itrenzheng

练习题

质量更高　　服务更好

https://www.itrenzheng.com

一年免费更新服务

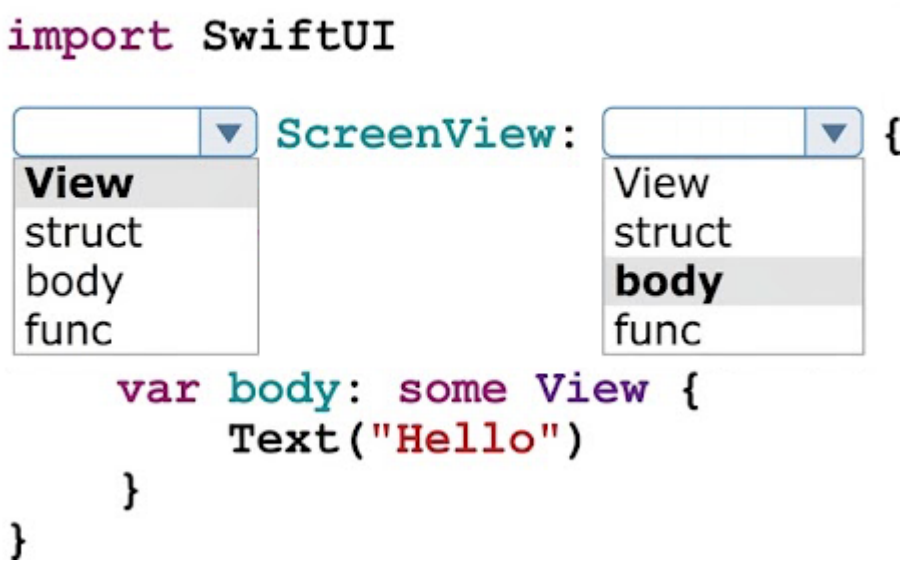
Exam : **App Development with Swift Certified User**

Title : App Development with Swift Certified User Exam

Version : DEMO

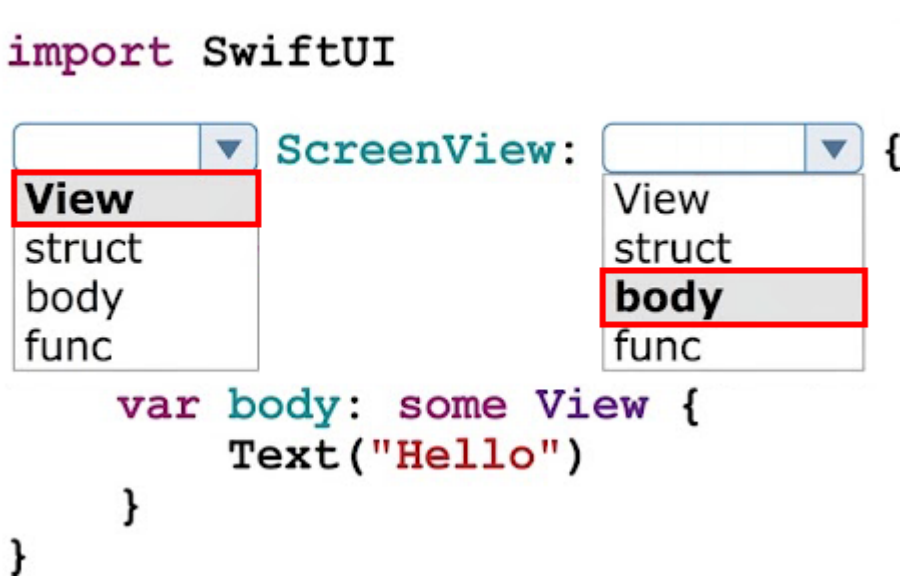
1.HOTSPOT
Complete the code that conforms to the View protocol by selecting the correct option from each drop-down list. Note: You will receive partial credit for each correct answer.

