

Les gestures sur Flutter

Table des matières

I. Comprendre les gestes	3
A. Pointeurs.....	3
B. Différents types de gestes	3
II. Exercice : Quiz	4
III. Utiliser les gestes	5
A. Gesture Detector.....	5
B. Utiliser les built-in	6
IV. Exercice : Quiz	7
V. Essentiel	8
VI. Auto-évaluation	9
A. Exercice	9
B. Test.....	9
Solutions des exercices	10

I. Comprendre les gestures

Contexte

Comme vous le savez, Flutter est un framework front-end, c'est-à-dire que nous l'utilisons pour imaginer des interfaces qui vont permettre de faire le lien entre un utilisateur et un système plus complexe (le back-end).

Lorsque les ordinateurs ont commencé à voir le jour, nous pouvions interagir avec au moyen de claviers. Puis la souris est apparue, ce qui nous a permis de transmettre des gestes de plus en plus complexes aux interfaces (le double-clic, par exemple !). Aujourd'hui, une grande partie de nos interfaces est dotée d'un écran tactile faisant office de souris, comme nos smartphones.

Vous l'aurez donc compris, les gestes sont indispensables pour pouvoir échanger avec nos ordinateurs, et c'est pourquoi Flutter est capable d'en détecter 6 catégories.

Dans ce cours, nous allons tout d'abord étudier comment Flutter détecte ces gestes et quelles sont ces catégories de gestes, avant de mettre tout ça en pratique en étudiant `GestureDetector` et les détecteurs intégrés.

A. Pointeurs

Avant de commencer à regarder comment inclure les *gestures* dans votre application, regardons ce qu'il se passe à « bas niveau » et comment Flutter réussit à interpréter nos gestes !

Avant même de pouvoir réagir en fonction de vos gestes, Flutter doit en effet être notifié que vous êtes en train d'interagir. Pour cela, il y a les pointeurs, qui sont des données brutes sur l'interaction d'un utilisateur avec l'écran.

Ces pointeurs peuvent correspondre à 4 types d'événements :

- Un `PointerDownEvent` correspondra à un pointeur qui aura touché l'écran à un endroit particulier.
- `PointerMoveEvent` correspond au fait de bouger le pointeur d'un endroit à un autre de l'écran.
- Si vous relevez le pointeur, ce sera un `PointerUpEvent`.
- Enfin, `PointerCancelEvent` correspond au fait de déclarer qu'un pointeur n'est plus lié à une certaine application ou widget.

Lorsqu'il y a un `PointerDownEvent`, Flutter détermine quel widget est situé à l'endroit où le pointeur a touché l'écran. Cet événement sera ensuite transmis au widget le plus proche. Puis l'événement est transmis du widget pour remonter à la racine.

Il est possible d'écouter ces événements directement depuis un widget avec un `Listener`. Néanmoins, la plupart du temps, nous utilisons les gestures, que nous verrons par la suite.

B. Différents types de gestures

Flutter est capable d'interpréter plusieurs événements de pointeurs pour déterminer quel geste nous faisons exactement.

Par exemple, si nous touchons l'écran (`PointerDownEvent`), que nous glissons notre doigt légèrement vers le haut (`PointerMoveEvent`), puis que nous relâchons (`PointerUpEvent`), le framework sera en mesure de déterminer qu'il s'agissait d'un « vertical drag ». En effet, il aura interprété le `PointerDownEvent` comme un `onVerticalDragStart`, le `PointerMoveEvent` comme un `VerticalDragUpdate`, et le `PointerUpEvent` comme un `onVerticalDragEnd`.

Flutter compte ainsi 6 grandes catégories de gestures :

- **Tap**

- `onTapDown`**

- Un pointeur peut entraîner un événement `Tap` à toucher l'écran à un endroit particulier.

onTapUp

Un pointeur a arrêté de toucher l'écran à un endroit particulier et va entraîner un événement Tap.

onTap

Le pointeur a déclenché les événements `onTapDown` et `onTapUp` ce qui crée un événement `onTap`.

onTapCancel

Le pointeur qui a déclenché `onTapUp` ne va finalement pas déclencher un événement Tap.

- **Double tap**

onDoubleTap

L'utilisateur a déclenché l'événement Tap deux fois dans un court laps de temps.

- **Long press**

onLongPress

Le pointeur est resté en contact avec l'écran pendant une longue période.

