|  |  |
| --- | --- |
| In this lesson we won’t add new functionality to the Paint Pot app. Instead, we will revise the code, leaving the app’s behavior unchanged. This process is called ***refactoring*** and programmers do this to improve the quality of their code in various ways -- e.g., to simplify its design, make it easier to read and easier to maintain.    In this case we will introduce the concept of a ***programmer-defined procedure*** that will help reduce the complexity of our code and make it easier to read and maintain. This is an example of ***procedural abstraction***, a very important concept and practice in programming.    **Objectives:**The objectives of this lesson are:   * to continue learning how to navigate the App Inventor online programming platform; * to introduce the concept of a programmer-defined procedure. | GnuScreen.png    ***[Click to watch Preview Video](http://www.youtube.com/watch?v=BVCrfDvjuIY)*** |

## 

## Getting Ready

Open App Inventor with the [Paint Pot Solutions template](http://ai2.appinventor.mit.edu/?repo=templates.appinventor.mit.edu/trincoll/csp/unit3/templates/PaintPotRefactor/PaintPotRefactorTemplate.asc) in a separate tab and follow along with the following tutorial. After the project opens, use the *Save As* option to rename your project *PaintPotWithProcedure*.

## 

## Code Walkthrough and Critique

If you completed each of the mini-projects in the Paint Pot Projects lesson, your code might look something like the code for our solutions:

## PPProjectsWorkspace.png

Notice that the circled segments are three occurrences of exactly the same code. The circled code performs the task of displaying the current dot size in the format *Dotsize = 5.*

This is an example of ***redundant code***  and it is generally not a good idea. For one thing, it makes it more complicated to modify and maintain the program. For example, suppose we decided that we want to change the display to say *Dot Radius: 5* instead of *Dot Radius = 5.* This is a subtle change but to effect this change we would have to find and change all three occurrences of that code. If we forgot one, that would introduce a bug into the program.

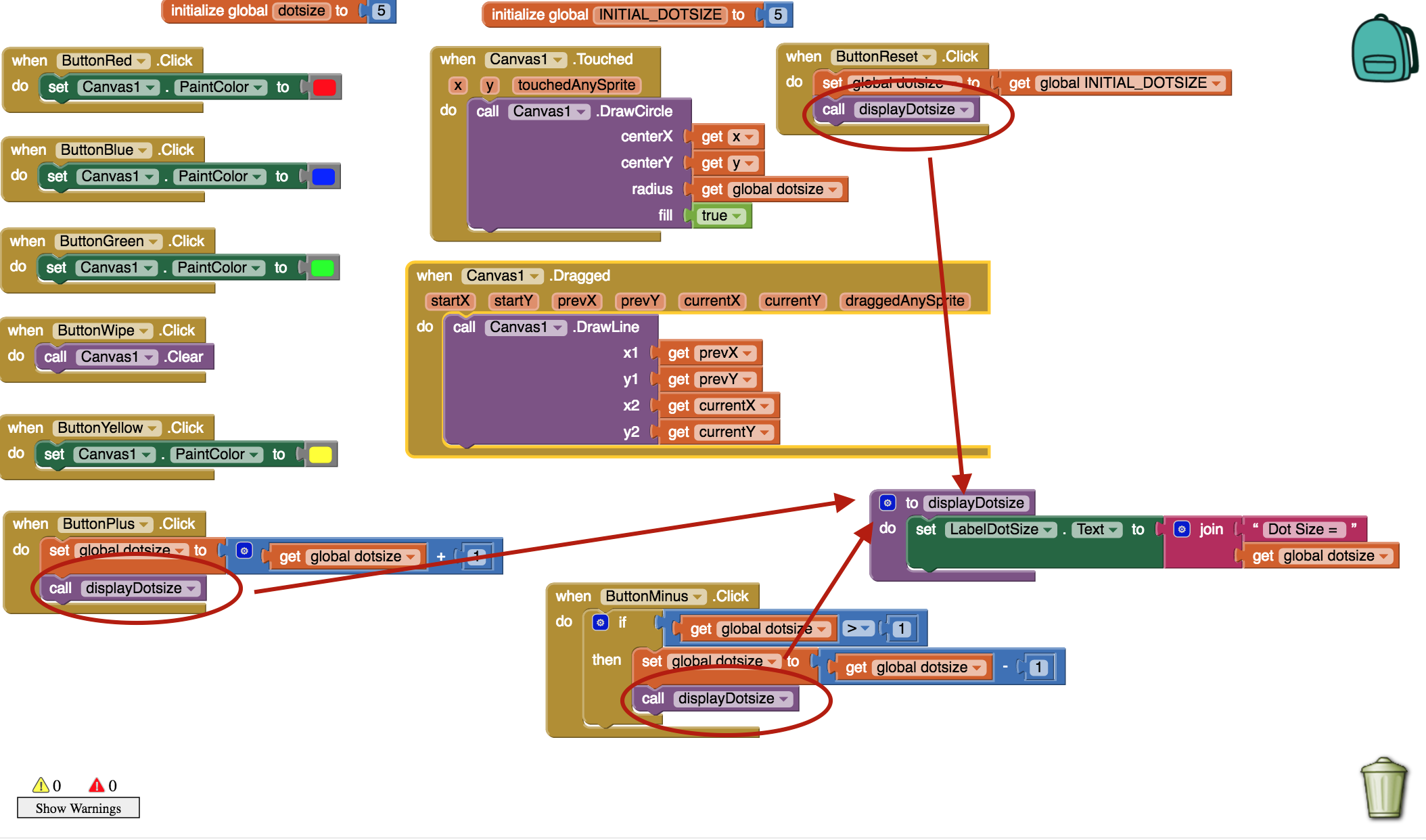
This is a fairly minor and innocuous example, but you can imagine cases where forgetting to change some piece of code could seriously affect the behavior of the program.

