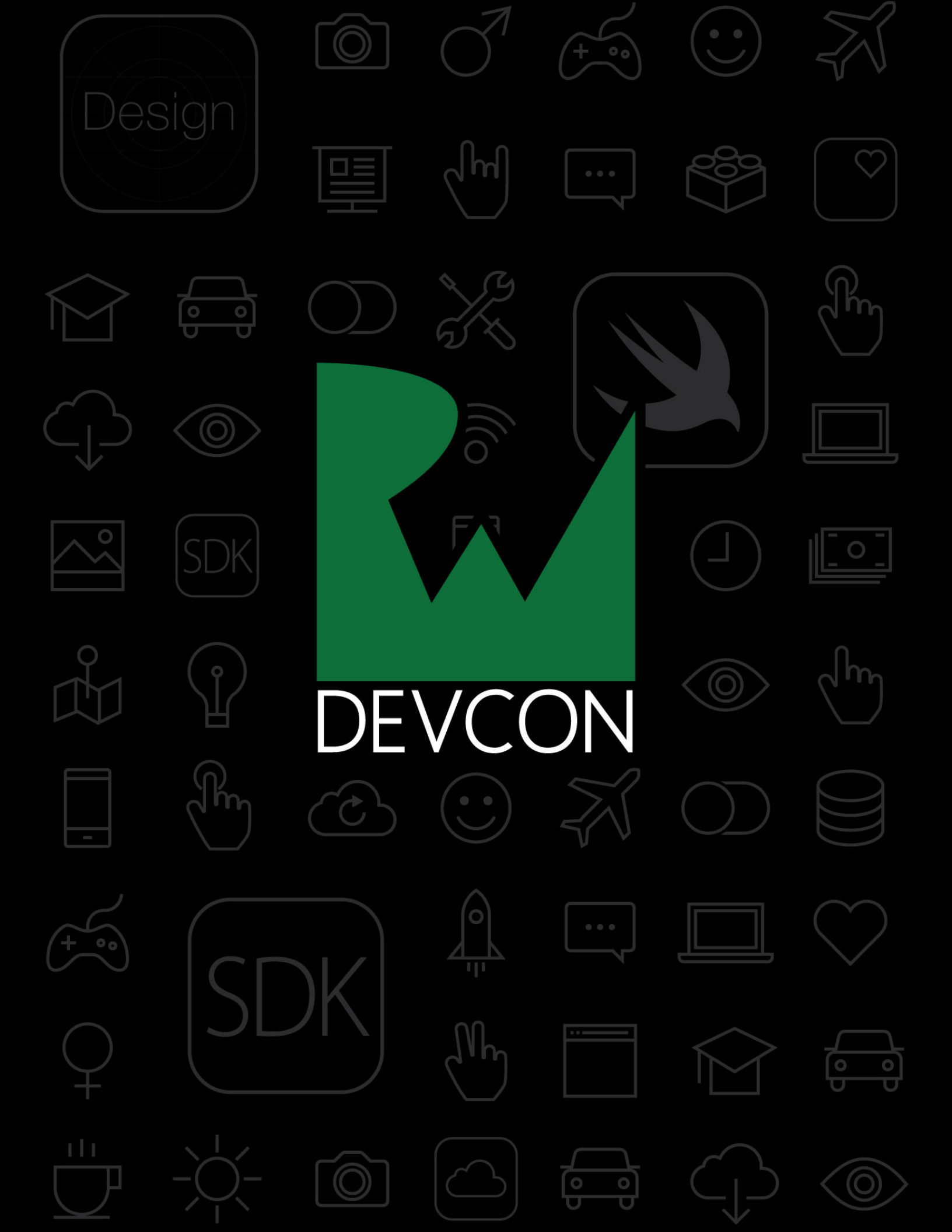
Design
SDK
DEVCON
SDK

RWDevCon 2018 Vault

By the raywenderlich.com Tutorial Team

Notice of Rights

Notice of Liability

Trademarks

Table of Contents: Overview

Table of Contents: Extended

17: Xcode Tips & Tricks

As an iOS developer, the most important tool you use is Xcode. Learn how to supercharge your efficiency with various tips and tricks.

49 Xcode Tips & Tricks: Demo 1

By Jawwad Ahmad

In this demo, you will learn about the View Debugger and Conditional Breakpoints.

The steps here will be explained in the demo, but here are the raw steps in case you miss a step or get stuck.

