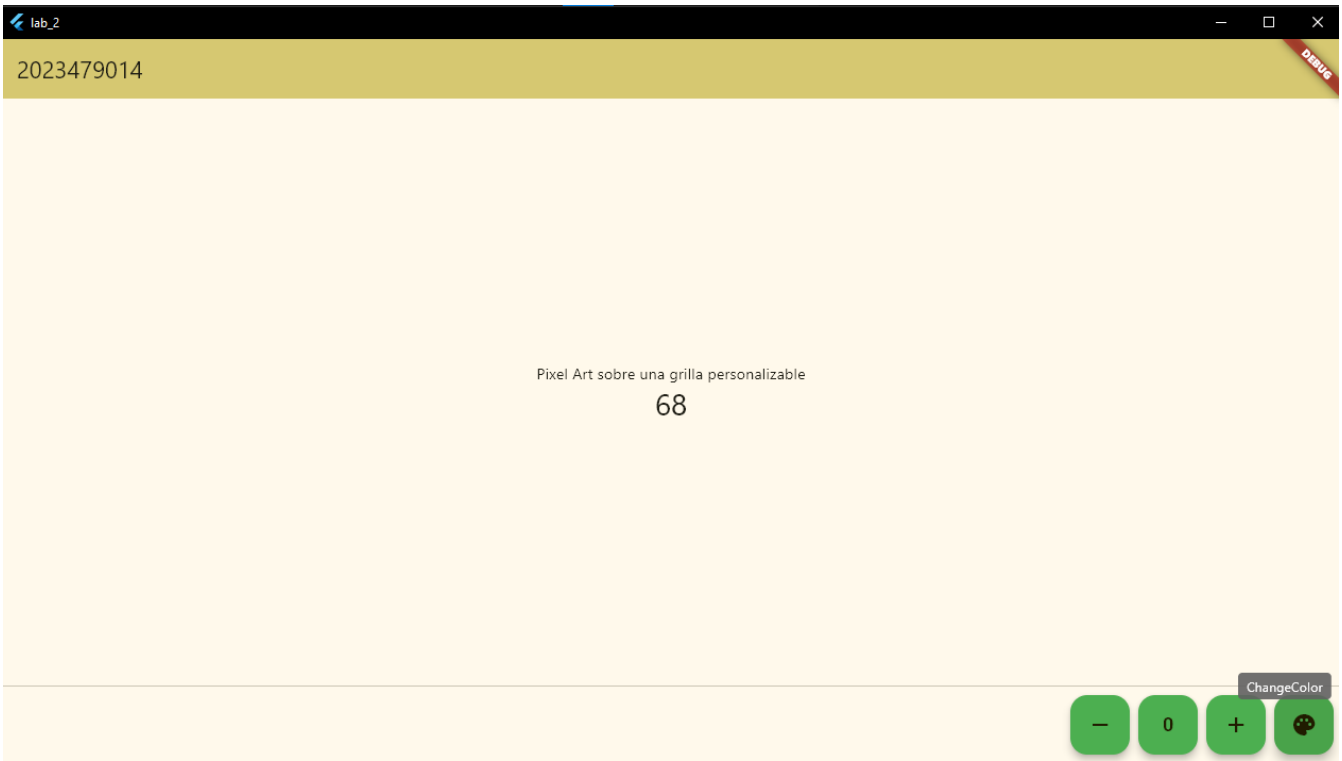
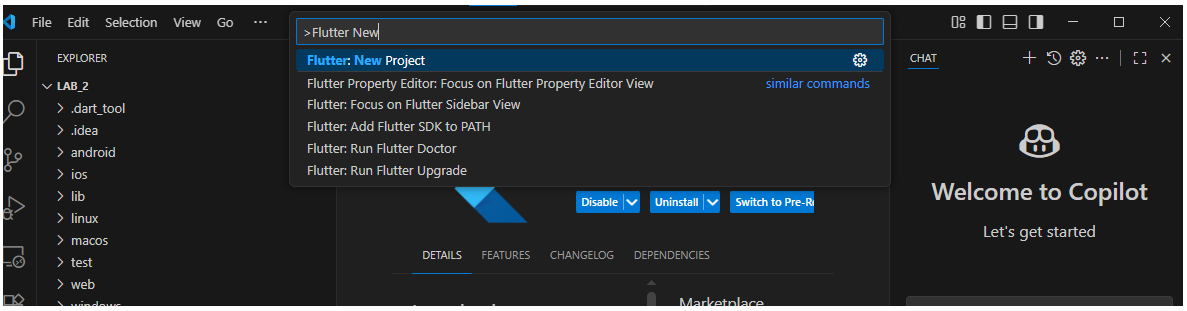
Laboratorio2

