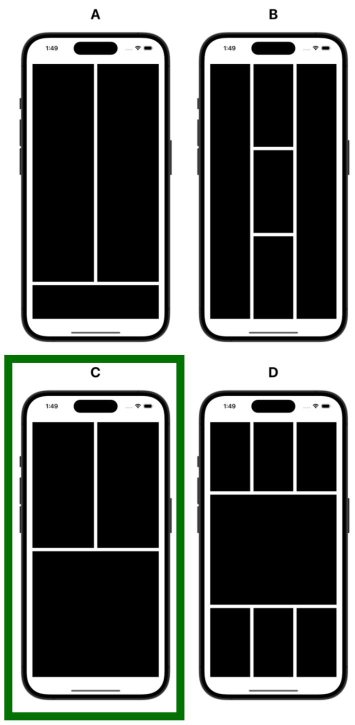
Quiz: User Interface Fundamentals

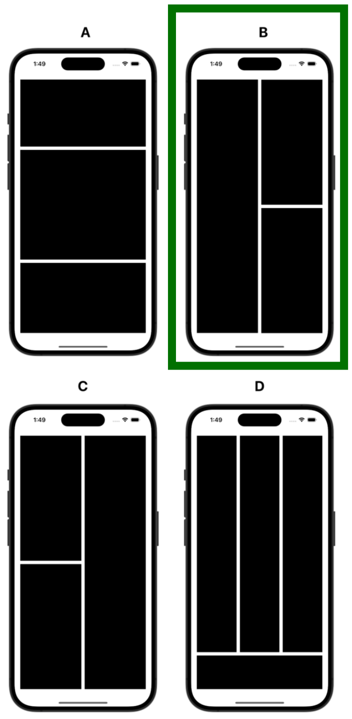
1. The following code will produce which layout shown at right? (circle one)

```
struct ContentView: View {
    var body: some View {
        VStack {
            HStack {
                Rectangle()
                Rectangle()
            }
            Rectangle()
        }
    }
}
```



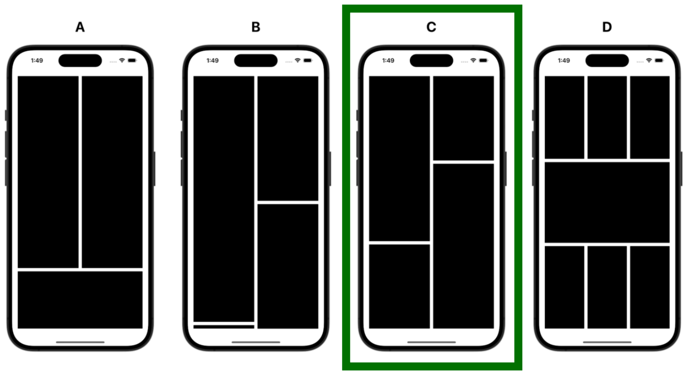
2. The following code will produce which layout shown at right? (circle one)

```
struct ContentView: View {
    var body: some View {
        HStack {
            Rectangle()
            VStack {
                Rectangle()
                Rectangle()
            }
        }
    }
}
```



3. The following code will produce which layout shown below? (circle one)

```
struct ContentView: View {
    var body: some View {
        HStack {
            VStack {
                Rectangle()
                    .containerRelativeFrame(.vertical, count: 5, span: 3, spacing: 0)
                Rectangle()
                    .containerRelativeFrame(.vertical, count: 5, span: 2, spacing: 0)
            }
            VStack {
                Rectangle()
                    .containerRelativeFrame(.vertical, count: 5, span: 2, spacing: 0)
                Rectangle()
                    .containerRelativeFrame(.vertical, count: 5, span: 3, spacing: 0)
            }
        }
    }
}
```



4. In SwiftUI, explain what people mean by a “push-out” view and a “pull-in” view. Give one example of each, and describe what happens when two push-out views are placed inside the same `VStack`.

A *push-out* view tries to take as much space as it can in the layout offered by its parent, one example being a `Rectangle()`

A *pull-in* view takes only the space it needs. Example: `Text("Hello")`.

If two push-out views (e.g., two `Rectangle()` views) are inside the same `VStack` with no frames set, the `VStack` divides the available vertical space between them evenly.

5. A SwiftUI view’s `body` must return exactly one view. Explain how a view can still show multiple items on screen.

`body` returns one top-level view, but that view can be a **container** (like a stack) that contains many child views—so multiple items appear on screen through **nesting**.

For example, the top-level view might be an `HStack`, and inside that single top-level view, there are several child views, which are distributed horizontally within the stack.