Basic Exercises Part 11.2 Internationalization

Internationalization

- Designing your apps with Internationalization in mind helps everyone use them, would help the app to climbing in the App Store and interact with a wide reach audience.
- Thanks to Internationalization features provided by Apple, there is a chance to release any app in over 150 countries, with a single click!
- Because changing languages can also change the size of UI elements, it is crucial to use Auto Layout in any app.
- What is Internationalization?
    - Is the process of designing and building the app for international compatibility.
    - For example, handle text input and output processing in the user's native language. Handle different date, time and number formats. Utilize the appropriate calendar and time zone for processing dates.
    - In an activity that you perform by utilizing system-provided APIs and making the necessary modifications into the code to make the app as good in Arabic, Chinese or French as it is in English. These modifications allow the app to be localized.
- What is Localization?
    - Is the process of translating an app's user interface and resources into different languages. Note: unless you happen to be fluent in other languages, this is something you can, and should, entrust to someone else.

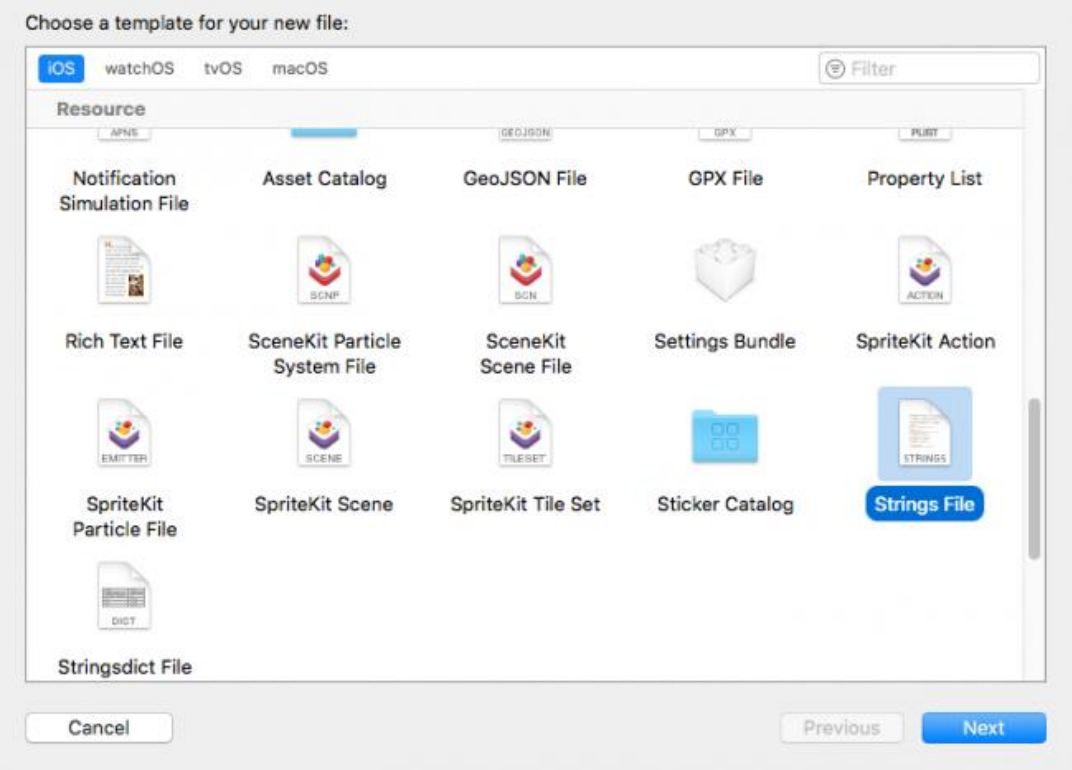
1.1 Getting started

Open the example app provided. Browse a moment to familiarize yourself with the structure.

Rather than hard-coding the strings in order to localize them, we need to retrieve the appropriate strings from a separate file in the app's bundle.

iOS uses files with the **.strings** file extension to store all of the localized strings used within the app, one more for each supported language. A simple function call will retrieve the requested string based on the current language in use on the device.

Chose **File** ▸ **New** ▸ **File** from the menu. In the resulting dialog, select **iOS** ▸ **Resource** ▸ **Strings File** and click **Next**.



Localizable.strings is the default filename iOS uses for localized text. Resist the urge to name the file something else unless you want to type the name of your **.strings** file *every* time you reference a localized string.