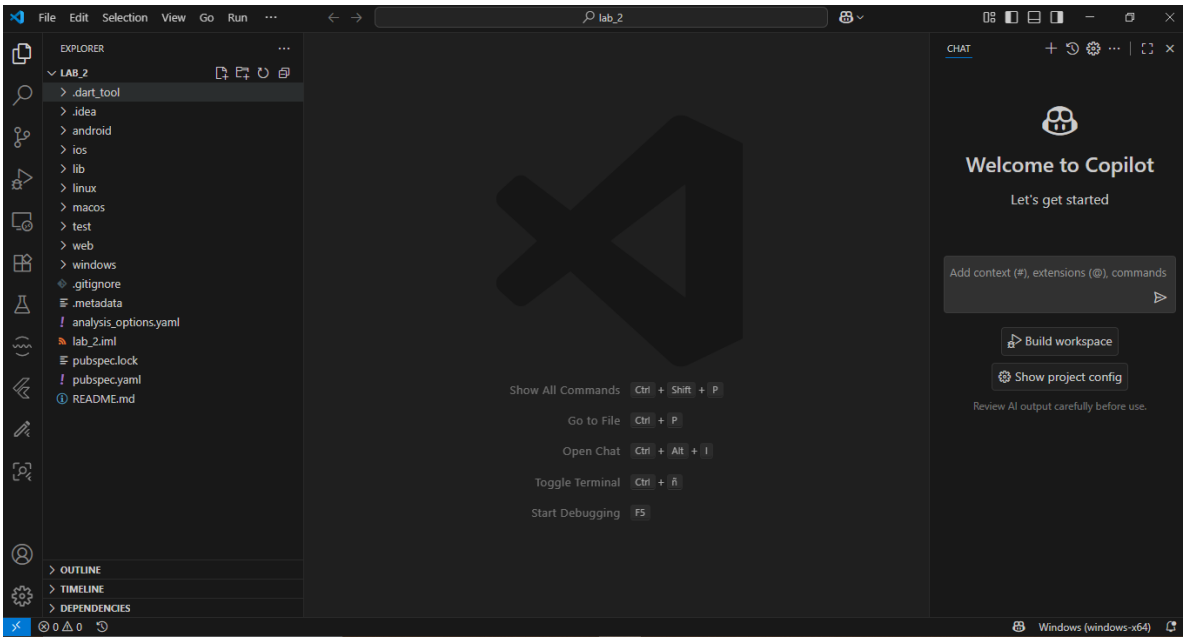
Link al repositorio: https://github.com/TheMob009/3479A321_2023479014

Descripción de lo realizado: En este laboratorio pasamos por los primeros pasos en la creación de un proyecto en Flutter. Se creo un proyecto de Flutter desde VS Code, se ejecuto y debuggeo, con el uso del widget inspector, se asocio el proyecto con un repositorio en Git, mediante consola. Se realizaron cambios visuales básicos, como el titulo y los colores de la aplicación. Se crearon botones persistentes al pie de la aplicación, con funciones designadas para sumar, restar y cambiar el valor de un contador mostrado en el centro del programa, además de uno para alternar entre dos colores para el fondo de los botones. Y finalmente, se hizo uso de la función Refactor para extraer código relacionado a los botones y ponerlo en un método aparte para mantener más limpio el código.