## Procedural Abstraction

A better way to code this task of displaying the dot size would be to ***define a named procedure*** that encapsulates the display algorithm and to ***call the procedure by name*** whenever we want to display the dot size. This is another important use of ***abstraction*** in computer science.

# Refactored Code

Consider the following ***refactored*** version of the program. In this case we have defined a procedure named ***displayDotsize,*** which now contains the code for displaying the dot size*.* And we are calling it, by invoking its name, from three different places in the program (circled).

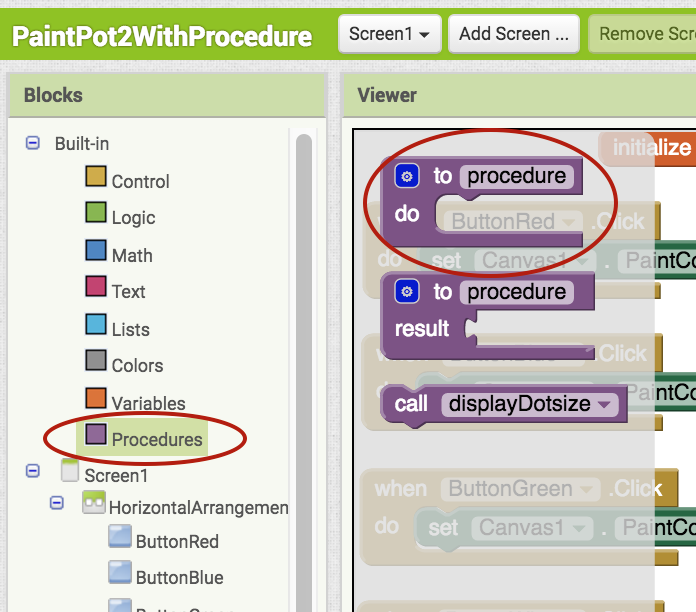


This is a much improved version of the program. For example, think about what we need to do now if we want to change the display to *Dot Radius = 5.* We need only change the code inside the procedure and the change will take effect throughout the program. The code we need to change is easy to find because it is ***encapsulated*** within the *displayDotsize procedure.*

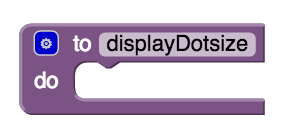
By defining and using a procedure in this way, we have ***reduced the complexity***  of our code. A good way to think about this is that the procedure enables us to reduce the number of details involved in a task so we can focus on the relevant aspects of the task. In this case, we can focus on the purpose of the task -- i.e., to display the dot size -- without having to worry about the exact details of how that is done.

## How to Define a Procedure

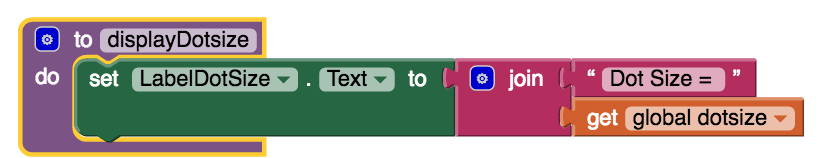
To define a procedure in App Inventor, you will need to open the *Procedures* drawer in the Blocks editor. In this case you want to select the first ***procedure-definition*** block in the drawer, the ***to procedure*** block, and drag it to the workspace. (The second *to procedure* block is used of defining functions, which are procedures that return a value, and we’ll get to them later.)