A **.strings** file contains any number of key-value pairs, just like a Dictionary. Conventional practice uses the development, or base, language translation as the key. The file has a specific, but simple, format:

```
"KEY" = "CONTENT";
```

Add the following to the end of Localizable.strings file:

```
"You have sold 1000 apps in %d months" = "You have sold 1000 apps in %d months";
"You like?" = "You like?";
```

Be aware, you may include format specifiers in either the key or the value portion of the string to allow you to insert real data at run time.

1.2 Introducing NSLocalizedString

NSLocalizedString is the primary tool to use in the code to access localized strings. Normally you specify only the key and the comment. The comment parameter is there for you to provide a hint to translators as to what purpose this string serves in the app's user experience.

In MainVC, add the following:

```
override func viewDidLoad() {
        super.viewDidLoad()
        likeButton.setTitle(NSLocalizedString("You like?", comment: "You like the result?"),
for: .normal)
}
```

Here, you update the title on the button using the localized value. Now, in the function likeButtonPressed() , add the following:

```
likeButton.setTitle(NSLocalizedString("You like?", comment: "You like the result?"), for: .normal)
let formatString = NSLocalizedString("You have sold 1000 apps in %d months",comment: "Time to sell 1000 apps")
salesCountLabel.text = String.localizedStringWithFormat(formatString, period)
```

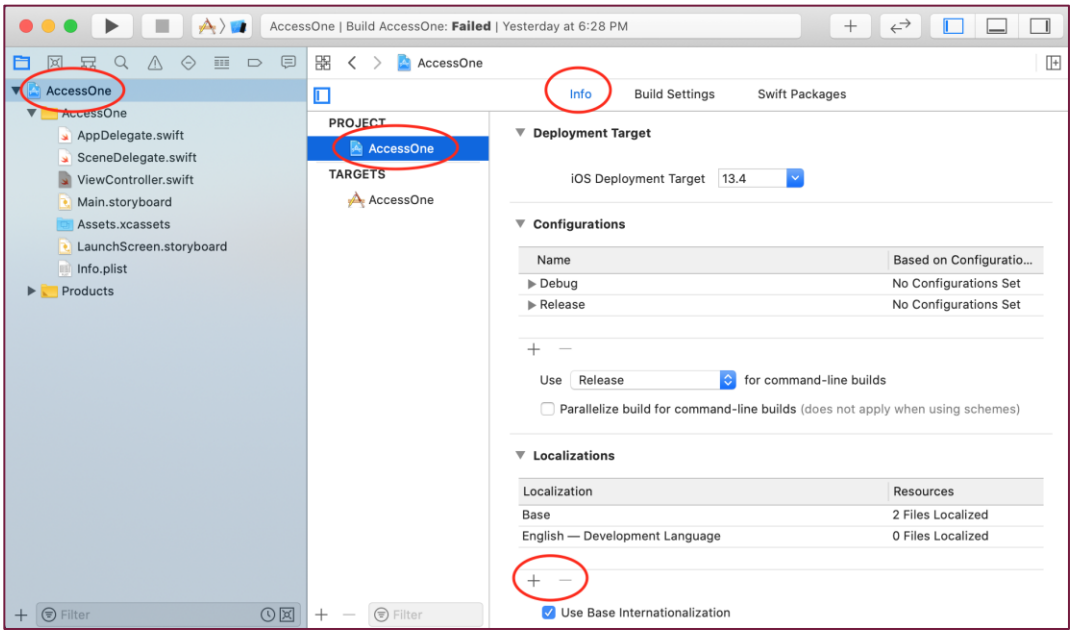
You use the String static function localizedStringWithFormat to substitute the number of months in your sales period into the localized string. This way performing the substitution respects the user[s locale setting. Remember to set the commet as descriptive as possible.

