**Ray Wenderlich** para mim 2 de ago

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

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).

If you press the Hit Me button a few times, you'll notice that the random number never changes. I'm afraid the game won't be much fun that way.

This happens because you generate the random number in `viewDidLoad()` and never again afterwards. The `viewDidLoad()` method is only called once when the view controller is created during app startup.

The item on the to-do list actually said: "Generate a random number **at the start of each round**". Let's talk about what a round means in terms of this game.

When the game starts, the player has a score of 0 and the round number is 1. You set the slider halfway (to value 50) and calculate a random number. Then you wait for the player to press the Hit Me button. As soon as they do, the round ends.

You calculate the points for this round and add them to the total score. Then you increment the round number and start the next round. You reset the slider to the halfway position again and calculate a new random number. Rinse, repeat.

## Start a new round

Whenever you find yourself thinking something along the lines of, "At this point in the app we have to do such and such," then it makes sense to create a new method for it. This method will nicely capture that functionality in a self-contained unit of its own.

➤ With that in mind, add the following new method to **ViewController.swift**.

```
func startNewRound() {
  targetValue = 1 + Int(arc4random_uniform(100))
  currentValue = 50
  slider.value = Float(currentValue)
}
```

It doesn't matter where you put it, as long as it is inside the `ViewController` implementation (within the class curly brackets), so that the compiler knows it belongs to the `ViewController` object.

## Use the new method

First, you'll call this new method from `viewDidLoad()` to set up everything for the very first round. Recall that `viewDidLoad()` happens just once when the app starts up, so this is a great place to begin the first round.

➤ Change `viewDidLoad()` to:

```
override func viewDidLoad() {
  super.viewDidLoad()
  startNewRound()
}
```

Note that you've removed some of the existing statements from `viewDidLoad()` and replaced them with just the call to `startNewRound()`.

You will also call `startNewRound()` after the player pressed the Hit Me! button, from within `showAlert()`.

➤ Make the following change to `showAlert()`:

```
@IBAction func showAlert() {
  . . .

  startNewRound()
}
```

The call to `startNewRound()` goes at the very end, right after `present(alert, …)`.

Until now, the methods from the view controller have been invoked for you by UIKit when something happened: `viewDidLoad()` is performed when the app loads, `showAlert()` is performed when the player taps the button, `sliderMoved()` when the player drags the slider, and so on. This is the event-driven model we talked about earlier.

It is also possible to call methods directly, which is what you're doing here. You are sending a message from one method in the object to another method in that same object.

In this case, the view controller sends the `startNewRound()` message to itself in order to set up the new round. The iPhone will then go to that method and execute its statements one-by-one. When there are no more statements in the method, it returns to the calling method and continues with that – either `viewDidLoad()`, if this is the first time, or `showAlert()` for every round after.

## Different ways to call methods

Sometimes you may see method calls written like this:

```
self.startNewRound()
```

That does the exact same thing as just `startNewRound()` without `self.` in front. Recall how I just said that the view controller sends the message to itself? Well, that's exactly what `self` means.

To call a method on an object you'd normally write:

```
receiver.methodName(parameters)
```

The `receiver` is the object you're sending the message to. If you're sending the message to yourself, then the receiver is `self`. But because sending messages to `self` is very common, you can also leave this special keyword out for most cases.

## The advantages of using methods

I hope you can see the advantage of putting the "new round" logic into its own method. If you didn't, the code for `viewDidLoad()` and `showAlert()` would look like this:

```
override func viewDidLoad() {
  super.viewDidLoad()

  targetValue = 1 + Int(arc4random_uniform(100))
  currentValue = 50
  slider.value = Float(currentValue)
}

@IBAction func showAlert() {
  . . .

  targetValue = 1 + Int(arc4random_uniform(100))
  currentValue = 50
  slider.value = Float(currentValue)
}
```

The same functionality is duplicated in two places. Sure, it is only three lines of code, but often, the code you would have to duplicate will be much larger.