Answer Area



Answer:

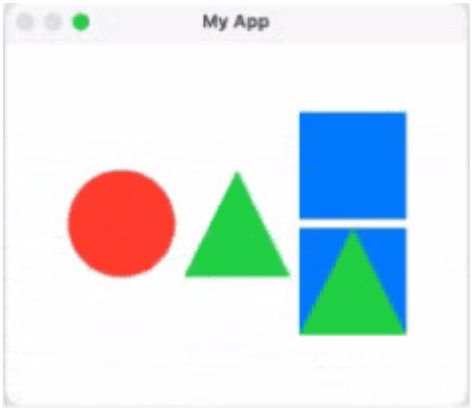
Answer Area



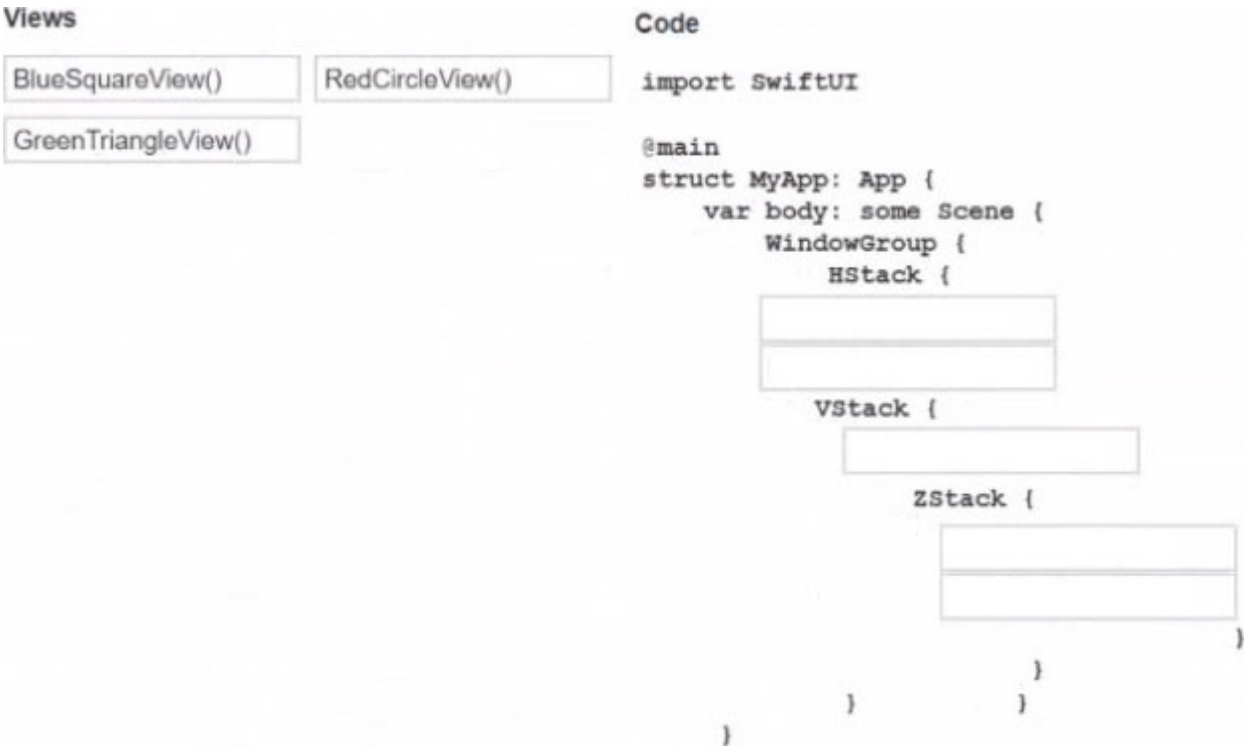
Explanation:
This question belongs to View Building with SwiftUI, especially the domain covering positioning and/or layout a single SwiftUI View with standard Views and modifiers and the foundational structure of a SwiftUI view. In SwiftUI, a custom screen is typically declared as a struct that conforms to the View protocol.
Apple's SwiftUI documentation shows the standard pattern:
struct ScreenView: View {
var body: some View {
Text("Hello")
}

}
Here, struct is required because SwiftUI views are commonly defined as structures. View is required after the colon because the type must conform to the View protocol. body is the required computed property that returns the content of the view as some View. Apple documents that every conforming View type must provide a body property that describes its content.
So the completed code is:
import SwiftUI
struct ScreenView: View {
var body: some View {
Text("Hello")
}
}
This is the canonical SwiftUI view declaration pattern and is one of the most fundamental concepts in App Development with Swift.

2.DRAG DROP
Drag the views on the left to the correct locations m the code on the fight to match the shown canvas.



You may use each View once, more than once, or not at all.



