|  |  |  |
| --- | --- | --- |
| |  | | --- | | Tweaks  Obviously, the game is not very pretty yet — you will get to work on that soon. In the mean time, there are a few smaller tweaks you can make.  The alert title  Unless you already changed it, the title of the alert still says “Hello, World!” You could give it the name of the game, **Bull’s Eye**, but I have a better idea. What if you change the title depending on how well the player did?  If the player put the slider right on the target, the alert could say: “Perfect!” If the slider is close to the target but not quite there, it could say, “You almost had it!” If the player is way off, the alert could say: “Not even close...” And so on. This gives the player a little more feedback on how well they did.  **Challenge:** Stop reading this email for a moment, and think of a way to accomplish this. Where would you put this logic and how would you program it? Hint: there are an awful lot of “if’s” in the preceding sentences.  Once you've got some ideas in mind, 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/eyJhY2NvdW50X2lkIjoiMjY0MDEzMCIsImRlbGl2ZXJ5X2lkIjoiNmphYjB2M2lqZGVmYjd0ejRzeXkiLCJ1cmwiOiJodHRwczovL2ZvcnVtcy5yYXl3ZW5kZXJsaWNoLmNvbS90L3BhcnQtOS1hZGRpbmctcG9saXNoLzM5MjkzP19fcz1tZ2F6aHQ5cjQ2bTBzNjdmd2hxbyJ9)), and open up **ViewController.swift**.  The right place for this logic is showAlert(), because that is where you create the UIAlertController. You already do some calculations to create the message text and now you will do something similar for the title text.  ➤ Here is the changed method in its entirety - replace the existing method with it:  @IBAction func showAlert() {  let difference = abs(targetValue - currentValue)  let points = 100 - difference  score += points  // add these lines  let title: String  if difference == 0 {  title = "Perfect!"  } else if difference < 5 {  title = "You almost had it!"  } else if difference < 10 {  title = "Pretty good!"  } else {  title = "Not even close..."  }  let message = "You scored \(points) points"  let alert = UIAlertController(title: title, // change this  message: message,  preferredStyle: .alert)  let action = UIAlertAction(title: "OK", style: .default,  handler: nil)  alert.addAction(action)  present(alert, animated: true, completion: nil)  startNewRound()  }  You create a new string constant named title, which will contain the text that is set for the alert title. Initially, this title doesn’t have any value. (We'll discuss the title variable and how it is set up a bit more in detail just a little further on.)  To decide which title text to use, you look at the difference between the slider position and the target:   * If it equals 0, then the player was spot-on and you set title to “Perfect!” . * If the difference is less than 5, you use the text “You almost had it!” * A difference less than 10 is “Pretty good!” * However, if the difference is 10 or greater, then you consider the player’s attempt “Not even close...”   Can you follow the logic here? It’s just a bunch of if statements that consider the different possibilities and choose a string in response.  When you create the UIAlertController object, you now give it this title string instead of some fixed text.  Constant initialization  In the above code, did you notice that title was declared explicitly as being a String constant? And did you ask yourself why type inference wasn't used there instead? Also, if title is actually a constant, how do we have code which sets its value in multiple places?  The answer to all of these questions lies in how constants (or letvalues, if you prefer) are initialized in Swift.  You could certainly have used type inference to declare the type for title by setting the initial declaration to:  let title = ""  But do you see the issue there? Now you've actually set the value for title and since it's a constant, you can't change the value again. So, the following lines where the if condition logic sets a value for title would now throw a compiler error since you are trying to set a value to a constant which already has a value. (Go on, try it out for yourself! You know you want to … :])  One way to fix this would be to declare title as a variable rather than a constant. Like this:  var title = ""  The above would work great, and the compiler error would go away. But you've got to ask yourself, do you really need a variable there? Or, would a constant do? I personally prefer to use constants where possible since they have less risk of unexpected side-effects because the value was accidentally changed in some fashion - for example, because one of your team members changed the code to use a variable that you had originally depended on being unchanged. That is why the code was written the way it was. However, you can go with whichever option you prefer since either approach would work.  But if you do declare title as a constant, how is it that your code above assigns multiple values to it? The secret is in the fact that while there are indeed multiple values being assigned to title , only one value would be assigned per each call to showAlert since the branches of an if condition are mutually exclusive. So, since title starts out without a value (the let title: String line only assigns a type, not a value), as long as the code ensures that titlewould always be initialized to a value before the value stored in title is accessed, the compiler will not complain.  