1.3 Add a Spanish Localization

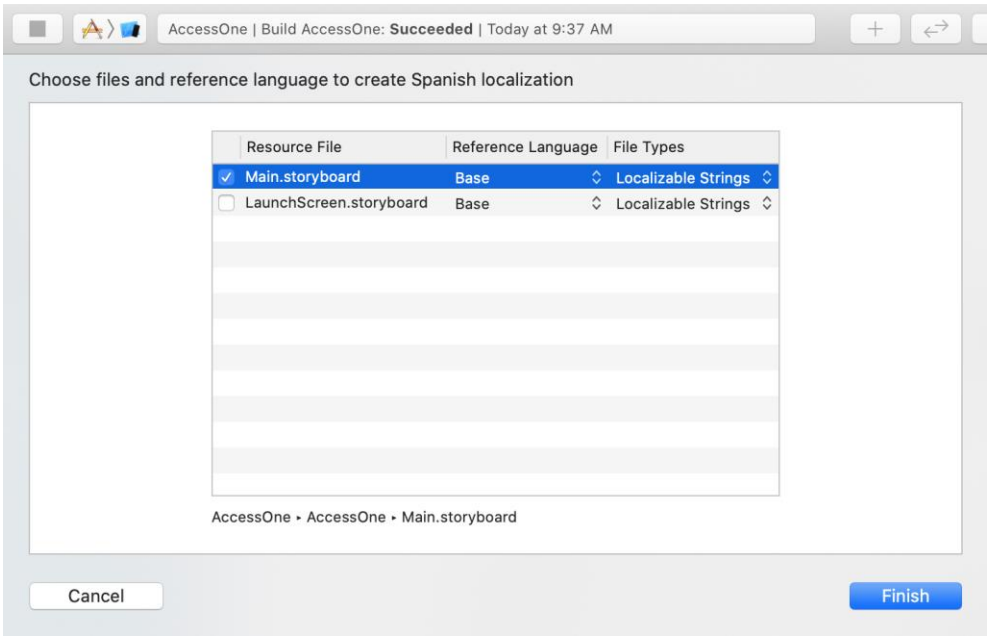
Why Spanish? This is the why:
https://en.wikipedia.org/wiki/File:Tree_map_of_languages_in_the_United_States.png :] and
https://www.babbel.com/en/magazine/most-spoken-languages-in-the-us

1.3 Adding support for another language

Click on the blue "**name**" project folder in the project navigator and select the project in the center pane (not the target). On the **Info** tab, you will see a section for **Localizations**. Click the "+" and choose **Spanish (es)** to add a Spanish localization to your project.

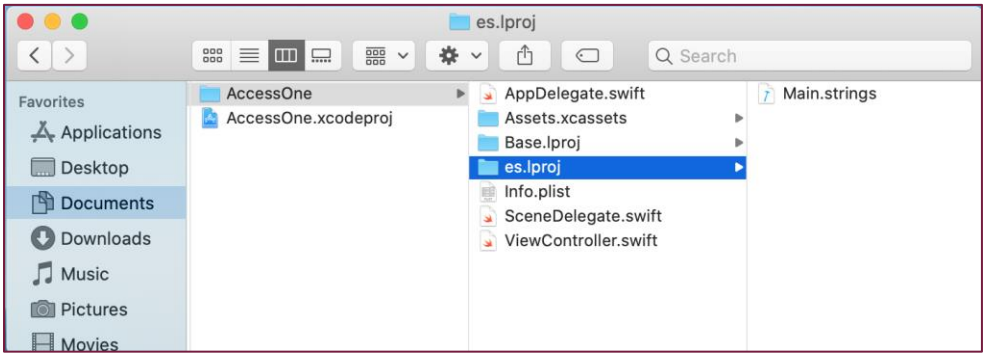


Xcode will list the localizable files in your project and allow you to select which ones it should update. Keep them all selected and click **Finish**.



Select the Main.storyboard and show the file inspector (Identity and Type). Enable the English localizable strings file:

At this point, Xcode has set up some directories, behind the scenes, to contain localized files for each language you selected. To see this for yourself, open your project folder using Finder, and you should see something similar to the following:



1.2 How to use VoiceOver

This feature is implemented using gestures:

- Apple documentation. Visit:
- Single-tap anywhere on the screen and VoiceOver will read identifying information from the item's accessibility attributes out loud.
- Single-swipe left or right and VoiceOver will select the next visible accessibility item and read it out loud. Right-swipes move forward and down, while left-swipes do the opposite.
- Single-swipe down to spell the focused item letter-by-letter.
- Double-tap to select the focused item.
- Three-finger-swipe left or right to navigate forward or backward in a page view.
- For a complete list of VoiceOver gestures, visit: https://support.apple.com/guide/iphone/learn-voiceover-gestures-iph3e2e2281/ios

1.3 Trying VoiceOver with an app in a real device

Use the project provided. Build and run on a physical device and triple click the home button to turn ON VoiceOver. Swipe left and right to navigate through the recipe list. VoiceOver reads the elements from top-left to bottom-right. It starts with the header name followed by each recipe's name and the name of the associated image.