- **Vertical drag**

onVerticalDragStart

Le pointeur a touché l'écran et va peut-être commencer à bouger verticalement.

onVerticalDragUpdate

Le pointeur qui est en contact avec l'écran et qui bouge est en train de bouger dans la direction verticale.

onVerticalDragEnd

Le pointeur qui était avant en contact avec l'écran et qui bougeait en direction verticale n'est plus en contact.

- **Horizontal drag**

Même chose que le Vertical Drag, mais pour la direction horizontale (`onHorizontalDragStart`, `onHorizontalDragUpdate`, `onHorizontalDragEnd`).

- **Pan**

Attention : ce *callback* peut causer un crash si les gestes `onHorizontalDragStart` ou `onVerticalDragStart` sont également définis.

onPanStart

Le pointeur a touché l'écran et peut commencer à bouger verticalement ou horizontalement.

onPanUpdate

Le pointeur est au contact de l'écran et bouge horizontalement ou verticalement.

onPanEnd

Le pointeur n'est plus en contact avec l'écran.

Remarque

Tous ces gestes peuvent être récupérés grâce à `GestureDetector` notamment, que nous étudierons en seconde partie.

Exercice : Quiz

[solution n°1 p.11]

Question 1

Quelle affirmation est vraie ?

- ☐ Un pointeur est issu d'un ou plusieurs gestures
- ☐ Un gesture est issu d'un ou plusieurs événements de pointeurs
- ☐ Un gesture est issu d'un unique événement de pointeur

Question 2

Si je touche l'écran et relâche aussitôt, qu'ai-je déclenché ? Plusieurs réponses possibles.

- ☐ `PointerMoveEvent`
- ☐ `PointerDownEvent`
- ☐ `PointerUpEvent`
- ☐ `onTapDown`
- ☐ `onTap`

Question 3

L'événement `OnVerticalDragStart` signifie :

- ☐ Que le pointeur a commencé à toucher l'écran et à bouger en direction verticale
- ☐ Que le pointeur a commencé à toucher l'écran et commence peut-être à bouger en direction verticale
- ☐ Que le pointeur a commencé à glisser verticalement

Question 4

Je peux définir `onPan` si l'événement `OnVerticalDragStart` est également défini.

- ☐ Vrai
- ☐ Faux

Question 5

Comment puis-je récupérer des événements directement sans passer par `GestureDetector` ?

- ☐ Grâce à un constructeur
- ☐ Grâce à un `Listener`
- ☐ Ce n'est pas possible

III. Utiliser les gestures

A. Gesture Detector

Maintenant que vous connaissez les différents types d'événements, il faut pouvoir les lier à vos widgets afin que ceux-ci réagissent en conséquence.

Pour récupérer un événement, nous pouvons dans presque tous les cas « envelopper » notre widget avec un `GestureDetector`.

En effet, `GestureDetector` est, comme son nom n'indique, un widget qui détecte les gestes ! Si celui-ci a un `Child`, il enveloppera celui-ci. S'il n'en a pas, il grandira pour s'adapter à son parent à la place.

Par exemple, ci-dessous, `GestureDetector` n'a pas de `Child`, et va donc prendre la taille du *container* qui l'englobe.

```
1 class _MyStatefulWidgetState extends State<MyStatefulWidget> {
2   Color _color = Colors.white;
3
4   @override
5   Widget build(BuildContext context) {
6     return Container(
7       color: _color,
8       height: 150.0,
9       width: 150.0,
10      child: GestureDetector(
11        onTap: () {
12          setState(() {
13            _color == Colors.red
14              ? _color = Colors.blue
15              : _color = Colors.red;
16          });
17        },
18      ),
19    );
20  }
21 }
```

Dans cet exemple, au moment où `GestureDetector` détectera un « *Tap* », il déclenche `setState`, ce qui aura pour effet de reconstruire le widget `Parent` avec la nouvelle couleur.

`GestureDetector` est capable de reconnaître un très grand nombre de gestes : des classiques que nous avons vus plus haut, jusqu'à l'action de dé-zoomer/zoomer (`onScaleStart`) !

Parfois, il peut être utile de voir quelle est la réelle taille d'action de `GestureDetector`, et pour du débogage, vous pouvez mettre la propriété `debugPaintPointersEnabled` en « *true* ».

Mais maintenant, peut-être vous demandez-vous comment sont gérés les gestes quand il y a plusieurs détecteurs de gestes au même endroit ?

