|  |  |
| --- | --- |
| In this version of Paint Pot we use variables to enable the app to draw different sized dots. A *variable* is a memory location that can store a certain value. Because the values it stores can change, a variable behaves like an *abstract symbol* that can represent different values. We’ll see why this is a powerful concept.    **Objectives:** In this lesson you will:   * follow an instructor-led walkthrough to create the *Paint Pot 2* app on a mobile device; * learn to use a variable to make the program more general. | ***[Click to watch Preview Video](http://www.youtube.com/watch?v=ieD3Ww_URuw)*** |

## 

## Getting Ready

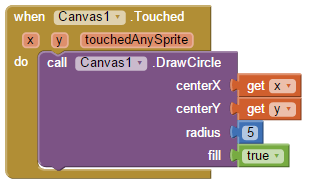
To get started, [open App Inventor with the Paint Pot 2 Template](http://ai2.appinventor.mit.edu/?repo=templates.appinventor.mit.edu/trincoll/csp/unit3/templates/PaintPot2Template/PaintPot2Template.asc) in a separate tab. Once the project is opened, use the *Save As* option to rename it ***PaintPot2****.*

# Paint Pot 2 Tutorial

([Video Tutorial](http://www.youtube.com/watch?v=7eY2aSeP9Oc))

## Here’s the Problem

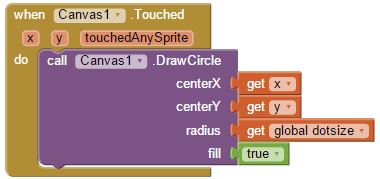
The current version of PaintPot paints dots that all have the same radius -- 5 pixels. Let’s take another look at the *DrawCircle* method that we used to draw a dot whenever the canvas was touched. As you see here, the radius, *r,* is always set to 5. Every dot will have a radius set to 5. The radius will never change!



Programmers refer to the number 5 here as a *constant* or a *literal value* because its value never changes -- it is literally 5. How can we make this more flexible? How can we enable the app to draw dots of different sizes?

## Abstraction to the Rescue

Let’s think about what would happen if we replaced the number 5 in the above block with a symbol, such as *dotsize,* that can represent any value:



Now, when a dot is painted, its radius will be whatever value that *dotsize* represents. If we set dotsize’s value to 5, then it would draw a dot of radius 5. If we set it to 8, it will draw a dot of radius 8. And so on. So, rather than just be a constant, such as 5, *dotsize* is an abstract variable that can stand for any value. This is a simple example of the *abstraction principle.* We will see many other examples in this course. But, we can’t implement this just yet. First, we will need to create a global variable.

### Creating and Using Global Variables

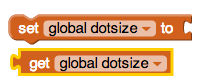
In App Inventor, *dotsize* is an example of a *global variable,* which means that it is a storage container that can store any value and it can be used *globally* throughout the app. Global variables have to be created and given an initial value using a special block that is found in the *Variables* drawer of the Toolbox. (If you look in that drawer you will find that App Inventor also has *local variables,* which we will also learn how to use eventually.) When creating a global variable you should give it a unique, but valid, name. Variables in App Inventor, as in many other computer languages, must be strings of letters and digits (no quotes) and cannot start with a digit. An example of an invalid name would be *10seconds.*

Here’s how we define and initialize our global *dotsize* variable:

initglobaldotsize.png

We have given *dotsize* an initial value of 5. So if we added these two blocks to our app and did nothing else, it would behave just as before -- all dots would have a radius of 5 because that’s the current value of *dotsize.*

Of course, App Inventor also has blocks in the *Variables* drawer to *set* and *get* the value of a global variable:



For the *setter block* we can set the value of *dotsize* to whatever number we put in its *to-slot.* By using a *getter block* we can get *dotsize*’s value and plug it into any slot where it will fit -- just like putting together a jigsaw puzzle.

Now that we’ve created the global variable *dotsize*, we can replace the constant *5* with the *getter block,* for *global dotsize*, as we discussed in the previous section.

## Example: Adding 1 to a Variable

Variables that store numbers, such as *dotsize,* can be used in arithmetic.To see this let’s pull an *addition block* and a *number block* out of the *Math* drawer:



(Don’t be confused by the blue circle on this block. It is a drop-down menu that lets you change the block so you can add more than two values with one block. The white ‘+’ sign in the middle of the two open slots is the plus operator.)

Notice that the open slots the addition block has the correct shape for plugging in either a value (such as 0) or a variable (such as *dotsize*):

dotsizeplus0.png

Thus, we have created an *expression* whose value is (dotsize + 0). Since we have initialized the *dotsize* to 5, then this expression has the value 5. If we change the number block to a 1



then the resulting expression would have the value 6, (dotsize + 1) or (5 +1).

Now the resulting expression block can itself be plugged into any slot where a value can be plugged. For example, we can plug it into *dotsize’s* setter block:

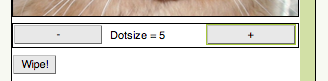
setdotsizeplus1.png

This is an App Inventor example of an *assignment statement*. The value of dotsize is set to the result of the expression which is its current value + 1. *dotsize* now has the value 6.

## Drawing Different Sized Dots

How might we use this new found ability to enable our app to draw different sized dots? One way would be to add two Buttons to the app, one labeled ‘+’, which adds 1 to dotsize whenever it is clicked, and the other labeled ‘-’, which subtracts 1 from dotsize whenever it is clicked. It might also be nice to add a *Label Component* that will display the current value of dotsize and to update its value whenever one of the buttons is clicked.

In other words, we want to change our UI so that it has the following additional components:



(NOTE: You could also add two Labels for this task, one for the prompt, “Dotsize = “, which never changes, and one for the current value of dotsize (e.g., 5), which will vary when the user changes the dotsize. In this example we will show how to use a single label that *concatenates* both the prompt and the value.)

You probably know how to do this now, but here are steps:

1. Drag and drop a *HorizontalArrangment* component right under the canvas.
2. Fill it with two buttons and a *Label* component, and set their respective *Text* properties as shown here.
3. Give these components descriptive names, such as *ButtonPlus, ButtonMinus,* and *LabelDotsize.*
4. Set the width of all 4 of these components to “Fill parent...”.

## Coding the Plus and Minus Buttons

Whenever the plus (‘+’) button is clicked it should perform the following operations:

1. Add 1 to the global dotsize.
2. Concatenate the prompte (“Dot Size = “) and the global dotsize using a ***join*** block from the Text drawer:



1. Display dotsize’s value in the *LabelDotsize.* To do this we set the label’s *Text* property.

Here’s what the block should look like:

## whenclickdotsize.png

It has two statements. First it adds 1 to *dotsize --* we saw how to do that above. Then it sets the label’s *Text* property to the string “Dot Size = ***dotsize*** “ where the variable *dotsize* is replaced by its current value. The coding for the (-) button is left as a mini-project.

***Nice work! Complete the Self-Check Exercises and Portfolio Reflection Questions as directed by your instructor.***