> **Note**: Begin work with the starter project in **Demo1\starter**.

1) Minimize distractions

Before we dive into the project, in order to focus effectively, a good idea is to minimize distractions, so we'll disable notifications for the duration of this talk.

On the top right-hand side of your screen in macOS, there is a notifications drawer. Click on it, scroll down and turn on **Do Not Disturb**.

2) Show Line Numbers

I'll occasionally be referring to line numbers so let's make sure that you have line numbers turned on in Xcode. Press **Command-comma** to open up Xcode's preferences, click on the **Text Editing** tab, and make sure there is a checkmark in **Show Line Numbers**.

While we are here, a quick aside, I'd highly recommend turning on the trim whitespace-only lines option, which helps in minimizing whitespace changes in diffs.

Hit **Escape** to dismiss preferences.

3) Build and run

Build and run the app.

Go to the **Calories to Fruits** tab. Press backspace a few times and note that you are not able to delete the final number.

4) Open the View Debugger

Go back to Xcode and click on the **Debug View Hierarchy** button in Xcode above the console area.

Now drag the snapshot view to the right to see a 3D representation of the app's view hierarchy.

5) Increase spacing between views

The slider on the bottom left can be used to increase the spacing between the views. Drag the knob to the right a bit, and bring it approximately to the middle.

The name of the view controller may be a bit difficult to read, but if you click on it to select it, you'll see the name of the view controller in the view hierarchy bar on the top right.

You'll see that the view controller's name is `CalorieConversionViewController`, and also note that it has an embedded `FruitDisplayViewController` as a child view controller rather than having a simple `UITableView`.

Also, click on the cell and note that the cell's name is `FruitDisplayCell`.

In Xcode 9 you can now see the names of the view controllers within the view hierarchy in the **Debug navigator** which makes it incredibly more useful.

Selecting the view or view controller in the **Debug navigator** automatically selects it in the snapshot view.

Select the `CalorieConversionViewController` in the **Debug navigator**, which is the 3rd view controller down, and note that the view controller gets selected in the snapshot view. In the **Debug navigator** click on the `UIView` below it, and you'll see that its view gets selected as well.

This automatic selection does not work the other way around, but there is a keyboard shortcut that you can use to do so.

In the **Snapshot view** select the frontmost view controller and note that the selection in the **Debug navigator** doesn't change.

Now use the **Command-Shift-D** shortcut to reveal the view controller in the **Debug navigator**. This is an important shortcut to know since the Reveal-in-Debug-Navigator option doesn't appear in the right-click context menu.

6) Determine the UITextField subclass

The app is using a custom subclass of `UITextField`, and you can determine the name of it by clicking on the text field in the view debugger.

Click on the label that contains the number **5**, and if you check the name in the top right in the view hierarchy bar, you'll see that it selects a view named **_UITextFieldContentView** which is an internal subview of `UITextField`.

Move your mouse to the left to where you see the **F** icon which is the text field. You'll see that it says **ApplesToOranges.NumericalInputTextField - 5**, so the text field's class is `NumericalInputTextField`.

Also, note that if you keep moving your mouse to the left, it will show you the complete name of the view or view controller that your mouse is over.

You can also navigate to code for the `NumericalInputTextField` class directly from here, but first, you'll have to select it. Click on the **ApplesToOranges.NumericalInputTextField - 5** dropdown in the view hierarchy bar and then just click on it again to select it.

7) Navigate to the class from the debugger

In the **Object inspector** pane on the right, click on the little right arrow next to the class name.

This should take you directly to the `NumericalInputTextField` class.

8) Show Document Items

Your first instinct may be to scroll down to the end of the file to skim it, but a better way to get an initial summary of what variables and methods a file contains is via **Control-6** so press that now.

The hamburger icon denotes all of the **// MARK:** comments and the **Ex** icon denotes any defined extensions.

9) Mark Format

Note that in the dropdown there is a horizontal line above **UITextFieldDelegate** but not above **Initializers** or **Variables**. This is because the `MARK` comment for `UITextFieldDelegate` includes a horizontal dash but the others don't.

10) Filter by name of method

The bug in the `textField(_:shouldChangeCharactersIn range: NSRange, replacementString string:)` method.

With the dropdown still showing (press **Control-6** if it isn't), type **should** which will filter it down to two methods, and type **c** to filter it down to one method.

