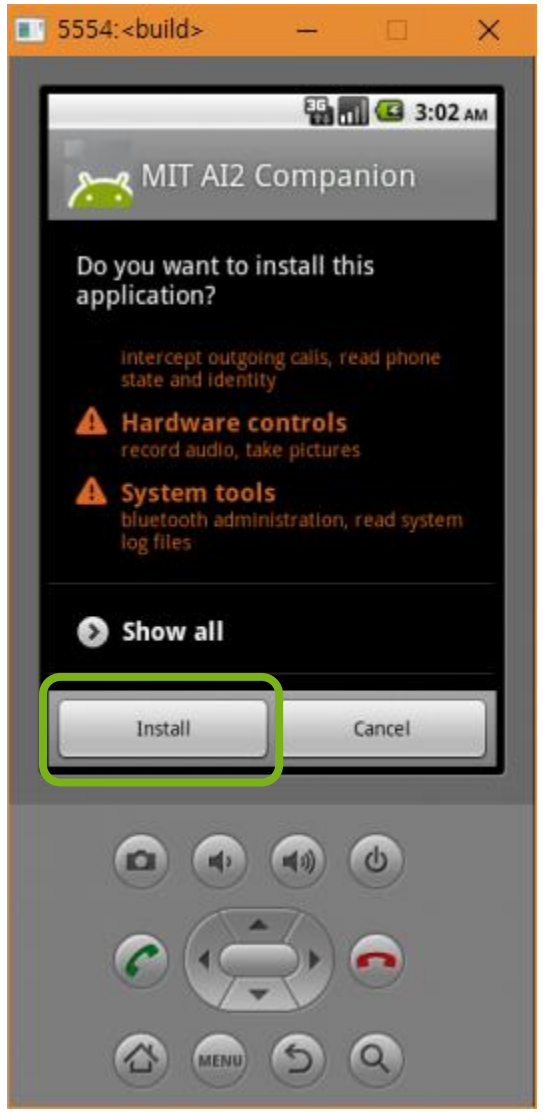
"OK" 버튼을 클릭

소프트웨어 업데이트

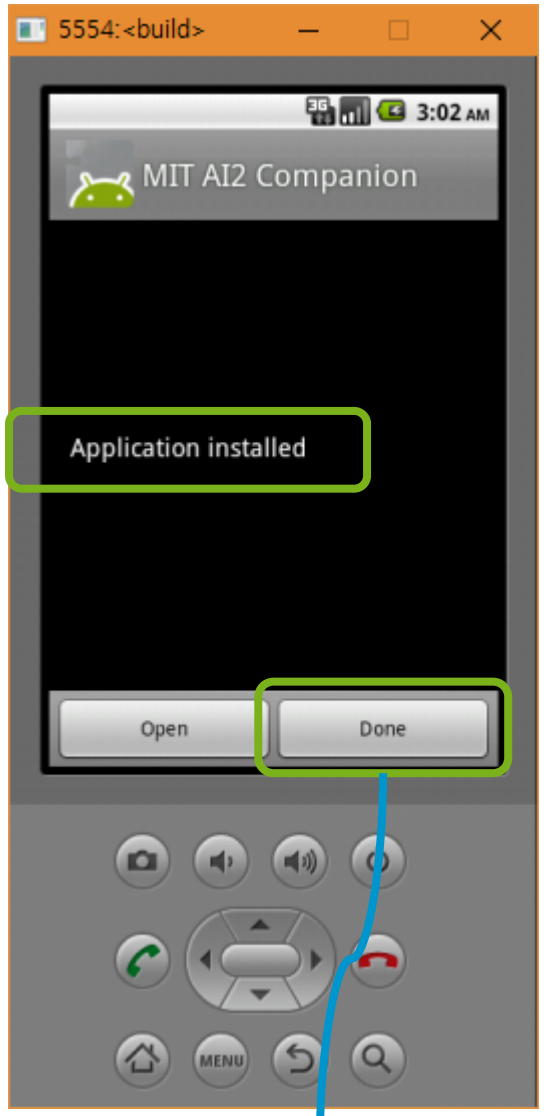


"Got It" 클릭

컴패니언 업데이트 진행...

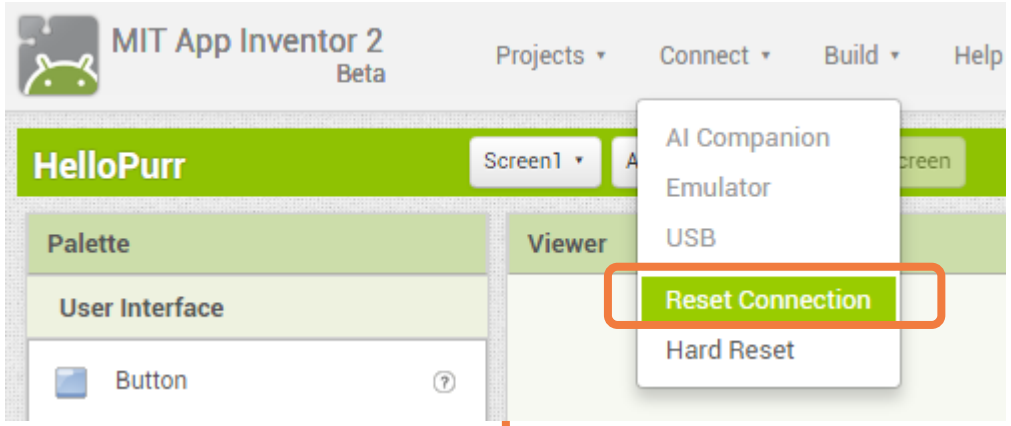


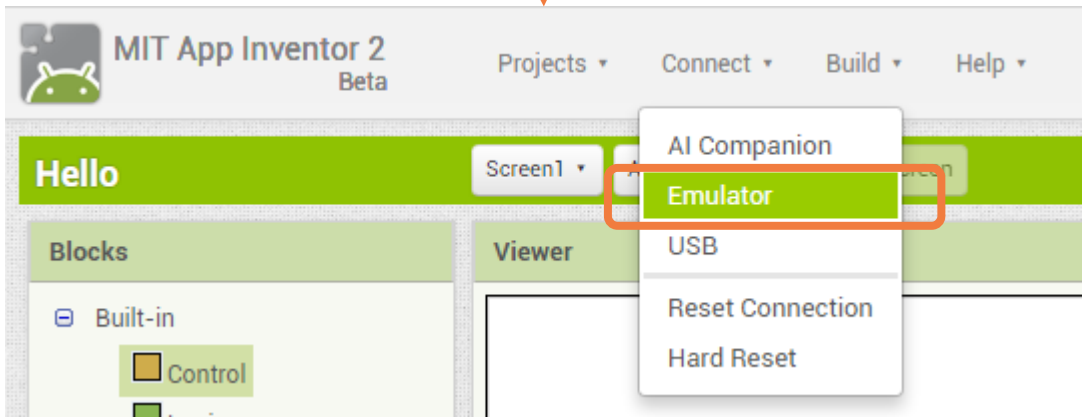
설치 완료.....



반드시 “ Done ” 선택

에뮬레이터 연결 리셋.....





설치 마무리 후 앱 실행.....


5554:<build>
3:05 AM
MIT AI2 Companion
Please Wait...
Installation Finishing
MENU
5554:<build>
3:05 AM
Screen1
Text for Button1
MENU
5554:<build>
3:05 AM
Screen1
Text for Button1
안녕하세요
MENU

Question & Answer