And what if you decide to make a change to this logic (as you will shortly)? Then you will have to make this change in two places as well.

You might be able to remember to do so if you recently wrote this code and it is still fresh in memory, but if you have to make that change a few weeks down the road,

chances are that you'll only update it in one place and forget about the other.

### Naming methods

The name of the method also helps to make it clear as to what it is supposed to be doing. Can you tell at a glance what the following does?

```
targetValue = 1 + Int(arc4random_uniform(100))
currentValue = 50
slider.value = Float(currentValue)
```

You probably have to reason your way through it: "It is calculating a new random number and then resets the position of the slider, so I guess it must be the start of a new round."

Some programmers will use a comment to document what is going on (and you can do that too), but in my opinion the following is much clearer than the above block of code with an explanatory comment:

```
startNewRound()
```

This line practically spells out for you what it will do. And if you want to know the specifics of what goes on in a new round, you can always look up the `startNewRound()` method and look inside.

➤ Run the app and verify that it calculates a new random number between 1 and 100 after each tap on the button.

### Type conversion

By the way, you may have been wondering what `Float(…)` and `Int(…)` do in these lines:

```
targetValue = 1 + Int(arc4random_uniform(100))
slider.value = Float(currentValue)
```

Swift is a **strongly typed** language, meaning that it is really picky about the shapes that you can put into the boxes. For example, if a variable is an `Int` you cannot put a `Float`, or a non-whole number, into it, and vice versa.

The value of a `UISlider` happens to be a `Float` – you've seen this when you printed out the value of the slider – but `currentValue` is an `Int`. So the following won't work:

```
slider.value = currentValue
```

The compiler considers this an error. Some programming languages are happy to convert the `Int` into a `Float` for you, but Swift wants you to be explicit about such conversions.

When you say `Float(currentValue)`, the compiler takes the integer number that's stored in `currentValue` and puts it into a new `Float` value that it can pass on to the `UISlider`.

Something similar happens with `arc4random_uniform()`, where the random number gets converted to an `Int` first before it can be stored in `targetValue`.

## Display the target value

Great, you figured out how to calculate the random number and how to store it in an instance variable, `targetValue`, so that you can access it later.

Now you are going to show that target number on the screen.

### Set up the storyboard

When you made the storyboard, you already added a label for the target value (top-right corner). The trick is to put the value from the `targetValue` variable into this label. To do that, you need to accomplish two things:

1. Create an outlet for the label so you can send it messages

2. Give the label new text to display

This will be very similar to what you did with the slider. Recall that you added an `@IBOutlet` variable so you could reference the slider anywhere from within the view controller. Using this outlet variable you could ask the slider for its value, through `slider.value`. You'll do the same thing for the label.

➤ In **ViewController.swift**, add the following line below the other outlet:

```
@IBOutlet weak var targetLabel: UILabel!
```

➤ In **Main.storyboard**, click to select the correct label - the one at the very top that says "100".

➤ Go to the **Connections inspector** and drag from **New Referencing Outlet** to the yellow circle at the top of your view controller in the central scene. (You could also drag to the **View Controller** in the Document Outline - there are many ways to do the same thing.)


➤ Select **targetLabel** from the popup, and the connection is made.

## Display the target value via code

➤ Now on to the good stuff. Add the following method below `startNewRound()` in **ViewController.swift**:

```
func updateLabels() {
  targetLabel.text = String(targetValue)
}
```

You're putting this logic into its own method because it's something you might use from different places.

The name of the method makes it clear what it does: it updates the contents of the labels. Currently it's just setting the text of a single label, but later on you will add code to update the other labels as well (total score, round number).

The code inside `updateLabels()` should have no surprises for you, although you may wonder why you cannot simply do:

```
targetLabel.text = targetValue
```

The answer again is that you cannot put a value of one data type into a variable of another type - the square peg just won't go in the round hole.