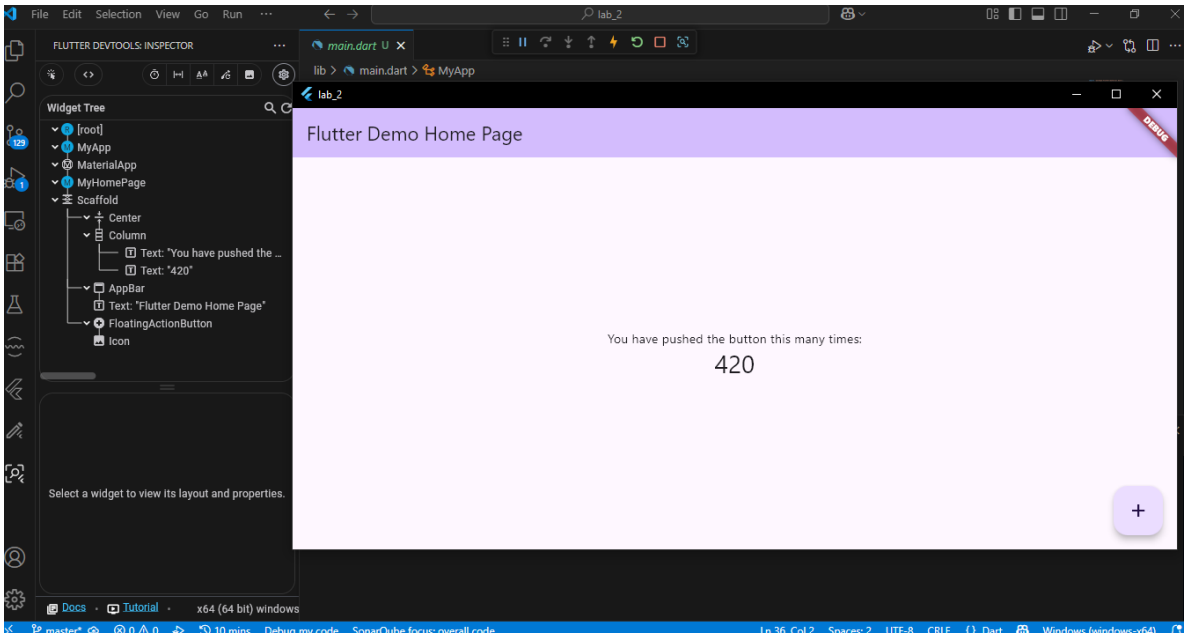


1. **Realizar la creación de un proyecto flutter a través de Visual Studio Code.**
   Para este punto debemos dirigirnos a la consola de Visual Studio Code, ahí yo cree el proyecto de Flutter mediante la línea “flutter crate lab_2”, esto luego de una breve espera creara el proyecto. Tambien se puede crear desde el Command Palette, y seleccionar “Flutter: New Project”.





2. **Realizar la ejecución y debugging del proyecto.**
   Para debuggear el proyecto, le damos al botón “Start Debugging” en la esquina superior derecha, luego de que se descarguen los componentes, se abrirá la siguiente ventana:



3. Realizar la asociación del directorio del proyecto con el repositorio.

Ahora debemos asociar el proyecto a nuestro repositorio de Git. Yo decidí hacerlo mediante consola, esto lo logré con los siguientes comandos, en el orden designado:

