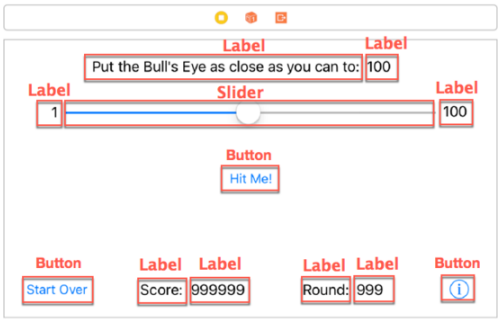
Ray Wenderlich para mim 31 de jul

Welcome back to the **raywenderlich.com** **iOS Apprentice Email Course** José!

Today, you'll add the rest of the controls necessary to complete the user interface of your app.

Your app already has a button but you still need to add the rest of the UI controls, also known as “views”. Here is the screen again, this time annotated with the different types of views:

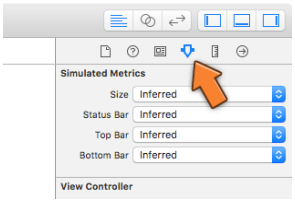


As you can see, I put placeholder values into some of the labels (for example, “999999”). That makes it easier to see how the labels will fit on the screen when they’re actually used. The score label could potentially hold a large value, so you’d better make sure the label has room for it.

Open up the Bull's Eye project where you left it off last time (or download the starter project from the corresponding forum discussion thread), and take a look at Xcode.

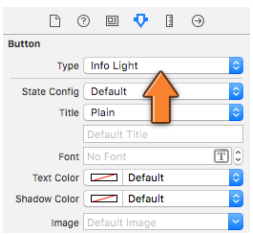
➤ Try to re-create the above screen on your own by dragging the various controls from the Object Library on to your scene. You’ll need a few new Buttons, Labels, and a Slider. You can see in the screenshot above how big the items should (roughly) be. It’s OK if you’re a few points off.

To tweak the settings of these views, you use the **Attributes inspector**. You can find this inspector in the right-hand pane of the Xcode window:

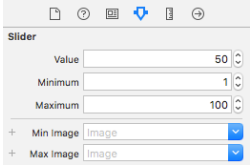


The inspector area shows various aspects of the item that is currently selected. The Attributes inspector, for example, lets you change the background color of a label or the size of the text on a button. You’ve already seen the Connections inspector that showed the button’s actions. As you become more proficient with Interface Builder, you’ll be using all of these inspector panes to configure your views.

➤ Hint: the ⓘ button is actually a regular Button, but its **Type** is set to **Info Light** in the Attributes inspector:



➤ Also use the Attributes inspector to configure the **slider**. Its minimum value should be 1, its maximum 100, and its current value 50.



When you're done, you should have 12 user interface elements in your scene: one slider, three buttons and a whole bunch of labels. Excellent!

➤ Run the app and play with it for a minute. The controls don't really do much yet (except for the button that should still pop up the alert), but you can at least drag the slider around.

The slider

The next item on your to-do list is: "Read the value of the slider after the user presses the Hit Me button."

If, in your messing around in Interface Builder, you did not accidentally disconnect the button from the `showAlert` action, you can modify the app to show the slider's value in the alert popup. (If you did disconnect the button, then you should hook it up again first. You know how, right?)

Remember how you added an action to the view controller in order to recognize when the user tapped the button? You can do the same thing for the slider. This new action will be performed whenever the user drags the slider.

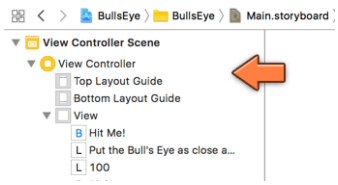
The steps for adding this action are largely the same as before.

➤ First, go to **ViewController.swift** and add the following at the bottom, just before the final closing curly bracket:

```
@IBAction func sliderMoved(_ slider: UISlider) {
  print("The value of the slider is now: \(slider.value)")
}
```

➤ Second, go to the storyboard and Control-drag from the slider to View Controller in the Document Outline. Let go of the mouse button and select **sliderMoved:** from the popup. Done!

Just to refresh your memory, the Document Outline sits on the left-hand side of the Interface Builder canvas. It shows the view hierarchy of the storyboard. Here you can see that the View Controller contains a view (succinctly named View) which in turn contains the sub-views you've added: the buttons and labels.

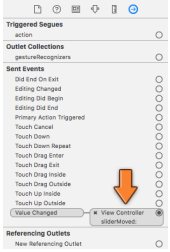


Remember, if the Document Outline is not visible, click the little icon at the bottom of the Xcode window to reveal it:

When you connect the slider, make sure to Control-drag to View Controller (the yellow circle icon), not View Controller Scene at the very top. If you don't see the yellow circle icon, then click the arrow in front of View Controller Scene (called the "disclosure triangle") to expand it.