Filtering it down to one method is useful because you can now press **Enter** to select the method, so go ahead and press **Enter** to navigate to the method.

The return value depends on `textAfterChange`, so you want to inspect its value. You could add a print statement, but then you'd have to build and run again. While this app is small, if your app is larger it would take some time to recompile. Even if your compile time is small, it will add up over time.

Also every time you build and run you'd have to navigate back that place in the app that has the bug.

So instead of using a print statement, we'll use a conditional breakpoint.

11) Add a breakpoint

Move your cursor to line 71 and use **Command-** to add a breakpoint and then **double-click** on the breakpoint to edit it.

Leave the **Condition** field blank and click on **Add Action**. From the dropdown select **Debugger Command**.

Since Xcode 9 now has autocomplete functionality in all breakpoint fields type **po text** and select **textAfterChange** from the dropdown list so that you end up with:

```
undefined
```

Finally, add a checkmark to **Automatically continue after evaluating actions** and hit **Escape** to dismiss the popup.

Note that the breakpoint now has a small white triangle in it indicating that it now has a condition, which is also a new Xcode 9 feature.

If you hover over the breakpoint with your mouse, you'll see that it says **This breakpoint has an action and will automatically continue**.

Go back to the simulator and type an **a** and you'll see the number with an **a** but since **5a** can't be converted to a **Double**, the method returns **false** which is on line 79.

Now press backspace a few times, and you'll see that an empty string is printed to the console. An empty string can't be converted to a double either, so the method disallows the change by returning false.

So you actually do want to allow an empty string. So the logic would be this:

```
if textAfterChange.isEmpty {
  return true
} else {
  return false
}
```

Which you can simplify to:

```
return textAfterChange.isEmpty ? true : false
```

Which is the same as saying:

```
return textAfterChange.isEmpty
```

Another benefit of using breakpoints instead of print statements is that you can keep them around for later use, whereas you'd have to delete a print statement.

Since we are done with this breakpoint, go ahead and disable it by clicking on it once.

12) Use the Log Message action

Next, you'll use the **Log Message** action to log a custom message.

Click on line 70 to add a new breakpoint. Now **double-click** on it to edit it, click **Add Action** and select **Log Message** from the dropdown.

Enter the following text:

```
before: @currentText@, after: @textAfterChange@
```

Now add and delete a few numbers and observe the text in the console.

Finally, click on the breakpoint once to disable it.

13) Use the Debugger Command action

Next, you'll use the **Debugger Command** action to modify the `max` variable without

recompiling the app.

Triple-click in the gutter of line 69 to add and edit a new breakpoint. Click **Add Action**, choose **Debugger Command** and type in the following:

```
expression max = 2001
```

Note that you can now enter numbers up to 2001.

14) Move the instruction pointer

Go to line 79 and add a breakpoint via the keyboard shortcut which is... do you remember, recall the "opposite of comment" mnemonic. It's **Command-**.

Type an **a**, and once you hit the breakpoint, drag the instruction pointer handle on the right up to line 73 to force the method to return `true`.

Click **Move** on the confirmation dialog, and click the **play** button to continue and note that the number now has an **a** at the end.

15) Navigate to the 2nd tab upon app startup

Sometimes when debugging, for convenience, you want your iOS app to start off on a different tab.

Use the **Command-Shift-O** shortcut launch the open quickly dialog, type in **fcvc** and hit **Enter**.

Use **Control-6** and type in **viddd** just to remind you that you can use fuzzy search here as well.

Press **Enter** and in `viewDidLoad` add a breakpoint to line 58.

Use the following for the expression command but use **e** which is shorthand for **expression**:

```
e tabBarController?.selectedIndex = 1
```

Build and run and note that your app will start up on the 2nd tab, reducing one step in your debugging workflow.

When you are done, disable the breakpoint.

16) Filtering breakpoints

Press **Command-8** to go to the breakpoints tab.

Click on the triangle dropdown but also hold down the **Option** key while clicking to expand all.

You can filter by enabled breakpoints and also by ones that have a condition using the two tiny buttons at the bottom. Try them both.

Press **Command-Option-J** to focus on the filter area. Type in `selectedIndex` to filter by that text.

You can also use the tiny buttons to the right of the filter command to filter by modified breakpoints or enabled breakpoints.

17) That's it!

Congrats, at this time you should have a good understanding of Conditional Breakpoints.

