|  |  |  |
| --- | --- | --- |
| |  | | --- | | **Action methods vs. normal methods**  An action method is really just the same as any other method. The only special thing is the @IBAction attribute. 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 attribute. 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 = Int.random(in: 1...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.    OK, so you have made quite a bit of progress on the game and the to-do list is getting ever shorter :] So what's next on the list now that you can generate a random number and display it on screen?  A quick look at the task list shows that you now have to "compare the value of the slider to that random number and calculate a score based on how far off the player is". Let's get to it!  **Note:** Prefer learning via video? You might like to check out videos #20-22 of the free [video version of this course](https://t.dripemail2.com/c/eyJhY2NvdW50X2lkIjoiMjY0MDEzMCIsImRlbGl2ZXJ5X2lkIjoib3kyNmllMjhueW9wdno1dHpndm8iLCJ1cmwiOiJodHRwczovL3d3dy5yYXl3ZW5kZXJsaWNoLmNvbS81OTkzLXlvdXItZmlyc3QtaW9zLWFwcD9fX3M9bWdhemh0OXI0Nm0wczY3ZndocW8ifQ), which correspond to this email.  Get the difference  Now that you have both the target value (the random number) and a way to read the slider’s position, you can calculate how many points the player scored.  The closer the slider is to the target, the more points for the player.  To calculate the score for each round, you look at how far off the slider’s value is from the target:    A simple approach to finding the distance between the target and the slider is to subtract currentValue from targetValue.  Unfortunately, that gives a negative value if the slider is to the right of the target because now currentValue is greater than targetValue.  You need some way to turn that negative value into a positive value – or you end up subtracting points from the player’s score (unfair!).  Doing the subtraction the other way around – currentValue minus targetValue – won’t always solve things either because then, the difference will be negative if the slider is to the left of the target instead of the right.  Hmm, it looks like we’re in trouble here…  And with that, it's time for a challenge!  How would you frame the solution to this problem if I asked you to solve it in natural language? Don’t worry about how to express it in computer language for now, just think it through in plain English.  **Challenge:** If you haven't already, stop reading this email and write down how to solve this problem in plain English. If you'd like to compare what you wrote with ours, [stop by the discussion thread](https://t.dripemail2.com/c/eyJhY2NvdW50X2lkIjoiMjY0MDEzMCIsImRlbGl2ZXJ5X2lkIjoib3kyNmllMjhueW9wdno1dHpndm8iLCJ1cmwiOiJodHRwczovL2ZvcnVtcy5yYXl3ZW5kZXJsaWNoLmNvbS90L3BhcnQtNy1yb3VuZHMtYW5kLXNjb3Jlcy8zODkyNz9fX3M9bWdhemh0OXI0Nm0wczY3ZndocW8ifQ) on our forums. We'd love to see what you came up with! :]  Algorithms  What you’ve just done is come up with an **algorithm**, which is a fancy term for a series of steps for solving a computational problem. This is only a very simple algorithm, but it is an algorithm nonetheless.  There are many famous algorithms, such as **quicksort** for sorting a list of items and **binary search** for quickly searching through such a sorted list. Other people have already invented many algorithms that you can use in your own programs - that'll save you a lot of thinking!  However, in the programs that you write, you’ll probably have to come up with a few algorithms of your own at some time or other. Some are simple such as the one above; others can be pretty hard and might cause you to throw up your hands in despair. But that’s part of the fun of programming. :]  The academic field of Computer Science concerns itself largely with studying algorithms and finding better ones.  You can describe any algorithm in plain English. It’s just a series of steps that you perform to calculate something. Often, you can perform that calculation in your head or on paper, the way you did above. But for more complicated algorithms doing that might take you forever, so at some point you’ll have to convert the algorithm to computer code.  The point I’m trying to make is this: if you ever get stuck and you don’t know how to make your program calculate something, take a piece of paper and try to write out the steps in English. Set aside the computer for a moment and think the steps through. How you would you perform this calculation by hand?  Once you know how to do that, converting the algorithm to code should be a piece of cake.  