Once you have the *to procedure* block, you’ll want to change the word *procedure* to a more descriptive name for the procedure -- in this case to *displayDotsize:*

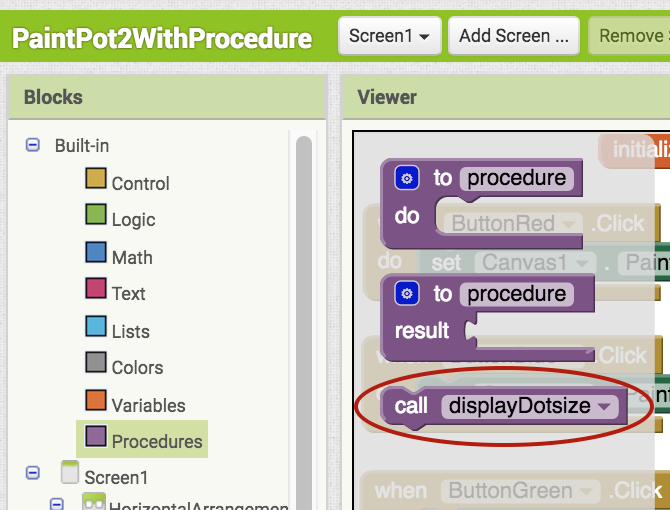


A procedure-definition block, such as this one, defines what it means to displayDotsize. And we do that by placing our algorithm -- the steps we take to *displayDotsize --*  inside its ***do-slot***:



## How to Call a Procedure

The above steps suffice to ***define***  the procedure. In order to use the procedure we need to ***call it*** or ***invoke it.*** Whenever you define a procedure, App Inventor automatically makes a ***procedure-call*** block and places it in the *Procedures* drawer:



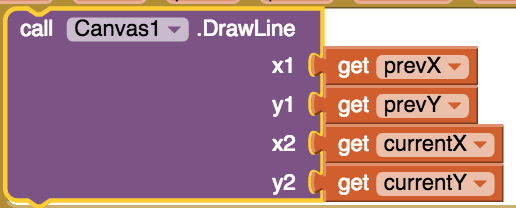
Now, whenever we want to perform the defined procedure, we simply call it, as shown in the refactored version of the program. We can call it as many times as we like from anywhere within the program. And we don’t have to worry about how to find its code if we want to change the algorithm because it is encapsulated within the procedure-definition block.

## Built-in and Programmer-defined Procedures

Notice that *procedure-definition* and *procedure-call* blocks are purple. In that respect they resemble the built-in procedure blocks we’ve been using in our apps. For example, in the *I Have a Dream* app we used built-in procedures to start, pause, and stop the audio player:



The algorithms required to perform these type of operations on the *Player* component are probably much more complex than our simple *displayDotsize* procedure. So in this case, the abstraction we’re using, the *PlayerMLK.Start* procedure, is hiding quite a bit of complexity. Similarly, in the *PaintPot* app, we use the *Canvas.DrawLine* procedure to draw lines on the Canvas:



Here too, using the built-in procedure allows us to focus on the relevant task -- i.e., drawing a line -- rather than having to worry about the underlying complexities involved.

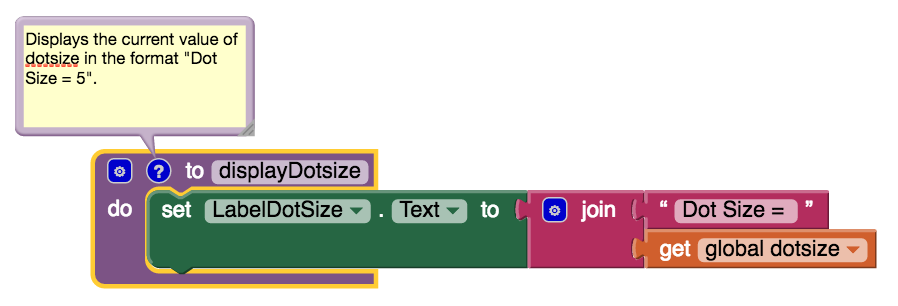
However, even though these are built-in procedures, the principle is the same: once we have a procedure, whether built-in or programmer-defined, using the procedure allows us to focus better on the relevant tasks rather than worrying about the complex details encapsulated in the procedure definition.

# Adding Comments to Code

An important feature of every programming language, including App Inventor, is the ability to add comments to the code. A ***comment*** is a non-executable block of text that can be added to a program to provide clarification and documentation of the code. Adding comments to one’s code is a standard practice that programmers employ to help others (and themselves) understand their code.

In App Inventor, each non-collapsed block comes with the capability of having a comment added to it. To access this capability you must ***right-click on the block*** and choose the ***Add Comment*** option. This will add a small ***comment-icon,*** a blue circle with a question mark, to block (as shown here).

To add or edit the comment, simply click on the comment-icon and type in your comment, as shown here:



## Good Commenting Practice

A good practice to follow is to provide comments in the following situations:

* To document every procedure that you define, as shown in this example here.
* To clarify a complex algorithm that isn’t clearly obvious.