**Which widget is used to implement a Material Design AppBar in Flutter?**

A) TopBar

B) MaterialBar

C) Header

D) AppBar

**Justification** :

The AppBar widget is used to create a Material Design-style app bar at the top of the screen. The AppBar provides a consistent and customizable way to include navigation, branding, and actions in your Flutter app

**Example**:

Widget build(BuildContext context) {

    return MaterialApp(

      home: Scaffold(

        appBar: AppBar(

          title: Text('Material Design AppBar'),

        ),

        body: Center(

          child: Text('Hello, Flutter!'),

        ),

      ),

    );

  }

**Which widget would allow for horizontal scrolling of its children?**

A) ListView.horizontal

B) Column

C) RowScroll

D) HListView

**Justification** : This widget allows you to create a scrollable list of children arranged horizontally, making it easy to scroll through items from left to right.

**Example** :

  ListView(

          scrollDirection:

              Axis.horizontal

          children: <Widget>[

            Container(

              width: 150.0,

              color: Colors.red,

              child: Center(child: Text('Item 1')),

            ),

            Container(

              width: 150.0,

              color: Colors.blue,

              child: Center(child: Text('Item 2')),

            ),

            Container(

              width: 150.0,

              color: Colors.green,

              child: Center(child: Text('Item 3')),

            ),

          ],

        ),

**What is the purpose of the Scaffold widget in Flutter?**

A) Displaying floating buttons

B) Managing state

C) Providing a visual structure for app layout

D) Creating animations

**Justification**:

The Scaffold widget in Flutter is to provide a visual structure for app layout. It serves as the basic framework for organizing the visual elements of a Flutter app's user interface.

**Which of the following is not a type of layout constraint in Flutter?**

A) BoxConstraints

B) AxisConstraints

C) FlexFit

D) Constraints.tight

**Justification**: "AxisConstraints" is not a standard term or type of constraint in Flutter's layout system. While the word "axis" is often used to describe the direction of widgets (horizontal or vertical), it is not used as a specific constraint type

**How would you ensure a widget only takes up the space necessary for its content?**

A) mainAxisSize: MainAxisSize.compact

B) fit: BoxFit.contain

C) mainAxisSize: MainAxisSize.min

D) fit: BoxFit.fitWidth

**Justification**: Setting the **mainAxisSize** property to **MainAxisSize.min** for a widget ensures that it takes up the minimum space required to accommodate its content. This property is commonly used with widgets like **Row** and **Column** to make them as compact as possible, based on their children's sizes.

**Which widget would you use to evenly distribute space between its children along the main axis?**

A) Stack

B) SpaceDistributor

C) Spacer

D) MainAxisAlignment.spaceEvenly

**Justification**: **MainAxisAlignment.spaceEvenly** is used to evenly distribute the available free space along the main axis between the children of a **Row** or **Column** widget. It ensures that the space is distributed equally not only between the children but also before the first child and after the last child  
  
**Example**:

  Column(

            mainAxisAlignment: MainAxisAlignment.spaceEvenly,

            children: [

              Container(

                width: 50,

                height: 50,

                color: Colors.blue,

              ),

              Container(

                width: 50,

                height: 50,

                color: Colors.green,

              ),

              Container(

                width: 50,

                height: 50,

                color: Colors.red, ),],),

**Which widget is commonly used to create a tabbed interface in Flutter?**

A) Tabs

B) TabView

C) TabBarView

D) TabNavigator

**Justification**: To create a tabbed interface in Flutter, you commonly use the **TabBar** and **TabBarView** widgets in combination.

Example:

Scaffold(

          appBar: AppBar(

            title: Text('Tabbed Interface Example'),

            bottom: TabBar(

              tabs: [

                Tab(text: 'Tab 1'),

                Tab(text: 'Tab 2'),

                Tab(text: 'Tab 3'),

              ],

            ),

          ),

          body: TabBarView(

            children: [

              Center(child: Text('Tab 1 Content')),

              Center(child: Text('Tab 2 Content')),

              Center(child: Text('Tab 3 Content')),

            ],

          ),

        ),

**For responsive design in Flutter, which widget helps determine the runtime environment?**

A) EnvironmentDetector

B) ResponsiveBuilder

C) MediaQuery

D) DeviceDetector

**Justification** : The MediaQuery widget in Flutter is a crucial tool for building responsive designs. It provides information about the current runtime environment, including screen size, orientation, and more.