Again, you can test this by removing the else condition in the block of code where a value is assigned to title. Since an if condition is only one branch of a test, you need an else branch in order for the tests (and the assignment to title) to be exhaustive. So, if you remove the else branch, Xcode will immediately complain with an error like: "Constant 'title' used before being initialized".    Run the app and play the game for a bit. You’ll see that the title text changes depending on how well you’re doing. That if statement sure is handy!    Bonus points  And now, it's time for a challenge!  Your challenge is to give players an additional 100 bonus points when they get a perfect score. This will encourage players to really try to place the bull’s eye right on the target. Otherwise, there isn’t much difference between 100 points for a perfect score and 98 or 95 points if you’re close but not quite there.  Now there is an incentive for trying harder – a perfect score is no longer worth just 100 but 200 points! Maybe you can also give the player 50 bonus points for being just one off.  **Challenge:** If you haven't already, stop reading this email and try writing some code to give the player 100 bonus points when they get a perfect score. If you'd like to compare what you wrote with ours, [stop by the discussion thread](https://t.dripemail2.com/c/eyJhY2NvdW50X2lkIjoiMjY0MDEzMCIsImRlbGl2ZXJ5X2lkIjoiNmphYjB2M2lqZGVmYjd0ejRzeXkiLCJ1cmwiOiJodHRwczovL2ZvcnVtcy5yYXl3ZW5kZXJsaWNoLmNvbS90L3BhcnQtOS1hZGRpbmctcG9saXNoLzM5MjkzP19fcz1tZ2F6aHQ5cjQ2bTBzNjdmd2hxbyJ9) on our forums. We'd love to see what you came up with! :]  When you're done, run the app to see if you can score some bonus points!  Local variables recap  I would like to point out once more the difference between local variables and instance variables. As you should know by now, a local variable only exists for the duration of the method that it is defined in, while an instance variable exists as long as the view controller (or any object that owns it) exists. The same thing is true for constants.  In showAlert(), there are six locals and you use three instance variables:  let difference = abs(targetValue - currentValue)  var points = 100 - difference  let title = . . .  score += points  let message = . . .  let alert = . . .  let action = . . .  **Challenge:** Stop reading for a moment, and point out which are the locals and which are the instance variables in the showAlert() method. Of the locals, which are variables and which are constants?  Locals are easy to recognize, because the first time they are used inside a method their name is preceded with let or var:  let difference = . . .  var points = . . .  let title = . . .  let message = . . .  let alert = . . .  let action = . . .  This syntax creates a new variable (var) or constant (let). Because these variables and constants are created inside the method, they are locals.  Those six items – difference, points, title, message, alert, and action – are restricted to the showAlert() method and do not exist outside of it. As soon as the method is done, the locals cease to exist.  You may be wondering how difference, for example, can have a different value every time the player taps the Hit Me button, even though it is a constant – after all, aren’t constants given a value just once, never to change afterwards?  Here’s why: each time a method is invoked, its local variables and constants are created anew. The old values have long been discarded and you get brand new ones.  When showAlert() is called a second time, it creates a completely new instance of difference that is unrelated to the previous one. That particular constant value is only used until the end of showAlert() and then it is discarded.  The next time showAlert() is called after that, it creates yet another new instance of difference (as well as new instances of the other locals points, title, message, alert, and action). And so on… There’s some serious recycling going on here!  But inside a single invocation of showAlert(), difference can never change once it has a value assigned. The only local in showAlert() that can change is points, because it’s a var.  The instance variables, on the other hand, are defined outside of any method. It is common to put them at the top of the file:  class ViewController: UIViewController {  var currentValue = 0  var targetValue = 0  var score = 0  var round = 0  As a result, you can use these variables inside any method, without the need to declare them again, and they will keep their values till the object holding them (the view controller in this case) ceases to exist.  If you were to do this:  @IBAction func showAlert() {  let difference = abs(targetValue - currentValue)  var points = 100 - difference  var score = score + points // doesn’t work!  . . .  }  Then things wouldn’t work as you’d expect them to. Because you now put var in front of score, you have made it a new local variable that is only valid inside this method.  In other words, this won’t add points to the **instance variable** score but to a new **local variable** that also happens to be named score. In this case, the instance variable score never gets changes, even though it has the same name.  Obviously that is not what you want to happen here. Fortunately, the above won’t even compile. Swift knows there’s something fishy about that line.  Good news - at this time, you're more than halfway through the course! And I'm betting you're ready for a break. :] | |  | |