-git init

-git remote add origin https://github.com/TheMob009/3479A321_2023479014.git

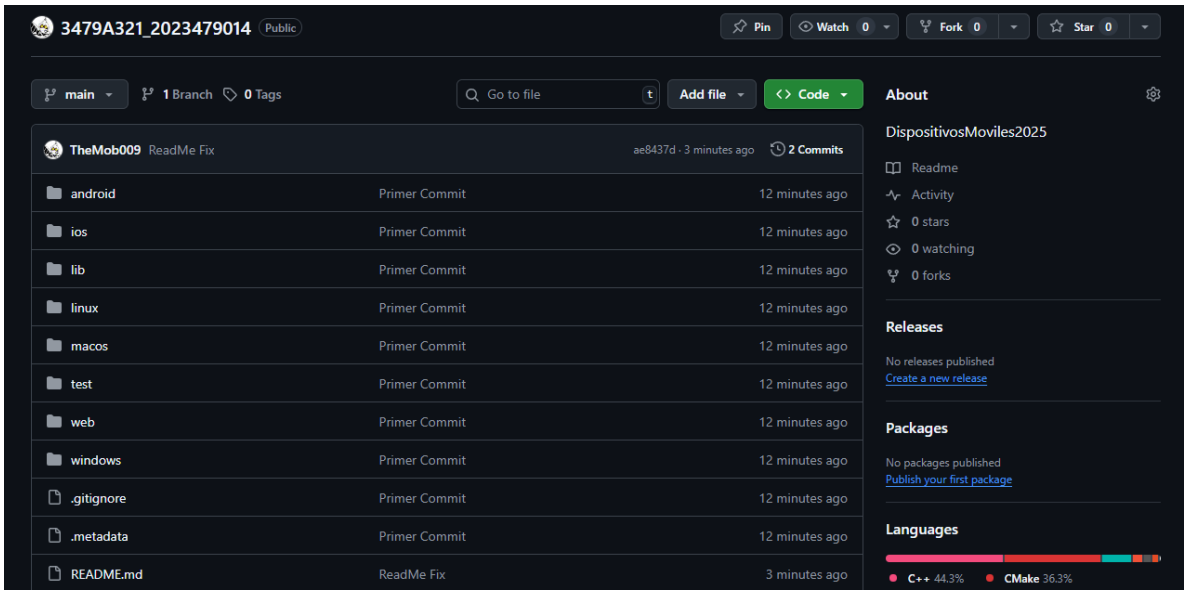
-git branch -M main

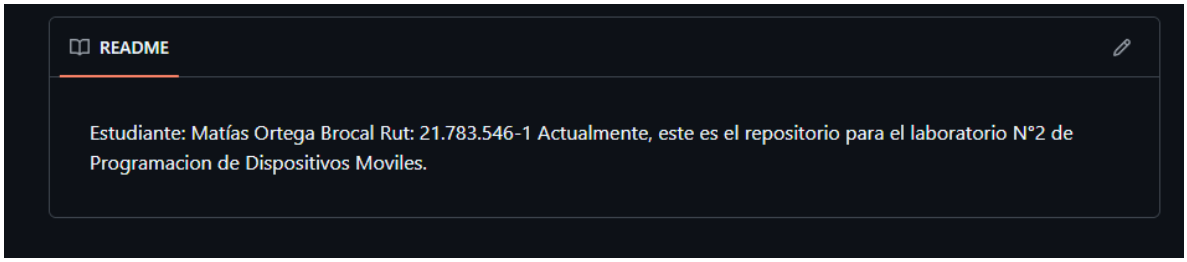
-git add .