**Example**:

 Widget build(BuildContext context) {

    final mediaQuery = MediaQuery.of(context);

    return MaterialApp(

      home: Scaffold(

        appBar: AppBar(

          title: Text('Responsive Design Example'),

        ),

        body: Center(

          child: Text(

            'Welcome to Flutter!',

            style: TextStyle(

              fontSize: mediaQuery.size.width < 600

                  ? 20

                  : 40,

            ),

          ),

        ),

      ),

    );

  }

**Which widget can be used to overlap several children widgets?**

A) OverlapBox

B) Stack

C) Pile

D) Overlay

**Justification**: The **Stack** widget allows you to place multiple children on top of each other, creating an overlapping effect. It's often used when you need to layer widgets in a specific order, such as having one widget on top of another.  
  
Example:

Center(

          child: Stack(

            children: <Widget>[

              Container(

                width: 200,

                height: 200,

                color: Colors.blue,

              ),

              Positioned(

                top: 50,

                left: 50,

                child: Container(

                  width: 100,

                  height: 100,

                  color: Colors.red,

                ),

              ),

            ],

          ),

        ),

**Which widget provides a drawer typically from the side of the screen in Flutter?**

A) SidePanel

B) SlidingDrawer

C) Drawer

D) SlideMenu

**Justification**: The **Drawer** widget is the standard and widely accepted way to implement a side drawer in Flutter applications

**Example**:

drawer: Drawer(

          child: ListView(

            padding: EdgeInsets.zero,

            children: <Widget>[

              DrawerHeader(

                decoration: BoxDecoration(

                  color: Colors.blue,

                ),

                child: Text(

                  'Drawer Header',

                  style: TextStyle(

                    color: Colors.white,

                    fontSize: 24,

                  ),

                ),

              ),

              ListTile(

                leading: Icon(Icons.home),

                title: Text('Home'),

                onTap: () {

                  Navigator.pop(context);

                },

              ),

              ListTile(

                leading: Icon(Icons.settings),

                title: Text('Settings'),

                onTap: () {

                  Navigator.pop(context);

                },

              ),

            ],

          ),

        ),

**Which widget is commonly used for user input in the form of selectable date in Flutter?**

A) DatePicker

B) DateSelector

C) CalendarPicker

D) DateInput

**Justification**: This widget provides a date picker interface that allows users to select a date easily. It's a standard and widely used widget in Flutter for date selection

**Example**:

 DateTime selectedDate = DateTime.now();

  Future<void> \_selectDate(BuildContext context) async {

    final DateTime? picked = await showDatePicker(

      context: context,

      initialDate: selectedDate,

      firstDate: DateTime(2000),

      lastDate: DateTime(2101),

    );

    if (picked != null && picked != selectedDate)

      setState(() {

        selectedDate = picked;

      });

  }

**Which layout widget wraps its children and arranges them sequentially in a flow?**

A) Flow

B) Wrap

C) Sequence

D) Aligner

**Justification**: The "Flow" layout widget in Flutter wraps its children and arranges them sequentially in a flow. It allows children widgets to be positioned one after another horizontally or vertically based on the direction specified.  
**Example** :

|  |  |
| --- | --- |
| Flow(        delegate: MyFlowDelegate(),        children: <Widget>[          if (context != null)            Container(              width: 100,              height: 100,              color: Colors.red,            ),          if (context != null)            Container(              width: 100,              height: 100,              color: Colors.green,            ),          if (context != null)            Container(              width: 100,              height: 100,              color: Colors.blue,            ),        ],      ); | @override    void paintChildren(FlowPaintingContext context) {      var x = 0.0;      var y = 0.0;      for (int i = 0; i < context.childCount; i++) {        var childSize = context.getChildSize(i);        if (childSize != null) {          context.paintChild(            i,            transform: Matrix4.translationValues(x, y, 0.0),          );          x += childSize.width + 30;       }      }    }    @override    bool shouldRepaint(covariant FlowDelegate oldDelegate) {      return false;    }}} |

**For spacing out multiple buttons in a row evenly, which widget would be best suited?**

A) ButtonBar

B) ButtonRow

C) SpacerRow

D) AlignButton

**justification**: Places the buttons horizontally according to the [buttonPadding](https://api.flutter.dev/flutter/material/ButtonBar/buttonPadding.html). The children are laid out in a [Row](https://api.flutter.dev/flutter/widgets/Row-class.html) with [MainAxisAlignment.end](https://api.flutter.dev/flutter/rendering/MainAxisAlignment.html).

**Example**:

 ButtonBar(

            alignment: MainAxisAlignment.spaceEvenly,

            children: <Widget>[

              ElevatedButton(

                onPressed: () {

                },

                child: Text('Button 1'),

              ),

              ElevatedButton(

                onPressed: () {

                },

                child: Text('Button 2'),

              ),

              ElevatedButton(

                onPressed: () {

                },

                child: Text('Button 3'),

              ),

            ],

          ),

**To position a widget relative to the top right corner of its parent, which property of Positioned widget would you primarily adjust?**

A) top and left

B) bottom and left

C) top and right

D) bottom and right

**Justification**: When you want to position a widget relative to the top right corner of its parent using the **Positioned** widget, you need to adjust the **top** property to specify the distance from the top edge of the parent and the **right** property to specify the distance from the right edge of the parent.