Answer:

Explanation:

RedCircleView()

GreenTriangleView() BlueSquareView() BlueSquareView() GreenTriangleView()

This question belongs to View Building with SwiftUI, specifically arranging views with HStack, VStack, and ZStack. In SwiftUI, an HStack lays views out horizontally, a VStack lays them out vertically, and a ZStack overlays views front-to-back. Apple's stack layout guidance describes these three containers exactly this way.

To match the canvas, the main HStack must show three items from left to right: a red circle, a green triangle, and then a right-side vertical group. That means the first two blanks inside HStack are RedCircleView() and GreenTriangleView(). On the right side, the VStack shows a blue square on top, so the next blank is BlueSquareView(). Under that, the lower-right shape is made by layering a green triangle on top of a blue square, which means the ZStack must contain BlueSquareView() first as the background and GreenTriangleView() second as the foreground. SwiftUI's documentation notes that ZStack aligns and overlays its children in depth order, which is why the square goes before the triangle. So the correct placement order is:
HStack {
RedCircleView()
GreenTriangleView()
VStack {
BlueSquareView()
ZStack {
BlueSquareView()
GreenTriangleView()
}
}
}
That arrangement reproduces the exact layout shown in the canvas.

3.HOTSPOT
When you press 'Show Button' on your app. a modal View appears.
Complete the code by selecting the correct option from each drop-down list. Note: You will receive partial credit for each correct answer.

Answer Area

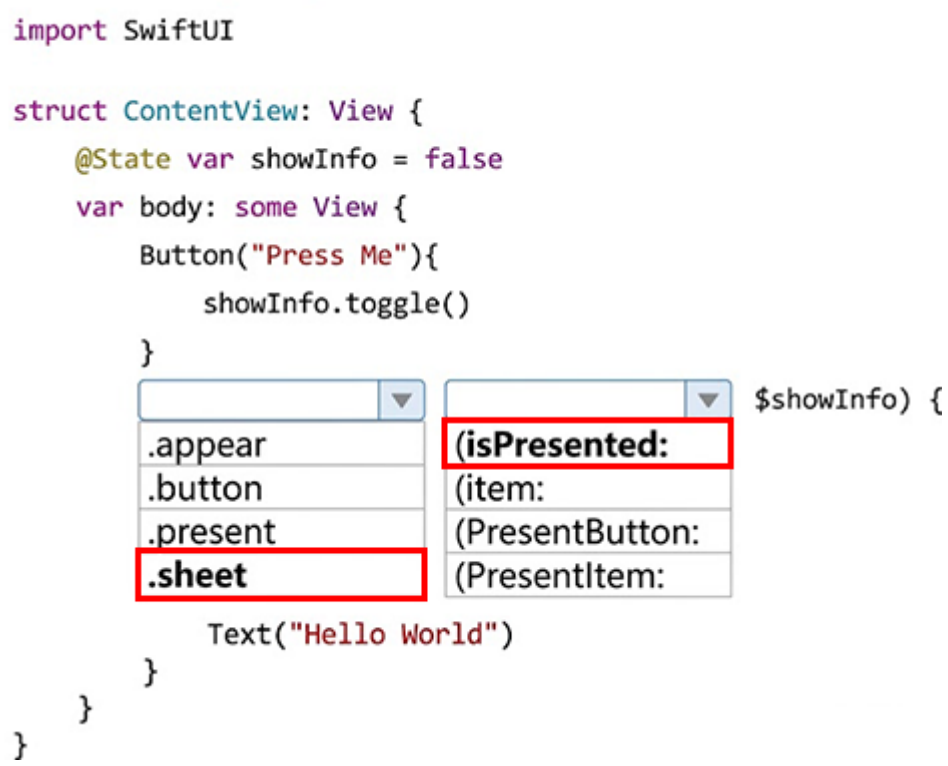
```
import SwiftUI

struct ContentView: View {
    @State var showInfo = false
    var body: some View {
        Button("Press Me"){
            showInfo.toggle()
        }
        [ ▼ ] [ ▼ ] $showInfo) {
```

.appear	(**isPresented:**
.button	(item:
.present	(PresentButton:
.sheet	(PresentItem:

```
            Text("Hello World")
        }
    }
}
```

Answer:

Answer Area



Explanation:

This question belongs to View Building with SwiftUI, specifically the domain on creating a multi-view app with navigation stacks, links, and sheets.