Try it on a real device.

There're a few problems we have to fix.

1. Say: "Image", is not a helpful description of the image views in each cell. You know is there but not what it is or what represents.
2. VoiceOver says nothing about the difficulty level of each recipe, rendering this feature useless for people with visual disabilities.

1.4 Using the Accessibility Inspector

In Xcode go the menu Open → Developer Tool → Accessibility Inspector.

Some of the things you can do with this tool:

- Runs through your app and finds common accessibility issues.
- Lets you check the accessibility attributes of UI elements in Inspection Mode.
- Provides live previews of accessibility elements without leaving your app.
- Supports all platforms including macOS, iOS, watchOS and tvOS.
-

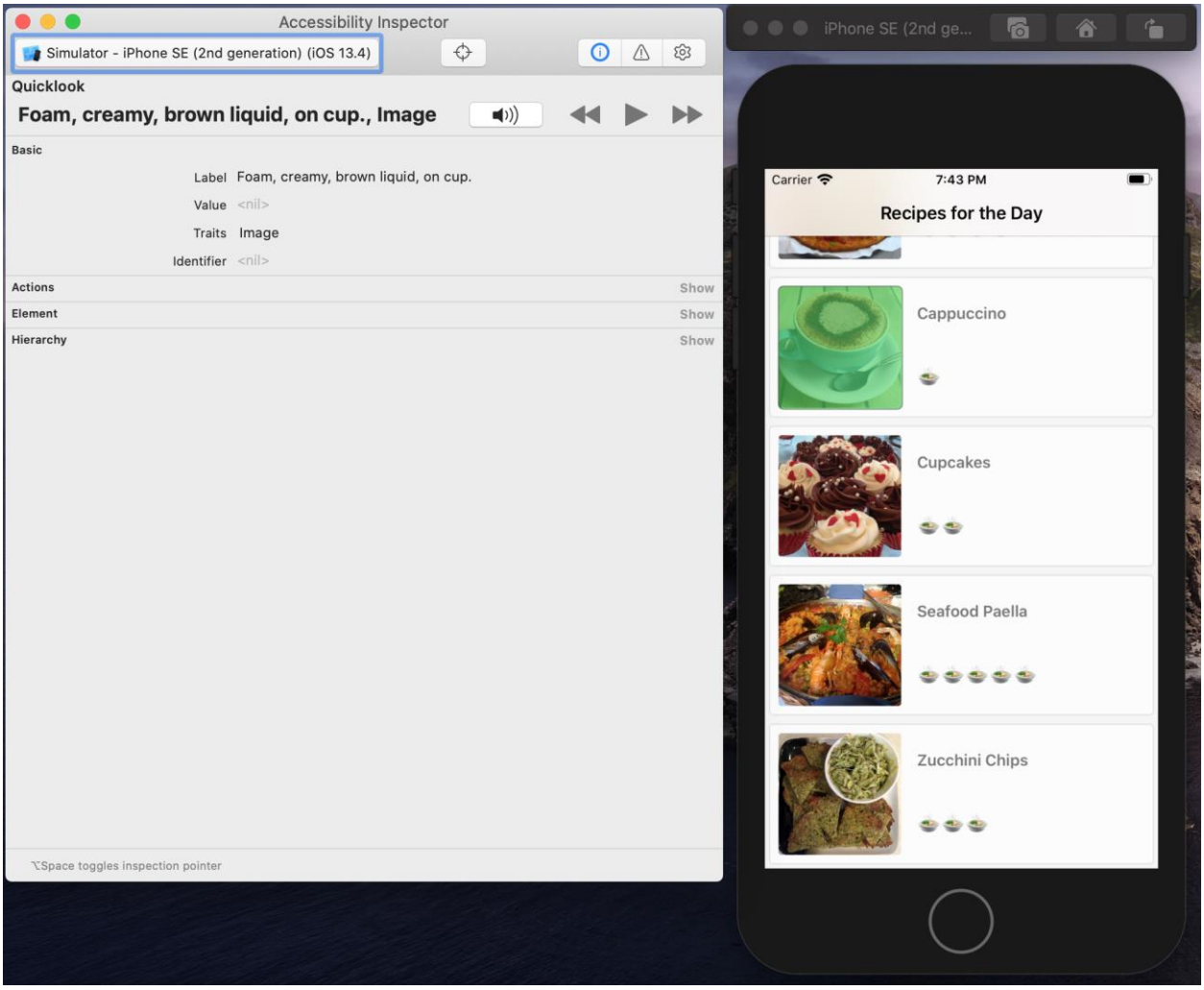
The inspector should look something like this:

Target chooser lets you pick which device you would like to inspect (your MacBook Pro, an iOS device, the simulator, etc.).