50 Xcode Tips & Tricks: Demo 2

By Jawwad Ahmad

In this demo, you'll learn about strategies to reduce friction in your development workflow. You'll learn about code snippets and keyboard shortcuts and a few other tips and tricks along the way.

The steps here will be explained in the demo, but here are the raw steps in case you miss a step or get stuck.

> **Note**: Begin work with the starter project in **Demo2\starter**.

1) Code snippets

In **FruitConversionViewController.swift** navigate to line 62 and add a `viewWillAppear` method:

```
override func viewWillAppear(_ animated: Bool) {
  super.viewWillAppear(animated)

}
```

Next, add a placeholder token for your code by adding text and surrounding it with <# and #> to turn it into a token:

```
override func viewWillAppear(_ animated: Bool) {
  super.viewWillAppear(animated)

  <#code#>
}
```

Use **Command-Option-Control-2** to reveal the code snippets area.

Select the method and keep the mouse button held down for a split second until you get the pointer and drag it into the code snippets area.

Give it a title of **viewWillAppear** as well as a completion shortcut of **viewWillAppear** and press **Enter**.

Now delete the method and start typing **viewWillAppear**. Select the one with the parentheses to the left of it which indicates that it is a code snippet.

Delete the newly added code since it isn't needed at the moment.

2) Snippet for MARK

Press **Control-6** and type in **data**, press enter to navigate to **// MARK: - UIPickerViewDataSource**. Now **triple-click** the line to select the complete line and drag it to the code snippets area.

Add **mark** for the title as well as the completion shortcut. Replace the **UIPickerViewDataSource** text with **<#text#>**. Click on **Done**.

3) Code Snippets Location

Code snippets are stored in `~/Library/Developer/Xcode/UserData/CodeSnippets`.

Open a finder window, press **Command-Shift-G** and enter **~/Library/Developer/Xcode/UserData/CodeSnippets/** as the folder to go to.

To copy your snippets to a different computer, you just need to move these files, and your snippets will show up on your other machine.

4) Naming your snippets

Select the snippet you just added. Press the spacebar to open quick-look to ensure that it's for **mark**. Delete the UUID, and replace it with a name, for example, **swift-mark** or just **mark**. Ensure that you don't delete the `.codesnippet` extension.

I prefer to prefixing the name with the language since I like to archive my Objective-C snippets if I'm not working on an Objective-C project.

5) Syncing snippets

If you use more than one computer, you'll also want to sync your snippets. If you don't, you won't be sure if the computer you are on will have that snippet you added and if it doesn't then you will have to add it.

There is no way to natively sync code snippets, but you can sync them on your own

using something like Dropbox.

If you are following along live, don't do these just yet but the steps you would take are the following:

First Time:

```
# Backup snippets just in case
cp -Pr ~/Library/Developer/Xcode/UserData/CodeSnippets{,.bak}

# Create the destination path where the CodeSnippets folder will be
stored in Dropbox. This can be anything.
mkdir -p ~/Dropbox/Library/Developer/Xcode/UserData

# Move CodeSnippets to Dropbox
mv ~/Library/Developer/Xcode/UserData/CodeSnippets ~/Dropbox/Library/
Developer/Xcode/UserData/

# Add a symlink to the Dropbox CodeSnippets folder
ln -s ~/Dropbox/Library/Developer/Xcode/UserData/CodeSnippets ~/Library/
Developer/Xcode/UserData
```

Subsequent Times:

```
# Add your code snippets on this computer to the existing folder in
Dropbox
cp ~/Library/Developer/Xcode/UserData/CodeSnippets/* ~/Dropbox/Library/
Developer/Xcode/UserData/CodeSnippets/

# Backup just in case
mv ~/Library/Developer/Xcode/UserData/CodeSnippets{,.bak}

# Add a symlink to the Dropbox CodeSnippets folder
ln -s ~/Dropbox/Library/Developer/Xcode/UserData/CodeSnippets ~/Library/
Developer/Xcode/UserData

# Restart Xcode
```

6) Move to beginning or end of a line

Use **Command-Left** to go to the beginning of a line and **Command-Right** to go to the end of a line.

You can also use **Control-A** and **Control-E** as well. I like the control versions of the shortcuts since you don't need to move your hand to the arrow keys.

7) Move to beginning or end of a word

You can hold down the **Option** key while using the **Left** and **Right** arrow keys to move directly to the beginning or the end of a word.

Quickly try this out if you haven't used this before.