To present a modal view in SwiftUI when a Boolean state changes, the correct modifier is.sheet.

The matching sheet API for a Boolean binding is:

sheet(isPresented: $showInfo) {

//modal content

}

So the first blank must be .sheet, and the second blank must be (isPresented:.

The logic works like this:

@State stores the local Boolean that controls presentation.

Pressing the button calls showInfo.toggle(), changing the value from false to true. When that Boolean becomes true, the .sheet(isPresented:) modifier presents the modal view.

When the modal is dismissed, SwiftUI updates the Boolean back as needed. There is also a typing issue in the screenshot: the state variable appears as ShowInfo, while the button and binding use showInfo. Swift is case-sensitive, so those names must match.

The corrected code should use the same identifier consistently, such as:

@State var showInfo = false

Therefore, the correct dropdown selections are:

sheet

(isPresented:

4.HOTSPOT

Complete the code by selecting the correct option from each drop-down list to create the following screen. Note: You will receive partial credit for each correct answer.

Answer Area

```
import SwiftUI

struct ContetView: View {
    var body: some View {
        VStack {
            Text("Press To Play")
            [ ▼ ] (action: {}, label: {
                Text
                Label
                Button
            [ ▼ ] ("Ready, Set, Go!")
                Label
                String
                Font
                Text
            [ ▼ ] (.largeTitle)
                .font
                .text
                .label
            .foregroundStyle(.blue)
            [ ▼ ] (15)
                .space
                .padding
            .border(.blue, width: 5)
            [ ▼ ] (.yellow)
                .color
                .backgroundColor
                background
                .textcolor
            })
        }
    }
}
```

Answer:

Answer Area

```
import SwiftUI

struct ContetView: View {
    var body: some View {
        VStack {
            Text("Press To Play")
                [ ▼ ] (action: {}, label: {
                    Text
                    Label
                    Button
                [ ▼ ] ("Ready, Set, Go!")
                    Label
                    String
                    Font
                    Text
                [ ▼ ] (.largeTitle)
                    .font
                    .text
                    .label
                .foregroundStyle(.blue)
                [ ▼ ] (15)
                    .space
                    .padding
                .border(.blue, width: 5)
                [ ▼ ] (.yellow)
                    .color
                    .backgroundColor
                    background
                    .textcolor
                })
        }
    }
}
```

5.Review the code snippet.

```
1  struct NewAppleProduct {
2      var name: String
3      var releaseDate: String?
4  }
5  var lastReleaseDate = "2020-04-17"
6  let nextApplePhone = NewAppleProduct(name: "iPhone 12", releaseDate: nil)
7  lastReleaseDate = nextApplePhone.releaseDate != nil ? _____
```

Which statement completes the code snippet so that:

• The lastReleaseDate remains the same when nextApplePhone.releaseDate is nil.

• The lastReleaseDate updates to the nextApplePhone.releaseDate when nextApplePhone.releaseDate is NOT nil.

A. nextApplePhone.releaseDate : lastReleaseDate

B. lastReleaseDate : nextApplePhone.releaseDate

C. lastReleaseDate : nextApplcPhone.rcleaseDate!

D. nextApplePhone.releaseDate! : lastReleaseDate

Answer: D

Explanation:

This question is from Swift Programming Language, especially the domains for Optional types, safe and unsafe unwrapping, and control flow.

The code uses the ternary conditional operator:

nextApplePhone.releaseDate != nil ? _____

In Swift, the ternary operator follows this structure:

condition ? valueIfTrue : valueIfFalse

So if nextApplePhone.releaseDate != nil is true, the expression must return the new release date. If it is false, it must keep lastReleaseDate unchanged.

That means the missing part must be:

nextApplePhone.releaseDate! : lastReleaseDate which is option D.

This works because nextApplePhone.releaseDate is declared as String?, so it is an optional. Once the condition confirms it is not nil, the code force-unwraps it with ! to access the underlying String value. If the optional is nil, the expression returns lastReleaseDate instead. Apple’s Swift documentation describes the ternary conditional operator as a shortcut for choosing one of two expressions based on a condition, and it explains that force unwrapping with ! accesses an optional’s wrapped value when you know it is not nil. The other options are incorrect because they reverse the true/false logic, omit the needed unwrap, or contain invalid identifiers.

Therefore, the correct completion is D.