-git commit -m "Primer commit"

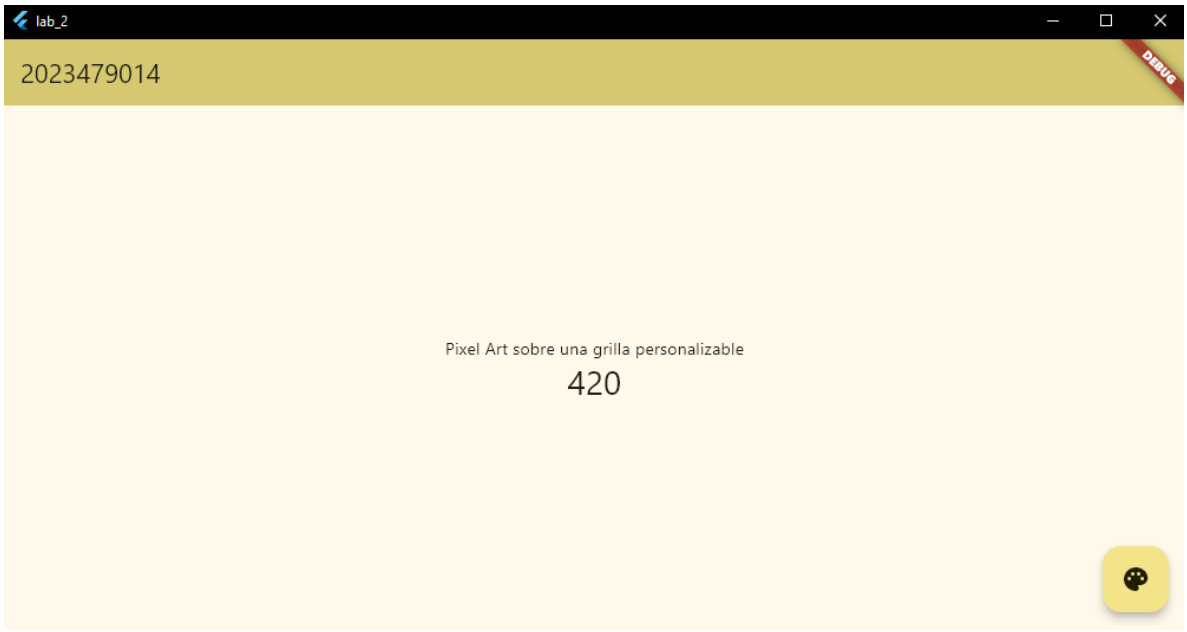
-git push -u origin main

Luego de esto, pedirá autenticar la sesión, luego de hacerlo, ya estará listo nuestro repositorio:





4. **Interactuar con la aplicación en su estado actual. Reconocer el funcionamiento que tiene vigente.**
   Realizar cambios simples en la aplicación actual es relativamente simple. Para cambiar el título, simplemente reemplace el texto actual en el código por el deseado, en este caso, mi matricula. Para cambiar el color reemplace el valor actual de colorScheme. Cambiar el mensaje a “Pixel Art sobre una grilla personalizada” fue lo mismo que cambiar el título. Y finalmente, para cambiar el icono, busque entre una amplia selección dentro de icons dentro de Google Fonts a uno adecuado. Luego se reemplaza en “const Icon(Icons.nombre_icon),”. Finalmente ha quedado así la aplicación por ahora:



5. **Realizar los cambios necesarios para que la aplicación pueda realizar diversas acciones.**
   Acá se nos solicito realizar diversas acciones para nuestro programa, las listare y explicare como realice cada una de estas:
   a. **Incorporar una función que permita disminuir el valor de contador**
      Cree una función en base a la función para incrementar el valor del contador, solo que reemplace el "_counter++" por "_counter--".

```
void _decrementCounter()
{
  setState(() {
    _counter--;
  });
}
```

   b. **Incorporar una función que permita restablecer el valor del contador. Para esto debe considerar la creación de variables para volver al valor "predefinido".**
      Cree una variable extra, "_default_counter", el cual lo establecí a 0, luego hice una nueva función la cual establece el "_counter" al valor de "_default_counter".

```
class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;
  final int _defaultCounter = 0;
```

c. **Incorporar una función que permite definir un nuevo color. Para esto debe considerar la creación de variable para poder almacenar un valor del tipo color.**