Ces détecteurs vont tous se mettre à écouter les mouvements du pointeur pour reconnaître des gestes spécifiques. `GestureDetector` décide quel geste essayer de reconnaître selon si leurs *callbacks* ne sont pas nuls, c'est-à-dire si on a défini ce que l'on souhaitait faire en cas de geste particulier (par exemple : `onTap: () → setState()`).

Si plusieurs gestes peuvent *a priori* correspondre à plusieurs *callbacks*, chaque détecteur se retrouve alors dans la « *gestures arena* ».

Par exemple, au moment où l'utilisateur pose son doigt sur l'écran, le `GestureDetector` ne sait pas encore si celui-ci va faire un mouvement vertical ou horizontal. Les deux vont recevoir le `PointerDownEvent`. Ensuite, si finalement le mouvement se déplace de plus de pixels horizontalement que verticalement, le détecteur vertical va déclarer forfait au profit du détecteur horizontal.

B. Utiliser les built-in

Dans Flutter, bon nombre de widgets intègrent déjà des détecteurs de gestes. C'est le cas de `IconButton`, `TextButton` ou `ListView` par exemple.

```
1 class _MyStatefulWidgetState extends State<MyStatefulWidget> {
2   @override
3   Widget build(BuildContext context) {
4     IconButton(
5       icon: const Icon(Icons.volume_up),
6       onPressed: () {
7         setState(() {
8           _volume += 10;
9         });
10      },
11    );
12  }
13 }
```

```
11 ),}}
```

Ici, il aura été possible de rajouter un *callback* à l'événement `onPressed` sans avoir à englober le widget avec `GestureDetector` !

Quand vous souhaitez détecter un certain geste, n'hésitez pas à explorer la documentation afin de voir si votre widget n'intègre pas déjà la méthode.

Exercice : Quiz

[solution n°2 p.12]

Question 1

Je souhaiterais rajouter un détecteur de gestes à mon widget, que puis-je faire ?

- ☐ Englober mon widget dans un `GestureDetector`
- ☐ Mettre `GestureDetector` en `Child`
- ☐ Mettre `GestureDetector` en `Child`, et lui attribuer également un `Child`

Question 2

Si je souhaite voir clairement la zone couverte par mon détecteur de geste, que puis-je faire ?

- ☐ Mettre la propriété `debugPaintPointersEnabled` en « *true* »
- ☐ Mettre la propriété `debugPaintPointersEnabled` en « *false* »

Question 3

`GestureDetector` va essayer de détecter absolument tous les gestes dès lors que l'écran est touché.

- ☐ Vrai
- ☐ Faux

Question 4

Tous les widgets intègrent une détection des mouvements.

- ☐ Vrai
- ☐ Faux

Question 5

Quelle version est juste ?

- ☐

```
GestureDetector(  
  onTap: () {  
    _color == Colors.red  
    ? _color = Colors.blue  
    : _color = Colors.red;  
  },  
  child: Container(  
    color: _color,  
    height: 150.0,  
    width: 150.0,  
    child: GestureDetector,  
  ),
```

```
);
```

```
O Container(
  color: _color,
  height: 150.0,
  width: 150.0,
  child: GestureDetector(
    onTap: () {
      setState(() {
        _color == Colors.red
        ? _color = Colors.blue
        : _color = Colors.red;
      });
    },
  ),
);
```

V. Essentiel

1. Comprendre les gestes

- Les gestes sont des éléments essentiels pour interagir avec un programme.
- Les pointeurs sont des données brutes sur l'interaction d'un utilisateur avec l'écran.
- Ces pointeurs peuvent correspondre à 4 types d'événements (`PointerDownEvent`, `PointerMoveEvent`, `PointerUpEvent` et `PointerCancelEvent`).
- Flutter est capable d'interpréter plusieurs événements de pointeurs pour déterminer quel geste nous faisons exactement.
- Il y a 6 grandes catégories de gestes : `Tap`, `Double Tap`, `Horizontal Drag`, `VerticalDrag`, `Long Press` et `Pan`.