8) Enable move subword shortcut

You can also move your cursor through words by camel-cased words, called subwords. There is a **Move Subword Forward** shortcut in Xcode **Control-Forward/Backward**, but it may or may not work by default since there is a system preferences shortcut that overrides it.

Press **Command-comma** to open Xcode's preferences, click on the **Key Bindings** tab and in the filter box type in **subword**. Select the **Move Subword Forward** line.

If at the bottom if it says **Shortcuts from System Preferences may take precedence over this shortcut** then you need to disable these shortcuts in system preferences.

Go to System Preferences > Keyboard > Shortcuts > Mission Control and uncheck **Move left a space** as well as **Move right a space**.

Now go back to Xcode, and you'll see that you can hold down **Control** and use the arrow keys to move through subwords.

You can also delete individual subwords by pressing **Backspace** while holding down **Control**

9) Jump to definition and back

Navigate to line 41 and place your cursor anywhere in `FruitDisplayViewController`.

Use **Control-Command-J** to jump to its definition.

Use **Control-Command-Left** to jump back.

If you know, one of these shortcuts its easier to remember the other since they both use the **Control-Command** modifier keys.

10) Refactoring

Refactoring for swift finally returned in Xcode 9. There is no default shortcut to refactor so right click on `FruitDisplayViewController` select **Refactor** and then **Rename**.

Note that the filename and the class name in the storyboard are included in the rename. However there is a comment that is not included, but you can click on the

plus to select it.

Use the right arrow key to place the cursor at the end of **FruitDisplayViewController**, and while holding down **Control** press the left button twice to position your cursor right before **ViewController**. Add the word **Table** and press **Enter**.

11) Edit all in Scope:

You also want to rename the `fruitDisplayViewController` variable to `fruitDisplayTableViewController`.

If you want to rename a local or private variable, it's easier to use **Edit All In Scope**.

Move your cursor to anywhere within the `fruitDisplayViewController` variable.

12) Find the shortcut:

Click on **Help** and type **edit scope**. You'll see the **Edit All in Scope** item. Use the down arrow key to select it.

You could just click it here to run the command but you want to use the shortcut which you can see is **Command-Control-E**.

Hit **Escape** twice to exit the help menu and use the **Command-Control-E** shortcut to edit it. Using **Control** and the arrow keys move your cursor to the middle of the variable, add the word **Table** and hit **Enter**.

Now your code will continue to work, it doesn't matter what your **Storyboard ID** is but your mentor says that they must match. -->

13) Update Storyboard IDs

One caveat to note here is that if you used a **Storyboard ID** Xcode won't replace it since there is technically no relation between a class name and a Storyboard ID, even though it is a convention that developers use.

Using **Command-Option-Shift-F**, replace all instances of **FruitDisplayViewController** with **FruitsDisplayTableViewController**.

14) The shortcut to look up shortcuts

You know that you can use the help menu to search for a shortcut, but shouldn't there be a shortcut to get to the help menu? There is, and it is **Command-?**, and since question mark is **Shift-/** its really **Command-Shift-/**.

15) View call hierarchy

You'll lookup the shortcut to find the call hierarchy of the configure method of `FruitDisplayTableViewController`.

Go to line 80 and position your cursor somewhere inside the word `configure`.

You could right-click on `configure`, and select **Find Call Hierarchy**, but you want to use the shortcut, and it doesn't show it to you there.

Press **Command-Shift-/** to open the help menu and type **hier**. Press the **Down** arrow key so that the **Find Call Hierarchy** menu item is selected.

It will reveal the **Find Call Hierarchy** command under the top level **Find** menu. Note that the shortcut is **Control-Shift-Command-H**.

Press **Escape** twice to completely exit the Help menu and press **Control-Shift-Command-H** to view the call hierarchy in the **Find** navigator.

Note that you can also drill down and find the callers of the callers as well.

16) Add a shortcut

Go to Xcode's preferences, and then go to the Key Bindings tab. Type in **blame**.

On the Version Editor > Show Blame View line, **double-click** on the area right under **Key** and enter **Control-Option-Command-B**. Click outside the box to enable it.

17) Filter by shortcut key

Clear **blame** from the filter box, and type in **b** instead. Click on the dropdown to the left of it and select **Key Equivalent**. You can now see all the shortcuts that use **B**.

Press **Esc** twice to exit the Filter text field and then exit Preferences.