Esta fue algo más complicada con relación a las otras, para esta cree dos variables finales de Color, "_colorGreen" y "_colorYellow", y una variable Color llamada "_currentColor", para así poder alternar entre ambas opciones.

```
final Color _colorYellow = Colors.yellow;
final Color _colorGreen = Colors.green;

Color _currentColor = Colors.yellow;
```

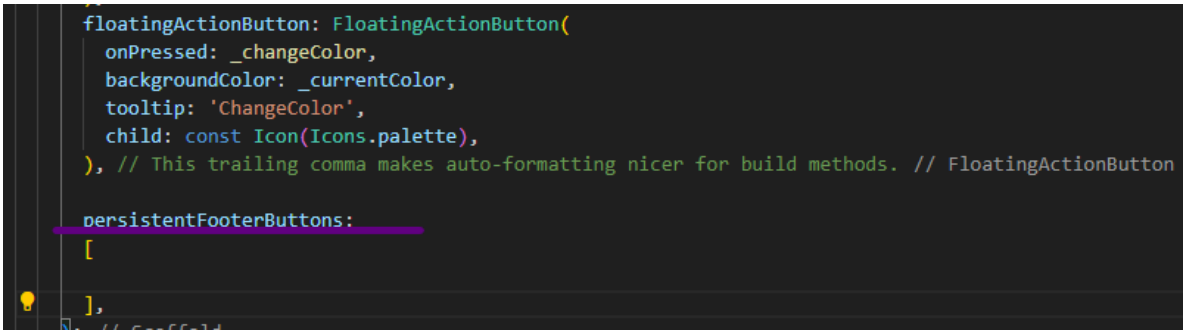
Teniendo ya estas variables, cree una función la cual alterna entre ambos colores según el color actual, cambiando de uno al otro.

```
void _changeColor()
{
  setState(()
  {
    if(_currentColor == _colorYellow)
    {
      _currentColor = _colorGreen;
    }
    else
    {
      _currentColor = _colorYellow;
    }
  });
}
```

La variable "_currentColor" es usada para el background de los botones, para implementar esto al momento de crear un nuevo botón, debemos agregar una línea que establece el color del background del botón a la variable "_currentColor", de manera que, al actualizar el estado del programa, también cambie el color de los botones.

```
floatingActionButton: FloatingActionButton(
  onPressed: _changeColor,
  backgroundColor: _currentColor,
  tooltip: 'ChangeColor',
  child: const Icon(Icons.palette),
```

d. **Incorporar al Widget Scaffold la opción de "Botones persistentes del pie" en la parte inferior de la pantalla.**
Implementar la función para tener botones persistentes al pie de la pantalla, debemos usar la siguiente línea:



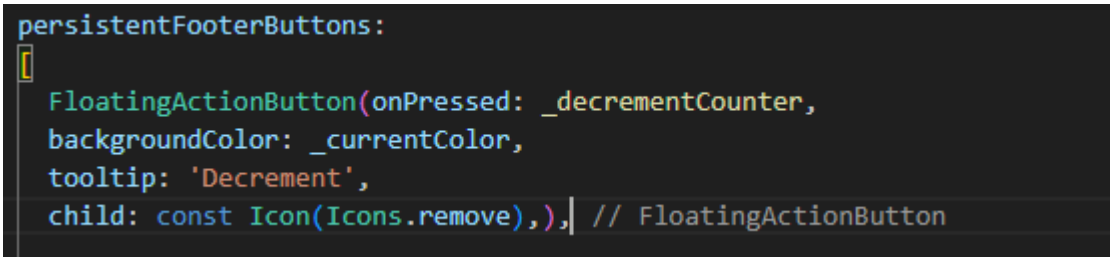
```
floatingActionButton: FloatingActionButton(
  onPressed: _changeColor,
  backgroundColor: _currentColor,
  tooltip: 'ChangeColor',
  child: const Icon(Icons.palette),
), // This trailing comma makes auto-formatting nicer for build methods. // FloatingActionButton

persistentFooterButtons:
[

],
```