Live previews of accessibility elements let you test right in the simulator. Since live previews are faster than listening to VoiceOver, this is where you'll do the bulk of your work. Build and run in a simulator, and change the Accessibility Inspector target to your simulator.

Inspection Detail pane has everything you need to review and interact with the accessibility attributes in your app:

- **Basic**: Displays the attribute properties for the currently-highlighted element.
- **Actions**: Lets you perform actions like tapping a button or scrolling the view. Pressing the Perform button in this pane will perform the action on your target.
- **Element**: Displays the class, address and controller of the current item. As of this writing, it doesn't work consistently.
- **Hierarchy**: Shows the view hierarchy for the element, making it easier to understand complex views.



1.5 Using Quicklook to check audio

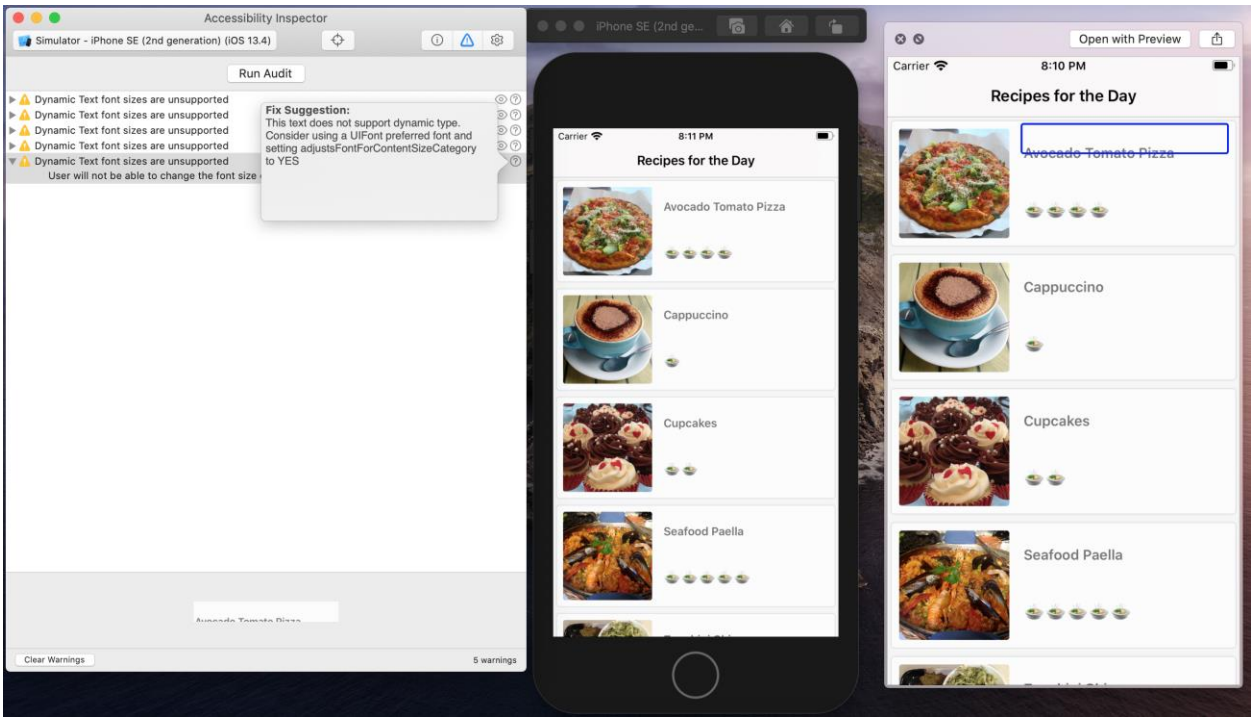
In the Inspection Detail pane, within the **Quicklook** section at the top, that allows you to simulate in Xcode the audio you'd hear on your device. This means you can check what your users hear when using your app without requiring an actual device.

Press the **Play** button while the app is running in a simulator, let the Accessibility Inspector cycle through the app and listen as it describes each element out loud.