18) That's it!

Congrats, at this time you should have a good understanding of using code snippets

and about how to learn to use keyboard shortcuts. Take a break, and then it's time to move onto Demo3.

Xcode Tips & Tricks: Demo 3

By Jawwad Ahmad

In this demo, you will learn everything else I wanted to show you but couldn't fit into Demo 1 and Demo 2!

The steps here will be explained in the demo, but here are the raw steps in case you miss a step or get stuck.

> **Note**: Begin work with the starter project in **Demo3\starter**.

1) Alias for opening a project or workspace

Use `open -a Xcode .` when you are within a folder in which you have an Xcode xcodeproj or a xcworkspace file, and it will open the correct one.

To make this simpler an alias, for example,**ox**, and just run `ox` when you are within a folder that has an Xcode project.

```
alias ox='open -a Xcode .'
```

2) Open Quickly to a method

You can also use the Open Quickly shortcut **Command-Shift-O** to navigate to methods and properties.

Say you wanted to navigate to the `tableView(_:titleForHeaderInSection:)` method but you can't remember what file it's in.Press **Command-Shift-O**, type in **titleforheader**, and press **Enter** to select the first match to go to that method.

3) Open Quickly to a variable

You also know there's only one `pickerView` in your project. Rather than try to remember the class its in just press **Command-Shift-O**, type **picker**, and hit **Enter** to select the first entry which is a variable.

4) Explore code structure

Open **NumericalInputTextField** by using **nitx** in open quickly, and go to the end of the file using **Command-Down**.

Hold down **Command** and move your mouse over the braces at the end of the file to visually see what part of code each brace is closing.

5) Explore Actions Menu

Command-click on the `if` in one of the `if` statements. Note the options that you get and that one of them is: Add "else if" statement. Click on it to explore what it will do, and then press **Command-Z** to undo the change.

Command-Click on `func` to see options related to modifying methods. And if you **Option-Click** on `class` at the top of the file you'll see class specific options such as **Add Method** and **Add Property**.

6) Add Option to open in the Assistant editor

As a general rule, add the **Option** key to an existing navigation command, have that navigation occur in the assistant editor.

Use the Open Quickly shortcut, type **fdcell**, and hold down the **Option** key while pressing **Enter**. You'll see the file open in the Assistant editor.

Press **Command-Enter** to close the Assistant editor.

7) Using color literals

Use **Command-Shift-O** and enter **colors** to open **NamedColors.swift**.

On line 36 you'll see `banannaYellow` defined. You will convert this to using a Color Literal.

Double-click on the first parentheses to select the parentheses and everything

between it. Press **Command-X** to cut. Press **Option-Backspace** to delete the UIColor text. Start typing **color** and select the **Color Literal** autocomplete.

Press **Command-/** to comment out the line to see the underlying `#colorLiteral` method that Xcode uses with the RGB values.

Double-click on the first parentheses and press **Command-V** to paste. Uncomment the line, and you'll see this in color literal format. You can also **double-click** on the color itself to change it.

Press **Command-/** again to uncomment to see the color literal again.

8) Using image literals

On the following line type `let image = image`, wait for auto-complete to kick in and select **Image Literal**. Now **double-click** on the image and select any image from the popup menu.

Note that a `UIImage` created this way will not be an optional, as compared to the `UIImage(named:)` initializer.

Delete the `let image` line.

9) Add shortcut for select word

I've found it annoying that Xcode doesn't have a shortcut to select the word under your cursor. However, you can add one yourself in Xcode's preferences.

The shortcut to select the word under the cursor in Sublime and Atom is **Command-D**, and it is **Control-W** in TextMate so I'd recommend using one of those depending on which of those editors you like to use.

Note: Xcode already uses **Command-D** for duplicate, but this only works on storyboards and is basically the same thing as selecting a view and then copying and pasting it, i.e. **Command-C,V**. I'd rather use **Command-D** for this purpose rather that one in which it doesn't really add much value. I'm still not sure why Xcode doesn't have a duplicate line shortcut.

10) Add shortcut for blame view

Xcode doesn't have a shortcut for `git blame`, and sometimes I see some code and think to myself "what were they thinking". And then I check git blame to see if they noted what they were thinking in the git commit.

I like to use the **Control-Option-Command-B** for git blame. Show Blame view,

mad enough that I have to press 3 modifier keys but not mad enough to contort my fingers to pressing 4 modifier keys.