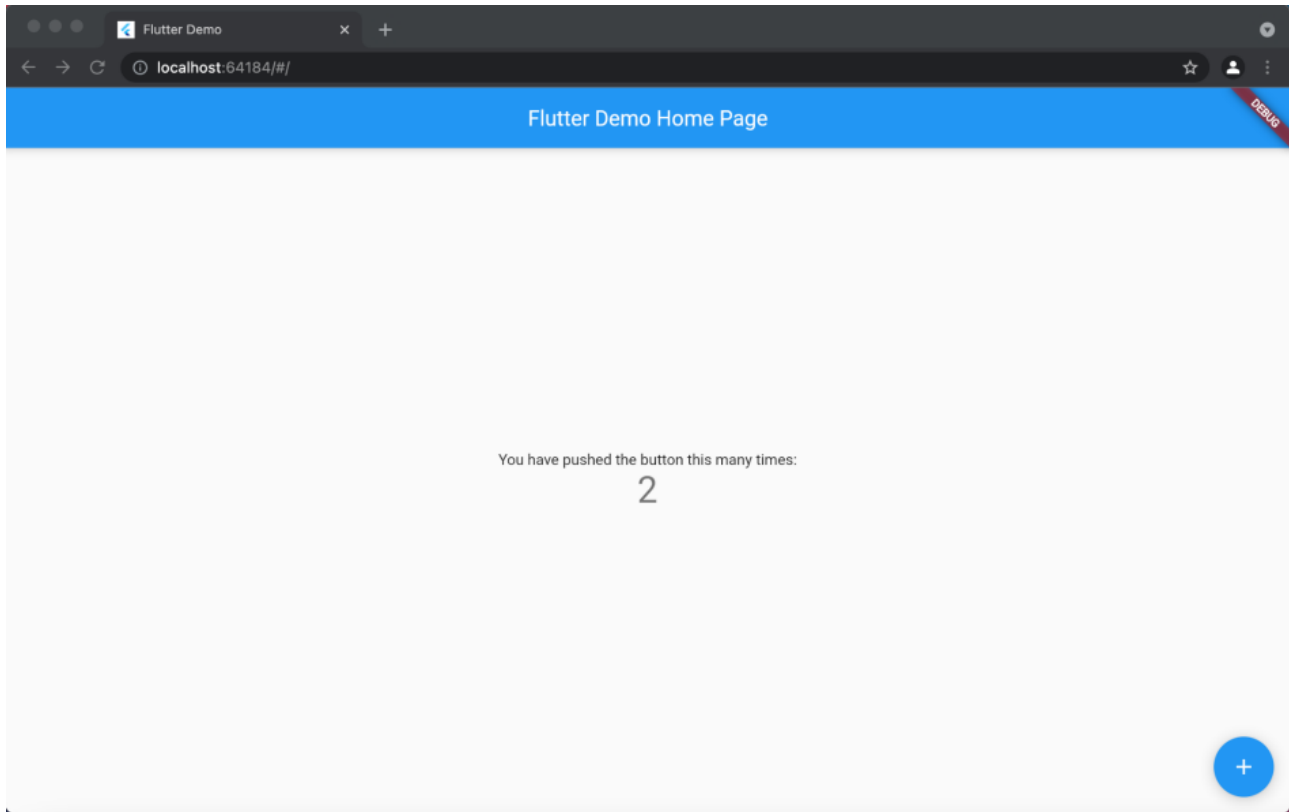
2. Utiliser les gestes

- Pour récupérer un événement, nous pouvons dans presque tous les cas « envelopper » notre widget avec un `GestureDetector`.
- Si le `GestureDetector` a un `child`, il couvrira toute la taille du `child`. En revanche, s'il n'a pas de `child`, il s'étendra pour s'adapter à son parent à la place.
- `GestureDetector` décide quels gestes essayer de reconnaître selon si leurs *callbacks* ne sont pas nuls (c'est-à-dire si on a défini ce que l'on souhaitait faire en cas de geste particulier, comme `onTap: () → setState()`).
- Si deux détecteurs reconnaissent le début d'un geste qui pourrait conduire à leur *callback*, ceux-ci entrent dans la « *gesture arena* ». Quand un geste correspond davantage à un détecteur que l'autre, il y a un gagnant et son *callback* s'exécute.

VI. Auto-évaluation

A. Exercice

Les gestes sont essentiels pour l'échange d'interactions entre l'utilisateur et l'écran. Si bien que, lorsque vous créez un projet Flutter, l'application par défaut consiste en un simple compteur vous permettant de voir que oui, votre premier projet fonctionne bien !



Et si nous reprenions cette app ?

Dans cet exercice, nous allons reprendre l'application fournie au moment de la création d'un projet Flutter et essayer de recréer les mêmes actions grâce à un `GestureDetector`.