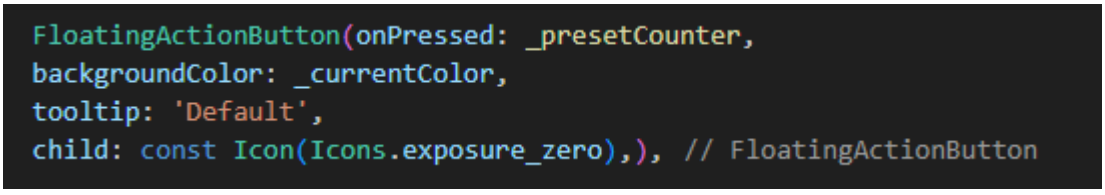
Esto creara un arreglo, donde se nos permite agregar botones flotantes, estos se ordenan automáticamente en la parte inferior de la pantalla.

e. **Agregar un botón que permita restar, con un icono correspondiente.**
Simple, creamos un nuevo botón dentro del arreglo creado anteriormente, le asignamos la función correspondiente al apretarlo, junto a un icono que represente correctamente la función que comete.



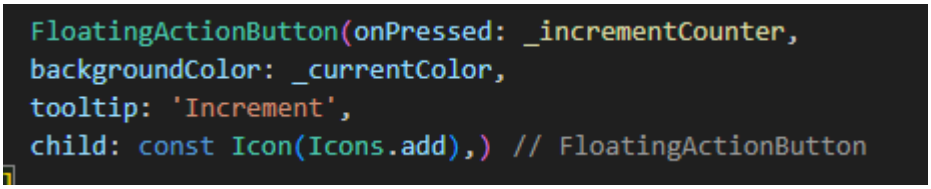
```
persistentFooterButtons:
[
  FloatingActionButton(onPressed: _decrementCounter,
  backgroundColor: _currentColor,
  tooltip: 'Decrement',
  child: const Icon(Icons.remove),), // FloatingActionButton
```

f. **Agregar un botón que permita sumar, con un icono correspondiente.**
Lo mismo que el botón anterior.



```
FloatingActionButton(onPressed: _presetCounter,
backgroundColor: _currentColor,
tooltip: 'Default',
child: const Icon(Icons.exposure_zero),), // FloatingActionButton
```

g. **Agregar un botón que permita restaurar, con un icono correspondiente**
Lo mismo que el botón anterior.



```
FloatingActionButton(onPressed: _incrementCounter,
backgroundColor: _currentColor,
tooltip: 'Increment',
child: const Icon(Icons.add),) // FloatingActionButton
```