11) Hide extra simulators

Do you have to support 3 versions of iOS and have a ton of simulators in the drop down menu? Hide extra simulators by going to **Add Additional Simulators...** in the device dropdown.

Then click on the Simulators tab, select a simulator and uncheck the option that says **Show as run destination**.

12) Experiment with git

Don't be afraid to experiment with git. Just create a backup branch before running a risky command.

For example run: `git branch backup-branch` which will create a new branch and will keep you on your old branch. When you are confident that the command worked use `git branch -D backup-branch` to delete it.

13) Use SourceTree

How can you tell if a command worked? A great way is to look at things visually. I really like using SourceTree for this and its free.

https://www.sourcetreeapp.com/

Using a GUI really helped me out in learning how git works, especially when I wanted to try out a command and visually verify that the result was what I think it should have been.

14) Git bisect

One final git tip, if you haven't used `git bisect` yet, you must, it's great.

Next time you are trying to find that one commit in which a certain behavior changed use `git bisect` to binary search your way to the offending commit.

15) Increase or decrease font size

Command-Plus and **Command-Minus** is a shortcut that works in almost every app to either zoom in or out or to increase or decrease the font size, and it finally works in Xcode. You no longer have to go into preferences to set your font size.

16) Show console automatically in Playgrounds

Playgrounds are likely to have a lot less logging that projects, so I find this setting really useful for playgrounds.

In Xcode's preferences go to the **Behaviors** tab, scroll to the end in the left-hand pane and the 2nd last item will be **Playgrounds**.

Under it, select **Generates output** and add a checkbox to **Show debugger with Current Views**.

Now anytime you use a print statement in a playground to print to the console, the console will automatically become visible if it isn't already.

Close the preferences window by hitting **Escape**.

17) Playgrounds Shortcut

If you want to really increase your use of playgrounds, this is an important one to know.

Press **Command-Control-Shift-N** to create a new playground.

You'll create what I call a "self-destructing" playground.

Press **Enter**, then **Command-Shift-G** for go to folder. Enter **/tmp** and hit **Enter** to Create.

Note that the console area is hidden.

Below `var str = "Hello, playground"` type:

```
print(str)
```

Wait for a second for it to execute. Note that the console appears automatically.

18) Use emoji in log messages

Xcode still doesn't have a way to log at different levels or in different colors. If your team is lax with leaving log statements in it can be difficult to actually view the log

statement you might need. What is useful in this case is adding an emoji to have the log message stand out.

Position your cursor right before `str` and press **Control-Command-Space** to open up the **Character Viewer**, select **Symbols** and **double-click** on an item to select it. Close out the character viewer by tapping on the **x**.

Add quotes around your emoji and a comma after it to see it printed out to the console.

```
print("😀", str)
```

The above statement is the same as:

```
print("😀 \(str)")
```

19) Clear the Console

Use **Command-K** to clear the console. This works in Xcode, Terminal, and iTerm as well.

Use the mnemonic: *Klear* the *Konsole* to remember that it uses a K.

Close the playground and don't worry about deleting it since you created it in your **/tmp** directory.

20) Scratch.playground in Dock

I find it very useful to have a link to a **Scratch.playground** file in my Dock. This way I can quickly open it up and test things. If there is something I can just **Save-As...** to my actual Playgrounds folder.

21) Use a clipboard manager

If you aren't using a clipboard history manager, I'd highly recommend using one. I like Flycut. It's free and available in the Mac App Store.

22) Spaced repetition

One of the keys to retaining information is spaced repetition, so let's set a reminder for you to review these tips.

Open **Reminders.app** press **Command-N** to add a new event and type in "Review Xcode Tips and Tricks next week at 3pm"

23) Schedule time on your calendar

One of the problems with reminders is that they are easier to dismiss and ignore. If you are serious about doing something you don't just need a reminder, you need to actually schedule time on your calendar to do it.

Open **Calendar.app**, use **Command-N** to add a new event and type in "Review Xcode Tips and Tricks, in 2 weeks at 3pm"

24) Re-enable Notifications

And finally **Option-click** on the notifications drawer to turn notifications back on again.

25) That's it!

Congrats, at this time you should have a good understanding of various Xcode Tips & Tricks to improve your workflow!

I find it really useful to have a short alias to go directly to my Xcode project folder.

```
alias go='cd ~/path/to/my-company-ios-app'
``` -->
```
```