Question

[solution n°3 p.13]

1. Créer un projet Flutter web Flutter¹.
2. Ouvrir l'app fournie par défaut grâce au navigateur.
3. Utiliser un `GestureDetector` qui englobe le bouton au lieu de la solution *built-in* pour réaliser la même chose.
4. Supprimer le `floating action button` et créer un `container` de 50 px par 150 px ayant un `GestureDetector` en `child`. Le placer dans la colonne. Ce bouton devra faire la même chose que le précédent bouton.

B. Test

Exercice 1 : Quiz

[solution n°4 p.14]

Question 1

¹ <https://flutter.dev/docs/get-started/web>

Ces pointeurs peuvent correspondre à ces types d'événements : `PointerDownEvent`, `PointerMoveEvent`, `PointerUpEvent` et `PointerCancelEvent`.

- ☐ Vrai
- ☐ Faux

Question 2

Il est tout aussi bien d'utiliser un `GestureDetector` qu'une solution intégrée.

- ☐ Vrai
- ☐ Faux

Question 3

`GestureDetector` va essayer de détecter absolument tous les gestes dès lors que l'écran est touché.

- ☐ Vrai
- ☐ Faux

Question 4

Il vaut mieux utiliser un widget avec un détecteur intégré qu'en créer un de toutes pièces.

- ☐ Vrai
- ☐ Faux
- ☐ Ça dépend

Question 5


Parmi ces gestes, lesquels vont entraîner un événement `Tap` ?

- ☐ `onTapDown`
- ☐ `onTapUp`
- ☐ `onTapCancel`

Solutions des exercices

Exercice p. 4 Solution n°1**Question 1**


Quelle affirmation est vraie ?

- ☐ Un pointeur est issu d'un ou plusieurs gestures
- ☒ Un gesture est issu d'un ou plusieurs événements de pointeurs
- ☐ Un gesture est issu d'un unique événement de pointeur
-  Les événements de pointeurs sont des événements à « bas niveau ». Les gestures sont issus de l'interprétation d'un ou plusieurs événements de pointeurs.

Question 2


Si je touche l'écran et relâche aussitôt, qu'ai-je déclenché ? Plusieurs réponses possibles.

- ☐ PointerMoveEvent
- ☒ PointerDownEvent
- ☒ PointerUpEvent
- ☒ onTapDown
- ☒ onTap

 Le fait de taper très rapidement l'écran va avoir créé deux événements de pointeurs : `PointerDownEvent` et `PointerUpEvent`. Ceux-ci seront interprétés comme des gestures `onTapDown` puis `onTap`.

Question 3


L'événement `OnVerticalDragStart` signifie :

- ☐ Que le pointeur a commencé à toucher l'écran et à bouger en direction verticale
- ☒ Que le pointeur a commencé à toucher l'écran et commence peut-être à bouger en direction verticale
- ☐ Que le pointeur a commencé à glisser verticalement
-  Au moment où le pointeur touche l'écran, le framework ne sait pas encore s'il va se déplacer horizontalement ou verticalement. Il doit donc être attentif aux mouvements vers les deux directions.

Question 4


Je peux définir `onPan` si l'événement `OnVerticalDragStart` est également défini.

- ☐ Vrai
- ☒ Faux

 L'utilisation de `onPan` et de `OnVerticalDragStart` peut causer un crash. En effet, les deux vont chercher les mêmes informations (le glissement du pointeur vers une direction), ce qui peut créer un conflit.

Question 5


Comment puis-je récupérer des événements directement sans passer par `GestureDetector` ?

- ☐ Grâce à un constructeur
- ☒ Grâce à un `Listener`
- ☐ Ce n'est pas possible
-  La plupart du temps, nous utilisons `GestureDetector` ou des détecteurs de mouvements déjà intégrés à des widgets (comme `Button`), mais il est aussi tout à fait possible de rester attentif à ces mouvements grâce à un `Listener`.

Exercice p. 7 Solution n°2


Question 1

Je souhaiterais rajouter un détecteur de gestes à mon widget, que puis-je faire ?

- ☒ Englober mon widget dans un `GestureDetector`
- ☒ Mettre `GestureDetector` en `Child`
- ☐ Mettre `GestureDetector` en `Child`, et lui attribuer également un `Child`
-  Englober votre widget dans un `GestureDetector` OU rajouter un `GestureDetector` en `child` (mais lui-même sans `child`) aura pour même effet de couvrir la surface de votre widget. Cependant, dès que je mets un `child` à `GestureDetector`, la zone couverte sera celle de son `child` et non plus de son parent.