**Example**:

Stack(

          children: [

            Container(

              color: Colors.blue,

              width: double.infinity,

              height: 500,

            ),

            Positioned(

              top:

                  10

              left:

                  10,

              child: Container(

                width: 50,

                height: 50,

                color: Colors.red,

              ),

            ),

          ],

        ),

**To achieve a blurred visual effect in Flutter, you would use:**

A) BlurredBox

B) Opacity

C) BackdropFilter

D) BlurEffect

**Justification** :

**Example** :

Center(child: Container(

            width: 200,

            height: 200,

            decoration: BoxDecoration(

              image: DecorationImage(

                image: AssetImage(

                    'assest/images.jpeg'),

                fit: BoxFit.cover,

              ),),child: BackdropFilter(

              filter: ImageFilter.blur(

                  sigmaX: 5, sigmaY: 5),

              child: Container(

                color: Colors.black

                    .withOpacity(0.1),

                child: Center(

                  child: Text(

                    'Blurred Text',

                    style: TextStyle(

                      fontSize: 24,

                      color: Colors.white,

                    ),),),),),),),

**Which widget is used to hide or show a child based on a boolean value?**

A) Visibility

B) ShowHide

C) Toggler

D) DisplaySwitch

**Justification** : In this Flutter example, we use the Visibility widget to show or hide a child widget (Text) based on the value of the isChildVisible boolean variable.

**Example** :

Visibility(

                visible:isChildVisible,

                child: Text(

                  'This is a visible child.',

                  style: TextStyle(fontSize: 18),

                ),

              ),

**To display a list of dynamic length where items are created on demand, you'd use:**

A) DynamicList

B) ListView.builder

C) OnDemandList

D) ItemGenerator

**Justification :** It is used to create the list of children But when we want to create a list recursively without writing code again and again then ListView.builder is used instead of ListView.  ListView.builder creates a scrollable, linear array of widgets**.**

**Example :**

ListView.builder(

          itemCount: dynamicList.length,

          itemBuilder: (BuildContext context, int index) {

            return ListTile(

              title: Text(dynamicList[index]),

            );

          },

        ),

**Which widget allows for pinching to zoom content in Flutter?**

A) Zoomable

B) PinchZoom

C) InteractiveViewer

D) ScaleDetector

**Justification :** In this Flutter example, we use the InteractiveViewer widget to enable pinch-to-zoom functionality for an image. You can replace the image URL with your own image. The boundaryMargin property sets the margin around the zoomable area, and minScale and maxScale define the minimum and maximum zoom levels. Users can pinch to zoom in and out on the image within the specified boundaries. The InteractiveViewer widget provides an intuitive way to implement pinch-to-zoom functionality for various content, such as images, maps, or custom widgets.

**Example** :

InteractiveViewer(

            child: Image.network(

              'https://cdn.pixabay.com/photo/2023/08/19/13/56/flower- 8200543\_1280.jpg',

              fit: BoxFit.contain,

            ),

            boundaryMargin: EdgeInsets.all(20.0),

            minScale: 0.1,

            maxScale: 2.0,

          ),

**To change the orientation of text in Flutter, which widget would you use?**

A) Rotate

B) TextOrientation

C) TurnText

D) Transform.rotate

**Justification** : The Transform.rotate widget allows you to apply transformations like rotation, scaling, and translation to child widgets, making it suitable for changing the orientation of text or other elements in your Flutter app.

**Example** :

Transform.rotate(

            angle: 1.5708

            child: Text(

              'Rotated Text',

              style: TextStyle(fontSize: 24),

            ),

          ),

**For keeping a widget on top of the keyboard when it appears, which widget would you use?**

A) TopKeeper

B) KeyboardAvoider

C) StayOnTop

D) KeyboardAvoidingView

**Justification:**

**Example:**

**Which widget is used for a basic linear layout in Flutter?**

A) LinearLayout

B) FlexLayout

C) Column

D) SingleLine

**Justification:** In Flutter, the Column widget is used for creating a basic linear layout. It arranges its children vertically, one below the other, in a linear fashion. This makes it suitable for building user interfaces where you need to stack widgets vertically, such as lists of items or forms.

**Example:**

Column(

          mainAxisAlignment: MainAxisAlignment.center,

          crossAxisAlignment: CrossAxisAlignment.end,

          children: <Widget>[

            Text('Widget 1'),

            Text('Widget 2'),

            Text('Widget 3'),

          ],

        ),