If you prefer to manually step through each element, you can either press the **Pause** button or press the **Audio** button inside the **Quicklook** section. Press the **Forward** or Back buttons to step through each component at your own pace.

1.6 Highlighting problems with the Inspector Audit

This feature gives you warnings about accessibility problems within your app. Built and run the app. Go to the recipe list. In the inspector, click the **Audit** icon and then **Run audit**. You will see the inspector gives several warnings, including that some of your elements lacks description.



When you click a warning, Xcode highlights the related element in the simulator as well as at the bottom of the Inspector Audit screen.

In this case, the image views associated with the cells have no description. This means VoiceOver won't be able to describe them to your readers.

Click the **Suggest Fixes** icon, which looks like a question mark in a circle, for one of the warnings and the inspector will offer suggestions about how to fix the issue. You'll act on these suggestions later.

Click the **Eye** icon to take a snapshot of the app. This is useful for anyone in quality assurance who needs to create accurate bug reports.

Also, take a look at the warnings in the console:

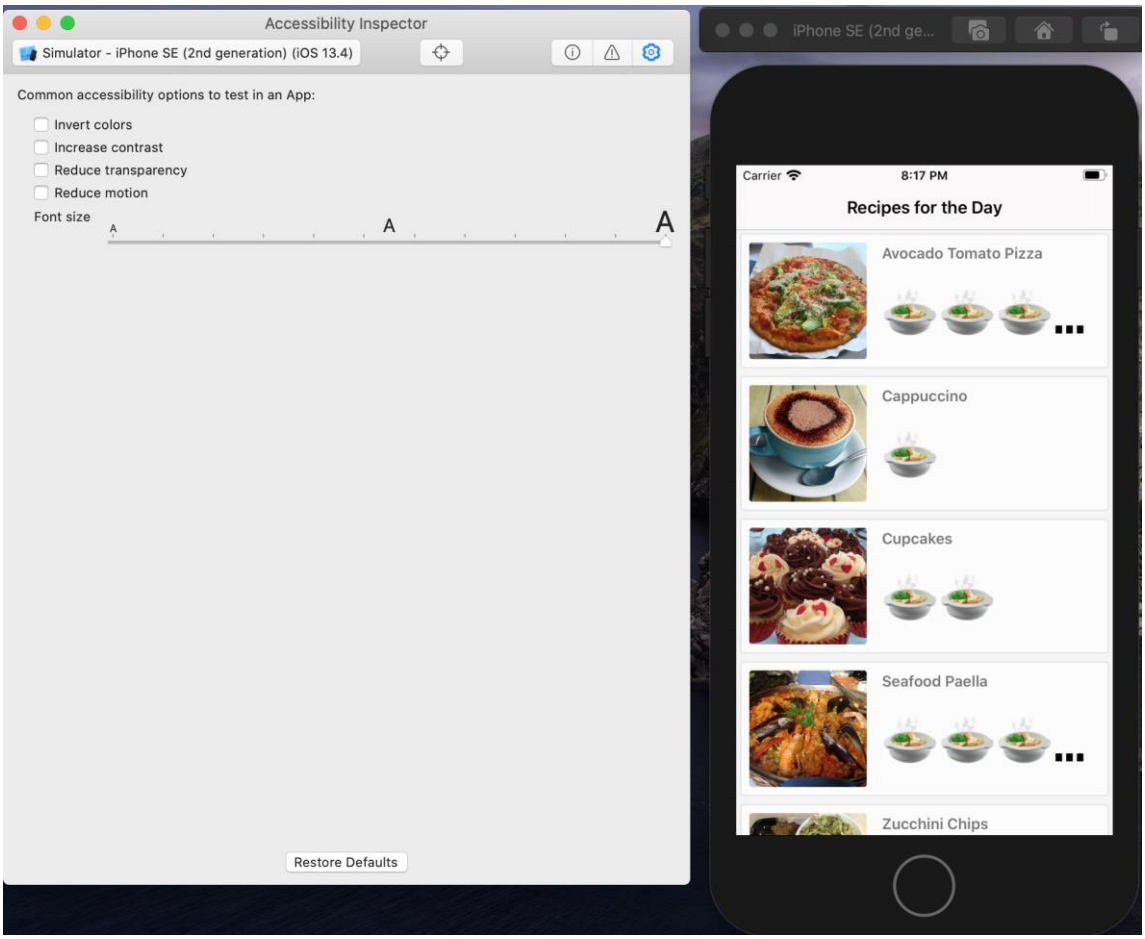
```
2020-04-18 20:09:50.331639-0400 Recipe[33410:3488886] [Accessibility] AX Audit: Element: @, with text: @, does not support these font sizes:@
2020-04-18 20:09:50.534997-0400 Recipe[33410:3488886] [Accessibility] AX Audit: Element: @ is inaccessible

2020-04-18 20:09:50.618699-0400 Recipe[33410:3488886] [Accessibility] AX Audit: Element: @ has an unknown contrast issue type.
```

