Stacks cheat sheet

In this lesson you will explore the three different types of stack views: `HStack`, `VStack`, and `ZStack`. You'll be able to identify the layout direction for the child views of each stack. You will explore different ways of aligning the child views of a stack. And finally you will learn different ways of creating space between the child views of a stack.

SwiftUI provides many ways to specify the layout of your user interface. One of the most common is to use the three invisible stack views: `HStack`, `VStack` and `ZStack`. Your interface may use several of these stacks, some as child views inside other stacks.

The three most common things that you need to know are:

- The layout direction of the children of the stack
- How to change the alignment of the views in the stack
- How to change the spacing of the views in the stack

You can use the code below for experimenting with different changes. Any changes display in the Xcode Canvas for your SwiftUI app. You can undo any of your experiments by replacing the `ContentView` with the code:

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        ZStack {
            Circle()
                .foregroundColor(Color.gray)
            Circle()
                .scale(x: 0.75, y: 0.75)
                .foregroundColor(Color.red)
            VStack {
                Image(systemName: "globe")
                    .imageScale(.large)
                    .foregroundColor(.accentColor)
                Text ("Hello, world!")
                HStack {
                    Button("One"){}
                    Button("Two"){}
                }
            }

        }
    }
}
```

Layout direction

Each stack lays out views in only one direction.

Use the provided code to change the order of the child views in each stack type.

HStack

Horizontal display of child views, from leading to trailing.

VStack

Vertical display of child views, from top to bottom.

ZStack

Spatial display of child views that stacks them from the bottom to the top. Child views that are higher in the stack are displayed over lower children.

Change alignment

There are two common ways to change the alignment of the stack's child views.

Optional alignment argument

First, you can use the optional `alignment` argument for a stack view. The default value is center alignment. For

example, this code creates a `ZStack` with leading alignment:

```
ZStack(alignment: .leading)
```

You can only change the alignment along one axis with this argument. The axis depends on the stack view.

Use the provided code to experiment with adding the `alignment` parameter to the different stack views. Try different alignments.

HStack

Align view children along the vertical axis: top, bottom, or center.

It can also align elements to the baseline of the first text item or the last text item. You usually use baseline alignment to make groups of text look good or to align controls.

VStack and ZStack

Align view children along the horizontal axis: leading, trailing, or center.

`.frame()` modifier

Sometimes the alignment for a stack doesn't move the views to the position you want.

When this happens, add your desired alignment in a `.frame()` modifier for the stack. You may need to add a size.

You may want to give a child view a different alignment. To do that, add a `.frame()` modifier to the child. Next add your desired alignment. Finally, set the maximum size for the frame in the direction of the alignment to infinity. For example, to align a child view to the leading edge, you set the alignment to leading and the maximum width to infinity.

One note of caution: a maximum height or width of infinity takes as much space as the parent will allow. Sometimes you need to add a frame to the parent to resize the child view.

Use the provided code to experiment with adding frames to stacks and to individual views. Experiment with different alignment values.

HStack child

This frame modifer aligns a child of an `HStack` to the top:

```
.frame(maxHeight: .infinity, alignment: .top)
```

VStack child

This frame modifer aligns a child of a `VStack` to the trailing edge:

```
.frame(maxWidth: .infinity, alignment: .trailing)
```

Change spacing

There are three ways you can change spacing when using stack views: an argument to the stack (except for `ZStack`), using `Spacer` views and using padding.

Use the provided code to experiment placing `Spacer` views in different places in the view hierarchy.

HStack and VStack

Both stacks include an optional `spacing` argument. Use this to add space between each pair of child views. It doesn't add space before the first view or after the last one.

This code sets the space between the two `Button` views to `20`

```
HStack(spacing: 20) {
    Button("One"){}
    Button("Two"){}
}
```

Spacer

A view that expands to take as much of the available space as possible. It's commonly used for creating a layout as it's not device dependent.

This code pushes the `Text` view to the top of the screen and the `Button` views in the `HStack` to the bottom:

```
VStack {
    Text ("Hello, world!")
    Spacer()
    HStack() {
        Button("One"){}
        Button("Two"){}
    }
}

```

Padding

Add a `.padding()` modifier to add space around a view. The default is to add the system-defined space to each of the view's four sides. You can also choose which sides and the amount of padding.

This code adds padding to a `Text` view:

```
Text ("Hello, world!")
    .padding()
```

Use the provided code to experiment with adding padding to stack views and individual views.

Conclusion

In this lesson you explored the three types of stack views: `HStack`, `VStack`, and `ZStack`. You learned the child view layout direction for each type of stack. You explored to two different ways of aligning the all the child views of a stack, and how to align a single child in a different direction. Finally you explored the three ways to change spacing between views using an argument for the stack: using the `spacing` parameter of `HStack` or `VStack`, adding a `Spacer` view, or adding the `.padding()` modifier.

Mark as completed

Like Dislike Report an issue