Question 2

Si je souhaite voir clairement la zone couverte par mon détecteur de geste, que puis-je faire ?

- ☒ Mettre la propriété `debugPaintPointersEnabled` en « *true* »
- ☐ Mettre la propriété `debugPaintPointersEnabled` en « *false* »
-  Il est toujours compliqué de réussir à déterminer réellement la surface couverte par mon détecteur. C'est pourquoi, pour pouvoir voir la zone couverte par mon détecteur, il faut mettre la propriété en « *true* ».


Question 3

`GestureDetector` va essayer de détecter absolument tous les gestes dès lors que l'écran est touché.

- ☐ Vrai
- ☒ Faux
-  `GestureDetector` va essayer de détecter uniquement les gestes dont le *callback* ne sera pas nul. Si vous n'avez défini que `onTap`, il ne cherchera à détecter que ces événements-là.

Question 4

Tous les widgets intègrent une détection des mouvements.

- ☐ Vrai
- ☒ Faux
-  Même s'il est vrai que nous pouvons rajouter la détection de mouvements à presque tous les widgets grâce à `GestureDetector`, tous n'intègrent pas cette méthode à l'origine. Ceux qui l'intègrent sont la plupart du temps des widgets issus du Material Design, comme `IconButton`.

Question 5

Quelle version est juste ?

```


○ GestureDetector(
  onTap: () {
    _color == Colors.red
    ? _color = Colors.blue
    : _color = Colors.red;
  },
  child: Container(
    color: _color,
    height: 150.0,
    width: 150.0,
    child: GestureDetector,
  ),
);

```

```

⦿ Container(
  color: _color,
  height: 150.0,
  width: 150.0,
  child: GestureDetector(
    onTap: () {
      setState(() {
        _color == Colors.red
        ? _color = Colors.blue
        : _color = Colors.red;
      });
    },
  ),
);

```

 Dans la première version, il manque `SetState()` ! Le `Gesturedetector` fonctionnera bien et englobera le `container`, mais il ne se passera pas grand-chose.

p.9 Solution n°3

Première réponse :