1.7 Additional Inspector Settings

Although they're outside the scope of this tutorial, it's good to know the Accessibility Inspector also lets you test the following accessibility settings:

- Invert colors
- Increase contrast
- Reduce transparency
- Reduce motion
- Change font size

You no longer have to use the **Settings** app to enable these features. The Accessibility Inspector currently offers only these five options, but Apple plans to add more in the future.



The Accessibility Inspector saves time when testing your app. Remember, however, you should still test VoiceOver manually to try out the actual user experience. This final step helps you catch any problems the inspector misses.

Now you've taken a tour of the Accessibility Inspector's features, it's time to get to work your app.

1.8 Making the App Accesible

When you tested your app on your device with VoiceOver, you noted the images' descriptions weren't very useful. The **audit** tool showed you the reason why: The **image view** didn't have an accessibility label. You're going to fix this now.

In Xcode, open RecipeCell.swift and add the following code to the bottom of the file:

```
// MARK: Accessibility

extension RecipeCell {
  func applyAccessibility(_ recipe: Recipe) {
    // 1
```

```
    foodImageView.accessibilityTraits = UIAccessibilityTraits.image
    // 2
    foodImageView.accessibilityLabel = recipe.photoDescription
  }
}
```

This code fills in the missing accessibility properties based on the `Recipe` object for the cell. Here's how it works:

- `accessibilityTraits` takes a mask of traits that characterize the accessibility element. In this case, `.image` indicates it is an image.
- You use `accessibilityLabel` to describe the element in VoiceOver. Here, it's set to `recipe.photoDescription`, which is a string that describes the contents of the image.

Now, you want to apply this to future recipes, too. Find `configureCell(_:)` in the `RecipeCell` class. Add the following line to the end of the method:

```
applyAccessibility(recipe)
```

Every time you create a cell, this code will apply the accessibility attributes to the image using properties in the recipe object.

Build and run on your device and enable **VoiceOver** with **three taps** on the home button. Test the recipe list to see if the image descriptions are more meaningful.

That's great! Instead of simply hearing "Image," which provided no specific details, you now hear a full description of the image. The user can now visualize the food instead of being frustrated at not knowing what the image is.

With the app still running in the simulator, run the Accessibility Inspector again and navigate to the recipe list. Make sure you clear all warnings in the inspector and tap **Run Audit**.

1.9 Labels with description

In the Accessibility Inspector, you see **potentially inaccessible text** warnings, which tell you the difficulty labels are invisible to a user with visual impairments. To fix these, you need to make the labels accessible and update their properties with a meaningful description.

For your next step, go to `RecipeCell.swift` and add the following to the end of applyAccessibility(_:):

```
  // 1
```

```
difficultyLabel.isAccessibilityElement = true
// 2
difficultyLabel.accessibilityTraits = UIAccessibilityTraits.none
// 3
difficultyLabel.accessibilityLabel = "Difficulty Level"
// 4
switch recipe.difficulty {
case .unknown:
  difficultyLabel.accessibilityValue = "Unknown"
case .rating(let value):
  difficultyLabel.accessibilityValue = "\(value)"
}
```

Here's some more detail about what this code does:

- `isAccessibilityElement` is a flag that makes the item visible to accessibility features when `true`. For most UIKit classes, the default is `true`, but for UILabel it's `false`.
- `accessibilityTraits` helps characterize the accessibility element. Since you don't need any interactions, you set it to have no traits.
- Next, you have VoiceOver concisely identify the intent of this label. `Difficulty Level` lets the user know exactly what they're dealing with.
- VoiceOver will read the `accessibilityValue` as part of the label description. Setting the difficulty level here makes this element much more useful.

Build and run your app on a physical device, **triple tap** the home button to enable VoiceOver and swipe through the recipe list.

