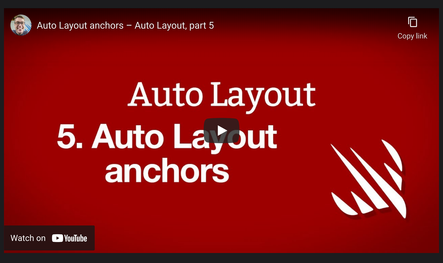
< Previous: Auto Layout metrics and priorities: constraints(withVisualFormat:)

Next: Wrap up >

Auto Layout anchors



You've seen how to create Auto Layout constraints both in Interface Builder and using Visual Format Language, but there's one more option open to you and it's often the best choice.

Every `UIView` has a set of anchors that define its layouts rules. The most important ones are `widthAnchor`, `heightAnchor`, `topAnchor`, `bottomAnchor`, `leftAnchor`, `rightAnchor`, `leadingAnchor`, `trailingAnchor`, `centerXAnchor`, and `centerYAnchor`.

Most of those should be self-explanatory, but it's worth clarifying the difference between `leftAnchor`, `rightAnchor`, `leadingAnchor`, and `trailingAnchor`. For me, left and leading are the same, and right and trailing are the same too. This is because my devices are set to use the English language, which is written and read left to right. However, for right-to-left languages such as Hebrew and Arabic, leading and trailing flip around so that leading is equal to right, and trailing is equal to left.

In practice, this means using `leadingAnchor` and `trailingAnchor` if you want your user interface to flip around for right to left languages, and `leftAnchor` and `rightAnchor` for things that should look the same no matter what environment.

The best bit about working with anchors is that they can be created relative to other anchors. That is you can say "this label's width anchor is equal to the width of its container," or "this button's top anchor is equal to the bottom anchor of this other button."

To demonstrate anchors, comment out your existing Auto Layout VFL code and replace it with this:

```
for label in [label1, label2, label3, label4, label5] {
    label.widthAnchor.constraint(equalTo: view.widthAnchor).isActive = true
    label.heightAnchor.constraint(equalToConstant: 88).isActive = true
}
```

That loops over each of the five labels, setting them to have the same width as our main view, and to have a height of exactly 88 points.

We haven't set top anchors, though, so the layout won't look correct just yet. What we want is for the top anchor for each label to be equal to the bottom anchor of the previous label in the loop. Of course, the first time the loop goes around there *is* no previous label, so we can model that using optionals:

```
var previous: UILabel?
```

The first time the loop goes around that will be nil, but then we'll set it to the current item in the loop so the *next* label can refer to it. If `previous` is not nil, we'll set a `topAnchor` constraint.

Replace your existing Auto Layout anchors with this:

```
var previous: UILabel?

for label in [label1, label2, label3, label4, label5] {
    label.widthAnchor.constraint(equalTo: view.widthAnchor).isActive = true
    label.heightAnchor.constraint(equalToConstant: 88).isActive = true

    if let previous = previous {
        // we have a previous label – create a height constraint
        label.topAnchor.constraint(equalTo: previous.bottomAnchor, constant:
    }

    // set the previous label to be the current one, for the next loop ite
    previous = label
}
```

That third anchor combines a different anchor with a constant value to get spacing between the views – these things are really flexible.

Run the app now and you'll see all the labels space themselves out neatly. I hope you'll agree that anchors make Auto Layout code really simple to read and write!

Anchors also let us control the safe area nicely. The "safe area" is the space that's actually visible inside the insets of the iPhone X and other such devices – with their rounded corners, notch and similar. It's a space that excludes those areas, so labels no longer run underneath the notch or rounded corners.

We can fix that using constraints. In our current code we're saying "if we have a previous label, make the top anchor of this label equal to the bottom anchor of the previous label plus 10." But if we add an `else` block we can push the first label away from the top of the safe area, so it doesn't sit under the notch, like this:

```
if let previous = previous {
    // we have a previous label – create a height constraint
    label.topAnchor.constraint(equalTo: previous.bottomAnchor, constant: 10
} else {
    // this is the first label
    label.topAnchor.constraint(equalTo: view.safeAreaLayoutGuide.topAnchor
}
```

If you run that code now, you should see all five labels start below the notch in iPhone X-style devices.

< Previous: Auto Layout metrics and priorities: constraints(withVisualFormat:)

Next: Wrap up >

@twostraws

paul@hackingwithswift.com