Il faut modifier le `FloatingActionButton`.

```

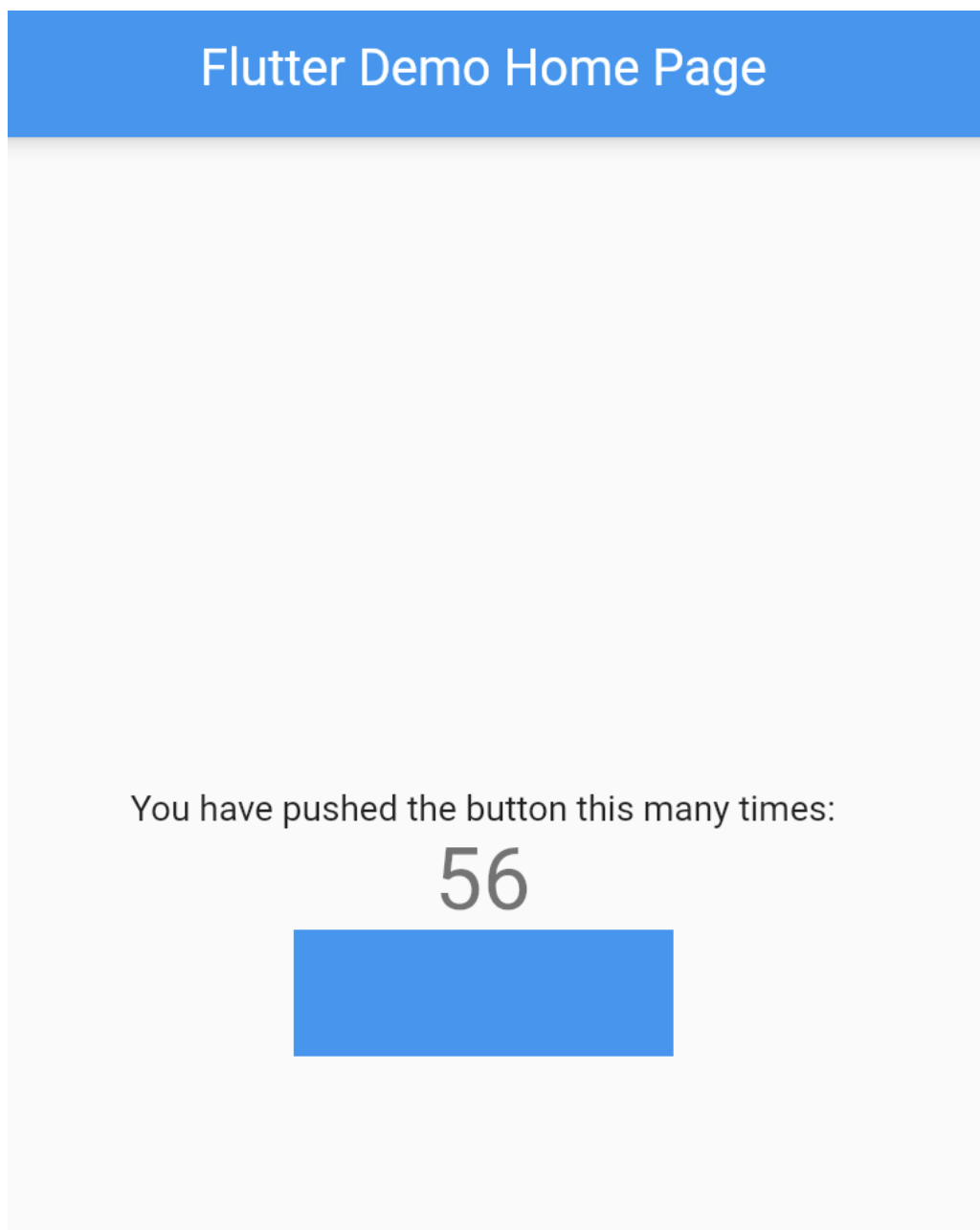
1 FloatingActionButton: GestureDetector(
2     onTap: _incrementCounter,
3     child: FloatingActionButton(
4         tooltip: 'Increment',
5         child: Icon(Icons.add),
6     ), // This trailing comma makes auto-formatting nicer for build methods.
7 );

```

Dans les faits, il ne faut jamais mettre un `GestureDetector` quand on a une solution *built-in* ! Mais ceci vous permet d'expérimenter toutes les possibilités.

Deuxième réponse :

```
1 Container(  
2     height: 50,  
3     width: 150,  
4     color: Colors.blue,  
5     child: GestureDetector(  
6         onTap: _incrementCounter,  
7     ))
```




Exercice p. 9 Solution n°4

Question 1

Ces pointeurs peuvent correspondre à ces types d'événements : `PointerDownEvent`, `PointerMoveEvent`, `PointerUpEvent` et `PointerCancelEvent`.

☒ Vrai

☐ Faux


 Vrai, ces pointeurs peuvent correspondre aux quatre types d'événements suivants : `PointerDownEvent`, `PointerMoveEvent`, `PointerUpEvent` et `PointerCancelEvent`.

Question 2

Il est tout aussi bien d'utiliser un `GestureDetector` qu'une solution intégrée.

☐ Vrai

☒ Faux


 Si une solution intégrée existe, utilisez-la ! Cela permet d'avoir moins de lignes de code et donc plus de lisibilité.

Question 3

`GestureDetector` va essayer de détecter absolument tous les gestes dès lors que l'écran est touché.

☐ Vrai

☒ Faux

 `GestureDetector` va essayer de détecter uniquement les gestes dont le *callback* ne sera pas nul. Si vous n'avez défini que `onTap`, il ne cherchera à détecter que ces événements-là.


Question 4

Il vaut mieux utiliser un widget avec un détecteur intégré qu'en créer un de toutes pièces.

☐ Vrai

☐ Faux

☒ Ça dépend

 Tout dépend de votre projet ! Si celui-ci correspond à un `IconButton`, il est bien sûr préférable de l'utiliser directement. Cependant, vous aurez tout intérêt à en créer un de toutes pièces s'il est loin du Material Design.


Question 5

Parmi ces gestes, lesquels vont entraîner un événement `Tap` ?

☒ `onTapDown`

☒ `onTapUp`

☐ `onTapCancel`

 Les gestes `onTapDown` et `onTapUp` vont entraîner un événement `Tap`. En revanche, le geste `onTapCancel` ne déclenchera pas d'événement `Tap`.