The `targetLabel` outlet references a `UILabel` object. The `UILabel` object has a `text` property, which is a `String` object. So, you can only put `String` values into `text`, but `targetValue` is an `Int`. A direct assignment won't fly because an `Int` and a `String` are two very different kinds of things.

So, you have to convert the `Int` into a `String`, and that is what `String(targetValue)` does. It's similar to what you've done before with `Float(…)` and `Int(…)`.

You could also convert `targetValue` to a `String` by using it as a string with a placeholder like you've done before:

```
targetLabel.text = "\(targetValue)"
```

Which approach you use is a matter of taste. Either approach will work fine.

Notice that `updateLabels()` is a regular method – it is not attached to any UI controls as an action – so it won't do anything until you actually call it. (You can tell because it doesn't say `@IBAction` anywhere.)

### Action methods vs. normal methods

An action method is really just the same as any other method. The only special thing is the `@IBAction` specifier. This allows Interface Builder to see the method so you can connect it to your buttons, sliders, and so on.

Other methods, such as `viewDidLoad()`, don't have the `@IBAction` specifier. This is good because all kinds of mayhem would occur if you hooked these up to your buttons.

This is the simple form of an action method:

```
@IBAction func showAlert()
```

You can also ask for a reference to the object that triggered this action, via a parameter:

```
@IBAction func sliderMoved(_ slider: UISlider)
@IBAction func buttonTapped(_ button: UIButton)
```

But the following method cannot be used as an action from Interface Builder:

```
func updateLabels()
```

That's because it is not marked as `@IBAction` and as a result Interface Builder can't see it. To use `updateLabels()`, you will have to call it yourself.

## Call the method

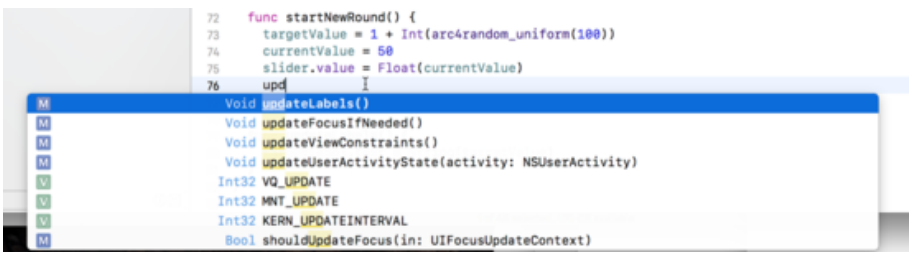
The logical place to call `updateLabels()` would be after each call to `startNewRound()`, because that is where you calculate the new target value. So, you could always add a call to `updateLabels()` in `viewDidLoad()` and `showAlert()` , but there's another way too!

What is this other way, you ask? Well, if `updateLabels()` is always (or at least in your current code) called after `startNewRound()`, why not call `updateLabels()` directly from `startNewRound()` itself? That way, instead of having two calls in two separate places, you can have a single call.

➤ Change `startNewRound()` to:

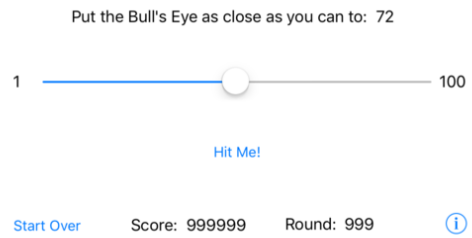
```
func startNewRound() {
    targetValue = 1 + Int(arc4random_uniform(100))
    currentValue = 50
    slider.value = Float(currentValue)
    updateLabels()  // Add this line
}
```

You should be able to type just the first few letters of the method name, like **upd**, and Xcode will show you a list of suggestions matching what you typed. Press **Enter** (or **Tab**) to accept the suggestion (if you are on the right item - or scroll the list to find the right item and then press Enter):




You don't have to start typing the method (or property) name you're looking from the beginning - Xcode uses fuzzy search and typing "date" or "label" should help you find "updateLabels" just as easily.

➤ Run the app and you'll actually see the random value on the screen. That should make it a little easier to aim for.




- Ray

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

---