The difference algorithm  It is possible you came up with a different way to solve this little problem, and I’ll show you two alternatives later, but let’s convert this algorithm to computer code first:  var difference: Int  if currentValue > targetValue {  difference = currentValue - targetValue  } else if targetValue > currentValue {  difference = targetValue - currentValue  } else {  difference = 0  }  The if construct is new. It allows your code to make decisions and it works much like you would expect:  if something is true {  then do this  } else if something else is true {  then do that instead  } else {  do something when neither of the above are true  }  Basically, you put a **logical condition** after the if keyword. If that condition turns out to be true, for example currentValue is greater than targetValue, then the code in the block between the { } brackets is executed.  However, if the condition is not true, then the computer looks at the else if condition and evaluates that. There may be more than one else if, and code execution moves one by one from top to bottom until one conditon proves to be true.  If none of the conditions are found to be valid, then the code in the final else block is executed.  In the implementation of this little algorithm, you first create a local variable named difference to hold the result. This will either be a positive whole number or zero, so an Int will do:  var difference: Int  Then you compare the currentValue against the targetValue. First, you determine if currentValue is greater than targetValue:  if currentValue > targetValue {  The > is the **greater-than** operator. The condition currentValue > targetValue is considered true if the value stored in currentValue is at least one higher than the value stored in targetValue. In that case, the following line of code is executed:  difference = currentValue - targetValue  Here you subtract targetValue (the smaller one) from currentValue (the larger one) and store the result in the difference variable.  Notice how I chose variable names that clearly describe what kind of data the variables contain. Often you will see code such as this:  a = b - c  It is not immediately clear what this is supposed to mean, other than that some arithmetic is taking place. The variable names “a”, “b” and “c” don’t give any clues as to their intended purpose or what kind of data they might contain.  Back to the if statement. If currentValue is equal to or less than targetValue, the condition is untrue (or **false** in computer-speak) and execution will move on to the next condition:  } else if targetValue > currentValue {  The same thing happens here as before, except now the roles of targetValue and currentValue are reversed. The computer will only execute the following line when targetValue is the greater of the two values:  difference = targetValue - currentValue  This time you subtract currentValue from targetValue and store the result in the difference variable.  There is only one situation you haven’t handled yet, and that is when currentValueand targetValue are equal. If this happens, the player has put the slider exactly at the position of the target random number, a perfect score.  In that case the difference is 0:  } else {  difference = 0  }  Since by now you’ve already determined that one value is not greater than the other, nor is it smaller, you can only draw one conclusion: the numbers must be equal.  Display the difference  Let’s put this code into action.  To start, 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](https://t.dripemail2.com/c/eyJhY2NvdW50X2lkIjoiMjY0MDEzMCIsImRlbGl2ZXJ5X2lkIjoib3kyNmllMjhueW9wdno1dHpndm8iLCJ1cmwiOiJodHRwczovL2ZvcnVtcy5yYXl3ZW5kZXJsaWNoLmNvbS90L3BhcnQtNy1yb3VuZHMtYW5kLXNjb3Jlcy8zODkyNz9fX3M9bWdhemh0OXI0Nm0wczY3ZndocW8ifQ)).  Open up **ViewController.swift**, and add this to the top of showAlert():  @IBAction func showAlert() {  var difference: Int  if currentValue > targetValue {  difference = currentValue - targetValue  } else if targetValue > currentValue {  difference = targetValue - currentValue  } else {  difference = 0  }  let message = "The value of the slider is: \(currentValue)" +  "\nThe target value is: \(targetValue)" +  "\nThe difference is: \(difference)"  . . .  }  Just so you can see that it works, you add the difference value to the alert message as well.  ➤ Run it and see for yourself.    Nice! Hopefully you are better at this game than I am. :]  It's time for another break. Next time I'll show you some better ways to work out the difference. | |  | |