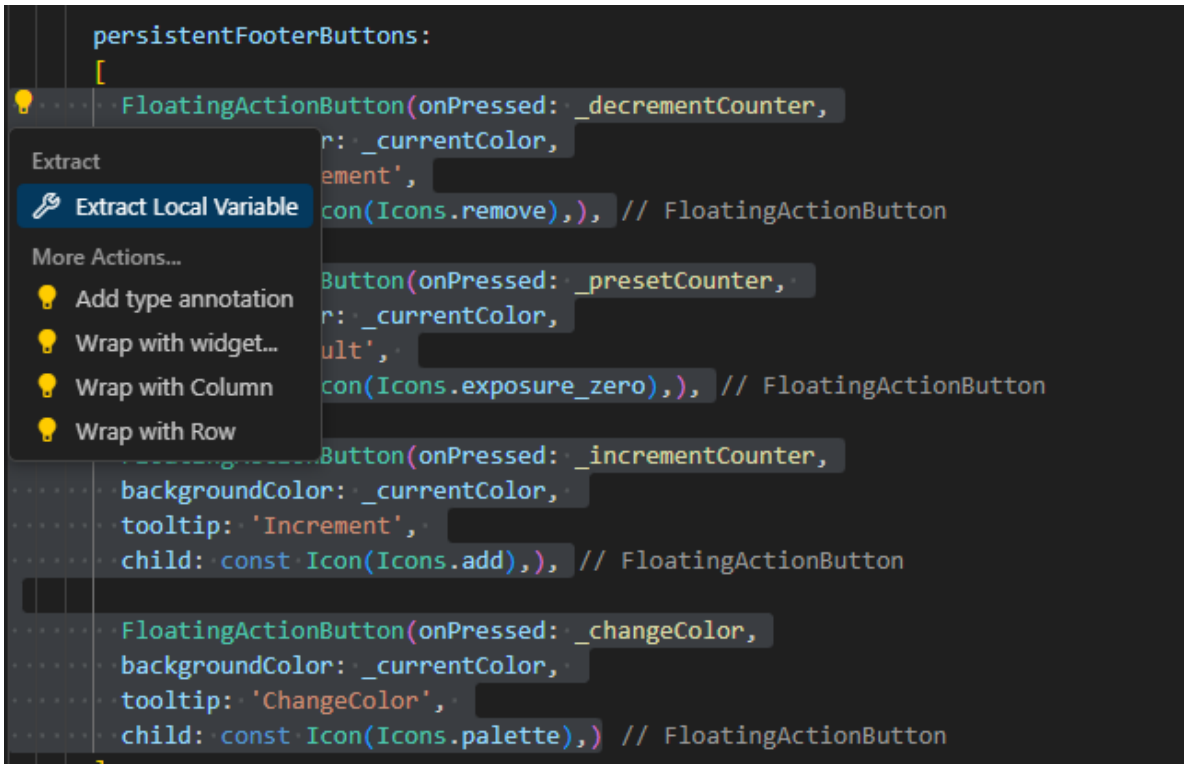
h. **Agregar un botón que permita seleccionar un color, con un icono correspondiente.**

Bueno, perdonar la redundancia. Lo mismo que el botón anterior.

```
FloatingActionButton(onPressed: _changeColor,
backgroundColor: _currentColor,
tooltip: 'ChangeColor',
child: const Icon(Icons.palette),) // FloatingActionButton
```

6. **Realizar la extracción del contenido incluido en los botones persistentes de la parte inferior a un método.**

Como ejercicio final del laboratorio, se nos solicita usar Refactor para ordenar nuestro código, y así familiarizarnos con esta función. Para usar Refactor en VS Code debemos seleccionar un bloque de código, luego darle a click derecho y elegir Refactor, o el atajo “Ctrl+Shift+R”, luego escogemos la opción adecuada. En este caso se nos pide un método que tenga los cuatro botones definidos en la parte inferior, para esto seleccionamos el siguiente código:



Acá elegimos la opción destacada, “Extract Local Variable”, esto nos dejara lo siguiente:

```
var persistentFooterButtons = [
    FloatingActionButton(onPressed: _decrementCounter,
    backgroundColor: _currentColor,
    tooltip: 'Decrement',
    child: const Icon(Icons.remove),), // FloatingActionButton

    FloatingActionButton(onPressed: _presetCounter,
    backgroundColor: _currentColor,
    tooltip: 'Default',
    child: const Icon(Icons.exposure_zero),), // FloatingActionButton

    FloatingActionButton(onPressed: _incrementCounter,
    backgroundColor: _currentColor,
    tooltip: 'Increment',
    child: const Icon(Icons.add),), // FloatingActionButton

    FloatingActionButton(onPressed: _changeColor,
    backgroundColor: _currentColor,
    tooltip: 'ChangeColor',
    child: const Icon(Icons.palette),) // FloatingActionButton
  ];
```

```
persistentFooterButtons:
persistentFooterButtons,
```

Para finalizar el laboratorio, decidí borrar el botón viejo de incrementar (a pesar de que no se solicite), el cual no estaba dentro del pie del programa, para que la aplicación se vea de la siguiente manera: