Debugging. If the app does not load or loads but does not behave correctly, then the code will need to be debugged. Debugging is the process of identifying and fixing errors in the code, so you can think of it as being a detective. You are given clues about the error (where the error might be, what it might be), and you use these clues to find and solve the problem. Thus, it requires patience, problem solving, logical reasoning as well as a lot of trial and error. The general process of debugging is as follows: identify the error, try to fix the error, run the app. If the error is resolved, then you can move onto the next error. If the error is not resolved, you will need to try to fix it again.

The programming software will usually tell you where the error is and what the error is. If you do not understand the error message, just copy and paste the message into Google along with the programming language (e.g., JavaScript, Objective-C, or Java), and look for solutions and explanations that other programmers have posted. There are two types of errors that you will need to debug: errors in the native codes and errors in your customization of ExperienceSampler.

Native code errors. Errors in the native code typically have to do with changes that Android or Apple has implemented. These errors are in the native language used to program apps (i.e., Java for Android and Objective-C for iOS). They typically arise when a new version of the OS has been released. There are two types of native code errors: warnings and compile errors.

Warnings are represented by yellow triangles. For the most part, you should not be worried about these errors because they are about deprecation. Deprecation is when a feature, or the syntax used to access a feature, will be or has been phrased out and has been replaced by a new feature or new syntax. The programmers who maintain Cordova and Cordova plugins will usually fix these errors, so you should check for updates regularly on the Cordova and Cordova

plugin sites. Even if you get a lot of these warnings, you will still be able to run your app on your emulator and package your app for distribution, so do not worry too much about these warnings.

Compile errors are represented with red circles. You will be unable to run your app if you have a compile error, so these errors need to be resolved immediately it. You can often find solutions to these errors by googling the error message along with "cordova" and the operating system for which you are programming (i.e., iOS or Android). Other programmers will often post solutions online. This is how we usually resolve native language codes without having to learn the native programming language. It is important to note that these errors are typically rare and are most likely to occur when a new OS version has been released. We will usually update the ExperienceSampler template files so that they are compatible with the latest OS release as soon as we can so that you will not encounter these native compile errors. Another way to avoid native compile errors is to always use the most up-to-date versions of Cordova and the various Cordova plugins in your customization of ExperienceSampler.[i]

ExperienceSampler errors. If your app loads a blank screen or if you are unable to advance to the next question screen, then you have at least one ExperienceSampler index.js error. To see which line contains the error, you will need to enable the developer extensions in your browsers. The error messages for your ExperienceSampler JavaScript file will be displayed in the web inspector windows of your browsers. We will discuss how to access the web inspectors for each OS below. The console window will usually display the error message. The error messages will reference the file, usually the index.js file, and the line number (which is the number that follows "index.js:") in which the error has occurred. There will also be a description of the error. An error in the index.js file will usually be accompanied by an error in the

index.html file that states the `app` variable cannot be found. This error will automatically be resolved when you fix the error in the JavaScript file.

Debugging iOS. To debug your iOS version, you will need to enable the **Develop** menu in your Safari browser. To do this, go to **Safari** and choose **Preferences**. Then click on the **Advanced** tab, and check the box next to **"Show Develop menu in menu bar**." Then you need to launch your app in the iOS simulator, open Safari, and open your web inspector by going to **Develop**, then **Simulator**, and then **index.html**. The Web Inspector will open and the number of compile errors will appear at the top of the Web Inspector. Click on the compile error symbol, which is a red stop sign with an exclamation point, and the error messages will appear in the Web Inspector. If you know there is an error but you do not see a red compile error symbol, click on the refresh symbol, which is on the top left of the Web Inspector. The error messages will appear on the bottom of the Web Inspector in the console.

Debugging Android. To debug your Android version, you will need to launch your app in Genymotion by typing `cordova android run` in the CLI. Then open your Google Chrome web browser and type **chrome://inspect**. The name of the Genymotion emulator should appear on the screen (i.e., the device model and operating system version). Underneath the name of the emulator is the name of the app and a link titled **inspect**. Click **inspect**. This should open a window titled **Developer Tools**, which looks similar to Safari's Web Inspector. Any problems with the app will appear on the right-hand side of the Developer Tools window, under the console tab, with a reference to the file and line number in which the problem appears.

Debugging strategies. Once you have identified the error, you will need to try to fix it. You should implement your solution in the index.js file in the `www` folder. To test if your solution works, type `cordova build` in the CLI, and then run your app on your emulator

either through pressing the play icon in Xcode (for the iOS version) or typing `cordova run` in the CLI after launching Genymotion (for the Android version). You will have to iterate through this cycle until you have resolved the error.

Most of the time, you will have syntactical errors (e.g., you forgot a bracket, semi-colon, or comma somewhere). The console log will usually tell you what you forgot. Remember that brackets come in pairs.

Other times errors arise because of your variables. You may have made a mistake when assigning a value to a variable, so the variable is now undefined. One way to resolve this error is to look for all instances of that specific variable in your code. You want to start from the beginning of the code and follow the variable throughout the code. When following the variable, look at whether you have declared it correctly, how you have changed the its value, and how you have used it; this will help you determine what the error is.

Alternatively, you may have assigned the wrong value to the variable. To determine what value you have assigned to the variable, you can make the app record the value it assigns by using either the `console.log()` or `alert()` function. The console log function will display the value in the console window, whereas the alert function will display the value in a pop-up window. Inside the brackets of `console.log` or `alert` function, you should specify the variable that you want the console to record or the alert to display. For example, `console.log(date)` or `alert(date)` will display the value of the `date` variable. You can then make the necessary adjustments to the formula you are using to assign the value to the variable.

If you think that a whole function is not working properly, you can test this by commenting out all the lines inside the function, run the app, and see if the app starts working

again. You can then slowly uncomment one line or a few lines at a time and then re-run the app to determine where the exact error is. We would advise new programmers to uncomment lines one at a time.

Another way to see if a function is problematic is to insert an `alert()` at the very beginning of the function and see if the function displays the alert. We like to use alerts that tell us what function we're in (e.g., `alert('I'm in the renderQuestion function!')`). If the alert is not displayed, it means that the function is not even being executed at all. You can also move the alert to other places inside the function to identify which line contains the error instead of commenting out lines. For example, the app may only execute part of the function, but you are unsure of where exactly the app stops executing. You think it might be somewhere in line 100, so you can place your alert command in line 99 and then try to run the app. If the app runs, the problem occurs after line 100. If it does not run, then the error is before line 100. When debugging, it is important to confirm your suspicions. Thus, if the app runs when you put the alert in line 99, move the alert to line 101 and then try to run the app again. Do not just assume that the error occurs after line 100.

---

[i] We recommend using the most stable up-to-date versions of Cordova and Cordsova plugins. This is the version that is automatically installed when no version is specified.