1.10 Making the text dynamic

The auditor is warning you that you are missing an important step to make your app usable by everyone: dynamic text. This is an important feature for accessibility, allowing users with partial vision impairments to increase the font size for readability. The non-dynamic font your app currently uses doesn't allow this.

Click the **Fix Suggestions** icon to see what the auditor recommends. It tells you to use a UIfont preferred font and to set `adjustsFontForContentSizeCategory` to true. You'll do this now.

Within **RecipeCell.swift** add the following code inside `applyAccessibility(_:)` at the very bottom:

```
dishNameLabel.font = .preferredFont(forTextStyle: .body)
dishNameLabel.adjustsFontForContentSizeCategory = true
```

```
difficultyLabel.font = .preferredFont(forTextStyle: .body)
difficultyLabel.adjustsFontForContentSizeCategory = true
```

This sets the `preferredFont` to a `body` style, which means iOS will style text as it would the body of a document. The specifics of the size and font depend on the accessibility settings. `adjustsFontForContentSizeCategory` indicates the font should update automatically when the user changes the text content size.

Testing how your app handles resizing fonts is easy, thanks to the Accessibility Inspector.

Build and run the recipe app alongside the Accessibility Inspector. Run the audit again and all your warnings should be gone.

1.11 Transforming the Details Screen

Now you've taken care of the list of recipe options, you want to see what happens when a user clicks one of the recipes. Run the app on your device, enable VoiceOver and look around the detail view.

There are some issues with the VoiceOver interaction in the detail view:

- **Left Arrow button** isn't a great description for the navigation. How would the user know what the button does?
- The emoji face states are: **heart shape eyes** and **confounded face**. Those explanations would confound any user!
- When the user selects a checkbox, it reads **icon empty**, which doesn't explain much.

In each of these cases, it's important to explain what the state of the control means, rather than how it looks. **Back button** is clearer than **Left Arrow button**. **Like** and **Dislike** succinctly explain the emojis. You'll make these two changes next.

To change the navigation, open **RecipeInstructionsViewController.swift**and add the following to `viewDidLoad`, after `assert(recipe != nil)`:

```
  backButton.accessibilityLabel = "back"
  backButton.accessibilityTraits = UIAccessibilityTraits.button
```

Instead of **Left Arrow button**, VoiceOver now says **Back button**.

Now, on to the emojis. In the same file, replace the contents of `isLikedFood(_:)` with the following:

```
func isLikedFood(_ liked: Bool) {
  if liked {
```

```
    likeButton.setTitle("😍", for: .normal)
    likeButton.accessibilityLabel = "Like"
    likeButton.accessibilityTraits = UIAccessibilityTraits.button
    didLikeFood = true
  } else {
    likeButton.setTitle("🤢", for: .normal)
    likeButton.accessibilityLabel = "Dislike"
    likeButton.accessibilityTraits = UIAccessibilityTraits.button
    didLikeFood = false
  }
}
```

For both Like and Dislike modes, you've added an `accessibilityLabel` that's clear about what the button does. You also set `accessibilityTraits` to identify it as a button, so the user knows how they can interact with it.

Build and run on a device and enable **VoiceOver**. Navigate to the detail recipe screen using VoiceOver to test your changes to the buttons at the top of the view. Now, each of these features has clear, short descriptions that help the user understand their intent

1.12 Improving Checkboxes

The final issue is with the checklist. For each checkbox, VoiceOver currently states **icon empty** followed by the recipe instruction. That's not clear at all!

To change this, open **InstructionCell.swift** and look for `shouldStrikeThroughText(_:)`. Replace the entire `if strikeThrough` statement with the following:

```
  // 1
  checkmarkButton.isAccessibilityElement = false

  if strikeThrough {
    // 2
    descriptionLabel.accessibilityLabel = "Completed: \(text)"
    attributeString.addAttribute(NSAttributedString.Key.strikethroughStyle, value: 2, range:
NSRange(text.startIndex..., in: text))
  } else {
    // 3
    descriptionLabel.accessibilityLabel = "Uncompleted: \(text)"
  }
```

Here’s what this code does:

- Turns off accessibility for **checkmark button** so VoiceOver reads it as one unit instead of two different accessibility elements.
- The `accessibilityLabel` for the description now uses the hard-coded string `"Completed"` followed by the text. This provides all the necessary info with a single visit to the label.
- As with the completed code, if a user marks an item as uncompleted, you add `"Uncompleted"` before the label description.

Build and run the app again and see how it sounds. It will be music to the ears of your users.