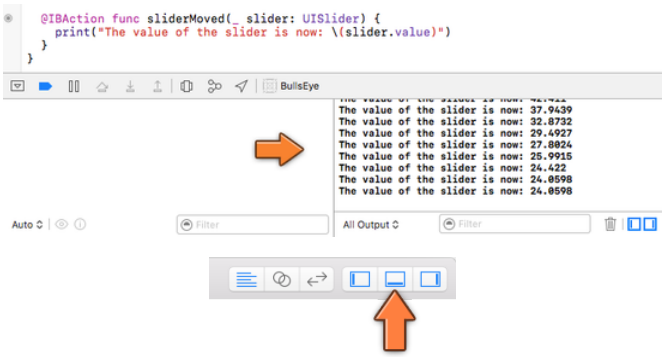
If all went well, the `sliderMoved:` action is now hooked up to the slider's Value Changed event. This means the `sliderMoved()` method will be called every time the user drags the slider to the left or right.

You can verify that the connection was made by selecting the slider and looking at the **Connections inspector**:



➤ Run the app and drag the slider.

As soon as you start dragging, the Xcode window opens a new pane at the bottom, the **Debug area**, showing a list of messages:



If you swipe the slider all the way to the left, you should see the value go down to 1. All the way to the right, the value should stop at 100.

The `print()` function is a great help to show you what is going on in the app. Its entire purpose is to write a text message to the **Console** - the right-hand pane in the Debug area. Here, you used `print()` to verify that you properly hooked up the action to the slider and that you can read the slider value as the slider is moved.

> **Note:** You may see a bunch of other messages in the Console too. This is debug output from UIKit and the iOS Simulator. You can safely ignore these messages.

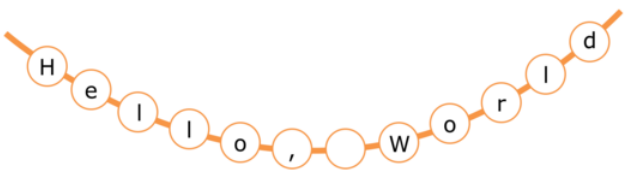
Strings

To put text in your app, you use something called a “string”. The strings you have used so far are:

```
"Hello, World"
"This is my first app!"
"Awesome"
"The value of the slider is now: \(slider.value)"
```

The first three were used to make the `UIAlertController`; the last one you used with `print()`.

Such a chunk of text is called a string because you can visualize the text as a sequence of characters, as if they were pearls in a necklace:



Working with strings is something you need to do all the time when you’re writing apps, so over this course you’ll get quite experienced in using strings.

In Swift, to create a string, simply put the text in between double quotes. In other languages you can often use single quotes as well, but in Swift they must be double quotes. And they must be plain double quotes, not typographic “smart quotes”.

To summarize:

```
// This is the proper way to make a Swift string:
"I am a good string"

// These are wrong:
'I should have double quotes'
''Two single quotes do not make a double quote''
“My quotes are too fancy”
@"I am an Objective-C string"
```

Anything between the characters `\(` and `)` inside a string is special. The `print()` statement used the string, `"The value of the slider is now: \(slider.value)"`. Think of the `\( … )` as a placeholder: `"The value of the slider is now: X"`, where X will be replaced by the value of the slider.

Filling in the blanks this way is a very common way to build strings in Swift.

Variables

Printing information with `print()` to the Console is very useful during development of the app, but it's absolutely useless to the user because they can't see the Console when the app is running on a device.

Let's improve this to show the value of the slider in the alert popup. So how do you get the slider's value into `showAlert()`?

When you read the slider's value in `sliderMoved()`, that piece of data disappears when the action method ends. It would be handy if you could remember this value until the user taps the Hit Me button.

Fortunately, Swift has a building block for exactly this purpose: the **variable**.

➤ Open **ViewController.swift** and add the following at the top, directly below the line that says `class ViewController`:

```
var currentValue: Int = 0
```

You have now added a variable named `currentValue` to the view controller object.

The code should look like this (I left out the method code, also known as the method implementations):

```
import UIKit

class ViewController: UIViewController {
  var currentValue: Int = 0

  override func viewDidLoad() {
    . . .
  }

  override func didReceiveMemoryWarning() {
    . . .
  }

  @IBAction func showAlert() {
    . . .
  }

  @IBAction func sliderMoved(_ slider: UISlider) {
    . . .
  }
}
```

It is customary to add the variables above the methods, and to indent everything with a tab, or two to four spaces. Which one you use is largely a matter of personal preference. I like to use two spaces. (You can configure this in Xcode's preferences panel. From the menu bar choose **Xcode** → **Preferences…** → **Text Editing** and go to the **Indentation** tab.)

Remember when I said that a view controller, or any object really, could have both data and functionality? The `showAlert()` and `sliderMoved()` actions are examples of functionality, while the `currentValue` variable is part of the view controller's data.

A variable allows the app to remember things. Think of a variable as a temporary storage container for a single piece of data. Similar to how there are containers of all sorts and sizes, data comes in all kinds of shapes and sizes.

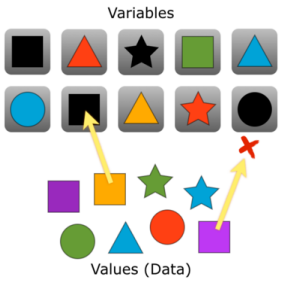
You don't just put stuff in the container and then forget about it. You will often replace its contents with a new value. When the thing that your app needs to remember changes, you take the old value out of the box and put in the new value.

That's the whole point behind variables: they can **vary**. For example, you will update `currentValue` with the new position of the slider every time the slider is moved.

The size of the storage container and the sort of values the variable can remember are determined by its **data type**, or just **type**.

You specified the type `Int` for the `currentValue` variable, which means this container can hold whole numbers (also known as "integers") between at least minus two billion and plus two billion. `Int` is one of the most common data types. There are many others though, and you can even make your own.

Variables are like children's toy blocks:



The idea is to put the right shape in the right container. The container is the variable and its type determines what "shape" fits. The shapes are the possible values that you can put into the variables.

You can change the contents of each box later as long as the shape fits. For example, you can take out a blue square from a square box and put in a red square - the only thing you have to make sure is that both are squares.

But you can't put a square in a round hole: the data type of the value and the data type of the variable have to match.

I said a variable is a **temporary** storage container. How long will it keep its contents? A variable will hold onto its value indefinitely, until you put a new value into that variable or until you destroy the container altogether.

Each variable has a certain lifetime (also known as its **scope**) that depends on exactly where in your program you defined that variable. In this case, `currentValue` sticks around for just as long as its owner, `ViewController`, does. Their fates are intertwined.

The view controller, and thus `currentValue`, is there for the duration of the app. They don't get destroyed until the app quits. Soon you'll also see variables that are short-lived (also known as "local" variables).

➤ Change the contents of the `sliderMoved()` method in **ViewController.swift** to the following:

```
@IBAction func sliderMoved(_ slider: UISlider) {
  currentValue = lroundf(slider.value)
}
```

You removed the `print()` statement and replaced it with this line:

```
currentValue = lroundf(slider.value)
```

What is going on here?

You've seen `slider.value` before, which is the slider's position at a given moment. This is a value between 1 and 100, possibly with digits behind the decimal point. And `currentValue` is the name of the variable you have just created.

To put a new value into a variable, you simply do this:

```
variable = the new value
```

This is known as "assignment". You **assign** the new value to the variable. It puts the shape into the box. Here, you put the value that represents the slider's position into the `currentValue` variable.

This game would be really hard if you made the player guess the position of the slider with an accuracy that goes beyond whole numbers.

To give the player a fighting chance, you use whole numbers only. That is why `currentValue` has a data type of `Int`, because it stores **integers**, a fancy term for whole

numbers.

You use the function `lroundf()` to round the decimal number to the nearest whole number and you then store that rounded-off number in `currentValue`.

➤ Now change the `showAlert()` method to the following:

```
@IBAction func showAlert() {
  let message = "The value of the slider is: \(currentValue)"

  let alert = UIAlertController(title: "Hello, World",
                                message: message,    // changed
                          preferredStyle: .alert)

  let action = UIAlertAction(title: "OK",          // changed
                             style: .default,
                           handler: nil)

  alert.addAction(action)

  present(alert, animated: true, completion: nil)
}
```

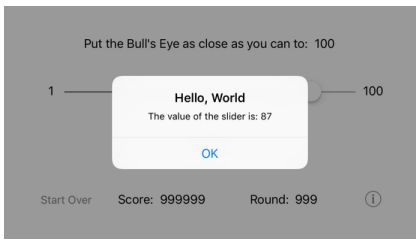
The line with `let message =` is new. Also note the other two small changes marked by comments.

As before, you create and show a `UIAlertController`, except this time its message says: “The value of the slider is: X”, where X is replaced by the contents of the `currentValue` variable (a whole number between 1 and 100).

Suppose `currentValue` is 34, which means the slider is about one-third to the left. The new code above will convert the string `"The value of the slider is: \(currentValue)"` into `"The value of the slider is: 34"` and put that into a new object named `message`.

The old `print()` did something similar, except that it printed the result to the Console. Here, however, you do not wish to print the result but show it in the alert popup. That is why you tell the `UIAlertController` that it should now use this new string as the message to display.

➤ Run the app, drag the slider, and press the button. Now the alert should show the actual value of the slider.



You have used a variable, `currentValue`, to remember a particular piece of data, the rounded-off position of the slider, so that it can be used elsewhere in the app, in this case in the alert’s message text.

If you tap the button again without moving the slider, the alert will still show the same value. The variable keeps its value until you put a new one into it.

Your first bug

There is a small problem with the app, though. Here is how to reproduce the problem:

➤ Press the Stop button in Xcode to completely terminate the app, then press Run again. Without moving the slider, immediately press the Hit Me button.

The alert now says: “The value of the slider is: 0”. But the slider’s knob is obviously at the center, so you would expect the value to be 50. You’ve discovered a bug!

And this is your next challenge in this course. Try and think what might be causing this bug, and try and fix it. If you get stuck, stop by the discussion thread on our forums, where you can check out the solution.

A large part of a developer's job is fixing bugs like this - even for experienced developers. So it's good to get started practicing early! :]

Good luck and talk to you again soon,

- Ray

If you'd like to stop receiving the iOS Apprentice Email Course but still stay subscribed to raywenderlich.com Weekly, click here.
